Quantum algorithms for data analysis

Alessandro Luongo

2024-12-08

Contents

Chapter 1

Preface

This open source project accessible on GitHub is only possible thanks to its many contributors. The website is licensed under CC BY-NC-SA 4.0. We are searching for talented people and researchers to contribute.

MEMO: we have some funding for motivated contributors

The aim of this book is twofold:

- First, we want to bridge the gap between introductory material in quantum computation and research material.
- Second, you should be able to use this book as a resource for state-of-the-art algorithms. Readers and scholars should find statements of theorems (along with their citations) and runtimes of the best quantum subroutines in literature, ready to be used in new quantum algorithms or applications.

These lecture notes were used to teach at:

- Politecnico di Milano (2019) - Quantum machine learning in practice.
- Politecnico di Milano (2021) - Applied quantum computing.

Are you using these lecture notes to support your course? Write us an email!

This book is dedicated to all cypherpunks: *civil liberties through complex mathematics.*

1.1 Abstract

In these lecture notes, we explore how we can leverage quantum computers and quantum algorithms for information processing. It has long been known that quantum computation can offer computational advantages over classical computation, and in this book we explore the consequences of this fact in current research areas of computer science.

Are there other reasons, besides getting a practical computational advantage for studying quantum algorithms? We argue that having faster algorithms is not the only reason for studying quantum computing.

One — perhaps shallow — reason is to satisfy our curiosity by studying how to use quantum mechanical systems for doing computation, and challenging yourself in finding a faster-then-classical algorithms. Studying quantum computation might also reveal profound insights into new ways to process information. For instance, it can give us ideas on processing data in a secure way (though, quantum cryptography is not discussed in these notes). A better understanding of quantum computing might lead to understanding the computational limits of nature: what can be computed in this world? What can be computed with classical computers? As an example, because of the interplay between research in classical and quantum computation, many new *classical* algorithms have been invented (i.e. the dequantizations of quantum machine learning algorithms, new classical algorithms for Gibbs sampling, classical simulations of quantum circuits, etc..). This, in turn, improved our understanding of physics, and ultimately of the world itself. One last reason for studying quantum algorithms — which a computer scientist can surely appreciate — is that quantum computers are posing a significant challenge to the *extended* Church-Turing thesis, which states that any “reasonable” model of computation can be *efficiently* simulated on a probabilistic Turing machine. However, there are many physical processes that we do not know how to simulate efficiently on classical computers, but for which we have efficient quantum algorithms! This is strong evidence that the strong Church-Turing thesis might be false!

You might often hear that there are only two real quantum algorithms: phase estimation and the Grover’s algorithm. This is true in the same way that we have only 12 notes in the western temperate scale, yet Pink Floyd was able to write The Dark Side of the Moon (and other musicians came up with “the rest” of the music).

The common thread of these algorithms is that they are faster than their best classical counterpart. Oftentimes, (especially for ML) the runtime will depend only poly-logarithmically on the number of elements of the dataset, and it is usually only linear in the number of features (classical algorithms are often either linear in the number of elements and quadratic in the number of features, or depend on the number of nonzero components of the matrix and depend polynomially on other parameters of the matrix). The runtime of a quantum machine learning algorithm also often depends on characteristics of the matrix that represents the data under analysis, such as its rank, the Frobenius norm (or other matrix norms), the sparsity, the condition number, and the error we tolerate in the analysis. For this, along with an error-corrected quantum computer, we assume to have quantum access to a dataset. In other words, we assume that the data is stored in a quantum memory: the corresponding quantum version of the classical random-access memory.

We will see that, for a new QML algorithm, one often needs to make sure that the real performances of the quantum algorithms offer concrete advantages with respect to the effective runtime and the accuracy that is offered by the best classical algorithms. As we don't have access to big-enough quantum computers *yet*, we can only assess the performance of these quantum algorithms via a classical simulation.

These lecture notes should prepare the future quantum data analyst to understand the potential and limitations of quantum computers, so as to unlock new capabilities in information processing and machine learning. The hope is that this kind of technology can foster further technological advancements that benefit society and humankind, as soon as the hardware that supports this kind of computation becomes ready.

Last but not least, we will also cover important algorithms that are not necessarily related to machine learning, but are the quantum counterpart of important classical algorithms. Don't get swayed by the "lack" of exponential speedups. Remember: the square root of 365 days is a little less than 3 weeks. Besides this, big polynomial speedups, small polynomial speedups in important problems, or polynomial speedups proposing new algorithmic techniques are all much welcome in quantum computer science. All in all, quantum algorithms can be seen as a way for making **impossible things possible**.

While reading these lecture notes you should always remember a quote from the good Simon Martiel:

"(quantum) Theoretical computer science is the fun part of mathematics."

To all of you, happy reading.

1.2 Changelog

- **August 2020**: Migrated the old blog on bookdown
- **December 2020**: Moved thesis in bookdown
- **January 2021**: First system for typesetting algorithms, more appendix on linear algebra
- **February 2021**: New subroutines for estimating ℓ_1 norms.
- **March 2021**: quantumalgorithms.org is proudly supported by the Unitary Fund, and quantumalgorithms.org is a project of the QOSF mentorship program: 5 students started creating new content!
- **April 2021**: Mobile version working, search functionality added, q-means, finding the minimum, new algo for dimensionality reduction, and factor score ratio estimation estimation.
- **June 2021**: Quantum Monte Carlo algorithms, lower bounds techniques for query complexity of quantum algorithms, quantum algorithms for graph problems. The output of the mentorship program of the QOSF foundation!
- **January 2022**: In the past months we improved the overall quality of

the website (graphics, typos) and added some content (Deutsch-Josza, Bernstein-Vazirani, swap and Hadamard tests)
- **February 2022**: We are happy to bring our swag (t-shirts, stickers, lanyards) to QIP. We added a whole chapter on perceptron algorithms and we doubled the chapter on quantum Monte Carlo algorithms with applications to finance and trace distance!
- **June 2022**: We are participating to the UnitaryHack! We also presented this work at QNLP2022 where we distributed our swag! Now the website can be compiled also with the latest version of RStudio,pandoc,bookdown.
- **August 2022**: Added in the appendix: polynomial approximation of 1/x, more on concentration inequalities (these were work started by contributors that discovered the project during the unitary.hack!). Improved css as a bounty to the unitary.hack. Working (on a separate project) to improve chapter 3.
- **May 2023**: We are improving the overall book with the precious help from the editor (Hue Jun Hao Alexander) and other contributors.

NEW Submit your solution of the exercises as PR on GitHub and get it published in the chapter on Solutions.

Coming soon: - quantum perceptrons - quantum random walk - quantum algorithms for dynamic programming - quantum convolutional neural networks - quantum random feature sampling

1.3 Teaching using this book

- Lecture 1 - **Axioms of quantum mechanics**: Chapter 2, section 2.2
- Lecture 2 - **Quantum computer science**: oracle model, BPP vs BQP, from boolean to quantum circuits, Solovay-Kitavev, randomized algorithms, Markov Inequality, Union bound and applications. Section 2.3.
- Lecture 3 **Foundational quantum algorithms**, Section 2.4: Deutsch-Josza, Bernstein-Vazirani, Swap-test, Hadamard-test.
- Lecture 4 - **Oracles, data representation and data loaders**: Oracles for accessing the data, data representation, sparse models. Chapter 3.
- Lecture 5 - **QFT and Grovers' algorithm**: Section 5.2, Section 5.1
- Lecture 6 - **Phase estimation, amplitude amplification, counting, finding the minimum, and applications**: Beginning of Chapter 5.
- Lecture 7 (optional) - **More foundational quantum algorithms**: Simons's and Shor's algorithm (`#TODO`)
- Lecture 8 (optional): **Quantum perceptron models**: This is a very simple quantum machine learning algorithm that shows the applications of different techniques seen so far. (`#TODO`, very soon!)
- Lecture 9: **Quantum numerical linear algebra pt. 1**: HHL algorithm, block-encodings and quantum singular value estimation. Mostly from Chapter 7.

- Lecture 10: **Quantum numerical linear algebra pt. 2**: Quantum singular value transformation, Hamiltonian simulation. Theory and examples.
- Lecture 11: **Distance, inner product, trace estimation**: From Chapter 5, especially Section 5.5, (`#TODO` trace estimations).
- Lecture 12: **Quantum monte carlo algorithms and applications**: Content from Chapter 8.
- Lecture 13: **Quantum machine learning pt. 1**: QPCA and QSFA. Chapter 9 Hamiltonian simulation with density matrices (`#TODO`).
- Lecture 14: **Quantum machine learning pt. 2**: Random feature sampling (`#TODO`, to convert from .tex to markdown, check github repository).
- Lecture 15: **Classical simulation of quantum algorithms**: How to simulate a quantum circuit? (`#TODO`) How to simulate a quantum machine learning algorithm? Section 12.
- Lecture 16: **Lower bounds**: Adversarial and polynomial method in Chapter 14, state discrimination (missing).
- Lecture 17 (optional): **Between the qubits and me**: what do we have between the theory of our algorithms and the hardware? An overview of different hardware architectures, error correction codes, and compilation techniques. (`#TODO`).
- Lecture 18: **Quantum algorithms on graphs**: Backtracking algorithms, NP-complete problems on graphs (`#TODO`).
- Lecture 19: **Quantum optimization**: Introduction to optimization. Quantum simplex, quantum zero-sum games. (`#TODO`).
- Lecture 20: **Quantum optimization**: Quantum SDP algorithms, quantum interior point methods, quantum branch-and-bound algorithms (`#TODO`).

Part I

Bridging the gap

Chapter 2

Quantum computing and quantum algorithms

In this chapter we will introduce the preliminaries needed for in this book. We will extensively use linear algebra (norm of matrices, SVD, properties of particular matrices, and so on), so the reader is highly encouraged to refer to the appendix regarding the notations adopted.

2.1 Getting rid of physics in quantum computing

Following one of the lectures of Susskind, we are going to start from a "handwavy" introduction of quantum mechanics, that starts from few considerations and lead straight to the Schrödinger equation. With a few mathematical tricks, we are going to justify the 4 axioms of quantum mechanics stated in the next sections intuitively. The hope is that the reader can be gently guided from a tiny physical intuition to a greater understanding of the **axioms of quantum mechanics**. While "axioms of quantum mechanics" implies that physics is involved, thanks to some of their formulation (see for instance in (Nielsen and Chuang, 2002), which we adopt), we can ignore the physics and think of them as "axioms of quantum computing". As Scott Aaronson rightly said, if you are not a physicist, you need to remove physics from quantum mechanics to understanding it!

The objective is that the reader should *not* feel the need to dig into quantum mechanical details of quantum computers, but can start solely from the 4 axioms of quantum mechanics and build (a lot) from there.

At the beginning of the 20th century, when physicists started to model quantum phenomena, they observed that the time and space evolution of quantum systems had two properties: it was continuous and reversible (as in classical mechanics). They decided to formalize these properties as follows. First, they decided to model the state of a quantum system at time p as a function $\psi(p)$, and they decided to model the evolution of $\psi(p)$ for time t as an operator $U(t)$ acting on $\psi(p)$s. Let I be the identity operator. Formally, the two requirements can be written as:

- $U(\epsilon) = I - i\epsilon H$ (continuity)
- $U^\dagger(t)U(t) = I$ (reversibility)

The first requirement reads that if we were to apply an evolution for a small amount of time ϵ, then U would behave almost as the identity I, differing by a small amount of another operator H. The second requirement reads that the operation U can be undone by applying a conjugate transpose of U. By doing this we obtain the identity operator. This means that we return to the initial state of the system before it evolved. From these two requirements, we can observe that:

$$I = U^\dagger(\epsilon)U(\epsilon) = (I + i\epsilon H)(I - i\epsilon H) = I - i\epsilon H + i\epsilon H^\dagger + O(\epsilon^2).$$

The only way for this equation to hold is for $H = H^\dagger$, i.e. the operator H should be equal to its transpose conjugate. In mathematics, we call them Hermitian operators! (More about these in the appendix!). Now we can ask ourselves what happens when we apply $U(\epsilon)$ to a quantum state $\psi(t)$? Well it's simple to see now:

$$U(\epsilon)\psi(t) = \psi(t + \epsilon) = \psi(t) - i\epsilon H\psi(t).$$

With a little algebra we can rewrite the previous equation as:

$$\frac{\psi(t+\epsilon) - \psi(t)}{\epsilon} = -iH\psi(t).$$

Note that the left-hand side part of this equation can be rewritten, under the limit that $\epsilon \mapsto 0$ as a derivative:

$$\frac{d}{dt}\psi(t) = -iH\psi(t).$$

But this is the well-known Schrödinger equation! Note that, as computer scientists, we take the right to remove some physical constant ($\hbar$) out of the equation. What should be the takeaway of these observations? We know that the Schrödinger equation is a differential equation whose solution is fully determined if we know the initial state of our system $\psi(p)$. Formally the solution can be written as:

$$\psi(p + t) = e^{-iHt}\psi(p).$$

From this last equation we can observe further (more on this in the appendix) that the exponential of an Hermitian matrix e^{-iHt} is *defined* through its Taylor expansion is just a *unitary* matrix: $U(t) = e^{-itH}$. Unitary matrices are exactly those matrices that describe isometries: applying a unitary matrix to a vector do not change its length. From this, we see that the two quantum states $\psi(p+t)$ and $\psi(p)$ could be taken just to be vectors of a fixed length, which - for practicality - we take to be unit vectors. Notation-wise, we denote unit vectors describing quantum states as “kets”, i.e. we rewrite this equation as:

$$|\psi(p+t)\rangle = U(t)|\psi(p)\rangle$$

Hopefully, this introduction should be enough for getting a better intuition of what comes next, and give you a “justification” for the axioms of quantum mechanics.

2.2 Axioms of quantum mechanics

The standard formalism used in quantum information is the Dirac’s “bra-ket” notation, which we will introduce in this section. We also state here the postulates of quantum mechanics, and take this opportunity to settle the quantum notation and quantum preliminaries used in this book. For the postulates, we follow the standard formulation in (Nielsen and Chuang, 2002).

Proposition 2.1 (Description of a quantum system)**.** *Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the state space of the system. The system is completely described by its state vector, which is a unit vector in the system’s state space.*

As quantum states are described by unit vectors, we write $|\psi\rangle$ for a unit vector $\psi \in \mathcal{H}^d$, where $\mathcal{H}^d$ is a d dimensional Hilbert space. So for a non-normalized vector $x \in \mathbb{R}^d$, the normalized quantum state is represented as $|x\rangle = \|x\|^{-1} x = \frac{1}{\|x\|}\sum_{i=0}^{n} x_i|i\rangle$, where $\|x\|$ is the ℓ_2 norm of the vector x.. We denote as $\{|i\rangle\}_{i\in[d]}$ the canonical (also called computational) basis for the Hilbert space. The transpose-conjugate of $|x\rangle$ is defined as $\langle x|$. We can think of $|x\rangle$ as a column vector, while $\langle x|$ as a row vector, whose entries have been conjugated. In Dirac’s notation, we denote the inner product between two vector as $\langle x|y\rangle$. Their outer product is denoted as $|x\rangle\langle y| = \sum_{i,j\in[d]} x_i y_j |i\rangle\langle j| \in \mathcal{H}^d \otimes \mathcal{H}^d$. The smallest (non-trivial) quantum system is called a qubit, and is a 2 dimensional unit vector in $\mathbb{C}^2$. A base for this vector space in quantum notation is denoted as $|0\rangle$ and $|1\rangle$. In this case, the vector $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$ for $\alpha, \beta \in \mathbb{C}$ represent a valid quantum state as long as $|\alpha|^2 + |\beta|^2 = 1$.

Proposition 2.2 (Postulate 2)**.** *The evolution of a closed quantum system is described by a unitary transformation. That is, the state $|\psi\rangle$ of the system at time t_1 is related to the state $|\psi\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2.*

A matrix $U \in \mathbb{C}^{d\times d}$ is said to be unitary if $UU^\dagger = U^\dagger U = I$, that is, if the inverse of U equal to its conjugate transpose. From this fact it follows that unitary matrices are norm-preserving, and thus can be used as suitable mathematical description of a pure quantum evolution. It is a standard exercise to see that the following are all equivalent definition of unitary matrices (de Wolf, 2019):

- $\langle Av, Aw \rangle = \langle v, w \rangle$ for all v, w.
- $\|Av\| = \|v\|$ for all v
- $\|Av\| = 1$ if and only if $\|v\| = 1$.
- A is a normal matrix with eigenvalues lying on the unit circle
- The columns and the rows of A form an orthonormal basis of $\mathcal{C}^d$
- A can be written as e^{iH} for some Hermitian operator H.

Example 2.1 (Determinant=1 is a necessary but not sufficient condition for being unitary)**.** Any 2×2 diagonal matrix A with entries 10 and 1/10 has determinant is 1, but it is not a unitary matrix.

It will be useful to recall that if we have a unitary that performs the mapping $|a_i\rangle \mapsto |b_i\rangle$, we can have the "matrix" form of the operator as $\sum_i |b_i\rangle\langle a_i|$. Recall also that the Pauli matrices are both unitary *and* Hermitian, and this fact will be useful in many places throughout this text.

Exercise 2.1 (From (Huang et al., 2019))**.** Let $k \in \{0,1\}^n$ be an arbitrary n-bitstring. Let $A = (\sigma_x^{(1)})^{k_1} \otimes \cdots \otimes (\sigma_x^{(n)})^{k_n}$ and $|b\rangle = |0^n\rangle$. What is the solution to the equation $A|x\rangle = |b\rangle$

Proposition 2.3 (Postulate 3)**.** *Quantum measurements are described by a collection $\{M_m\}$ of measurement operators. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement, then the probability that the result m occurs is given by*

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle$$

and the state of the system after the measurement is

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}$$

The measurement operators satisfy the completeness equation

$$\sum_m M_m^\dagger M_m = I$$

In practice, we will mostly perform projective measurements (also called von Neumann measurements). A projective measurement is described by an *observable*: an Hermitian operator M on the state space of the system being observed. The observable has a spectral decomposition:

$$M = \sum_m m P_m$$

where P_m is a projector into the eigenspace of M associated with the eigenvalue m. This means that the measurement operator will satisfy the following properties:

- P_m is positive definite
- P_m is Hermitian
- $\sum_m P_m = I$
- $(P_m)(P_n) = \delta_{mn}(P_m)$ are orthogonal projections.

Recall that an orthogonal projector P has the properties that $P = P^\dagger$ and $P^2 = P$. Note that the second property derives from the first: all positive definite operators on $\mathbb{C}$ are Hermitian (this is not always the case for positive definite operators on $\mathbb{R}$, as it is simple to find positive definite matrices that are not symmetric). Projective measurements can be understood as a special case of Postulate 3: in addition to satisfying the completeness relation $\sum_m M_m^\dagger M_m = I$ they also are orthogonal projectors. Given a state $|\psi\rangle$, the probability of measuring outcome m is given by:

$$p(m) = \langle\psi|P_m|\psi\rangle. \tag{2.1}$$

If we were to measure outcome m, then the state of the quantum system after the measurement would be:

$$\frac{P_m|\psi\rangle}{\sqrt{p(m)}}.$$

They have some useful properties. For example, the average value of a projective measurement in a state $|\psi\rangle$ is defined as:

$$\begin{aligned} E(M) &= \sum_m p(m)m \\ &= \sum_m m\langle\psi|P_m|\psi\rangle \\ &= \langle\psi|(\sum_m mP_m)|\psi\rangle \\ &= \langle\psi|M|\psi\rangle \end{aligned} \tag{2.2}$$

In practice, our projective operators will be projectors in the computational basis, i.e. $P_m = \sum_{m\in[d]} |m\rangle\langle m|$. As a conseguence of these rules, the probability that a measurement on a state $|x\rangle = \frac{1}{\|x\|}\sum_i x_i|i\rangle$ gives outcome i is $|x_i|^2/\left\|x\right\|^2$.

Proposition 2.4 (Postulate 4)**.** *The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered from 1 through n, and each state is described as $|\psi_i\rangle$, the join state of the total system is $\bigotimes_{j=1}^{n} |\psi_i\rangle = |\psi_1\rangle|\psi_2\rangle \ldots |\psi_n\rangle$.*

The tensor product between two vectors $|y\rangle \in \mathbb{R}^{d_1}$ and $|y\rangle \in \mathbb{R}^{d_2}$ is a vector $|z\rangle \in \mathbb{R}^{d_1 \times d_2}$. We can use the tensor operation to describe the joint evolution of separate quantum system.

Even if it is not explicitly used much in quantum algorithms, it is useful to recall the definition of entangled pure state.

Definition 2.1 (Entangled state)**.** A quantum state that cannot be expressed as a tensor product of two quantum state is said to be entangled.

The same thing can be done for operators. Let U_1 be the unitary describing the evolution of a quantum state $|x\rangle$ and U_2 the unitary describing the evolution of a quantum state $|y\rangle$. Then $U_1 \otimes U_2$ describes the evolution of the quantum system $|x\rangle \otimes |y\rangle$.

From the definition of tensor product, we see that the tensor product of n qubits (i.e.n different 2-dimensional vectors) is a vector of size 2^n. Thus, to build a quantum state that stores d numbers in its amplitudes we need $\lceil \log d \rceil$ qubits. More precisely, observe that for a vector $\vec{v} \in \mathcal{H}^d$ if we want to build $|\vec{v}]\rangle = \|\vec{v}\|\vec{v}$ we need only $\lceil \log d \rceil$ numbers. If d is not a power of 2, we consider the vector padded with 0 to the nearest power of 2 bigger than d. This will allow us to build states in the form of $\frac{1}{\|\vec{v}\|}\sum_{i=0}^{\lceil \log d \rceil} v_i|i\rangle$. This fact will used a lot in our quantum algorithms, especially in quantum machine learning. We will discuss in chapter @ref(#chap-classical-data-quantum-computers) how to create these kind of quantum states.

2.2.1 Review of important statements in quantum computation

Before delving into a review of quantum algorithms, we would like to state here a few important lemmas.

Lemma 2.1 (Hadamard on a bitstring (Nielsen and Chuang, 2002))**.** *Let $x \in \{0,1\}^n$ be a bitstring, and H be an Hadamard gate. Then:*

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z_1,\ldots z_n \in \{0,1\}^n} (-1)^{x_1 z_1 + x_2 z_2 + \cdots + x_n z_n} |z_1, \ldots, z_n\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{x^T z} |z\rangle, \tag{2.3}$$

where $x^T z$ is the bitwise inner product of strings in Z_2^n modulo 2.

2.3 Measuring complexity of quantum algorithms

This section is an attempt to organize in a coherent way some fundamental concepts in quantum computer science. Some refereces for this section are (Dörn, 2008), (de Wolf, 2019), and (Kuperberg, 2011). In this section we assume the read is comfortable with the big O notation.

Intuitively, the time complexity of an algorithm is the number of operations the computer has to perform from the beginning to the end of the execution of the algorithm. This model is easy to understand when we think about a personal computer with one CPU (more precisely, a von Neumann architecture). What is the right way of measuring the complexity of (quantum) circuits? For example, suppose to have a quantum circuit of n qubits, and for each qubit we apply one Hadamard gate: what is the time complexity for running this circuit? Is it $O(n)$ or $O(1)$?. We will discuss here different ways for measuring the complexity of a quantum algorithm: the number of gates, the depth of the circuit, and the number of queries to an oracle. The first two are a direct generalization of the measure of complexity of boolean circuits, where we measure the number of gates (serial time) or the depth of the circuit (parallel time). The third measure of complexity is the query complexity, which is the number of times we use (or query) an oracle: a circuit or a data structure that we can use as a black box. In this section we will define more formally the query complexity of a quantum algorithm, and in the next chapter we will explain possible implementations of different kinds of oracles. The *time* complexity, which is ultimately the most relevant measure of complexity of an algorithm for most practical purposes, is more tricky to define properly. If we assume that our hardware is capable of executing all the gates at the same depth in parallel, than the measure of time complexity becomes the depth of the quantum algorithm. We denote with $T(U)$ as the time complexity needed to implement a quantum circuit implementing a unitary U, and this is measured in terms of **number of gates**, i.e. the size of the circuit (this is quite standard see for example the section the introduction in (Ambainis et al., 2022)). This is a concept that bears some similarity with the clock rate of classical CPUs. If our hardware is not capable of executing all the gates at the same depth in one clock, we might have to consider the time complexity as the number of gates (eventually divided by the number of gates that can be executed at the same time). As an important exception to this rule we have the query complexity, where we consider the cost of a single query of an oracle as $O(1)$, i.e. as any other gate. This is somehow justified because for certain oracles we have efficient (i.e. with depth polylogarithmic in the size of the oracle) implementations. For this, the (often non said) assumption is that the part of the quantum computer dedicated to run the main part of the circuit has to be evaluated with serial time (i.e. the number of gates), while the part of the circuit dedicated to executing the oracle has to be evaluated for its parallel time complexity. An example of this, which we will treat in more details in the next chapter is the QRAM and the QRAG gate. This assumption allows to justifiably conflate the query complexity of an algorithm as its time complexity.

Last but not least, everything is made even more complicated by error correction. In fact, in some architecture and some error correction codes, the time complexity of the algorithm is well approximated by the number of T gates of the circuit.

In this book, we use the standard notation of $\widetilde{O}$ to hide the polylogarithmic factors in the big-O notation of the algorithms: $O(n\log(n)) = \widetilde{O}(n)$. The meticulous reader may notice that in this way we might make the notation ambiguous, as it might not be clear what is hidden in the $\widetilde{O}$ notation. The correct way of using this notation is to hide only factors that appear with a dependency bigger than the logarithm (i.e. $O(n\log(n)) = \widetilde{O}(n)$ but $O(n\log(n)\log(1/\epsilon)) = \widetilde{O}(n\log(1/\epsilon)))$. Another possibility, to make our runtimes more precise, is to use a subscript to specify the factors that we choose to hide:

$$\widetilde{O}_{\epsilon,\delta}\left(\frac{1}{\epsilon}\right) = O\left(\frac{1}{\epsilon}\log(1/\epsilon)\log(1/\delta)\right) \tag{2.4}$$

Definition 2.2 (Quantum query or oracle access to a function)**.** Let $\mathcal{H}$ be a finite-dimensional Hilbert space with basis $\{0,1\}^n$. Given $f : \{0,1\}^n \to \{0,1\}^m$, we say that we have quantum query access to f if we have access to a unitary operator U_f on $\mathcal{H} \otimes \mathbb{C}^{2^m}$ such that $U|x\rangle|b\rangle = |x\rangle|b \oplus f(x)\rangle$ for any bit string $b \in \{0,1\}^m$. One application of U_f costs T_f operations.

Definition 2.3 (Quantum computation in the query model)**.** Let O_x be a unitary operator that encodes the input of our computation, and acts in a non-trivial way on its associated Hilbert space. A quantum computation with T queries to an oracle $O_x : |i,b,z\rangle \mapsto |i,b \oplus x_i, z\rangle$ is a sequence of unitary transformations:

$$U_T O_x U_{T-1} O_x \dots U_1 O_x U_0$$

Where each U_t is a unitary that does not depend on the input of the algorithm. The output of the computation is obtained by measuring the rightmost register (by convention).

Note that the second register holds the XOR of the i-th component of the input with the previous state of the register (i.e. the b). This is to make the computation reversible. Importantly, the definition 2.2 is just an example of function for which we can have query access. We can assume query access to unitaries creating various kind of quantum states as output. We will see many examples of oracles as definition 3.8, 3.12, 3.13, and **??**.

This is the so-called query model, or oracle model of (quantum) computation. An important thing here is the last statement of Definition 2.2, on the cost of applying U_f, which is $O(1)$. There are multiple reasons for working in this model, which might appear unrealistic at this time. First, it is often the case that queries to these oracles are actually efficient (as we will see in many example), so the query complexity is actually equivalent (up to multiplicative polylogarithmic factors) to the depth of the quantum circuit that is going to be executed. Another reason is that in the oracle model is relatively simple to prove lower bounds and results about the complexity of an algorithm in terms of the number of queries to an oracle that encodes the input of the problem. It is customary, for complex results in quantum algorithms to separate the study of the query complexity of the problem and the depth or gate complexity of the quantum circuit which is executed on the real hardware. We formalize more this difference in the following definition.

Definition 2.4 (Query complexity of an algorithm)**.** The quantum query complexity of a quantum algorithm $\mathcal{A}$ is the number of queries to a black-box made by $\mathcal{A}$ in order to compute f.

If we just care about the **relativized** complexity, we might limit ourselves to compare two algorithms that solve the same problem in terms of the number of queries to a given oracle, we might observe that one is faster than the other. This is a **relativized** speedup. The oppositive is an **absolute** speedup, i.e. when we also take into account the complexity of the operations that are **not** queries to an oracle. In the case of quantum algorithms, these might simply be the gate depth of the circuit.

Definition 2.5 (Circuit complexity or time complexity)**.** The quantum circuit complexity (or time complexity) of a quantum algorithm $\mathcal{A}$ is the depth of the quantum circuit implementing $\mathcal{A}$.

Quantum computing is not the only place where we measure the complexity in terms of query to an oracle. In fact, it's sufficient to do a few "queries" (pun intended) on your search engine to realize that in many computational models we have adopted this measure of computational complexity.

Note that the query complexity of an algorithm is a lower bound on the gate complexity of the quantum circuit. It is often simpler to study first the query complexity of a quantum algorithm and then study the time complexity. For most quantum algorithms (but not all!) the time complexity coincides with the query complexity, up to a logarithmic factor. Note that, if we find a way to have an oracle whose depth (i.e. circuit complexity) is only (poly)logarithmic in the input size, then the query complexity and the gate complexity coincide up to a negligible polylogarithmic factor. There are some exceptions. Most notably, there is a quantum algorithm for the important *hidden subgroup problem* with only polynomial query complexity, while the classical counterpart has a query complexity that is exponential in the input size. Nevertheless, the overall time complexity of the quantum algorithm is (to date) still exponential, and polynomial-time quantum algorithms are known only for a few specializations of the problem.

We will clarify better some definitions that are used to describe the probabilistic behavior of an algorithm:

Definition 2.6 (Randomized algorithms)**.** Let $f : \{0,1\}^N \mapsto \{0,1\}$ be a Boolean function. An algorithm computes f:

- **exactly** if the outputs equals $f(x)$ with probability 1 for all $x \in \{0,1\}^N$
- with **zero error** if it is allowed to give the answer "UNDEFINED" with probability smaller than 1/2 (but if the output is 0 or 1 it must be correct)

- with **bounded error** if the output equals $f(x)$ with probability greater than 2/3 for all $x \in \{0,1\}^N$.

A bounded error (quantum or classical) algorithm that fails with probability 1/3 (or any other constant smaller than 1/2) is meant to fail *in the worst-case.* We do not expect the algorithm to fail in the average case, i.e. for most of the inputs (see Appendix of (de Wolf, 2019)).

If a (quantum or classical) algorithm is said to output the right answer in **expected** (oftain said "in expectation") running time T, we can quickly create another algorithm that has **worst-case** guarantees on the runtime. This is obtained using the Markov's inequality, i.e. theorem C.2 as follows. Run the algorithm for kT steps, i.e.. stop the execution after kT steps if it hasn't terminated already. If X is the random variable of the runtime of the computation (so $\mathbb{E}[X] = T$), then:

$$Pr\left[X > kT\right] \leq \frac{1}{k}$$

So with probability $\geq 1 - \frac{1}{k}$ we will have the output of the algorithm.

2.4 Review of famous quantum algorithms

In this Chapter we will explore some introductory quantum algorithms. While some of them are not directly related to data analysis nor machine learning, it is important to report them here because they help us better understand the model of quantum computation we adopt. Others will prove to be really useful for the quantum machine learning practitioner.

2.4.1 Deutsch-Josza

Definition 2.7 (Constant function)**.** A function $f : \{0,1\}^n \mapsto \{0,1\}$ is constant if $f(x) = 0 \forall x \in \{0,1\}^n$ or $f(x) = 1 \forall x \in \{0,1\}^n$.

Definition 2.8 (Balanced function)**.** A function $f : \{0,1\}^n \mapsto \{0,1\}$ is balanced if $f(x) = 0$ for half of the inputs and $f(x) = 1$ for the other half.

Theorem 2.1 (Deutsch-Josza (Deutsch and Jozsa, 1992))**.** *Assume to have quantum access (as definition 2.2) to a unitary U_f that computes the function $f : \{0,1\}^n \mapsto \{0,1\}$, which we are promised to be either constant or balanced. There is a quantum algorithm that decides which is the case with probabiliy 1, using U_f only once and using $O(\log(n))$ other gates.*

Proof. We start our quantum computer initializing n qubit as $|0\rangle$ state follwed by a single ancilla qubit initialized in state $|1\rangle$, which we will use for the phase-kickback. Then, we apply the Hadamard transform on each of them. Mathematically, we are performing the following mapping:

$$|0\rangle^{\otimes n}|1\rangle \mapsto \left(\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle\right)|-\rangle \tag{2.5}$$

Now we apply U_f using the first register (i.e. the first n qubits) as input and the ancilla register (the last qubit) as output. Our quantum computer is now in the state

$$\left(\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}|x\rangle\right)|-\rangle$$

Now we apply n Hadamard gates to the n qubits in the first registers. Recalling lemma 2.1, this gives the state

$$\left(\frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}\sum_{j\in\{0,1\}^n}(-1)^{xj}|j\rangle\right)|-\rangle = \left(\frac{1}{2^n}\sum_{x\in\{0,1\}^n}\sum_{j\in\{0,1\}^n}(-1)^{f(x)+xj}|j\rangle\right)|-\rangle$$

In this state, note that the normalization factor has changed from $\frac{1}{\sqrt{2^n}}$ to $\frac{1}{2^n}$, and recall that $(-1)^{xj}$ is read as $(-1)^{\sum_p x_p j_p \bmod 2}$. The key idea of the proof of this algorithm lies in asking the right question to the previous state: what is the probability of measuring the state $|0\rangle^n$ in the first register? The answer to this question will conclude the proof of this theorem. Before looking at the probability, observe that the amplitude of the state $|j=0\rangle$ we will see that it is just $\frac{1}{2^n}\sum_x(-1)^{f(x)}$, as $x^Tj=0$ if $j=0_1\dots 0_n$, for all x. Then,

$$\frac{1}{2^n}\sum_{i\in\{0,1\}^n}(-1)^f(x) = \begin{cases}1 & \text{if } f(x)=0\forall x\\ -1 & \text{if } f(x)=1\forall x\\ 0 & \text{if } f(x)\text{is balanced}\end{cases} \tag{2.6}$$

To conclude, reckon that if the function f is constant (first two cases), we will measure $|0\rangle^{\otimes n}$ with probability 1, and if the function is balanced, we will measure some bitstring of n bits that is different than the string $0_1\dots 0_n$. □

It's simple to see that if we want to solve this problem with a classical *deterministic* algorithm, we need exactly $2^n/2+1$ queries. However, with the usage of a randomized algorithm we can drastically reduce the number of queries by admitting a small probability of failure.

Exercise 2.2. Can you think of an efficient randomized classical algorithm for solving this problem? Perhaps you can use the tools in the Appendix for randomized algorithms.

We now turn our attention to the first learning problem of this book. This is rarely stressed that the following algorithm can be interpreted as a learning algorithm.

2.4.2 Bernstein-Vazirani

Theorem 2.2 (Bernstein-Vazirani). *Assume to have quantum access (as definition 2.2) to a unitary U_f that computes the function $f : \{0,1\}^n \mapsto \{0,1\}$, which computes $f_a(x) = (x,a) = (\sum_i^n x_i a_i) \mod 2$ for a secret string $a \in \{0,1\}^n$. There is a quantum algorithm that learns a with probability 1, using U_f only once and $O(\log(n))$ other gates.*

Proof. The algorithm follows exactly the same steps as the Deutsch-Josza algorithm. The proof is slightly different, and start by noting that, after the application of the oracle U_f, the register of our quantum computer is in the following state:

$$\left(\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}|x\rangle\right)|-\rangle = \left(\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{a^Tx}|x\rangle\right)|-\rangle \qquad (2.7)$$

Now we resort again to Lemma 2.1, and we use the fact that the Hadamard it is also a self-adjoint operator (i.e. it is the inverse of itself: $H^2 = I$). Thus applying n Hadamard gates to the first register leads to the state $|a\rangle$ deterministically. □

Exercise 2.3. Can you think of an efficient randomized classical algorithm for solving Berstain-Vazirani problem? You can use the tools in the Appendix for randomized algorithms.

Other material for learning about Deutsch-Josza and Bernstein-Vazirani algorithms are the lecture notes of Ronald de Wolf that you can find here.

Figure 2.1: Expect to see here Simon's algorithm - Contribute here!

2.4.3 Hadamard test

Let U be a unitary acting on n qubits, and $|\psi\rangle$ a quantum state on n qubit (generated by another unitary V). We also require to be able to apply the controlled version of the unitary U. Then, the Hadamard test is a quantum circuit that we can use to estimate the value of $\langle\psi|U|\psi\rangle$. The circuit is very simple, it consists in a Hadamard gate applied on an ancilla qubit, the controlled application of the unitary U and another Hadamard gate.

The initial operation leads to $(H \otimes V)|0\rangle|0\rangle = |+\rangle|\psi\rangle$, then we have:

$$\begin{aligned}|\psi_{\text{final}}\rangle &= (H \otimes I)(cU)|+\rangle|\psi\rangle = (H \otimes I)\frac{1}{\sqrt{2}}\left(|0\rangle|\psi\rangle + |1\rangle U|\psi\rangle\right)\\ &= \frac{1}{2}\left(|0\rangle\left(|\psi\rangle + U|\psi\rangle\right) + |1\rangle\left(|\psi\rangle - U|\psi\rangle\right)\right)\end{aligned}$$

Note that the last state could be written equivalently, by just factoring out the $|\psi\rangle$ state as $|\psi_{\text{final}}\rangle = \frac{1}{2}\left(|0\rangle(I+U)|\psi\rangle + |1\rangle(I-U)|\psi\rangle\right)$. The probability of measuring 0 in the first qubit is:

$$\begin{aligned}p(0) =& \left\|\frac{1}{2}(I+U)|\psi\rangle\right\|_2^2 = \frac{1}{4}\left(\langle\psi| + \langle\psi|U^\dagger\right)\left(|\psi\rangle + U|\psi\rangle\right) & (2.8)\\ =& \frac{2 + \langle\psi(U+U^\dagger)\psi\rangle}{4} = \frac{2 + 2\text{Re}(\langle\psi|U|\psi\rangle)}{4} & (2.9)\end{aligned}$$

Where we used Postulate 2.3 with the observable $|0\rangle\langle 0| \otimes I$. The probability of measuring 1 in the first register follows trivially.

Exercise 2.4. Can you tell what is the expected value of the observable Z of the ancilla qubit? Remember that the possible outcome of the observable Z are $\{+1, -1\}$.

However, we might be interested in the imaginary part of $\langle\psi|U|\psi\rangle$. To estimate that, we need to slightly change the circuit. After the first Hadamard gate, we apply on the ancilla qubit a phase gate S, which gives to the state $|1\rangle$ a phase of $-i$. To get the intuition behind this, let's recall that the imaginary part of a complex number $z = (a + ib)$ is defined as: $\text{Im}(z) = \frac{z-z^*}{2i} = \frac{i(z-z^*)}{-2} = \frac{-2b}{-2} = b$, where after the definition, we just complicated the series of equations by multiplying the numerator and denominator by i, a trick that we will use later. The rest of the circuit of the Hadamard test stays the same. The evolution of our state in the quantum computer is the following:

$$\begin{aligned}|\psi_{\text{final}}\rangle &= (H \otimes I)(cU)\left(|0\rangle - i|1\rangle\right)|\psi\rangle = (H \otimes I)\frac{1}{\sqrt{2}}\left(|0\rangle|\psi\rangle - i|1\rangle U|\psi\rangle\right) & (2.10)\\ &= \frac{1}{2}\left(|0\rangle\left(|\psi\rangle - iU|\psi\rangle\right) + |1\rangle\left(|\psi\rangle + iU|\psi\rangle\right)\right) & (2.11)\end{aligned}$$

The probability of measuring 0 is given by the following equation.

$$p(0) = \frac{1}{4}\left(\langle\psi| + iU\langle\psi|\right)\left(|\psi\rangle - iU|\psi\rangle\right) = \frac{1}{4}\left(2 - i\langle\psi|U|\psi\rangle + i\langle\psi|U^\dagger|\psi\rangle\right) \quad (2.12)$$

Note that when taking the conjugate of our state, we changed the sign of i. We now have only to convince ourselves that $-i\langle\psi|U|\psi\rangle + i\langle\psi|U^\dagger|\psi\rangle = i\langle\psi|U^\dagger - U|\psi\rangle$ is indeed the real number corresponding to $2\text{Im}(\langle\psi|U|\psi\rangle)$, and thus the whole equation can be a probability.

Exercise 2.5. Can you check if the S gate that we do after the first Hadamard can be performed before the last Hadamard gate instead?

2.4.4 Modified Hadamard test

In this section we complicate a little the results obtained in the previous one, by finding the number of samples that we need to draw out of a circuit in order to estimate the expected value or the probability of interested with a certain level of accuracy and with a certain probability.

Theorem 2.3 (Modified Hadamard test (no amplitude amplification))**.** *Assume to have access to a unitary U_1 that produces a state $U_1|0\rangle = |\psi_1\rangle$ and a unitary U_2 that produces a state $|\psi_2\rangle$, where $|\psi_1\rangle, |\psi_2\rangle \in \mathbb{C}^N$ for $N = 2^n, n \in \mathbb{N}$. There is a quantum algorithm that allows to estimate the quantity $\langle\psi_1|\psi_2\rangle$ with additive precision ϵ using controlled applications of U_1 and U_2 $O(\frac{\log(1/\delta)}{\epsilon^2})$ times, with probability $1-\delta$*

Proof. Create a state $|0\rangle|0\rangle$ where the first register is just an ancilla qubit, and the second register has n qubits. Then, apply an Hadamard gate to the first qubit, so to obtain $|+\rangle|0\rangle$. Then, controlled on the first register being 0, we apply the unitary U_1, and controlled on the register being 1, we apply the unitary U_2. Then, we apply again the Hadamard gate on the ancilla qubit. The state that we obtain is the following:

$$(H \otimes I)\frac{1}{\sqrt{2}}\left(|0\rangle|\psi_1\rangle + |1\rangle|\psi_2\rangle\right) \quad (2.13)$$

$$= \frac{1}{2}\left(|0\rangle(|\psi_1\rangle + |\psi_2\rangle) + |1\rangle(|\psi_1\rangle - |\psi_2\rangle)\right) \quad (2.14)$$

Again, now it is easy to state that the probability of measuring 0 is:

$$p(0) = \frac{2 + 2\text{Re}[\langle\psi_1|\psi_2\rangle]}{4} \quad (2.15)$$

We conclude the proof by recalling the Chernoff bound in theorem C.9, as we did for the proof of the swap test. □

Can you think of the reasons that might lead one to prefer the swap test over the Hadamard test, or vice versa? At the end of the day, aren't they both computing the same thing? For instance, note that for the Hadamard test, we are requiring the ability to call the *controlled* version of the unitaries U_1, and U_2, while for the swap test, we can just treat them as black-boxes: these can be quantum states that we obtain from a quantum process, or that we obtain from a quantum communication channel.

2.4.5 Swap test

The swap test was originally proposed in (Buhrman et al., 2001a), in the context of quantum fingerprinting, but it has been quickly extended to many other context. For us, the swap test is a way to obtain an estimate of an inner product between two quantum states. The difference between the swap test and the Hadamard test is that in this case we don't assume to have access to the unitary creating the states, hence we cannot perform controlled operations with this unitary. You can think that we receive the states from a third-party, i.e. via a communication protocol.

Theorem 2.4 (Swap test (no amplitude amplification))**.** *Assume to have access to a unitary U_1 that produces a state $U_1|0\rangle = |\psi_1\rangle$ and a unitary U_2 that produces a state $|\psi_2\rangle$, where $|\psi_1\rangle, |\psi_2\rangle \in \mathbb{C}^N$ for $N = 2^n, n \in \mathbb{N}$. There is a quantum algorithm that allows to estimate the quantity $|\langle\psi_1|\psi_2\rangle|^2$ with additive precision ϵ using U_1 and U_2 $O(\frac{\log(1/\delta)}{\epsilon^2})$ times with probability $1-\delta$*

Proof. Create a state $|0\rangle|0\rangle|0\rangle$ where the first register is just an ancilla qubit, and the second and third register have n qubits each. Then, apply an Hadamard gate to the first qubit, so to obtain $|+\rangle|0\rangle|0\rangle$. Then, apply U_1 and U_2 to the second and third register, and then apply a controlled swap gate controlled on the ancilla qubit, targeting the two registers. More precisely, we apply n controlled swap gates, each controlling a single qubit of the second and third register. Thus, we obtain the state:

$$\frac{1}{\sqrt{2}}\left[|0\rangle(|\psi_1\rangle|\psi_2\rangle) + |1\rangle(|\psi_2\rangle|\psi_1\rangle)\right] \tag{2.16}$$

we now apply another Hadamard gate on the ancilla qubit, in order to obtain the following state:

$$|\phi\rangle = \frac{1}{\sqrt{2}}\left[\frac{1}{\sqrt{2}}\left(|0\rangle(|\psi_1\rangle|\psi_2\rangle) + |1\rangle(|\psi_1\rangle|\psi_2\rangle)\right) + \frac{1}{\sqrt{2}}\left(|0\rangle(|\psi_2\rangle|\psi_1\rangle) - |1\rangle(|\psi_2\rangle|\psi_1\rangle)\right)\right] \tag{2.17}$$

$$= \frac{1}{2}\left[|0\rangle\left(|\psi_1\rangle|\psi_2\rangle) + |\psi_2\rangle|\psi_1\rangle\right) + |1\rangle\left(|\psi_1\rangle|\psi_2\rangle) - |\psi_2\rangle|\psi_1\rangle\right)\right] \tag{2.18}$$

Now we consider the probability of measuring 0 and 1 in the ancilla qubit. More in detail, we want to estimate $p(0) = \langle\phi|M_0|\phi\rangle$. For this, we recall our Postulate 2.3, and more precisely equation (2.1), with $M_0 = |0\rangle\langle 0| \otimes I$, where I is the identiy operator over n qubits. It is simple to see that $p(0) = \frac{2-2|\langle\psi_1|\psi_2\rangle|^2}{4}$.

By repeating this measurement $O(\log(1/\delta)/\epsilon^2)$ times, duly following the statement of the Chernoff bound in theorem C.9, we have that the number of samples needed to obtain an error ϵ with probability $1-\delta$ is $t = \frac{\log(1/\delta)}{2\epsilon^2}$. Once we have obtained an estimate of $p(0)$, we can estimate the sought-after quantity of interest as $|\langle\psi_1|\psi_2\rangle|^2 = 1 - 2p(0)$. □

Exercise 2.6 (Obtain the absolute value of the inner product)**.** In the previous theorem we obtain an estimate of $|\langle\psi_1|\psi_2\rangle|^2$ with a certain error ϵ. If we just take the square root of that number, what is the error in the estimation of $|\langle\psi_1|\psi_2\rangle|$? You are encouraged to read the section in the appendix D on error propagation.

–>

–>

Chapter 3

Classical data and quantum computers

Contributors: Alessandro Luongo, Jun Hao Hue, Francesco Ghisoni, João F. Doriguello

Version: 0.7.1

Pending updates

In this chapter discuss how to represent and load classically available data on a quantum computer. First, we describe how to represent data, which reduces to understanding the possible ways of storing information in quantum states. Then, we introduce the quantum memory model of computation, which is the model that we use to load data (which we assume to know classically) into a quantum computer. We finally look at the problem of retrieving data from a quantum computer, discussing the complexity of the problem. The main takeaway from this chapter is the understanding of the tools that are often used at the very start and very end of many quantum algorithms, which will set us up to understanding quantum algorithms in future chapters. This chapter can be tought as the study of the I/O interface of our quantum computer.

3.1 Representing data in quantum computers

We'll begin our journey into quantum algorithms by understanding how we can represent and store data as a quantum state. Data plays a key role and is at the heart of most modern algorithms and knowing the best way to encode it on a quantum computer might pave the way for intuitions in solving problems, an essential step towards quantum advantage (as noted also in (Schuld et al., 2015)).

There are various ways to achieve this task. Some are borrowed from classical computation, such as the *binary* encoding, which consist in representing encoding boolean strings of length n using n qubits, while some leverage quantum properties, such as the *amplitude* encoding, which consits in representing vectors as linear combination of computational basis. We note that some of the presented schemes depend heavily on the accuracy of the available quantum computers in manipulating quantum states (i.e. developments in metrology and sensing). For example, techniques that rely on precise amplitudes of a state will be hindered by the current noisy hardware, or incour in high overhead of the quantum error correction. Considerations on the practical feasibility of an encoding technique are out of scope for this book.

3.1.1 Binary encoding

The first method is a way to represent natural numbers on a quantum computer by using the binary expansion of the number to determine the state of a sequence of qubits. Each qubit is set to either the state $|0\rangle$ or $|1\rangle$, corresponding to each bit in the binary representation of the number. To represent a natural number $x \in \mathbb{N}$ on a quantum computer, we consider the binary expansion of x as a list of m bits, and we set the state of the i^{th} qubit as the value of the i^{th} bit of x:

$$|x\rangle = \bigotimes_{i=0}^{m} |x_i\rangle. \tag{3.1}$$

When extending this definition to signed integers we can, for example, use an additional qubit to store the sign of $x \in \mathbb{Z}$. Another possibility, is to represent signed integer using 2s complement. This is actually the representation of choice for classical and quantum arithmetic (Luongo et al., 2024). For real numbers we consider that, as on classical computers, $x \in \mathbb{R}$ can be approximated with binary representation up to a certain precision. As before, we need a bit to store the sign, some bits to store the integer part, and some bits to store the fractional part. This is more precisely stated in the following definition, which is a possible way to represent number with fixed precision.

Definition 3.1 (Fixed-point encoding of real numbers (Rebentrost et al., 2021))**.** Let c_1, c_2 be positive integers, and $a \in \{0,1\}^{c_1}$, $b \in \{0,1\}^{c_2}$, and $s \in \{0,1\}$ be bit strings. Define the rational number as

$$\mathcal{Q}(a,b,s) := (-1)^s \left(2^{c_1-1}a_{c_1} + \dots + 2a_2 + a_1 + \frac{1}{2}b_1 + \dots + \frac{1}{2^{c_2}}b_{c_2}\right) \in [-R, R], \tag{3.2}$$

where $R = 2^{c_1} - 2^{-c_2}$.

If c_1, c_2 are clear from the context, we use the shorthand notation for a number $z := (a,b,s)$ and write $\mathcal{Q}(z)$ instead of $\mathcal{Q}(a,b,s)$. Given an n-dimensional vector $v \in (\{0,1\}^{c_1} \times \{0,1\}^{c_2} \times \{0,1\})^n$ the notation $\mathcal{Q}(v)$ means an n-dimensional vector whose j-th component is $\mathcal{Q}(v_j)$, for $j \in [n]$.

We note that the choice of c_1 and c_2 in definition 3.1 depends both on the problem at hand and the implemented algorithm. For the purposes of optimizing a quantum circuit, these constants can be dynamically changed. For example, if at some point of a computation we are required to work with numbers between 0 and 1, then we can neglect the c_1 bits.

One of the utilities of having a definition to express numbers on a quantum computer to a fixed point precision is the analysis of numerical errors, which is essential to ensure the validity of the solution. This is often done numerically (via simulations, which we will discuss in Chapter 12), or during the implementation of the algorithm on real hardware. This binary encoding encompasses other kinds of encoding like 2-complement encoding and a possible quantum implementation of floating point representation. Howevever, we observe that the floating point encoding has a relatively high circuital overhead and, therefore, is not a common choice. A further layer of complexity arises in understanding how to treat arithmetic operations. This is addressed in the section below.

3.1.1.1 Arithmetic model

The advantage of using a binary encoding is that we can use quantum circuits for arithmetic operations. As we will discuss more in depth in 3.3.1.1 any Boolean circuit can be made reversible, and any reversible circuit can be implemented using single-qubit NOT gates and three-qubit Toffoli gates. Since most of the classical Boolean circuits for arithmetic operations operate with a number of gates of $O(\text{poly}(c_1, c_2))$, this implies a number of quantum gates of $O(\text{poly}(c_1, c_2))$ for the corresponding quantum circuit. Extending the analogy with classical computation allows us to introduce the arithmetic model of computation for performing operations on binary encoded numbers in constant time.

Definition 3.2 (Quantum arithmetic model (Doriguello et al., 2022))**.** Given $c_1, c_2 \in \mathbb{N}$ specifying fixed-point precision numbers as in Definition 3.1, we say we use a quantum arithmetic model of computation if the four arithmetic operations can be performed in constant time in a quantum computer.

Beware that using Definition 3.1 is not the only possibile choice. For example, most of the non-modular and modular arithemtic circuits are expressed in 2s complement. For a comprehensive and optimized list of results about this topic, the interest reader can read (Luongo et al., 2024). As for the classical counterpart, a quantum algorithm's complexity does not take into account the cost of performing arithmetic operations, as the number of digits of precisions used to represents numbers is a constant, and does not depend on the input size. However, when estimating the resources needed to run an algorithm on a quantum computer, specifying these values becomes important. For a good example of a complete resource analysis, including arithmetic operations in fixed-point precision, of common algorithms in quantum computational finance we refer to (Chakrabarti et al., 2021).

3.1.2 Amplitude encoding

Amplitude encoding is a way to represent a vector x of size n (where n is a power of 2) in the amplitudes of an $\log(n)$ qubit pure state. We can map a vector $x \in \mathbb{R}^N$ (or even $\in \mathbb{C}^N$) to the following quantum state:

$$|x\rangle = \frac{1}{\|x\|} \sum_{i=0}^{N-1} x_i |i\rangle = \|x\|^{-1} x, \tag{3.3}$$

Sometimes, amplitude encoding is also known as *quantum sampling access*, as sampling from a quantum state we prepare can also be interpreted as sampling from a probability distribution. In fact, if the amplitude of a computational basis $|i\rangle$ is α_i, then we sample $|i\rangle$ with probability $|\alpha_i|^2$. Observe from the state in the above equation we are actually representing an ℓ_2-normalized version of the vector x. Therefore we have "lost" the information on the norm of the vector x. However we will see how this is not a problem when we work with more than one vector. This type of encoding can be generalized to matrices. Let $x(i)$ be the i-th row of $X \in \mathbb{R}^{n \times d}$, a matrix with n rows and d columns (here we take again n and d to be a power of 2). Then we can encode X with $\lceil log(d) \rceil + \lceil log(n) \rceil$ qubits as:

$$|X\rangle = \frac{1}{\sqrt{\sum_{i=1}^{n} \|x(i)\|^2}} \sum_{i=1}^{n} \|x(i)\| |i\rangle |x(i)\rangle \tag{3.4}$$

Exercise 3.1. Check that Equation (3.4) is equivalent to

$$|X\rangle = \frac{1}{\sqrt{\sum_{i,j=1}^{n,d} |X_{ij}|^2}} \sum_{i,j=1}^{n,d} X_{ij} |i\rangle |j\rangle, \tag{3.5}$$

From an algorithms perspective, amplitude encoding is useful because it requires a logarithmic number of qubits with respect to the vector size, which might seem to lead to an exponential saving in physical resources when compared to classical encoding techniques. A major drawback of amplitude encoding is that, in the worst case, for the majority of states, it requires a circuit of size $\Omega(N)$.

3.1.3 Block encoding

Block encoding is another type of encoding for working with matrices on a quantum computer. More precisely, we want to encode a matrix into a unitary that has a circuit representation on a quantum computer. As it will become clear in the next chapters, being able to perform such encoding unlocks many possibilities in terms of new quantum algorithms.

Definition 3.3 (Block encoding of a matrix)**.** Let $A \in \mathbb{R}^{N \times N}$ be a square matrix for $N = 2^n$ for $n \in \mathbb{N}$, and let $\alpha \geq 1$. For $\epsilon > 0$, we say that a $(n + a)$-qubit unitary U_A is a (α, a, ϵ)-block encoding of A if

$$\|A - \alpha(\langle 0|^{\otimes a} \otimes I)U_A(|0\rangle^{\otimes a} \otimes I)\|_2 \leq \epsilon \tag{3.6}$$

It is useful to observe that an (α, a, ϵ)-block encoding of A is just a $(1, a, \epsilon)$-block encoding of A/α. Often, we do not want to take into account the number of qubits a we need to create the block encoding because these are expected to be negligible. Therefore, some definitions of block encoding in the literature use the notation (α, ϵ) or the notation α-block encoding if the error is 0. Note that the matrix $U_A \in \mathbb{R}^{(N+\kappa)\times(N+\kappa)}$ has the matrix A encoded in the top-left part:

$$U_A = \begin{pmatrix} A & \cdot \\ \cdot & \cdot \end{pmatrix}. \tag{3.7}$$

3.1.4 Angle encoding

Another way to encode vectors, as defined by (Schuld and Petruccione, 2021), is with angle encoding. This technique encodes information as angles of the Pauli rotations $\sigma_x(\theta)$, $\sigma_y(\theta)$, $\sigma_z(\theta)$. Given a vector $x \in \mathbb{R}^n$, with all elements in the interval $[0, 2\pi]$; the technique seeks to apply $\sigma_\alpha^i(x_i)$, where $\alpha \in \{x, y, z\}$ and i refers to the target qubit. The resulting state is said to be an angle encoding of x and has a form given by

$$|x\rangle = \prod_{i=1}^{n} \sigma_\alpha^i(x_i)|0\rangle^{\otimes n} \tag{3.8}$$

This technique's advantages lies in its efficient resource utilization, which scales linearly for number of qubits. One major drawback is that it is difficult to perform arithmetic operations on the resulting state, making it difficult to apply to quantum algorithms.

3.1.5 Graph encoding

A graph is as a tuple $G = (V, E)$, where V are the vertices in the graph and E are the edges, where $E \subseteq V \times V$. For graph encoding we require unidirected graphs in which if $(v_i, v_j) \in E$ then $(v_j, v_i) \in E$. Unidirected graphs can be either simple or multigraphs. A simple unidirected graph is one without self loops and at most a single edge connecting two vertices, whilst a unidirected multigraph can have self loops or multiple edges between two vertices. Graph encoding is possible for unidirected multigraphs with self loops but at most a single edge between two vertices.

A graph G will be represented as an $N = |V|$ qubit pure quantum state $|G\rangle$ such that

$$K_G^v |G\rangle = |G\rangle, \quad \forall v \in V \tag{3.9}$$

where $K_G^v = \sigma_x^v \prod_{u \in N(v)} \sigma_z^u$, and σ_x^u and σ_z^u are the Pauli operators σ_x and σ_z applied to the u^{th} qubit.

Given a graph G with V vertices and edges E, take $N = |V|$ qubits in the $|0\rangle^{\otimes N}$ state, apply $H^{\otimes N}$, producing the $|+\rangle^{\otimes N}$ state where $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. Then apply a controlled Z rotation between qubits connected by an edge in E. It is worth noting that 2 different graphs can produce the same graph state $|G\rangle$. In particular if a graph state $|\tilde{G}\rangle$ can be obtained from a graph state $|G\rangle$ by only applying local Clifford group operators, the 2 graphs are said to be LC-equivalent. The work by (Zhao et al., 2016) has interesting application of this type of encoding.

3.1.6 One-hot encoding

Another possible way of encoding vectors as quantum states introduced by (Mathur et al., 2022) is **one-hot amplitude encoding**, also known as **unary amplitude encoding**, which encodes a normalized vector $x \in \mathbb{C}^n$ onto n qubits. The vector values $x_i \in \mathbb{C}$ will be stored in the amplitudes of the states that form the 2^n-dimensional canonical basis, i.e, the states with only one 1 and the rest 0s. This corresponds to preparing the state:

$$|x\rangle = \frac{1}{||x||} \sum_{i=1}^{n} x_i |e_i\rangle, \tag{3.10}$$

where, for some integer i, the states e_i take the form $e_i = 0^{i-1}10^{n-i}$.

3.2 Quantum memory

Having seen possible ways to represent data on a quantum computer, we will now take the first step toward understanding how to create quantum states that are representing numbers using these encodings. The first step involves understanding quantum memory, which plays a key role in various quantum algorithms/problems such as: Grover's search, solving the dihedral hidden subgroup problem, collision finding, phase estimation for quantum chemistry, pattern recognition, machine learning algorithms, cryptanalysis, and state preparation.

To work with quantum memory we need to define a quantum memory model of computation, which enables us to accurately calculate the complexity of quantum algorithms. In this framework we divide a quantum computation into a data pre-processing step and a computational step. Quantum memory allows us to assume that the pre-processed data can be easily accessed (as in classical computers). In this model, since the pre-processing is negligible, and has to be performed only once, the complexity of a quantum algorithm is fully characterized by the computational step. This understanding formalizes a quantum computation in two distinct components: a quantum processing unit and a quantum memory device. Two notable examples of quantum memory devices are the quantum random access memory (QRAM) and the quantum random access gates (QRAG). It is important to note that having access to a quantum memory device is associated with fault tolerant quantum computers.

This section will first introduce the quantum memory model of computation (3.2.1). This will be followed by the formalization of a quantum computation in the memory model via a quantum processing unit (QPU) and a quantum memory device (QMD) (3.2.2), where the QRAM and QRAG will be presented as possible implementation of the QMD.

3.2.1 The quantum model of computation with and without memory

As discussed in the Section 2.3 of the previous chapter, in quantum computing we often work in a oracle model, also called black-box model of quantum computation. This section is devoted to the formalization of this model of computation. The word "oracle", (a word referencing concepts in complexity theory), is used to imply that an application of the oracle has $\mathcal{O}(1)$ cost, i.e. we do not care about the cost of implementing the oracle in our algorithm. A synonym of quantum oracle model is quantum query model, which stresses the fact that we can only use the oracle to perform queries.

To appreciate the potential of quantum algorithms, it is important to understand the quantum memory model. This is because we want to compare quantum algorithms with classical algorithms. Understanding the quantum memory model makes sure that we not favor any of the approaches, ensuring a fair evaluation of their performance across different computational models and memory architectures. Understanding classical memory is also important for classical algorithms. Memory limitations make the analysis of big datasets challenging. This limitation is exacerbated when the random-access memory is smaller than the dataset to analyze, as the bottleneck of computational time switch from being the number of operations to the time to move data from the disk to the memory. Hence, algorithms with super linear runtime (such as those based on linear algebra) become impractical for large input size.

As we will formalize later, the runtime for analyzing a dataset represented by a matrix $A \in \mathbb{R}^{n\times d}$ using a quantum computer is given by the time to preprocess the data (i.e., creating quantum accessible data structures) and the runtime of the quantum algorithm. Importantly, the pre-processing step needs to be done only once, allowing one to run (different) quantum algorithms on the same matrix. We can see this pre-processing step as a way of encoding and/or storing the data: once the matrix is pre-processed, we can always retrieve the matrix in the original representation (i.e. it is a loss less encoding). This step bears some similarities with the process of loading the data from the disk in RAM Therefore, because the pre-processing step is analyzed differently from the runtime, when we work with a quantum algorithm that has quantum access to some classical data, we have the following model in mind.

Definition 3.4 (Costing in the quantum memory model)**.** An algorithm in the quantum memory model that processes a data-set of size m has two steps:

- A pre-processing step with complexity $\widetilde{O}(m)$ that constructs an efficient quantum access to the data
- A computational step where the algorithm has quantum access to the data structures constructed in step 1.

The complexity of the algorithm in this model is measured by the cost for step 2.

Let's consider an example. We will see that many of the quantum algorithms considered in this book have a computational complexity expressed (in number of operations of a certain kind) as some functions of the matrix and the problem. Consider a classical algorithm with a runtime of $\widetilde{O}\left(\frac{\|A\|_0 \kappa(A)}{\epsilon^2}\log(1/\delta)\right)$ calls to the classical memory (and coincidentally, CPU operations). Here ϵ is some approximation error in the quantity we are considering, $\kappa(A)$ is the condition number of the matrix, δ the failure probability. The quantum counterpart of this algorithm has a runtime of $O(\|A\|_0)$ classical operation for pre-processing and

$$\widetilde{O}(\text{poly}(f(A)), \text{poly}(\kappa(A)), \text{poly}(1/\epsilon), \text{poly}(\log(nd)), \text{poly}(\log(1/\delta))) \tag{3.11}$$

queries to the quantum memory (and coincidentally, number of operations). Here, $f(A)$ represents some size-independent function of the matrix that depends on the properties of A which can be chosen to be $\|A\|_F$: the Frobenius norm of the matrix. Importantly, note that in the runtime of the quantum algorithm there is no dependence on $\|A\|_0$.

The first step, i.e., loading data (for example a matrix A) onto a quantum memory gives an additive price of $\widetilde{O}(\|A\|_0)$, and is computationally easy to implement. In some cases this can be done on the fly, with only a single pass over the dataset, for example while receiving each of the rows of the matrix. For more complex choices of $f(A)$, the construction of the data structure needs only a few (constant) numbers of passes over the dataset. As pre-processing the data is negligible, we expect quantum data analysis to be faster than classical data analysis. However, there is no need to employ quantum data analysis for small datasets if classical data analysis is sufficient.

In the quantum memory model, we assumed that the pre-processing step to be negligible in cost, and thereby claim that there is significant practical speedup when using quantum algorithms compared to classical algorithms. However, a proper comparison for practical applications needs to include the computational cost of the loading process, which may or may not remove the exponential gap between the classical and the quantum runtime. Nevertheless, even when the pre-processing step is included, we expect the overall computational cost to largely favor the quantum procedure. This analysis can be done only with a proper understanding of the quantum memory model.

Having a clear and deep understanding of the quantum memory model can help us understand the power and limitations of classical computers as well. The past few years saw a trend of works proposing "dequantizations" of quantum machine learning algorithms. These algorithms explored and sharpened some ideas (Tang, 2018) to leverage a classical data structure to perform importance sampling on input data to have classical algorithm with polylogarithmic runtimes in the size of the input. This data structure is very similar to the one used in many quantum machine learning algorithms (see Section 3.3.2.2). As a result, many quantum algorithms which had an exponential separation with their classical counterpart now have at most a polynomial speedup compared to the classical algorithm. However, these classical algorithms have a worse dependence in other parameters (like condition number, Frobenius norm, rank, and so on) that will make them disadvantageous in practice (i.e., they are slower than the fastest classical randomized algorithms (Arrazola et al., 2020)). With that said, having small polynomial speedup is not something to be critical about: even constant speedups matter a lot in practice! Overall, dequantizations and polynomial speedups highlight the importance of clearly understanding the techniques behind loading classical data in quantum computers.

3.2.2 The quantum processing unit and quantum memory device

In this section, we formally define a model of a quantum computer with quantum access to memory. We can intuitively understand this model by separating the available Hilbert space in two: a part dedicated to computing, the Quantum Processing Unit (QPU) and a part dedicated to storing, the Quantum Memory Device (QMD).

The qubits which comprise the QPU are assigned to either an input register or a workspace register; whilst the qubits which comprise the QMD are assigned to either a ancillary register or a memory register. Two other registers, the address register and the target register, are shared by the QPU and QMD and allow for communication between the two Hilbert spaces. A depiction of the architecture of a QPU with access to a QMD can be seen in Figure 3.1. Before defining a model of a quantum computer with quantum access to memory, we will first formally define a computation with only the quantum processing unit QPU.

Definition 3.5 (Quantum Processing Unit (Allcock et al., 2023))**.** A Quantum Processing Unit(QPU) of size m is defined as a tuple $(\mathtt{I}, \mathtt{W}, \mathcal{G})$ consisting of

- an $m_{\mathtt{I}}$-qubit Hilbert space called *input register* $\mathtt{I}$;
- an $(m - m_{\mathtt{I}})$-qubit Hilbert space called *workspace* $\mathtt{W}$;
- a constant-size universal gate set $\mathcal{G} \subset \mathcal{U}(\mathbb{C}^{4\times 4})$.

The qubits in the workspace $\mathtt{W}$ are called ancillary qubits or simply ancillae. An input to the QPU, or quantum circuit, is a tuple $(T, |\psi_{\mathtt{I}}\rangle, C_1, \dots, C_T)$ where $T \in \mathbb{N}$, $|\psi_{\mathtt{I}}\rangle \in \mathtt{I}$, and, for each $t \in \{1, \dots, T\}$, $C_t \in \mathcal{I}(\mathcal{G})$ is a set of instructions from a set $\mathcal{I}(\mathcal{G})$ of possible instructions. Starting from the state $|\psi_0\rangle := |\psi_{\mathtt{I}}\rangle|0\rangle_{\mathtt{W}}^{\otimes(m-m_{\mathtt{I}})}$, at each time step $t \in \{1, \dots, T\}$ we obtain the state $|\psi_t\rangle = C_t|\psi_{t-1}\rangle \in \mathtt{I} \otimes \mathtt{W}$. The instruction set $\mathcal{I}(\mathcal{G}) \subset \mathcal{U}(\mathbb{C}^{2^m \times 2^m})$ consists of all m-qubit unitaries on $\mathtt{I} \otimes \mathtt{W}$ of the form

$$\prod_{i=1}^{k} (\mathsf{U}_i)_{\to I_i} \tag{3.12}$$

for some $k \in \mathbb{N}$, $\mathsf{U}_1, \dots, \mathsf{U}_k \in \mathcal{G}$ and pair-wise disjoint non-repeating sequences $I_1, \dots, I_k \in [m]^{\leq 2}$ of at most 2 elements. We say that $\sum_{i=1}^{k} |I_i|$ is the *size* of the corresponding instruction. We say that T is the *depth* of the input to the QPU, while its *size* is the sum of the sizes of the instructions $C_1, \dots, C_T$.

Note that in this definition the circuit size differs from the standard notion of circuit size, which is the number of selected gates from $\mathcal{G}$, up to a factor of at most 2.

Exercise 3.2. Can you explain why the circuit size differs from the standard notion of circuit size by up to a factor of at most 2

Moreover, in this framework, the locations of the address and target registers are fixed. One could imagine a more general setting where the address and target registers are freely chosen from the workspace. This case can be handled by this model with minimal overhead, e.g. by performing ℓ-SWAP gates to move the desired workspace qubits into the address or target register locations.

Adding access to a QMD changes how we define the model of computation. In practice, a call to the QMD sees the address register selecting a unitary from a set of unitaries $\mathcal{V}$ and applying it to both the target and memory register. It is important to stress that even though a call to the QMD might require gates from a universal gate set, the underlying quantum circuit implementing such a call is *fixed*, i.e., does not change throughout the execution of a quantum algorithm by the QPU, or even between different quantum algorithms. Below we find the full definition of a quantum computation of a QPU with access to a QMD.

Definition 3.6 (Quantum Processing Unit and Quantum Memory Device (Allcock et al., 2023))**.** We consider a model of computation comprising a Quantum Processing Unit(QPU) of size poly $\log(n)$ and a Quantum Memory Device (QMD) of n memory registers, where each register is of ℓ-qubit size (for n a power of 2). A QPU and a QMD are collectively defined by a tuple $(\mathtt{I}, \mathtt{W}, \mathtt{A}, \mathtt{T}, \mathtt{Aux}, \mathtt{M}, \mathcal{G}, \mathsf{V})$ consisting of

- two $(\text{poly}\log n)$-qubit Hilbert spaces called *input register* $\mathtt{I}$ and *workspace* $\mathtt{W}$ owned solely by the QPU;
- a $(\log n)$-qubit Hilbert space called *address register* $\mathtt{A}$ shared by both QPU and QMD;
- an ℓ-qubit Hilbert space called *target register* $\mathtt{T}$ shared by both QPU and QMD;
- a $(\text{poly}n)$-qubit Hilbert space called *auxiliary register* $\mathtt{Aux}$ owned solely by the QMD;
- an $n\ell$-qubit Hilbert space called *memory* $\mathtt{M}$ comprising n registers $\mathtt{M}_0, \dots, \mathtt{M}_{n-1}$, each containing ℓ qubits, owned solely by the QMD;
- a constant-size universal gate set $\mathcal{G} \subset \mathcal{U}(\mathbb{C}^{4\times 4})$;
- a function $\mathsf{V} : [n] \to \mathcal{V}$, where $\mathcal{V} \subset \mathcal{U}(\mathbb{C}^{2^{2\ell}\times 2^{2\ell}})$ is a $O(1)$-size subset of 2ℓ-qubit gates.

The qubits in $\mathtt{W}$, $\mathtt{A}$, $\mathtt{T}$, and $\mathtt{Aux}$ are called ancillary qubits or simply ancillae. An input to the QPU with a QMD, or quantum circuit, is a tuple $(T, |\psi_{\mathtt{I}}\rangle, |\psi_{\mathtt{M}}\rangle, C_1, \dots, C_T)$ where $T \in \mathbb{N}$, $|\psi_{\mathtt{I}}\rangle \in \mathtt{I}$, $|\psi_{\mathtt{M}}\rangle \in \mathtt{M}$, and, for each $t \in \{1, \dots, T\}$, $C_t \in \mathcal{I}(\mathcal{G}, \mathsf{V})$ is an instruction from a set $\mathcal{I}(\mathcal{G}, \mathsf{V})$ of possible instructions. The instruction set $\mathcal{I}(\mathcal{G}, \mathsf{V})$ is the set $\mathcal{I}(\mathcal{G})$ acting on $\mathtt{I} \otimes \mathtt{W} \otimes \mathtt{A} \otimes \mathtt{T}$ augmented with the call-to-the-QMD instruction that implements the unitary

$$|i\rangle_{\mathtt{A}}|b\rangle_{\mathtt{T}}|x_i\rangle_{\mathtt{M}_i}|0\rangle_{\mathtt{Aux}}^{\otimes \text{poly} n} \mapsto |i\rangle_{\mathtt{A}}(\mathsf{V}(i)|b\rangle_{\mathtt{T}}|x_i\rangle_{\mathtt{M}_i})|0\rangle_{\mathtt{Aux}}^{\otimes \text{poly} n}, \qquad \forall i \in [n], b, x_i \in \{0,1\}^{\ell}. \tag{3.13}$$

Starting from $|\psi_0\rangle|0\rangle_{\mathtt{Aux}}^{\otimes \text{poly} n}$, where $|\psi_0\rangle := |\psi_{\mathtt{I}}\rangle|0\rangle_{\mathtt{W}}^{\otimes \text{poly}\log n}|0\rangle_{\mathtt{A}}^{\otimes \log n}|0\rangle_{\mathtt{T}}^{\otimes \ell}|\psi_{\mathtt{M}}\rangle$, at each time step $t \in \{1, \dots, T\}$ we obtain the state $|\psi_t\rangle|0\rangle_{\mathtt{Aux}}^{\otimes \text{poly} n} = C_t(|\psi_{t-1}\rangle|0\rangle_{\mathtt{Aux}}^{\otimes \text{poly} n})$, where $|\psi_t\rangle \in \mathtt{I} \otimes \mathtt{W} \otimes \mathtt{A} \otimes \mathtt{T} \otimes \mathtt{M}$.

This model can be seen as a refined version of the one described in (Buhrman et al., 2022), where the authors divide the qubits of a quantum computer into work and memory qubits. Given M memory qubits, their workspace consists of $O(\log M)$ qubits, of which the address and target qubits are always the first $\lceil \log M \rceil + 1$ qubits. However, address and target qubits are not considered to be shared by the QMD, and there is no mention of ancillary qubits mediating a call to the QMD. The inner structure of the QMD is abstracted away by assuming access to the unitary of a QRAG (see Definition 3.10 later). This model, in contrast, "opens" the quantum memory device, and allows for general fixed unitaries, including QRAM and QRAG.

In addition, this model does not include measurements. These can easily be performed on the output state $|\psi_T\rangle$ if need be. Furthermore the position of the qubits is not fixed within the architecture, allowing for long-range interactions through, for example, multi-qubit entangling gates. This feature is not always

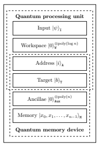

Quantum processing unit
Input $|\psi\rangle_{\mathtt{I}}$
Workspace $|0\rangle_{\mathtt{W}}^{\otimes \mathrm{poly}(\log n)}$
Address $|i\rangle_{\mathtt{A}}$
Target $|b\rangle_{\mathtt{T}}$
Ancillae $|0\rangle_{\mathtt{Aux}}^{\otimes \mathrm{poly}(n)}$
Memory $|x_0, x_1, \ldots, x_{n-1}\rangle_{\mathtt{M}}$
Quantum memory device

Figure 3.1: The architecture of a Quantum Processing Unit(QPU) with access to a quantum memory device(QMD). The QPU is composed of a poly $\log(n)$-qubit input register I and workspace W. The QMD is composed of an nl-qubit memory array M, composed of n memory cells each of size l-qubits, and a poly(n)-qubit auxiliary register Aux. Two registers, the $\log(n)$-qubit address register A and an l-qubit target register T, are shared between the QPU and the QMD.

possible in physical the real world since some quantum devices, such as superconducting quantum computers, don't allow for long-range interactions between qubits. For a model of computation which take in consideration physically realistic device interactions we suggest the work by (Beals et al., 2013).

We stress the idea that call to the QMD is defined by the function V and quantum memory device is defined by the unitary that it implements. In many applications, one is interested in some form of reading a specific entry from the memory, which corresponds to the special cases where the $\mathsf{V}(i)$ unitaries are made of controlled single-qubit gates, and to which the traditional QRAM belongs.

3.2.2.1 The LAQCC model of quantum computation

We can also consider an alternative model of quantum computation to the one in Definition **??**: the LAQCC model, which stands for Local Alternating Quantum Classical Computations. LAQCC is a hybrid quantum-classical computational framework (Buhrman et al., 2023a). This model assumes quantum hardware with specific topology and connectivity, enabling the measurement of certain (but not necessarily all) qubits. The measurement results are then used to perform intermediate classical computations, which, in turn, control subsequent quantum circuits. This process can alternate between quantum and classical layers multiple times, allowing for iterative computation. The authors did not consider an explicit formulation of the quantum memory device. However, it is possible to perform amplitude encoding of a certain class of states in constant depth. Formally, it is defined as follows.

Definition 3.7 (Local Alternating Quantum Classical Computations (Buhrman et al., 2023a))**.** Let LAQCC$(\mathcal{Q}, \mathcal{C}, d)$ be the class of circuits such that:

- Every quantum layer implements a quantum circuit $Q \in \mathcal{Q}$ constrained to a grid topology;

- Every classical layer implements a classical circuit $C \in \mathcal{C}$;
- There are d alternating layers of quantum and classical circuits;
- After every quantum circuit Q, a subset of the qubits is measured;
- The classical circuit receives input from the measurement outcomes of previous quantum layers;
- The classical circuit can control quantum operations in future layers.

The allowed gates in the quantum and classical layers are given by $\mathcal{Q}$ and $\mathcal{C}$, respectively. Furthermore, we require a circuit in $\text{LAQCC}(\mathcal{Q}, \mathcal{C}, d)$ to deterministically prepare a pure state on the all-zeroes initial state.

3.2.2.2 The QRAM

A type of QMD of particular interest is the QRAM, which is the quantum equivalent of a classical Random Access Memory (RAM), that stores classical or quantum data and allows for superposition-based queries. More specifically, a QRAM is a device comprising a memory register that stores data, an address register that points to the memory cells to be addressed, and a target register into which the content of the addressed memory cells is copied. If necessary, it also includes an auxiliary register supporting the overall operation, which is reset to its initial state at the end of the computation. Formally, we define it as:

Definition 3.8 (Quantum Random Access Memory)**.** Let $n \in \mathbb{N}$ be a power of 2 and $f(i) = \mathsf{X}$ for all $i \in [n]$. A *quantum random access memory* QRAM of memory size n is a QMD with $\mathsf{V}(i) = \mathsf{C}_{\mathtt{M}_i}\text{-}\mathsf{X}_{\to\mathtt{T}}$. Equivalently, it is a QMD that maps

$$|i\rangle_{\mathtt{A}}|b\rangle_{\mathtt{T}}|x_0, \dots, x_{n-1}\rangle_{\mathtt{M}} \mapsto |i\rangle_{\mathtt{A}}(f(i)^{x_i}|b\rangle_{\mathtt{T}})|x_0, \dots, x_{n-1}\rangle_{\mathtt{M}} \qquad \forall i \in [n], b, x_0, \dots, x_{n-1} \in \{0,1\}. \tag{3.14}$$

A unitary performing a similar mapping often goes under the name of quantum read-only memory (QROM) The difference with QRAM is that that this term stresses that they don't allow data to be added or modified. Oftentimes, the authors using this term are considering a circuit, as described in section **??**.

Instead assuming to have access to a QRAM requires a protocol for the preprocessing of the data and creation of a data structure in time which is asymptotically linear in the data size (as indicated by definition **??**).

Equipped with definition 3.8) we can formalize what it means to have quantum query access, which is also referred to as QRAM access or as having "x is in the QRAM". We will formalize the case of having a vector $x \in (\{0,1\}^m)^N$ stored in the QRAM.

Definition 3.9 (Quantum query access to a vector stored in the QRAM)**.** Given $x \in (\{0,1\}^m)^N$, we say that we have quantum query access to x stored in the

QRAM if we have access to a unitary operator U_x such that $U_x|i\rangle|b\rangle = |i\rangle|b \oplus x_i\rangle$ for any bit string $b \in \{0,1\}^m$.

In practical terms when analyzing the complexity of a quantum algorithm with a QRAM we need to take in consideration three factors: the circuit size of the quantum algorithm as introduced in definition **??**, the number of queries to the QRAM and the complexity each QRAM query. We emphasize that the complexity that arises due to a query to the QRAM is still an open question. Details of some possible implementations will be discussed in section 3.3.

Table 3.1: Table for QRAM for $N = 2^n$ memory elements of size m. 2 is (Giovannetti et al., 2008a), 4 is (Low et al., 2018), 5 and 6 are form (Allcock et al., 2023), 8 is Lemma 10 from (Yuan and Zhang, 2023). Gate count or depth with * is expressed in number of T gates. Results expressed with + are in terms of FanOut gates of arity $O(n)$. TODO number 4 should have a parameter lambda for number of ancilla qubits?

	Method	Size	Ancilla	Depth	Comment
1	Multiplexer	O(nm)		O(nm)	
2	Bucket Brigade				
3	FanOut BB				Uses FanOut gates
4	Trading	$O(n)$	$2n+2$	$O(n)$	
5	One-hot encoding FanOut	$6n\log n + O(n\log\log n)$ $^{+}$	$2n\log n\log\log n + O(n\log n)$	$O(1)$	
6	One-hot encoding GT-gate	$6n\log n + O(n\log\log n)$ $^{+}$	$2n\log n\log\log n + O(n\log n)$	$O(1)$	
7	Fourier				
8	Circuit	$O(N2^m)$	λ		$\lambda > Nm$

3.2.2.3 The QRAG

Another type of quantum memory device is the quantum random access gate(QRAG). This quantum memory device was introduced in the paper of (Ambainis, 2007) and performs a SWAP gate between the target register and some portion of the memory register specified by the address register. The QRAG finds applications in quantum algorithms for element distinctness, collision finding and random walks on graphs. The formal definition is:

Definition 3.10 (Quantum Random Access Gate)**.** Let $n \in \mathbb{N}$ be a power of 2. A *quantum random access gate* QRAG of memory size n is a QMD with $\mathsf{V}(i) = \mathsf{SWAP}_{\mathtt{M}_i \leftrightarrow \mathtt{T}}$, $\forall i \in [n]$. Equivalently, it is a QMD that maps

$$|i\rangle_{\mathtt{A}}|b\rangle_{\mathtt{T}}|x_0,\dots,x_{n-1}\rangle_{\mathtt{M}} \mapsto |i\rangle_{\mathtt{A}}|x_i\rangle_{\mathtt{T}}|x_0,\dots,x_{i-1},b,x_{i+1},\dots,x_{n-1}\rangle_{\mathtt{M}} \quad \forall i \in [n], b, x_0, \dots, x_{n-1} \in \{0,1\}. \tag{3.15}$$

It turns out that the QRAG can be simulated with a QRAM, but the QRAM can be simulated with the QRAG by requiring single qubit operations (which are not present in the model of computation of definition 3.6). We will present the proof for the simulation of the QRAM with a QRAG and leave the opposite proof as exercise.

Theorem 3.1 (Simulating QRAM with QRAG.)**.** *A query to a* QRAM *of memory size n can be simualted using 2 queries to a* QRAG *of memory size n, 3 two-qubit gates, and* 1 *workspace qubit.*

Proof. Start with the input $|i\rangle_{\mathtt{A}}|0\rangle_{\mathtt{Tmp}}|b\rangle_{\mathtt{T}}|x_0,\dots,x_{n-1}\rangle_{\mathtt{M}}$ by using an ancillary qubit `Tmp` for the workspace. Use the $\mathtt{SWAP}_{\mathtt{T}\leftrightarrow\mathtt{Tmp}}$ gate to obtain $|i\rangle_{\mathtt{A}}|b\rangle_{\mathtt{Tmp}}|0\rangle_{\mathtt{T}}|x_0,\dots,x_{n-1}\rangle_{\mathtt{M}}$. A query to the QRAG then leads to $|i\rangle_{\mathtt{A}}|b\rangle_{\mathtt{Tmp}}|x_i\rangle_{\mathtt{T}}|x_0,\dots,x_{n-1}\rangle_{\mathtt{M}}$. Use a $\mathtt{C}_{\mathtt{T}}\text{-}\mathtt{X}_{\rightarrow\mathtt{Tmp}}$ from register `T` to register `Tmp`, and query again the QRAG, followed by a $\mathtt{SWAP}_{\mathtt{T}\leftrightarrow\mathtt{Tmp}}$ gate, to obtain the desired state $|i\rangle_{\mathtt{A}}|b\oplus x_i\rangle_{\mathtt{T}}|x_0,\dots,x_{n-1}\rangle_{\mathtt{M}}$ after discarding the ancillary qubit. □

Exercise 3.3. Assuming that single-qubit gates can be freely applied onto the memory register `M` of any QRAM, then show that a QRAG of memory size n can be simulated using 3 queries to a QRAM of memory size n and $2(n+1)$ Hadamard gates.

TODO make teheorem of the collowing comment.

3.2.2.4 Memory compression in sparse QRAG models

Assuming that the data is sparse is a common assumption when developing quantum algorithms since it significantly simplifies computations. Applying it to compress quantum algorithms with access to a QRAG was first proposed by (Ambainis, 2007), elaborated further in (Jeffery, 2014), (Bernstein et al., 2013), and finally formalized in (Buhrman et al., 2022).

Informally, a quantum algorithm is considered sparse if the number of queries to a QRAG of size M are made with a small number of quantum states. More formally, we start by recalling that in a quantum computational model of definition 3.6 we split the qubits in several registers including a M qubit memory register and a W qubit working register. If throughout the computation the queries to the QRAG are made using only a constant set of quantum states which have a maximum Hamming weight(which is the number of 1's in the bit string) of $m \lll M$, then the algorithm is said to be m-sparse. The trick to making this definition work is realizing that the `SWAP` gate can be used to exchange states between the working register and the states with low hamming weight of the target register.

Confusingly, the authors of (Buhrman et al., 2022) decided to call a machine that works under this model as QRAM: quantum random-access machine. The formal definition of an m-sparse quantum algorithm with a QRAG is the following:

Definition 3.11 (Sparse QRAG algorithm (Buhrman et al., 2022))**.** Let $\mathcal{C} = (n, T, W, M, C_1, \dots, C_T)$ be a QRAG algorithm using time T, W work qubits, and M memory qubits. Then, we say that C is m-sparse, for some $m \leq M$, if at every time-step $t \in \{0, \dots, T\}$ of the algorithm, the state of the memory qubits is supported on computational basis vectors of Hamming weight $\leq m$. i.e., we always have

$$|\psi_t\rangle \in \text{span}\left(|u\rangle|v\rangle \;\middle|\; u \in \{0,1\}^W, v \in \binom{[M]}{\leq m} \right) \tag{3.16}$$

In other words, if $|\psi_t\rangle$ is written in the computational basis:

$$|\psi_t\rangle = \sum_{u\in\{0,1\}^W} \sum_{v\in\{0,1\}^M} \alpha_{u,v}^{(t)} \cdot \underbrace{|u\rangle}_{\text{Work qubits}} \otimes \underbrace{|v\rangle}_{\text{Memory qubits}} , \tag{3.17}$$

then $\alpha_{u,v}^{(t)} = 0$ whenever $|v| > m$, where $|v|$ is the Hamming weight of v.

Now that we have seen sparse QRAG algorithms, we can look at how memory compression is performed. In particular, any m-sparse quantum algorithm running in time T and utilizing M memory qubits can be simulated up to an additional error ϵ by a quantum algorithm running in time $O(T \log(\frac{T}{\epsilon}) \log(M))$ using $O(m \log(M))$ qubits.

Theorem 3.2 (Memory compression for m-sparse QRAG algorithms (Buhrman et al., 2022))**.** *Let T, W, $m < M = 2^\ell$ be natural numbers, with M and m both powers of 2, and let $\epsilon \in [0, 1/2)$. Suppose we are given an m-sparse* QRAG *algorithm using time T, W work qubits and M memory qubits, that computes a Boolean relation F with error ϵ.*

Then we can construct a QRAG *algorithm which computes F with error $\epsilon' > \epsilon$, and runs in time $O(T \cdot \log(\frac{T}{\epsilon' - \epsilon}) \cdot \gamma)$, using $W + O(\log M)$ work qubits and $O(m \log M)$ memory qubits.*

3.3 Implementations

In this section we'll be creating oracles that can perform the encodings that were presented in section 3.1. Of the presented oracles only 3 will make use of the quantum memory device introduced in definition 3.6: the bucket brigade, KP-trees and the block encoding from data structure. It is interesting to note that these oracles actually aid each other. In fact, KP-trees rely on the existence of a QMD that can perform binary encoding and similarly block encoding from data structures requires the existence of a QMD that can perform amplitude encoding. The other oracles will either make use of specific properties of the input data, such as sparsity, or will encode a probability distribution as a quantum state. The key insight lies in the fact that all oracles, with or without QMD, have a constant complexity which allows us to work in the quantum memory model of computation of definition 3.4. All the presented oracles with their interconnection can be seen in figure 3.2, where the oracle which require a QMD have been indicated with a *.

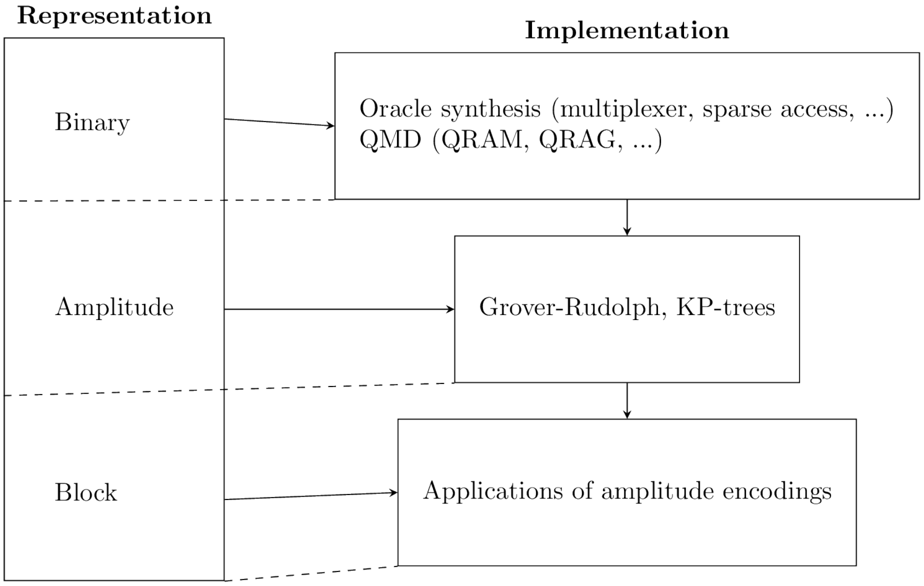


Figure 3.2: This figure shows the different types of data encoding techniques with the corresponding oracles. The vertical lines on the right hand side indicate (possible) dependencies between oracles.

3.3.1 Binary encoding

In this section we are discussing implementations a unitary giving query access to a list of m-bits values. A possible way of reading this section is throught the lenses of finding the "best" gate decomposition of that unitary, which has the following form:

$$U = \sum_{i=0}^{N-1} |i\rangle\langle i| \otimes U_i = \begin{bmatrix} U_0 & & & \\ & U_1 & & \\ & & \ddots & \\ & & & U_{N-1} \end{bmatrix}, \tag{3.18}$$

where $U_i|0\rangle = |x_i\rangle$ and $x_i \in \{0,1\}^m$. Importantly, when considering gate decompositions of this unitary, we are allowed to act in a larger space, as long as in the subspace of interested our application acts accoring to Eq.(3.18).

3.3.1.1 Oracle synthesis

As we briefly anticipated in Section 2.3, if we know a function that maps $x_i = f(i)$ we can create a circuit for getting query access to x_i. If our data is represented by the output of a function, we can consider these techniques for data loading.

The idea of creating a quantum circuit from a classical Boolean function is relatively simple and can be found in standard texts in quantum computation ((Nielsen and Chuang, 2002) or the this section on the Lecture notes of Dave Bacon). There is a simple theoretical trick that we can use to see that for any (potentially irreversible) Boolean circuit there is a reversible version for it. This observation is used to show that non-reversible circuits are *not* more powerful than reversible circuits. To recall, a reversible Boolean circuit is just bijection between domain and image of the function. Let $f : \{0,1\}^m \mapsto \{0,1\}^n$ be a Boolean function (which we assume is surjective, i.e. the range of f is the whole $\{0,1\}^n$). We can build a circuit $f' : \{0,1\}^{m+n} \mapsto \{0,1\}^{m+n}$ by adding some ancilla qubits, as it is a necessary condition for reversibility that the dimension of the domain matches the dimension of the range of the function. We define f' as the function performing the mapping $(x,y) \mapsto (x, y \oplus f(x))$. It is simple to see by applying twice f' that the function is reversible (check it!).

Now that we have shown that it is possible to obtain a reversible circuits from any classical circuit, we can ask: what is an (rather inefficient) way of getting a quantum circuit? Porting some code (or circuit) from two similar level of abstraction is often called *transpiling*. Again, this is quite straightforward (Section 1.4.1 (Nielsen and Chuang, 2002)). Every Boolean circuit can be rewritten in any set of universal gates, and as we know, the NAND port is universal for classical computation. It is simple to see (check the exercise) that we can use a Toffoli gate to simulate a NAND gate, so this gives us a way to obtain a quantum circuit out of a Boolean circuit made of NAND gates. With these two steps we described a way of obtaining a quantum circuit from any Boolean function f.

Exercise 3.4 (Toffoli as NAND)**.** Prove that a Toffoli gate, along with an ancilla qubit, can be used to obtain a quantum version of the NAND gate?

However, an application of the quantum circuit for f, will result in a garbage register of some unwanted qubits. To get rid of them we can use this trick:

$$|x\rangle|0\rangle|0\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle|k(f,x)\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle|k(f,x)\rangle|f(x)\rangle \mapsto |x\rangle|f(x)\rangle. \tag{3.19}$$

Let's explain what we did here. In the first step we apply the circuit that computes f'. In the second step we perform a controlled NOT operation (controlled on the third and targeting the fourth register), and in the last step we undo the application of f', thus obtaining the state $|x\rangle|f(x)\rangle$ with no garbage register.

Importantly, the idea of obtaining a quantum circuit from a classical reversible circuit is not practical, and is only relevant didactically. The task of obtaining an efficient quantum circuit from a Boolean function is called "oracle synthesis". Oracle synthesis is far from being a problem of only theoretical interest, and it has received a lot of attention in past years (Soeken et al., 2018) (Schmitt et al., 2021) (Shende et al., 2006). Today software implementations can be easily found online in most of the quantum programming languages/library. For this problem we can consider different scenarios, as we might have access to the function in form of reversible Boolean functions, non-reversible Boolean function, or the description of a classical circuit. The problem of oracle syntheses is a particular case of quantum circuit synthesis (Table 2.2 of (de Brugière, 2020)) and is a domain of active ongoing research.

If we want to prove the runtime of a quantum algorithm in terms of gate complexity (and not only number of queries to an oracle computing f) we need to keep track of the gate complexity of the quantum circuits we use. For this we can use the following theorem.

Theorem 3.3 ((Buhrman et al., 2001b) version from (Bausch et al., 2021)). *For a probabilistic classical circuit with runtime $T(n)$ and space requirement $S(n)$ on an input of length n there exists a quantum algorithm that runs in time $O(T(n)^{\log_2(3)}$ and requires $O(S(n)\log(T(n))$ qubits.*

3.3.1.2 Sparse access

Sparse matrices are very common in quantum computing and quantum physics, so it is important to formalize a quantum access for sparse matrices. This model is sometimes called in literature "sparse access" to a matrix, as sparsity is often the key to obtain an efficient circuit for encoding such structures without a QRAM. Of course, with a vector or a matrix stored in a QRAM, we can also have efficient (i.e. in time $O(\log(n))$ if the matrix is of size $n \times n$) query access to a matrix or a vector, even if they are not sparse. It is simple to see how we can generalize query access to a list or a vector to work with matrices by introducing another index register to the input of our oracle. For this reason, this sparse access is also called quite commonly "query access".

Definition 3.12 (Query access to a matrix). Let $V \in \mathbb{R}^{n\times d}$. There is a data structure to store V, (where each entry is stored with some finite bits of precision) such that, a quantum algorithm with access to the data structure can perform $|i\rangle|j\rangle|z\rangle \rightarrow |i\rangle|j\rangle|z \oplus v_{ij}\rangle$ for $i \in [n], j \in [d]$.

A matrix can be accessed also with another oracle.

Definition 3.13 (Oracle access in adjacency list model). Let $V \in \mathbb{R}^{n\times d}$, there is an oracle that allows to perform the mappings:

- $|i\rangle \mapsto |i\rangle|d(i)\rangle$ where $d(i)$ is the number of non-zero entries in row i, for $i \in [n]$, and

- $|i,l\rangle \mapsto |i,l,\nu(i,l)\rangle$, where $\nu(i,l)$ is the l-th nonzero entry of the i-th row of V, for $l \leq d(i)$.

The previous definition is also called *adjacency array* model. The emphasis is on the word *array*, contrary to the adjacency list model in classical algorithms (where we usually need to go through all the list of adjacency nodes for a given node, while here we can query the list as an array, and thus use superposition) (Dürr et al., 2004).

It is important to recall that for Definition 3.12 and 3.13 we could use a QRAM, but we also expect **not** to use a QRAM, as there might be other efficient circuit for performing those mapping. For instance, when working with graphs (remember that a generic weighted and directed graph $G = (V, E)$ can be seen as its adjacency matrix $A \in \mathbb{R}^{|E|\times|E|}$), many algorithms call Definition 3.12 **vertex-pair-query**, and the two mappings in Definition 3.13 as **degree query** and **neighbor query**. When we have access to both queries, we call that **quantum general graph model** (Hamoudi and Magniez, 2018). This is usually the case in all the literature for quantum algorithms for Hamiltonian simulation, graphs, or algorithms on sparse matrices.

3.3.1.3 Circuits

TODO

What if we want to use a quantum circuit to have quantum access to a vector of data? In this section we are going to describe few circuits for the task, each showcasing different properties and trade-offs between depth and space. For example the simplest circuit that we can come up with, has a depth that is linear in the length of the vector. This kind of circuit is often used in literature, e.g. for computing functions using space-time trade-offs (Krishnakumar et al., 2022; Gidney and Ekerå, 2021). This circuit (which sometimes goes under the name QROM, or circuit for table lookups (Hann et al., 2021), or multiplexer, is simply a series of multi-control operations, each of which is writing (using X gates) some binary string on the target register. Controlled on the index register being in the state $|0\rangle$, we write in the output register the value of our vector in position x_0, controlled in the index register being $|1\rangle$, we write on the output register the value of our vector in position x_1, etc.. This will result in a circuit with a depth that is linear in the length of the vector that we are accessing, however this circuit won't require any ancilla qubit. We will discuss more some hybrid architecture that allows a trade off between depth and ancilla qubits in Section **??**. The Toffoli count of this circuit can be improved in various ways (Babbush et al., 2018; Zhu et al., 2024). Importantly, the depth of this circuit depends on the number of oracle entries N, and in simple implementations depends also linearly in m. As an example, consider the following circuit.

The most general version of the circuit is the following, which includes some optimization.

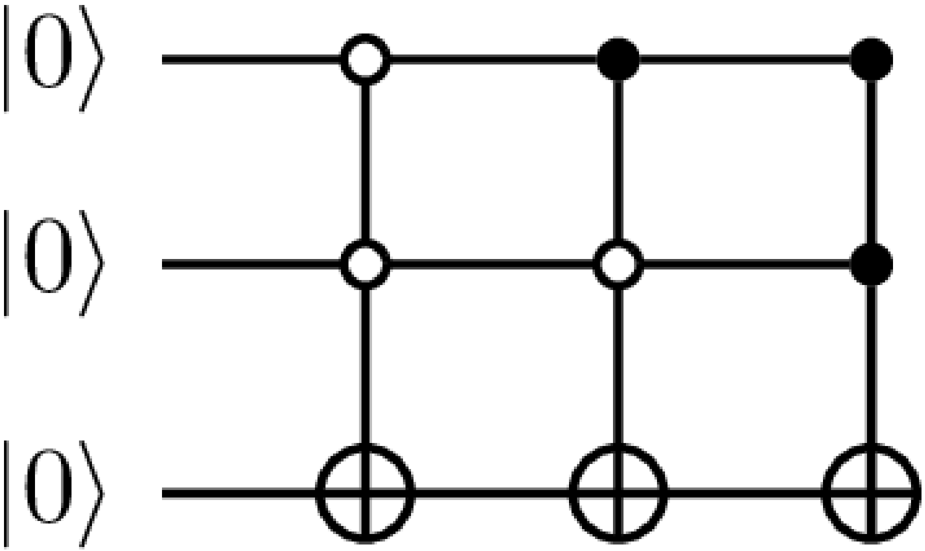


Figure 3.3: This is the example of a multiplexer circuit for the list of values x=[1,1,0,1]. Indeed, if we initialize the first two qubits with zeros, the output of the previous circuit will be a 1 in the third register, and so on.

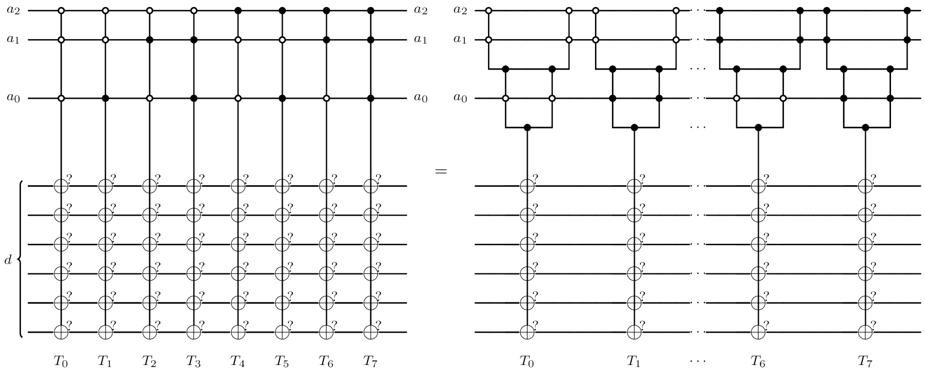

a_2
a_1
a_0
a_2
a_1
a_0
d
=
a_2
a_1
a_0
T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7
T_0 T_1 ⋯ T_6 T_7

Figure 3.4: The implementation of the multiplexer of (Babbush et al., 2018). Importantly, only one T gate is required to write in the target register a single memory entry. The memory entry is written using a Fan-Out gate (which can be decomposed into CNOT gates. Note that the depth of the circuit is linear in the number of memory entries.The angular lines in the gates are representing the so-called 'logical-AND', which is a CCNOT gate targeting an ancilla qubit starting in a particular state (ground state on which we apply a T gate) and gets uncomputed using measurament-based uncomputation.

The bucket brigade architecture (in short BB architecture) is another possible architecture. This protocol was originally developed to be implemented with qutrits (Giovannetti et al., 2008b) but recent work has shown it can be implemented with qubits as well (Hann et al., 2021). First, we will mention the idea with qutrits since it is more intuitive to understand. Again, using the terminology introduced in definition 3.6, we'll have: an address register, a target register (which will be referred to as a bus register) and a memory register. The input will be a vector of binary numbers $X \in (\{0,1\}^m)^n$ and the aim is to load a specific entry $X_i \in \{0,1\}^m$ on the bus register.

The BB protocol will use the memory register in such a way that it has access to tree like structure. The tree saves the entries of X in the leaves, which are referred to as memory cells. Each memory cell is connected to a parent node which form a set of intermediate nodes up until the root of the tree. Each intermediate node (up to the root) is called a quantum router (Figure 1b of (Hann et al., 2021)) and is a qutrit (i.e., a three level quantum system), which can be in state $|0\rangle$ (route left), $|1\rangle$ (route right), and $|W\rangle$ (wait).

When we want to perform a query, we prepare the address register with the index of the memory cell that we want to reach and we set all the router registers to the $|W\rangle$ state. Conditioned on the first qubit of the address register, the root of the tree changes from $|W\rangle$ to either $|0\rangle$(left) or $|1\rangle$(right). This is followed by a similar operation which uses as control the second qubit of the address register to change the state of the next node in the tree to either $|0\rangle$ or $|1\rangle$. The process of changing the state of the routers is repeated until the last layers of the tree(i.e. the memory cell) is reached. Now, the memory register will be in the state of the binary number X_i. This can be copied to the bus register by simply applying a series of CNOT gates (and thus we do not violate the no-cloning theorem).

Studying an error model of the BB architecture is hard. An attempt was first made in (Arunachalam et al., 2015) which gave initial, but rather pessimistic result. More recently, a series of developments in (Hann et al., 2021) and (Hann, 2021) (accessible here) have shone light on the noise resilience of the BB QRAM. The results presented in these more recent works are much more positive. Some resource estimations can be found in (Di Matteo et al., 2020), which do not take into account the new developements in the study of the error.

The metric of choice to test whether a quantum procedure has faithfully recreated a desired state is the fidelity F, with the infidelity defined as $1-F$. Given a addressable memory of size N(i.e. $\log N$ layers in the binary tree) and a bucket brigade which requires T time-steps with a probability of error per time step of ϵ, the infidelity of the bucket brigade scales as:

$$1 - F \approx \sum_{l=1}^{\log N} (2^{-l})\epsilon T 2^l = \epsilon T \log N, \tag{3.20}$$

Exercise 3.5. Calculate $\sum_{l=1}^{\log N} l$

The time required to perform a query, owing to the tree structure of the BB, is $T = O(\log N)$. This can be seen trivially from the fact that $T \approx \sum_{l=0}^{\log N-1} l = \frac{1}{2}(\log N)(\log N + 1)$, but can be decreased to $O(\log N)$ (Appendix A of (Hann et al., 2021)). This leaves us with the sought-after scaling of the infidelity of $\widetilde{O}(\epsilon)$ where we are hiding in the asymptotic notation the terms that are polylogarithmic in N. The error that happen with probability ϵ can be modeled with Kraus operators makes this error analysis general and realistic (Appendix C (Hann et al., 2021)), and is confirmed by simulations. For a proof of Equation (3.20) see Section 3 and Appendix D of (Hann et al., 2021).

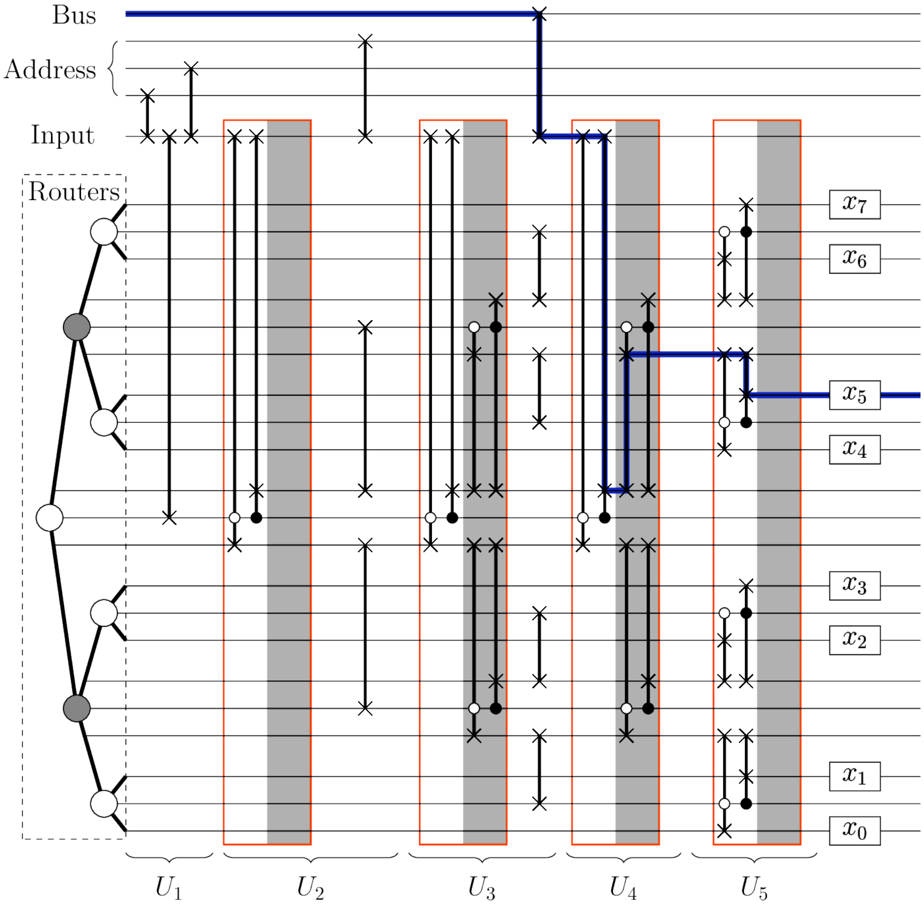


Figure 3.5: A possible implementation of the bucket-brigade QRAM in the circuit model, from (Hann et al., 2021)

Exercise 3.6. The last block of the circuit depicted in Figure 3.6 has linear depth in the number of memory elements, i.e. has exponential depth in the

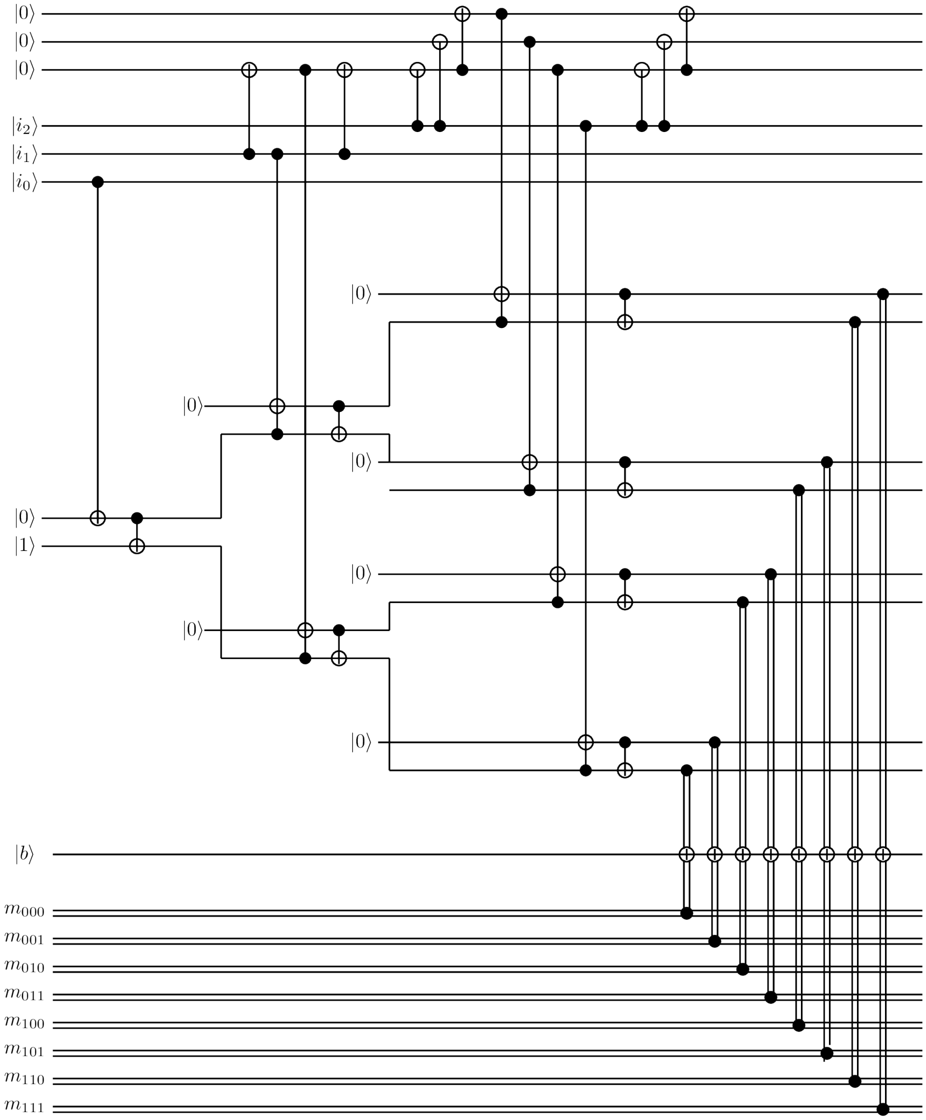


Figure 3.6: A possible implementation of the bucket-brigade QRAM in the circuit model, from (Doriguello et al., 2024; Arunachalam et al., 2015). In every layer, before the parallel layer of Toffoli gates, a log-depth linear-size gadget is copying the index register. In this way, the Toffoli gates can be executed in parallel.

number of qubits in the index regsiter. Eventually using some ancilla qubits, can you create a gadget that writes to the output register in logarithmic depth?

The following statement gives a resource estimation for a QRAM of logarithmic depth using the quantum architecture proposed in (Litinski and Nickerson, 2022).

Lemma 3.1 (Complexity of QRAM using (Doriguello et al., 2024))**.** *One bucket-brigade* QRAM *call of size* 2^n *and precision* κ *requires (already including its uncomputation)* $2^n - 2$ Toffoli *gates,* $2^{n+1} - n - 1$ *dirty ancillae (plus* $n + \kappa$ *input/output qubits), and has* Toffoli*-width of* 2^{n-1}*, reaction depth of* $2(n-1)$*, and active volume of* $(25 + 1.5\kappa + C_{|CCZ\rangle})2^n$*.*

In the following, we report the resource count of some QRAM constructions using different techniques.

Theorem 3.4 (Complexity of QRAM using Fourier analysis (Allcock et al., 2023))**.** *Let* $n \in \mathbb{N}$ *be a power of* 2*. A* QRAM *of memory size* n *can be implemented in* $O(1)$*-depth using*

- *either* $\frac{1}{2}n^2 \log n + O(n^2)$ *ancillae and* $2n^2 + O(n \log n)$ *Fan-Out gates with arity* $\leq 1 + n^2$*,*
- *or* $2n^2$ *ancillae and* 2 GT *gates with arity* $\leq \frac{1}{2}n^2 \log n + O(n^2)$*.*

Theorem 3.5 (Complexity of QRAM using one-hot encoding ideas (Allcock et al., 2023))**.** *For every* $n \in \mathbb{N}$ *a power of* 2*, a* QRAM *of memory size* n *can be implemented in* $O(1)$*-depth using*

- *either* $2n \log n \log \log n + O(n \log n)$ *ancillae and* $6n \log n + O(n \log \log n)$ *Fan-Out gates with arity* $\leq n + 1$*,*
- *or* $3n \log n + O(n \log \log n)$ *ancillae and* 6 GT *gates with arity* $\leq n \log n + O(n \log \log n)$*.*

Theorem 3.6 (Complexity of QRAM using recursive tricks (Allcock et al., 2023))**.** *For every* $n, d \in \mathbb{N}$*, a* QRAM *of memory size* n *can be performed in* $O(d)$*-depth~using*

- *either* $O(n \log^{(d)} n \log^{(d+1)} n)$ *ancillae and* $O(n \log^{(d)} n)$ *Fan-Out gates,*
- *or* $O(n \log^{(d)} n)$ *ancillae and* $16d - 10$ GT *gates.*

3.3.2 Amplitude encoding

We now move our attention to amplitude encoding, which was first introduced in section 3.1.2. In amplitude encoding, we encode a vector of numbers in the amplitude of a quantum state. Implementing a quantum circuit for amplitude encoding can be seen as preparing a specific quantum states, for which we know the amplitudes. In other words, this is actually a *state preparation problem* in disguise, and we can use standard state preparation methods to perform amplitude encoding. However, note that amplitude encoding is a specific example of state preparation, when the amplitudes of the state are known classically or via an oracle. There are other state preparation problems that are not amplitude encoding, like ground state preparation, where the amplitudes of the quantum state is not known and only the Hamiltonian of the system is given. In the following, we briefly discuss the main techniques developed in the past decades for amplitude encoding.

What are the lower bounds for the size and depth complexity of circuits performing amplitude encoding? Since amplitude encoding can be seen as a quantum state preparation, without assuming any kind of oracle access, we have a lower bound of $\Omega\left(2^n\right)$ (Plesch and Brukner, 2011; Shende et al., 2004). For the depth, we have a long history of results. For example, there is a lower bound of $\Omega(\log n)$ that holds for some states (and hence puts a lower bound on algorithms performing generic state preparation) using techniques from algebraic topology (Aharonov and Touati, 2018). Without ancilla qubits (Plesch and Brukner, 2011) proposed a bound of $\Omega(\frac{2^n}{n})$. The bound on the depth has been refined to a $\Omega(n)$, but only when having arbitrarily many ancilla qubits (Zhang et al., 2021). The more accurate bound is of $\Omega\left(\max\{n, \frac{4^n}{n+m}\}\right)$ (Theorem 3 of (Sun et al., 2023)), where m is the number of ancilla qubits. The algorithms of (Yuan and Zhang, 2023), which we discuss later, saturates this bound.

We can also study the complexity of the problem in the oracle model. For example, if we assume an oracle access to $f : \{0,1\}^n \mapsto [0,1]$, using amplitude amplification techniques on the state $\sum_x |x\rangle \left(f(x)|0\rangle + \sqrt{1-f(x)}|1\rangle\right)$, there is a quadratic improvement in the number of queries to the oracle, yielding $\widetilde{O}(\sqrt{N})$ complexity (Grover, 2000), where $N = 2^n$. This can be seen if we imagine a vector with only one entry with the value 1, where the number of queries to amplify the subspace associated with the rightmost qubit scales with $\sqrt{N}$. Few years later, we find another work by (Grover and Rudolph, 2002) which, under some mildly stronger assumptions improved the complexity of the algorithms for a very broad class of states. This algorithm is better discussed in Section 3.3.2.1.

Alternatively, we can assume a direct oracle access to the amplitudes (Sanders et al., 2019). Under this assumption, we have access to an oracle storing the ith amplitude α_i with n bits, (actually, they use a slightly different model, where the oracle for the amplitude α_i is $|i\rangle|z\rangle \mapsto |i\rangle|z \oplus \alpha_i^{(n)}\rangle$ where $\alpha_i^{(n)} = \lfloor 2^n \alpha_i \rfloor$). Ordinarily, the circuit involves the mapping $|i\rangle|\alpha_i^{(n)}\rangle|0\rangle \mapsto |i\rangle|\alpha_i\rangle\left(\sin(\theta_i)|0\rangle + \cos(\theta_i)|1\rangle\right)$, which requires control rotations and arithmetic circuits to compute the angles $\theta_i = \arcsin(\alpha_i/2^n)$. However, by substituting the arithmetic circuit by a comparator operator (Gidney, 2018; Cuccaro et al., 2004; Luongo et al., 2024), the circuit can be implemented either with $2n$ non-Clifford gates or n non-Clifford gates and n ancilla qubits. This scheme can even be extended to encode complex amplitudes in both Cartesian and polar forms, or apply to the root coefficient problem of real amplitudes, where we have an oracle access to the square of the amplitude α_i^2 instead of α_i. For positive or complex amplitudes, this algorithm involves

$\frac{\pi}{4}\frac{\sqrt{N}}{\|\alpha\|_2}$ exact amplitude amplifications, so it has a runtime of $\frac{\pi}{4}\frac{t\sqrt{N}}{\|\alpha\|_2} + O(1)$ non-Clifford gates, where t is the number of bits of precision used to specify an amplitude (the authors preferred to count the number of non-Clifford gates, as they are the most expesive one to implement in (most of) the error corrected architectures, and serves as a lower bound for the size complexity of a circuit. For the root coefficient problem, the runtime becomes $\frac{\pi}{4}\frac{n\sqrt{N}}{\|\alpha\|_1} + O\left(n\log\left(\frac{1}{\epsilon}\right)\right)$ non-Clifford gates. For certain sets of coefficients, this model can further be improved to reduce the number of ancilla qubits needed per bits of precision from a linear dependence (Sanders et al., 2019) to a log dependence (Table 2 of (Bausch, 2022)). Also the work of (McArdle et al., 2022) works assuming an oracle returning the amplitudes of the state we want to build $\frac{1}{\mathcal{N}_f}\sum_{x=0}^{N-1} f(x)|x\rangle$ (where $\mathcal{N}_f$ is the usual normalization factor), and does not use arithmetic, and uses $O(\frac{nd_\epsilon}{\mathcal{F}_{\widetilde{f}^{[N]}}})$ (where $\widetilde{f}^{[N]}$ is called "discretized ℓ_2-norm filling-fraction", and d_ϵ is the degree of a polynomial approximation that depends on ϵ, the approximation error in the quantum state) and uses only 4 ancilla qubits.

Instead of treating the problem as a state preparation problem, we can also perform amplitude encoding using multivariable quantum signal processing (M-QSP) (See Chapter **??**). The principle behind this method is to interpret the amplitude of a quantum state as a function of a multivariable polynomial (McArdle et al., 2022; Mori et al., 2024; Rosenkranz et al., 2024). Particularly, using Linear Combination of Unitaries techniques, we can approximate the quantum state as a multivariable function by a truncated Fourier or Chebyshev series (Rosenkranz et al., 2024). The truncated Fourier series approximation requires $O(d^D + Dn\log d)$ two-qubit gates, while the truncated Chebyshev series approximation requires $O(d^D + Ddn\log n)$ two-qubit gates, where D is the number of dimensions and d is the degree of the polynomial used in the approximation. The number of qubits in both techniques scales as $O(Dn + D\log d)$. The exponential dependence in D can further be improved by the following theorems.

Theorem 3.7 (Bivariate state preparation (Mori et al., 2024))**.** *Given a Fourier series f of degree (d_1, d_2) that can be constructed with M-QSP, we can prepare a quantum state $|\psi_f\rangle$ using $O((n_1 d_1 + n_2 d_2)/\mathcal{F}_f)$ gates, where $\mathcal{F}_f = \mathcal{N}_f/(\sqrt{N_1 N_2}|f|_{max})$, $\mathcal{N}_f = \sqrt{\sum_{i,j}(f(x_1^{(i)}), f(x_2^{(j)}))}$, whereas n_1 and n_2 are the number of bits used to specify the value of the variable x_1 and x_2, respectively.*

Theorem 3.8 (Multivariate state preparation (Mori et al., 2024))**.** *Given a Fourier series f of degree $(d_1, \dots, d_D)$ that can be constructed with multivariate quantum signal processing, we can prepare a quantum state $|\psi_f\rangle$ using $O(ndD/\mathcal{F}_f)$ gates, where $n = \max(n_1, \dots, n_D)$ and $d = \max(d_1, \dots, d_D)$*

Meanwhile, if trade-offs are allowed for state preparation, we can further improve the complexity of the algorithm. For example, we can build a state over n qubits with depth $\widetilde{O}\left(\frac{2^n}{m+n} + n\right)$ and size $\widetilde{O}\left(2^n\right)$ if we have m available ancillas (Sun et al., 2023). On the other hand, we can reduce the number of T-gates to $O(\frac{N}{\lambda} + \lambda \log \frac{N}{\epsilon} \log \frac{\log N}{\epsilon})$ if we allow a tunable number of $\lambda \frac{\log N}{\epsilon}$ dirty qubits (Low et al., 2018). A dirty qubit is an auxiliary qubit that is left entangled with another register at the end of computation. It cannot be reused by subsequent computations without being disentangled, but can be uncomputed using measurement-based uncomputation tricks.

In addition to the algorithms (Sun et al., 2023; Rosenthal, 2021), trade-offs can introduce additional circuits that can achieve the lower bound of the depth complexity. For example, using $O\left(2^n\right)$ ancilla qubits, we can perform amplitude encoding with circuit depth $\Theta\left(n\right)$, which further relaxes the connectivity requirements for M-QSP (Zhang et al., 2022). This technique also improves upon sparse state preparation, with a circuit depth $\Theta\left(\log kN\right)$, where k is the sparsity. This represents an exponential improvement in circuit depth over previous works (Gleinig and Hoefler, 2021; de Veras et al., 2022). This leads to a deterministic algorithm that achieves the lower bounds in circuit depth if we allow m ancilla qubits, which is summarized in the following theorem.

Theorem 3.9 (Circuit for controlled state preparation (Theorem 1 of (Yuan and Zhang, 2023)))**.** *For any $k \in \mathbb{N}$ and quantum states $\{|\psi_i\rangle | i \in \{0,1\}^k\}$ there is a circuit performing*

$$|i\rangle|0\rangle \mapsto |i\rangle|\psi_i\rangle, \forall i \in \{0,1\}^k,$$

which can be implemented by a circuit of depth $O(n + k + \frac{2^{n+k}}{n+k+m})$ and size $O(2^{n+k})$ with m ancillary qubits. These bounds are optimal for any $m, k \geq 0$.

Theorem 3.10 (Circuit for state preparation (Theorem 2 of (Yuan and Zhang, 2023)))**.** *For any $m > 0$, any n-qubit quantum state $|\psi_v\rangle$ can be generated by a quantum circuit using single qubit gates and CNOT gates, of depth $O(n + \frac{2^n}{n+m})$ and size $O(2^n)$ with m ancillary qubits. These bounds are optimal for any $m \geq 0$.*

There are also other trade-off techniques that can be used, like probabilistic state preparation via measurements (Zhang et al., 2021) or approximate state preparation problem (Zhang et al., 2024). However, these techniques are beyond the scope of this chapter and will not be discussed. Interested readers can refer to the respective articles. However, even if we allow intermediate measurements we have lower bounds on the T-count, which is reached by the algorithm of (Gosset et al., 2024).

Theorem 3.11 (Quantum state preparation with optimal T-count (Gosset et al., 2024))**.** *Any n-qubit state can be prepared up to error ϵ by a Clifford+T circuit starting with the all-zeroes state using*

$$O\left(\sqrt{2^n \log(1/\epsilon)} + \log(1/\epsilon)\right)$$

T gates and ancillas. Furthermore, no Clifford+T circuit (even with measurement and adaptivity) can use asymptotically fewer T-gates.

In summary, there are many methods to perform amplitude encoding, each with different complexities through various trade-offs. In general, the data set that can be encoded using amplitude encoding lies under two main categories: (i) those discrete data that come from a vector or a matrix, or (ii) those that come from a discretized probability distribution. In literature, amplitude encoding of vectors or matrices is called state preparation via a KP-tree, while amplitude encoding of discretized probability distribution is called Grover-Rudolph (GR) state preparation (Grover and Rudolph, 2002). The main difference between the KP-tree method and the GR state preparation is that the KP-tree method requires a quantum memory to store some precomputed values in a data structure or a tree, while the GR state preparation does not require a quantum memory. In fact, GR state preparation is designed such that there are efficient circuits to implement the oracle in the quantum computer using 3.3.1.1.

Table 3.2: Recap for the different methods proposed to implement amplitude encoding, together with the their gate count and ancilla complexity, along with the function type needed. This table is adapted from (Mori et al., 2024), 1 (Mori et al., 2024), 2 (Grover and Rudolph, 2002), 3 (Rattew and Koczor, 2022) 4 (Sanders et al., 2019; Bausch, 2022) 5 (Moosa et al., 2023) 6 (Rosenkranz et al., 2024) 7 (Shende et al., 2006) 8 (Sun et al., 2023) 9 (Gosset et al., 2024). Gate count or depth with * is expressed in number of T gates.

	Method	Size	Ancilla	Depth	Function type
1	M-SQP	$O(\frac{ndD}{\mathcal{F}})$	1		
2	GR	$O(nT_{oracle})$	$O(t_{oracle})$	log	Eff. int.
3	Adiabatic	$O(\frac{T_{oracle}}{\mathcal{F}^4})$	$O(t_{oracle})$	-	Arb.
4	Black-box	$O(\frac{T_{oracle}}{\mathcal{F}})$	$O(t_{oracle})$	-	Arb.
5	FSL	$O(d^D + Dn^2)$	0	-	Arb.
6	LCU-based	$O(d^D + Dn\log d)$	$O(D\log d)$	-	Arb.
7	Circuit	$N\log\left(\frac{N}{\epsilon}\right)$*	$O(n)$	$N\log\left(\frac{N}{\epsilon}\right)$*	Arb.
8	Circuit	$N\log\left(\frac{N}{\epsilon}\right)$	$O\left(n+\lambda\right)$	$\frac{N}{n+\lambda}\log\left(\frac{N}{\epsilon}\right)+ n\log\left(\frac{N}{\epsilon}\right)$ *	Arb.
9	Circuit	$O\left(\sqrt{2^n\log(\frac{1}{\epsilon})}+\log(\frac{1}{\epsilon})\right)$*	$O\left(\sqrt{2^n\log(\frac{1}{\epsilon})}+\log(1/\epsilon)\right)$		Arb.

By looking at different models of quantum computation, we find that state preparation can be performed in constant depth assuming unbounded Fan-Out circuits (Corollary 4.2) (Rosenthal, 2021). The key idea behind this work was to link state preparation to a DNF (disjunctive normal form) boolean formula, which is evaluated in the quantum algorithm. Compiling Fan-Out gates using CNOT gates, this leads to a circuit of depth $O(n)$ and $O(n2^n)$ ancillas. State preparation can also be studied under a different lens in complexity theory, leading to new interesting insights. For example, people studied the complexity of generating quantum states (Rosenthal and Yuen, 2021; Metger and Yuen, 2023). For example, the states that can be generated by a (space-uniform) polynomial-sized quantum circuit, forms the class of StatePSPACE. This class has been proven to be equivalent to StateQIP (the class of states that a polynomial-time quantum verifier can generate with interactions with a all-powerful and untrusted quantum prover), echoing the equivalence between the complexity classes QIP and PSPACE.

There are many other works in state preparation, and we refer the interested reader to (Bergholm et al., 2005; Plesch and Brukner, 2011; Araujo et al., 2021; Bausch, 2022; Rattew and Koczor, 2022; Plesch and Brukner, 2011; Rosenthal, 2021; Zhang et al., 2022; Buhrman et al., 2023b; Rosenthal, 2024; Gleinig and Hoefler, 2021; Holmes and Matsuura, 2020; Moosa et al., 2023; Zhao et al., 2021). Now we consider two very didactic and general models of quantum state preparation. The former is known as Grover-Rudolph state preparation (Grover and Rudolph, 2002) whilst the latter is known as a state prepration via a precomputed data structure that is quantum accessible, called KP-tree. A difference between the Grover-Rudolph and the KP-tree method is that GR is assuming a query access to an oracle which does not need to be necessarily needs to be implemented with a quantum memory. While the KP-tree method assumes some precomputation of a data structure (a tree) which is specifically stored into the QRAM. In fact, for the kinds of quantum states that GR was designed to create, there are efficient circuits for the implementing the oracle, that can be implemented in a quantum computer using Section3.3.1.1. For both, the total depth of the circuit (considering the QMD as part of the quantum computer) is $O(\log^2(N))$, while the size of the circuit is $O(N \log N)$.

Finally we note that in (Prakash, 2014) (Section 2.2.1), Prakash shows subroutines for generating $|x\rangle$ for a sparse x in time $O(\sqrt{\|x\|_0})$.

In this model is it possible to prepare states with long-range entanglement with only constant quantum-depth and logarithmic classical depth. However, the authors do not generalize their algorithm for state preparation for any quantum state, but only for a restricted class of quantum states with constraints on the Hamming weights.

3.3.2.1 Grover-Rudolph state preparation, its problems, and the solutions

In (Grover and Rudolph, 2002) the authors discussed how to efficiently create quantum states proportional to functions satisfying certain integrability condition, i.e. the function considered must be square-integrable. An example of functions with this properties are log-concave probability distributions. Let $p(x)$ be a probability distribution over $\mathbb{R}$. We denote by x_i^n is the points of the discretization over the domain, i.e $x_i^{(n)} = -w + 2w\frac{i}{2^n}$ for $i = 0, \ldots, 2^n$, and $[-w, w]$ is the window of discretization, for a constant $w \in \mathbb{R}_+$. In this case, n acts as the parameter that controls how coarse or fine is the discretization.

Consider referencing the appendix for more informations about measure theory and probability distributions. We want to create the quantum state

$$|\psi_n\rangle = \sum_{i=0}^{2^n-1} \sqrt{p_i^{(n)}}|i\rangle \tag{3.21}$$

with

$$p_i^{(n)} = \int_{x_i^{(n)}}^{x_{i+1}^{(n)}} p(x)\mathrm{d}x. \tag{3.22}$$

Actually, the probabilities $p_i^{(n)}$ will be normalized by $\int_{-w}^{w} p(x)\mathrm{d}x$. This is equivalent to discretizing the sample space Ω in $N = 2^n$ intervals with $N+1$ points, so that we can identify the samples ω of our discretized random variable with the elements of the set $[N]$. To create the state $|\psi_n\rangle$ we proceed recursiveliy in n, starting from initial state $|0\rangle$. To go from $|\psi_m\rangle = \sum_{i=0}^{2^m-1}\sqrt{p_i^{(m)}}|i\rangle$ to $|\psi_{m+1}\rangle = \sum_{i=0}^{2^{m+1}-1}\sqrt{p_i^{(m+1)}}|i\rangle$ we proceed by performing a query to an oracle that gives us an angle θ_i, for $i \in [2^m]$, which is used to perform the following rotation:

$$|i\rangle|\theta_i\rangle|0\rangle \mapsto |i\rangle|\theta_i\rangle\left(\cos\theta_i|0\rangle + \sin\theta_i|1\rangle\right), \tag{3.23}$$

In this case, the value θ_i is defined as $\arccos\sqrt{f(i)}$, where the function $f : [2^m] \mapsto [0,1]$ is defined as:

$$f(i) = \frac{\int_{x_L^i}^{\frac{x_L^i + x_R^i}{2}} p(x)dx}{\int_{x_L^i}^{x_R^i} p(x)dx}, \tag{3.24}$$

The value of $f(i)$ is the probability that the i-th sample x^i (which lies in the interval $[x_L^i, x_R^i]$) lines in the leftmost part of this interval (i.e. $[x_L^i, x_R^i + x_L^i/2]$). After the rotation, we undo the mapping that gives us the θ_i. These operations resulted in the following state:

$$\sum_{i=0}^{2^m-1}\sqrt{p_i^{(m)}}|i\rangle\left(\cos\theta_i|0\rangle + \sin\theta_i|1\rangle\right) = |\psi_{m+1}\rangle, \tag{3.25}$$

Computing the mapping for the angles θ_i can be done efficiently only for square-integrable probability distributions, i.e. for probability distribution for which the integral in Equation (3.23) can be approximated efficiently. Fortunately, this is the case for most of the probability distribution that we care about.

Creating quantum sample access to a probability distribution is a task often used to obtain quadratic speedups. A recent work (Herbert, 2021) pointed out that in certain cases, the time needed to prepare the oracle used to create $|\psi\rangle$ might cancel the benefits of the speedup. This is the case when we don't have an analytical formulation for integrals of the form $\int_a^b p(x)dx$, and we need to resort to numerical methods.

Often in quantum algorithms we want to estimate expected values of integrals of the form $\mathbb{E}[x] := \int_x xp(x)dx$ (e.g. see Chapter 8), Following a garbage-in-garbage-out argument, (Herbert, 2021) was able to show that if we require a precision ϵ in $\mathbb{E}[x]$, we also need to require the same kind of precision for the state preparation of our quantum computer. In particular, in our quantum Monte Carlo algorithms we have to create a state $|\psi\rangle$ encoding a (discretized) version of $p(x)$ as $|\psi\rangle = \sum_{i=0}^{2^n-1} \sqrt{p(i)}|i\rangle$.

Let's define μ as the mean of a probability distribution $p(x)$ and $\widehat{\mu} = \mathbb{E}(\mathbb{x})$ be an estimate of μ. The error of choice for this kind of problem (which comes from applications that we will see in Section 8) is called the Root Mean Square Error (RMSE), i.e. $\widehat{\epsilon} = \sqrt{\mathbb{E}((\widehat{\mu} - \mu)^2)}$. The proof shows that an error of ϵ in the first rotation of the GR algorithm, due to an error in the computation of the first $f(i)$, would propagate in the final error of the expected value of μ. To avoid this error, we should compute $f(i)$ with accuracy at least ϵ. The best classical algorithms allows us to perform this step at a cost of $O(\frac{1}{\epsilon^2})$, thus canceling the benefits of a quadratic speedup. Mitigating this problem is currently active area of research.

However, if we restrict ourselves to considering loading probabilities from a Gaussian distributions, then we can retain the quadratic speedup of the GR algorithm. This is because when we create quantum sample access to the Gaussian distribution, we must compute integrals of the form

$$I_{i,m}\left(\sigma\right) = \int_{x_i^{(m)}}^{x_{i+1}^{(m)}} \frac{1}{\sigma\sqrt{\pi}} e^{-x^2/\sigma^2} \mathrm{d}x = \int_{x_i^{(m)}/\sigma}^{x_{i+1}^{(m)}/\sigma} \frac{1}{\sqrt{\pi}} e^{-x^2} \mathrm{d}x \,,$$

where the second equality is obtained through the substitution $x \mapsto \frac{x}{\sigma}$ in the integral, $m = 1, \ldots, n$ determines the size of the interval partition $\frac{1}{2^m}$, $i = 0, \ldots, 2^m$ indexes the interval points, σ is the standard deviation of the Gaussian distribution, and w determines the end point of the integration, which is chosen such that the interval points $x_i^{(m)} = w\sigma\left(\frac{i}{2^{m-1}} - 1\right)$ is linear in σ. By the choice of the interval points, $I_{i,m}\left(\sigma\right) = I_{i,m}\left(1\right)$. Therefore, there is only one set of integrals to be evaluated for all values of σ, and we can store the integrals classically to high precision. This iterative construction is thus effective, retaining the quadratic speedup benefits of the GR algorithm.

3.3.2.2 KP-Trees

We now move to discussing the most used technique to load classical data into a quantum computer using ampitude encoding. This technique is the same as the one in the previous section, with the difference that the oracle returning the angles for the controlled rotations are retrived by a quantum memory. In their PhD thesis, Prakash (Prakash, 2014) introduced the first procedure to efficiently perform the amplitude encoding of a matrix using a tree-like classical data structure. Given a matrix $V \in \mathbb{R}^{n\times d}$ the procedure precomputes a data structure with size scaling $\widetilde{O}(|V|_0)$ and where the time to update the tree with a new entry scales as $O(\text{poly}\log(nd))$. This technique was called "KP-trees" in (Rebentrost and Lloyd, 2018), after the authors (Kerenidis and Prakash) used it in a quantum recommendation system (Kerenidis and Prakash, 2017). The proof will make use of the following lemma

Lemma 3.2 (Implementing Rotations with Cascades of Controlled Unitary Gates)**.** *Given: a register* $\mathtt{A}$ *composed of* t *qubits with the binary encoding of the fixed-point representation of a number* $\theta \in (0, 2\pi]$*, a target qubit* $\mathtt{b}$*, and a single qubit rotation parameterized by a single angle* $\mathtt{R}(\theta) \in \mathbb{C}^{2\times 2}$*, then the unitary*

$$\mathtt{C}_{\mathtt{A}}\mathtt{R}_{\mapsto \mathtt{b}}(\theta) = \prod_{i=0}^{t-1} \mathtt{C}_i\mathtt{R}_{\mapsto \mathtt{b}}\left(2^{\lfloor \log_2(\theta)\rfloor - i}\right)$$

is equivalent to applying $\mathtt{R}(\theta)$ *on the target qubit if the rotation holds the property* $R(\theta_1+\theta_2) = R(\theta_1)R(\theta_2)$*.*

Proof. For a number $\theta \in (0, 2\pi]$ on t qubits, the fixed point representation will be of the form $\mathcal{Q}(a, b, 0) := \mathcal{Q}(z, 0)$, with $c_1 = \lfloor \log_2(\theta)\rfloor + 1$ and $c_2 = t - c_1$. \ The application of $\mathtt{C}_{\mathtt{A}}\mathtt{R}_{\mapsto \mathtt{b}}(\theta)$ is equivalent to t single qubit application of $\mathtt{R}(\theta)$ on the target qubit $\mathtt{b}$, with the identity on all other qubits, where the angles have to be adjusted. %with parameters $z_i 2^{\lfloor \log(\theta)\rfloor - i}$. In particular:

$$\prod_{i=0}^{t-1} \mathtt{C}_i\mathtt{R}_{\mapsto \mathtt{b}}\left(2^{\lfloor \log_2(\theta)\rfloor - i}\right) = \prod_{i=0}^{t-1} \mathtt{R}_{\mapsto \mathtt{b}}\left(z_i 2^{\lfloor \log_2(\theta)\rfloor - i}\right)$$

where $z_i \in \{0, 1\}$ is the state of qubit i and we have omitted the tensor product with the identity on all other qubits of register $\mathtt{A}$ for simplicity. Then:

$$\prod_{i=0}^{t-1} \mathtt{R}_{\mapsto \mathtt{b}}\left(z_i 2^{\lfloor \log_2(\theta)\rfloor - i}\right) = R_{\mapsto \mathtt{b}}\left(\sum_{i=0}^{t-1} z_i 2^{\lfloor \log_2(\theta)\rfloor - i}\right) = R_{\mapsto \mathtt{b}}(\theta),$$

where we have used that $R(\phi_1+\phi_2) = R(\phi_1)R(\phi_2)$ and observing that $\sum_{i=0}^{t-1} z_i 2^{\lfloor \log(\theta)\rfloor - i}$ is the fixed-point encoding of θ.\ Since the proof only makes use of the fact that the parameterized gate needs to have the property $R_n(\phi_1+\phi_2) = R_n(\phi_1)R_n(\phi_2)$, we can easily extend it to the Phase gate P and the y rotation gate R_y. □

Exercise 3.7. Prove that $\sigma_y(\theta + \phi) = \sigma_y(\theta)\sigma_y(\phi)$ for the σ_y rotation given by

$$\sigma_y(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} \tag{3.26}$$

Prove the same property for the phase gate $P(\theta + \phi) = P(\theta)P(\phi)$ defined by

$$P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{3.27}$$

The original work developed a procedure to perform the amplitude encoding of a matrix $V \in \mathbb{R}^{n\times d}$ with KP-trees in 2 steps: a pre-processing step and circuit implementation. In the data pre-processing step, for each row $v_i \in \mathbb{R}^n$, which is composed of elements $v_{ij} \in \mathbb{R}$, a binary tree is created where the leaves contain the values v_{ij}^2 and the sign of v_{ij}. Each intermediate node contains the sum of the leaves of the sub tree rooted at that node. The circuit implementation makes use of the QMD to access the next layer of the desired tree v_i until the leaves are reach.

Here we'll present an optimized version of KP-trees for state preparation of complex matrices. In this version each tree is pruned such that it only saves the angles that are required for the state preparation rather than the partial norms, which halves the size of the memory. Furthermore, we clearly lay out the circuits for the implementation of the state preparation, which can be seen in 3.7. In addition the complex matrices are handled by storing the phase of each complex number. The circuit implementation of the phase makes use of a circuit as depicted in figures 3.8.

Theorem 3.12 (Optimized KP-trees for complex vectors)**.** *Let $V \in \mathbb{C}^{n\times d}$, where we assume that n and d are powers of 2, then there is a data structure to store the rows of V such that:*

- *The size of the data structure is $O(\|V\|_0 \log^2(nd))$.*
- *The time to insert, update or delete a single entry v_{ij} is $O(\log^2(nd))$.*
- *A quantum algorithm that has quantum access to the data structure can perform the mapping $\tilde{U} : |i\rangle|0\rangle \to |i\rangle|V_i\rangle$ for $i \in [m]$, corresponding to the rows of the matrix currently stored in memory and the mapping $\tilde{V} : |0\rangle|j\rangle \to |\tilde{V}\rangle|j\rangle$, for $j \in [n]$, where $\tilde{V} \in \mathbb{R}^m$ has entries $\tilde{V}_i = ||V_i||$, in time $(2+t)\log(nd) + 2t$, where t is the precision of the result.*

Proof. The data structure is composed of $n+1$ binary trees, one for each row of the matrix, and an additional one for the vector of norms of the rows. For the i^{th} row of V(i.e. V_i) we build the binary tree B_i. Given that that d is a power of 2, there will be $\log(d)$ layers to a tree B_i.

For an incoming element j of the vector V_i of the form $v_{ij}e^{i\theta_{ij}}$, the j^{th} leaf of the i^{th} tree will store the tuple (v_{ij}^2, θ_{ij}). Once a leaf is updated, the internal nodes will also be updated such that an internal node l will store the sum of the moduli v_{ij} of the leaves of a sub tree rooted at l. This procedure continues until all the matrix is loaded.

At the end of the procedure, we represent an internal node l in the i^{th} tree at depth d as B^l_{id}. If j_b represents the b^{th} bit of j then:

$$B^l_{id} = \sum_{\substack{j_1,\ldots,j_d=l \\ j_{d+1},\ldots,j_{\log(n)}\in\{0,1\}}} v^2_{ij}, \tag{3.28}$$

This implies that the first d bits of j written in binary are fixed to l, indicating that we are at depth d. This procedure requires $O(\log^2(nd))$ time.

After the tree has been filled it will be pruned. The pruning involves changing the values in the internal nodes such that they store the angle required to perform the state preparation. This will allow to remove v^2_{ij} from the leaves which will now only store the phases θ_{ij}.

Given a node B^l_{id}, the pruned tree will have nodes given by:

$$B'^l_{i,d} = \begin{cases} 2\arccos\left(\sqrt{\frac{B^{2l}_{i,d+1}}{B^l_{i,d}}}\right), & \text{if } d < \log(n) \\ \theta^l_{i,\log(n)}, & \text{if } d = \log(n) \end{cases}, \tag{3.29}$$

The size of the data structure scales as $O(|V|_0 \log^2(nd))$.

Since the first $\log(n) - 1$ layers are concerned with preparing the real amplitudes, $B'^l_{i,k} \in [0, \pi]$ for $k \in [\log(n) - 1]$, whilst the last layer will be bounded by $B'^l_{i,\log(n)} \in [0, 2\pi]$. Each angle in the tree will be stored in its fixed-point representation. Since the first $\log(n) - 1$ layers are concerned with preparing the real amplitudes, $B'^l_{i,k} \in [0, \pi]$ for $k \in [\log(n) - 1]$ meaning their fixed-point representation $\mathcal{Q}(z)$ will have $c_1 = 1$ and $c_2 = t$ for some precision t. The last layer contains the phases which are bounded by $B'^l_{i,\log(n)} \in [0, 2\pi]$, meaning their fixed-point representation $\mathcal{Q}(z)$ will have $c_1 = 2$ and $c_2 = t - 1$ for some precision t . This means that for some precision t the address register will require $t' = t + 1$ qubits.

The circuit will require quantum access to the data structure $B'_{i,d}$ as well as 3 registers: an index register $|i\rangle$ which contains $\lceil\log(i)\rceil$ qubits and the binary representation of i; a t'-qubit address(angle) register which will load the fixed-point representation of the angles up to a precision t; and the main register composed of $\log(n)$ qubits.

A quantum query to the data structure will load from B'_i the fixed-point representation of the angle on the address register which will be followed by a cascade of controlled rotations, as seen in lemma **??**, in order to produce the next layer of the data structure B_i. Given the first k qubits of the register being in a state Ψ_k, then the action on the $k+1$ qubit be as follows:

$$|i\rangle|0\rangle^{\otimes t'}|\Psi_k\rangle|0\rangle \xrightarrow{\text{QMD}} |i\rangle|B^{'}_{i,d}\rangle|\Psi_k\rangle|0\rangle \xrightarrow{\sigma_y(B^{'}_{i,d})} |i\rangle|B^{'}_{i,d}\rangle|\Psi_k\rangle\frac{1}{\sqrt{B^{l}_{i,d}}}\left(\sqrt{B^{2l}_{i,d+1}}|0\rangle + \sqrt{B^{2l+1}_{i,d+1}}|1\rangle\right), \tag{3.30}$$

Where $\xrightarrow{\sigma_y(B^{'}_{i,k})}$ indicates a cascade of σ_y rotations controlled on the address register and targeted on the $k+1$ qubit of the main register.

Repeating this procedure for the first $n-1$ layers of the tree $B^{'}_i$ will leave the main register in the state $|\Psi_{\log(n)}\rangle$, i.e. the amplitude encoding of the moduli of the vector V_i. The addition of the phases will require a quantum query to the classical data structure to access the final layer of the binary tree. Following the query, a cascade of phase gates on the $\log(n)^{th}$ qubit is performed. This is followed by a σ_x gate and then another cascade of phase rotations and finally another σ_x gate. The σ_x gate is required since the phase gates only acts on the state where the last bit is a 1. The transformation will then be:

$$|i\rangle|0\rangle^{\otimes t'}|\Psi_{\log(n)}\rangle \xrightarrow{\text{QMD}} |i\rangle|B^{'}_{i,\log(n)}\rangle|\Psi_{\log(n)}\rangle \xrightarrow{\text{CP}(B^{'}_{i,\log(n)})} |i\rangle|B^{'}_{i,\log(n)}\rangle e^{i\theta_{il}}|\Psi_{\log(n)}\rangle, \tag{3.31}$$

Where θ_{il} is the phase saved in the leaf l. From this we can see that performing $\tilde{U} : |i\rangle|0\rangle \to |i\rangle|V_i\rangle$ for $i \in [m]$ requires $2\log(n)+2$ queries to the classical data structure. In addition, a precision of t will require at most $t'(\log(n)+2)$ controlled rotations.

The final tree is for the implementation of $\tilde{V} : |0\rangle|j\rangle \to |\tilde{V}\rangle|j\rangle$, for $j \in [n]$. As before we construct the tree such the leaves store $||V_i||^2$ and an internal node l stores the entries of the subtree rooted at l. It is important to note that in this case the node does not need to store the phases since we are interesting in the moduli of the rows, which halves the dimension of this tree. Following an analogous procedure of pruning, a circuit of controlled σ_y rotations will grant the application of the unitary $\tilde{V}$ to the main register. □

We now show an example of the state preparation with pruned KP-trees of a vector of size 4. Figure 3.9 shows the difference between an original KP-tree and a pruned version, whilst 3.10 shows the full circuit implementation to load the vector.

Example 3.1. As a worked example we will take a 4 element vector for which we will construct a data structure and then, using a QMD, we will load it into a quantum circuit. The vector is:

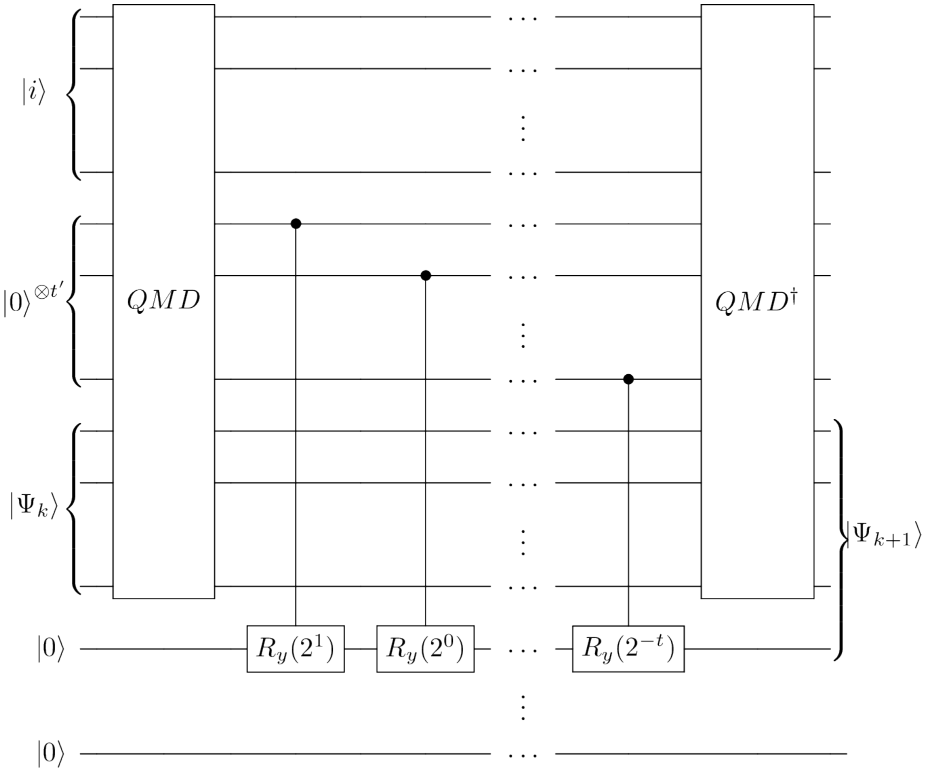


Figure 3.7: A section of the circuit for state preparation with pruned KP-trees. For a vector $V_i \in \mathbb{C}^n$ we require 3 register: an index register $|i\rangle$; a t-qubit address register which will load the angles up to a precision t; and the main register composed of $\log(n)$ qubits. At a intermediate step of the procedure the main register will hold a state which represents the k-th layer of the KP-tree Ψ_k and the aim of the circuit will be to prepare the $k+1$ layer of the tree, Ψ_{k+1}. This starts with a quantum query to the data structure which loads the angles to the address register. This is followed by a cascade of controlled rotations and finally an inverse call to the data structure. Repeating this circuit $\log(n)$ times produces the state $\Psi_{\log(n)}$ which is a vector which hold the moduli of the components of V_i. In total the circuit will have a depth of $t\log(n)$ and will require $2\log(n)$ queries to the data structure.

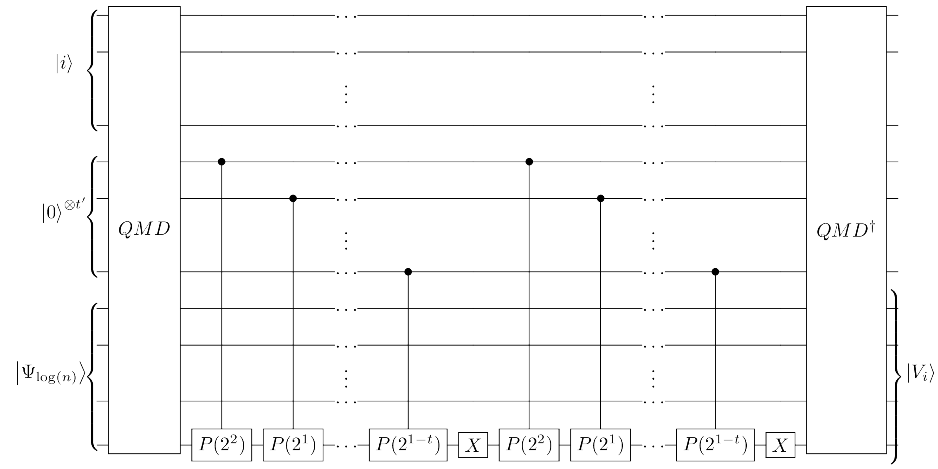


Figure 3.8: The circuit that adds the phase in state preparation with pruned KP-trees. This starts with a quantum query to the data structure which loads the angles to the address register, which is followed by a cascade of phase rotation, a NOT gate and another cascade. The requirmenet of 2 cascades is that the controlled phase rotations only acts on the quantum states with a 1 as the last bit. The application of this circuit produces the desired vector V_i.

$$V^T = \left(0.4e^{i\frac{\pi}{4}}, 0.4e^{i\frac{\pi}{12}}, 0.8e^{i\frac{\pi}{3}}, 0.2e^{i\frac{\pi}{6}}\right), \tag{3.32}$$

In particular our aim is to produce the state:

$$|\phi\rangle = 0.4e^{i\frac{\pi}{4}}|00\rangle + 0.4e^{i\frac{\pi}{12}}|01\rangle + 0.8e^{i\frac{\pi}{3}}|10\rangle + 0.2e^{i\frac{\pi}{6}}|11\rangle, \tag{3.33}$$

The first step is the construction of the data structure. Figure 3.9 shows the initial tree and the pruned tree. The difference between the two is that the initial tree stores the partial norms of the entries, whilst the pruned tree stores the angles required to implement the partial norms as σ_y rotations.

Since we are working with a vector we will only have 1 qubit in the index register which will be set to the state $|0\rangle$. The angle will be stored with a precision of t which we will assume is a large number. This will require the angle(address) register to be composed of t qubits. Finally, since the vector has 4 elements, we will require 2 qubits in the main register.

An initial call to the QMD will load the angle on the address register, this will be followed by a cascade of σ_y rotations on the first qubit and an inverse call to the QMD to remove the angle from the address register. This will produce the state:

$$|0\rangle|0\rangle^{\otimes t}\left(\sqrt{0.32}|0\rangle + \sqrt{0.68}|1\rangle\right)|0\rangle, \tag{3.34}$$

A similar process is done is the second step where in this case the call to the QMD is made using the index register and the first qubit and will access the second row of the pruned KP-tree. This will be followed by a similar cascade of σ_y rotations on the second qubit of the main register and inverse call to the QMD to remove the angle from the address register. This will produce the state:

$$|0\rangle|0\rangle^{\otimes t}\left(\sqrt{0.32}|0\rangle\left(\sqrt{\frac{0.16}{0.32}}|0\rangle + \sqrt{\frac{0.16}{0.32}}|1\rangle\right) + \sqrt{0.68}|1\rangle\left(\sqrt{\frac{0.64}{0.68}}|0\rangle + \sqrt{\frac{0.2}{0.68}}|1\rangle\right)\right), \tag{3.35}$$

$$= |0\rangle|0\rangle^{\otimes t}\left(0.4|00\rangle + 0.4|01\rangle + 0.8|10\rangle + 0.2|11\rangle\right), \tag{3.36}$$

Now we need to add the phases. As already seen, this will be done with a query to the QMD to access the final row of the pruned KP-tree, then a cascade of phase rotations P. This will produce the state:

$$|0\rangle|\theta\rangle\left(0.4|00\rangle + 0.4e^{i\frac{\pi}{12}}|01\rangle + 0.8|10\rangle + 0.2e^{i\frac{\pi}{6}}|11\rangle\right), \tag{3.37}$$

Where the state $|\theta\rangle$ is some superposition containing the binary representation of all the leaves of the pruned KP-tree. Because the nature of the phase gate, a σ_x gate followed by another cascade of phase rotations is required to load the remaining phases. Finally, a σ_x gate and a inverse query to the QMD will produce the desired state!

$$|0\rangle|0\rangle^{\otimes t}\left(0.4e^{i\frac{\pi}{4}}|00\rangle + 0.4e^{i\frac{\pi}{12}}|01\rangle + 0.8e^{i\frac{\pi}{3}}|10\rangle + e^{i\frac{\pi}{6}}0.2|11\rangle\right), \tag{3.38}$$

The full circuit for this can be seen in the figure 3.10.

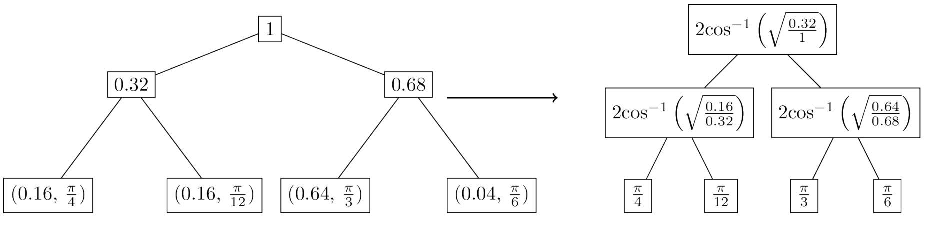

1
0.32
0.68
$(0.16,\ \frac{\pi}{4})$
$(0.16,\ \frac{\pi}{12})$
$(0.64,\ \frac{\pi}{3})$
$(0.04,\ \frac{\pi}{6})$
$2\cos^{-1}\left(\sqrt{\frac{0.32}{1}}\right)$
$2\cos^{-1}\left(\sqrt{\frac{0.16}{0.32}}\right)$
$2\cos^{-1}\left(\sqrt{\frac{0.64}{0.68}}\right)$
$\frac{\pi}{4}$
$\frac{\pi}{12}$
$\frac{\pi}{3}$
$\frac{\pi}{6}$

Figure 3.9: The original and pruned tree used for the example. The difference between the two is that the initial tree stores the partial norms of the vector entries, whilst the pruned tree only stores the angles required to implement the required σ_y and phase rotations.

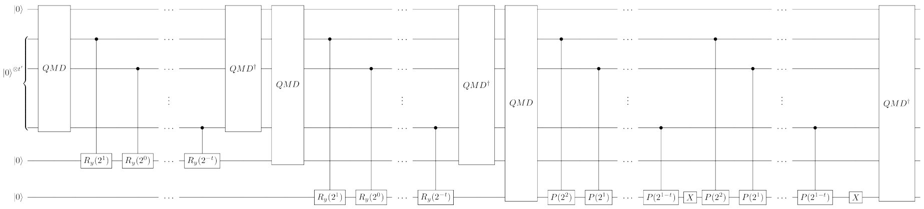


Figure 3.10: The circuit to perform state preparation of the vector reported in the worked example. An initial call to the QMD loads the angle on the address register, which is followed by a cascade of σ_y rotations on the first qubit and an inverse call to the QMD to remove the angle from the address register. A similar process is done for the second step where the call to the QMD is made using the index register and the first qubit and will access the first layer of the pruned KP-tree. This will be followed by a similar cascade of σ_y rotations on the second qubit of the main register and inverse call to the QMD. The final step requires adding the phases. This will be done with a query to the QMD to access the final row of the pruned KP-tree, then a cascade of phase rotation P. Because the nature of the phase gate, a σ_x gate followed by another cascade of phase rotations is required to load the remaining phases. Finally, a σ_x gate and a inverse query to the QMD will produce the amplitude encoding.

The following exercise might be helpful to clarify the relation between quantum query access to a vector and quantum sampling access.

Exercise 3.8. Suppose you have quantum access to a vector $x = [x_1, \dots, x_N]$, where each $x_i \in [0, 1]$. What is the cost of creating quantum sampling access to x, i.e. the cost of preparing the state $\frac{1}{Z}\sum_{i=1}^{N} x_i|i\rangle$. Hint: query the state in superposition and perform a controlled rotation. Can you improve the cost using amplitude amplification? What if $x_i \in [0, B]$ for a $B > 1$?

Lower bounds in query complexity can be used to prove that the worst case for performing state preparation with the technique used in the exercise (i.e. without KP-trees/quantum sampling access) are $O(\sqrt{N})$.

3.3.3 Block encoding

We first turn our attention to creating block encodings from oracles doing amplitude encoding. In the following, we use the shortcut notation to define the matrix $A^{(p)}$ to be a matrix where each entry $A^{(p)}_{ij} = (A_{ij})^p$.

Exercise 3.9. Let $X \in \mathbb{R}^{n\times d}$. Suppose you have access to U_R and U_L defined as:

- $U_R|i\rangle|0\rangle = |i\rangle|x_i\rangle = |i\rangle\sum_j (x_i)_j|j\rangle$;
- $U_L|0\rangle|i\rangle = |\tilde{x}\rangle|i\rangle$ where $\tilde{x}$ is the vector of the norms of the rows of the matrix x.

Using once U_R and U_L, build a unitary U_X that performs the mapping $|0\rangle \mapsto \frac{1}{\|X\|_F}\sum_{i,j} x_{ij}|i,j\rangle$.

Theorem 3.13 (Creation of state matrix)**.** *Assume to have the unitaries U_L and U_R for a matrix $X \in \mathbb{R}^{n\times n}$. Then $U_R^\dagger U_L$ is a $(\|X\|_F, 0)$-block encoding of X.*

Proof. Define the matrix $P \in \mathbb{R}^{nm}$ by the column vectors $|i\rangle|x_i\rangle$ for $i \in [m]$, and the matrix Q by $Q \in \mathbb{R}^{nn\times nn}$ defined by the column vectors $|\overline{x}\rangle|i\rangle$. One can verify that

$$(P^\dagger Q)_{ij} = \langle i, x_i|\overline{x}, j\rangle = \frac{X_{ij}}{\|X\|_F}.$$

To conclude the proof, it suffices to recall the definition of block encoding:

$$\|X - \|X\|_F(\langle 0| \otimes I)U_R^\dagger U_L(|0\rangle \otimes I)\| = \|X - \|X\|_F P^\dagger Q\| = 0.$$

□

Definition 3.14 (Possible parameterization of for the KP-trees)**.** For $s_p(A) = \max_{i\in[n]}\sum_{j\in[d]} A_{ij}^p$, we chose $\mu_p(A)$ to be:

$$\mu_p(A) = \min_{p\in[0,1]}(\|A\|_F\,, \sqrt{s_{2p}(A)s_{(1-2p)}(A^T)}). \tag{3.39}$$

Lemma 3.3 (Block encodings from quantum data structures (Chakraborty et al., 2019))**.** *Let $A \in \mathbb{C}^{M\times N}$, and $\overline{A} \in \mathbb{C}^{(M+N)\times(M+N)}$ be the symmetrized matrix defined as:*

$$\overline{A} = \begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix}. \tag{3.40}$$

- *Fix $p \in [0,1]$. If $A \in \mathbb{C}^{M\times N}$, and $A^{(p)}$ and $(A^{(1-p)})^\dagger$ are both stored in quantum-accessible data structures with sufficient precision, then there exist unitaries U_R and U_L that can be implemented in time $O\left(polylog(MN/\epsilon)\right)$ such that $U_R^\dagger U_L$ is a $(\mu_p(A), \lceil\log(N+M+1)\rceil, \epsilon)$-block encoding of $\overline{A}$.*
- *On the other hand, if A is stored in a quantum-accessible data structure with sufficient precision, then there exist unitaries U_R and U_L that can be implemented in time $O(polylog(MN)/\epsilon)$ such that $U_R^\dagger U_L$ is a $(\|A\|_F, \lceil\log(M+N)\rceil, \epsilon)$-block encoding of $\overline{A}$.*

The second point of the previous theorem is equivalent to what we saw in Section 3.3.2.2.

3.3.3.1 Block encoding from sparse access

Using this we can then see how to perform a block encoding. We start by reporting this result from (Gilyén et al., 2019).

Theorem 3.14 (Block-Encoding from sparse access (Gilyén et al., 2019))**.** *Let $A \in \mathbb{C}^{2^w \times 2^w}$ be a matrix that is s_r-row-sparse and s_c-column-sparse, and each element of A has absolute value at most 1. Suppose that we have access to the following sparse access oracles acting on two $(w+1)$ qubit registers:*

$$O_r : |i\rangle|k\rangle \mapsto |i\rangle|r_{ik}\rangle \forall i \in [2^w]-1, k \in [s_r], and \tag{3.41}$$

$$O_c : |l\rangle|j\rangle \mapsto |c_l j\rangle|j\rangle \forall l \in [s_c], j \in [2^w]-1, where \tag{3.42}$$

r_{ij} is the index of the j-th non-zero entry of the i-th row of A, or if there are less than i non-zero entries, than it is $j+2^w$, and similarly $c_i j$ is the index for the i-th non-zero entry of the $j-th$ column of A, or if there are less than j non-zero entries, than it is $i+2^w$. Additionally assume that we have access to an oracle O_A that returns the entries of A in binary description:

$$O_A : |i\rangle|j\rangle|0\rangle^{\otimes b} \mapsto |i\rangle|j\rangle|a_{ij}\rangle \forall i, j \in [2^w]-1 where \tag{3.43}$$

a_{ij} is a b-bit description of the ij-matrix element of A. Then we can implement a $(\sqrt{s_r s_c}, w+3, \epsilon)$-block encoding of A with a single use of O_r, O_c, two uses of O_A and additionally using $O(w + \log^{2.5}(\frac{s_r s_c}{\epsilon}))$ one and two qubit gates while using $O(b, \log^{2.5}\frac{s_r s_c}{\epsilon})$ ancilla qubits.

The previous theorems can be read more simply as: “under reasonable assumptions (**quantum general graph model** for rows and for columns - see previous section), we can build $(\sqrt{s_r s_c}, w{+}3, \epsilon)$-block encodings of matrices A with circuit complexity of $O(\log^{2.5}(\frac{s_r s_c}{\epsilon}))$ gates and constant queries to the oracles”.

The interested reader can read (Camps et al., 2024, @yang2024quantum) to learn how to create block encodings from sparse access.

3.4 Use case: working with classical probability distributions

We have 4 ways of working with classical probability distributions in a quantum computer (Gur et al., 2021):

- Purified query access
- Sample access
- Query access to a frequency vector of a distribution

- Drawing samples classically and perform some computation on a quantum computer

Let's start with a formal definition of the frequency vector model. This is a kind of query access, but since it is used only for probability distrubtions we decided to include it in this section.

Definition 3.15 (Quantum query access to a probability distribution in the frequency vector model)**.** Let $p = (p_1, p_2, ... p_n)$ be a probability distribution on $\{1, 2, ..., n\}$, and let $n \geq S \in \mathbb{N}$ be such that there is a set of indices $S_i \subseteq [S]$ for which $p_i = \frac{|S_i|}{S}$. We have quantum access to a probability distribution in the frequency vector model if there is an quantum oracle that, for $\forall s \in [S_i]$ performs the mapping $O_p|s\rangle \mapsto |s\rangle|i\rangle$.

Exercise 3.10. Given a matrix $M := \sum_{j=0}^{n-1} a_j U_j$, which is a linear combination of unitary matrices U_j, consider the unitary transformation

$$U_M = (U_{\text{PREP}}^{\dagger} \otimes I) U_{\text{SEL}} (U_{\text{PREP}} \otimes I) \tag{3.44}$$

where $U_{\text{PREP}}|0\rangle = |a\rangle = \sum_{j=0}^{n-1} a_j |j\rangle$ and $U_{\text{SEL}} = \sum_{j=0}^{n-1} |j\rangle\langle j| \otimes U_j$. Then show that U_M is a block encoding of M:

$$(\langle 0| \otimes I)\, U_M\, (|0\rangle \otimes I) = M = |a_j|^2 \sum_{j=0}^{n-1} b_j U_j \tag{3.45}$$

3.5 Retrieving Data

In order to retrieve information from a quantum computer, we are going to use some efficient procedures that allow to reconstruct classically the information stored in a quantum state. These procedures can be thought of as ways of sampling from a pure state $|x\rangle$. The idea for an efficient quantum tomography is that we want to minimize the number of times that the state $|x\rangle$ is created.

Most of the quantum algorithms discussed here will work with pure quantum states. We assume to have access to the unitary that creates the quantum state that we would like to retrieve, and that we have access to the unitary that creates the state (and that we can control it). Under these conditions, the process of performing tomography is greatly simplified. According to the different error guarantees that we require, we can chose among two procedures.

Theorem 3.15 (Vector state tomography with L2 guarantees (Kerenidis and Prakash, 2018))**.** *Given access to unitary U such that $U|0\rangle = |x\rangle$ and its* *controlled version, there is a tomography algorithm with calls U and its controlled version for $\frac{d \log d}{\epsilon^2}$) times, that produces a unit vector $\tilde{x} \in \mathbb{R}^d$ such that $\|\tilde{x} - |x\rangle\|_2 \leq \epsilon$ with probability at least $(1 - 1/poly(d))$.*

Theorem 3.16 (Vector state tomography with L∞ guarantees (Kerenidis et al., 2019b))**.** *Given access to unitary U such that $U|0\rangle = |x\rangle$ and its controlled version, there is a tomography algorithm with calls U and its controlled version $\frac{\log d}{\epsilon^2}$) that produces a unit vector $\tilde{x} \in \mathbb{R}^d$ such that $\|\tilde{x} - |x\rangle\|_{\ell_\infty} \leq \epsilon$ with probability at least $(1 - 1/poly(d))$.*

Note that in both kinds of tomography the dependence on the error in the denominator is quadratic, and this is because of the Hoeffding inequality. Another remark on the hypothesis of the algorithms for tomography is that they require a unitary U such that $U|0\rangle \mapsto |x\rangle$ for the $|x\rangle$ in question. Often times, due to the random error in the quantum subroutines used inside the algorithms, this state $|x\rangle$ might slightly change every time.

Exercise 3.11. Can you obtain ℓ_2 tomography with error ϵ on a d dimensional state if you have have only access to an algorithm that perform ℓ_∞ tomography with error ϵ^* on the same state? (I.e. what should you set the value of ϵ^*?).

Exercise 3.12. Consider Theorem 3.15 and Theorem 3.16. What is the sample complexity of a tomography algorithm returning error in norm ℓ_p? In other words, find the sample complexity to return a vector such that $\|\tilde{x} - |x\rangle\|_{\ell_p} \leq \epsilon$ with probability at least $(1 - 1/poly(d))$.

3.5.1 Density matrices

Much of the current literature in quantum tomography is directed towards reconstructing a classical description of density matrices. This problem is considerably harder than reconstructing a pure state.

Theorem 3.17 (Efficient quantum tomography (O'Donnell and Wright, 2016))**.** *An unknown rank-r mixed state $\rho \in \mathbb{C}^{d \times d}$ can be estimated to error ϵ in Frobenius distance using $n = O(d/\epsilon^2)$ copies, or to error ϵ in trace distance using $n = O(rd/\epsilon^2)$ copies.*

Different techniques have been recently developed in (Zhang et al., 2020). There, the authors used the assumption on doing tomography on a state $|x\rangle$ that is in the row space of a rank r matrix A for which we have quantum access. They propose an algorithm to obtain the classical description of the coefficients x_i in the base spanned by the rows $\{A_i\}_{i=0}^{r}$of A, so that $|x\rangle = \sum_i^r x_i |A_i\rangle$. This requires $\widetilde{O}(poly(r))$ copies of the output states and $\widetilde{O}(poly(r), \kappa^r)$ queries to input oracles. While this procedure has the benefit of not being linear in the output dimension of the final state, the high dependence on the rank might hide the advantages compared to the previous quantum tomography procedures. For completeness, the result is as follows.

Theorem 3.18 (Improved quantum tomography (Zhang et al., 2020))**.** *For the state $|v\rangle$ lies in the row space of a matrix $A \in \mathbb{R}^{n\times d}$ with rank r and condition number $\kappa(A)$, the classical form of $|v\rangle$ can be obtained by using $O(r^3\epsilon^2)$ queries to the state $|v\rangle$, $O(r^{11}\kappa^{5r}\epsilon^{-2}\log(1/\delta))$ queries to* QRAM *oracles of A and $O(r^2)$ additional inner product operations between rows, such that the ℓ_2 norm error is bounded in ϵ with probability at least $1-\delta$.*

Chapter 4

Classical machine learning

Machine learning, also called narrow artificial intelligence, has been defined as “the study of computer algorithms that allow computer programs to automatically improve through experience (Mitchell et al., 1997). Machine learning is often divided into supervised and unsupervised methods. We use supervised learning when the dataset is supervised, i.e. when the dataset consist of pairs of input objects (usually vectors) and a desired output value (called the supervised signal), which can be a label or a number. In case the output is a label the supervised problem is said to be called classification, and we call regression the other case. Supervised learning can be thought of as the task of learning a mapping or a function form pairs of input and output. When the dataset in unsupervised, the problem is called clustering, and consist in finding hidden structure of the process that has generated the dataset. Computationally, much of the machine learning algorithms can be described by operations on vectors and matrices. For instance, many machine learning algorithms are reduced to computing the eigenvectors of matrices obtained from the data. In the last 15 years machine learning has been applied in all the sectors of information technology. In this chapter, we review and introduce some classical machine learning. Special emphasis is put on formalizing the connection between the machine learning problems and their linear-algebraic formulation.

The dataset that we manipulate in this work are represented by a matrix $V \in \mathbb{R}^{n\times d}$, i.e. each row can be thought as a vector $v_i \in \mathbb{R}^d$ for $i \in [n]$ that represents a single data point. We denote as V_k the optimal rank k approximation of V, that is $V_k = \sum_{i=0}^{k} \sigma_i u_i v_i^T$, where u_i, v_i are the row and column singular vectors respectively and the sum is over the largest k singular values σ_i. We denote as $V_{\geq \tau}$ the matrix $\sum_{i=0}^{\ell} \sigma_i u_i v_i^T$ where σ_ℓ is the smallest singular value which is greater than τ. For a matrix M and a vector x, we define as $M^{+}_{\leq \theta,\delta} M_{\leq \theta,\delta} x$ the projection of x onto the space spanned by the singular vectors of M whose corresponding singular values are smaller than θ, and some subset of singular vectors whose corresponding singular values are in the interval $[\theta, (1+\delta)\theta]$.

4.1 Supervised learning

Supervised (or predictive) machine learning is the part of machine learning that deals with supervised datasets, i.e. data where each sample x_i comes along with supervised information, i.e. a piece of data y_i. It helps the intuition thinking that the supervised information comes from a stochastic process that maps vectors x_i to vectors y_i. The goal is to model the mapping on the whole input space $\mathcal{X}$ to the output space $\mathcal{Y}$ given a set of input-output pairs $D = \{(x_i, y_i)\}_{i=0}^{n}$. Usually, the input space is a subset of $\mathbb{R}^d$, and the output space is usually either $\mathbb{R}$ or a finite set K of small cardinality. It is practical, for the sake of exposition to consider the training set organized into a matrix $X \in \mathbb{R}^{n\times d}$ and the matrix $Y \in R^n$ or $Y \in [K]^n$. The components of a vector x_i, i.e. a row of X are called *features, attributes, or covariates.* The matrix X is called *design matrix*, or simply the dataset. The vector y_i is called the *response variable.* If the response variable is categorical (or nominal), the problem is known as classification, or pattern recognition. If the response variable is real-valued we interpret this problem as learning a function $f : \mathbb{R}^d \mapsto \mathbb{R}$ and we call this problem regression. Different assumptions on the structure of f lead to different machine learning models. Each model can be trained (or fitted) with different algorithms.

4.2 Unsupervised learning

Unsupervised learning (Murphy, 2012), which sometimes goes under the name of knowledge discovery, is the part of machine learning that deals with understanding unlabeled data. In the dataset $D = \{x_i\}_{i=0}^{n}$ we don't have anymore any supervised information. In this case, it is common to understand the structure of the process generating the samples by formalizing a *density estimation* problem: we want to learn the parameters θ of a function $p(x_i|\theta)$ that models the distribution of the process that has generated the samples. The importance of unsupervised learning lies in the stunning similarity with human and animal learning. Furthermore, most of the dataset that we have are unsupervised, as it is costly to provide supervised information from experts or humans. The most common example of unsupervised learning is clustering, where we want to partition into groups a given dataset. As an example, imagine having a set comprising of images of cats and dogs, without knowing which image is a cat or which is a dog. An unsupervised learning algorithm is supposed to learn how to split the dataset correctly, by understanding the characteristics and features that allows discriminating between images of different kinds. Just to name a few of the more concrete examples, in astronomy, clustering is often used to discover new kinds of stars, in biology, it is used to find new kinds of cells, in cybersecurity, to perform anomaly detection, and so on.

We refer to the number of clusters in the dataset with a letter K. The first goal in clustering is to understand the right number of different groups in the data (which might not be known a-priori). The second goal is to estimate which cluster each point x_i belongs to. We define z_i for $z_i \in [K]$ as the cluster to which point x_i is assigned to. The value of z_i is often called *hidden* or *latent variable*. Unsupervised learning can be seen as the task of guessing the value of the hidden variable, by computing $z_i = \arg\max_k p(z_i = k|x_i, \theta)$. For this, an unsupervised learning algorithm has to model (implicitly or explicitly) the joint probability distribution $p(x, y)$.

While latent variables have extensive applications, in this thesis we will focus on the case where latent variables are used to represent a discrete latent state (as in clustering).

4.3 Generative and discriminative models

There is another insightful way of organizing machine learning models. They can either be *generative* or *discriminative*. A discriminative model learns just the mapping $p(y|x)$, and provides a way to classify points (i.e. infer the value of y_i), without actually knowing “how’ ’ the point x_i has been generated. Examples of such models are: k-nearest neighbors, Logistic regression, Support Vector Machines, Decision Trees, Random Forest, Neural Networks, and so on. Examples of such models are the QFDC, QSFA in Chapter 9.2.1. On the other way, generative models output a model for the joint probability distribution $p(x, y)$. This is similar to the unsupervised learning case, but in this cases the dependence on y (which can be a hidden variable) is made explicit. In general, discriminative models make fewer assumptions, as generative models often need to do some assumption on the structure of $p(x)$. Such generative models are one of the most promising approaches to unsupervised problems. The goal of a generative model is to learn a probability distribution that is most likely to have generated the data collected in a training set. Fitting the model consists of learning the parameters of a probability distribution p in a certain parameterized family, that best describes our vectors x_i, y_i. In case the data is unsupervised, generative models learn the probability distribution $p(x_i)$ assuming the existence of some hidden variables z_i. Examples of such models in this thesis are q-means and GMM, i.e. chapter @ref{chap-q-means} and @ref{chap-qem}. A possible way to fit a generative model is to formulate the problem of finding the parameters of the family of distribution as an optimization problem. This is often done using the so-called maximum likelihood estimation (MLE). One can think of the *likelihood* as the function that we use to measure how good a model is for explaining a given dataset. For a given machine learning model with parameters γ, the likelihood of our data set X is the probability that the data have been generated by the model with parameters γ, assuming each point is independent and identically distributed. We think of the likelihood as a function of γ, holding the dataset X fixed. For $p(x_i|\gamma)$ the probability that a point x_i comes from model γ, the likelihood is defined as:

$$L(\gamma; X) := \prod_{i=1}^{n} p(x_i|\gamma) \tag{4.1}$$

From this formula, we can see that in order to find the best parameters γ^* of our model we need to solve an optimization problem. For numerical and analytical reasons, instead of maximizing the likelihood L, it is common practice to find the best model by maximizing the *log-likelihood* function $\ell(\gamma; X) = \log L(\gamma; X) = \sum_{i=1}^{n} \log p(x_i|\gamma)$. In this context, we want to find the model that maximizes the log-likelihood:

$$\gamma_{ML}^{*} := \arg\max_{\gamma} \sum_{i=1}^{n} \log p(x_i|\gamma). \tag{4.2}$$

The procedure to calculate the log-likelihood depends on the specific algorithm used to model the data. A possible solution would be to use a gradient-based optimization algorithm on ℓ. It is often the case that, due to the indented landscape of the likelihood function, gradient-based techniques often do not perform well. Therefore, it is common to find other strategies to find do maximum likelihood estimation. One of which is the Expectation-Maximization (EM) algorithm, detailed in chapter 11.

4.4 Dimensionality reduction

Dimensionality reduction (DR), a technique used both in supervised and unsupervised learning, refers to the procedure by which the dimension of the input data is reduced while retaining most of the meaningful information contained therein. It is often a necessary step when trying to solve practical problems in machine learning and there are many techniques for performing it. For instance, it is used to decrease the variance of a model, since it can lead to models with a fewer number of parameters, and it might just reduce the noise in the data. It is also necessary when the runtime of the algorithm has polynomial dependence on the number of features, as it is often the case for nowadays datasets. In the context of big data analysis, by removing features that carry low information (like features that are strictly proportional to other features, or features for which the data contains too little information), it is possible to optimize the storage space. It can be also used for data visualization. Most importantly, supervised algorithms often suffer from the *curse of dimensionality*: by allowing large dimensionality of data, the informative power of the data points in the training set decreases, thus leading to a degradation in classification performances. One solution to improve the accuracy would be to increase the number of elements in the training set, but this is not always possible nor desirable, so the common route is to decrease the dimension of the data. Mathematically, the idea of the dimensionality reduction algorithms is to map vectors from a high dimensional space $\mathcal{X}$ to a low dimensional space $\mathcal{Y}$, such that the most meaningful information (according to some criteria) is preserved. Of course, understanding which criterion to use is far from trivial.

The choice of the right DR algorithm depends on the nature of the data as well as on the type of algorithm that will be applied after the dimensionality reduction. A very well-known DR algorithm is the Principal Component Analysis (PCA), which projects the data points onto the subspace spanned by the eigenvectors associated to the k largest eigenvalues of the covariance matrix of the data. In this way, the projection holds "most of the information' ' of the dataset. It is possible to show (Murphy, 2012) that for a subspace of dimension k, this choice of eigenvectors minimizes the reconstruction error, i.e. the distance between the original and the projected vectors. However, PCA is not always the best choice of dimensionality reduction. PCA projects the data into the subspace along which the data has more variance. This does not take into consideration the information that different points might belong to different classes, and there are cases in which PCA can worsen the performance of the classifier. Other methods, like Fisher Linear Discriminant and Slow Feature Analysis take into account the variance of every single class of points. Indeed, FLD projects the data in a subspace trying to maximize the distance between points belonging to different clusters and minimizing the distance between points belonging to the same cluster, thus preserving or increasing the accuracy.

4.5 Generalized eigenvalue problems in machine learning

Here we review the connection between the so-called Generalized Eigenvalue Problem (GEP) and some models in machine learning. In classical literature, this is a well-known subject (Ghojogh et al., 2019), (De Bie et al., 2005), (Borga et al., 1997).

Definition 4.1 (Generalized Eigenvalue Problem)**.** Generalized Eigenvalue Problem] Let $A, B \in \mathbb{R}^{d \times d}$ be two SPD matrices. The GEP is defined as:

$$AW = BW\Lambda$$

The columns $w_i \in \mathbb{R}^d$ of W and the values $\lambda_i = \Lambda_{ii} \in \mathbb{R}$ of the diagonal matrix Λ are the so-called generalized eigenvector and eigenvalues.

The generalized eigenvalue problem is denoted by (A, B) (note that the order in the pair matters). As is evident, the canonical eigenvalue problem is a special case of the GEP where $B = I$. In this work, we will often consider the case when matrices A and B consist of expectation values from stochastic processes, that is, these are covariance matrices of some sort. Furthermore, while A can be symmetric semi-positive definite, we require B to be invertible, and thus symmetric positive definite. The GEP is related to the so-called *Raylight quotient*: a variational extremum problem related to the ratio of two quadratic forms involving matrix A and B:

$$\rho(w) := \frac{w^T Aw}{w^T Bw} \tag{4.3}$$

There many different optimization problems that can be reduced to a GEP, which we report here for completeness (De Bie et al., 2005), (Ghojogh et al., 2019). One can see that the norm of w does not change the value of the optimization problem. Therefore, we can impose an additional constraint on w. In this way, we can reformulate the problem as a constrained optimization problem, without losing any solution. This constraint is $w^T Bw = 1$. We will describe the relation between Equation (4.3) and Equation in definition 4.1. (To appear)

4.6 How to evaluate a classifier

Practitioners in quantum machine learning should not only build their skills in quantum algorithms, and having some basic notions of statistics and data science won't hurt. In the following the see some ways to evaluate a classifier. What does it means in practice? Imagine you have a medical test that is able to tell if a patient is sick or not. You might want to consider the behavior of your classier with respect to the following parameters: the cost of identifying a sick patient as healthy is high, and the cost of identifying a healthy patient as sick. For example, if the patient is a zombie and it contaminates all the rest of the humanity you want to minimize the occurrences of the first case, while if the cure for "zombiness" is lethal for a human patient, you want to minimize the occurrences of the second case. With P and N we count the number of patients tested Positively or Negatively. This is formalized in the following definitions, which consists in statistics to be calculated on the test set of a data analysis.

- **TP True positives (statistical power)** : are those labeled as sick that are actually sick.

- **FP False positives (type I error)**: are those labeled as sick but that actually are healthy
- **FN False negatives (type II error)** : are those labeled as healthy but that are actually sick.
- **TN True negative**: are those labeled as healthy that are healthy.

Given this simple intuition, we can take a binary classifier and imagine to do an experiment over a data set. Then we can measure:

- **True Positive Rate (TPR) = Recall = Sensitivity**: is the ratio of correctly identified elements among all the elements identified as sick. It answer the question: "how are we good at detecting sick people?".

  $$\frac{TP}{TP+FN} + \frac{TP}{P} \simeq P(test = 1|sick = 1)$$

  This is an estimator of the probability of a positive test given a sick individual.

- **True Negative Rate (TNR) = Specificity** is a measure that tells you how many are labeled as healthy but that are actually sick.

  $$\frac{TN}{TN+FP} = p(test = 0|sick = 0)$$

  How many healthy patients will test negatively to the test? How are we good at avoiding false alarms?

- **False Positive Rate = Fallout**

  $$FPR = \frac{FP}{FP+TN} = 1 - TNR$$

- **False Negative Rate = Miss Rate**

  $$FNR = \frac{FN}{FN+TP} = 1 - TPR$$

- **Precision, Positive Predictive Value (PPV)**:

  $$\frac{TP}{TP+FP} \simeq p(sick = 1|positive = 1)$$

  How many positive to the test are actually sick?

- F_1 **score** is a more compressed index of performance which is a possible measure of performance of a binary classifier. Is simply the harmonic mean of Precision and Sensitivity:

  $$F_1 = 2\frac{Precision \times Sensitivity}{Precision + Sensitivity}$$

- **Receiver Operating Characteristic (ROC)** Evaluate the TRP and FPR at all the scores returned by a classifier by changing a parameter. It is a plot of the true positive rate against the false positive rate for the different possible value (cutpoints) of a test or experiment.
- The **confusion matrix** generalize these 4 combination of (TP TN FP FN) to multiple classes: is a $l \times l$ where at row i and column j you have the number of elements from the class$i$ that have been classified as elements of class j.

Other links: here

Chapter 5

A useful toolbox

5.1 Phase estimation

In this section we report the theorems for the subroutines that are often used to create new quantum algorithms. Often, we report multiple formulations, so that the working quantum algorithm researcher can pick the best version for them.

Theorem 5.1 (Phase estimation (Kitaev, 1996))**.** *Let U be a unitary operator with eigenvectors $|v_j\rangle$ and eigenvalues $e^{i\theta_j}$ for $\theta_j \in [-\pi, \pi]$, i.e. we have $U|v_j\rangle = e^{i\theta_j}|v_j\rangle$ for $j \in [n]$. For a precision parameter $\epsilon > 0$, there exists a quantum algorithm that runs in time $O(\frac{T(U)\log(n)}{\epsilon}))$ and with probability $1 - 1/poly(n)$ maps a state $|\phi_i\rangle = \sum_{j\in[n]} \alpha_j|v_j\rangle$ to the state $\sum_{j\in[n]} \alpha_j|v_j\rangle|\bar{\theta_j}\rangle$ such that $|\bar{\theta}_j - \theta_j| < \epsilon$ for all $j \in [n]$.*

Theorem 5.2 (Error and probability of failure of phase estimation (Nielsen and Chuang, 2002) Section 5.2 and (Nannicini, 2019))**.** *Let $0.a = a_1, \dots a_q$ be the output of phase estimation when applied to an eigenstate with phase ϕ. If we use $q + \lceil \log(2 + \frac{1}{2\delta}) \rceil$ qubits of precision, the first q bits of a will be accurate with probability at least $1 - \delta$, i.e.*

$$Pr[|\phi - \sum_{j=1}^{q} a_j 2^{-j}| \leq 2^{-q}] \geq 1 - \delta$$

While the standard implementation of phase estimation is based on the quantum Fourier transform (QFT) circuit (Nielsen and Chuang, 2002), there have been various improvements (Ahmadi and Chiang, 2010) which try to soften the dependence on the QFT circuit, while retaining the accuracy guarantees offered by the QFT in estimating the angles θ_j.

Note that the same algorithm described in theorem 5.1 can be made ‘’consistent’’, in the sense of (Ta-Shma, 2013) and (Ambainis, 2012b). While in the original formulation of phase estimation two different runs might return different estimates for $\overline{\theta}_j$, with a consistent phase estimation this estimate is fixed, with high probability. This means that the error between two different runs of phase estimation is almost deterministic.

5.2 Grover's algorithm, amplitude games

Theorem 5.3 (Grover's algorithm). *Let $N = 2^n$ for $n > 0$. Given quantum oracle access $O_x : |i\rangle \mapsto |i\rangle|x_i\rangle$ to a vector $x = \{0,1\}^N$ where only one element of x is 1, there is a quantum algorithm that finds the index of that element using O_x only $O(\sqrt{N})$ times.*

This problem can be generalized to the case where there are multiple elements "marked" as good solutions. If we know the number of solutions in advance, the algorithm can be modified such that it fails with probability 0.

5.2.1 Amplitude estimation

Amplitude amplification and amplitude estimation are two of the workhorses of quantum algorithms.

Theorem 5.4 (Amplitude estimation, (Brassard et al., 2002)). *Given a quantum algorithm*

$$A : |0\rangle \to \sqrt{p}|y,1\rangle + \sqrt{1-p}|G,0\rangle$$

where $|G\rangle$ is some garbage state, then the amplitude estimation algorithm, using exactly P iterations of the algorithm A for any positive integer P, outputs $\tilde{p}$ $(0 \leq \tilde{p} \leq 1)$ such that

$$|\tilde{p} - p| \leq 2\pi \frac{\sqrt{p(1-p)}}{P} + \left(\frac{\pi}{P}\right)^2$$

with probability at least $8/\pi^2$. If $p = 0$ then $\tilde{p} = 0$ with certainty, and if $p = 1$ and P is even, then $\tilde{p} = 1$ with certainty.

Theorem 5.5 (Amplitude estimation (Brassard et al., 2002), formulation of (Montanaro, 2015)). *There is a quantum algorithm called amplitude estimation which takes as input one copy of a quantum state $|\psi\rangle$, a unitary transformation $U = 2|\psi\rangle\langle\psi| - I$, a unitary transformation $V = I - 2P$ for some projector P, and an integer t. The algorithm outputs $\tilde{a}$, an estimate of $a = \langle\psi|P|\psi\rangle$, such that:*

$$|\tilde{a} - a| \leq 2\pi \frac{\sqrt{a(1-a)}}{t} + \frac{\pi^2}{t^2}$$

with probability at least $8/\pi^2$, using U and V t times each. If $a = 0$ then $\tilde{a} = 0$ with certainty, and if $a = 1$ and t is even, then $\tilde{a} = 1$ with certainty.

In the original version of the Grover's algorithm we assume to know the number of marked elements, and therefore we can derive the correct number of iterations. Later on, a fixed-point version of amplitude amplification was proposed (Brassard et al., 2002) (Grover, 2005), which was then optimized in (Yoder et al., 2014). These versions do not require to know the number of iterations in advance. These results fundamentally leverage the trick that we reported in Proposition 5.1.

Let's see in practice how to use Theorem 5.4. Suppose that we want to estimate a with relative error ϵ. What is the number of times that we have to use the two unitaries? Let's check that it suffices to take $t = \frac{2\pi}{\epsilon\sqrt{a}}$, as

$$\begin{aligned} |a - \tilde{a}| \leq & \frac{2\pi\sqrt{a}\sqrt{a(1-a)}\epsilon}{2\pi} + \frac{\pi^2\epsilon^2 a}{4\pi^2} = \epsilon a\sqrt{1-a} + \frac{\epsilon^2 a}{4} \\ \leq & \epsilon a(1 - \frac{a}{2}) + \frac{\epsilon^2 a}{4} = \epsilon a \left(1 - \frac{a}{2} + \frac{\epsilon}{4}\right) \leq \epsilon a. \end{aligned} \tag{5.1}$$

In the previous equation we used the Taylor expansion of $\sqrt{1-x}$ to the second order, i.e. $\sqrt{1-x} \leq 1 - x/2$, and the fact that $\epsilon, a < 1$ in the last inequality. The asymptotic runtime of the algorithm is thus $O(\frac{1}{\epsilon\sqrt{a}})$.

What if we want to have an absolute error now? We have some of options. The simplest one is to note that a relative error of a quantity between 0 and 1 automatically translates in an absolute error. But this might not be the most elegant thing to do: since an absolute error for a quantity between 0 and 1 is "worse" than the relative error on the same quantity, we might want to save some resources, i.e. decrease the number of calls to the oracles. Let's set $t = \frac{2\pi}{\epsilon}$ and observe that

$$\begin{aligned} |a - \tilde{a}| \leq & \frac{2\pi\sqrt{a(1-a)}\epsilon}{2\pi} + \frac{\pi^2\epsilon^2}{4\pi^2} = \epsilon\sqrt{a}\sqrt{1-a} + \frac{\epsilon^2}{4} \\ \leq & \epsilon\sqrt{a}(1 - \frac{a}{2}) + \frac{\epsilon^2}{4} = \epsilon \left(\sqrt{a}(1 - \frac{a}{2}) + \frac{\epsilon}{4}\right) \leq \epsilon. \end{aligned} \tag{5.2}$$

Here, in addition to the tricks used in the relative error, we also used that $\sqrt{a} \leq 1$.

Exercise 5.1 ((Hard))**.** Another idea is to realize that we could run the algorithm returning the relative error as a black box, and set the error to $\epsilon' = \epsilon/a$. In this way we might estimate a relative error $\epsilon' a = \epsilon$, with the hope to save some resources. What is the impact of this operation in the runtime of the algorithm? It's simple to see that the runtime becomes $O(\frac{1}{\frac{\epsilon}{a}\sqrt{a}}) = O(\frac{\sqrt{a}}{\epsilon})$. Can we check if setting $t = \frac{2\pi\sqrt{a}}{\epsilon}$ can give an absolute error in $O(\frac{\sqrt{a}}{\epsilon})$ runtime? What is difficult about it?

The solution to the previous exercise consist in adding a term $\frac{1}{\sqrt{\epsilon}}$ in the number of iterations t. If we set $t = \lceil 2\pi \left(\frac{2\sqrt{a}}{\epsilon}\right) + \frac{1}{\sqrt{\epsilon}} \rceil$ we can get an absolute error.

Perhaps a simpler formulation, which hides the complexity of the low-level implementation of the algorithm, and is thus more suitable to be used in quantum algorithms for machine learning is the following:

Lemma 5.1 (Amplitude amplification and estimation (Kerenidis and Prakash, 2020))**.** *If there is a unitary operator U such that $U|0\rangle^l = |\phi\rangle = \sin(\theta)|x,0\rangle + \cos(\theta)|G,0^\perp\rangle$ then $\sin^2(\theta)$ can be estimated to multiplicative error η in time $O(\frac{T(U)}{\eta\sin(\theta)})$ and $|x\rangle$ can be generated in expected time $O(\frac{T(U)}{\sin(\theta)})$ where $T(U)$ is the time to implement U.*

Recently, various researches worked on improvements of amplitude estimation by getting rid of the part of the original algorithm that performed the phase estimation (i.e. the Quantum Fourier Transform (Nielsen and Chuang, 2002)) (Grinko et al., 2019), (Aaronson and Rall, 2020). As the QFT is not considered to be a NISQ subroutine, these results bring more hope to apply these algorithms in useful scenarios in the first quantum computers.

FIrst proposed MLQAE.

QIP2023 with a particular choice of probability, while before we had results "on average".

Theorem 5.6 (Variable Time Search (Ambainis, 2012a))**.** *Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be quantum algorithms that return true or false and run in unknown times $T_1, \ldots, T_n$, respectively. Suppose that each $\mathcal{A}_i$ outputs the correct answer with probability at least $2/3$. Then there exists a quantum algorithm with success probability at least $2/3$ that checks whether at least one of $\mathcal{A}_i$ returns true and runs in time*

$$\widetilde{O}\left(\sqrt{T_1^2 + \ldots + T_n^2}\right).$$

Definition 5.1 (Variabile-stopping-time algorithm (Ambainis, 2012a) (Chakraborty et al., 2022))**.** A quantum algorithms $\mathcal{A}$ acting on $\mathcal{H}$ that can be written as m quantum sub-algorithms $\mathcal{A} = \mathcal{A}_m\mathcal{A}_{m-1}\ldots\mathcal{A}_1$ is called a variable stopping time algorithm if $\mathcal{H} = \mathcal{H}_C \otimes \mathcal{H}_A$, where $\mathcal{H}_C \otimes_{i=1}^m \mathcal{H}_{C_i}$ with $\mathcal{H}_{C_i} = Span(|0\rangle, |1\rangle)$ and each unitary $\mathcal{A}_j$ acts on $\mathcal{H}_{C_i} \otimes \mathcal{H}_A$ controlled on the first $j-1$ qubits $|0\rangle^{\otimes j-1} \in \otimes_{i=1}^{j-1}\mathcal{H}_{C_i}$ being in the all zero state.

5.2.2 Amplitude amplification

5.2.3 Example: estimating average and variance of a function

Albeit the ideas treated in this post are somehow well-digested in the mind of many quantum algorithms developers, this example is very useful to get a practical understanding of amplitude estimation. Notably, much more precise and involved discussion around this topic can be found in chapter 8.

Suppose we have a random variable X described by a certain probability distribution over N different outcomes, and a function $f : \{0, \cdots N\} \to [0,1]$ defined over this distribution. How can we use quantum computers to evaluate some properties of f such as expected value and variance faster than classical computers?

Let's start by translating into the quantum realm these two mathematical objects. The probability distribution is (surprise surprise) represented in our quantum computer by a quantum state over $n = \lceil \log N \rceil$ qubits.

$$|\psi\rangle = \sum_{i=0}^{N-1} \sqrt{p_i}|i\rangle$$

where the probability of measuring the state $|i\rangle$ is p_i, for $p_i \in [0,1]$. Basically, each bases of the Hilbert space represent an outcome of the random variable.

The quantization of the function f is represented by a linear operator F acting on a new ancilla qubit (here on the right) as:

$$F : |i\rangle|0\rangle \to |i\rangle\left(\sqrt{1-f(i)}|0\rangle + \sqrt{f(i)}|1\rangle\right)$$

If we apply F with $|\psi\rangle$ as input state we get:

$$\sum_{i=0}^{N-1} \sqrt{1-f(i)}\sqrt{p_i}|i\rangle|0\rangle + \sum_{i=0}^{N-1} \sqrt{f(i)}\sqrt{p_i}|i\rangle|1\rangle$$

Observe that the probability of measuring $|1\rangle$ in the ancilla qubit is $\sum_{i=0}^{N-1} p_i f(i)$, which is $E[f(X)]$. By sampling the ancilla qubit we won't get any speedup compared to a classical randomized algorithm with oracle access to the function, but applying amplitude estimation (Brassard et al., 2002) to the ancilla qubit on the right, we can get an estimate of $E[F(X)]$ with precision ϵ, quadratically faster than a classical computer: in only $O(\frac{1}{\epsilon})$ queries to f.

Finally, observe that:

- if we chose $f(i) = \frac{i}{N-1}$ we are able to estimate $E[\frac{X}{N-1}]$ (which, since we know N gives us an estimate of the expected value $E[X]$)
- if we chose $f(i) = \frac{i^2}{(N-1)^2}$ instead, we can estimate $E[X^2]$ and using this along with the previous choice of f we can estimate the variance of X: $E[X^2] - E[X]^2$.

Exercise 5.2. Can you prove the previous two points?

We will see in Chapter 8 why in many real-world cases this is not the best way of estimating $\mathbb{E}(f(X))$.

5.3 Finding the minimum

We now want to show an algorithm that finds the minimum among N unsorted values $u_{j\in[N]}$. As the Grover's algorithm, we work in the oracle model, so we assume to have quantum access to a the vector u. Without loss of generality we can take $N = 2^n$, but if the length of our list is not a power of 2, we can just pad the rest of the elements in the list with zeroes. The statement of the theorem is the following.

Lemma 5.2 (Quantum minimum finding (Durr and Hoyer, 1996))**.** *Given quantum access to a vector $u \in [0,1]^N$ via the operation $|j\rangle|\bar{0}\rangle \to |j\rangle|u_j\rangle$ on $\mathcal{O}(\log N)$ qubits, where u_j is encoded to additive accuracy $\mathcal{O}(1/N)$. Then, we can find the minimum $u_{\min} = \min_{j\in[N]} u_j$ with success probability $1-\delta$ with $\mathcal{O}\left(\sqrt{N}\log\left(\frac{1}{\delta}\right)\right)$ queries and $\widetilde{\mathcal{O}}\left(\sqrt{N}\log\left(\frac{1}{\delta}\right)\right)$ quantum gates.*

Another formulation is the following:

Theorem 5.7 (Quantum Minimum Finding (Durr and Hoyer, 1996) formulation of (ambainis2019quantum))**.** *Let $a_1, \dots, a_n$ be integers, accessed by a procedure $\mathcal{P}$. There exists a quantum algorithm that finds $\min_{i=1}^{n}\{a_i\}$ with success probability at least $2/3$ using $O(\sqrt{n})$ applications of $\mathcal{P}$.*

This algorithm utilizes a subroutine called quantum exponential searching algorithm (QESA), which is again composed of amplitude amplification (Grover) and amplitude estimation. For simplicity, we assume those values are distinct. The general idea is marking indices of values below a chosen threshold, returning and picking one of them as the new threshold with equal probability. After some iterations, it is expected to yield the lowest value. Note that the step of marking lower values is taken as an oracle with gate complexity of $O(1)$. We first present to algorithm and keep the explanation for later.

Algorithm 1 Finding the minimum

```
Require: Quantum access to a N-dimensional vector $u$.
Ensure: Find the index of the minimum value of $u$ with probability at least 1/2.

Uniformly choose a threshold index $j$ from $0..N-1$.
repeat
    Initialize the memory as $\sum_i \frac{1}{\sqrt{N}}|i\rangle|j\rangle$. Mark every item $i$ for which $u_i < u_j$.
    Apply QESA.
    Measure the index register. Suppose the output is $i'$.
      Assign $j \leftarrow i'$ to be the new threshold index if $u_{i'} < u_j$.
until the queries to $U$ are more than $22.5\sqrt{N} + 1.4\log^2 N$.
Return $j$.
```

Figure 5.1: Finding the minimum

```
Algorithm 1 Quantum Exponential Searching Algorithm (QESA)
Require: Solution $x$, N-dimensional vector $u$.
Ensure: Find $i$ such that $u_i = x$ in $\mathcal{O}(\sqrt{N/t})$ expected time for $t \in [1, 3N/4]$
    being the unknown number of potential solutions. (If $t > 3N/4$, classical
    sampling works in constant expected time.)

1: Initialize $m = 1$ and set $\lambda = 6/5$ (or any $\lambda$ strictly between 1 and 4/3).
2: Choose $k$ uniformly random from the set of nonnegative integers smaller
   than $m$.
3: Apply $k$ iterations of Grover's algorithm starting from initial state $\sum_i \frac{1}{\sqrt{N}}|i\rangle$.
4: Measure the register to obtain some $i$.
5: If $u_i = x$, the problem is solved: exit.
6: Otherwise, set $m$ to $\min(\lambda m, \sqrt{N})$ and go back to step 2.
```

Figure 5.2: Quantum Exponential Searching Algorithm

In the basic Grover's algorithm with one solution, $m = CI\left(\frac{\pi}{4}\sqrt{N}\right)$ iterations give the true output with error probability at most $1/N$ for $N \gg 1$, where $CI(x)$ is the closest integer to x. The result can be generalized in the case of $t \ll N$ solutions that the error rate is reduced to at most t/N after exactly $m = CI\left(\frac{\pi - 2\arcsin\sqrt{t/N}}{4\arcsin\sqrt{t/N}}\right) = \left\lfloor \frac{\pi}{4\arcsin\sqrt{t/N}} \right\rfloor$ iterations (and thus oracle calls), with an upper bound $m \leq \frac{\pi}{4}\sqrt{\frac{N}{t}}$. However, a problem arises when we have to amplify amplitudes corresponding to values below the threshold. In practice, the number of candidates is unknown and varies for each threshold. QESA is a generalized algorithm to find a solution for unknown t with probability at least $1/4$ in $O(\sqrt{N/t})$, which is motivated by the following observation.

Proposition 5.1 (Quantum Exponential Searching Algorithm (Boyer et al., 1998))**.** *Let t be the (unknown) number of solutions and let θ be such that $\sin^2\theta = t/N$. Let m be an arbitrary positive integer. Let k be an integer chosen at random according to the uniform distribution between* 0 *and $m-1$. If we observe the register after applying k iterations of Grover's algorithm starting from the initial state $\sum_i \frac{1}{\sqrt{N}}|i\rangle$, the probability of obtaining a solution is exactly $P_m = \frac{1}{2} - \frac{\sin(4m\theta)}{4m\sin(2\theta)}$. In particular, $P_m \geq 1/4$ when $m \geq 1/\sin(2\theta)$.*

As QESA is expected to be done in $O(\sqrt{N/t})$ queries, one can deduce that the expected number of queries for the minimum-finding algorithm with success probability at least 1/2 is $O(\sqrt{N})$. Repeating the algorithm c times increases the success probability to $1 - 1/2^c$. In terms of quantum gates, the Grover part and the initialization part use $O(\sqrt{N})$ and $O(\log N)$ respectively.

5.4 Quantum linear algebra

A central tool for quantum algorithm in machine learning are the subroutines for performing quantum linear algebraic routines on a quantum computer. From the first work of (Harrow et al., 2009) (known in the literature as HHL algorithm) that proposed a quantum algorithm for matrix inversion, a lot of progress has been made in improving quantum algorithms for these problems. In this section, we briefly recall some of the results, and we conclude by citing the state-of-the art techniques for performing not only matrix inversion and matrix multiplication, but also for applying a certain class of functions to the singular values of a matrix. The HHL algorithm - under suitable assumptions - was able to create a quantum state proportional to the solution to a sparse linear system of equations $Ax = b$ using only a number of queries to the oracle that was polylogarithmic in the size of the matrix A. The assumption are the following: A must be symmetric and sparse (and we have quantum query access to A, and quantum sample access to $|b\rangle$, as we defined more precisely in 3). The runtime of the first quantum algorithm for this problem was $\widetilde{O}\left(s^2\kappa(A)^2/\epsilon\right)$, where s is the maximum value of non-zero entries per rows.

5.4.1 Singular value estimation

A notable result after HHL was the ability to perform a quantum version of the singular value decomposition. You can think of this result as a generalized phase estimation, i.e. a phase estimation that works on non-unitary matrices. It was first proposed in (Kerenidis and Prakash, 2017), and later improved in (Kerenidis and Prakash, 2020) and (Chakraborty et al., 2019). This idea is detailed in the following theorem.

Theorem 5.8 (Singular Value Estimation (Kerenidis and Prakash, 2020))**.** *Let $M \in \mathbb{R}^{n\times d}$ be a matrix with singular value decomposition $M = \sum_i \sigma_i u_i v_i^T$ for which we have quantum access. Let $\varepsilon > 0$ the precision parameter. There is an algorithm with running time $\widetilde{O}(\mu(M)/\varepsilon)$ that performs the mapping $\sum_i \alpha_i |v_i\rangle \to \sum_i \alpha_i |v_i\rangle |\tilde{\sigma_i}\rangle$, where $|\tilde{\sigma_i} - \sigma_i| \leq \varepsilon$ for all i with probability at least $1-1/poly(n)$.*

Recall that quantum access to a matrix is defined in theorem **??**, and the parameter μ is defined in definition 3.14. The relevance of theorem 5.8 for quantum machine learning is the following: if we are able to estimate the singular values of a matrix, then we can perform a conditional rotation controlled by these singular values and hence perform a variety of linear algebraic operations, including matrix inversion, matrix multiplication or projection onto a subspace. Based on this result, quantum linear algebra was done using the theorem stated below.

Theorem 5.9 (Old method for quantum linear algebra)**.** *Let $M := \sum_i \sigma_i u_i v_i^T \in \mathbb{R}^{d\times d}$ such that $\|M\|_2 = 1$, and a vector $x \in \mathbb{R}^d$ for which we have quantum access. There exist quantum algorithms that with probability at least $1-1/poly(d)$ returns*

- *a state* $|z\rangle$ *such that* $||z\rangle - |Mx\rangle| \leq \epsilon$ *in time* $\tilde{O}(\kappa^2(M)\mu(M)/\epsilon)$
- *a state* $|z\rangle$ *such that* $||z\rangle - |M^{-1}x\rangle| \leq \epsilon$ *in time* $\tilde{O}(\kappa^2(M)\mu(M)/\epsilon)$
- *a state* $|M^{+}_{\leq\theta,\delta}M_{\leq\theta,\delta}x\rangle$ *in time* $\tilde{O}(\frac{\mu(M)\|x\|}{\delta\theta\left\|M^{+}_{\leq\theta,\delta}M_{\leq\theta,\delta}x\right\|})$

One can also get estimates of the norms with multiplicative error η *by increasing the running time by a factor* $1/\eta$*.*

Recall that we denote as $V_{\geq\tau}$ the matrix $\sum_{i=0}^{\ell}\sigma_i u_i v_i^T$ where σ_ℓ is the smallest singular value which is greater than τ. For a matrix M and a vector x, we define as $M^{+}_{\leq\theta,\delta}M_{\leq\theta,\delta}x$ the projection of x onto the space spanned by the singular vectors of M whose corresponding singular values are smaller than θ, and some subset of singular vectors whose corresponding singular values are in the interval $[\theta,(1+\delta)\theta]$.

For a symmetric matrix $M \in \mathbb{R}^{d\times d}$ with spectral norm $\|M\| = 1$ for which we have quantum access, the running time of these algorithms depends on the condition number $\kappa(M)$ of the matrix, that can be replaced by $\kappa_\tau(M)$, a condition threshold where we keep only the singular values bigger than τ, and the parameter $\mu(M)$, a matrix dependent parameter defined in definition 3.14. The running time also depends logarithmically on the relative error ϵ of the final outcome state. Recall that these linear algebra procedures above can also be applied to any rectangular matrix $V \in \mathbb{R}^{n\times d}$ by considering instead the symmetric matrix

$$\overline{V} = \begin{pmatrix} 0 & V \\ V^T & 0 \end{pmatrix}.$$

5.4.2 Linear combination of unitaries

We continue our journey in quantum linear algebra by discussing the state-of-the-art technique beneath quantum linear algebra.

The research of quantum algorithms for machine learning has always used techniques developed in other areas of quantum algorithms. Among the many, we cite quantum algorithms for Hamiltonian simulation and quantum random walks. In fact, using quantum random walks, it is possible to decrease the dependence on the error parameter, from polynomial to polylog$(1/\epsilon)$ (Childs and Wiebe, 2012). Stemming from the research in Hamiltonian simulation (Berry et al., 2015b), (Low and Chuang, 2019), (Berry et al., 2015a), (Low and Chuang, 2017),(Subramanian et al., 2019), these techniques have been further optimized, pushing them to the limit of almost optimal time and query complexity. Significant progress in the direction of quantum algorithms for linear algebra was the so-called LCU, or linear combination of unitaries (Childs and Wiebe, 2012), which again was developed in the context of the Hamiltonian simulation problem.

Lemma 5.3 (Linear combination of unitaries (Childs et al., 2015))**.** *Let* $M = \sum_i \alpha_i U_i$ *be a linear combination of unitaries* U_i *with* $\alpha_i > 0$ *for all* i*. Let* V *be* *any operator that satisfies* $V|0^{\otimes m}\rangle := \frac{1}{\sqrt{\alpha}}\sum_i \sqrt{\alpha_i}|i\rangle$, *where* $\alpha := \sum_i \alpha_i$. *Then* $W := V^{\dagger}UV$ *satisfies*

$$W|0^{\otimes m}\rangle|\psi\rangle = \frac{1}{\alpha}|0^{\otimes m}\rangle M|\psi\rangle + |\Psi^{\perp}\rangle \tag{5.3}$$

for all states $|\psi\rangle$, *where* $U := \sum_i |i\rangle\langle i| \otimes U_i$ *and* $(|0^{\otimes m}\rangle\langle 0^{\otimes m}| \otimes I)|\Psi^{\perp}\rangle = 0$.

5.4.3 Singular value transformation

The research in quantum linear algebra culminated with the work of (Chakraborty et al., 2019), (Gilyén et al., 2019) with some improvements in (Chakraborty et al., 2022). We now briefly go through the machinery behind these results, as it will be used extensively this work. Before that, we recall the definition of block-encoding from Chapter 3.

Definition 5.2 (Block encoding of a matrix)**.** Let $A \in \mathbb{C}^{2^s \times 2^s}$. We say that a unitary $U \in \mathbb{C}^{(s+a)\times(s+a)}$ is a (α, a, ϵ) block encoding of A if:

$$\|A - \alpha(\langle 0|^a \otimes I)U(|0\rangle^a \otimes I)\| \leq \epsilon$$

We will see that having quantum access to a matrix $A \in \mathbb{C}^{2^w \times 2^w}$, as described in the setting of theorem 3.8, it is possible to implement a $(\mu(A), w+2, \text{polylog}(\epsilon))$ block-encoding of A [1]. Given matrix U which is a (α, a, δ) block encoding of A, and a matrix V which is a (β, b, ϵ) block encoding of B, it is simple to obtain a $(\alpha\beta, a+b, \alpha\epsilon + \beta\delta)$ block encoding of AB.

For practical purposes, having a block encoding of a matrix A, allows one to manipulate its spectra using polynomial approximation of analytic functions. In the following theorem, the notation $P_{\mathfrak{R}}(A)$ means that we apply the polynomial P to the singular values of the matrix A, i.e. $P_{\mathfrak{R}}(A) = \sum_i^r P(\sigma_i)u_i v_i^T$.

Theorem 5.10 (Polynomial eigenvalue transformation of arbitrary parity (Gilyén et al., 2019))**.** *Suppose that* U *is an* (α, a, ϵ)*-block encoding of the Hermitian matrix* A. *If* $\delta \geq 0$ *and* $P_{\mathfrak{R}} \in \mathbb{R}[x]$ *is a degree-d polynomial satisfying that:*

- *for all* $x \in [-1, 1]$, $|P_{\mathfrak{R}}(x)| \leq \frac{1}{2}$.

Then there is a quantum circuit $\tilde{U}$, *which is an* $(1, a+2, 4d\sqrt{\epsilon/\alpha}+\delta)$*-encoding of* $P_{\mathcal{R}}(A/\alpha)$, *and consists of* d *applications of* U *and* $U^{\dagger}$ *gates, a single application of controlled-U and* $O((a+1)d)$ *other one- and two-qubit gates. Moreover we can compute a description of such a circuit with a classical computer in time* $O(polyd, \log(1/\delta))$.

[1] This polylog(ϵ) in the block encoding is due to approximation error that one commits when creating quantum access to the classical data structures, i.e. is the approximation that derives from truncating a number $n \in \mathbb{R}$ (which rerepsent an entry of the matrix) up to a certain precision ϵ lemma 25 of [@CGJ18].

We will discuss in Section 5.4.4 how to use these subroutines for solving the linear system problems. You can find in Appendix @ref(#polyapprox-1overx) the polynomial approximation that was originally used in (Childs et al., 2015) to get what almost-tight gate complexity for this problem. This result has been improved in (Gribling et al., 2021), leading to a polynomial approximation allowing several orders of magnitude faster algorithms for linear system solving.

Given a (α, a, ϵ)-block encoding for a matrix A and a quantum state $|b\rangle$, we can obtain a good approximation of $A|b\rangle / \|Ab\|$ by first creating the state $|0^a, b\rangle$ and then applying the block encoding of A to it. Then, we can amplify the part of the subspace associated with the state $|0\rangle^{\otimes a} A|b\rangle$. Differently, one might use advanced amplification techniques and reach a similar result. This concept is detailed in the following lemma.

Lemma 5.4 (Applying a block-encoded matrix to a quantum state (Chakraborty et al., 2019))**.** *Fix any $\varepsilon \in (0, 1/2)$. Let $A \in \mathbb{C}^{N \times N}$ such that $\|A\| \leq 1$ and $|b\rangle$ a normalized vector in $\mathbb{C}^N$, such that $\|A|b\rangle\| \geq \gamma$. Suppose that $|b\rangle$ can be generated in complexity T_b and there is a (α, a, ϵ)-block encoding of A for some $\alpha \geq 1$, with $\epsilon \leq \varepsilon\gamma/2$, that can be implemented in cost T_A. Then there is a quantum algorithm with complexity*

$$O\left(min(\frac{\alpha(T_A + T_b)}{\gamma}, \frac{\alpha T_A \log(1/\epsilon) + T_B}{\gamma})\right)$$

that terminates with success probability at least $2/3$, and upon success generates the state $A|b\rangle / \|A|b\rangle\|$ to precision ε.

For sake of completeness, we briefly discuss how to prove the first upper bound. Generating $|b\rangle$ and applying the block encoding of A to it, we create a state that is (ϵ/α)-close to:

$$|0\rangle^{\otimes a}(\frac{1}{\alpha}A|b\rangle) + |0^{\perp}\rangle$$

From the hypothesis, we know that $\left\|\frac{1}{\alpha}A|b\rangle\right\| \geq \gamma/\alpha$. We can use $O(\alpha/\gamma)$ calls to amplitude amplification on the initial register being $|0\rangle^{\otimes a}$, to get $\frac{\epsilon}{\gamma}$ close to $|0\rangle^{\otimes a}\frac{A|b\rangle}{\|A\||b\rangle}$. The second upper bound is shown by other techniques based on amplitude amplification of singular values of block encoded matrices (i.e. (Chakraborty et al., 2019, lemma 47), (Low and Chuang, 2017, theorem 2,8)).

Regarding the usage of block-encodings for solving with a quantum computer a linear system of equations (i.e. multiplying a quantum state by the inverse of a matrix, and creating a state $|x\rangle$ proportional to $A^{-1}|b\rangle$) we can proceed in an analogous way. First, we need to create block encoding access to A^{-1}. Using the following lemma, (where they denoted with κ the condition number of A) we can implement negative powers of Hermitian matrices.

Lemma 5.5 (Implementing negative powers of Hermitian matrices (Chakraborty et al., 2019) lemma 9)**.** *Let $c \in (0, \infty), \kappa \geq 2$, and let*

A be an Hermitian matrix such that $I/\kappa \leq A \leq I$. Suppose that $\delta = o(\epsilon/(\kappa^{1+c}(1+c)\log^3 \frac{k^{1+c}}{\epsilon}))$ and U is an (α, a, δ)-block encoding of A that can be implemented using T_U gates. Then, for any ϵ, we can implement a unitary $\tilde{U}$ that is a $(2\kappa^c, a, \epsilon)$-block encoding of H^{-c} in cost:

$$O\left(\alpha\kappa(a+T_U)(1+c)\log^2(\frac{k^{1+c}}{\epsilon})\right)$$

Nevertheless, the algorithm that we can obtain by using the previous lemma has a quadratic dependence on κ. To decrease it to an algorithm linear in κ the authors used variable time amplitude amplifications(Ambainis, 2012a). Hence, we can restate the theorem 5.9, with the improved runtimes, as follows.

Theorem 5.11 (Quantum linear algebra (Chakraborty et al., 2019),(Gilyén et al., 2019))**.** *Let $M := \sum_i \sigma_i u_i v_i^T \in \mathbb{R}^{d\times d}$ such that $\|M\|_2 = 1$, and a vector $x \in \mathbb{R}^d$ for which we have quantum access in time T_χ. There exist quantum algorithms that with probability at least $1 - 1/poly(d)$ return*

- *a state $|z\rangle$ such that $||z\rangle - |Mx\rangle| \leq \epsilon$ in time $\tilde{O}(\kappa(M)(\mu(M) + T_\chi)\log(1/\epsilon))$*
- *a state $|z\rangle$ such that $||z\rangle - |M^{-1}x\rangle| \leq \epsilon$ in time $\tilde{O}(\kappa(M)(\mu(M) + T_\chi)\log(1/\epsilon))$*
- *a state $|M^+_{\leq\theta,\delta}M_{\leq\theta,\delta}x\rangle$ in time $\tilde{O}(T_\chi \frac{\mu(M)\|x\|}{\delta\theta\left\|M^+_{\leq\theta,\delta}M_{\leq\theta,\delta}x\right\|})$*

One can also get estimates of the norms with multiplicative error η by increasing the running time by a factor $1/\eta$.

This algorithm is leveraging Theorem 5.10 and the low-degree polynomial approximation of $1/x$ in Appendix E.2.

Another important advantage of the new methods is that it provides easy ways to manipulate sums or products of matrices.

Theorem 5.12 (Quantum linear algebra for product of matrices (Chakraborty et al., 2019),(Gilyén et al., 2019))**.** *Let $M_1, M_2 \in \mathbb{R}^{d\times d}$ such that $\|M_1\|_2 = \|M_2\|_2 = 1$, $M = M_1 M_2$, and a vector $x \in \mathbb{R}^d$ for which we have quantum access. There exist quantum algorithms that with probability at least $1 - 1/poly(d)$ return*

- *a state $|z\rangle$ such that $||z\rangle - |Mx\rangle| \leq \epsilon$ in time $\tilde{O}(\kappa(M)(\mu(M_1) + \mu(M_2))\log(1/\epsilon))$*
- *a state $|z\rangle$ such that $||z\rangle - |M^{-1}x\rangle| \leq \epsilon$ in time $\tilde{O}(\kappa(M)(\mu(M_1) + \mu(M_2))\log(1/\epsilon))$*
- *a state $|M^+_{\leq\theta,\delta}M_{\leq\theta,\delta}x\rangle$ in time $\tilde{O}(\frac{(\mu(M_1)+\mu(M_2))\|x\|}{\delta\theta\left\|M^+_{\leq\theta,\delta}M_{\leq\theta,\delta}x\right\|})$*

One can also get estimates of the norms with multiplicative error η by increasing the running time by a factor $1/\eta$.

More generally, applying a matrix M which is the product of k matrices, i.e. $M = M_1 \ldots M_k$ will result in a runtime of $\kappa(M)(\sum_i^k \mu(M_i))\log(1/\epsilon)$ factors in the runtime.

5.4.4 Matrix inversion after HHL

In this section we briefly recap the progress that we had in the last decade for solving the quantum linear system problem (QLSP). First, we stress the fact that the problem solved by this algorithm is fundamentally different than solving a linear system of equation on a classical computer (i.e. when we obtain $x = A^{-1}b$), as with a classical computer, once we finish the computation we obtain a classical description of the vector x. Instead, on a quantum computer we obtain a quantum state $|x\rangle$.

For a few years the techniques developed in (Kerenidis and Prakash, 2017) and (Kerenidis and Prakash, 2020) were the state of the art

After Ambainis used variable-time amplitude amplification to reduce the dependence on the condition number from quadratic do linear, Childs et al. (Childs et al., 2015) used the LCU framework, variable-time amplitude amplification, and the polynomial approximation of $1/x$ to solve the QLSP with a runtime dependece on the condition number of $O(\kappa(A)\log(\kappa))$, but also with an exponential improvement in the precision, i.e. now the error dependence appear as $O(\log(1/\epsilon))$ in the runtime. The authors used quantum walks to represent x as linear combination of polynomials $\sum_{i=1}^{n} \alpha_n T_n(x/d)$ where T_n is the Chebychev polynomial of the first kind, d is the sparsity of the matrix A, and α_i the coefficients of the polynomial expansions. For this, they had to give the first efficient polynomial approximation of $1/x$ (Lemma 17,18,19 of (Childs et al., 2015)) that you can find explained in greater deatails in the Appendix E.2. Interestingly, the QLSP has been also studied in the adiabatic setting, first in (Subaşı et al., 2019), and later improved in (An and Lin, 2022), and with eigenstate filtering (Lin and Tong, 2020) to an optimal scaling of $O(\kappa(A))$ (i.e. without a $O(\log(\kappa(A)))$ factor in the runtime, which to our knowledge remains unbeated by other gate based quantum algorithms) (Costa et al., 2021).

Last but not least, matrix inversion can be seen as the problem of implementing the singular value transformation of $x \mapsto 1/x$. For this, one needs to get a polynomial approximation of the function $1/x$. While this might seem a simple task, there are small complications. First, one usually does not consider the whole interval $[-1, 1]$. In practice, one excludes the subset of the domain where the function has singularities (i.e. for $1/x$ is around zero). It is preferable to pick a polynomial of a small degree, as the depth of the circuit depends linearly on the degree of the polynomial. In short with both LCU and QSVT we can use polynomial approximations to solve the QLSP problem. With LCU we have an additional multiplicative factor in the depth of the circuit compared to the QSVT framework, and some more ancillary qubits (Gribling et al., 2021).

A review of the progress made in the first 9 years after HHL can be found in (Dervovic et al., 2018) and (Gribling et al., 2021), from which we take this exercise. More recently, (Gribling et al., 2021) shows the latest techniques for the polynomial approximation of $1/x$ (and other functions), which improves the polynomial approximation of (Childs et al., 2015).

Exercise 5.3. In (Gribling et al., 2021) they say the following:

$$\|Ax - b\| \leq \|x - A^{-1}b\| \leq \kappa\|Ax - b\|$$

can you prove it?

5.5 Distances, inner products, norms, and quadratic forms

In this section, we report two lemmas that can be used to estimate the inner products, distances, and quadratic forms between vectors. The lemma 5.6 has been developed in the work (Kerenidis et al., 2019a). The lemma 5.8 and the other lemmas for inner product estimation in the query model come from (Hamoudi et al., 2020).

5.5.1 Inner products and quadratic forms with KP-trees

We can rephrase this theorem saying that we have quantum access to the matrices, but for simplicity we keep the original formulation. Also, in the remark following the proof of this lemma, we give the runtime of the same algorithm, but when we compute all the distances in superposition. Thanks to the union bound, we incour only in a logarithmic cost.

Lemma 5.6 (Distance and Inner Products Estimation (Kerenidis et al., 2019a))**.** *Assume for a matrix $V \in \mathbb{R}^{n\times d}$ and a matrix $C \in \mathbb{R}^{k\times d}$ that the following unitaries $|i\rangle|0\rangle \mapsto |i\rangle|v_i\rangle$, and $|j\rangle|0\rangle \mapsto |j\rangle|c_j\rangle$ can be performed in time T and the norms of the vectors are known. For any $\Delta > 0$ and $\epsilon > 0$, there exists a quantum algorithm that computes*

- $|i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle|\overline{d^2(v_i, c_j)}\rangle$ *where* $|\overline{d^2(v_i, c_j)} - d^2(v_i, c_j)| \leqslant \epsilon$ *w.p.* $\geq 1 - 2\Delta$
- $|i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle|\overline{(v_i, c_j)}\rangle$ *where* $|\overline{(v_i, c_j)} - (v_i, c_j)| \leqslant \epsilon$ *w.p.* $\geq 1 - 2\Delta$

in time $\widetilde{O}\left(\frac{\|v_i\|\|c_j\|T\log(1/\Delta)}{\epsilon}\right)$.

Proof. Let us start by describing a procedure to estimate the square ℓ_2 distance between the normalized vectors $|v_i\rangle$ and $|c_j\rangle$. We start with the initial state

$$|\phi_{ij}\rangle := |i\rangle|j\rangle\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle$$

Then, we query the state preparation oracle controlled on the third register to perform the mappings $|i\rangle|j\rangle|0\rangle|0\rangle \mapsto |i\rangle|j\rangle|0\rangle|v_i\rangle$ and $|i\rangle|j\rangle|1\rangle|0\rangle \mapsto |i\rangle|j\rangle|1\rangle|c_j\rangle$.

The state after this is given by,

$$\frac{1}{\sqrt{2}}\left(|i\rangle|j\rangle|0\rangle|v_i\rangle + |i\rangle|j\rangle|1\rangle|c_j\rangle\right)$$

Finally, we apply an Hadamard gate on the the third register to obtain,

$$|i\rangle|j\rangle\left(\frac{1}{2}|0\rangle\left(|v_i\rangle + |c_j\rangle\right) + \frac{1}{2}|1\rangle\left(|v_i\rangle - |c_j\rangle\right)\right)$$

The probability of obtaining $|1\rangle$ when the third register is measured is,

$$p_{ij} = \frac{1}{4}(2 - 2\langle v_i\rangle c_j) = \frac{1}{4}d^2(|v_i\rangle, |c_j\rangle) = \frac{1 - \langle v_i|c_j\rangle}{2}$$

which is proportional to the square distance between the two normalized vectors.

We can rewrite $|1\rangle\left(|v_i\rangle - |c_j\rangle\right)$ as $|y_{ij}, 1\rangle$ (by swapping the registers), and hence we have the final mapping

$$A : |i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle(\sqrt{p_{ij}}|y_{ij}, 1\rangle + \sqrt{1 - p_{ij}}|G_{ij}, 0\rangle) \tag{5.4}$$

where the probability p_{ij} is proportional to the square distance between the normalized vectors and G_{ij} is a garbage state. Note that the running time of A is $T_A = \widetilde{O}(T)$.

Now that we know how to apply the transformation described in Equation (5.4), we can use known techniques to conclude our subroutine to perform the distance estimation within additive error ϵ with high probability. The method uses two tools, amplitude estimation, and the median evaluation lemma D.2 from (Wiebe et al., 2018), which is a quantum version of the well-known powering-lemma C.1.

First, using amplitude estimation (theorem 5.4) with the unitary A defined in Equation (5.4), we can create a unitary operation that maps

$$\mathcal{U} : |i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle\left(\sqrt{\alpha}|\overline{p_{ij}}, G, 1\rangle + \sqrt{(1-\alpha)}|G', 0\rangle\right)$$

where G, G' are garbage registers, $|\overline{p_{ij}} - p_{ij}| \leq \epsilon$ and $\alpha \geq 8/\pi^2$. The unitary $\mathcal{U}$ requires P iterations of A with $P = O(1/\epsilon)$. Amplitude estimation thus takes time $T_{\mathcal{U}} = \widetilde{O}(T/\epsilon)$. We can now apply theorem D.2 for the unitary $\mathcal{U}$ to obtain a quantum state $|\Psi_{ij}\rangle$ such that,

$$\||\Psi_{ij}\rangle - |0\rangle^{\otimes L}|\overline{p_{ij}}, G\rangle\|_2 \leq \sqrt{2\Delta}$$

The running time of the procedure is $O(T_{\mathcal{U}} \ln(1/\Delta)) = \widetilde{O}(\frac{T}{\epsilon}\log(1/\Delta))$.

Note that we can easily multiply the value $\overline{p_{ij}}$ by 4 in order to have the estimator of the square distance of the normalized vectors or compute $1 - 2\overline{p_{ij}}$ for the normalized inner product. Last, the garbage state does not cause any problem in calculating the minimum in the next step, after which this step is uncomputed. The running time of the procedure is thus $O(T_{\mathcal{U}} \ln(1/\Delta)) = O(\frac{T}{\epsilon}\log(1/\Delta))$. The last step is to show how to estimate the square distance or the inner product of the unnormalized vectors. Since we know the norms of the vectors, we can simply multiply the estimator of the normalized inner product with the product of the two norms to get an estimate for the inner product of the unnormalized vectors and a similar calculation works for the distance. Note that the absolute error ϵ now becomes $\epsilon \left\lVert v_i \right\rVert \left\lVert c_j \right\rVert$ and hence if we want to have in the end an absolute error ϵ this will incur a factor of $\left\lVert v_i \right\rVert \left\lVert c_j \right\rVert$ in the running time. This concludes the proof of the lemma. □

Remark. Lemma 5.6 can be used to compute $\sum_{i,j=1}^{n,d} |i\rangle|\overline{\langle v_i, c_j\rangle}\rangle$ where every $\overline{\langle v_i, c_j\rangle}$ has error ϵ with an additional multiplicative cost of $O(\log(nd))$.

Exercise 5.4. Can you use the Union bound (i.e. Theorem C.1) to prove the following remark? The solution can be found in (Bellante and Zanero, 2022).

It is relatively simple to extend the previous algorithm to one that computes an estimate of a quadratic form. We will consider the case where we have quantum access to a matrix A and compute the quadratic forms v^TAv and $v^TA^{-1}v$. The extension to the case when we have two different vectors, i.e. v^TAu and $v^TA^{-1}u$ is trivial.

Lemma 5.7 (Estimation of quadratic forms)**.** *Assume to have quantum access to a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$ such that $\|A\| \leq 1$, and to a matrix $V \in \mathbb{R}^{n\times d}$. For $\epsilon > 0$, there is a quantum algorithm that performs the mapping $|i\rangle|0\rangle \mapsto |i\rangle|\overline{s_i}\rangle$, for $|s_i - \overline{s_i}| \leq \epsilon$, where s_i is either:*

- *$(|v_i\rangle, A|v_i\rangle)$ in time $O(\frac{\mu(A)}{\epsilon})$*
- *$(|v_i\rangle, A^{-1}|v_i\rangle)$ in time $O(\frac{\mu(A)\kappa(A)}{\epsilon})$*

The algorithm can return an estimate of $\overline{(v_i, Av_i)}$ such that $\overline{(v_i, Av_i)} - (v_i, Av_i) \leq \epsilon$ using quantum access to the norm of the rows of V by increasing the runtime by a factor of $\left\lVert v_i \right\rVert^2$.

Proof. We analyze first the case where we want to compute the quadratic form with A, and after the case for A^{-1}. Recall that the matrix A can be decomposed in an orthonormal basis $|u_i\rangle$. We can use theorem 5.11 to perform the following mapping:

$$|i\rangle|v_i\rangle|0\rangle = |i\rangle\frac{1}{N_i}\sum_j^n \alpha_{ij}|u_j\rangle|0\rangle \mapsto |i\rangle\frac{1}{N_i}\sum_i^n \Big(\lambda_i\alpha_{ij}|u_i,0\rangle + \sqrt{1-\gamma^2}|G,1\rangle\Big) = \tag{5.5}$$

$$|i\rangle\Big(\,\|Av_i\|\,|Av_i,0\rangle + \sqrt{1-\gamma^2}|G,1\rangle\Big) = |i\rangle|\psi_i\rangle, \tag{5.6}$$

where $N_i = \sqrt{\sum_j^n \alpha_{ij}^2}$. We define $|\phi_i\rangle = |v_i,0\rangle$. Using controlled operations, we can then create the state:

$$\frac{1}{2}|i\rangle\left(|0\rangle(|\phi_i\rangle + |\psi_i\rangle) + |1\rangle(|\phi_i\rangle - |\psi_i\rangle)\right) \tag{5.7}$$

It is simple to check that, for a given register $|i\rangle$, the probability of measuring 0 is:

$$p_i(0) = \frac{1 + \|Av_i\|\,\langle Av_i|v_i\rangle}{2}$$

We analyze the case where we want to compute the quadratic form for A^{-1}. For a $C = O(1/\kappa(A))$, we create instead the state:

$$|i\rangle\frac{1}{\sqrt{\sum_i^n \alpha_i^2}}\sum_i^n\left(\frac{C}{\lambda_i}\alpha_i|v_i,0\rangle + \sqrt{1-\gamma^2}|G,1\rangle\right) = |i\rangle|\psi_i\rangle \tag{5.8}$$

$$U_2|i\rangle|0\rangle \mapsto \frac{1}{2}|i\rangle\left(\sqrt{\alpha}|p_i(0),y_i,0\rangle + \sqrt{1-\alpha}|G_i,1\rangle\right) \tag{5.9}$$

and estimate $p_i(0)$ such that $|p_i(0) - \overline{p_i(0)}| < \epsilon$ for the case of $v_i^TAv_i$ and we choose a precision ϵ/C for the case of $v_i^TA^{-1}v_i$ to get the same accuracy. Amplitude estimation theorem, i.e. theorem @ref(thm:ampest_orig) fails with probability $\leq \frac{8}{\pi^2}$. The runtime of this procedure is given by combining the runtime of creating the state $|\psi_i\rangle$, amplitude estimation, and the median lemma. Since the error in the matrix multiplication step is negligible, and assuming quantum access to the vectors is polylogarithmic, the final runtime is $O(\log(1/\delta)\mu(A)\log(1/\epsilon_2)/\epsilon)$, with an additional factor $\kappa(A)$ for the case of the quadratic form of A^{-1}.

Note that if we want to estimate a quadratic form of two unnormalized vectors, we can just multiply this result by their norms. Note also that the absolute error ϵ now becomes relative w.r.t the norms, i.e. $\epsilon\left\|v_i\right\|^2$. If we want to obtain an absolute error ϵ', as in the case with normalized unit vectors, we have to run amplitude estimation with precision $\epsilon' = O(\epsilon/\left\|v_i\right\|^2)$. To conclude, this subroutine succeeds with probability $1-\gamma$ and requires time $O(\frac{\mu(A)\log(1/\gamma)\log(1/\epsilon_?)}{\epsilon_1})$, with an additional factor of $\kappa(A)$ if we were to consider the quadratic form for A^{-1}, and an additional factor of $\|v_i\|^2$ if we were to consider the non-normalized vectors v_i. This concludes the proof of the lemma. □

Note that this algorithm can be extended by using another index register to query for other vectors from another matrix W, for which we have quantum access. This extends the capabilities to estimating inner products in the form $|i\rangle|j\rangle|w_i^T Av_i\rangle$.

5.5.2 Inner product and l1-norm estimation with query access

Lemma 5.8 (Quantum state preparation and norm estimation)**.** *Let $\eta > 0$. Given a non-zero vector $u \in [0,1]^N$, with $\max_j u_j = 1$. Given quantum access to u via the operation $|j\rangle|\bar{0}\rangle \to |j\rangle|u_j\rangle$ on $O(\log N + \log 1/\eta)$ qubits, where u_j is encoded to additive accuracy η. Then:*

- *There exists a unitary operator that prepares the state $\frac{1}{\sqrt{N}}\sum_{j=1}^N |j\rangle \left(\sqrt{u_j}|0\rangle + \sqrt{1-u_j}|1\rangle\right)$ with two queries and number of gates $O(\log N + \log 1/\eta)$. Denote this unitary by U_χ.*
- *Let $\epsilon > 0$ such that $\eta \leq \epsilon/(2N)$ and $\delta \in (0,1)$. There exists a quantum algorithm that provides an estimate Γ_u of the ℓ_1-norm $\|u\|_1$ such that $|\|u\|_1 - \Gamma_u| \leq \epsilon\|u\|_1$, with probability at least $1-\delta$. The algorithm requires $O(\frac{\sqrt{N}}{\epsilon}\log(1/\delta))$ queries and $\widetilde{O}(\frac{\sqrt{N}}{\epsilon}\log(1/\delta))$ gates.*
- *Let $\xi \in (0,1]$ such that $\eta \leq \xi/4N$ and $\delta \in (0,1)$. An approximation $|\tilde{p}\rangle = \sum_{j=1}^N \sqrt{\tilde{p}_j}|j\rangle$ to the state $|u\rangle := \sum_{j=1}^N \sqrt{\frac{u_j}{\|u\|_1}}|j\rangle$ can be prepared with probability $1-\delta$, using $O(\sqrt{N}\log(1/\delta))$ calls to the unitary of (i) and $\widetilde{O}(\sqrt{N}\log(1/\xi)\log(1/\delta))$ gates. The approximation in ℓ_1-norm of the probabilities is $\left\|\tilde{p} - \frac{u}{\|u\|_1}\right\|_1 \leq \xi$.*

Proof. For the first point, prepare a uniform superposition of all $|j\rangle$ with $O(\log N)$ Hadamard gates. With the quantum query access, perform

$$\frac{1}{\sqrt{N}}\sum_{j=1}^N |j\rangle|\bar{0}\rangle \to \frac{1}{\sqrt{N}}\sum_{j=1}^N |j\rangle|u_j\rangle|0\rangle$$

$$\to \frac{1}{\sqrt{N}}\sum_{j=1}^N |j\rangle|u_j\rangle\left(\sqrt{u_j}|0\rangle + \sqrt{1-u_j}|1\rangle\right).$$

The steps consist of an oracle query and a controlled rotation. The rotation is well-defined as $u_j \leq 1$ and costs $O(\log 1/\eta)$ gates. Then uncompute the data register $|u_j\rangle$ with another oracle query.

For part 2, define a unitary $\mathcal{U} = U_\chi \left(\mathbb{1} - 2|\bar{0}\rangle\langle\bar{0}|\right) \left(U_\chi\right)^\dagger$, with U_χ from part 1, and here $\mathbb{1}$ is the identiy matrix. Define another unitary by $\mathcal{V} = \mathbb{1} - 2\mathbb{1} \otimes |0\rangle\langle 0|$. Using K applications of $\mathcal{U}$ and $\mathcal{V}$, amplitude estimation 5.4 allows to provide an estimate $\tilde{a}$ of the quantity $a = \frac{\|u\|_1}{N}$ to accuracy $|\tilde{a} - a| \leq 2\pi\frac{\sqrt{a(1-a)}}{K} + \frac{\pi^2}{K^2}$. Following the idea in (van Apeldoorn et al., 2020) (of dividing the elements that we are summing using amplitude estimation by their maximum, that we can find using the finding the minimum subroutine), take $K > \frac{6\pi}{\epsilon}\sqrt{N}$, which obtains

$$\begin{aligned}|\tilde{a} - a| \leq & \qquad \frac{\pi}{K}\left(2\sqrt{a} + \frac{\pi}{K}\right) < \frac{\epsilon}{6}\sqrt{\frac{1}{N}}\left(2\sqrt{a} + \frac{\epsilon}{6}\sqrt{\frac{1}{N}}\right) \\ \leq & \qquad \frac{\epsilon}{6}\sqrt{\frac{1}{N}}\left(3\sqrt{a}\right) = \frac{\epsilon\sqrt{\|u\|_1}}{2N}. \end{aligned} \tag{5.10}$$

Since $\|u\|_1 \geq 1$ by assumption, we have $|\tilde{a} - a| \leq \frac{\epsilon\|u\|_1}{2N}$.

Also, there is an inaccuracy arising from the additive error η of each u_j. As it was assumed that $\eta \leq \epsilon/(2N)$, the overall multiplicative error ϵ is obtained for the estimation. For performing a single run of amplitude estimation with K steps, we require $O(K) = O(\frac{\sqrt{N}}{\epsilon})$ queries to the oracles and $O\left(\frac{\sqrt{N}}{\epsilon}\left(\log N + \log(N/\epsilon)\right)\right)$ gates.

For part 3, rewrite the state from part 1 as

$$\sqrt{\frac{\|u\|_1}{N}}\sum_{j=1}^{N}\sqrt{\frac{u_j}{\|u\|_1}}|j\rangle|0\rangle + \sqrt{1 - \frac{\|u\|_1}{N}}\sum_{j=1}^{N}\sqrt{\frac{1-u_j}{N - \|u\|_1}}|j\rangle|1\rangle.$$

Now amplify the $|0\rangle$ part using Amplitude Amplification (Brassard et al., 2002) via the exponential search technique without knowledge of the normalization, to prepare $\sum_{j=1}^{N}|j\rangle\sqrt{\frac{u_j}{\|u\|_1}}$ with success probability $1-\delta$. The amplification requires $O(\sqrt{\frac{N}{\|u\|_1}}\log(1/\delta)) = O(\sqrt{N}\log(1/\delta))$ calls to the unitary of part 1, as $\|u\|_1 \geq 1$. The gate complexity derives from the gate complexity of part 1.

Denote the η-additive approximation to u_j by $\tilde{u}_j$, and evaluate the ℓ_1-distance of the probabilities. First, $|\|u\|_1 - \|\tilde{u}\|_1| \leq N\eta$. One obtains $\left\|\tilde{p} - \frac{u}{\|u\|_1}\right\|_1 = \left\|\frac{\tilde{u}}{\|\tilde{u}\|_1} - \frac{u}{\|u\|_1}\right\|_1 \leq \sum_j\left|\frac{\tilde{u}_j}{\|\tilde{u}\|_1} - \frac{u_j}{\|\tilde{u}\|_1}\right| + \sum_j\left|\frac{u_j}{\|\tilde{u}\|_1} - \frac{u_j}{\|u\|_1}\right| \leq \frac{N\eta}{\|\tilde{u}\|_1} + \frac{N\eta}{\|\tilde{u}\|_1}$.

We also obtain

$$\frac{1}{\|\tilde{u}\|_1} \leq \frac{1}{\|u\|_1 - N\eta} \leq \frac{2}{\|u\|_1}$$

for $\eta \leq \|u\|_1/2N$.\$ Since $\eta \leq \|u\|_1\xi/(4N)$, the distance is $\left\|\tilde{p} - \frac{u}{\|u\|_1}\right\|_1 \leq \xi$ as desired. □

Lemma 5.9 (Quantum inner product estimation with relative accuracy)**.** *Let* $\epsilon, \delta \in (0,1)$. *Given quantum access to two vectors* $u, v \in [0,1]^N$, *where* u_j *and*

v_j are encoded to additive accuracy $\eta = O1/N$. Then, an estimate I for the inner product can be provided such that $|I - u \cdot v/\|u\|_1| \leq \epsilon\ u \cdot v/\|u\|_1$ with success probability $1-\delta$. This estimate is obtained with $O\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ queries and $\widetilde{O}\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ quantum gates.

Proof. Via lemma 5.2, determine $u_{\max}$ with success probability $1-\delta$ with $O\left(\sqrt{N}\log\frac{1}{\delta}\right)$ queries and $\widetilde{O}\left(\sqrt{N}\log\left(\frac{1}{\delta}\right)\right)$ quantum gates.

Apply lemma 5.8 with the vector $\frac{u}{u_{\max}}$ to obtain an estimate Γ_u of the norm $\left\|\frac{u}{u_{\max}}\right\|_1$ to relative accuracy $\epsilon_u = \epsilon/2$ with success probability $1-\delta$.

This estimation takes $O(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right))$ queries and $\widetilde{O}(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right))$ quantum gates.

Define the vector z with $z_j = u_j v_j$. Via lemma 5.2, determine $z_{\max}$ with success probability $1-\delta$ with $O\left(\sqrt{N}\log\frac{1}{\delta}\right)$ queries and $\widetilde{O}\left(\sqrt{N}\log\left(\frac{1}{\delta}\right)\right)$ quantum gates. If $z_{\max} = 0$ up to numerical accuracy, the estimate is $I = 0$ and we are done. Otherwise, apply lemma 5.8 with the vector $\frac{z}{z_{\max}}$ to obtain an estimate Γ_z of the norm $\left\|\frac{z}{z_{\max}}\right\|_1$ to relative accuracy $\epsilon_z = \epsilon/2$ with success probability $1-\delta$. This estimation takes $O\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ queries and $\widetilde{O}\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ quantum gates.

With lemma D.1, which gives a nice bound for the ratio between two relative errors, we have

$$\left|\frac{\Gamma_z}{\Gamma_u} - \frac{u_{\max}}{z_{\max}}\frac{u \cdot v}{\|u\|_1}\right| \leq \frac{u_{\max}}{z_{\max}}\frac{u \cdot v}{\|u\|_1}\frac{\epsilon_z + \epsilon_u}{(1-\epsilon_u)} \tag{5.11}$$

$$\leq 2\epsilon\frac{u_{\max}}{z_{\max}}\frac{u \cdot v}{\|u\|_1}, \tag{5.12}$$

since $\epsilon_u < 1/2$. Set

$$I = \frac{z_{\max}}{u_{\max}}\frac{\Gamma_z}{\Gamma_u}, \tag{5.13}$$

and we have $|I - u \cdot v/\|u\|_1| \leq 2\epsilon\ u \cdot v/\|u\|_1$. The total success probability of the four probabilistic steps is at least $1-4\delta$ via a union bound (theorem C.1). Choosing $\epsilon \to \epsilon/2$ and $\delta \to \delta/4$ leads to the result. □

Lemma 5.10 (Quantum inner product estimation with additive accuracy)**.** *Let $\epsilon, \delta \in (0,1)$. Given quantum access to a non-zero vector $u \in [0,1]^N$ and another vector $v \in [-1,1]^N$, where u_j and v_j are encoded to additive accuracy $\eta = O1/N$. Then, an estimate I for the inner product can be provided such that $|I - u \cdot v/\|u\|_1| \leq \epsilon$ with success probability $1-\delta$. This estimate is obtained with $O\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ queries and $\widetilde{O}\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ quantum gates.*

Note that as a byproduct, the value $u_{\max}$ and an estimate of $\|u/u_{\max}\|_1$ with relative accuracy ϵ can be provided with probability at least $1-\delta$.

Proof. Via lemma 5.2, determine $\|u\|_{\max}$ with success probability $1-\delta$ with $O\left(\sqrt{N}\log\frac{1}{\delta}\right)$ queries and $\widetilde{O}\left(\sqrt{N}\log\left(\frac{1}{\eta}\right)\log\left(\frac{1}{\delta}\right)\right)$ quantum gates. Apply lemma 5.8 with the vector $\frac{u}{u_{\max}}$ to obtain an estimate Γ_u of the norm $\left\|\frac{u}{u_{\max}}\right\|_1$ to relative accuracy $\epsilon_u = \epsilon/2$ with success probability $1-\delta$.

This estimation takes $O\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ queries and $\widetilde{O}\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ quantum gates.

Similarly, consider the vector z with elements $z_j := u_j\left(v_j+3\right) \in [0,4]$. Determine $\|z\|_{\max}$ with success probability $1-\delta$ with $O\left(\sqrt{N}\log\frac{1}{\delta}\right)$ queries and $\widetilde{O}\left(\sqrt{N}\log\left(\frac{1}{\delta}\right)\right)$ quantum gates. Apply lemma 5.8 with the vector $z/z_{\max}$ to obtain an estimate Γ_z of the norm $\|z/z_{\max}\|_1$ to relative accuracy $\epsilon_z = \epsilon/2$ with success probability $1-\delta$.

This estimation takes $O\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ queries and $\widetilde{O}\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$.

It takes some steps to see that the exact quantities are related via

$$\frac{u\cdot v}{\|u\|_1} = \frac{z_{\max}}{u_{\max}}\frac{\|\frac{z}{z_{\max}}\|_1}{\|\frac{u}{u_{\max}}\|_1} - 3. \tag{5.14}$$

Considering the estimator $I = \frac{z_{\max}}{u_{\max}}\frac{\Gamma_z}{\Gamma_u} - 3$, from lemma D.1, we have

$$\begin{aligned}\left|I - \frac{u\cdot v}{\|u\|_1}\right| = & \quad \frac{z_{\max}}{u_{\max}}\left|\frac{\Gamma_z}{\Gamma_u} - \frac{\|\frac{z}{z_{\max}}\|_1}{\|\frac{u}{u_{\max}}\|_1}\right| \\ \leq & \quad \frac{\epsilon_u+\epsilon_z}{1-\epsilon_u}\frac{\|z\|_1}{\|u\|_1} \leq 8\epsilon.\end{aligned} \tag{5.15}$$

In the last steps we have used the observation that

$$\frac{\|z\|_1}{\|u\|_1} \equiv \frac{\sum_j u_j(v_j+3)}{\sum_j u_j} \leq \frac{4\sum_j u_j}{\sum_j u_j} = 4, \tag{5.16}$$

and $\epsilon_u < 1/2$. All steps together take $O\left(\frac{\sqrt{N}}{\epsilon}\log\frac{1}{\delta}\right)$ queries and $\widetilde{O}\left(\frac{\sqrt{N}}{\epsilon}\log\left(\frac{1}{\delta}\right)\right)$ gates. The total success probability of all the probabilistic steps is at least $1-4\delta$ via a union bound. Choosing $\epsilon \to \epsilon/8$ and $\delta \to \delta/4$ leads to the result. □

5.6 Hamiltonian simulation

These notes based on Childs' Lecture notes, i.e. (Andrew, 2017), Section 5

5.6.1 Introduction to Hamiltonians

The only way possible to start a chapter on Hamiltonian simulation would be to start from the work of Feynman, who had the first intuition on the power of quantum mechanics for simulating physics with computers. We know that the Hamiltonian dynamics of a closed quantum system, weather its evolution changes with time or not is given by the Schrödinger equation:

$$\frac{d}{dt}|\psi(t)\rangle = H(t)|\psi(t)\rangle$$

Given the initial conditions of the system (i.e. $|\psi(0)\rangle$) is it possible to know the state of the system at time $t : |\psi(t)\rangle = e^{-i(H_1 t/m)}|\psi(0)\rangle$.

As you can imagine, classical computers are supposed to struggle to simulate the process that builds $|\psi(t)\rangle$, since this equation describes the dynamics of any quantum system, and we don't think classical computers can simulate that efficiently for any general Hamiltonian H. But we understood that quantum computers can help to simulate the dynamics of another quantum system.

Why we might want to do that?

An example would be the following. Imagine you are a quantum machine learning scientist, and you have just found a new mapping between an optimization problem and an Hamiltonian dynamics, and you want to use quantum computer to perform the optimization (Otterbach et al., 2017). You expect a quantum computers to run the Hamiltonian simulation for you, and then sample useful information from the resulting quantum sate. This result might be fed again into your classical algorithm to perform ML related task, in a virtuous cycle of hybrid quantum-classical computation, which we will discuss more in another chapter.

Another example, perhaps more akin to the original scope of Hamiltonian simulation, is related to quantum chemistry. Imagine that are a chemist, and you have developed a hypothesis for the Hamiltonian dynamics of a chemical compound. Now you want to run some experiments to see if the formula behaves according to the experiments (and you cannot simulate numerically the experiment). Or maybe you are testing the properties of complex compounds you don't want to synthesize.

We can formulate the problem of Hamiltonian simulation in this way:

Definition 5.3 (Hamiltonian simulation problem)**.** Given a state $|\psi(0)\rangle$ and an Hamiltonian H, obtain a state $\overline{|\psi(t)\rangle}$ such that $\|\overline{|\psi(t)\rangle} - e^{-iHt}|\psi(0)\rangle\| \leq \epsilon$ in some norm.

(Note that also this problem can be reformulated in the context of density matrices, where usually the trace norm is used as a distance between quantum states).

This leads us to the definition of efficiently simulable Hamiltonian:

Definition 5.4 (Hamiltonian simulation)**.** Given a state $|\psi(0)\rangle$ and an Hamiltonian H acting on n qubits, we say H can be efficiently simulated if, $\forall t \geq 0, \forall \epsilon \geq$

0 there is a quantum circuit U such that $\|U - e^{-iHt}\| < \epsilon$ using a number of gates that is polynomial in $n, t, 1/\epsilon$.

Part II

Quantum Machine Learning

Chapter 6

Quantum perceptron

Contributors: Armando Bellante, Samantha Buck

Figure 6.1: This section is a work in progress

The following chapter is an investigation into the quantum version of the classical perceptron algorithm, and is based on the previous works of (Kapoor et al., 2016)

The chapter is organized as follows: we first introduce the fundamentals of the classical version of the perceptron, then we present the online quantum perceptron algorithm and the version space quantum perceptron algorithm.

6.1 Classical perceptron

The perceptron is a machine learning algorithm that can be thought of as the most basic fundamental building block of more complex artificial neural networks (ANNs), or alternatively as a very simple form of neural network in and of itself.

The perceptron is a linear classifier used for binary predictions: its goal is to classify incoming data in one of two given categories. Like in any supervised learning tasks, the classifier is trained using a dataset of couples data points-labels and its goal is to generalize to previously unseen data points.

Unlike more complex learners, the textbook perceptron can only deal with *linearly separable* datasets. A dataset is said to be linearly separable if there exists at least one hyperplane that can successfully separate the elements of the dataset into distinct groups.

Two sets, A and B, in an $n-$space are linearly separable only if there exists an $(n-1)$ hyperplane that shatters the two sets. In a 2-dimension space, the hyperplane is a line. In a 3D space it is a plane, and so on. (Note that in 2D, for instance, it could not be a curve! The linearity of this classifier make it so that you can only shatter the sets using hyperplanes. One could overcome this limitation by introducing kernels in the perceptron formulation or by projecting the data in a space, via “feature engineering”, so that they are linearly separable in the new space.)

Let us introduce the concept of linear separability using proper mathematical formalism.

Definition 6.1 (Linear Separability)**.** Two sets A and B of points in an n-dimensional space are called absolutely linearly separable if $n+1$ real numbers w_1, ..., w_n, b exist, such that every point $(x_1$, x_2, ..., $x_n)$ in A satisfies :

$$\sum_{i=1}^{n} w_i x_i > b \tag{6.1}$$

every point B satisfies :

$$\sum_{i=1}^{n} w_i x_i < b \tag{6.2}$$

The general equation of the separation hyperplane would then be $\sum_{i=1}^{n} w_i x_i + b = 0$. The numbers w_i and b are often referred to as weights and bias respectively. Without the bias term, the hyperplane that w defines would always have to go through the origin. It is common practise to absorb b into the vector w by adding one additional constant dimension to the data points and the weigths vector. In doing so, x becomes $[\begin{smallmatrix} x \\ 1 \end{smallmatrix}]$, and w becomes $[\begin{smallmatrix} w \\ b \end{smallmatrix}]$.

We can then verify that their inner product will yield,

$$\begin{bmatrix} x \\ 1 \end{bmatrix}^\top \begin{bmatrix} w \\ b \end{bmatrix} = w^\top x + b \tag{6.3}$$

From now on, we will write x and w assuming the bias is included in this way.

We will soon focus on how the perceptron learns the weights w_i and b. However, assume for a while that the perceptron knows these numbers (i.e., it has been trained) and only needs to use them to classify the incoming data points in one of the two classes.

Given a data point x, its coordinates x_i are weighted in a sum $\sum w_i x_i$ and this result is passed to one "neuron" that has an activation function g. The output of this activation function $y = g(\sum w_i x_i)$ is a binary value that represents one of the two possible classes. In general, the activation function can take many forms (ex: the sigmoid function, hyperbolic, step function, etc.) and return whichever couple of values (e.g., {0,1}, {-1,1}). For the remainder of this chapter, we will assume the sign function is used.

In other words, given a data point x and the weights w, the perceptron classifies the point in the following way:

$$y\left(x\right) = \text{sign}\left(w^\top x\right) \tag{6.4}$$

where $y(x) = -1$ means that the sample belongs to one class and $y(x) = +1$ means it belongs to the other, by a matter of convention.

The hyperplane identified by the weigths w is also called decision boundary as, using the sign activation function, the position of the data points w.r.t. it determines the output class.

6.1.1 Training the perceptron

In this book, we focus only on the *online training* algorithm, as this is handy for our quantum algorithms. We invite the interested reader to read more about the *batch training.*

The perceptron, like we *should* do, learns from its own errors but it also need somewhere to start from. Its training process happens in the following way. We initiate the perceptron with some random weights, feed it with many training couples data point-label, and let it shot its guess for each point. If the guess is correct, nothing happens. If the guess is wrong, the weights need to be updated.

Using the sign function, and considering a couple data point-label ($x^{(i)}$, $y^{(i)}$) from the training set, the perceptron classifies the point correctly if $y^{(i)}\left(w^\top x^{(i)}\right) > 0$ (this is because the class $y^{(i)}$ has been encoded using values in $\{-1, +1\}$).

If a classification error is detected, the weights get modified by an amount that is proportional to the input $x^{(i)}$. This process continues for a certain number of iterations, known as "epochs." In the online learning regime, which is the one we investigate in this chapter, the update to the weight and bias vectors occurs for each and every misclassified data point iteratively throughout every epoch. During an epoch, each training data point is considered once. The training ends when there are no misclassified points in one epoch. We remind the reader that the training goal is to determine the weights that produce a linear decision boundary that correctly classifies the predictions.

The following pseudocode highlights the structure of the perceptron algorithm and identifies at which step in the processing schedule the error calculation and subsequent parametric updates occur.

The update is computed by adding the amount $\Delta w_i = \eta \cdot error \cdot x^{(i)}$ to the weights w, where η is a learning rate end $error = w^\top x - y^{(i)}$. Notice that if $error$ and $x_j^{(i)}$ have the same sign, the increment of w_j is positive (the strength increases) and otherwise decreases.

To sum up in an overall picture, the perceptron iteratively updates the linear boundary function (i.e., the hyperplane equation) that acts as the classifying condition between the two classes of the data, until no errors are detected and the algorithm converges. The training happens by minimizing the error on the training data set provided to the algorithm. Ultimately the identification of the weights allows for the distinction of data points belonging on one side of the boundary or the other.

Of course, if the data were not linearly separable, the training algorithm would run forever. The online training requires the existence of a margin between the two classes.

Definition 6.2 (Margin of a dataset)**.** Let $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$ a dataset of n couples data points-labels, with $x^{(i)} \in \mathbb{R}^m$ and $y^{(i)} \in \{-1, +1\}$ for all $i \in [n]$. We define the margin of this dataset as:

$$\gamma = \max_{q \in \mathbb{R}^m} \min_{i \in [n]} \frac{y^{(i)} q^\top x^{(i)}}{\|q\| \|x^{(i)}\|}. \tag{6.5}$$

It naturally follows that a dataset is linearly separable if and only if its margin is different from 0.

6.2 Online quantum perceptron

General Idea: Uses Grover's search to more efficiently search for misclassified data points in the data set. This allows the perceptron algorithm to more quickly converge on a correct classifying hyperplane.

Where the quantum version of the online perceptron differs from that of its classical analogue is in *how* the data points are accessed for use within each individual epoch and in the number of times the perceptron needs to be used to classify the data points. In the classical version of the online perceptron, the training points are fed into the classification algorithm successively one by one, and a weight update is performed every time a point is misclassified. Imagine we have n data points and only the last ones of the epoch get misclassified: this requires $O(n)$ evaluations of the classification function before the weights can be updated. The quantum version of the online perceptron deviates away from this mechanism of accessing the data points within an epoch in a ”one at a time” fashion and lowers the number of time we need to call the classification function. The idea is to access the data points in superposition in a quantum state and apply the classification function to this state, so to apply the classification function linearly to all the data points at once, searching for the misclassified one.

Assume we have a dataset of $\{x^{(1)}, \dots, x^{(N)}\}$ vectors and $\{y^{(1)}, \dots, y^{(N)}\}$ labels. Without loss of generality, we assume that the training set consists of *unit* vectors and one-bit labels. Furthermore, we assume that the $\{x^{(1)}, \dots, x^{(N)}\}$ vectors can be classically represented using B bits. Then, with $|x^{(1)}\rangle, \dots, |x^{(n)}\rangle$ we denote the $(B+1)$-bit representations of the data vectors, followed by the label bit. Note that each state $|x^{(j)}\rangle$ corresponds to a state of the computational basis of the $(B+1)$-dimensional Hilbert space.

Next, in order to construct our desired online quantum perceptron, we will need to have a mechanism with which to access the training data. We assume that the data is accessible via the following oracle U.

$$U|j\rangle|0\rangle = |j\rangle|x^{(j)}\rangle \tag{6.6}$$

$$U^\dagger|j\rangle|x^{(j)}\rangle = |j\rangle|0\rangle \tag{6.7}$$

As we discussed in previous chapters, because U and $U^\dagger$ are linear operators, we have that $U\frac{1}{\sqrt{n}}\sum_{j=1}^{n}|j\rangle|0\rangle = \frac{1}{\sqrt{n}}\sum_{j=1}^{n}|j\rangle|x^{(j)}\rangle$. A quantum computer can therefore access each training vector simultaneously using a single operation, while only requiring enough memory to store one of the $|x^{(j)}\rangle$.

Next, we need a mechanism to test if a given weight configuration w, correctly classifies a data point. First of all, we need to build a quantum circuit that implements the Boolean function $f : \left(w, x^{(j)}, y^{(j)}\right) \mapsto \{0,1\}$, where $f\left(w, x^{(j)}, y^{(j)}\right)$ is 1 if and only if the perceptron with weights w misclassifies the training sample $(x^{(j)}, y^{(j)})$. This circuit needs to adapt to the current weight configuration w, which can be either “hardcoded” in the circuit (and the circuit would need to be updated everytime the vector changes) or be given as an input via use of extra qubits. Given access to such circuit, then we can define an operator, $\mathcal{F}_w$:

$$\mathcal{F}_w|x^{(j)}\rangle = (-1)^{f(w,x^{(j)},y^{(j)})}|x^{(j)}\rangle. \tag{6.8}$$

$\mathcal{F}_w$ is easily implemented on a quantum computer using a multiply controlled phase gate and a quantum implementation of the perceptron classification algorithm f_w.

Now, we can use the unitary $\mathcal{F}_w$ as an oracle for Grover's algorithm (see Section 5.2). This allows for a quadratic reduction in the number of times that the training vectors need to be accessed by the classification function.

The following pseudocode sums up the online quantum perceptron algorithm.

This algorithm gives birth to the following theorem, which formalizes the quadratic speedup.

Theorem 6.1 (Online quantum perceptron (Kapoor et al., 2016)). *Given a training set that consists of unit vectors $\{x^{(1)}, \dots, x^{(n)}\}$ and labels $\{y^{(1)}, \dots, y^{(n)}\}$, with margin γ, the number of applications of f needed to learn the weights w such that $P\left(\exists j : f_w\left(x^{(j)}, y^{(j)}\right) = 1\right) \leq \epsilon$ using a quantum computer is n_{quant} where*

$$\Omega(\sqrt{n}) \ni n_{quant} \in O\left(\frac{\sqrt{n}}{\gamma^2}\log\left(\frac{1}{\epsilon\gamma^2}\right)\right) \tag{6.9}$$

whereas the number of queries to f needed in the classical setting, n_{class} , where the training vectors are found by sampling uniformly from the training data is bounded by

$$\Omega(n) \ni n_{class} \in O\left(\frac{n}{\gamma^2}\log\left(\frac{1}{\epsilon\gamma^2}\right)\right). \tag{6.10}$$

Note that if the training data is supplied as a stream (as in the standard online model), then the upper bound for the classical model changes to $n_{\text{class}} \in O\left(n/\gamma^2\right)$ because all n training vectors can be deterministically checked to see if they are correctly classified by the perceptron. A quantum advantage is therefore obtained if $n \gg \log^2\left(1/\epsilon\gamma^2\right)$. (see (Kapoor et al., 2016) for a better explaination.)

Theorem 6.1 can easily be proved using Grover's theorem (with exponential search, since we do not know the exact number of answers to the search problem. i.e., multiple data points can be misclassified) and two lemmas.

Lemma 6.1. *Given only the ability to sample uniformly from the training vectors, the number of queries to f_w needed to find a training vector that the curnent perceptron model fails to classify correctly, or conclude that no such example exists, with probability $1 - \epsilon\gamma^2$ is at most $O\left(n\log\left(1/\epsilon\gamma^2\right)\right)$.*

Lemma 6.2. *Assuming that the training vectors $\{x^{(1)}, \dots, x^{(n)}\}$ are unit vectors and that they are drawn from two classes separated by a maryin of γ in feature space, the online quantum perceptron algorithm will either update the perceptron weights, or conclude that the current model provides a separating hyperplane between the two classes, using a number of queries to f_w that is bounded above by $O\left(\sqrt{n}\log\left(1/\epsilon\gamma^2\right)\right)$ with probability of failure at most $\epsilon\gamma^2$.*

The interested reader is encouraged consult the appendix of (Kapoor et al., 2016) for proofs of those lemmas. Furthermore, Novikoff's theorem states that the $1/\gamma^2$ is an upper bound in the number of times the algorithms described in these lemmas need to be applied, which is the reason for the first line of the algorithm and the $1/\gamma^2$ term in the runtime.

6.3 Version space quantum perceptron

General idea: Discretize the space in which weight vectors live and use Grover's search algorithm to find one vector that correctly classifies all the data points. Instead of searching among the data points, we are searching among weight vectors.

The key of the version space training algorithm is to search, among the possible weight vectors, one that correctly shatters the data points. Instead of iterating over the data points to update a weight vector, we will iterate over a discrete set of weight vectors to search a good one. The reason why this algorithm is called version space quantum perceptron lies in the space where the weight vector live, which is referred to as the version space.

Theorem 6.2. *Given a training set with margin γ, a weight vector w sampled from a spherical gaussian distribution $\mathcal{N}(0, 1)$ perfectly separates the data with probability $\Theta(\gamma)$.*

The main idea is to sample k sample hyperplanes $w^{(1)}, \dots, w^{(k)}$ from a spherical Gaussian distribution $\mathcal{N}(0, \mathbb{1})$ such that k is large enough to guarantee us that there is at least one good weight vector. Theorem 6.2 tells us that the expected number of samples k scales as $O(1/\gamma)$, and classically we would need to test all the n data points for at least $O(1/\gamma)$ different vectors, with a cost $\Omega(n/\gamma)$. By exploiting the power of Grover's search we can have a quadratic speedup on k and achieve $O(n/\sqrt{\gamma})$.

Like in the previous case, to use Grover, we need one unitary U that creates the weight vectors among which we search and one oracle $\mathcal{F}$ that modifies the phase of the correct weight vectors.

The oracle U must do the following job:

$$U|j\rangle|0\rangle = |j\rangle|w^{(j)}\rangle \tag{6.11}$$

for $j \in \{1, \dots, k\}$, giving access to the k sampled weight vectors. Clearly U is a linear operator and can be used with indexes in superposition, to give access to all the weight vectors simultaneously $U\frac{1}{\sqrt{k}}\sum_{j=1}^{k}|j\rangle|0\rangle = \frac{1}{\sqrt{k}}\sum_{j=1}^{k}|j\rangle|w^{(j)}\rangle$.

On the other hand, the oracle $\mathcal{F}$ should perform the following operation:

$$\mathcal{F}|w^{(j)}\rangle = (-1)^{1+\left(f(w^{(j)},x^{(1)},y^{(1)})\vee\cdots\vee f(w^{(j)},x^{(n)},y^{(n)})\right)}|w^{(j)}\rangle. \tag{6.12}$$

Here $f\left(w^{(j)}, x^{(i)}, y^{(i)}\right)$ works as in the quantum online perceptron algorithm: returns 1 if the data sample $(x^{(i)}, y^{(i)})$ is not correctly predicted by the weight

$w^{(j)}$. Basically, $1 + \left(f\left(w^{(j)}, x^{(1)}, y^{(1)}\right) \vee \dots \vee f\left(w^{(j)}, x^{(n)}, y^{(n)}\right)\right)$ returns 1 if the vector $w^{(j)}$ correctly classifies all the data points and 0 if there exists a training sample that is not correctly classified. Similarly to the previous case, this circuit can be built by "hardcoding" the knowledge of the data samples in each of the n functions or by creating one flexible function that accepts the vector and the points as input (by using additional qubits for the data points). This operation can be implemented with $O(n)$ queries to f.

Once we have the oracles U and $\mathcal{F}$, we can sum up the algorithm in pseudocode.

The following theorem formalizes the runtime of the algorithm.

Theorem 6.3 (Version space quantum perceptron (kapoor2016quantum))**.** *Given a training set that consists of unit vectors $x^{(1)}, \dots, x^{(n)}$ and labels $y^{(1)}, \dots, y^{(n)}$, separated by a margin of γ, the number of queries to f needed to infer a perceptron model w, with probability at least $1 - \epsilon$, using a quantum computer is $O\left(\frac{n}{\sqrt{\gamma}} \log\left(\frac{1}{\epsilon}\right)\right)$.*

The proof of this theorem is straightforward from the statement of Grover's algorithm and some statistics to bound the failure probability.

Chapter 7

SVE-based quantum algorithms

In the following section, we will cover some quantum algorithms based on singular value estimation. Some of them are here just because they are simple enough to have a good pedagogical value, while some of them we believe will be really useful in performing data analysis.

7.1 Spectral norm and the condition number estimation

We will elaborate more on this result soon. For the moment, we report the main statement.

Theorem 7.1 (Spectral norm estimation)**.** *Let there be quantum access to the matrix $A \in \mathbb{R}^{n\times m}$, and let $\epsilon > 0$ be a precision parameter. There exists a quantum algorithm that estimates $\|A\|$ to error $\epsilon\|A\|_F$ in time $\widetilde{O}\left(\frac{\log(1/\epsilon)}{\epsilon}\frac{\|A\|_F}{\|A\|}\right)$.*

7.2 Explained variance: estimating quality of representations

The content of this section is extracted from (Bellante and Zanero, 2022).

Let $A = U\Sigma V^T$ be the singluar value decomposition of a matrix $A \in \mathbb{R}^{n\times m}$. We call *factor scores* of A, and denote them with $\lambda_i = \sigma_i^2$, the squares of its singular values. Similarly, we call *factor score ratios* the relative magnitudes of the factor scores $\lambda^{(i)} = \frac{\lambda_i}{\sum_j^{rank(A)}\lambda_j} = \frac{\sigma_i^2}{\sum_j^{rank(A)}\sigma_j^2}$. The factor score ratios are a measure of the amount of variance explained by the singular values.

We state here some nice examples of SVE based algorithms: the first allows us to assess whether a few singular values/factor scores explain most of the variance of the matrix of the dataset; the second one allows computing the cumulative sum of the factor score ratios associated to singular values grater or equal than a certain threshold; the third one is a modified version of the spectral norm estimation result and allows us to define a threshold for the smallest singular value such that the the sum of the above explains more than a given percentage of the total variance; finally, the last two algorithms allow retrieving a classical description of the singular vectors that correspond to the most relevant singular values.

The main intuition behind the first algorithm is that it is possible to create the state $\sum_i^r \sqrt{\lambda^{(i)}}|u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle$ and that the third register, when measured in the computational basis, outputs the estimate $\overline{\sigma}_i$ of a singular value with probability equal to its factor score ratio $\lambda^{(i)}$. This allows us to sample the singular values of A directly from the factor score ratios' distribution. When a matrix has a huge number of small singular values and only a few of them that are very big, the ones with the greatest factor score ratios will appear many times during the measurements, while the negligible ones are not likely to be measured. This intuition has already appeared in literature (Gyurik et al., 2020) and (Cade and Montanaro, 2017). Nevertheless, the analysis and the problem solved are different, making the run-time analysis unrelated. This idea in the context of data representation and analysis, this intuition has only been sketched for sparse or low rank square symmetric matrices, by (Lloyd et al., 2013), without a precise formalization. We thoroughly formalize it, in a data representation and analysis context, for any real matrix.

Algorithm Quantum factor score ratio estimation.

```
$S = 0$
**while** $S < \widetilde{O}\left(\frac{1}{\gamma^2}\right)$ **do**
    Prepare the state $\frac{1}{\|A\|_F}\sum_i^n \sum_j^m a_{ij}\,|i\rangle\,|j\rangle$.
    Apply SVE to get $\frac{1}{\sqrt{\sum_j^r \sigma_j^2}}\sum_i^r \sigma_i\,|u_i\rangle\,|v_i\rangle\,|\overline{\sigma}_i\rangle$.
    Measure the last register and store a counter of how many times each distinct value $|\overline{\sigma}_i\rangle$ is measured.
    $S = S + 1$
**end while**
For each distinct $\overline{\sigma}_i$ measured, output $\overline{\sigma}_i$ and its factor score $\overline{\lambda}_i = \overline{\sigma}_i^2$.
For each distinct $\overline{\sigma}_i$ measured, output its factor score ratio $\overline{\lambda}^{(i)} = \frac{\zeta_{\overline{\sigma}_i}}{S}$, where $\zeta_{\overline{\sigma}_i}$ is the number of times $|\overline{\sigma}_i\rangle$ has been observed among the measurements. Each estimate comes with confidence level $z$.
```

Figure 7.1: Quantum factor score ratio estimation

Theorem 7.2 (Quantum factor score ratio estimation)**.** *Assume to have quantum access to a matrix $A \in \mathbb{R}^{n\times m}$ and $\sigma_{max} \leq 1$. Let γ, ϵ be precision parameters.*

There exists a quantum algorithm that, in time $\widetilde{O}\left(\frac{1}{\gamma^2}\frac{\mu(A)}{\epsilon}\right)$, estimates:

- *the factor score ratios $\lambda^{(i)}$, such that $\|\lambda^{(i)} - \overline{\lambda}^{(i)}\| \leq \gamma$, with high probability;*
- *the correspondent singular values σ_i, such that $\|\sigma_i - \overline{\sigma}_i\| \leq \epsilon$, with probability at least $1 - 1/poly(n)$;*
- *the correspondent factor scores λ_i, such that $\|\lambda_i - \overline{\lambda}_i\| \leq 2\epsilon$, with probability at least $1 - 1/poly(n)$.*

The parameter γ is the one that controls how big a factor score ratio should be for the singular value/factor score to be measured. If we choose γ bigger than the least factor scores ratio of interest, the estimate for the smaller ones is likely to be 0, as $\|\lambda^{(i)} - 0\| \leq \gamma$ would be a plausible estimation.

Often in data representations, the cumulative sum of the factor score ratios is a measure of the quality of the representation. By slightly modifying Algorithm in Figure 7.1 to use Theorem 3.16, it is possible to estimate this sum such that $\|\sum_i^k \lambda^{(i)} - \sum_i^k \overline{\lambda}^{(i)}\| \leq k\epsilon$ with probability $1 - 1/\text{poly}(r)$.

However, a slight variation of the algorithm of Theorem 7.1 provides a more accurate estimation in less time, given a threshold θ for the smallest singular value to retain.

Theorem 7.3 (Quantum check on the factor score ratios' sum)**.** *Assume to have efficient quantum access to the matrix $A \in \mathbb{R}^{n\times m}$, with singular value decomposition $A = \sum_i \sigma_i u_i v_i^T$. Let η, ϵ be precision parameters and θ be a threshold for the smallest singular value to consider. There exists a quantum algorithm that estimates $p = \frac{\sum_{i:\overline{\sigma}_i \geq \theta} \sigma_i^2}{\sum_j^r \sigma_j^2}$, where $\|\sigma_i - \overline{\sigma}_i\| \leq \epsilon$, to relative error η in time $\widetilde{O}\left(\frac{\mu(A)}{\epsilon}\frac{1}{\eta\sqrt{p}}\right)$.*

Moreover, we borrow an observation from (Kerenidis and Prakash, 2020) on Theorem 7.1, to perform a binary search of θ given the desired sum of factor score ratios.

Theorem 7.4 (Quantum binary search for the singular value threshold)**.** *Assume to have quantum access to the matrix $A \in \mathbb{R}^{n\times m}$. Let p be the factor ratios sum to retain. The threshold θ for the smallest singular value to retain can be estimated to absolute error ϵ in time $\widetilde{O}\left(\frac{\log(1/\epsilon)\mu(A)}{\epsilon\sqrt{p}}\right)$.*

We will see in the next chapters that in problems such as PCA, CA, and LSA, the desired sum of factor score ratios to retain is a number in the range $p \in [1/3, 1]$ with precision up to the second decimal digit. In practice, the complexity of these last two algorithms scales as $\widetilde{O}\left(\frac{\mu(A)}{\epsilon}\right)$.

7.3 Extracting the SVD representations

After introducing the procedures to test for the most relevant singular values, factor scores and factor score ratios of A, we present an efficient routine to extract the corresponding right/left singular vectors. The inputs of this algorithm, other than the matrix, are a parameter δ for the precision of the singular vectors, a parameter ϵ for the precision of the singular value estimation, and a threshold θ to discard the non interesting singular values/vectors. The output guarantees a unit estimate $\overline{x}_i$ of each singular vector such that $||x_i - \overline{x}_i||_\ell \leq \delta$ for $\ell \in \{2, \infty\}$, ensuring that the estimate has a similar orientation to the original vector. Additionally, this subroutine can provide an estimation of the singular values greater than θ, to absolute error ϵ.

Algorithm Quantum top-k singular vectors extraction.

```
1: Prepare the state $\frac{1}{\|A\|_F}\sum_i^n\sum_j^m a_{ij}\left|i\right\rangle\left|j\right\rangle$.
2: Apply SVE to get $\frac{1}{\sqrt{\sum_j^r \sigma_j^2}}\sum_i^r \sigma_i\left|u_i\right\rangle\left|v_i\right\rangle\left|\overline{\sigma}_i\right\rangle$.
3: Append a quantum register $\left|0\right\rangle$ to the state and set it to $\left|1\right\rangle$ if $\left|\overline{\sigma}_i\right\rangle < \theta$.
4: Perform amplitude amplification for $\left|0\right\rangle$, to get the state $\frac{1}{\sqrt{\sum_j^k \sigma_j^2}}\sum_i^k \sigma_i\left|u_i\right\rangle\left|v_i\right\rangle\left|\overline{\sigma}\right\rangle$.
5: Append a second ancillary register $\left|0\right\rangle$ and perform the controlled rotation $\frac{C}{\|A^{(k)}\|_F}\sum_i^k \frac{\sigma_i}{\overline{\sigma}_i}\left|u_i\right\rangle\left|v_i\right\rangle\left|\overline{\sigma}_i\right\rangle\left|0\right\rangle + \frac{1}{\|A^{(k)}\|_F}\sum_i^k\sqrt{1-\frac{C^2}{\overline{\sigma}_i^2}}\left|u_i\right\rangle\left|v_i\right\rangle\left|\overline{\sigma}_i\right\rangle\left|1\right\rangle$ where $C$ is a normalization constant.
6: Perform again amplitude amplification for $\left|0\right\rangle$ to get $\frac{1}{\sqrt{k}}\sum_i^k\left|u_i\right\rangle\left|v_i\right\rangle\left|\overline{\sigma}_i\right\rangle$ .
7: Measure the last register and, according to the measured $\left|\overline{\sigma}_i\right\rangle$, apply state-vector tomography on $\left|u_i\right\rangle$ for the $i^{th}$ left singular vector or on $\left|v_i\right\rangle$ for the right one.
8: Repeat 1-7 until the tomography requirements are met.
9: Output the $k$ singular vectors $u_i$ or $v_i$ and, optionally, the singular values $\overline{\sigma}_i$.
```

Figure 7.2: Quantum algorithm for top-k singular vector extractor

Theorem 7.5 (Top-k singular vectors extraction)**.** *Let there be efficient quantum access to the matrix $A \in \mathbb{R}^{n\times m}$, with singular value decomposition $A = \sum_i^r \sigma_i u_i v_i^T$ and $\sigma_{max} \leq 1$. Let $\delta > 0$ be a precision parameter for the singular vectors, $\epsilon > 0$ a precision parameter for the singular values, and $\theta > 0$ be a threshold such that A has k singular values greater than θ. Define $p = \frac{\sum_{i:\overline{\sigma}_i\geq\theta}\sigma_i^2}{\sum_j^r \sigma_j^2}$. There exist quantum algorithms that estimate:*

- *The top k left singular vectors u_i of A with unit vectors $\overline{u}_i$ such that $\|u_i - \overline{u}_i\|_2 \leq \delta$ with probability at least $1 - 1/poly(n)$, in time $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{kn}{\delta^2}\right)$;*
- *The top k right singular vectors v_i of A with unit vectors $\overline{v}_i$ such that $\|v_i - \overline{v}_i\|_2 \leq \delta$ with probability at least $1 - 1/poly(m)$, in time $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{km}{\delta^2}\right)$.*

- *The top k singular values σ_i and factor scores λ_i of A to precision ϵ and 2ϵ with probability at least $1 - 1/poly(m)$, in time $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)k}{\epsilon}\right)$ or any of the two above.*

Besides p being negligible, it is interesting to note that the parameter θ can be computed using:

- the procedures of Theorems @ref(thm:factor_score_estimation) and @ref(thm:check_explained_variance);
- the binary search of Theorem **??**;
- the available literature on the type of data stored in the input matrix A.

About the latter, the original paper of latent semantic indexing (Deerwester et al., 1990) states that the first $k = 100$ singular values are enough for a good representation. We believe that, in the same way, fixed thresholds θ can be defined for different machine learning applications. The experiments that you can read in Chapter 12 on the run-time parameters of the polynomial expansions of the MNIST dataset support this expectation: even though in qSFA they keep the k smallest singular values and refer to θ as the biggest singular value to retain, this value does not vary much when the the dimensionality of their dataset grows. In our experiments, we observe that different datasets for image classification have similar θs.

A similar statement to Theorem @ref(thm:top-k_sv_extraction) can be stated with ℓ_∞ guarantees on the vectors (see Corollary 13 of (Bellante and Zanero, 2022)).

As we discussed before, given a vector with d non-zero entries, performing ℓ_∞ tomography with error $\frac{\delta}{\sqrt{d}}$ provides the same guarantees of ℓ_2 tomography with error δ.

In practice, this result implies that the extraction of the singular vectors, with ℓ_2 guarantees, can be faster if we can assume some prior assumptions on their sparseness: $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{kd}{\delta^2}\right)$.

The main intuition behind these algorithms is that it is possible to create the state $\sum_i^r \sqrt{\lambda^{(i)}}|u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle$.

The first algorithm uses a parameter γ to control how big a factor score ratio should be for the corresponding singular value/factor score to be measured. If we choose γ bigger than the least factor scores ratio of interest, only the biggest singular values/factor scores will be likely to be measured.

Theorem 7.6 (Quantum factor score ratio estimation)**.** *Let there be quantum access to a matrix $A \in \mathbb{R}^{n\times m}$, with singular value decomposition $A = \sum_i \sigma_i u_i v_i^T$ and $\sigma_{max} \leq 1$. Let γ, ϵ be precision parameters. There exists a quantum algorithm that, in time $\widetilde{O}\left(\frac{1}{\gamma^2}\frac{\mu(A)}{\epsilon}\right)$, estimates:*

- *the factor score ratios $\lambda^{(i)}$, such that $\|\lambda^{(i)} - \overline{\lambda}^{(i)}\| \leq \gamma$, with high probability;*

- *the correspondent singular values σ_i, such that $\|\sigma_i - \overline{\sigma}_i\| \leq \epsilon$, with probability at least $1 - 1/poly(n)$;*
- *the correspondent factor scores λ_i, such that $\|\lambda_i - \overline{\lambda}_i\| \leq 2\epsilon$, with probability at least $1 - 1/poly(n)$.*

Proof. We provide an algorithm that satisfies the above guarantees. As a first step, one creates the state

$$|A\rangle = \frac{1}{\|A\|_F} \sum_i^n \sum_j^m a_{ij} |i\rangle|j\rangle = \frac{1}{\sum_j^r \sigma_j^2} \sum_i^r \sigma_i |u_i\rangle|v_i\rangle.$$

This step costs $\widetilde{O}(1)$, assuming that the data are stored in an adeguate data structure. From this state we apply SVE in time $\widetilde{O}(\mu(A)/\epsilon)$

$$|A'\rangle = \frac{1}{\sum_j^r \sigma_j^2} \sum_i^r \sigma_i |u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle \tag{7.1}$$

encoding the singular values with absolute precision ϵ. If we ignore the first two registers, we have the state $|A''\rangle = \sum_i^r \sqrt{\lambda^{(i)}} |\overline{\sigma}_i\rangle$, from which we can measure the singular values (and factor scores) with probability equal to their factor score ratios. To evaluate the number S of measurements needed on $|A''\rangle$ to satisfy the guarantees of the theorem, we can model the measurement process as performing r Bernoulli trials: one for each $\overline{\lambda}^{(i)}$, so that if we measure $\overline{\sigma}_i$ it is a success for the i^{th} Bernoulli trial and a failure for all the others. We use the estimator $\overline{\lambda}^{(i)} = \frac{\zeta_{\overline{\sigma}_i}}{S}$, where $\zeta_{\overline{\sigma}_i}$ is the number of times $\overline{\sigma}_i$ appears in the measurements and S is the number of total measurements. Given a confidence level z and an absolute error γ, it is possible to use the Wald confidence interval to determine a value for S such that $\|\lambda^{(i)} - \overline{\lambda}^{(i)}\| \leq \gamma$ with confidence level z. It is possible to show that $\gamma \leq \frac{z}{2\sqrt{S}}$ (Schuld and Petruccione, 2018), from which we get $S = \frac{z^2}{4\gamma^2}$. Since z is a small number, we can state that the complexity of the algorithm is $\widetilde{O}\left(\frac{1}{\gamma^2}\frac{\mu(A)}{\epsilon}\right)$.

Finally, note that the error on the factor scores is 2ϵ. Suppose that we run SVE with precision τ. For each λ_i, the worst estimate is $\overline{\lambda_i} = (\sigma_i \pm \tau)^2 = \sigma_i^2 \pm 2\tau\sigma_i + \tau^2$ and since $0 \leq \sigma_i \leq 1$, we can say that the worst case is $\sigma_i^2 + (2\tau + \tau^2)$. Solving the equation $2\tau + \tau^2 = \epsilon$ for $\tau > 0$ leads to $\tau = \sqrt{1+\epsilon} - 1$. Finally,

$$\tau = \sqrt{1+\epsilon} - 1 = \frac{(\sqrt{1+\epsilon} - 1)(\sqrt{1+\epsilon} + 1)}{(\sqrt{1+\epsilon} + 1)} = \frac{1 + \epsilon - 1}{\sqrt{1+\epsilon} + 1} = \frac{\epsilon}{\sqrt{1+\epsilon} + 1} \sim \frac{\epsilon}{2}$$

which proves the error guarantees. □

Theorem 7.7 (Quantum check on the factor score ratios' sum)**.** *Let there be efficient quantum access to the matrix $A \in \mathbb{R}^{n \times m}$, with singular value decomposition $A = \sum_i \sigma_i u_i v_i^T$. Let η, ϵ be precision parameters and θ be a threshold* *for the smallest singular value to consider. There exists a quantum algorithm that estimates* $p = \frac{\sum_{i:\overline{\sigma}_i \geq \theta} \sigma_i^2}{\sum_j^r \sigma_j^2}$, *where* $\|\sigma_i - \overline{\sigma}_i\| \leq \epsilon$, *to relative error* η *in time* $\widetilde{O}\left(\frac{\mu(A)}{\epsilon}\frac{1}{\eta\sqrt{p}}\right)$.

Proof. As reported in the proof above, creating the state $|A''\rangle = \sum_i^r \sqrt{\lambda^{(i)}}|\overline{\sigma}_i\rangle$ costs $\widetilde{O}(\mu(A)/\epsilon)$. From this state it is possible to append a quantum register that is $|0\rangle$ if $\overline{\sigma}_i < \theta$ and $|1\rangle$ otherwhise: $|\varphi\rangle = \sum_{i:\overline{\sigma}_i \geq \theta} \sqrt{\lambda^{(i)}}|\overline{\sigma}_i\rangle|0\rangle + \sum_{j:\overline{\sigma}_j < \theta} \sqrt{\lambda^{(i)}}|\overline{\sigma}_i\rangle|1\rangle$. The probability of measuring $|0\rangle$ is $p = \sum_{i:\overline{\sigma}_i \geq \theta} \lambda^{(i)} = \frac{\sum_{i:\overline{\sigma}_i \geq \theta} \sigma_i^2}{\sum_j^r \sigma_j^2}$. Using amplitude estimation (Lemma 5.1), we can estimate p in time $\widetilde{O}\left(\frac{\mu(A)}{\sqrt{p}\epsilon}\right)$. □

Theorem 7.8 (Top-k singular vectors extraction)**.** *Let there be efficient quantum access to the matrix* $A \in \mathbb{R}^{n \times m}$, *with singular value decomposition* $A = \sum_i^r \sigma_i u_i v_i^T$ *and* $\sigma_{max} \leq 1$. *Let* $\delta > 0$ *be a precision parameter for the singular vectors,* $\epsilon > 0$ *a precision parameter for the singular values, and* $\theta > 0$ *be a threshold such that A has k singular values greater than* θ. *Define* $p = \frac{\sum_{i:\overline{\sigma}_i \geq \theta} \sigma_i^2}{\sum_j^r \sigma_j^2}$. *There exist quantum algorithms that estimate:*

- *The top k left singular vectors* u_i *of A with unit vectors* $\overline{u}_i$ *such that* $||u_i - \overline{u}_i||_2 \leq \delta$ *with probability at least* $1 - 1/poly(n)$, *in time* $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{kn}{\delta^2}\right)$;
- *The top k right singular vectors* v_i *of A with unit vectors* $\overline{v}_i$ *such that* $||v_i - \overline{v}_i||_2 \leq \delta$ *with probability at least* $1 - 1/poly(m)$, *in time* $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{km}{\delta^2}\right)$;
- *The top k singular values* σ_i *and factor scores* λ_i *of A to precision* ϵ *and* 2ϵ *with probability at least* $1 - 1/poly(m)$, *in time* $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)k}{\epsilon}\right)$ *or any of the two above.*

Proof. As reported in the proofs above, creating the state $|A'\rangle = \sum_i^r \sqrt{\lambda^{(i)}}|u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle$ costs $\widetilde{O}(\mu(A)/\epsilon)$. On this state we append an additional register, set to $|0\rangle$ if $\overline{\sigma}_i \geq \theta$ and to $|1\rangle$ otherwise. The cost of this operation is negligible as it only depends on the binary encoding of $|\overline{\sigma}_i\rangle$. On this state we perform amplitude amplification (Lemma 5.1) on $|0\rangle$ and obtain the state

$$|\varphi\rangle = \sum_i^k \sqrt{\lambda^{(i)}}|u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle \tag{7.2}$$

with a superposition of the k most relevant singular values and vectors. Creating $|\varphi\rangle$ costs $\widetilde{O}\left(\frac{\mu(A)}{\sqrt{p}\epsilon}\right)$. On this state we append an additional register and perform a conditional rotation

$$\frac{C}{\sum_i^k \sigma_i^2}\sum_i^k \frac{\sigma_i}{\overline{\sigma}_i}|u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle|0\rangle + \frac{1}{\sum_i^k \sigma_i^2}\sum_i^k \sqrt{1-\frac{C^2}{\overline{\sigma}_i^2}}|u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle|1\rangle \tag{7.3}$$

The constant C is a normalization factor in the order of $\widetilde{O}(1/\kappa(A^{(k)}))$ where $\kappa(A^{(k)}) = \frac{\sigma_{max}}{\sigma_{min}}$ is the condition number of the low-rank (k-rank) matrix $A^{(k)}$. Since for construction $\sigma_{max} \leq 1$ and $\sigma_{min} \geq \theta$, we can bound the condition number $\kappa(A^{(k)}) \leq \frac{1}{\theta}$. From the famous work of Harrow, Hassidim and Lloyd (Harrow et al., 2009) we know that applying amplitude amplification on the state above, with the the third register being $|0\rangle$, would cost $\widetilde{O}(\kappa(A^{(k)})T(U)$, where $T(U)$ is the run-time to create the state. In our case, the cost till this step is $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\right)$, and amplitude amplification leaves the registers in the state

$$\frac{1}{\sqrt{\sum_i^k \frac{\sigma_i^2}{\overline{\sigma}_i^2}}}\sum_i^k \frac{\sigma_i}{\overline{\sigma}_i}|u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle \sim \frac{1}{\sqrt{k}}\sum_i^k |u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle$$

where $\overline{\sigma}_i \in [\sigma_i - \epsilon, \sigma_i + \epsilon]$ and $\frac{\sigma_i}{\sigma_i \pm \epsilon} \to 1$ for $\epsilon \to 0$. The last step of the proof is to compute the time complexity of the state-vector tomography on $\frac{1}{\sqrt{k}}\sum_i^k |u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle$. When measuring the last register of state in the computational basis and obtaining $|\overline{\sigma}_i\rangle$, the first two registers collapse in the state $|u_i\rangle|v_i\rangle$. On $|u_i\rangle|v_i\rangle$, it is possible to perform ℓ_2 vector-state tomography using Theorem 3.15 either on the first register, to retrieve $\overline{u}_i$, or on the second one, to retrieve $\overline{v}_i$. Since $u_i \in R^n$ and $v_i \in R^m$, performing state-vector tomography on the first register takes time $O(\frac{n\log n}{\delta^2})$ and performing it on the second takes time $O(\frac{m\log m}{\delta^2})$. Using a coupon collector's argument, if the k states $|\overline{\sigma}_i\rangle$ are uniformly distributed, to get all the k possible couples $|u_i\rangle|v_i\rangle$ at least once we would need $k\log k$ measurements on average. This proves that it is possible to estimate all the singular values greater than θ, with the guarantees above, in time $\widetilde{O}(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)k}{\epsilon})$. To perform tomography on each state-vector, one should satisfy the coupon collector the same number of times as the measurements needed by the tomography procedure. The costs of the tomography for all the vectors $\{\overline{u}_i\}_i^k$ and $\{\overline{v}_i\}_i^k$ are $O\left(T(U)\frac{k\log k \cdot n\log n}{\delta^2}\right)$, and $O\left(T(U)\frac{k\log k \cdot m\log m}{\delta^2}\right)$, where $T(U)$ is the run-time to create the state on which to perform tomography. It easily follows that the following complexities are proven: $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{kn}{\delta^2}\right), \widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{km}{\delta^2}\right)$. □

Corollary 7.1 (Fast top-k singular vectors extraction)**.** *The run-times of the theorem above can be improved to* $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{k}{\delta^2}\right)$ *with estimation guarantees on the* ℓ_∞ *norms.*

The proof of this corollary consists in a variant of the proof above that uses ℓ_∞ tomography (Theorem 3.16) to extract the singular vectors.

Besides p being negligible, it is interesting to note that the parameter θ can be computed using:

- the procedures of Theorems 7.6 and 7.7;
- the binary search of Theorem **??**;
- the available literature on the type of data stored in the input matrix A.

About the latter, for instance, the experiments of (Kerenidis and Luongo, 2020) on the run-time parameters of the polynomial expansions of the MNIST dataset support this expectation: even though in qSFA they keep the k smallest singular values and refer to θ as the biggest singular value to retain, this value does not vary much when the the dimensionality of their dataset grows. In the experiments chapter, we observe similar θs in different datasets for image classification. Note that, given a vector with d non-zero entries, performing ℓ_∞ tomography with error $\frac{\delta}{\sqrt{d}}$ provides the same guarantees of ℓ_2 tomography with error δ. In practice, this result implies that the extraction of the singular vectors, with ℓ_2 guarantees, can be faster if we can assume some prior assumptions on their sparseness: $\widetilde{O}\left(\frac{1}{\theta}\frac{1}{\sqrt{p}}\frac{\mu(A)}{\epsilon}\frac{kd}{\delta^2}\right)$.

7.4 Singular value estimation of a product of two matrices

This is an example of an algorithm that has been superseded by recent development in singular value transformation. Nevertheless, it is a non-trivial way of using SVE, which a nice mathematical error analysis.

Theorem 7.9 (SVE of product of matrices)**.** *Assume to have quantum access to matrices $P \in \mathbb{R}^{d\times d}$ and $Q \in \mathbb{R}^{d\times d}$. Define $W = PQ = U\Sigma V^T$ and $\epsilon > 0$ an error parameter. There is a quantum algorithm that with probability at least $1 - poly(d)$ performs the mapping $\sum_i \alpha|v_i\rangle \rightarrow \sum_i \alpha_i|v_i\rangle|\overline{\sigma_i}\rangle$ where $\overline{\sigma_i}$ is an approximation of the eigenvalues σ_i of W such that $|\sigma_i - \overline{\sigma}_i| \leq \epsilon$, in time $\tilde{O}\left(\frac{(\kappa(P)+\kappa(Q))(\mu(P)+\mu(Q))}{\varepsilon}\right)$.*

Proof. We start by noting that for each singular value σ_i of W there is a corresponding eigenvalue $e^{-i\sigma_i}$ of the unitary matrix e^{-iW}. Also, we note that we know how to multiply by W by applying theorem 5.9 sequentially with Q and P. This will allow us to approximately apply the unitary $U = e^{-iW}$. The last step will consist of the application of phase estimation to estimate the eigenvalues of U and hence the singular values of W. Note that we need W to be a symmetric matrix because of the Hamiltonian simulation part. In case W is not symmetric, we redefine it as

$$W = \begin{bmatrix} 0 & PQ \\ (PQ)^T & 0 \end{bmatrix}$$

Note we have $W = M_1 M_2$ for the matrices M_1, M_2 stored in QRAM and defined as

$$M_1 = \begin{bmatrix} P & 0 \\ 0 & Q^T \end{bmatrix}, M_2 = \begin{bmatrix} 0 & Q \\ P^T & 0 \end{bmatrix}.$$

We now show how to approximately apply $U = e^{-iW}$ efficiently. Note that for a symmetric matrix W we have $W = V\Sigma V^T$ and using the Taylor expansion of the exponential function we have

$$U = e^{-iW} = \sum_{j=0}^{\infty} \frac{(-iW)^j}{j!} = V \sum_{j=0}^{\infty} \frac{(-i\Sigma)^j}{j!} V^T$$

With $\widetilde{U}$ we denote our first approximation of U, where we truncate the sum after ℓ terms.

$$\widetilde{U} = \sum_{j=0}^{\ell} \frac{(-iW)^j}{j!} = V \sum_{j=0}^{\ell} \frac{(-i\Sigma)^j}{j!} V^T$$

We want to chose ℓ such that $\left\|U - \widetilde{U}\right\| < \epsilon/4$. We have:

$$\begin{aligned} \left\|U - \widetilde{U}\right\| &\leq \| \sum_{j=0}^{\infty} \frac{(-iW)^j}{j!} - \sum_{j=0}^{\ell} \frac{(-iW)^j}{j!} \| \leq \| \sum_{j=\ell+1}^{\infty} \frac{(-iW)^j}{j!} \| \leq \sum_{j=\ell+1}^{\infty} \| \frac{(-iW)^j}{j!} \| \leq \sum_{j=\ell+1}^{\infty} \frac{1}{j!} \\ &\leq \sum_{j=\ell+1}^{\infty} \frac{1}{2^{j-1}} \leq 2^{-\ell+1} \end{aligned}$$

where we used triangle inequality and that $\left\|W^j\right\| \leq 1$. Choosing $\ell = O(\log 1/\varepsilon)$ makes the error less than $\epsilon/4$. %We can approximate a positive series where the term a_n satisfy the following two conditions: $0 \leq a_n \leq Kr^n$ with $K > 0, 0 < r < 1$ by expressing the error as the geometric series $\frac{Kr^{N+1}}{1-r}$. In our case $K = 1$ and $r = 1/2$. For a given ϵ we have to find ℓ such that $\frac{(\frac{1}{2})^{\ell+1}}{1-(\frac{1}{2})} \leq \epsilon$. By taking $\ell = O(\log 1/\epsilon)$ we can easily satisfy the error guarantee.

In fact, we cannot apply $\widetilde{U}$ exactly but only approximately, since we need to multiply with the matrices $W^j, j \in [\ell]$ and we do so by using the matrix multiplication algorithm for the matrices M_1 and M_2. For each of these matrices, we use an error of $\frac{\epsilon}{8\ell}$ which gives an error for W of $\frac{\epsilon}{4\ell}$ and an error for W^j of at most $\frac{\epsilon}{4}$. The running time for multiplying with each W^j is at most $O(\ell(\kappa(M_1)\mu(M_1)\log(8\ell/\epsilon) + \kappa(M_2)\mu(M_2)\log(8\ell/\epsilon)))$ by multiplying sequentially. Hence, we will try to apply the unitary U by using the Taylor expansion up to level ℓ and approximating each $W^j, j \in [\ell]$ in the sum through our matrix multiplication procedure that gives error at most $\frac{\epsilon}{4}$.

In order to apply U on a state $|x\rangle = \sum_i \alpha_i |v_i\rangle$, let's assume $\ell + 1$ is a power of two and define $N_l = \sum_{j=0}^{l} (\frac{(-i)^j}{j!})^2$. We start with the state

$$\frac{1}{\sqrt{N_l}} \sum_{j=0}^{l} \frac{-i^j}{j!} |j\rangle |x\rangle$$

Controlled on the first register we use our matrix multiplication procedure to multiply with the corresponding power of W and get a state at most $\epsilon/4$ away from the state

$$\frac{1}{\sqrt{N_l}} \sum_{j=0}^{l} \frac{-i^j}{j!} |j\rangle |W^j x\rangle.$$

We then perform a Hadamard on the first register and get a state $\epsilon/4$ away from the state

$$\frac{1}{\sqrt{\ell}} |0\rangle \left(\frac{1}{\sqrt{N'}} \sum_{j=0}^{l} \frac{-i^j}{j!} |W^j x\rangle \right) + |0^{\perp}\rangle |G\rangle$$

where N' just normalizes the state in the parenthesis. Note that after the Hadamard on the first register, the amplitude corresponding to each $|i\rangle$ is the first register is the same. We use this procedure inside an amplitude amplification procedure to increase the amplitude $1/\sqrt{\ell}$ of $|0\rangle$ to be close to 1, by incurring a factor $\sqrt{\ell}$ in the running time. The outcome will be a state $\epsilon/4$ away from the state

$$\left(\frac{1}{\sqrt{N'}} \sum_{j=0}^{l} \frac{-i^j}{j!} |W^j x\rangle \right) = |\tilde{U} x\rangle$$

which is the application of $\widetilde{U}$. Since $\left\| U - \widetilde{U} \right\| \leq \epsilon/4$, we have that the above procedure applies a unitary $\overline{U}$ such that $\left\| U - \overline{U} \right\| \leq \epsilon/2$. Note that the running time of this procedure is given by the amplitude amplification and the time to multiply with W^j, hence we have that the running time is

$$O(\ell^{3/2}(\kappa(M_1)\mu(M_1)\log(8\ell/\epsilon) + \kappa(M_2)\mu(M_2)\log(8\ell/\epsilon))$$

Now that we know how to apply $\overline{U}$, we can perform phase estimation on it with error $\epsilon/2$. This provides an algorithm for estimating the singular values of W with overall error of ϵ. The final running time is

$$O(\frac{\ell^{3/2}}{\epsilon}(\kappa(M_1)\mu(M_1)\log(8\ell/\epsilon) + \kappa(M_2)\mu(M_2)\log(8\ell/\epsilon))$$

We have $\mu(M_1) = \mu(M_2) = \mu(P) + \mu(Q)$ and $\kappa(M_1) = \kappa(M_2) = \frac{max\{\lambda^P_{max},\lambda^Q_{max}\}}{min\{\lambda^P_{min},\lambda^Q_{min}\}} \leq \kappa(P) + \kappa(Q)$, and since $\ell = O(\log 1/\epsilon)$the running time can be simplified to

$$\widetilde{O}(\frac{(\kappa(P)+\kappa(Q))(\mu(P)+\mu(Q))}{\epsilon}).$$

□

7.5 A last example: Slow algorithms for log-determinant

A very simple example of the utility of the SVE subroutines is to estimate quantities associated to a given matrix. In this case we are going to study the log-determinant. As the name sais, this is just the logarithm of the determinant of a (symmetric positive definite) SPD matrix.

Definition 7.1 (Log-determinant of an SPD matrix)**.** Let $A \in \mathbb{R}^{n\times n}$ be a SPD matrix with singular value decomposition $A = U\Sigma V^T$. The log-determinant of A is defined as:

$$\log\det(A) = \log(\prod_i^n \sigma_i) = \sum_i^n \log(\sigma_i)$$

Please keep in mind that this is *not* the fastest algorithm for estimating the log-determinant (we will see that in the appropriate chapter on spectral sums), but it's worth mentioning here because it perhaps the first thing one would try to do in order to estimate this quantity. It also is a good example of the power of quantum singular value estimation, and checking the correctness of this proof might be a good exercise to learn more some mathematical tricks that are very useful to upper bound quantities that appear in the error analysis or the runtime analysis of algorithms.

Theorem 7.10 (SVE based algorithm for log-determinant)**.** *Assuming to have quantum access to an SPD matrix A, the algorithm in figure 7.3 returns* $\overline{\log\det(A)}$ *such that* $|\overline{\log\det(A)} - \log\det(A)| < \epsilon|\log\det(A)|$ *in time* $\widetilde{O}(\mu\kappa^3/\epsilon^2)$.

Proof. We can rewrite the quantum state encoding the representation of A (which we can create with quantum access to A) as follow:

$$|A\rangle = \frac{1}{\|A\|_F}\sum_{i,j=1}^n a_{ij}|i,j\rangle = \frac{1}{\|A\|_F}\sum_{j=1}^n \sigma_j|u_j\rangle|u_j\rangle. \tag{7.4}$$

Starting from the state $|A\rangle$, we can apply SVE (see lemma 5.8 to $|A\rangle$ up to precision ϵ_1 to obtain

$$\frac{1}{\|A\|_F}\sum_{j=1}^n \sigma_j|u_j\rangle|u_j\rangle|\tilde{\sigma}_j\rangle,$$

```
Require: Quantum access to the SPD matrix $A \in \mathbb{R}^{n\times n}$ such that $\|A\| \leq 1$.
    Choose $\epsilon \in (0,1)$, $\epsilon_1 = \epsilon/\kappa\log\kappa$ and $\epsilon_2 = \epsilon/\kappa^2\log\kappa$.
Ensure: An estimate $|\overline{\log\det(A)} - \log\det(A)| \leq n\epsilon$.
Prepare
    $$|A\rangle = \frac{1}{\|A\|_F}\sum_{i,j=1}^{n} a_{ij}|i,j\rangle.$$
Perform SVE up to precision $\epsilon_1$ and do control rotations to prepare
    $$\frac{1}{\|A\|_F}\sum_{j=1}^{n}\sigma_j|u_j\rangle|u_j\rangle|\tilde{\sigma}_j\rangle\left(C\frac{\sqrt{|\log\tilde{\sigma}_j|}}{\tilde{\sigma}_j}|0\rangle + |\bot\rangle\right),$$
    where $C = \min_j \tilde{\sigma}_j/\sqrt{|\log\tilde{\sigma}_j|} \approx \sigma_{\min}/\sqrt{|\log\sigma_{\min}|} = 1/\kappa\sqrt{\log\kappa}$.
Apply amplitude estimation to estimate the probability of $|0\rangle$ to precision
    $\epsilon_2$. Set the result as $P$.
Return $\overline{\log\det(A)} = -\kappa^2(\log\kappa)\|A\|_F^2 P$.
```

Figure 7.3: SVE based algorithm to estimatethe log-determinant of a matrix

where $|\tilde{\sigma}_j - \sigma_j| \leq \epsilon_1$. Since $\|A\| \leq 1$, using controlled operations, we can prepare

$$\frac{1}{\|A\|_F}\sum_{i=1}^{n}\sigma_j|u_j\rangle|u_j\rangle|\tilde{\sigma}_j\rangle\left(C\frac{\sqrt{-\log\tilde{\sigma}_j}}{\tilde{\sigma}_j}|0\rangle + |0^{\bot}\rangle\right), \tag{7.5}$$

where $C = \min_j \tilde{\sigma}_j/\sqrt{|\log\tilde{\sigma}_j|} \approx \sigma_{\min}/\sqrt{|\log\sigma_{\min}|} = 1/\kappa\sqrt{\log\kappa}$. The probability of $|0\rangle$ is

$$P = -\frac{C^2}{\|A\|_F^2}\sum_{j=1}^{n}\frac{\sigma_j^2}{\tilde{\sigma}_j^2}\log\tilde{\sigma}_j.$$

First, we do the error analysis. Note that

$$
\begin{align}
\left|\sum_{j=1}^{n}\frac{\sigma_j^2}{\tilde{\sigma}_j^2}\log\tilde{\sigma}_j-\sum_{j=1}^{n}\log\sigma_j\right| &\leq \left|\sum_{j=1}^{n}\frac{\sigma_j^2}{\tilde{\sigma}_j^2}\log\tilde{\sigma}_j-\sum_{j=1}^{n}\frac{\sigma_j^2}{\tilde{\sigma}_j^2}\log\sigma_j\right|+\left|\sum_{j=1}^{n}\frac{\sigma_j^2}{\tilde{\sigma}_j^2}\log\sigma_j-\sum_{j=1}^{n}\log\sigma_j\right| \tag{7.6}\\
&\leq \sum_{j=1}^{n}\frac{\sigma_j^2}{\tilde{\sigma}_j^2}|\log\tilde{\sigma}_j-\log\sigma_j|+\sum_{j=1}^{n}\frac{|\sigma_j^2-\tilde{\sigma}_j^2|}{\tilde{\sigma}_j^2}|\log\sigma_j| \tag{7.7}\\
&\leq \sum_{j=1}^{n}(1+\frac{\epsilon_1}{\tilde{\sigma}_j})^2(\frac{\epsilon_1}{\tilde{\sigma}_j}+O(\frac{\epsilon_1^2}{\tilde{\sigma}_j^2}))+(2\kappa\epsilon_1+\kappa^2\epsilon_1^2)|\log\det(A)| \tag{7.8}\\
&\leq n(\kappa\epsilon_1+O(\kappa^2\epsilon_1^2))+(2\kappa\epsilon_1+\kappa^2\epsilon_1^2)|\log\det(A)| \tag{7.9}\\
&= (n+2|\log\det(A)|)(\kappa\epsilon_1+O(\kappa^2\epsilon_1^2)). \tag{7.10}
\end{align}
$$

In the third inequality, we use the result that $\sigma_j \leq \tilde{\sigma}_j + \epsilon_1$.

Denote P' as the ϵ_2-approximation of P obtained by amplitude estimation, then the above analysis shows that $-\|A\|_F^2 P'/C^2$ is an $(n+2|\log\det(A)|)(\kappa\epsilon_1 + O(\kappa^2\epsilon_1^2)) + \epsilon_2\|A\|_F^2/C^2$ approximation of $\log\det(A)$. Note that

$$
\begin{align}
& (n+2|\log\det(A)|)(\kappa\epsilon_1+O(\kappa^2\epsilon_1^2))+\epsilon_2\|A\|_F^2/C^2 \tag{7.11}\\
= & (n+2|\log\det(A)|)(\kappa\epsilon_1+O(\kappa^2\epsilon_1^2))+\epsilon_2\|A\|_F^2\kappa^2\log\kappa \tag{7.12}\\
\leq & (n+2n\log\kappa)(\kappa\epsilon_1+O(\kappa^2\epsilon_1^2))+n\epsilon_2\kappa^2\log\kappa \tag{7.13}\\
= & O(n\epsilon_1\kappa\log\kappa+n\epsilon_2\kappa^2\log\kappa). \tag{7.14}
\end{align}
$$

To make sure the above error is bounded by $n\epsilon$ it suffcies to choose $\epsilon_1 = \epsilon/\kappa\log\kappa$ and $\epsilon_2 = \epsilon/\kappa^2\log\kappa$.

Now we do the runtime analysis. The runtime of the algorithm mainly comes from the using of SVE and the performing of amplitude estimation on the state in ((7.5)). Using quantum singular value estimation, the complexity to obtain the state (7.5) is $\widetilde{O}(\mu/\epsilon_1)$. The complexity to perform amplitude estimation is $\widetilde{O}(\mu/\epsilon_1\epsilon_2) = \widetilde{O}(\mu\kappa^3(\log\kappa)^2/\epsilon^2)$. □

Chapter 8

Quantum algorithms for Monte Carlo

Contributors: Michele Vischi, Alessandro Luongo

In this chapter we focus on how quantum computing speeds up classical Monte Carlo techniques. Monte Carlo methods are ubiquitous across science and find many broad applications such as the evaluation of integrals and many more specific applications such as the pricing of financial derivatives, the computation of distances among probability distributions and so on. The essence of Monte Carlo methods is to estimate some expected value of a function f of one (or more) random variable X, $\mathbb{E}[f(X)]$. As already discussed in chapter 3, in the quantum setting, one needs a procedure to load data on quantum registers. In solving Monte Carlo problems, one needs access to the probability distributions pertaining to the random variables. In this chapter we will usually implicitly assume to have quantum query or oracle access to functions 2.2. We will try to state clearly whenever it is not the case or whenever we need more specific quantum access such as **??**.

In the first part of the chapter we review the work done on the quantum speedup of Monte Carlo techniques found in this paper (Montanaro, 2015). Previous works (upon which (Montanaro, 2015) is based) on the same topic can be found in (Brassard et al., 2011), (Wocjan et al., 2009) and (Heinrich, 2002). In reviewing the paper, we will focus on the most important conceptual steps, while also presenting the statements of the main theorems. In the second part of the chapter, we present some of the most appealing applications of Monte Carlo quantum speedup, such as option pricing (Rebentrost et al., 2018) and the estimation of the total variation distance between probability distributions (Montanaro, 2015). Our main contributions are the statements of the theorems that allow for quantum speedup in both the option pricing and the total variation distance problems.

Notation We recap the notation that we will be using in the following. As already mentioned, the main goal of Monte Carlo methods is to estimate the expected value μ of a randomized algorithm A. The idea is that we have a random variable X taking values in a subset χ of $\mathbb{R}^d$, for some dimension d. The random variable X has probability density f and typically there is a function $\Phi : \chi \to \mathbb{R}_+$. Monte Carlo methods want to approximate:

$$\mu = \mathbb{E}[\Phi(X)] = \int_\chi \Phi(x) f(x) dx \tag{8.1}$$

They do so by producing an appropriate number N of i.i.d. samples, each corresponding to an independent execution of A. The sample mean $\tilde{\mu}$ is then used as an approximation of μ.

We now give a general definition for the sample mean. We will use the symbol $\tilde{\mu}$, which should not be confused with the above sample mean pertaining to the algorithm A. In general, for some random variable X_i, for $i = 1, \ldots, N$ the sample mean is defined as (Kennedy, 2016):

$$\tilde{\mu}_N = \frac{1}{N} \sum_{i=1}^{N} X_i \,. \tag{8.2}$$

The $\tilde{\mu}_N$ is a linear combination of the random variables X_i, and so it is itself a random variable. In the language of statistics, the $\tilde{\mu}_N$ is called an estimator. Note that:

- we do not know how many samples N we need to get a good estimate of μ;
- even worse, we still don't know whether this procedure leads to a good estimate of μ;
- if the estimate is good for some N we don't know how much good it is.

  Two fundamental theorems in probability theory help the way out. The law of large numbers ensures that the sample mean $\tilde{\mu}_N$ is a good approximation of μ for large enough N, while the central limit theorem states how close is $\tilde{\mu}_N$ to μ for a given value of N.

Theorem 8.1 (Law of large numbers)**.** *Let $X_1, \ldots, X_N$ be independent and identically distributed random variables. Assume that their expected value is finite and call it μ. Then, as N tends to infinity:*

$$\tilde{\mu}_N = \frac{1}{N} \sum_{i=1}^{N} X_i \to \mu, \, (a.s), \tag{8.3}$$

which means that:

$$P\left(\lim_{n \to \infty} \frac{1}{N} \sum_{k=1}^{N} X_k = \mu \right) = 1 \,. \tag{8.4}$$

Note that since all the X_i random variables have expected value μ the expected value of the sample mean $\tilde{\mu}_N$ is:

$$\mathbb{E}[\tilde{\mu}_N] = \frac{1}{N}\mathbb{E}[X_1 + \cdots + X_N] = \frac{1}{N}N\mu = \mu\,. \tag{8.5}$$

When this happens we say that $\tilde{\mu}_N$ is an unbiased estimator of μ. The law of large numbers tells us that $\tilde{\mu}_N$ converges to μ for N going to infinity. However in a computation we can not have N going to infinity: we must choose a (eventually large but) finite N. How close is $\tilde{\mu}_N$ to μ for a given value of N? The law of large numbers does not reply to that question (Kennedy, 2016). To get a first answer, suppose that X_i have finite variance and call it σ^2. The variance of the sample mean is then:

$$var(\tilde{\mu}_N) = \frac{1}{N^2}var(X_1 + \cdots + X_N) = \frac{1}{N^2}N\sigma^2 = \frac{\sigma^2}{N}\,. \tag{8.6}$$

This shows that the difference of $\tilde{\mu}_N$ from μ should be of order $\sigma/\sqrt{N}$ (Kennedy, 2016). The central limit theorem gives a more refined statement on the accuracy of the approximation (Kennedy, 2016).

Theorem 8.2 (Central limit theorem)**.** *Let X_i be a sequence of independent and identically distributed random variables, such that they have finite mean μ and finite variance σ^2. The random variable:*

$$\frac{1}{\sigma\sqrt{N}}\sum_{i=1}^{N}(X_i - \mu) \tag{8.7}$$

converges in distribution to a standard normal random variable. This means that:

$$\lim_{n\to\infty} P(a \le \frac{1}{\sigma\sqrt{N}}\sum_{k=1}^{N}(X_k - \mu) \le b) = \int_a^b \frac{1}{\sqrt{2\pi}}e^{-x^2/2}\,dx\,. \tag{8.8}$$

The central limit theorem can be used to construct confidence intervals for our estimate $\tilde{\mu}$, indeed the probability to estimate μ up to additive error ϵ is:

$$\begin{aligned} P(-\epsilon \le \tilde{\mu} - \mu \le \epsilon) &= P\Big(-\frac{\epsilon\sqrt{N}}{\sigma} \le \frac{(\tilde{\mu} - \mu)\sqrt{N}}{\sigma} \le \frac{\epsilon\sqrt{N}}{\sigma}\Big) && (8.9)\\ &\approx P(-\frac{\epsilon\sqrt{N}}{\sigma} \le Z \le \frac{\epsilon\sqrt{N}}{\sigma})\,, && (8.10)\end{aligned}$$

where Z is a standard normal random variable. We can find values of N such that the probability to have a good estimate $\tilde{\mu}_N$ up to fixed additive error ϵ is almost 1. Usual values for this probability are 95% or 99%. So for example if we want $P(-z_c \le Z \le z_c) = 0.99$ where $z_c = \epsilon\sqrt{N}/\sigma$ then $z_c = \epsilon\sqrt{N}/\sigma = 2.58$ because of the properties of the normal distribution. Estimating μ with additive error ϵ would require $N = 6.6564\,\sigma^2/\epsilon^2$ samples.

You can feel the power of Monte Carlo!! All of the above does not depend on the dimension of the sample space of the random variables X_i, but just on the number of repetitions N and on the variance σ^2.

This is amazing at first sight but we want to make two remarks. First note we don't know the value μ because we are trying to estimate it. How can we then know the value of σ^2, and use it to construct the confidence intervals? To address this problem we use another estimator: the sample variance. It is defined as:

$$s^2 = \frac{1}{N-1}\sum_{k=1}^{N}(X_i - \tilde{\mu})^2 \tag{8.11}$$

where the prefactor is chosen in such a way so that $\mathbb{E}[s^2] = \sigma^2$, i.e. s^2 is an unbiased estimator. One can prove that confidence intervals for μ can be built as above but with σ replaced by s (Kennedy, 2016).

Second we highlight the main flaw of the Monte Carlo procedure. We saw that estimating μ up to additive error ϵ with 99% success probability requires $n = O(\sigma^2/\epsilon^2)$ repetitions, independently on the dimension of the sample space. This is remarkable but not so efficient. It means that if we want to maintain the confidence at 99% and we want to decrease the additive error ϵ by a factor of 10 we need to increase the number of iteration by a factor of 10^2. Imagine if we want to estimate μ up to four digits. We would need to run A more than 100 million times (Montanaro, 2015).

8.1 Monte Carlo with quantum computing

Here is where quantum computing comes to help. The number of uses of A can be reduced almost quadratically beyond the classical bound (Montanaro, 2015). The result is based on amplitude estimation.

We first show the quadratic advantage for an algorithm A whose output is bounded between 0 and 1. This speedup will be smartly used to speedup more general classes of algorithms. This section is also meant to give insights on why quantum speedup happens in the first place.

Now we will make use of the notation introduced at the beginning of the chapter. Without loss of generality, we assume that A is a quantum algorithm operated on an n qubit quantum register, whose initial state is the $|0^{\otimes n}\rangle$ state. We assume that A makes no measurements until the end of the algorithm, when $k \leq n$ qubits are measured in the computational basis. The outcome of the measurement $x \in \{0,1\}^k$ is then plugged in a function $\Phi : \{0,1\}^k \to [0,1]$. We call $\nu(A)$ the random variable representing the output of A. Our aim is to find an estimate of $\mathbb{E}[\nu(A)]$. As usual, we also assume to have access to the inverse of the unitary part of the algorithm[1], which we write as A^{-1}.

[1] This is needed to run amplitude estimation

The algorithm A, as a classical randomized algorithm, is built in such a way that when we run it k times, we get some random $\Phi(x_1), \dots, \Phi(x_k)$ with probabilities $|\alpha_{x_i}|^2$ (for some, in general, complex α_{x_i}).

That is to say values of the random variable $\nu(A)$ are distributed according to the probabilities $p_x = |\alpha_x|^2$. We will see that with quantum computers, there is a smarter way to estimate $\mathbb{E}[\nu(A)]$, instead of repeatedly sampling from A? Using a quantum computer we can assume that A is now a quantum algorithm (by taking the classical circuit, making it reversible, and then obtaining a quantum circuit for it). Then, we can use various tricks to encode the value of $\nu(A)$ in the amplitude of a quantum register, and then estimate it using amplitude estimation.

8.2 Bounded output

In this section we will show a direct application of the amplitude estimation theorem, and we will formulate everything using the notation found in (Montanaro, 2015). The procedure when the output of A is bounded between 0 and 1 is the following. Instead of doing the measurement at the end of the circuit, we add an ancillary qubit and perform the unitary operation W on $k+1$ qubits defined as follows:

$$W|x\rangle|0\rangle = |x\rangle(\sqrt{1-\Phi(x)}|0\rangle + \sqrt{\Phi(x)}|1\rangle) \tag{8.12}$$

This is a controlled operation which in general requires to first encode $\Phi(x)$ in a register. This is done by performing k subsequent controlled rotations for some specific angle on the $k+1$th qubit for each of the k controlling qubits. Then it is easy to rotate back the k register to $|x\rangle$ (more on this can be found in (Duan et al., 2020)).

Assuming that we know how to do W, when A is run with its final measurement replaced with operation W, we would formally end up in a state of the form:

$$|\psi\rangle = (I\otimes W)(A\otimes I)|0^n\rangle|0\rangle = \sum_x \alpha_x|\psi_x\rangle|x\rangle(\sqrt{1-\Phi(x)}|0\rangle+\sqrt{\Phi(x)}|1\rangle)\,. \tag{8.13}$$

Why is this state useful?

When the sum over x is developed, it turns out that $|\psi\rangle$ is such that it can be viewed as a superposition of two orthogonal states. A “good” state, $|\Psi_G\rangle$ which is a superposition of all the states in the expansion whose qubit sequence ends up with a 1 (i.e. the ancilla qubit is in state $|1\rangle$ and a “bad” state $|\Psi_B\rangle$, which is a superposition of all the other states. Unfortunately there is no ugly state, otherwise we would have been screening a spaghetti western cult. The good and bad states are orthogonal to each other and thus they can be viewed as spanning two orthogonal sub-spaces in the Hilbert space, which without fantasy we’ll call good and bad sub-spaces.

Projecting $|\psi\rangle$ onto the good subspace is equivalent to find the amplitude of the good state, which for construction is the expected value of $\nu(A)$ (exaclty what we wanted!!). Indeed calling P the projector onto the good subspace, which is in our case $P = I \otimes |1\rangle\langle 1|$, we have that:

$$\langle\psi|P|\psi\rangle = \sum_x |\alpha_x|^2 \Phi(x) = \mathbb{E}[\nu(A)]\,. \tag{8.14}$$

Notice that the bound on the output of A between 0 and 1 and the form of the operation W leads to a normalized state $|\psi\rangle$. This means that we can write:

$$|\psi\rangle = \sin\theta|\Psi_G\rangle + \cos\theta|\Psi_B\rangle \tag{8.15}$$

for some angle θ. We can restate our problem as follows: we want to estimate the amplitude squared of the good state, i.e. $\sin^2(\theta)$. Indeed we created a circuit that create a state where the expected value of $\nu(A)$ is now encoded in that amplitude of the quantum state, more specifically it is encoded in the probability of measuring 1 in an ancilla qubit. The latter is estimated with amplitude estimation, see theorem 5.4. This algorithm actually returns an estimate of the angle θ for which we can derive an estimate of $\sin^2(\theta)$, which in turns return the expectation of the random variable $\mathbb{E}[\nu(A)]$.

Let's recap how to use this algorithm. Amplitude estimation takes as input a unitary operator on $n+1$ qubits such that:

$$|\psi\rangle = \sqrt{1-a}|\Psi_B\rangle|0\rangle + \sqrt{a}|\Psi_G\rangle|1\rangle \tag{8.16}$$

The final qubit acts as a label to distinguish good states from bad states.

Then it applies for a number of times t the following sequence of projections: first apply $V = I - 2P$ where P is some projector and then $U = 2|\psi\rangle\langle\psi| - I$. The form of P depends in general on the good state (in our case $P = I \otimes |1\rangle\langle 1|$). The projector U could be seen as a reflection through $|\psi\rangle$ and V as a reflection through the bad state $|\Psi_B\rangle$ in a plane spanned by the good and the bad states (de Wolf, 2019). The output of amplitude estimation is an estimate $\tilde{a}$ of $a = \langle\psi|P|\psi\rangle = \sin^2(\theta)$.

U can also be written as AS_0A^{-1} where S_0 changes the phase from $+1$ to -1 of all the basis states but the $|0^n\rangle$ state. This is why before we required to have access to the inverse of A. V is instead the operation of changing the phase of all the basis states of the good sub-space from $+1$ to -1. This operation is usually done by setting a target qubit to $1/\sqrt{2}(|0\rangle - |1\rangle)$ as we have:

$$U_f|x\rangle 1/\sqrt{2}\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = (-1)^{f(x)}|x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \tag{8.17}$$

for any function $f(x)$ with binary values 0 or 1. The target qubit will always be just one and applying U_f is an eigenstate. So we can ignore its presence and consider U_f as acting only on the first register.

We are then ready to state the quantum algorithm to estimate $\mathbb{E}[\nu(A)]$ when the output of A is bounded between 0 and 1.

```
**Algorithm** Monte Carlo with bounded output
**Require:** A quantum algoritm $A$ on $n$ qubits initialized to $|0^n\rangle$ such that 0
    $\nu(A) \leq 1$, integer $t$, real $\delta > 0$.
**Ensure:** An estimate of $\mathbb{E}[\nu(A)]$.

If necessary, modify $A$ such that it makes no measurement until the end
    the algorithm; operates on initial input state $|0^n\rangle$; and its final measureme
    is a measurement of the last $k \leq n$ of these qubits in the computational ba
Let $W$ be the unitary operator on $k+1$ qubits defined by
    $W|x\rangle|0\rangle = |x\rangle\left(\sqrt{1-\phi(x)}|0\rangle + \sqrt{\phi(x)}|1\rangle\right.$
    where each computational basis state $x \in \{0,1\}^k$ is associated with a r
    number $\phi(x) \in [0,1]$, such that $\phi(x)$ is the value output by $A$ when m
    surement $x$ is recieved.
Repeat the following step $O(\log(1/\delta)$ times and output the median of t
    results: Apply $t$ iterations of amplitude estimation, setting $|\psi\rangle = (I$
    $W)(A \otimes I)|0^{n+1}\rangle$, $P = I \otimes |1\rangle\langle 1|$.
```

Applying amplitude estimation together with the powering lemma C.1, one can prove the following theorem valid for algorithm **??**.

Theorem 8.3 (Quantum Monte Carlo with bounded output)**.** *Let $|\psi\rangle$ as in Eq.*(8.13) *and set $U = 2|\psi\rangle\langle\psi| - I$. Algorithm in figure* **??** *uses $O(\log(1/\delta))$ copies of $|\psi\rangle = A|0^n\rangle$, uses U $O(t\log(1/\delta))$ times, and outputs an estimate $\tilde{\mu}$ such that*

$$|\tilde{\mu} - \mathbb{E}[\nu(A)]| \leq C\left(\frac{\sqrt{\mathbb{E}[\nu(A)]}}{t} + \frac{1}{t^2}\right)$$

with probability at least $1-\delta$, where C is a universal constant. In particular for any fixed $\delta > 0$ and any ϵ such that $0 \leq \epsilon \leq 1$, to produce an estimate $\tilde{\mu}$ such that with probability at least $1-\delta$, $|\tilde{\mu} - \mathbb{E}[\nu(A)]| \leq \epsilon\mathbb{E}[\nu(A)]$ it suffices to take $t = O(1/(\epsilon\sqrt{\mathbb{E}[\nu(A)]}))$. To achieve $|\tilde{\mu} - \mathbb{E}[\nu(A)]| \leq \epsilon$ with probability at least $1-\delta$ it suffices to take $t = O(1/\epsilon)$.

We already described the main idea to get quantum speedup: encode the expected value of $\nu(A)$ in a suitably defined amplitude of the quantum register and then estimate it. The error bound of theorem 8.3 is the error bound of the amplitude estimation algorithm 5.4. The same applies to the success probability which, in this 5.4 formulation of the amplitude estimation theorem, is $8/\pi^2$. This can be improved to $1-\delta$ with the powering lemma C.1. That is why, for example, U is used $O(t\log(1/\delta))$ times: t times by amplitude estimation which is repeated $\log(1/\delta)$ to apply the powering lemma. To get the absolute error to ϵ we just need to set $t = O(1/\epsilon)$, and to obtain a relative error we set $t = O(1/(\epsilon\sqrt{\mathbb{E}[\nu(A)]}))$.

8.3 Bounded ℓ_2 norm

The algorithm described by theorem 8.3 is limited by the requirement of having an output value bounded between 0 and 1. We want more. We want a quantum algorithm to compute the expected value of any random variable. We start by relaxing the requirement of having bounded output, and just requiring bounded ℓ_2 norm.

To arrive at this result, the first step is to show quantum speedup for the class of algorithms whose output value is positive and has bounded l_2 norm. This means that there is a bound on $\|\nu(A)\|_2 = \sqrt{\mathbb{E}[\nu(A)^2]}$. The key idea is the following: given an algorithm A belonging to this class, we consider its possible output value as the sum of numbers belonging to intervals of the type $[0,1)$, $[1,2)$, $[2,4)$, $[4,8)$ and so on. Notice that besides the first interval ($[0,1]$), all the others can be compactly written as $[2^{l-1}, 2^l]$ for $l = 1, ..., k$. If we divide all of these intervals by 2^l we would get k intervals bounded between 0 and 1 (actually 1/2 and 1). In this way we can apply algorithm 8.3 for the interval $[0,1]$ and for all these other k intervals. If we multiply each of the k estimates we get by 2^l (for $l = 1, ..., k$) and add them together with the estimate for the $[0,1]$ interval, we would get an estimate for $\mathbb{E}[\nu(A)]$.

You may have noticed that the index l goes from 1 to k and wondered what is this k? If not, we wonder for you. Set for example $k = 2$ and imagine that the expected value of the algorithm is 16. Surely we can not obtain 16 as a sum of three numbers in the intervals $[0,1]$, $[1,2]$, $[2,4]$. So how must k be chosen? And even if k is chosen properly, how we can be sure that what is left in this approximation is negligible? We will see in a moment that assuming a bound on the l_2 norm of $\nu(A)$ gives us a way to solve this problem elegantly.

Before going on, we introduce a useful notation. Given any algorithm A we write as $A_{<x}$, $A_{x,y}$, $A_{\geq y}$ the algorithms defined by executing A to produce a value $\nu(A)$ and:

- $A_{<x}$: if $\nu(A) < x$, output $\nu(A)$ otherwise output 0;
- $A_{x,y}$: if $x \leq \nu(A) < y$, output $\nu(A)$ otherwise output 0;
- $A_{\geq y}$: if $\nu(A) \geq y$, output $\nu(A)$ otherwise output 0.

Moreover whenever we have an algorithm A and a function $f : \mathbb{R} \to \mathbb{R}$, we write as $f(A)$ the algorithm produced by evaluating $\nu(A)$ and then computing $f(\nu(A))$.

We can now state the second quantum algorithm which estimates the mean value of $\nu(A)$ with bounded ℓ_2 norm.

One can prove the following theorem.

Theorem 8.4 (Quantum algorithms with bounded l2 norm). *Let $|\psi\rangle$ as in Eq.*(8.13)*, $U = |\psi\rangle\langle\psi| - 1$. Algorithm in figure 8.1 uses $O(\log(1/\epsilon)\log\log(1/\epsilon))$ copies of $|\psi\rangle$, uses U $O((1/\epsilon)\log^{3/2}(1/\epsilon)\log\log(1/\epsilon))$ times, and estimates*

Algorithm Monte Carlo with bounded ℓ_2 norm

```
Require: A quantum algoritm $A$ such that $\nu(A) \geq 0$, an accuracy $\epsilon < 1/2$.
Ensure: An estimate of $\mathbb{E}[\nu(A)]$.

Set $k = \log_2(1/\epsilon)$, $t_0 = \frac{D\sqrt{\log_2 1/\epsilon}}{\epsilon}$, where $D$ is a universal constant to be chosen later.
Use algorithm 1 with $t = t_0$, $\delta = 1/10$ to estimate $\mathbb{E}[\nu(A_{0,1})]$. Let the estimate be $\widetilde{\mu}_0$.
For $l = 1, \ldots, k$: Use algorithm 1 with $t = t_0$, $\delta = 1/(10k)$ to estimate $\mathbb{E}[\nu(A_{2^{l-1},2^l}/2^l)]$. Let the estimate be $\widetilde{\mu}_l$.
Output $\widetilde{\mu} = \widetilde{\mu}_0 + \sum_{l=1}^{k} 2^l \widetilde{\mu}_l$.
```

Figure 8.1: Quantum algorithm for Monte Carlo (bounded l2 norm)

$\mathbb{E}[\nu(A)]$ *up to additive error* $\epsilon(||\nu(A)|| + 1)^2$ *with probability at least* $4/5$.

We will give here a sketch of the proof for this theorem. The curious can check it on (Montanaro, 2015). The resource bounds follow from the resource bounds of algorithm 8.3, multiplied by the number of times we use it. Indeed $\delta = \Omega(1/log(1/\epsilon))$ and we are using algorithm 8.3 $O(\log(1/\epsilon))$ times. So the overall number of copies of $|\psi\rangle$ in all of the uses of algorithm 8.3 is $O(\log(1/\epsilon)\log\log(1/\epsilon))$ and the total number of uses of U is $O((1/\epsilon)\log^{3/2}(1/\epsilon)\log\log(1/\epsilon))$.

To estimate the total error (supposing that every use of algorithm in theorem 8.3 succeed) we write:

$$\mathbb{E}[\nu(A)] = [\nu(A_{0,1})] + \sum_{l=1}^{k} 2^l \mathbb{E}[\nu(A_{2^{l-1},2^l})/2^l] + \mathbb{E}[\nu(A_{\geq 2^k})]\,. \tag{8.18}$$

Now substituting this expression in the one for the error and using the triangle inequality we have that:

$$|\tilde{\mu} - \mathbb{E}[\nu(A)]| \leq |\tilde{\mu}_0 - \mathbb{E}[\nu(A_{0,1})]| + \sum_{l=1}^{k} 2^l |\tilde{\mu}_l - \mathbb{E}[\nu(A_{2^{l-1},2^l})/2^l]| + \mathbb{E}[\nu(A_{\geq 2^k})]\,. \tag{8.19}$$

You can see that the last term is a term that we did not estimate with 8.3. This is exactly the leftover we were talking about some lines above. What the last equation says is that when choosing k as in algorithm 8.1, this term is bounded by the l_2 norm squared of $\nu(A)$. Indeed denoting with $p(x)$ the probability that A outputs x we have that:

$$\mathbb{E}[\nu(A_{\geq 2^k})] = \sum_{x \geq 2^k} p(x)x \leq \frac{1}{2^k}\sum_x p(x)x^2 = \frac{\|\nu(A)\|^2}{2^k}, . \tag{8.20}$$

Getting the error bound of theorem 8.4 is not really interesting from the conceptual point of view. It is just a matter of math and the usage of previous results. Indeed, using the error bound obtained for algorithm in theorem 8.3 for $|\tilde{\mu}_0 - \mathbb{E}[\nu(A_{0,1})]|$ and all the $|\tilde{\mu}_l - \mathbb{E}[\nu(A_{2^{l-1},2^l})/2^l]|$, Cauchy-Schwartz, and other techqnieus (refer to (Montanaro, 2015) for the full calculations), one could finally write that:

$$|\tilde{\mu} - \mathbb{E}[\nu(A)]| = \epsilon\left(\frac{C}{D}\left(1 + \frac{5}{D} + 2||\nu(A)||_2\right) + ||\nu(A)||_2^2\right) \tag{8.21}$$

which for a sufficiently large constant D is upper bounded by $\epsilon(||\nu(A)||_2 + 1)^2$. Here C another constant that comes from the error bound of algorithm of theorem 8.3, wich in turns comes from the constant factors of amplitude estimation.

Finally we comment on the success probability: it is at least 4/5, when one uses the union bound, i.e. theorem C.1 on the success probability of all the uses of theorem 8.3. Note that we can exploit the powering lemma to increase this success probability up to $1-\delta$ for any δ, by repeating this algorithm $O(\log(1/\delta)$ times and taking the median.

8.4 Bounded variance

We are ready to demonstrate quantum speedup for the most general case: algorithms whose random output have bounded variance. In this case one obtains quantum speedup combining the classical Chebyeshev inequality and the use of quantum speedup of algorithm 8.1. For clarity, we recall that the bound on the variance of $\nu(A)$ means that $Var(\nu(A)) = \mathbb{E}[(\nu(A) - \mathbb{E}[\nu(A)])^2] \leq \sigma^2$.

To get to our desired result, we will also need to scale and shift the random variables of interest. Starting with a random variable X distributed according to some distribution with mean μ_x and standard deviation σ_x, whenever we scale the random variable by a factor λ, we obtain a random variable $Z = \lambda X$ with a new distribution with mean $\mu_z = \lambda\mu_x$ and standard deviation $\sigma_z = \lambda\sigma_x$. When we shift the random variable by a scalar k, we obtain a random variable $Z = X + k$ with a new distribution with mean $\mu_z = \mu_x + k$ and standard deviation $\sigma_z = \sigma_x$.

The first step of our algorithm is to run the algorithm $A' = A/\sigma$ obtained by scaling A with $\lambda = 1/\sigma$. For what we said above $\nu(A')$ will be a random variable with mean and standard deviation bounded by $\mu' \leq \mu/\sigma$ and $\sigma' \leq 1$ respectively. Having a standard deviation of order unity means that the output $\widetilde{m}$ of a run of algorithm A' is with high probability pretty close to the actual mean μ'. This is because of Chebyeshev inequality, i.e. theorem C.4:

$$P(|\nu(A') - \mu'| \geq 3) \leq \frac{1}{9}\,. \tag{8.22}$$

Therefore we can assume with high confidence that $|\widetilde{m} - \mu'| \leq 3$. The second step is to consider algorithm B, which is produced by executing A' and shift it by subtracting $\widetilde{m}$. The random variable $\nu(B)$ has a bound on the ℓ_2 norm. Indeed:

$$\begin{aligned} ||\nu(B)||_2 &= \mathbb{E}[\nu(B)^2]^{1/2} = \mathbb{E}[((\nu(A') - \mu') + (\mu' - \widetilde{m}))^2]^{1/2} \\ &\leq \mathbb{E}[(\nu(A') - \mu')^2]^{1/2} + \mathbb{E}[(\mu' - \widetilde{m})^2]^{1/2} = \sigma' + \mathbb{E}[(\mu' - \widetilde{m})^2]^{1/2} \leq 4. \end{aligned} \tag{8.23}$$

This bound can also be stated as $||\nu(B)/4||_2 \leq 1$.

This means that the expected value of $\nu(B)/4$ could be estimated with algorithm 8.1. Actually algorithm 8.1 also requires a positive $\nu(B) \geq 0$. We will estimate $\mathbb{E}[\nu(B_{\geq 0})/4]$ and $\mathbb{E}[\nu(-B_{<0})/4]$ using algorithm 8.1 with accuracy $\epsilon/(32\sigma)$ and failure probability 1/9. Notice that both $\nu(B_{\geq 0})/4$ and $\nu(-B_{\leq 0})/4$ are positive and the bound $||\nu(B)/4||_2 \leq 1$ implies $||\nu(B_{<0})/4||_2 \leq 1$ and $||\nu(B_{\geq 0})/4||_2 \leq 1$.

Algorithm Monte Carlo with bounded variance

Require: A quantum algorithm A such that $Var(\nu(A)) \leq \sigma^2$ for some known σ, an accuracy ϵ such that $\epsilon < 4\sigma$.
Ensure: An estimate of $\mathbb{E}[\nu(A)]$.

```
Set $A' = A/\sigma$.
Run $A'$ once and let $\widetilde{m}$ be the output.
Let $B$ be the algorithm produced by executing $A'$ and subtracting $\widetilde{m}$.
Apply algorithm 2 to algorithms $-B_{<0}/4$ and $B_{\geq 0}/4$ with accuracy $\epsilon/(32\sigma)$
  and failure probability 1/9, to produce estimates $\widetilde{\mu}^-$, $\widetilde{\mu}^+$ of $\mathbb{E}[\nu(-B_{<0}/4)]$
  and $\mathbb{E}[\nu(B_{\geq 0}/4)]$, respectively.
Set $\widetilde{\mu} = \widetilde{m} - 4\widetilde{\mu}^- + 4\widetilde{\mu}^+$.
Output $\sigma\widetilde{\mu}$.
```

This algorithm has a quadratic speedup over the classical Monte Carlo method.

Theorem 8.5 (Quantum Monte Carlo with bounded variance - additive error)**.** *Let* $|\psi\rangle$ *as in Eq.*(8.13)*,* $U = 2|\psi\rangle\langle\psi| - I$*. Algorithm* **??** *uses* $O(\log(\sigma/\epsilon)\log\log(\sigma/\epsilon))$ *copies of* $|\psi\rangle$*, uses* U $O((\sigma/\epsilon)\log^{3/2}(\sigma/\epsilon)\log\log(\sigma/\epsilon))$ *times and estimates* $\mathbb{E}[\nu(A)]$ *up to additive error* ϵ *with success probability at least* 2/3*.*

The additive error for this theorem is ϵ because we required accuracy $\epsilon/32\sigma$ for both uses of algorithm 8.1. Indeed, this implies that both the estimates we would get are accurate up to $(\epsilon/32\sigma)(||\nu(B_{\geq 0}/4)||_2+1)^2 \leq (\epsilon/32\sigma)(1+1)^2 = \epsilon/8\sigma$. Now we just multiply by a 4 factor the estimates of $\mathbb{E}[\nu(B_{\geq 0})/4]$ and $\mathbb{E}[\nu(B_{<0})/4]$ to get the estimates of $\mathbb{E}[\nu(B_{\geq 0})]$ and $\mathbb{E}[\nu(B_{<0})]$. The error then gets $4\epsilon/8\sigma = \epsilon/2\sigma$. Combining these two errors, one has a total additive error for the estimate of A' given by ϵ/σ. Since $A = \sigma A'$ the error for A is exactly $\epsilon = \sigma(\epsilon/\sigma)$.

The success probability is at least 2/3 when using the union bound (theorem C.1) on the success probability of the uses of algorithm 8.1 and the success probability for $\widetilde{m}$ to be close to μ'. Also in this case we can exploit the powering lemma to increase this probability up to $1 - \delta$ for any δ by repeating this algorithm $O(\log(1/\delta)$ times and take the median.

8.5 Applications

We have developed some tools that show how Monte Carlo methods can run faster in a quantum computer than in a classical computer. As it happens for most of the algorithm found in this book, all of the above is an advantage at the level of the mathematical formulation of the algorithm and it is detached from the problem at hand. Indeed, Monte Carlo methods are used to compute the expectation value of a function of random variables. A little more effort is needed to formulate a specific problem that is currently solved by classical Monte Carlo methods and to exploit the quantum speedup. When a problem is identified, one need to adapt the general quantum algorithms that we described above to the peculiar form of the problem. As we have already highlighted, two main ingredients are required: the creation of a quantum state distributed according to the right probability distribution, and a query to an oracle function, what we called ν, in order to perform amplitude estimation.

8.5.1 Pricing of financial derivatives

A wide usage of Monte Carlo methods is devoted to the problem of pricing of financial derivatives. Through the financial market, buyers and sellers exchange so called financial market instruments. These can be divided into two species. The “underlying” such as stocks, bonds, commodities etc., and their “derivatives”. These are usually claims that, for example, promise some payment in the future contingent on an underlying stock’s behavior. An example of a derivative is a contract agreeing to pay off the difference between a stock price and some agreed future price. Given our ignorance on the dynamics of price of stocks, we can model it as a random process. This randomness filters through to the price of derivatives, they appear to be random too, so that a certain amount of risk seems to be involved when agreeing on some derivative. But still, math is a powerful tool and can suggest a way to cope with this randomness. Sometimes math can give us nice analytic solutions, sometimes we need to use approximate solutions. Monte Carlo integration is such a technique for the pricing of financial derivatives. We consider a stock driven by a single Brownian motion, and the European call option (Rebentrost et al., 2018). One of the simplest framework to model option pricing is the so called Black-Scholes-Merton (BSM) model. This model considers a single benchmark asset (“stock”), the price of which is driven by a Brownian motion. In addition, it assumes a risk-free investment into a bank account (“bond”).

A Brownian motion W_t is a stochastic process characterized by the following properties: 1) $W_0 = 0$; 2) W_t is continuous; 3) W_t has independent increments; 4) $W_t - W_s \sim N(0, t-s)$ for $t > s$. Here $N(\mu, \sigma)$ is the normal distribution with mean μ and standard deviation σ. We denote by $\mathbb{P}$ the probability measure under which W_t is a Brownian motion.

In modeling our problem, we have a genuine uncertainty on the evolution of the price of the asset over time, so we model it as a stochastic process, which depends on the value of a Brownian motion. More formally the price dynamics is governed by a stochastic differential equation:

$$dS_t = S_t \alpha \, dt + S_t \sigma \, dW_t$$

where α is called drift, σ volatility and dW_t is a Brownian increment. α dictates a deterministic rate of change of the stock price over time, while the volatility σ indicates a stochastic variation of the price. Indicating with S_0 the initial stock price, the formal solution of this equation is given by:

$$S_t = S_0 e^{\sigma W_t + (\alpha - \sigma^2/2)t} \tag{8.24}$$

which is to say, that the value of the stock price over time is a stochastic process built out of the Brownian process W_t. We called this solution formal, in the sense that in it infinitely many realizations are included: at every time the stock price takes a value that depends on the value of W_t, which is a random variable. This leads to infinitely many different trajectories of the price over time. Of course trying to take into account all the different trajectories becomes soon an intractable task.

Among the different derivatives we consider options. We take one of the simplest option: the European call option. This option gives the owner of the option the right to buy the stock at a fixed moment in time $T \geq 0$, the maturity date,for a pre-agreed strike price K.

A buyer is interested in the following problem: how can he know in advance if it is convenient to enter the contract? To answer this question, we need a method to price the option payoff. This is done by defining the payoff function. For the European call option it is defined as:

$$f(S_T) = \max\{0, S_T - K\} \, . \tag{8.25}$$

It clearly depends on the stock price at the maturity date T only. If the strike price K is lower than the stock price S_T at the maturity date, their difference is the amount of money one could earn, the payoff, by buying the stock today. The last statement is not exactly correct, it comes with a little subtlety. We have to take into account that also the bond evolves over time. In our model we will make the simpplyfing assumption that the bond price follows a deterministic equation: \begin{equqtion} dB_t = rB_t,dt \end{equation} with a constant r. We can informally view this equation as a formulation of the fact that as the time passes, the value of money changes. If I have 1 euro/dollar/yen now, it will have a different value in the future. But the pricing of an option must be performed in the present, or the payoff, that clearly depends on the stock price at the future time T, should be discounted to its present value. One need to estimate the amount of money he/she could make by entering the contract **today**. The discount factor in this model is just e^{-rT}, the inverse of the growth factor for the value of money over time e^{rT}. This is clear by the above equation, as its solution is given by $B_T = e^{rT}B_0$. For this reason r is also called interest rate.

We are almost done, but there is another subtlety. Indeed we require that pricing should be performed under a probability measure that shall not allow for arbitrage. One can show that this measure, usually denoted by $\mathbb{Q}$, is the one for which the drift of the stock price α is equal to the interest rate r. This may sound a little confusing, so instead of going into all the details of arbitrage, we will just comment on the main property that $\mathbb{Q}$ satisfies, as this is sufficient to get the main idea:

$$S_0 = e^{-rT}\mathbb{E}_{\mathbb{Q}}[S_T] \tag{8.26}$$

This is usually referred to as the Martingale property: the expectation value (under $\mathbb{Q}$) of the stock price at time T multiplied by the discount factor, is the present stock price S_0. This means that at any time, and on average (!!), the price of the stock can not increase (decrease) above (below) the market rate r. Or informally we might say that the expected value of the stock price, on average (!!), will not change over time, so that there is no chance for the buyer or seller of any option to earn money out of nothing. The price dynamics under the arbitrage probability measure is described by:

$$dS_t = S_t r dt + S_t \sigma d\tilde{W}_t \tag{8.27}$$

where $\tilde{W}_t$ is again a Brownian motion under the probability measure $\mathbb{Q}$. (One can define $\tilde{W}_t$ out of W_t and show that it is a Brownian motion *da verificare questa cosa!*). The whole problem of option pricing is then the evaluation of the following quantity:

$$\Pi = e^{-rT}\mathbb{E}_{\mathbb{Q}}[f(S_T)] \tag{8.28}$$

This is the expected value of the option payoff discounted to its present value under the arbitrage free probability measure. The function $f()$ is now any generic payoff function for the given option. This is in essence the BSM model. When $f(S_T)$ is the payoff function of the European call option that we saw above, one can find an analytical solution to Π. Notice that this can not always be true. For complicated payoff functions or more complex asset price dynamics, analytical methods are useless and one often resorts to Monte Carlo methods, because the task is to estimate an expected value of some function of random variables. So what can be exactly different with respect to European option? European options' payoff function depends on the asset price at a single future time T. Given another such payoff function, if it is nonlinear or there are different asset prices that are assumed to be correlated, one can not find an analytical solution. Another complication enters when the option can be claimed at different times and not just at a single future time. American options, for example, allow the buyer to exercise the option at any point in time between the option start and the option maturity. These options are priced using Monte Carlo methods. When the pricing of an option requires Monte Carlo methods, we expect to use all of the quantum machinery developed above to solve this problem in the quantum realm, speeding up the computation.

8.5.1.1 European option pricing

We consider a stock driven by a single Brownian motion and the European option payoff function. Even if in this case Π has an analytical solution, we will use this case to simplify the discussion and as a first representative example. We can write:

$$\Pi = e^{-rT}\mathbb{E}_{\mathbb{Q}}[v(W_T)] \tag{8.29}$$

where W_T is the Brownian motion at time T and $v(x)$ is derived from the payoff function $f(x)$ ($v(x)$ plays the role of ν in the previous sections). As we saw, W_T is a Gaussian random variable $N(0,T)$ with zero mean and T variance, and with probability density given by:

$$p_T(x) = \frac{1}{\sqrt{2\pi T}}e^{-\frac{x^2}{2T}} \tag{8.30}$$

To use our quantum algorithms, we need to prepare a state which is a superposition with the square root of these probabilities at each value x, which are infinitely many. In real life, we will have a finite register of qubits and we can only prepare approximately this quantum state because we can only take a finite set of values x_i. To do so, we take the support of the density distribution from $]-\infty,+\infty[$ to $[-x_{max}, x_{max}]$ and discretize this interval with 2^n points, where n is the integer number of qubits. We can define the discretization points as $x_j := -x_{max} + j\Delta x$, with $\Delta x = 2x_{max}/(2^n-1)$ and $j = 0, \dots, (2^n-1)$. We then define the probabilities $p_j = p_T(x_j)/C$ where the normalization $C = \sum_{j=0}^{2^n-1} p_T(x_j)$ is required so to have 2^n probabilities that sum up to 1. Through an oracle that gives us quantum sampling access **??** to the probability distribution, we can run a quantum circuit G such that we can prepare:

$$G|0^n\rangle = \sum_{j=0}^{2^n-1}\sqrt{p_j}|j\rangle \tag{8.31}$$

Note that we are implicitly identifying $|j\rangle$ with $|x_j\rangle$, the circuit G prepares the superposition state, and thus it plays the role of A found in the previous sections.

Having the right superposition state, we need to compute the function and perform the controlled rotation. For the European call option, the payoff function is:

$$v_{euro}(x) = \max\{0, S_0e^{\sigma x+(r-1/2\sigma^2)T} - K\} \tag{8.32}$$

At the discretization points of the Brownian motion we set $v(j) := v_{euro}(x_j)$. One can find a binary approximation to this function over n bits, i.e. $\tilde{v}(j) : \{0,1\}^n \to \{0,1\}^n$, because remember that now j is a sequence of n bits, and we need $\tilde{v}(j)$ to be encoded as a binary expansion in a quantum register, thus to be another sequence of bits. We can then implement the operation:

$$|j\rangle|0^n\rangle \to |j\rangle|\tilde{v}(j)\rangle \tag{8.33}$$

thanks to an auxiliary register of n qubits, initialized in the state $|0^n\rangle$. Using the arithmetic model 3.1, we can then apply the controlled rotation W to a further ancillary qubit to end up with:

$$|j\rangle|\tilde{v}(j)\rangle \to |j\rangle|\tilde{v}(j)\rangle\left(\sqrt{1-\tilde{v}(j)}|0\rangle + \sqrt{\tilde{v}(j)}|1\rangle\right) \tag{8.34}$$

Now we can inverse the circuit that got us from $|j\rangle|0^n\rangle$ to $|j\rangle|\tilde{v}(j)\rangle$ so that the auxiliary register goes back to $|0^n\rangle$ and we can omit it. Probably you have seen this in many other places, and you will use this fact again.

Note that now, since we discretized the probability distribution and we approximated the function to $\tilde{v}$, an estimate $\hat{\mu}$ of the option price is already characterized by an error ν, that can be shown to be $O(2^{-n})$ (Rebentrost et al., 2018). This makes sense, as the more qubits we have, the better the approximation will be.

Everything is set up to apply theorem 8.5, except for the fact that we need to show that the payoff function have bounded variance. One can show that the European call option has bounded variance $\mathbb{V}_{\mathbb{Q}}[f(S_T)] \leq \lambda^2$, where $\lambda^2 = O(poly(S_0, e^{rT}, e^{\sigma^2 T}, K))$. Thus we know that through theorem 8.5, using $O(\log(\lambda/\epsilon)\log\log(\lambda/\epsilon))$ copies of the state $|\psi\rangle = (\mathbb{I} \otimes W)(G \otimes \mathbb{I})|0^n\rangle|0\rangle$ and $O((\lambda/\epsilon)\log^{3/2}(\lambda/\epsilon)\log\log(\lambda/\epsilon))$ applications of $U = 2|\psi\rangle\langle\psi| - I$ to provide an estimate for μ up to additive error ϵ with success probability at least $2/3$. As usual, the success probability can be increased to $1-\delta$ for any δ by taking $O(\log(1/\delta))$ repetitions C.1. The total error is then

$$|\hat{\mu} - \mathbb{E}_{\mathbb{Q}}[f(S_T)]| \leq \epsilon + \nu \tag{8.35}$$

which takes into account the additive error ϵ from theorem 8.5 and the discretization error ν. Discounting $\hat{\mu}$ then retrieves an estimation of the option price $\hat{\Pi}$ up to additive error $\nu + \epsilon$.

8.5.1.2 Asian option pricing

As we already pointed out, the European call option can be priced analytically in the BSM framework, and thus Monte Carlo methods are not required. Another type of option is the so called Asian option, whose payoff depends on the average asset price before the maturity date. The payoff function for an Asian call option is defined as:

$$f(A_T) = \max\{0, A_T - K\} \tag{8.36}$$

where K is the strike price and T is the maturity date. The definition of A_T is not unique. Arithmetic mean Asian option value is defined via:

$$A_T^{arith} = \frac{1}{L}\sum_{l=1}^{L} S_{t_l} \tag{8.37}$$

while the geometric mean option value is defined via:

$$A_T^{geo} = \exp\frac{1}{L}\sum_{l=1}^{L} \log S_{t_l} \tag{8.38}$$

for predefined time points $0 < t_1 < ... < t_L \leq T$ with $L \geq 1$. Note that if you develop the summation in the geometric mean, and use the facts that $a\log x = \log x^a$ and $\exp\log x = x$, one retrives the more common definition of the geometric mean $A_T^{geo} = \left(\prod_{l=1}^{L} S_{t_l}\right)^{1/L}$. We assume in the following that all adjacent time points are separated by the interval Δt, i.e. $t_{l+1} - t_l = \Delta t = T/L$ for all $l = 1, ..., L-1$. Analogously to the European option, we assume to have quantum sampling access **??** to a state that corresponds to the Gaussian distribution with variance Δt:

$$|p_{\Delta t}\rangle = G|0^m\rangle = \sum_{j=0}^{2^m-1} \sqrt{p_{\Delta t}(x_j)}|j\rangle \tag{8.39}$$

where now m is the number of discrete points over which one can define a value for the stock price. The above state has m qubits and it can be prepared thanks to suitable quantum sampling access **??**. With such a state we performed the first key step to apply Monte Carlo techniques in the quantum regime: we stored the right probability distribution in the amplitudes of the qubit register for a given time t_i. Indeed for a given i, the stock price is distributed according to a normal random variable with variance Δt. We want to prepare the product of such states for every t_i with $i = 1, ..., L$. To do so, we use L registers of m qubits and prepare the state $|p_{\Delta t}\rangle$ in each register:

$$|p\rangle = |p_{\Delta t}\rangle ... |p_{\Delta t}\rangle \tag{8.40}$$

Next we define the operation:

$$|j_1, ..., j_L\rangle|0\rangle \to |j_1, ..., j_L\rangle|A(S_{t_1}(x_{j_1}), ..., S_{t_L}(x_{j_L}))\rangle \tag{8.41}$$

here $A(S_{t_1}(x_{j_1}), ..., S_{t_L}(x_{j_L}))$ is the average stock price corresponding to the path $x_{j_1}, ... x_{j_L}$. If you wonder why to introduce such operation, consider a state $|j_1, \dots, j_L\rangle$. This is associated to the following stock price path: at time t_1 the stock price is $S_{t_1}(x_{j_1})$, at time t_2 the stock price is $S_{t_2}(x_{j_2})$ and so on (with their corresponding probabilities). To this stock price path is associated the corresponding average value $A(S_{t_1}(x_{j_1}), \dots, S_{t_L}(x_{j_L}))$. In this way, when using such operation on the state $|p\rangle$ we will get a superposition in which the ancillary registers encode the averages associated to any possible stock price path. Since we want an estimate of the mean of these averages, we should be not surprised anymore.

To be able to apply such a transformation, each index j is mapped to its corresponding point x_j via $x_j = -x_{max} + j\Delta x$. Starting from the known S_0, we can compute the stock price at each time point by iteration:

$$S_{t_{l+1}}(x) = S_{t_l} e^{\sigma x + (r - \sigma^2/2)\Delta t} \tag{8.42}$$

Notice that we embedded the values of σ and r in the circuit. In this way we can get a state where the label $|j_1, \dots, j_L\rangle$ is associated to the corresponding stock price path

$$|j_1, \dots, j_L\rangle |S_{t_1}(x_{j_1}), \dots, S_{t_L}(x_{j_L})\rangle \tag{8.43}$$

Now the average can be computed in a sequential manner. What we mean is that we can implement the step:

$$\begin{aligned} &|j_1, \dots, j_L\rangle |S_{t_l}(x_{j_l})\rangle |A(S_{t_1}(x_{j_1}), \dots, S_{t_l}(x_{j_l}))\rangle \to \\ &|j_1, \dots, j_L\rangle |S_{t_{l+1}}(x_{j_{l+1}})\rangle |A(S_{t_1}(x_{j_1}), \dots, S_{t_{l+1}}(x_{j_{l+1}}))\rangle \end{aligned} \tag{8.44}$$

these steps are repeated until the final time t_L is reached and $A(S_{t_1}(x_{j_1}), \dots, S_{t_L}(x_{j_L}))$ is stored in a register of qubits. Then we just reverse the quantum operation in the ancillary registers storing the intermediate steps, so that ancillary registers go back to an all $|0\rangle$ product state.

Applying these operations to the product state prepared before one finally gets:

$$\sum_{j_1, \dots, j_L = 0}^{2^m - 1} \sqrt{p_{j_1, \dots, j_L}} |j_1, \dots, j_L\rangle |A(S_{t_1}(x_{j_1}), \dots, S_{t_L}(x_{j_L}))\rangle \tag{8.45}$$

where $\sqrt{p_{j_1, \dots, j_L}} = \sqrt{p_{\Delta t}(x_1)} \dots \sqrt{p_{\Delta t}(x_L)}$. Now we can use append an ancillary qubit in $|0\rangle$ state and apply the controlled rotation encoding the value of the payoff function on the $|1\rangle$ subspace. Using theorem 8.5 on this state gives the expectation value that we want.

Notice that we are allowed to use theorem 8.5: Asian options have a bounded variance (the arithmetic mean upper bounds the geometric mean) and it is upper bounded by the expected maximum of the stock price $\max\{S_{t_1}, \dots, S_{t_L}\}$. The variance of this maximum can be bounded. We thus get a quantum algorithms for the pricing of Asian option that needs N queries for accuracy ϵ, while, as we explained at the beginning of the chapter, classical Monte Carlo methods would need N^2 samples to get the same accuracy ϵ.

Note that the algorithms that we presented to price European and Asian options have a running time that depends on a bound on the variance of this variables. Of course if the bound is tight, this means that we can have an optimal running time. As we will see there exist a quantum algorithm to estimate the expectation value of random variables that does not require to know beforehand a bound on the variance.

Chapter 9

Dimensionality reduction

Contributors: Alessandro Luongo, Armando Bellante

We devote this chapter to the study of quantum algorithms for dimensionality reduction. We first delve into unsupervised methods, and then look at the supervised case. In both case we discuss two main subroutines. In one case we can use quantum computers to fit a machine learning model. This means that we can obtain, from a quantum computer, a classical description of the machine learning model (i.e. a set of vectors). In the second case we can imagine using a quantum computer to perform the mapping from an original feature space to the dimensionality-reduced feature space.

9.1 Unsupervised algorithms

In this section we describe three algorithms: the QPCA, QCA and QLSA. These algorithms are mainly from (Bellante and Zanero, 2022), which solves in a single framework many of these eigenvalue problems that appears in fitting some simple machine learning models.

9.1.1 Quantum PCA

Principal component analysis is a widely-used multivariate statistical method for continuous variables that finds many applications in machine learning, ranging from outlier detection to dimensionality reduction and data visualization. Consider a matrix $A \in \mathbb{R}^{n \times m}$ that stores information about n data points using m coordinates (e.g. think of n images described through m pixels each), its *principal components* are the set of orthogonal vectors along which the variance of the data points is maximized. The goal of PCA is to compute the principal components, with the amount of variance they capture, and rotate the data-points to make the axis coincide with the principal components. During the process it is possible to perform dimensionality reduction and reduce the number of variables taken into account, i.e. reducing $A \in \mathbb{R}^{n \times m}$ to $A \in \mathbb{R}^{n \times k}$ where $k \leq m$, and express the original data in terms of fewer latent variables that account for the majority of the variance of the original data. If it is possible to find a few latent variables that retain a large amount of variance of the original variables, it is possible to use PCA to visualize even high dimensional datasets.

9.1.1.0.1 Connection to Singular Value Decomposition The model of PCA is closely related to the singular value decomposition of the data matrix A, shifted to row mean 0. The principal components coincide with the right singular vectors v_i. The factor scores $\lambda_i = \sigma_i^2$ represent the amount of variance along each of them and the factor score ratios $\lambda^{(i)} = \frac{\lambda_i}{\sum_j^r \lambda_j}$ express the percentage of the variance retained. For datasets with 0 mean, the transformation consists in a rotation along the principal components: $Y = AV = U\Sigma V^T V = U\Sigma \in \mathbb{R}^{n \times m}$. Therefore, the data points in the new subspace can be computed using the left singular vectors and the singular values of A. When performing dimensionality reduction it suffice to use only the top k singular values and vectors $Y^{(k)} = AV^{(k)} = U\Sigma V^T V^{(k)} = U^{(k)}\Sigma^{(k)} \in \mathbb{R}^{n \times m}$.

9.1.1.0.2 Quantum algorithms for PCA Using the procedures from section 7.2 it is possible to extract the model for principal component analysis. Theorems 7.6, 7.7, **??** allow to retrieve information on the factor scores and on the factor score ratios, while Theorem 7.8 allows extracting the principal components. The run-time of the model extraction is the sum of the run-times of the theorems: $\widetilde{O}\left(\left(\frac{1}{\gamma^2} + \frac{km}{\theta\delta^2}\right)\frac{\mu(A)}{\epsilon}\right)$. The model comes with the following guarantees: $\|\sigma_i - \overline{\sigma}_i\| \leq \frac{\epsilon}{2}$; $\|\lambda_i - \overline{\lambda}_i\| \leq \epsilon$; $\|\lambda^{(i)} - \overline{\lambda}^{(i)}\| \leq \gamma$; $\|v_i - \overline{v}_i\| \leq \delta$ for $i \in \{0, k-1\}$. This run-time is generally smaller than the number of elements of the input data matrix, providing polynomial speed-ups on the best classical routines for non-sparse matrices. In writing the time complexity of the routines, we omitted the term $\frac{1}{\sqrt{p}}$ because usually the amount of variance to retain p is chosen to be a number greater than 0.5 (generally in the order of 0.8/0.9).

When performing dimensionality reduction, the goal is to obtain the matrix $Y = U\Sigma \in \mathbb{R}^{n\times k}$, where $U \in \mathbb{R}^{n\times k}$ and $\Sigma \in \mathbb{R}^{k\times k}$ are composed respectively by the top k left singular vectors and singular values. When this is the case, the user might want to extract the top k u_i and σ_i rather than the principal components, to avoid matrix multiplication. For this reason, we provide a lemma to bound the error on the retrieved mode. For sake of completeness, the error bound is also stated for $V\Sigma$.

Lemma 9.1 (Accuracy of qPCA's representation (classical))**.** *Let $A \in \mathbb{R}^{n\times m}$ be a matrix with $\sigma_{max} \leq 1$. Given some approximate procedures to retrieve estimates $\overline{\sigma}_i$ of the singular values σ_i such that $\|\sigma_i - \overline{\sigma}_i\| \leq \epsilon$ and unit estimates $\overline{u}_i$ of the left singular vectors u_i such that $\|\overline{u}_i - u_i\|_2 \leq \delta$, the error on $U\Sigma$ can be bounded as $\|U\Sigma - \overline{U\Sigma}\|_F \leq \sqrt{k}(\epsilon + \delta)$. Similarly, $\|V\Sigma - \overline{V\Sigma}\|_F \leq \sqrt{k}(\epsilon + \delta)$.*

Proof. The first step of the proof consists in bounding the error on the columns of the matrices: $||\overline{\sigma_i u_i} - \sigma_i u_i||$.

$$\|\overline{\sigma}_i\overline{u}_i - \sigma_i u_i\| \leq \|(\sigma_i \pm \epsilon)\overline{u}_i - \sigma_i u_i\| = \|\sigma_i(\overline{u}_i - u_i) \pm \epsilon\overline{u}_i\|$$

Because of the triangular inequality, $\|\sigma_i(\overline{u}_i - u_i) \pm \epsilon\overline{u}_i\| \leq \sigma_i\|\overline{u}_i - u_i\| + \epsilon\|\overline{u}_i\|$. Also by hypothesis, $\|(\overline{u}_i - u_i)\| \leq \delta$ and $||\overline{u}_i|| = 1$. Thus, $\sigma_i\|\overline{u}_i - u_i\| + \epsilon\|\overline{u}_i\| \leq \sigma_i\delta + \epsilon$. From the error bound on the columns and the fact that $f(x) = \sqrt{x}$ is an increasing monotone function, it is possible to prove the error bound on the matrices:

$$\|\overline{U\Sigma} - U\Sigma\|_F = \sqrt{\sum_i^n\sum_j^k \|\overline{\sigma}_j\overline{u}_{ij} - \sigma_j u_{ij}\|^2} = \sqrt{\sum_j^k \left(\|\overline{\sigma}_j\overline{u}_j - \sigma_j u_j\|\right)^2}$$

$$\leq \sqrt{\sum_j^k \left(\epsilon + \delta\sigma_j\right)^2} \leq \sqrt{k\left(\epsilon + \delta\sigma_{max}\right)^2} \leq \sqrt{k}(\epsilon + \delta\|A\|)$$

Finally, since $\sigma_{max} \leq 1$, we get that $\|\overline{U\Sigma} - U\Sigma\|_F \leq \sqrt{k}(\epsilon + \delta)$. □

The fact that the matrix has been normalized to have a spectral norm smaller than one is usually not relevant for the final applications of PCA. However, if one desires to represent transformation of the not-normalized matrix A, the error bounds become the ones of the lemma below.

Lemma 9.2 (Non-normalized accuracy of qPCA's representation (classical))**.** *The estimated representations of the lemma above, for the not-normalized matrix*

A, are $\|A\|\overline{U\Sigma}$ and $\|A\|\overline{V\Sigma}$. The error bounds become $\left\| ||A||U\Sigma - ||A||\overline{U\Sigma} \right\|_F \leq \sqrt{k}||A||(\epsilon + \delta)$ and $\left\| ||A||V\Sigma - ||A||\overline{V\Sigma} \right\|_F \leq \sqrt{k}||A||(\epsilon + \delta)$

Proof. Firstly, it is easy to see that to get the desired data representations it suffices to multiply the output matrix by $\|A\|$. Indeed, in our quantum memory, the singular values of the non-normalized matrix are scaled by a factor $\frac{1}{\|A\|}$, while the singular vectors remain the same. To prove the bounds, we can just multiply both sides of the inequalities from Lemma 9.1 by $||A||$, which is a positive quantity:

$$\left\| \|A\|\overline{U\Sigma} - ||A||U\Sigma \right\|_F \leq \sqrt{k}\|A\|(\epsilon + \delta), \left\| \|A\|\overline{V\Sigma} - \|A\|V\Sigma \right\|_F \leq \sqrt{k}\|A\|(\epsilon + \delta).$$

□

Based on Lemma 5.4, we also provide algorithms to produce quantum states proportional to the data representation in the new feature space. After $V^{(k)} \in \mathbb{R}^{m \times k}$ has been extracted, these routines create the new data points in almost constant time and are therefore useful when these states are used in a quantum machine learning pipeline.

Corollary 9.1 (qPCA: vector dimensionality reduction)**.** *Let ξ be a precision parameter. Let there be efficient quantum access to the top k right singular vectors $\overline{V}^{(k)} \in \mathbb{R}^{m \times k}$ of a matrix $A = U\Sigma V^T \in \mathbb{R}^{n \times m}$, such that $\|V^{(k)} - \overline{V}^{(k)}\| \leq \frac{\xi}{\sqrt{2}}$. Given efficient quantum access to a row a_i of A, the quantum state $|\overline{y}_i\rangle = \frac{1}{\|\overline{y}_i\|}\sum_i^k \overline{y}_k |i\rangle$, proportional to its projection onto the PCA space, can be created in time $\widetilde{O}\left(\frac{\|a_i\|}{\|\overline{y}_i\|}\right)$ with probability at least $1 - 1/poly(m)$ and precision $\||y_i\rangle - |\overline{y}_i\rangle\| \leq \frac{\|a_i\|}{\|\overline{y}_i\|}\xi$. An estimate of $\|\overline{y}_i\|$, to relative error η, can be computed in $\widetilde{O}(1/\eta)$.*

Proof. In the proof we use V to denote the matrix $V^{(k)} \in \mathbb{R}^{m \times k}$. Given a vector a_i, its projection onto the k-dimensional PCA space of A is $y_i^T = a_i^T V$, or equivalently $y_i = V^T a_i$. Note that $\|y_i\| = \|V^T a_i\|$. It is possible to use Lemma 5.4 to multiply the quantum state $|a_i\rangle$ by V^T, appropriately padded with 0s to be a square $\mathbb{R}^{m \times m}$ matrix. Lemma 5.4 states that it is possible to create an approximation $|\overline{y}_i\rangle$ of the state $|y_i\rangle = |V^T a_i\rangle$ in time $\widetilde{O}\left(\frac{\mu(V^T)\log(1/\epsilon)}{\gamma}\right)$ with probability $1 - 1/\text{poly}(m)$, such that $\||y_i\rangle - |\overline{y}_i\rangle\| \leq \epsilon$. Since V^T has rows with unit ℓ_2 norm, we can use a result from Theorem IV.1 of (Kerenidis and Prakash, 2020) to prepare efficient quantum access to it with $\mu(V^T) = 1$. Choosing the parameter as $\gamma = \|V^T a_i\|/\|a_i\|$, we get a run-time of $\widetilde{O}(\frac{\|a_i\|}{\|y_i\|}\log(1/\epsilon))$. We can consider the term $\log(1/\epsilon)$ to be negligible, as, for instance, an error $\epsilon = 10^{-17}$ would not be relevant in practice, while accounting for a factor 17 in the run-time. We conclude that the state $|y_i\rangle$ can be created in time $\widetilde{O}\left(\frac{\|a_i\|}{\|y_i\|}\right)$ with probability $1 - 1/\text{poly}(m)$ and that its norm can be estimated to relative error η in time $\widetilde{O}\left(\frac{\|a_i\|}{\|y_i\|}\frac{1}{\eta}\right)$.

For what concerns the error, we start by bounding $\|y_i - \overline{y}_i\|$ and use Lemma D.4 to bound the error on the quantum states. We assume to have estimates $\overline{v}_i$ of the columns of V such that $\|v_i - \overline{v}_i\| \leq \delta$.

$$\|V - \overline{V}\|_F = \sqrt{\sum_i^n \sum_j^k \left(v_{ij} - \overline{v}_{ij}\right)^2} \leq \sqrt{k}\delta$$

Considering that $\|y_i - \overline{y}_i\| = \|a_i^T V^{(k)} - a_i^T \overline{V}^{(k)}\| \leq \|a_i\|\sqrt{k}\delta$, we can use Lemma D.4 to state

$$\||y_i\rangle - |\overline{y}_i\rangle\| \leq \frac{\|a_i\|}{\|y_i\|}\sqrt{2k}\delta = \frac{\|a_i\|}{\|y_i\|}\xi.$$

We can set $\delta = \frac{\xi}{\sqrt{2k}}$ and require $\|V - \overline{V}\|_F \leq \frac{\xi}{\sqrt{2}}$. □

This result also holds when a_i is an new data point, not necessarily stored in A. Note that $\frac{\|y_i\|}{\|a_i\|}$ is expected to be close to 1, as it is the percentage of support of a_i on the new feature space spanned by $V^{(k)}$. We formalize this better using Definition 9.1 below.

Corollary 9.2 (qPCA: matrix dimensionality reduction)**.** *Let ξ be a precision parameter and p be the amount of variance retained after the dimensionality reduction. Let there be efficient quantum access to $A = U\Sigma V^T \in \mathbb{R}^{n\times m}$ and to its top k right singular vectors $\overline{V}^{(k)} \in \mathbb{R}^{m\times k}$, such that $\|V^{(k)} - \overline{V}^{(k)}\| \leq \frac{\xi\sqrt{p}}{\sqrt{2}}$. There exists a quantum algorithm that, with probability at least $1 - 1/poly(m)$, creates the state $|\overline{Y}\rangle = \frac{1}{\|Y\|_F}\sum_i^n \|y_{i,\cdot}\||i\rangle|y_{i,\cdot}\rangle$, proportional to the projection of A in the PCA subspace, with error $\||Y\rangle - |\overline{Y}\rangle\| \leq \xi$ in time $\widetilde{O}(1/\sqrt{p})$. An estimate of $\|\overline{Y}\|_F$, to relative error η, can be computed in $\widetilde{O}(\frac{1}{\sqrt{p}\eta})$.*

Proof. In the proof we use V to denote the matrix $V^{(k)} \in \mathbb{R}^{m\times k}$. Using the same reasoning as the proof above and giving a closer look at the proof of Lemma 5.4 (Lemma 24 (Chakraborty et al., 2019)), we see that it is possible to create the state $|0\rangle(\overline{V}^T|a_i\rangle) + |0_\perp\rangle$ in time $\widetilde{O}(1)$ and that the term $\widetilde{\frac{1}{\gamma}}$ is introduced to boost the probability of getting the right state. If we apply Lemma 5.4 without the amplitude amplification step to the superposition of the rows of A, we obtain the following mapping in time $\widetilde{O}(1)$:

$$|A\rangle = \frac{1}{\|A\|_F}\sum_i^n \|a_{i,\cdot}\||i\rangle|a_{i,\cdot}\rangle \mapsto \frac{1}{\|A\|_F}\sum_i^n(\|y_{i,\cdot}\||0\rangle|i\rangle|y_{i,\cdot}\rangle + \|y_{i,\cdot\perp}\||0_\perp\rangle), \quad (9.1)$$

where $\|y_{i,\cdot\perp}\|$ are normalization factors. Keeping in mind that $\|A\|_F = \sqrt{\sum_i^r \sigma_i^2}$ and $\|Y\|_F = \sqrt{\sum_i^n \|y_{i,\cdot}\|^2} = \sqrt{\sum_i^k \sigma_i^2}$, we see that the amount of explained variance is $p = \frac{\sum_i^k \sigma_i^2}{\sum_j^r \sigma_j^2} = \left(\frac{\|Y\|_F}{\|A\|_F}\right)^2$. The probability of obtaining $|Y\rangle = \frac{1}{\|Y\|_F}\sum_i^n \|y_{i,\cdot}\||i\rangle|y_{i,\cdot}\rangle$ is $p = \frac{\|Y\|_F^2}{\|A\|_F^2} = \frac{\sum_i^n \|y_{i,\cdot}\|^2}{\|A\|_F^2}$. We conclude that, using $\widetilde{O}(1/\sqrt{p})$ rounds of amplitude amplification, we obtain $|Y\rangle$ with proability $1 - 1/\text{poly}(m)$ (5.1).

Considering that $\|Y - \overline{Y}\| = \|AV^{(k)} - A\overline{V}^{(k)}\| \leq \|A\|\sqrt{k}\delta$, we can use Lemma D.4 to state

$$\||Y\rangle - |\overline{Y}\rangle\| \leq \frac{\|A\|_F}{\|Y\|_F}\sqrt{2k}\delta = \xi. \tag{9.2}$$

We can set $\delta = \frac{\xi}{\sqrt{2k}}\frac{\|Y\|_F}{\|A\|_F} = \frac{\xi\sqrt{p}}{\sqrt{2k}}$, so we require $\|V - \overline{V}\|_F \leq \frac{\xi\sqrt{p}}{\sqrt{2}}$. □

The error requirements of the two corollary propagate to the run-time of the model extraction in the following way.

Corollary 9.3 (qPCA: fitting time)**.** *Let ϵ be a precision parameter and $p = \frac{\sum_{i:\overline{\sigma}_i \geq \theta} \sigma_i^2}{\sum_j^r \sigma_j^2}$ the amount of variance to retain, where $\|\sigma_i - \overline{\sigma}_i\| \leq \epsilon$. Given efficient quantum access to a matrix $A \in \mathbb{R}^{n\times m}$, the run-time to extract $V^{(k)} \in \mathbb{R}^{m\times k}$ for corollaries 9.1, @ref{cor:qpca:matrix} is $\widetilde{O}\left(\frac{\mu(A)k^2m}{\theta\epsilon\xi^2}\right)$.*

We state the proof for Corollary 9.2, as its error is more demanding than the one of Corollary 9.1. This way, the proof stands for both cases. ::: {.proof} The procedure to train the model consists in using Theorem **??** to extract the threshold θ, given the amount of variance to retain p, and to leverage Theorem 7.8 to extract the k right singular vectors that compose $V \in \mathbb{R}^{m\times k}$. The run-time of Theorem **??** is smaller than the one of Theorem 7.8, so we can focus on the last one. From the proof of Corollary 9.2, we know that to have $\|V - \overline{V}\|_F \leq \frac{\xi\sqrt{p}}{\sqrt{2}}$ we need $\|v_i - \overline{v}_i\| \leq \frac{\xi\sqrt{p}}{\sqrt{2k}}$. Substituting $\delta = \frac{\xi\sqrt{p}}{\sqrt{2k}}$ in the run-time of Theorem 7.8, we get $\widetilde{O}(\frac{\mu A k^2 m}{p^{3/2}\theta\epsilon\xi^2})$. If we consider that p to be a reasonable number (e.g., at least grater than 0.05), we can consider it a constant factor that is independent from the input's size. The asymptotic run-time is proved to be $\widetilde{O}(\frac{\mu A k^2 m}{\theta\epsilon\xi^2})$. :::

We see that is the algorithm is training the model for Corollary 9.2, the run-time has a dependency on $1/p^{3/2}$, but this term is constant and independent from the size of the input dataset. With this additional $1/p^{3/2}$ cost, the error of Corollary 9.1 drops to ξ for every row of the matrix and generally decreases in case of new data points.

Definition 9.1 (PCA-representable data)**.** A set of n data points described by m coordinates, represented through a matrix $A = \sum_i^r \sigma_i u_i v_i^T \in \mathbb{R}^{n\times m}$ is said to be PCA-representable if there exists $p \in [\frac{1}{2}, 1], \varepsilon \in [0, 1/2], \beta \in [p - \varepsilon, p + \varepsilon], \alpha \in [0, 1]$ such that: - $\exists k \in O(1)$ such that $\frac{\sum_i^k \sigma_i^2}{\sum_i^m \sigma_i^2} = p$ - for at least αn points a_i it holds $\frac{\|y_i\|}{\|a_i\|} \geq \beta$, where $\|y_i\| = \sqrt{\sum_i^k |\langle a_i|v_j\rangle|^2} \, \|a_i\|$.

Thanks to the this statement, it is possible to bound the run-time of Corollary 9.1 with a certain probability.

Lemma 9.3 (qPCA on PCA-representable datasets)**.** *Let a_i be a row of $A \in \mathbb{R}^{n\times d}$. Then, the runtime of Corollary 9.1 is $\frac{\|a_i\|}{\|y_i\|} = \frac{1}{\beta} = O(1)$ with probability greater than α.*

It is known that, in practical machine learning datasets, α is a number fairly close to one. We have tested the value of ρ for the MNIST dataset, the interested reader can read more about it in the section about the experiments. Using the same framework and proof techniques, it is possible to produce similar results for the representations of Correspondence Analysis and Latent Semantic Analysis, that are introduced in the next two sections.

Remark: Note that Theorem 1 from (Yu et al., 2019) propose a lower bound for a quantity similar to α. However, their result seems to be a loose bound: using their notation and setting $\eta = 1, \theta = 1$ they bound this quantity with 0, while a tight bound should be 1.

9.1.2 Quantum Correspondence Analysis

Correspondence analysis is a multivariate statistical tool, from the family of *factor analysis* methods, used to explore relationships among categorical variables. Consider two random variables X and Y with possible outcomes in $\{x_1, \cdots, x_n\}$ and $\{y_1, \cdots, y_m\}$ respectively, the model of Correspondence Analysis allows to represent the outcomes as vectors in two related Euclidean spaces. These spaces can be used for analysis purposes in data visualization, exploration and in other unsupervised machine learning tasks.

9.1.2.0.1 Connection to Singular Value Decomposition Given a contingency table for X and Y, it is possible to compute the matrix $A = D_X^{-1/2}(\hat{P}_{X,Y} - \hat{p}_X\hat{p}_Y^T)D_Y^{-1/2} \in \mathbb{R}^{n\times m}$, where $\hat{P}_{X,Y} \in \mathbb{R}^{n\times m}$ is the estimated matrix of joint probabilities, $\hat{p}_X \in R^n$ and $\hat{p}_X \in \mathbb{R}^m$ are the vectors of marginal probabilities, and $D_X^{-1/2} = diag(\hat{p}_X)$, $D_Y^{-1/2} = diag(\hat{p}_Y)$. The computation of A can be done in time proportional to the number of non zero entries of the contingency table. The singular value decomposition of A is strictly related to the model of correspondence analysis (Greenacre, 1984), (Hsu et al., 2019). The vector space for X is $D_X^{-1/2}U \in \mathbb{R}^{n\times k}$, while the one for Y is $D_Y^{-1/2}V \in \mathbb{R}^{m\times k}$. Note that these spaces are not affected by the normalization of A. Like in PCA, it is possible to choose only a subset of the orthogonal factors as coordinates for the representation. Factor scores and factor score ratios measure of how much “correspondence’ ’ is captured by the respective orthogonal factor, giving an estimate of the quality of the representation.

9.1.2.0.2 Quantum algorithms for CA Similarly to what we have already discussed, it is possible to extract the model for CA by creating quantum access to the matrix A and using Theorems 7.6, 7.7, and 7.8 to extract the orthogonal factors, the factor scores and the factor score ratios in time $\widetilde{O}\left(\left(\frac{1}{\gamma^2}+\frac{k(n+m)}{\theta\delta^2}\right)\frac{\mu(A)}{\epsilon}\right)$. We provide a theoretical bound for the data representations in Lemma 9.4.

Lemma 9.4 (Accuracy of qCA's representation (classical))**.** *Let $A \in \mathbb{R}^{n\times m}$ be a matrix. Given some approximate procedures to retrieve unit estimates $\overline{u}_i$ of the left singular vectors u_i such that $\|\overline{u}_i - u_i\| \leq \delta$, the error on $D_X^{-1/2}U$ can be bounded as $\left|\left|D_X^{-1/2}U - D_X^{-1/2}\overline{U}\right|\right|_F \leq \|D_X^{-1/2}\|_F\sqrt{k}\delta$. Similarly, $\left|\left|D_Y^{-1/2}V - D_Y^{-1/2}\overline{V}\right|\right|_F \leq \|D_Y^{-1/2}\|_F\sqrt{k}\delta$.*

Proof.

$$\left|\left|D_X^{-1/2}\overline{U} - D_X^{-1/2}U\right|\right|_F \leq \|D_X^{-1/2}\|_F \left|\left|\overline{U} - U\right|\right|_F \leq \|D_X^{-1/2}\|_F\sqrt{k}\delta.$$

□

Note that CA's representation does is independent of the scaling of the singular vectors, so the normalization of the dataset does not affect the representation in any way.

9.1.3 Quantum Latent Semantic Analysis

Latent semantic analysis is a data representation method to represent words and text documents as vectors in Euclidean spaces so that it is possible to make comparisons among terms, among documents and between terms and documents. The representation spaces of LSA automatically model synonymy and polysemy (Deerwester et al., 1990), and their applications in machine learning range from topic modeling to document clustering and retrieval.

9.1.3.0.1 Connection to Singular Value Decomposition The input of LSA is a contingency table of n words and m documents $A \in \mathbb{R}^{n\times m}$. Inner products of rows $AA^T = U\Sigma^2U^T$ express the distances between words. Inner products of columns $A^TA = V\Sigma^2V^T$ express the distances between documents. The a_{ij} element of $A = U\Sigma V^T$ expresses the distance between word i and document j. With this definitions it is possible to compute: - a space for words comparisons $U\Sigma \in \mathbb{R}^{n\times k}$; - a space for documents comparisons $V\Sigma \in \mathbb{R}^{m\times k}$; - two spaces for words and documents comparisons $U\Sigma^{1/2} \in \mathbb{R}^{n\times k}$ and $V\Sigma^{1/2} \in \mathbb{R}^{m\times k}$. When using LSA for latent semantic indexing, one wishes to represent the query as a vector in the document comparison space. The new vector is computed in the following way $v_q^T = x_q^TU\Sigma^{-1}$, where $x_q \in \mathbb{R}^n$ is obtained using the same criteria used to store a document in A. The orthogonal factors used to compare documents can be seen as latent topics whose importance is proportional to the corresponding factor score ratios.

9.1.3.0.2 Quantum algorithms for LSA As previously discussed, the cost of model extraction is $\widetilde{O}\left(\left(\frac{1}{\gamma^2} + \frac{k(n+m)}{\theta\delta^2}\right)\frac{\mu(A)}{\epsilon}\right)$. In some applications, such as document retrieval, the data analyst maintains a fixed number of singular values and vectors, regardless of the factor score ratios. In (Deerwester et al., 1990), $k = 100$ is found to be a good number for document indexing. Similarly, we believe that it is possible to empirically determine a threshold θ to use in practice. Determining such threshold would reduce the complexity of model computation to the one of Theorem 7.8: $\widetilde{O}\left(\frac{k(n+m)}{\theta\delta^2}\frac{\mu(A)}{\epsilon}\right)$.

For what concerns the error bounds on the retrieved data representation models, we already know from Lemma 9.1 that it is possible to retrieve an approximation $\overline{U\Sigma}$ and $\overline{V\Sigma}$ with precision $\sqrt{k}(\delta+\epsilon)$, where δ is the precision on the singular vectors and ϵ the precision on the singular values. To provide bounds on the estimations of $U\Sigma^{1/2}$, $V\Sigma^{1/2}$, and $U\Sigma^{-1}$ we introduce Lemma 9.5 and Lemma @ref{lem:accuracyUE-1eVE-1}.

Lemma 9.5 (Accuracy of qLSA's representations (classical))**.** *Let $A \in \mathbb{R}^{n\times m}$ be a matrix with $\sigma_{max} \leq 1$. Given some approximate procedures to retrieve estimates $\overline{\sigma}_i$ of the singular values σ_i such that $\|\overline{\sigma}_i - \sigma_i\| \leq \epsilon$ and unitary estimates $\overline{u}_i$ of the left singular vectors u_i such that $\|\overline{u}_i - u_i\| \leq \delta$, the error on $U\Sigma^{1/2}$ can be bounded as $\left\|U\Sigma^{1/2} - \overline{U\Sigma}^{1/2}\right\|_F \leq \sqrt{k}\left(\delta + \frac{1}{2\sqrt{\theta}}\right)$. Similarly, $\left\|V\Sigma^{1/2} - \overline{V\Sigma}^{1/2}\right\|_F \leq \sqrt{k}\left(\delta + \frac{1}{2\sqrt{\theta}}\right)$.*

Proof. We start by bounding $\|\sqrt{\overline{\sigma}_i} - \sqrt{\sigma_i}\|$. Let's define $\epsilon = \gamma\sigma_i$ as a relative error:

$$\|\sqrt{\sigma_i+\epsilon} - \sqrt{\sigma_i}\| = \|\sqrt{\sigma_i+\gamma\sigma_i} - \sqrt{\sigma_i}\| = \|\sqrt{\sigma_i}(\sqrt{1+\gamma}-1)\| =$$

$$\sqrt{\sigma_i}\left\|\frac{(\sqrt{1+\gamma}-1)(\sqrt{1+\gamma}+1)}{\sqrt{1+\gamma}+1}\right\| = \sqrt{\sigma_i}\left\|\frac{\gamma+1-1}{\sqrt{1+\gamma}+1}\right\| \leq \sqrt{\sigma_i}\frac{\gamma}{2}.$$

By definition $\gamma = \frac{\epsilon}{\sigma_i}$ and we know that $\sigma_{min} \geq \theta$:

$$\|\sqrt{\overline{\sigma}_i} - \sqrt{\sigma_i}\| \leq \frac{\sqrt{\sigma_i}}{\sigma_1}\frac{\epsilon}{2} = \frac{\epsilon}{2\sqrt{\sigma_i}} \leq \frac{\epsilon}{2\sqrt{\theta}}.$$

Using the bound on the square roots, we can bound the columns of $\overline{U\Sigma}^{1/2}$:

$$\left\|\sqrt{\overline{\sigma}_i}\overline{u}_i - \sqrt{\sigma_i}u_i\right\| \leq \left\|\left(\sqrt{\sigma_i} + \frac{\epsilon}{2\sqrt{\theta}}\right)\overline{u}_i - \sqrt{\sigma_i}u_i\right\| =$$

$$\left\|\sqrt{\sigma_i}(\overline{u}_i - u_i) + \frac{\epsilon}{2\sqrt{\theta}}\overline{u}_i\right\| \leq \sqrt{\sigma_i}\delta + \frac{\epsilon}{2\sqrt{\theta}} \leq \delta\sqrt{\|A\|} + \frac{\epsilon}{2\sqrt{\theta}}.$$

From the error bound on the columns we derive the bound on the matrices:

$$\left\|\overline{U\Sigma}^{1/2} - U\Sigma^{1/2}\right\|_F = \sqrt{\sum_j^k \left(\left\|\sqrt{\overline{\sigma}_j}\overline{u}_j - \sqrt{\sigma_j}u_j\right\|\right)^2} \leq \sqrt{k}\left(\delta\sqrt{\|A\|} + \frac{\epsilon}{2\sqrt{\theta}}\right).$$

Finally, since $\sigma_{max} \leq 1$, we get that $\|\overline{U\Sigma}^{1/2} - U\Sigma^{1/}\|_F \leq \sqrt{k}(\delta + \frac{\epsilon}{2\sqrt{\theta}})$. □

Lemma 9.6 (Accuracy of qLSA's representation for new document queries (classical))**.** *Let $A \in \mathbb{R}^{n\times m}$ be a matrix. Given some approximate procedures to retrieve estimates $\overline{\sigma}_i$ of the singular values σ_i such that $|\overline{\sigma}_i - \sigma_i| \leq \epsilon$ and unitary estimates $\overline{u}_i$ of the left singular vectors u_i such that $\|\overline{u}_i - u_i\| \leq \delta$, the error on $U\Sigma^{-1}$ can be bounded as $\left\|U\Sigma^{-1} - \overline{U\Sigma}^{-1}\right\|_F \leq \sqrt{k}\left(\frac{\delta}{\theta} + \frac{\epsilon}{\theta^2 - \theta\epsilon}\right)$. Similarly, $\left\|V\Sigma^{-1} - \overline{V\Sigma}^{-1}\right\|_F \leq \sqrt{k}(\frac{\delta}{\theta} + \frac{\epsilon}{\theta^2 - \theta\epsilon})$.*

Proof. We start by bounding $\|\frac{1}{\overline{\sigma}_i} - \frac{1}{\sigma_i}\|$, knowing that $\sigma_{min} \geq \theta$ and $\epsilon < \theta$:

$$\left\|\frac{1}{\overline{\sigma}_i} - \frac{1}{\sigma_i}\right\| \leq \left\|\frac{1}{\sigma_i - \epsilon} - \frac{1}{\sigma_i}\right\| \leq \frac{\epsilon}{\theta^2 - \theta\epsilon}.$$

From the bound on the inverses, we can obtain the bound on the columns of $\overline{U\Sigma}^{-1}$:

$$\left\|\frac{1}{\overline{\sigma}_i}\overline{u}_i - \frac{1}{\sigma_i}u_i\right\| \leq \left\|\left(\frac{1}{\sigma_i} \pm \frac{\epsilon}{\theta^2 - \theta\epsilon}\right)\overline{u}_i - \frac{1}{\sigma_i}u_i\right\| \leq \frac{1}{\sigma_i}\delta + \frac{\epsilon}{\theta^2 - \theta\epsilon} \leq \frac{\delta}{\theta} + \frac{\epsilon}{\theta^2 - \theta\epsilon}.$$

To end the proof, we can compute the bound on the matrices:

$$\left\|\overline{U\Sigma}^{-1} - U\Sigma^{-1}\right\|_F = \sqrt{\sum_j^k \left(\left\|\frac{1}{\overline{\sigma}_j}\overline{u}_j - \frac{1}{\sigma_j}u_j\right\|\right)^2} \leq \sqrt{k}\left(\frac{\delta}{\theta} + \frac{\epsilon}{\theta^2 - \theta\epsilon}\right).$$

□

As for qPCA, we provide the bounds in case we want to undo data normalization step. The proofs for these lemmas proceeds like the one of Lemma 9.2.

Lemma 9.7 (Non-normalized accuracy of qLSA's representations (classical))**.** *The estimated representations of Lemma 9.5, for the not-normalized matrix A, are $\sqrt{\|A\|}\ \overline{U\Sigma}^{1/2}$ and $\sqrt{\|A\|}\ \overline{V\Sigma}^{1/2}$. The error bounds become $\left\|\sqrt{\|A\|}\ \overline{U\Sigma}^{1/2} - \sqrt{\|A\|}U\Sigma^{1/2}\right\|_F \leq \sqrt{k\|A\|}(\epsilon{+}\delta)$ and $\left\|\sqrt{\|A\|}\ \overline{V\Sigma}^{1/2} - \sqrt{\|A\|}V\Sigma^{1/2}\right\|_F \leq \sqrt{k\|A\|}(\epsilon + \delta)$.*

Lemma 9.8 (Non-normalized accuracy of qLSA's representation for new document queries (classical))**.** *The estimated representations of Lemma 9.6, for the not-normalized matrix A, are* $\frac{1}{||A||}\overline{U\Sigma}^{-1}$ *and* $\frac{1}{||A||}\overline{V\Sigma}^{-1}$*. The error bounds become* $\left\|\frac{1}{||A||}\overline{U\Sigma}^{-1} - \frac{1}{||A||}U\Sigma^{1/2}\right\| \leq \frac{\sqrt{k}}{||A||}\left(\frac{\delta}{\theta} + \frac{\epsilon}{\theta^2-\theta\epsilon}\right)$ *and* $\left\|\frac{1}{||A||}\overline{V\Sigma}^{-1} - \frac{1}{||A||}V\Sigma^{1/2}\right\| \leq \frac{\sqrt{k}}{||A||}\left(\frac{\delta}{\theta} + \frac{\epsilon}{\theta^2-\theta\epsilon}\right)$.

9.2 Supervised algorithms

9.2.1 Quantum Slow Feature Analysis

Slow Feature Analysis (SFA) is a dimensionality reduction technique proposed in the context of computational neurosciences as a way to model part of the visual cortex of humans. In the last decades, it has been applied in various areas of machine learning. In this chapter we propose a quantum algorithm for slow feature analysis, and detail its application for performing dimensionality reduction on a real dataset. We also simulate the random error that the quantum algorithms might incur. We show that, despite the error caused by the algorithm, the estimate of the model that we obtain is good enough to reach high accuracy on a standard dataset widely used as benchmark in machine learning. Before providing more details on this result, we give a brief description of dimensionality reduction and introduce the model of slow feature analysis in this context.

SFA has been shown to model a kind of neuron (called complex cell) situated in the cortical layer in the primary visual cortex (called V1) (Berkes and Wiskott, 2005). SFA can be used in machine learning as a DR algorithm, and it has been successfully applied to enhance the performance of classifiers (Zhang Zhang and Dacheng Tao, 2012), (Berkes, 2005). SFA was originally proposed as an *online, nonlinear, and unsupervised algorithm* (Wiskott Laurenz and Wiskott, 1999). Its task was to learn slowly varying features from generic input signals that vary rapidly over time (Berkes, 2005) (Wiskott Laurenz and Wiskott, 1999). SFA has been motivated by the *temporal slowness principle*, that postulates that while the primary sensory receptors (like the retinal receptors in an animal's eye) are sensitive to very small changes in the environment and thus vary on a very fast time scale, the internal representation of the environment in the brain varies on a much slower time scale. The slowness principle is a hypothesis for the functional organization of the visual cortex and possibly other sensory areas of the brain (Wiskott et al., 2011) and it has been introduced as a way to model the transformation invariance in natural image sequences (Zhang Zhang and Dacheng Tao, 2012). SFA is an algorithm that formalizes the slowness principle as a nonlinear optimization problem. In (Blaschke and Wiskott, 2004, sprekeler2014extension), SFA has been used to do nonlinear blind source separation. Although SFA has been developed in the context of computational neurosciences, there have been many applications of the algorithm to solve ML related tasks. A prominent advantage of SFA compared to other algorithms is that it is almost hyperparameter-free. The only parameters to chose are in the preprocessing of the data, e.g. the initial PCA dimension and the nonlinear expansion that consists of a choice of a polynomial of (usually low) degree p. Another advantage is that it is guaranteed to find the optimal solution within the considered function space (Escalante-B and Wiskott, 2012). For a detailed description of the algorithm, we suggest (Sprekeler and Wiskott, 2008). With appropriate preprocessing, SFA can be used in conjunction to a supervised algorithm to acquire classification capabilities. For instance it has been used for pattern recognition to classify images of digits in the famous MNIST database (Berkes, 2005). SFA can be adapted to be used to solve complex tasks in supervised learning, like face and human action recognition (Gu et al., 2013) , [Zhang Zhang and Dacheng Tao (2012; Sun et al., 2014).

We can use SFA for classification in the following way. One can think of the training set a set of vectors $x_i \in \mathbb{R}^d, i \in n$. Each x_i belongs to one of K different classes. A class T_k has $|T_k|$ vectors in it. The goal is to learn $K-1$ functions $g_j(x_i), j \in [K-1]$ such that the output $y_i = [g_1(x_i), \cdots, g_{K-1}(x_i)]$ is very similar for the training samples of the same class and largely different for samples of different classes. Once these functions are learned, they are used to map the training set in a low dimensional vector space. When a new data point arrive, it is mapped to the same vector space, where classification can be done with higher accuracy. SFA projects the data points onto the subspace spanned by the eigenvectors associated to the k smallest eigenvalues of the derivative covariance matrix of the data, which we define in the next section.

9.2.1.1 The SFA model

Now we introduce the optimization problem in its most general form as it is commonly stated for classification (Berkes, 2005). Let $a = \sum_{k=1}^{K} \binom{|T_k|}{2}$. For all $j \in [K-1]$, minimize:

$$\Delta(y_j) = \frac{1}{a} \sum_{k=1}^{K} \sum_{\substack{s,t \in T_k \\ s<t}} \left(g_j(x_s) - g_j(x_t)\right)^2$$

with the following constraints:

- $\frac{1}{n} \sum_{k=1}^{K} \sum_{i \in T_k} g_j(x_i) = 0 \quad \forall j \in [K-1]$
- $\frac{1}{n} \sum_{k=1}^{K} \sum_{i \in T_k} g_j(x_i)^2 = 1 \quad \forall j \in [K-1]$
- $\frac{1}{n} \sum_{k=1}^{K} \sum_{i \in T_k} g_j(x_i) g_v(x_i) = 0 \quad \forall v < j$

The minimization of the delta values $\Delta(y_j)$ encodes the requirement on the output signal to vary "as slow as possible", and thus the delta values are our measure of slowness. They are the average of the square of the first order derivative (over time) of the j-th component of the output signal y_t. The first requirement states that the average over time of each component of the signal should be zero, and it is stated just for convenience, such that the other two requirements take a simple form. The second requirement asks for the variance over time of each component of the signal to be 1. It is equivalent to saying that each signal should carry some information and avoid the trivial solution $g_j(\cdot) = 0$. The third requirement is to say that we want the signals to be decorrelated with each other. This also introduces an order, such that the first signal is the slowest, the second signal is the second slowest and so on. The first and the second constraint also avoid the trivial solution $y_i = 0$. Intuitively, the decorrelation constraint forces different functions g_j to encode different "aspects' ' of the input, maximizing the information carried by the output signal.

In order for the minimization problem to be computationally feasible at scale, the g_j's are restricted to be linear functions w_j such that the output signal becomes $y_i = [w_1^T x_i, \cdots w_{K-1}^T x_i]^T$ or else $Y = XW$, where $X \in \mathbb{R}^{n \times d}$ is the matrix with rows the input samples and $W \in \mathbb{R}^{d \times (K-1)}$ the matrix that maps the input matrix X into a lower dimensional output $Y \in \mathbb{R}^{n \times (K-1)}$. In case it is needed to capture nonlinear relations in the dataset, one performs a standard nonlinear polynomial expansion on the input data as preprocessing. Usually, a polynomial expansion of degree 2 or 3 is chosen. For example we can take:

$$x = [x_1, x_2, x_3] \to \left[x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2, x_1, x_2, x_3\right].$$

The choice of the nonlinear expansion is important for using SFA in machine learning contexts. If it is a low dimensional expansion, it might not solve the task with high accuracy, while if the dimension is too high, it might overfit the training data, and therefore not generalize properly to test data. This technique also goes under the name of polynomial kernel.

We also need to satisfy the constraint on the average of the signal being zero and have unit variance. This is not a strong requirement, since it can be enforced beforehand, by preprocessing the dataset. This requires only linear time with respect to the dimensions of the data, and in practice consist in removing the mean and scale the components by their variance. Namely, we assume that the j-th component of the i-th vector in the dataset satisfies the condition:

$$(x_i)_j := \frac{(\tilde{x}_i)_j - E[(\tilde{x}_i)_j]}{\sqrt{E[((\tilde{x}_i)_j - E[(\tilde{x}_i)_j])^2]}},$$

where with $\tilde{x}(i)$ we define a raw signal with arbitrary mean and variance, $E[\tilde{x}_j(i)]$ the expected value of a single component of the vectors. This allows us to rewrite the minimization problem including the constraints of zero mean and unit variance. We can restate the definition of the delta function as a generalized eigenvalue problem:

$$\Delta(y_j) = \frac{w_j^T A w_j}{w_j^T B w_j}, \tag{9.3}$$

where the matrix B is called the sample covariance matrix and defined as:

$$B := \frac{1}{n}\sum_{i\in[n]} x_i x_i^T = X^T X \tag{9.4}$$

and the matrix A is called the sample derivative covariance matrix and defined as:

$$A := \frac{1}{a}\sum_{k=1}^{K}\sum_{\substack{i,i'\in T_k\\ i<i'}} (x_i - x_{i'})(x_i - x_{i'})^T = \frac{1}{a}\sum_{k=1}^{K} \dot{X_k}^T \dot{X}_k := \dot{X}^T\dot{X}. \tag{9.5}$$

Note also, that we can approximate the matrix A by subsampling from all possible pairs $(x_i, x_{i'})$ from each class and this is indeed what happens in practice.

9.2.1.1.1 Slowly varying signals We formalize the concept of slowly varying signals. While this won't have any utility in the classical algorithm, it will allow us to bound nicely the runtime of the quantum algorithm, in the case when the data has "structure' ' that can be extracted by the SFA algorithm. In general, a slow signal is a signal that change slowly over time. This concept is formalized in the context of SFA by requiring that the whitened signal can be reconstructed without too much error from the projection on a few slow feature vectors. Formally, for a given K, and a set of slow feature vectors $w_1 \dots w_{K-1}$, we define a slowly varying signal as follows.

Definition 9.2 (Slowly varying signal)**.** Let $X \in \mathbb{R}^{n\times d}$ and $Y \in [K]^n$ be a dataset and its labels. Let the rows x_i of X have whitened representation z_i. For the $K-1$ slow feature vectors $w_j, j \in [K]$, let y_i be the slow feature representation of x_i. We say that X is γ_K-slow if:

$$\frac{\sum_{i=0}^{n}\|z_i\|}{\sum_{i=0}^{n}\|y_i\|} \leq \gamma_K$$

If we use SFA for doing dimensionality reduction in the context of supervised learning, a dataset is slowly varying if all the elements in the same class are similar to each other. In this way, by projecting the original images in the subspace spanned by a small number of slow feature vectors, we can reconstruct most of the information of the original dataset. We stress that this assumption is not needed for the quantum algorithm to work, but instead it will only be useful to give guarantees on the runtime of the algorithm.

9.2.1.1.2 The SFA algorithm The SFA algorithm basically provides a solution to the generalized eigenvalue problem $AW = \Lambda BW$ and outputs the eigenvectors corresponding to the smallest eigenvalues. As we said we assume that the data has been normalized and polynomially expanded.

The first step of the algorithm is to whiten the data. This will reduce the problem into a normal eigenvalue problem; the second step is to perform PCA in order to find the eigenvalues and eigenvectors. We refer to (Escalante-B and Wiskott, 2012) for a more complete description.

9.2.1.1.2.1 Step 1: Whitening the data Recall that $X \in \mathbb{R}^{n\times d}, A, B \in \mathbb{R}^{d\times d}$. We now show how to whiten the data by right multiplying with the matrix $B^{-1/2} = [(X^TX)]^{-1/2}$. Then, the input matrix becomes $Z = XB^{-1/2}$ and the covariance matrix of the whitened data Z^TZ is thus the identity.

Lemma 9.9. *Let \$Z:=XB^{-1/2} \$ be the matrix of whitened data. Then* $Z^TZ = I.\%$ *and* $B^{-1/2}$ *is symmetric.*

Proof. Let $X = U\Sigma V^T$. We defined $B = V\Sigma^2V^T$. As $Z = U\Sigma V^T(V\Sigma^{-1}V^T) = UIV$ It follows that $Z^TZ = I$. □

As in the classical algorithm, we will whiten the data by left-applying the whitening matrix $B^{-1/2}$. We will use matrix multiplication algorithms to create a state $|Z\rangle$ proportional to the whitened data.

9.2.1.1.2.2 Step 2: Projection in slow feature space The second step of SFA consists of outputting the $K-1$ “slowest’ ’ eigenvectors of the derivative covariance matrix of the whitened data $A := \dot{Z}^T\dot{Z}$, where $\dot{Z}$ is defined similar to $\dot{X}$ by using the whitened samples instead of the original ones. Note that taking the derivatives of the whitened data is equal to whitening the derivatives.

Lemma 9.10. *Let* $A = \dot{Z}^T\dot{Z}$. *Then* $A = (B^{-1/2})^T\dot{X}^T\dot{X}B^{-1/2}$.

Proof.

$$\begin{aligned}
A &= \dot{Z}^T\dot{Z} = \frac{1}{a}\sum_{k=1}^{K}\sum_{\substack{i,i'\in T_k\\ i<i'}}(z_i - z_{i'})(z_i - z_{i'})^T \\
&= (B^{-1/2})^T\frac{1}{a}\sum_{k=1}^{K}\sum_{\substack{i,i'\in T_k\\ i<i'}}(x_i - x'_i)(x_i - x_{i'})^TB^{-1/2} \\
&= (B^{-1/2})^T\dot{X}^T\dot{X}B^{-1/2}
\end{aligned}$$

□

This observation allow us to whiten the data with a quantum procedure. Recall that the matrix A is usually approximated with a small fraction of all the possible derivatives, roughly linear (and not quadratic) on the number of data points. In our case we take the number of rows of the derivative matrix to be just double the number of data points, and in the experiment we show that this does not compromise the accuracy.

9.2.1.2 Quantum Slow Feature Analysis

We are finally ready to put forward a quantum procedure for SFA. Specifically, we want to map the input matrix X to the output matrix $Y = XW$, and then we will see how to estimate classically the slow feature vectors $W \in \mathbb{R}^{(K-1)\times d}$. For this, we assume to have quantum access to the matrices X and $\dot{X}$, as in definition **??**. We start by describing a circuit that approximately performs the unitary $U_{QSFA} : |X\rangle \to |Y\rangle$ where $|X\rangle$ is the quantum state we obtain by having quantum access to X, the dataset, and $|Y\rangle := \frac{1}{\|Y\|_F}\sum_{i=0}^{n} \|y_i\| \, |i\rangle|y_i\rangle$. As the classical algorithm, QSFA is divided in two parts. In the first step we whiten the data, i.e. we map the state $|X\rangle$ to the state $|Z\rangle = |XB^{-1/2}\rangle$, and in the second step we approximately project $|Z\rangle$ onto the subspace spanned by the smallest eigenvectors of the whitened derivative covariance matrix $A = \dot{Z}^T\dot{Z}$.

9.2.1.3 Step 1: Whitening the data

Recall that $X = \sum_i \sigma_i u_i v_i^T \in \mathbb{R}^{n\times d}$, and $A, B \in \mathbb{R}^{d\times d}$. We now show how to whiten the data having quantum access to the matrix X. As $B^{-1/2}$ is a symmetric matrix with eigenvectors the column singular vectors of X and eigenvalues equal to $1/|\sigma_i|$. Using quantum linear algebra procedure, i.e. theorem 5.11, we can multiply with $B^{-1/2}$ our state $|X\rangle$. Thus, we have the following corollary.

Corollary 9.4 (Whitening algorithm)**.** *Assume to have quantum access to $X = \sum_i \sigma_i u_i v_i^T \in \mathbb{R}^{n\times d}$, as in theorem* **??***. Let $Z = XB^{-1/2}$ the matrix of whitened data. There exists a quantum algorithm that produces as output a state $|\bar{Z}\rangle$ such that $||\bar{Z}\rangle - |Z\rangle| \leq \varepsilon$ in expected time $\widetilde{O}(\kappa(X)\mu(X)\log 1/\varepsilon))$.*

9.2.1.4 Step 2: Projection in slow feature space

The previous Corollary gives a way to build quantum access to the rows of the whitened matrix Z, up to some error ϵ. Now we want to project this state onto the subspace spanned by the eigenvectors associated to the $K-1$ "slowest' ' eigenvectors of the whitened derivative covariance matrix $A := \dot{Z}^T\dot{Z}$, where $\dot{Z}$ is the whitened derivative matrix $\dot{Z} = \dot{X}B^{-1/2}$. Let θ a threshold value and δ a precision parameter, that governs the error we tolerate in the projection threshold. Recall that $A_{\leq\theta,\delta}$ we denote a projection of the matrix A onto the vector subspace spanned by the union of the singular vectors associated to singular values that are smaller than θ and some subset of singular vectors whose corresponding singular values are in the interval $[\theta, (1+\delta)\theta]$.

To perform the projection, we will need a threshold for the eigenvalues that will give us the subspace of the $K-1$ slowest eigenvectors. A priori, we don't know the appropriate threshold value, and thus it must be found experimentally through binary search since it depends on the distribution of singular values of the matrix representing the dataset. We can now describe and analyse the entire QSFA algorithm.

As in the previous section, we note that the eigenvalues of $A_{\dot{Z}}$ are the squares of the singular values of $\dot{Z}$, and the two matrices share the same column space: $\dot{Z} = U\Sigma V^T$, and $A_{\dot{Z}} = V\Sigma^2 V^T$. Claim 9.10 tells us that whitening the derivatives of the signal is equal to taking the derivatives of the whitened data. theorem 5.12 provides exactly the procedure for accessing the rows of $\dot{Z}$, since we know how to multiply with $\dot{X}$ and with $B^{-1/2}$.

Theorem 9.1 (QSFA algorithm)**.** *Assume to have quantum access to* $X = \sum_i \sigma_i u_i v_i^T \in \mathbb{R}^{n \times d}$ *and its derivative matrix* $\dot{X} \in \mathbb{R}^{n \log n \times d}$*. Let* $\epsilon, \theta, \delta, \eta > 0$*. There exists a quantum algorithm that produces as output a state* $|\overline{Y}\rangle$ *with* $||\overline{Y}\rangle - |A^+_{\leq\theta,\delta}A_{\leq\theta,\delta}Z\rangle| \leq \epsilon$ *in time*

$$\tilde{O}\left(\frac{(\kappa(X)+\kappa(\dot{X}))(\mu(X)+\mu(\dot{X}))}{\delta\theta}\gamma_{K-1}\right)$$

and an estimator $\overline{\|Y\|}$ *with* $|\overline{\|Y\|} - \|Y\|| \leq \eta \|Y\|$ *with an additional* $1/\eta$ *factor.*

Proof. QSFA consists of two steps. The first step is the whitening, which can be performed in time $\tilde{O}(\kappa(X)\mu(X)\log(1/\epsilon))$ and provide the state $|\overline{Z}\rangle$ using Corollary 9.4. It is simple to verify that creating a state $|Z\rangle$ of whitened data such that $Z^TZ = I$ can be done using quantum access just to the matrix X, as $Z = XB^{-1/2}$. The second step is the projection of whitened data in the slow feature space, which is spanned by the eigenvectors of $A = \dot{Z}^T\dot{Z}$. This matrix shares the same right eigenvectors of $\dot{X}B^{-1/2}$, which is simple to check that we can efficiently access using the QRAM constructions of X and $\dot{X}$. Using the algorithm for quantum linear algebra, i.e. theorem 5.12, we know that the projection (without the amplitude amplification) takes time equal to the ratio $\mu(X) + \mu(\dot{X})$ over the threshold parameter, in other words it takes time $\tilde{O}(\frac{(\mu(X)+\mu(\dot{X})}{\delta\theta})$. Finally, the amplitude amplification and estimation depends on the size of the projection of $|\overline{Z}\rangle$ onto the slow eigenspace of A, more precisely it corresponds to the factor $O(\frac{\|\overline{Z}\|}{\|A^+_{\leq\theta,\kappa}A_{\leq\theta,\kappa}\overline{Z}\|})$, which is roughly the same if we look at Z instead of $\overline{Z}$. Note also that Z is the whitened data, which means that each whitened vector should look roughly the same on each direction. This implies that the ratio should be proportional to the ratio of the dimension of the whitened data over the dimension of the output signal. The final runtime of the algorithm is:

$$\tilde{O}\left(\left(\kappa(X)\mu(X)\log(1/\varepsilon) + \frac{(\kappa(X)+\kappa(\dot{X}))(\mu(X)+\mu(\dot{X}))}{\delta\theta}\right)\frac{\|Z\|}{\left\|A^+_{\leq\theta,\delta}A_{\leq\theta,\delta}Z\right\|}\right)$$

Note that the last ratio in this runtime was defined as γ_{K-1} in definition 9.2. From this, the runtime in the statement of the theorem follows. □

```
**QSFA** Quantum Slow Feature Analysis
Require:
    Quantum access to matrices $X \in \mathbb{R}^{n \times d}$ and $\dot{X} \in \mathbb{R}^{n \times d}$, parameters $\epsilon, \theta, \delta, \eta > 0$.
Ensure:
    A state $|\overline{Y}\rangle$ such that $|\,|Y\rangle - |\overline{Y}\rangle\,| \leq \epsilon$, with $Y = A^{+}_{\leq\theta,\delta} A_{\leq\theta,\delta} Z$

1: Create the state $|X\rangle := \frac{1}{\|X\|_F} \sum_{i=1}^{n} \|x_i\| \, |i\rangle \, |x_i\rangle$
2: (Whitening algorithm) Map $|X\rangle$ to $|\overline{Z}\rangle$ with $|\,|\overline{Z}\rangle - |Z\rangle\,| \leq \epsilon$ and $Z = XB^{-1/2}$.
3: (Projection in slow feature space) Use theorem ?? project $|\overline{Z}\rangle$ onto the slow eigenspace of $A$ using threshold $\theta$ and precision $\delta$ (i.e. $A^{+}_{\leq\theta,\delta} A_{\leq\theta,\delta} \overline{Z}$)
4: Perform amplitude amplification and estimation on the register $|0\rangle$ with the unitary $U$ implementing steps 1 to 3, to obtain $|\overline{Y}\rangle$ with $|\,|\overline{Y}\rangle - |Y\rangle\,| \leq \epsilon$ and an estimator $\overline{\|Y\|}$ with multiplicative error $\eta$.
```

Figure 9.1: Quantum algorithm for Slow Feature Analysis

Chapter 10

q-means

Contributors: Alessandro Luongo - based on work (Kerenidis et al., 2019a)

In this section we detail a quantum algorithm for unsupervised learning, which can be seen as the quantum version of the well known k-means algorithm. This algorithm is one of the simplest, yet most commonly used clustering algorithms. We first introduce the classical algorithm, then propose a definition of the k-mean model that makes is robust to error in the model. Then, we explain how to derive a quantum version of the k-means algorithm and show its performance on experimental data.

10.1 The k-means algorithm

The k-means algorithm was introduced in (Lloyd, 1982), and is extensively used for unsupervised problems. The inputs to k-means algorithm are vectors $x_i \in \mathbb{R}^d$ for $i \in [n]$. These points must be partitioned in k subsets according to a similarity measure, which in k-means is the Euclidean distance between points. The output of the k-means algorithm is a list of k cluster centers, which are called *centroids*. The algorithm starts by selecting k initial centroids randomly or using efficient heuristics like the k-means++ (Arthur and Vassilvitskii, 2007). It then alternates between two steps: (i) Each data point is assigned the label of the closest centroid. (ii) Each centroid is updated to be the average of the data points assigned to the corresponding cluster. These two steps are repeated until convergence, that is, until the change in the centroids during one iteration is sufficiently small.

More precisely, we are given a dataset X of vectors $x_i \in \mathbb{R}^d$ for $i \in [n]$. At step t, we denote the k clusters by the sets C_j^t for $j \in [k]$, and each corresponding centroid by the vector c_j^t. At each iteration, the data points x_i are assigned to a cluster C_j^t such that $C_1^t \cup C_2^t \cdots \cup C_K^t = V$ and $C_i^t \cap C_l^t = \emptyset$ for $i \neq l$. Let

$d(x_i, c_j^t)$ be the Euclidean distance between vectors x_i and c_j^t. The first step of the algorithm assigns each x_i a label $\ell(x_i)^t$ corresponding to the closest centroid, that is

$$\ell(x_i)^t = \text{argmin}_{j\in[k]}(d(x_i, c_j^t)).$$

The centroids are then updated, $c_j^{t+1} = \frac{1}{|C_j^t|}\sum_{i\in C_j^t} x_i$, so that the new centroid is the average of all points that have been assigned to the cluster in this iteration. We say that we have converged if for a small threshold τ (which might be data dependent) we have:

$$\frac{1}{k}\sum_{j=1}^{k} d(c_j^t, c_j^{t-1}) \leqslant \tau.$$

The loss function that this algorithm aims to minimize is the RSS (residual sums of squares), the sum of the squared distances between points and the centroid of their cluster.

$$\text{RSS} := \sum_{j\in[k]}\sum_{i\in C_j} d(c_j, x_i)^2$$

As the RSS decrease at each iteration of the k-means algorithm, the algorithm therefore converges to a local minimum for the RSS. The number of iterations T for convergence depends on the data and the number of clusters. A single iteration has complexity of $O(knd)$ since the n vectors of dimension d have to be compared to each of the k centroids. The centroids obtained at time t are stored in the matrix $C^t \in \mathbb{R}^{k\times d}$, such that the j^{th} row c_j^t for $j \in [k]$ represents the centroid of the cluster $\mathcal{C}_j^t$.

From a computational complexity point of view, we recall that it is NP-hard to find a clustering that achieves the global minimum for the RSS. There are classical clustering algorithms based on optimizing different loss functions, however the k-means algorithm uses the RSS as the objective function. The algorithm can be super-polynomial in the worst case (the number of iterations is $2^{\omega(\sqrt{n})}$ (Arthur and Vassilvitskii, 2006)), but the number of iterations is usually small in practice. The k-means algorithm with a suitable heuristic like k-means++ (described later on) to initialize the centroids finds a clustering such that the value for the RSS objective function is within a multiplicative $O(\log n)$ factor of the minimum value (Arthur and Vassilvitskii, 2007).

10.1.1 $\delta-$k-means

We now consider a δ-robust version of the k-means in which we introduce some noise. The noise affects the algorithms in both of the steps of k-means: label assignment and centroid estimation.

- Let c_i^* be the closest centroid to the data point x_i. Then, the set of possible labels $L_\delta(x_i)$ for x_i is defined as follows:

  $$L_\delta(x_i) = \{c_p : |d^2(c_i^*, x_i) - d^2(c_p, x_i)| \leq \delta\ \}$$

  The assignment rule selects arbitrarily a cluster label from the set $L_\delta(x_i)$.

- We add $\delta/2$ noise during the calculation of the centroid. Let $\mathcal{C}_j^{t+1}$ be the set of points which have been labeled by j in the previous step. For δ-k-means we pick a centroid $c^{t+1}_j $ with the property that:

$$\left\| c_j^{t+1} - \frac{1}{|\mathcal{C}_j^{t+1}|} \sum_{x_i \in \mathcal{C}_j^{t+1}} x_i \right\| < \frac{\delta}{2}.$$

One way to see this is to perturb the centroid with some noise.

Let us add two remarks on the δ-k-means. First, for a dataset that is expected to have clusterable data, and for a small δ, the number of vectors on the boundary that risk to be misclassified in each step, that is the vectors for which $|L_\delta(x_i)| > 1$ is typically much smaller compared to the vectors that are close to a unique centroid. Second, we also increase by $\delta/2$ the convergence threshold from the k-means algorithm. All in all, δ-k-means is able to find a clustering that is robust when the data points and the centroids are perturbed with some noise of magnitude $O(\delta)$. As we will see in this work, q-means is the quantum equivalent of δ-k-means.

10.2 The q-means algorithm

The q-means algorithm is given as Algorithm 10.1. At a high level, it follows the same steps as the classical k-means algorithm (and the EM algorithm for GMM), where we now use quantum subroutines for distance estimation, finding the minimum value among a set of elements, matrix multiplication for obtaining the new centroids as quantum states, and efficient tomography. First, we pick some random initial points, using the quantum version of a classical techniques (like the k-means++ idea (Arthur and Vassilvitskii, 2007), for which we give a quantum algorithm later). Then, in Steps 1 and 2 all data points are assigned to a cluster. In Steps 3 and 4 we update the centroids of the clusters and retrieve the information classically. The process is repeated until convergence.

10.2.1 Step 1: Centroid distance estimation

The first step of the algorithm estimates the square distance between data points and clusters using a quantum procedure.

Theorem 10.1 (Centroid Distance estimation)**.** *Let a data matrix $V \in \mathbb{R}^{n\times d}$ and a centroid matrix $C \in \mathbb{R}^{k\times d}$ be stored in QRAM, such that the following unitaries $|i\rangle|0\rangle \mapsto |i\rangle|x_i\rangle$, and $|j\rangle|0\rangle \mapsto |j\rangle|c_j\rangle$ can be performed in time $O(\log(nd))$ and the norms of the vectors are known. For any $\Delta > 0$ and $\epsilon_1 > 0$, there exists a quantum algorithm that performs the mapping*

$$\frac{1}{\sqrt{n}}\sum_{i=1}^{n} |i\rangle \otimes_{j\in[k]} (|j\rangle|0\rangle) \mapsto \ \frac{1}{\sqrt{n}}\sum_{i=1}^{n} |i\rangle \otimes_{j\in[k]} (|j\rangle|\overline{d^2(x_i, c_j)}\rangle),$$

Algorithm *q*-means.

```
Require: Data matrix $X \in \mathbb{R}^{n \times d}$ stored in QRAM data structure. Precision
    parameters $\delta$ for $k$-means, error parameters $\epsilon_1$ for distance estimation, $\epsilon_2$
    and $\epsilon_3$ for matrix multiplication and $\epsilon_4$ for tomography.
Ensure: Outputs vectors $c_1, c_2, \cdots, c_k \in \mathbb{R}^d$ that correspond to the centroids
    at the final step of the $\delta$-$k$-means algorithm.

    Select $k$ initial centroids $c_1^0, \cdots, c_k^0$ and store them in QRAM data structure.
    t=0
    repeat
    Step 1: Centroid Distance Estimation
    Perform the mapping
        $$\frac{1}{\sqrt{N}} \sum_{i=1}^{n} |i\rangle \otimes_{j \in [k]} |j\rangle |0\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{i=1}^{n} |i\rangle \otimes_{j \in [k]} |j\rangle |\overline{d^2(x_i, c_j^t)}\rangle \qquad (1)$$
    where $|\overline{d^2(x_i, c_j^t)} - d^2(x_i, c_j^t)| \leq \epsilon_1$.

    Step 2: Cluster Assignment
    Find the minimum distance among $\{d^2(x_i, c_j^t)\}_{j \in [k]}$, then uncompute Step
    1 to create the superposition of all points and their labels
        $$\frac{1}{\sqrt{n}} \sum_{i=1}^{n} |i\rangle \otimes_{j \in [k]} |j\rangle |\overline{d^2(x_i, c_j^t)}\rangle \mapsto \frac{1}{\sqrt{n}} \sum_{i=1}^{n} |i\rangle |\ell^t(x_i)\rangle \qquad (2)$$
    Step 3: Centroid states creation
    3.1 Measure the label register to obtain a state $|\chi_j^t\rangle = \frac{1}{\sqrt{|\mathcal{C}_j^t|}} \sum_{i \in \mathcal{C}_j^t} |i\rangle$, with
    prob. $\frac{|\mathcal{C}_j^t|}{N}$
    3.2 Perform matrix multiplication with matrix $V^T$ and vector $|\chi_j^t\rangle$ to ob-
    tain the state $|c_j^{t+1}\rangle$ with error $\epsilon_2$, along with an estimation of $\left\|c_j^{t+1}\right\|$ with
    relative error $\epsilon_3$

    Step 4: Centroid Update
    4.1 Perform tomography for the states $|c_j^{t+1}\rangle$ with precision $\epsilon_4$ using the op-
    eration from Steps 1-3 and get a classical estimate $\overline{c}_j^{t+1}$ for the new centroids
    such that $|c_j^{t+1} - \overline{c}_j^{t+1}| \leq \sqrt{\eta}(\epsilon_3 + \epsilon_4) = \epsilon_{centroids}$
    4.2 Update the QRAM data structure for the centroids with the new vectors
    $\overline{c}_0^{t+1} \cdots \overline{c}_k^{t+1}$.
    t=t+1
    until convergence condition is satisfied.
```

Figure 10.1: The q-means algorithm

where $|\overline{d^2(x_i, c_j)} - d^2(x_i, c_j)| \leqslant \epsilon_1$ *with probability at least* $1 - 2\Delta$, *in time* $\widetilde{O}\left(k \frac{\eta}{\epsilon_1} \log(1/\Delta)\right)$ *where* $\eta = \max_i(\|x_i\|^2)$.

The proof of the theorem follows rather straightforwardly from lemma 5.6. In fact one just needs to apply the distance estimation procedure k times. Note also that the norms of the centroids are always smaller than the maximum norm of a data point which gives us the factor η.

10.2.2 Step 2: Cluster assignment

At the end of Step 1, we have coherently estimated the square distance between each point in the dataset and the k centroids in separate registers. We can now select the index j that corresponds to the centroid closest to the given data point, written as $\ell(x_i) = \text{argmin}_{j \in [k]}(d(x_i, c_j))$. As taking the square of a number is a monotone function, we do not need to compute the square root of the distance in order to find $\ell(x_i)$.

Lemma 10.1 (Circuit for finding the minimum of k registers)**.** *Given* k *different* $\log p$*-bit registers* $\otimes_{j \in [k]}|a_j\rangle$, *there is a quantum circuit* U_{min} *that maps in time* $O(k \log p)$

$$(\otimes_{j \in [p]}|a_j\rangle)|0\rangle \rightarrow (\otimes_{j \in [k]}|a_j\rangle)|argmin(a_j)\rangle.$$

Proof. We append an additional register for the result that is initialized to $|1\rangle$. We then repeat the following operation for $2 \leq j \leq k$, we compare registers 1 and j, if the value in register j is smaller we swap registers 1 and j and update the result register to j. The cost of the procedure is $O(k \log p)$. □

The cost of finding the minimum is $\widetilde{O}(k)$ in step 2 of the q-means algorithm, while we also need to uncompute the distances by repeating Step 1. Once we apply the minimum finding lemma 10.1 and undo the computation we obtain the state

$$|\psi^t\rangle := \frac{1}{\sqrt{N}} \sum_{i=1}^{n} |i\rangle|\ell^t(x_i)\rangle. \tag{10.1}$$

10.2.3 Step 3: Centroid state creation

The previous step gave us the state $|\psi^t\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{n} |i\rangle|\ell^t(x_i)\rangle$. The first register of this state stores the index of the data points while the second register stores the label for the data point in the current iteration. Given these states, we need to find the new centroids $|c_j^{t+1}\rangle$, which are the average of the data points having the same label.

Let $\chi_j^t \in \mathbb{R}^N$ be the characteristic vector for cluster $j \in [k]$ at iteration t scaled to unit ℓ_1 norm, that is $(\chi_j^t)_i = \frac{1}{|C_j^t|}$ if $i \in \mathcal{C}_j$ and 0 if $i \notin \mathcal{C}_j$. The creation of the quantum states corresponding to the centroids is based on the following simple claim.

Proposition 10.1. *Let $\chi_j^t \in \mathbb{R}^N$ be the scaled characteristic vector for $\mathcal{C}_j$ at iteration t and $X \in \mathbb{R}^{n \times d}$ be the data matrix, then $c_j^{t+1} = X^T \chi_j^t$.*

Proof. The k-means update rule for the centroids is given by $c_j^{t+1} = \frac{1}{|C_j^t|}\sum_{i \in C_j} x_i$. As the columns of X^T are the vectors x_i, this can be rewritten as $c_j^{t+1} = X^T \chi_j^t$. □

The above claim allows us to compute the updated centroids c_j^{t+1} using quantum linear algebra operations. In fact, the state $|\psi^t\rangle$ can be written as a weighted superposition of the characteristic vectors of the clusters.

$$|\psi^t\rangle = \sum_{j=1}^{k} \sqrt{\frac{|C_j|}{N}} \left(\frac{1}{\sqrt{|C_j|}} \sum_{i \in \mathcal{C}_j} |i\rangle \right) |j\rangle = \sum_{j=1}^{k} \sqrt{\frac{|C_j|}{N}} |\chi_j^t\rangle |j\rangle$$

By measuring the last register, we can sample from the states $|\chi_j^t\rangle$ for $j \in [k]$, with probability proportional to the size of the cluster. We assume here that all k clusters are non-vanishing, in other words they have size $\Omega(n/k)$. Given the ability to create the states $|\chi_j^t\rangle$ and given that the matrix V is stored in QRAM, we can now perform quantum matrix multiplication by X^T to recover an approximation of the state $|X^T \chi_j\rangle = |c_j^{t+1}\rangle$ with error ϵ_2, as stated in theorem 5.11. Note that the error ϵ_2 only appears inside a logarithm. The same theorem allows us to get an estimate of the norm $\left\|X^T \chi_j^t\right\| = \left\|c_j^{t+1}\right\|$ with relative error ϵ_3. For this, we also need an estimate of the size of each cluster, namely the norms $\left\|\chi_j\right\|$. We already have this, since the measurements of the last register give us this estimate, and since the number of measurements made is large compared to k (they depend on d), the error from this source is negligible compared to other errors.

The running time of this step is derived from theorem 5.11 where the time to prepare the state $|\chi_j^t\rangle$ is the time of Steps 1 and 2. Note that we do not have to add an extra k factor due to the sampling, since we can run the matrix multiplication procedures in parallel for all j so that every time we measure a random $|\chi_j^t\rangle$ we perform one more step of the corresponding matrix multiplication. Assuming that all clusters have size $\Omega(N/k)$ we will have an extra factor of $O(\log k)$ in the running time by a standard coupon collector argument. We set the error on the matrix multiplication to be $\epsilon_2 \ll \frac{{\epsilon_4}^2}{d \log d}$ as we need to call the unitary that builds c_j^{t+1} for $O(\frac{d \log d}{{\epsilon_4}^2})$ times. We will see that this does not increase the runtime of the algorithm, as the dependence of the runtime for matrix multiplication is logarithmic in the error.

10.2.4 Step 4: Centroid update

In Step 4, we need to go from quantum states corresponding to the centroids, to a classical description of the centroids in order to perform the update step.

For this, we will apply the ℓ_2 vector state tomography algorithm, i.e theorem 3.15, on the states $|c_j^{t+1}\rangle$ that we create in Step 3. Note that for each $j \in [k]$ we will need to invoke the unitary that creates the states $|c_j^{t+1}\rangle$ a total of $O(\frac{d \log d}{\epsilon_4^2})$ times for achieving $\left\| |c_j\rangle - |\overline{c_j}\rangle \right\| < \epsilon_4$. Hence, for performing the tomography of all clusters, we will invoke the unitary $O(\frac{k(\log k) d(\log d)}{\epsilon_4^2})$ times where the $O(k \log k)$ term is the time to get a copy of each centroid state.

The vector state tomography gives us a classical estimate of the unit norm centroids within error ϵ_4, that is $\left\| |c_j\rangle - |\overline{c_j}\rangle \right\| < \epsilon_4$. Using the approximation of the norms $\left\|c_j\right\|$ with relative error ϵ_3 from Step 3, we can combine these estimates to recover the centroids as vectors. The analysis is described in the following proposition:

Proposition 10.2. *Let ϵ_4 be the error we commit in estimating $|c_j\rangle$ such that $\left\| |c_j\rangle - |\overline{c_j}\rangle \right\| < \epsilon_4$, and ϵ_3 the error we commit in the estimating the norms, $|\left\|c_j\right\| - \overline{\left\|c_j\right\|}| \leq \epsilon_3 \left\|c_j\right\|$. Then $\left\|\overline{c_j} - c_j\right\| \leq \sqrt{\eta}(\epsilon_3 + \epsilon_4) = \epsilon_{centroid}$.*

Proof. We can rewrite $\left\|c_j - \overline{c_j}\right\|$ as $\left\| \left\|c_j\right\| |c_j\rangle - \overline{\left\|c_j\right\|} |\overline{c_j}\rangle \right\|$. It follows from triangle inequality that:

$$\left\| \overline{\left\|c_j\right\|} |\overline{c_j}\rangle - \left\|c_j\right\| |c_j\rangle \right\| \leq \left\| \overline{\left\|c_j\right\|} |\overline{c_j}\rangle - \left\|c_j\right\| |\overline{c_j}\rangle \right\| + \left\| \left\|c_j\right\| |\overline{c_j}\rangle - \left\|c_j\right\| |c_j\rangle \right\|$$

We have the upper bound $\left\|c_j\right\| \leq \sqrt{\eta}$. Using the bounds for the error we have from tomography and norm estimation, we can upper bound the first term by $\sqrt{\eta}\epsilon_3$ and the second term by $\sqrt{\eta}\epsilon_4$. The claim follows. □

10.2.4.1 Tomography on approximately pure states

Let us make a remark about the ability to use theorem 3.15 to perform tomography in our case. The updated centroids will be recovered in Step 4 using the vector state tomography algorithm in theorem 3.15 on the composition of the unitary that prepares $|\psi^t\rangle$ and the unitary that multiplies the first register of $|\psi^t\rangle$ by the matrix V^T. The input of the tomography algorithm requires a unitary U such that $U|0\rangle = |x\rangle$ for a fixed quantum state $|x\rangle$. However, the labels $\ell(x_i)$ are not deterministic due to errors in distance estimation, hence the composed unitary U as defined above therefore does not produce a fixed pure state $|x\rangle$. Nevertheless, two different runs of the same unitary returns a quantum state which we can think of a mixed state that is a good approximation of a pure state. (Remark that in this notes we never discussed nor defined mixed states, but it's simple to get an idea of what they are by searching online, or looking at (Nielsen and Chuang, 2002)$).

We therefore need a procedure that finds labels $\ell(x_i)$ that are a deterministic function of x_i and the centroids c_j for $j \in [k]$. One solution is to change the update rule of the δ-k-means algorithm to the following: Let $\ell(x_i) = j$ if $\overline{d(x_i, c_j)} < \overline{d(x_i, c_{j'})} - 2\delta$ for $j' \neq j$ where we discard the points to which no label can be assigned. This assignment rule ensures that if the second register is measured and found to be in state $|j\rangle$, then the first register contains a uniform superposition of points from cluster j that are δ far from the cluster boundary (and possibly a few points that are δ close to the cluster boundary). Note that this simulates exactly the δ-k-means update rule while discarding some of the data points close to the cluster boundary. The k-means centroids are robust under such perturbations, so we expect this assignment rule to produce good results in practice.

A better solution is to use consistent phase estimation instead of the usual phase estimation for the distance estimation step , which can be found in Ta-Shma (2013); Ambainis (2012b), and we briefly discussed in section 5.1 The distance estimates are generated by the phase estimation algorithm applied to a certain unitary in the amplitude estimation step. The usual phase estimation algorithm does not produce a deterministic answer and instead for each eigenvalue λ outputs with high probability one of two possible estimates $\overline{\lambda}$ such that $|\lambda - \overline{\lambda}| \leq \epsilon$. Instead, here as in some other applications we need the consistent phase estimation algorithm that with high probability outputs a deterministic estimate such that $|\lambda - \overline{\lambda}| \leq \epsilon$.

For what follows, we assume that indeed the state in Equation (10.1) is almost a pure state, meaning that when we repeat the procedure we get the same state with very high probability.

10.2.5 Initialization of q-means++

Before running k-means, one usually chooses the first k centroids by using the k-means++ technique from (Arthur and Vassilvitskii, 2007). A first centroid is chosen uniformly at random and we compute its distance to all points of the dataset. Then we sample one point with a weighted probability distribution corresponding to their squared distance to the centroid. We repeat the previous step with this new point as centroid, until k centroids have been chosen.

A quantum analogue, q-means++, can be implemented efficiently using the distance subroutine, Lemma 5.6. Starting with a random index i we compute the following state in time $\widetilde{O}(\frac{\eta}{\epsilon_1})$:

$$|i\rangle \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} |j\rangle |d(x_i, x_j)\rangle$$

Where x_i is the initial centroid. We can then convert the distance register as amplitudes using a controlled rotation after a simple arithmetic circuit.

$$|i\rangle \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} |j\rangle |d(x_i, x_j)\rangle \left(\frac{d(x_i, x_j)}{2\eta} |0\rangle + \beta |1\rangle \right)$$

Each distance has been normalized by $2\eta \geq max_{i,j}(d(x_i, x_j))$ to be a valid amplitude. After undoing the distance computation subroutine in the second register, we perform an amplitude amplification on $|0\rangle$. This creates the state

$$|++\rangle := \frac{1}{Z}\sum_{j=0}^{n-1} d(x_i, x_j)|j\rangle$$

where Z is the normalization factor $\sqrt{\sum_{j=0}^{n-1} d^2(x_i, x_j)}$. We can sample a value j that will represent the next centroid chosen for iteration $t = 0$. To create the state $|{+}{+}\rangle$ we need to perform amplitude amplification, and repeat $O(1/\sqrt{P(0)})$ times the distance estimation procedure, with $P(0)$ being the probability of measuring $|0\rangle$. Since $P(0) = \frac{1}{n}\left(\sum \frac{d(x_i,x_j)}{2\eta}\right)^2$

$$\frac{1}{\sqrt{P(0)}} = \frac{2\eta}{\sqrt{\frac{1}{N}\left(\sum d^2(x_i, x_j)\right)^2}} \leq \frac{2\eta}{\sqrt{\frac{1}{N}\sum d^2(x_i, x_j)}}$$

In the end we repeat $k-1$ times this circuit, for a total time of $\widetilde{O}(k\frac{4\eta^2}{\epsilon_1\sqrt{\mathbb{E}(d^2(x_i,x_j))}})$. In order to be adapt this initialization subroutine with δ-k-means algorithm, it suffice to pick $\epsilon_1 < \delta/2$.

10.3 Analysis

We provide the theorem of the running time and accuracy of the q-means algorithm.

Theorem 10.2 (q-means iteration)**.** *For a data matrix $X \in \mathbb{R}^{n\times d}$ for which we have quantum access, and parameter $\delta > 0$, the q-means algorithm with high probability outputs centroids consistent with the classical δ-k-means algorithm, in time*

$$\widetilde{O}\left(kd\frac{\eta}{\delta^2}\kappa(X)(\mu(X) + k\frac{\eta}{\delta}) + k^2\frac{\eta^{1.5}}{\delta^2}\kappa(V)\mu(V)\right)$$

per iteration.

We prove the theorem in the next two sections.

10.3.1 Error analysis

In this section we determine the error parameters in the different steps of the quantum algorithm so that the quantum algorithm behaves the same as the classical δ-k-means. More precisely, we will determine the values of the errors $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4$ in terms of δ. In this way, the cluster assignment of all data points made by the q-means algorithm is consistent with a classical run of the δ-k-means algorithm, and also that the centroids computed by the q-means after each iteration are again consistent with centroids that can be returned by the δ-k-means algorithm.

The cluster assignment in q-means happens in two steps. The first step estimates the square distances between all points and all centroids. The error in this procedure is of the form

$$|\overline{d^2(c_j, x_i)} - d^2(c_j, x_i)| < \epsilon_1.$$

for a point x_i and a centroid c_j.

The second step finds the minimum of these distances without adding any error. For the q-means to output a cluster assignment consistent with the δ-k-means algorithm, we require that:

$$\forall j \in [k], \quad |\overline{d^2(c_j, x_i)} - d^2(c_j, x_i)| \leq \frac{\delta}{2}$$

which implies that no centroid with distance more than δ above the minimum distance can be chosen by the q-means algorithm as the label. Thus we need to take $\epsilon_1 < \delta/2$.

After the cluster assignment of the q-means (which happens in superposition), we update the clusters, by first performing a matrix multiplication to create the centroid states and estimate their norms, and then a tomography to get a classical description of the centroids. The error in this part is $\epsilon_{centroids}$, as defined in Claim 10.2, namely:

$$\left\|\overline{c}_j - c_j\right\| \leq \epsilon_{centroid} = \sqrt{\eta}(\epsilon_3 + \epsilon_4).$$

Again, for ensuring that the q-means is consistent with the classical δ-k-means algorithm we take $\epsilon_3 < \frac{\delta}{4\sqrt{\eta}}$ and $\epsilon_4 < \frac{\delta}{4\sqrt{\eta}}$. Note also that we have ignored the error ϵ_2 that we can easily deal with, since it only appears in a logarithmic factor in the runtime.

10.3.2 Runtime analysis

As the classical algorithm, the runtime of q-means depends linearly on the number of iterations, so here we analyze the cost of a single step. The cost of tomography for the k centroid vectors is $O(\frac{kd\log k\log d}{{\epsilon_4}^2})$ times the cost of preparation of a single centroid state $|c_j^t\rangle$. A single copy of $|c_j^t\rangle$ is prepared applying the matrix multiplication by V^T procedure on the state $|\chi_j^t\rangle$ obtained using square distance estimation. The time required for preparing a single copy of $|c_j^t\rangle$ is $O(\kappa(V)(\mu(V) + T_\chi)\log(1/\epsilon_2))$ by theorem 5.11 where T_χ is the time for preparing $|\chi_j^t\rangle$. The time T_χ is $\widetilde{O}\left(\frac{k\eta\log(\Delta^{-1})\log(nd)}{\epsilon_1}\right) = \widetilde{O}(\frac{k\eta}{\epsilon_1})$ by lemma 5.6. The cost of norm estimation for k different centroids is independent of the tomography cost and is $\widetilde{O}(\frac{kT_\chi\kappa(V)\mu(V)}{\epsilon_3})$. Combining together all these costs and suppressing all the logarithmic factors we have a total running time of:

$$\widetilde{O}\left(kd\frac{1}{\epsilon_4^2}\kappa(X)(\mu(X)+k\frac{\eta}{\epsilon_1})+k^2\frac{\eta}{\epsilon_3\epsilon_1}\kappa(X)\mu(X)\right) \tag{10.2}$$

The analysis in the previous section shows that we can take $\epsilon_1 = \delta/2$, $\epsilon_3 = \frac{\delta}{4\sqrt{\eta}}$ and $\epsilon_4 = \frac{\delta}{4\sqrt{\eta}}$. Substituting these values in the above running time, it follows that the running time of the q-means algorithm is:

$$\widetilde{O}\left(kd\frac{\eta}{\delta^2}\kappa(V)\left(\mu(V)+k\frac{\eta}{\delta}\right)+k^2\frac{\eta^{1.5}}{\delta^2}\kappa(V)\mu(V)\right).$$

This completes the proof of the theorem.

A few concluding remarks regarding the running time of q-means. For dataset where the number of points is much bigger compared to the other parameters, the running time for the q-means algorithm is an improvement compared to the classical k-means algorithm. For instance, for most problems in data analysis, k is eventually small (< 100). The number of features $d \leq n$ in most situations, and it can eventually be reduced by applying a quantum dimensionality reduction algorithm first (which have running time polylogarithmic in d). To sum up, q-means has the same output as the classical δ-k-means algorithm (which approximates k-means), it conserves the same number of iterations, but has a running time only polylogarithmic in n, giving an exponential speedup with respect to the size of the dataset.

Chapter 11

Quantum Expectation-Maximization

In this chapter we discuss the quantum version of Expectation-Maximization (EM). EM is an iterative algorithm that have been broadly used (and re-discovered) in many part of machine learning and statistics. As is common in machine learning literature, we introduce the Expectation-Maximization algorithm by using it to fit Gaussian mixture models (GMM). As the name hints, a Gaussian mixture model is a way of describing a probability density function as a combination of different Gaussian distributions. GMM, and in general all the mixture models, are a popular generative model in machine learning.

11.1 Expectation-Maximization for GMM

The intuition behind mixture models is to model complicated distributions by using a group of simpler (usually uni-modal) distributions. In this setting, the purpose of the learning algorithm is to model the data by fitting the joint probability distribution which most likely have generated our samples. It might not be surprising thinking that, given a sufficiently large number of mixture components, it is possible to approximate any density defined in $\mathbb{R}^d$ (Murphy, 2012). In this section we describe formally GMM, which is a popular mixture model used to solve unsupervised classification problems. Then we propose the first quantum algorithm to fit a GMM with a quantum computer. While there are many classical algorithms that can be used to fit a mixture of Gaussians (which we detail later), this quantum algorithm resemble as much as possible Expectation-Maximization: a iterative method to find (local) maxima of maximum likelihood and maximum a posteriori optimization problems, especially used in unsupervised settings.

Recall that in the unsupervised case, we are given a training set of unlabeled vectors $v_1 \cdots v_n \in \mathbb{R}^d$ which we represent as rows of a matrix $V \in \mathbb{R}^{n \times d}$. Let $y_i \in [k]$ one of the k possible labels for a point v_i. We posit that for a GMM the joint probability distribution of the data $p(v_i, y_i) = p(v_i|y_i)p(y_i)$, is defined as follow: $y_i \sim \text{Multinomial}(\theta)$ for $\theta \in \mathbb{R}^{k-1}$, and $p(v_i|y_i = j) \sim \mathcal{N}(\mu_j, \Sigma_j)$. The θ_j are the *mixing weights*, i.e. the probabilities that $y_i = j$, and $\mathcal{N}(\mu_j, \Sigma_j)$ is the Gaussian distribution centered in $\mu_j \in \mathbb{R}^d$ with covariance matrix $\Sigma_j \in \mathbb{R}^{d \times d}$.

Note that the variables y_i are unobserved, and thus are called *latent* variables. There is a simple interpretation for this model. We assume the data is created by first selecting an index $j \in [k]$ by sampling according to Multinomial(θ), and then a vector v_i is sampled from $\mathcal{N}(\mu_j, \Sigma_j)$. Fitting a GMM to a dataset reduces to finding an assignment for the parameters:

$$\gamma = (\theta, \vec{\mu}, \vec{\Sigma}) = (\theta, \mu_1, \cdots, \mu_k, \Sigma_1, \cdots, \Sigma_k)$$

that best maximize the log-likelihood (defined in Section 4) for a given dataset. Note that while a μ_j represents a vector, we define $\vec{\mu}$ as the vector of vectors μ_j, and the same goes for $\vec{\Sigma}$. We will now see how the log-likelihood is defined for a GMM. We use the letter ϕ to represent our *base distribution*, which in this case is the probability density function of a Gaussian $\mathcal{N}(\mu, \Sigma)$:

$$\phi(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \tag{11.1}$$

With this formulation, a GMM is expressed as:

$$p(v) = \sum_{j=1}^{k} \theta_j \phi(v; \mu_j, \Sigma_j) \tag{11.2}$$

where θ_j are the *mixing weights* of the multinomial distribution such that $\sum_{j=1}^k \theta_j = 1$. The probability for an observation v_i to be assigned to the component j is given by:

$$r_{ij} = p(y_i = j|v_i; \theta, \mu, \Sigma) = \frac{\theta_j \phi(v_i; \mu_j, \Sigma_j)}{\sum_{l=1}^k \theta_l \phi(v_i; \mu_l, \Sigma_l)}. \tag{11.3}$$

This value is called *responsibility*, and corresponds to the posterior probability of the sample i being assigned label j by the current model. More generally, for any base distribution in the mixture, the responsibility of the i-th vector in cluster j can be written as (Murphy, 2012):

$$r_{ij} = \frac{p(y_i = j; \gamma)p(v_i|y_i = j; \gamma)}{\sum_{j'=1}^k p(y_i = j'; \gamma)p(v_i|y_i = j'; \gamma)} \tag{11.4}$$

As anticipated, to find the best parameters of our generative model, we maximize the log-likelihood of the data. To conclude, for GMM, the likelihood is given by the following formula (Ng, 2012):

$$\ell(\gamma; V) = \ell(\theta, \vec{\mu}, \vec{\Sigma}; V) = \sum_{i=1}^{n} \log p(v_i \,;\, \theta, \vec{\mu}, \vec{\Sigma}) = \sum_{i=1}^{n} \log \sum_{y_i=1}^{k} p(v_i | y_i \,;\, \vec{\mu}, \vec{\Sigma}) p(y_i; \theta) \tag{11.5}$$

Alas, it is seldom possible to solve maximum likelihood estimation analytically (i.e. by finding the zeroes of the derivatives of Equation (11.5), and this is one of those cases. Expectation-Maximization is an iterative algorithm that solves numerically the optimization problem of ML estimation. To complicate things, the likelihood function for GMM is not convex, and thus we might find some local minima (Hastie et al., 2009). Note that the algorithm used to fit GMM can return a local minimum which might be different than γ^*: the model that represents the global optimum of the likelihood function.

11.2 Expectation-Maximization

The intuition behind EM is simple. If we were to know the latent variable y_i, then the log-likelihood for GMM would be:

$$\ell(\gamma; V) = \sum_{i=1}^{n} \log p(v_i \mid y_i; \vec{\mu}, \vec{\Sigma}) + \log p(y_i; \theta) \tag{11.6}$$

This formula can be easily maximized with respect to the parameters $\theta, \vec{\mu}$, and $\vec{\Sigma}$. In the Expectation step we calculate the missing variables y_i, given a guess of the parameters $(\theta, \vec{\mu}, \vec{\Sigma})$ of the model. Then, in the Maximization step, we use the estimate of the latent variables obtained in the Expectation step to update the estimate of the parameters. While in the Expectation step we calculate a lower bound on the likelihood, in the Maximization step we maximize it. Since at each iteration the likelihood can only increase, the algorithm is guaranteed to converge, albeit possibly to a local optimum (see (Hastie et al., 2009) for the proof). During the Expectation step all the responsibilities are calculated, while in the Maximization step we update our estimate on the parameters $\gamma^{t+1} = (\theta^{t+1}, \vec{\mu}^{t+1}, \vec{\Sigma}^{t+1})$.

Again, note that the γ^{t+1} might never converge to the global optimum $\gamma^* = \arg\max_\gamma \ell(\gamma; V)$: since Equation (11.5) is non-convex, any randomized algorithm can get stuck in local minima.

The stopping criterion for GMM is usually a threshold on the increment of the log-likelihood: if the log-likelihood changes less than a threshold between two iterations, then the algorithm stops. Notice that, since the value of the log-likelihood significantly depends on the amount of data points in the training sets, it is often preferable to adopt a scale-free stopping criterion, which does not depend on the number of samples. For instance, in the toolkit scikit-learn (Pedregosa et al., 2011) the stopping criterion is given by a tolerance on the average increment of the log-probability, which is chosen to be smaller than a certain ϵ_τ, say 10^{-3}. More precisely, the stopping criterion is $|\mathbb{E}[\log p(v_i;\gamma^t)] - \mathbb{E}[\log p(v_i;\gamma^{t+1})]| < \epsilon_\tau$ which we can estimate as $|\frac{1}{n}\sum_{i=1}^n \log p(v_i;\gamma^t) - \frac{1}{n}\sum_{i=1}^n \log p(v_i;\gamma^{t+1})| < \epsilon_\tau$.

```
Algorithm 1 Expectation-Maximization for GMM
Require: Dataset $V$, tolerance $\tau > 0$.
Ensure: A GMM $\gamma^t = (\theta^t, \vec{\mu}^t, \vec{\Sigma}^t)$ that maximizes locally the likelihood $\ell(\gamma; V)$
    up to tolerance $\tau$.

    Select $\gamma^0 = (\theta^0, \vec{\mu}^0, \vec{\Sigma}^0)$ using some initialization strategies.
$t = 0$
repeat
    Expectation
$\forall i, j$, calculate the responsibilities as:
        $$r_{ij}^t = \frac{\theta_j^t \phi(v_i; \mu_j^t, \Sigma_j^t)}{\sum_{l=1}^k \theta_l^t \phi(v_i; \mu_l^t, \Sigma_l^t)} \tag{1}$$
    Maximization
Update the parameters of the model as:
        $$\theta_j^{t+1} \leftarrow \frac{1}{n}\sum_{i=1}^n r_{ij}^t \tag{2}$$
        $$\mu_j^{t+1} \leftarrow \frac{\sum_{i=1}^n r_{ij}^t v_i}{\sum_{i=1}^n r_{ij}^t} \tag{3}$$
        $$\Sigma_j^{t+1} \leftarrow \frac{\sum_{i=1}^n r_{ij}^t (v_i - \mu_j^{t+1})(v_i - \mu_j^{t+1})^T}{\sum_{i=1}^n r_{ij}^t} \tag{4}$$
    t=t+1
until
        $$|\ell(\gamma^{t-1}; V) - \ell(\gamma^t; V)| < \tau \tag{5}$$
Return $\gamma^t = (\theta^t, \vec{\mu}^t, \vec{\Sigma}^t)$
```

Figure 11.1: Classical Expectation-Maximization for Gaussian mixture models

11.2.1 Initialization strategies for EM

Unlike k-means clustering, choosing a good set of initial parameters for a mixture of Gaussian is by no means trivial, and in multivariate context is known that the solution is problem-dependent. There are plenty of proposed techniques, and here we describe a few of them. Fortunately, these initialization strategies can be directly translated into quantum subroutines without impacting the overall running time of the quantum algorithm.

The simplest technique is called *random EM*, and consists in selecting initial points at random from the dataset as centroids, and sample the dataset to estimate the covariance matrix of the data. Then these estimates are used as the starting configuration of the model, and we may repeat the random sampling until we get satisfactory results.

A more standard technique borrows directly the initialization strategy of k-means++, proposed in (Arthur and Vassilvitskii, 2007), and extends it to make an initial guess for the covariance matrices and the mixing weights. The initial guess for the centroids is selected by sampling from a suitable, easy to calculate distribution. This heuristic works as following: Let c_0 be a randomly selected point of the dataset, as first centroid. The other $k-1$ centroids are selected by selecting a vector v_i with probability proportional to $d^2(v_i, \mu_{l(v_i)})$, where $\mu_{l(v_i)}$ is the previously selected centroid that is the closest to v_i in ℓ_2 distance. These centroids are then used as initial centroids for a round of k-means algorithm to obtain $\mu_1^0 \cdots \mu_j^0$. Then, the covariance matrices can be initialized as $\Sigma_j^0 := \frac{1}{|\mathcal{C}_j|}\sum_{i \in \mathcal{C}_j}(v_i - \mu_j)(v_i - \mu_j)^T$, where $\mathcal{C}_j$ is the set of samples in the training set that have been assigned to the cluster j in the previous round of k-means. The mixing weights are estimated as $\mathcal{C}_j/n$. Eventually Σ_j^0 is regularized to be a PSD matrix.

There are other possible choices for parameter initialization in EM, for instance, based on *Hierarchical Agglomerative Clustering (HAC)* and the *CEM* algorithm. In CEM we run one step of EM, but with a so-called classification step between E and M. The classification step consists in a hard-clustering after computing the initial conditional probabilities (in the E step). The M step then calculates the initial guess of the parameters (Celeux and Govaert, 1992). In the small EM initialization method we run EM with a different choice of initial parameters using some of the previous strategies. The difference here is that we repeat the EM algorithm for a few numbers of iterations, and we keep iterating from the choice of parameters that returned the best partial results. For an overview and comparison of different initialization techniques, we refer to (Blömer and Bujna, 2013) (Biernacki et al., 2003).

11.2.1.1 Special cases of GMM

What we presented in the previous section is the most general model of GMM. For simple datasets, it is common to assume some restrictions on the covariance matrices of the mixtures. The translation into a quantum version of the model should be straightforward. We distinguish between these cases:

- *Soft k-means.* This algorithm is often presented as a generalization of k-means, but it can actually be seen as special case of EM for GMM - albeit with a different assignment rule. In soft k-means, the assignment function is replaced by a softmax function with *stiffness* parameter β. This β represents the covariance of the clusters. It is assumed to be equal for all the clusters, and for all dimensions of the feature space. Gaussian Mixtures with constant covariance matrix (i.e. $\Sigma_j = \beta I$ for $\beta \in \mathbb{R}$) can be interpreted as a kind of soft or fuzzy version of k-means clustering. The probability of a point in the feature space being assigned to a certain cluster j is:
$$r_{ij} = \frac{e^{-\beta\|x_i-\mu_i\|^2}}{\sum_{l=1}^{k} e^{-\beta\|x_i-\mu_l\|^2}}$$
where $\beta > 0$ is the stiffness parameter. This is the case where all the Gaussians have the same diagonal covariance matrix, which is uniform in all directions.
- *Spherical.* In this model, each component has its own covariance matrix, but the variance is uniform in all the directions, thus reducing the covariance matrix to a multiple of the identity matrix (i.e. $\Sigma_j = \sigma_j^2 I$ for $\sigma_j \in \mathbb{R}$).
- *Diagonal.* As the name suggests, in this special case the covariance matrix of the distributions is a diagonal matrix, but different Gaussians might have different diagonal covariance matrices.
- *Tied.* In this model, the Gaussians share the same covariance matrix, without having further restriction on the Gaussian.
- *Full.* This is the most general case, where each of the components of the mixture have a different, SDP, covariance matrix.

11.2.2 Dataset assumptions in GMM

We make explicit an assumption on the dataset, namely that all elements of the mixture contribute proportionally to the total responsibility:

$$\frac{\sum_{i=1}^{n} r_{ij}}{\sum_{i=1}^{n} r_{il}} = \Theta(1) \quad \forall j, l \in [k] \tag{11.7}$$

This is equivalent to assuming that $\theta_j/\theta_l = \Theta(1) \quad \forall j, l \in [k]$. This resembles the assumption of “well-clusterability’ ’ in q-means, which we saw in the previous chapter. The algorithm can be used even in cases where this assumption does not hold. In this case, the running time will include a factor as in Eq. (11.7) which for simplicity we have taken as constant in what follows. Note that classical algorithms would also find it difficult to fit the data in certain cases, for example when some of the clusters are very small. In fact, it is known (and not surprising) that if the statistical distance between the probability density function of two different Gaussian distributions is smaller than 1/2, then we can not tell for a point v from which Gaussian distribution it belongs to, even if we knew the parameters (Moitra, 2018). Only for convenience in the analysis, we also assume the dataset as being normalized such that the shortest vector has norm 1 and define $\eta := max_i \|v_i\|^2$ to be the maximum norm squared of a vector in the dataset.

11.2.2.1 EM and other mixture models

Expectation-Maximization is widely used for fitting mixture models in machine learning (Murphy, 2012). Most mixture models use a base distribution in the exponential family: Poisson (Church and Gale, 1995) (when the observations are a mixture of random counts with a fixed rate of occurrences), Binomial and Multinomial (when the observations have 2 or multiple possible outcomes, like answers in a survey or a vote) and log-normal (Dexter and Tanner, 1972), exponential (when samples have a latent variable that represents a failure of a certain kind, which is often modeled by the exponential distribution) (Ghitany et al., 1994), Dirichlet multinomial (Yin and Wang, 2014), and others.

Besides fitting mixture models based on the exponential family, the EM algorithm has several other applications. It has been used to fit mixtures of experts, mixtures of the student T distribution (which does not belong to the exponential family, and can be fitted with EM using (Liu and Rubin, 1995)) and for factor analysis, probit regression, and learning Hidden Markov Models (Murphy, 2012).

Theorem 11.1 (Multivariate mean-value theorem (Rudin et al., 1964))**.** *Let U be an open set of $\mathbb{R}^d$. For a differentiable functions $f : U \mapsto \mathbb{R}$ it holds that $\forall x, y \in U, \exists c$ such that $f(x) - f(y) = \nabla f(c) \cdot (x - y)$.*

Proof. Define $h : [0, 1] \mapsto U$ where $h(t) = x + t(y - x)$. We can define a new function $g(t) := f \circ h = f(x + t(y - x))$. Note that both functions are differentiable, and so its their composition. Therefore, we can compute the derivative of g using the chain rule: $g' = (f \circ h)' = (f' \circ h)h'$. This gives:

$$g'(t) = (\nabla f(h(t)), h'(t)) = (f(x + t(y - a)), y - x)$$

Because of the one-dimensional Mean Value theorem applied to $g'(t)$, we have that $\exists t_0$ such that $g(1) - g(0) = f(y) - f(x) = g'(t_0) = (f(x + t_0(y - x)), y - x)$. Setting $c = x + t_0(y - x)$, we have that $f(y) - f(x) = \nabla f(c) \cdot (y - x)$. □

Theorem 11.2 (Componentwise *softmax* function is Lipschitz continuous)**.** *For $d > 2$, let $\sigma_j : \mathbb{R}^d \mapsto (0, 1)$ be the softmax function defined as $\sigma_j(v) = \frac{e^{v_j}}{\sum_{l=1}^d e^{v_l}}$. Then σ_j is Lipschitz continuous, with $K \leq \sqrt{2}$.*

Proof. We need to find the K such that for all $x, y \in \mathbb{R}^d$, we have that $\left\|\sigma_j(y) - \sigma_j(x)\right\| \leq K \left\|y - x\right\|$. Observing that σ_j is differentiable and that if we apply Cauchy-Schwarz to the statement of the mean-value-theorem we derive that \$ x,y U, : c\$ such that $\|f(x) - f(y)\| \leq \|\nabla f(c)\|_F \|x - y\|$. So to show Lipschitz continuity it is enough to select $K \leq \left\|\nabla \sigma_j\right\|_F^* = \max_{c \in \mathbb{R}^d} \left\|\nabla \sigma_j(c)\right\|$. The partial derivatives $\frac{d\sigma_j(v)}{dv_i}$ are $\sigma_j(v)(1 - \sigma_j(v))$ if $i = j$ and $-\sigma_i(v)\sigma_j(v)$ otherwise. So $\left\|\nabla \sigma_j\right\|_F^2 = \sum_{i=1}^{d-1}(-\sigma(v)_i \sigma_j(v))^2 + \sigma_j(v)^2(1 - \sigma_j(v))^2 \leq \sum_{i=1}^{d-1} \sigma(v)_i \sigma_j(v) + \sigma_j(v)(1 - \sigma_j(v)) \leq \sigma_j(v) \sum_{i=0}^{d-1} \sigma_i(v) + 1 - \sigma_j(v) \leq 2\sigma_j(v) \leq 2$. In our case we can deduce that: $\left\|\sigma_j(y) - \sigma_j(x)\right\| \leq \sqrt{2} \left\|y - x\right\|$ so $K \leq \sqrt{2}$. □

11.3 Quantum Expectation-Maximization for GMM

In this section, we present a quantum Expectation-Maximization algorithm to fit a GMM. The algorithm can also be adapted to fit other mixtures models where the probability distributions belong to the exponential family. As the GMM is both intuitive and one of the most widely used mixture models, our results are presented for the GMM case.

11.3.0.1 An approximate version of GMM

Here we define an approximate version of GMM, that we fit with QEM algorithm. The difference between this formalization and the original GMM is simple. Here we make explicit in the model the approximation error introduced during the iterations of the training algorithm.

Definition 11.1 (Approximate GMM)**.** Let $\gamma^t = (\theta^t, \vec{\mu}^t, \vec{\Sigma}^t) = (\theta^t, \mu_1^t \cdots \mu_k^t, \Sigma_1^t \cdots \Sigma_k^t)$ a model fitted by the standard EM algorithm from γ^0 an initial guess of the parameters, i.e. γ^t is the error-free model that standard EM would have returned after t iterations. Starting from the same choice of initial parameters γ^0, fitting a GMM with the QEM algorithm with $\Delta = (\delta_\theta, \delta_\mu)$ means returning a model $\overline{\gamma}^t = (\overline{\theta}^t, \overline{\vec{\mu}}^t, \overline{\vec{\Sigma}}^t)$ such that:

- $\left\|\overline{\theta}^t - \theta^t\right\| < \delta_\theta$,
- $\left\|\overline{\mu_j}^t - \mu_j^t\right\| < \delta_\mu$: for all $j \in [k]$,
- $\left\|\overline{\Sigma_j}^t - \Sigma_j^t\right\| \leq \delta_\mu \sqrt{\eta}$: for all $j \in [k]$.

11.3.0.2 Quantum access to the mixture model

Here we explain how to load into a quantum computer a GMM and a dataset represented by a matrix V. This is needed for a quantum computer to be able to work with a machine learning model. The definition of quantum access to other kind of models is analogous. For ease of exposure, we define what does it means to have quantum access to a GMM and its dataset. This definition is basically an extension of theorem **??**.

Definition 11.2 (Quantum access to a GMM)**.** We say that we have quantum access to a GMM of a dataset $V \in \mathbb{R}^{n\times d}$ and model parameters $\theta_j \in \mathbb{R}$, $\mu_j \in \mathbb{R}^d, \Sigma_j \in \mathbb{R}^{d\times d}$ for all $j \in [k]$ if we can perform in time $O(\text{polylog}(d))$ the following mappings:

- $|j\rangle|0\rangle \mapsto |j\rangle|\mu_j\rangle$,
- $|j\rangle|i\rangle|0\rangle \mapsto |j\rangle|i\rangle|\sigma_i^j\rangle$ for $i \in [d]$ where $ _i^{j} $ is the i-th rows of $\Sigma_j \in \mathbb{R}^{d\times d}$,
- $|i\rangle|0\rangle \mapsto |i\rangle|v_i\rangle$ for all $i \in [n]$,
- $|i\rangle|0\rangle|0\rangle \mapsto |i\rangle|\text{vec}[v_iv_i^T]\rangle = |i\rangle|v_i\rangle|v_i\rangle$ for $i \in [n]$,
- $|j\rangle|0\rangle \mapsto |j\rangle|\theta_j\rangle$.

Algorithm 1 QEM for GMM

```
Require: Quantum access to a GMM model, precision parameters $\delta_\theta, \delta_\mu$, and threshold $\epsilon_\tau$.
Ensure: A GMM $\overline{\gamma}^t$ that maximizes locally the likelihood $\ell(\gamma; V)$, up to tolerance $\epsilon_\tau$.

Use a heuristic (like lemma $q$-means++) to determine the initial guess $\gamma^0 = (\theta^0, \vec{\mu}^0, \vec{\Sigma}^0)$, and build quantum access those parameters.
Use lemma ?? to estimate the log determinant of the matrices $\{\Sigma_j^0\}_{j=1}^k$.
t=0
repeat
    Step 1: Get an estimate of $\theta^{t+1}$ using lemma ?? such that $\overline{\theta}^{t+1} - \theta^{t+1} \leq \delta_\theta$.
    Step 2: Get an estimate $\{\overline{\mu_j}^{t+1}\}_{j=1}^k$ by using lemma ?? to estimate each $\mu_j^{t+1}$ and $\mu_j^{t+1}$ such that $\mu_j^{t+1} - \overline{\mu_j}^{t+1} \leq \delta_\mu$.
    Step 3: Get an estimate $\{\overline{\Sigma_j}^{t+1}\}_{j=1}^k$ by using lemma ?? to estimate $\Sigma_j^{t+1}{}_F$ and $\Sigma_j^{t+1}$ such that $\Sigma_j^{t+1} - \overline{\Sigma_j}^{t+1} \leq \delta_\mu\sqrt{\eta}$.
    Step 4: Estimate $E[\overline{p(v_i; \gamma^{t+1})}]$ up to error $\epsilon_\tau/2$ using theorem ??.
    Step 5: Build quantum access to $\gamma^{t+1}$, and use lemma ?? to estimate the determinants $\{\overline{\log det(\Sigma_j^{t+1})}\}_{j=0}^k$.
    $t = t + 1$
until
    $$|E[\overline{p(v_i; \gamma^t)}] - E[\overline{p(v_i; \gamma^{t-1})}]| < \epsilon_\tau$$
Return $\overline{\gamma}^t = (\overline{\theta}^t, \overline{\vec{\mu}}^t, \overline{\vec{\Sigma}}^t)$
```

Figure 11.2: Classical Expectation-Maximization for Gaussian mixture models

11.3.0.3 Quantum initialization strategies

For the initialization of γ^0 in the quantum algorithm we can use the same initialization strategies as in classical machine learning. For instance, we can use the classical **random EM** initialization strategy for QEM.

A quantum initialization strategy can also be given using the **k-means++** initializion strategy, which we discuss in Chapter 10. It returns k initial guesses for the centroids $c_1^0 \cdots c_k^0$ consistent with the classical algorithm in time $\left(k^2 \frac{2\eta^{1.5}}{\epsilon\sqrt{\mathbb{E}(d^2(v_i,v_j))}}\right)$, where $\mathbb{E}(d^2(v_i, v_j))$ is the average squared distance between two points of the dataset, and ϵ is the tolerance in the distance estimation. From there, we can perform a full round of q-means algorithm and get an estimate for $\mu_1^0 \cdots \mu_k^0$. With q-means and the new centroids store in the QRAM we can create the state

$$|\psi^0\rangle := \frac{1}{\sqrt{n}} \sum_{i=1}^{n} |i\rangle|l(v_i)\rangle. \tag{11.8}$$

Where $l(v_i)$ is the label of the closest centroid to the i-th point. By sampling $S \in O(d)$ points from this state we get two things. First, from the frequency f_j of the second register we can have an guess of $\theta_j^0 \leftarrow |\mathcal{C}_j|/n \sim f_j/S$. Then, from the first register we can estimate $\Sigma_j^0 \leftarrow \sum_{i\in S}(v_i - \mu_j^0)(v_i - \mu_j^0)^T$. Sampling $O(d)$ points and creating the state in Equation (11.8) takes time $\widetilde{O}(dk\eta)$ by theorem 5.6 and the minimum finding procedure, i.e. lemma 5.2.

Techniques illustrated in (Miyahara et al., 2020) can also be used to quantize the CEM algorithm which needs a hard-clustering step. Among the different possible approaches, the **random** and the **small EM** greatly benefit from a faster algorithm, as we can spend more time exploring the space of the parameters by starting from different initial seeds, and thus avoid local minima of the likelihood.

11.3.1 Expectation

In this step of the quantum algorithm we are just showing how to compute efficiently the responsibilities as a quantum state. First, we compute the responsibilities in a quantum register, and then we show how to put them as amplitudes of a quantum state. At each iteration of Quantum Expectation-Maximization (specifically, in the Expectation step), we assume to have quantum access to the determinant of the covariance matrices. In the next Chapters we will also detail quantum algorithms for the problem of computing the log-determinant. From the error analysis we will see that the cost of comping the log-determinant of the covariance matrices (even with classical algorithms) is smaller than the cost of the other quantum step, we can discard the cost of computing the log-determinant in the analysis of the quantum algorithms. Thus, we do not explicitly write the time to compute the determinant from now on in the algorithm and when we say that we update Σ we include an update on the estimate of $\log(\det(\Sigma))$ as well. Classical algorithms often depend linearly on $nnz(\Sigma)|\log(\det(\Sigma))|$, which can be upper bounded by $\widetilde{O}(d^2)$, where d is the dimension of the covariance matrix. Note that it is often the case that GMM is run with diagonal covariance matrix, thus making the estimation of the determinant trivial.

Lemma 11.1 (Quantum Gaussian Evaluation)**.** *Suppose we have stored in the QRAM a matrix $V \in \mathbb{R}^{n\times d}$, the centroid $\mu \in \mathbb{R}^d$ and a SPD covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$ of a multivariate Gaussian distribution $\phi(v|\mu, \Sigma)$, such that $\|\Sigma\| \leq 1$. Also assume to have an absolute $\epsilon_1/2$ estimate for* $\log(\det(\Sigma))$*. Then for $\epsilon_1 > 0$, there exists a quantum algorithm that with probability* $1-\gamma$ *performs the mapping $U_{G,\epsilon_1} : |i\rangle|0\rangle \to |i\rangle|\overline{s_i}\rangle$ such that $|s_i - \overline{s_i}| < \epsilon_1$, where $s_i = -\frac{1}{2}((v_i-\mu)^T\Sigma^{-1}(v_i-\mu) + d\log 2\pi + \log(det(\Sigma)))$ is the exponent for the Gaussian probability density function in Equation* (11.1)*. The running time of the algorithm is,*

$$T_{G,\epsilon_1} = O\left(\frac{\kappa(\Sigma)\mu(\Sigma)\log(1/\gamma)}{\epsilon_1}\eta\right).$$

Proof. We use quantum linear algebra and inner product estimation to estimate the quadratic form $(v_i-\mu)^T\Sigma^{-1}(v_i-\mu)$ to error ϵ_1. We decompose the quadratic form as $v_i^T\Sigma^{-1}v_i - 2v_i^T\Sigma^{-1}\mu + \mu^T\Sigma^{-1}\mu$ and separately approximate each term in the sum to error $\epsilon_1/8$ using lemma 5.7. The runtime for this operation is $O(\frac{\mu(\Sigma)\kappa(\Sigma)\eta}{\epsilon_1})$. With this, we obtain an estimate for $\frac{1}{2}((v_i-\mu)^T\Sigma^{-1}(v_i-\mu)$ within error ϵ_1. Recall that (through the algorithm in lemma **??** we also have an estimate of the log-determinant to error $\epsilon_1/2$. With these factors, we obtain an approximation for $-\frac{1}{2}((v_i-\mu)^T\Sigma^{-1}(v_i-\mu) + d\log 2\pi + \log(\det(\Sigma)))$ within error ϵ_1. □

Using controlled operations it is simple to extend the previous theorem to work with multiple Gaussians distributions (μ_j, Σ_j). That is, we can control on a register $|j\rangle$ to do $|j\rangle|i\rangle|0\rangle \mapsto |j\rangle|i\rangle|\phi(v_i|\mu_j,\Sigma_j)\rangle$. In the next lemma we will see how to obtain the responsibilities r_{ij} using the previous theorem and standard quantum circuits for doing arithmetic, controlled rotations, and amplitude amplification. The lemma is stated in a general way, to be used with any probability distributions that belong to an exponential family.

Lemma 11.2 (Error in the responsibilities of the exponential family)**.** *Let $v_i \in \mathbb{R}^n$ be a vector, and let $\{p(v_i|\nu_j)\}_{j=1}^k$ be a set of k probability distributions in the exponential family, defined as $p(v_i|\nu_j) := h_j(v_i)exp\{o_j(\nu_j)^T T_j(v_i) - A_j(\nu_j)\}$. Then, if we have estimates for each exponent with error ϵ, then we can compute each r_{ij} such that $|\overline{r_{ij}} - r_{ij}| \leq \sqrt{2k}\epsilon$ for $j \in [k]$.*

Proof. The proof follows from rewriting the responsibility of Equation (11.3) and (11.4) as:

$$r_{ij} := \frac{h_j(v_i)\exp\{o_j(\nu_j)^T T(v_i) - A_j(\nu_j) \ \ + \log\theta_j\}}{\sum\limits_{l=1}^{k} h_l(v_i)\exp\{o_l(\nu_l)^T T(v_i) - A_l(\nu_l) \ \ + \log\theta_l\}} \tag{11.9}$$

In this form, it is clear that the responsibilities can be seen a softmax function, and we can use theorem 11.2 to bound the error in computing this value. %Note that in this case the error in $\log\theta_j$ is also relative, so it will not impact the whole error in the exponent.

Let $T_i \in \mathbb{R}^k$ be the vector of the exponent, that is $t_{ij} = o_j(\nu_j)^T T(v_i) - A_j(\nu_j) + \log\theta_j$. In an analogous way we define $\overline{T_i}$ the vector where each component is the estimate with error ϵ. The error in the responsibility is defined as $|r_{ij} - \overline{r_{ij}}| = |\sigma_j(T_i) - \sigma_j(\overline{T_i})|$. Because the function σ_j is Lipschitz continuous, as we proved in theorem 11.2 with a Lipschitz constant $K \leq \sqrt{2}$, we have that, $|\sigma_j(T_i) - \sigma_j(\overline{T_i})| \leq \sqrt{2}\left\|T_i - \overline{T_i}\right\|$. The result follows as $\left\|T_i - \overline{T_i}\right\| < \sqrt{k}\epsilon$. □

`The next lemma provides a quantum algorithm for calculating the responsibilities for t`

Lemma 11.3 (Calculating responsibilities). *Suppose we have quantum access to a GMM with parameters $\gamma^t = (\theta^t, \vec{\mu}^t, \vec{\Sigma}^t)$. There are quantum algorithms that can:*

- *Perform the mapping $|i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle|\overline{r_{ij}}\rangle$ such that $|\overline{r_{ij}} - r_{ij}| \leq \epsilon_1$ with probability $1-\gamma$ in time:*

$$T_{R_1,\epsilon_1} = \widetilde{O}(k^{1.5} \times T_{G,\epsilon_1})$$

- *For a given $j \in [k]$, construct state $|\overline{R_j}\rangle$ such that $\left\||\overline{R_j}\rangle - \frac{1}{\sqrt{Z_j}}\sum\limits_{i=0}^{n} r_{ij}|i\rangle\right\| < \epsilon_1$ where $Z_j = \sum\limits_{i=0}^{n} r_{ij}^2$ with high probability in time:*

$$T_{R_2,\epsilon_1} = \widetilde{O}(k^2 \times T_{R_1,\epsilon_1})$$

Proof. For the first statement, we start by recalling the definition of responsibility: $r_{ij} = \frac{\theta_j\phi(v_i;\mu_j,\Sigma_j)}{\sum_{l=1}^{k}\theta_l\phi(v_i;\mu_l,\Sigma_l)}$. With the aid of U_{G,ϵ_1} of lemma 11.1 we can estimate $\log(\phi(v_i|\mu_j,\Sigma_j))$ for all j up to additive error ϵ_1, and then using the current estimate of θ^t, we can calculate the responsibilities create the state,

$$\frac{1}{\sqrt{n}}\sum_{i=0}^{n}|i\rangle\Big(\bigotimes_{j=1}^{k}|j\rangle|\overline{\log(\phi(v_i|\mu_j,\Sigma_j))}\rangle\Big)\otimes|\overline{r_{ij}}\rangle.$$

The estimate $\overline{r_{ij}}$ is computed by evaluating a weighted softmax function with arguments $\overline{\log(\phi(v_i|\mu_j,\Sigma_j)}$ for $j \in [k]$. The estimates $\overline{\log(\phi(v_i|\mu_j,\Sigma_j)}$ are then uncomputed. The runtime of the procedure is given by calling k times lemma 11.1 for Gaussian estimation (the runtime for arithmetic operations to calculate the responsibilities are absorbed).

Let us analyze the error in the estimation of r_{ij}. The responsibility r_{ij} is a softmax function with arguments $\log(\phi(v_i|\mu_j, \Sigma_j))$ that are computed up to error ϵ_1 using lemma 11.1. As the softmax function has a Lipschitz constant $K \leq \sqrt{2}$ by lemma 11.2, we choose precision for lemma 11.1 to be $\epsilon_1/\sqrt{2k}$ to get the guarantee $|\overline{r_{ij}} - r_{ij}| \leq \epsilon_1$. Thus, the total cost of this step is $T_{R_1,\epsilon_1} = k^{1.5} T_{G,\epsilon_1}$.

We we see how to encode this information in the amplitudes, as stated in the second claim of the lemma. We estimate the responsibilities r_{ij} to some precision ϵ and perform a controlled rotation on an ancillary qubit to obtain,

$$\frac{1}{\sqrt{n}}|j\rangle \sum_{i=0}^{n} |i\rangle|\overline{r_{ij}}\rangle\Big(\overline{r_{ij}}|0\rangle + \sqrt{1-\overline{r_{ij}}^2}|1\rangle\Big). \tag{11.10}$$

We then undo the circuit on the second register and perform amplitude amplification on the rightmost auxiliary qubit being $|0\rangle$ to get $|\overline{R_j}\rangle := \frac{1}{\|\overline{R_j}\|}\sum_{i=0}^{n} \overline{r_{ij}}|i\rangle$. The runtime for amplitude amplification on this task is $O(T_{R_1,\epsilon} \cdot \frac{\sqrt{n}}{\|\overline{R_j}\|})$.

Let us analyze the precision ϵ required to prepare $|\overline{R_j}\rangle$ such that $\||R_j\rangle - |\overline{R_j}\rangle\| \leq \epsilon_1$. As we have estimates $|r_{ij} - \overline{r_{ij}}| < \epsilon$ for all i, j, the ℓ_2-norm error $\|R_j - \overline{R_j}\| = \sqrt{\sum_{i=0}^{n} |r_{ij} - \overline{r_{ij}}|^2} < \sqrt{n}\epsilon$.

Applying Claim D.4, the error for the normalized vector $|R_j\rangle$ can be bounded as $\||R_j\rangle - |\overline{R_j}\rangle\| < \frac{\sqrt{2n}\epsilon}{\|R_j\|}$. By the Cauchy-Schwarz inequality we have that $\|R_j\| \geq \frac{\sum_i^n r_{ij}}{\sqrt{n}}$. We can use this to obtain a bound $\frac{\sqrt{n}}{\|R_j\|} < \frac{\sqrt{n}}{\sum_i r_{ij}}\sqrt{n} = O(k)$, using the dataset assumptions in section 11.2.2. If we choose ϵ such that $\frac{\sqrt{2n}\epsilon}{\|R_j\|} < \epsilon_1$, that is $\epsilon \leq \epsilon_1/k$ then our runtime becomes $T_{R_2,\epsilon_1} := \widetilde{O}(k^2 \times T_{R_1,\epsilon_1})$. □

11.3.2 Maximization

Now we need to get a new estimate for the parameters of our model. This is the idea: at each iteration we recover the new parameters from the quantum algorithms as quantum states, and then by performing tomography we can update the QRAM that gives us quantum access to the GMM for the next iteration. In these sections we will show how.

11.3.2.1 Updating mixing weights θ

Lemma 11.4 (Computing mixing weights)**.** *We assume quantum access to a GMM with parameters γ^t and let $\delta_\theta > 0$ be a precision parameter. There exists* *an algorithm that estimates* $\overline{\theta}^{t+1} \in \mathbb{R}^k$ *such that* $\left\|\overline{\theta}^{t+1} - \theta^{t+1}\right\| \leq \delta_\theta$ *in time*

$$T_\theta = O\left(k^{3.5}\eta^{1.5}\frac{\kappa(\Sigma)\mu(\Sigma)}{\delta_\theta^2}\right)$$

Proof. An estimate of θ_j^{t+1} can be recovered from the following operations. First, we use lemma 11.3 (part 1) to compute the responsibilities to error ϵ_1, and then perform the following mapping, which consists of a controlled rotation on an auxiliary qubit:

$$\frac{1}{\sqrt{nk}}\sum_{\substack{i=1\\j=1}}^{n,k}|i\rangle|j\rangle|\overline{r_{ij}}^t\rangle \mapsto \frac{1}{\sqrt{nk}}\sum_{\substack{i=1\\j=1}}^{n,k}|i\rangle|j\rangle\Big(\sqrt{\overline{r_{ij}}^t}|0\rangle + \sqrt{1-\overline{r_{ij}}^t}|1\rangle\Big)$$

The previous operation has a cost of T_{R_1,ϵ_1}, and the probability of getting $|0\rangle$ is $p(0) = \frac{1}{nk}\sum_{i=1}^n\sum_{j=1}^k r_{ij}^t = \frac{1}{k}$. Now observe that, by definition, $\theta_j^{t+1} = \frac{1}{n}\sum_{i=1}^n r_{ij}^t$.

Let $Z_j = \sum_{i=1}^n \overline{r_{ij}}^t$ and define state $|\sqrt{R_j}\rangle = \left(\frac{1}{\sqrt{Z_j}}\sum_{i=1}^n\sqrt{\overline{r_{ij}}^t}|i\rangle\right)|j\rangle$. After amplitude amplification on $|0\rangle$ we have the state,

$$\begin{aligned}|\sqrt{R}\rangle &:= \frac{1}{\sqrt{n}}\sum_{\substack{i=1\\j=1}}^{n,k}\sqrt{\overline{r_{ij}}^t}|i\rangle|j\rangle \\ &= \sum_{j=1}^k\sqrt{\frac{Z_j}{n}}\left(\frac{1}{\sqrt{Z_j}}\sum_{i=1}^n\sqrt{\overline{r_{ij}}^t}|i\rangle\right)|j\rangle \\ &= \sum_{j=1}^k\sqrt{\overline{\theta}_j^{t+1}}|\sqrt{R_j}\rangle|j\rangle. \end{aligned} \tag{11.11}$$

The probability of obtaining outcome $|j\rangle$ if the second register is measured in the standard basis is $p(j) = \overline{\theta}_j^{t+1}$. An estimate for θ_j^{t+1} with precision ϵ can be obtained by either sampling the last register, or by performing amplitude estimation. In this case, we can estimate each of the values θ_j^{t+1} for $j \in [k]$. Sampling requires $O(\epsilon^{-2})$ samples to get epsilon accuracy on θ (by the Chernoff bounds), but does not incur any dependence on k. As the number of cluster k is relatively small compared to $1/\epsilon$, we chose to do amplitude estimation to estimate all θ_j^{t+1} for $j \in [k]$ to error $\epsilon/\sqrt{k}$ in time,

$$T_\theta := O\left(k \cdot \frac{\sqrt{k}T_{R_1,\epsilon_1}}{\epsilon}\right). \tag{11.12}$$

We analyze the error in this procedure. The error introduced by the estimation of responsibility in lemma 11.3 is $|\overline{\theta}_j^{t+1} - \theta_j^{t+1}| = \frac{1}{n}\sum_i |\overline{r_{ij}}^t - r_{ij}^t| \leq \epsilon_1$ for all $j \in [k]$, pushing the error on the vector $\theta^{t+1} \in \mathbb{R}^k$ up to $\left\|\overline{\theta}^{t+1} - \theta^{t+1}\right\| \leq \sqrt{k}\epsilon_1$.

The total error in ℓ_2 norm due to amplitude estimation is at most ϵ as it estimates each coordinate of $\overline{\theta}_j^{t+1}$ to error $\epsilon/\sqrt{k}$. As the errors sums up additively, we can use the triangle inequality to bound them. The total error is at most $\epsilon+\sqrt{k}\epsilon_1$. As we require the final error to be upper bounded by $\left\|\overline{\theta}^{t+1} - \theta^{t+1}\right\| < \delta_\theta$, we choose parameters $\sqrt{k}\epsilon_1 < \delta_\theta/2 \Rightarrow \epsilon_1 < \frac{\delta_\theta}{2\sqrt{k}}$ and $\epsilon < \delta_\theta/2$. With these parameters, the overall running time of the quantum procedure is $T_\theta = O(k^{1.5}\frac{T_{R_1,\epsilon_1}}{\epsilon}) = O\left(k^{3.5}\frac{\eta^{1.5}\cdot\kappa^2(\Sigma)\mu(\Sigma)}{\delta_\theta^2}\right)$. □

11.3.2.2 Updating the centroids μ_j

We use quantum linear algebra to transform the uniform superposition of responsibilities of the j-th mixture into the new centroid of the j-th Gaussian. Let $R_j^t \in \mathbb{R}^n$ be the vector of responsibilities for a Gaussian j at iteration t. The following claim relates the vectors R_j^t to the centroids μ_j^{t+1}.

Lemma 11.5. *Let $R_j^t \in \mathbb{R}^n$ be the vector of responsibilities of the points for the Gaussian j at time t, i.e. $(R_j^t)_i = r_{ij}^t$. Then $\mu_j^{t+1} \leftarrow \frac{\sum_{i=1}^n r_{ij}^t v_i}{\sum_{i=1}^n r_{ij}^t} = \frac{V^T R_j^t}{n\theta_j}$.*

The proof is straightforward.

Lemma 11.6 (Computing new centroids)**.** *We assume we have quantum access to a GMM with parameters γ^t. For a precision parameter $\delta_\mu > 0$, there is a quantum algorithm that calculates $\{\overline{\mu_j}^{t+1}\}_{j=1}^k$ such that for all $j \in [k]$ $\left\|\overline{\mu_j}^{t+1} - \mu_j^{t+1}\right\| \leq \delta_\mu$ in time*

$$T_\mu = \widetilde{O}\left(\frac{kd\eta\kappa(V)(\mu(V) + k^{3.5}\eta^{1.5}\kappa(\Sigma)\mu(\Sigma))}{\delta_\mu^3}\right)$$

Proof. A new centroid μ_j^{t+1} is estimated by first creating an approximation of the state $|R_j^t\rangle$ up to error ϵ_1 in the ℓ_2-norm using part 2 of lemma 11.3. Then, we use the quantum linear algebra algorithms in theorem 5.11 to multiply R_j by V^T, and obtain a state $|\overline{\mu_j}^{t+1}\rangle$ along with an estimate for the norm $\|V^T R_j^t\| = \left\|\overline{\mu_j}^{t+1}\right\|$ with error ϵ_3. The last step of the algorithm consists in estimating the unit vector $|\overline{\mu_j}^{t+1}\rangle$ with precision ϵ_4, using ℓ_2 tomography. The tomography depends linearly on d, which we expect to be bigger than the precision required by the norm estimation. Thus, we assume that the runtime of the norm estimation is absorbed by the runtime of tomography. We obtain a final runtime of $\widetilde{O}\left(k\frac{d}{\epsilon_4^2}\cdot\kappa(V)\left(\mu(V) + T_{R_2,\epsilon_1}\right)\right)$.

We now analyze the total error in the estimation of the new centroids. In order to satisfy the condition of the robust GMM of definition 11.1, we want the error on the centroids to be bounded by δ_μ. For this, Claim D.3 help us choose the parameters such that $\sqrt{\eta}(\epsilon_{tom} + \epsilon_{norm}) = \delta_\mu$. Since the error ϵ_2 for quantum linear algebra appears as a logarithmic factor in the running time, we can choose $\epsilon_2 \ll \epsilon_4$ without affecting the runtime. □

Lemma 11.7 (Computing covariance matrices)**.** *We assume we have quantum access to a GMM with parameters γ^t. We also have computed estimates $\overline{\mu_j}^{t+1}$ of all centroids such that $\left\|\overline{\mu_j}^{t+1} - \mu_j^{t+1}\right\| \leq \delta_\mu$ for precision parameter $\delta_\mu > 0$. Then, there exists a quantum algorithm that outputs estimates for the new covariance matrices $\{\overline{\Sigma}_j^{t+1}\}_{j=1}^k$ such that $\left\|\Sigma_j^{t+1} - \overline{\Sigma}_j^{t+1}\right\|_F \leq \delta_\mu\sqrt{\eta}$ with high probability, in time,*

$$T_\Sigma := \widetilde{O}\Big(\frac{kd^2\eta\kappa(V)(\mu(V') + \eta^{2.5}k^{3.5}\kappa(\Sigma)\mu(\Sigma))}{\delta_\mu^3}\Big)$$

Proof. It is simple to check, that the update rule of the covariance matrix during the maximization step can be reduced to (Murphy, 2012) Exercise 11.2.

$$\Sigma_j^{t+1} \leftarrow \frac{\sum_{i=1}^n r_{ij}(v_i - \mu_j^{t+1})(v_i - \mu_j^{t+1})^T}{\sum_{i=1}^n r_{ij}} = \frac{\sum_{i=1}^n r_{ij} v_i v_i^T}{n\theta_j} - \mu_j^{t+1}(\mu_j^{t+1})^T \tag{11.13}$$

$$= \Sigma'_j - \mu_j^{t+1}(\mu_j^{t+1})^T \tag{11.14}$$

First, let's note that we can use the previously obtained estimates of the centroids to compute the outer product $\mu_j^{t+1}(\mu_j^{t+1})^T$ with error $\delta_\mu \|\mu\| \leq \delta_\mu\sqrt{\eta}$. The error in the estimates of the centroids is $\overline{\mu} = \mu + e$ where e is a vector of norm δ_μ. Therefore $\left\|\mu\mu^T - \overline{\mu}\,\overline{\mu}^T\right\| < 2\sqrt{\eta}\delta_\mu + \delta_\mu^2 \leq 3\sqrt{\eta}\delta_\mu$. Because of this, we allow an error of $\sqrt{\eta}\delta_\mu$ also for the term Σ'_j. Now we discuss the procedure for estimating Σ'_j. We estimate $|\text{vec}[\Sigma'_j]\rangle$ and $\|\text{vec}[\Sigma'_j]\|$. To do it, we start by using quantum access to the norms and part 1 of lemma 11.3. With them, for a cluster j, we start by creating the state $|j\rangle\frac{1}{\sqrt{n}}\sum_i |i\rangle|r_{ij}\rangle$, Then, we use quantum access to the norms to store them into another register $|j\rangle\frac{1}{\sqrt{n}}\sum_i |i\rangle|r_{ij}\rangle|\,\|v_i\|\rangle$. Using an ancilla qubit we can obtain perform a rotation, controlled on the responsibilities and the norm, and obtain the following state:

$$|j\rangle\frac{1}{\sqrt{n}}\sum_i^n |i\rangle|r_{ij}\rangle|\,\|v_i\|\rangle\left(\frac{r_{ij}\,\|v_i\|}{\sqrt{\eta}}|0\rangle + \gamma|1\rangle\right)$$

We undo the unitary that created the responsibilities in the second register and the query on the norm on the third register, and we perform amplitude amplification on the ancilla qubit being zero. The resulting state can be obtained in time $O(R_{R_1,\epsilon_1}\frac{\sqrt{n\eta}}{\|V_R\|})$, where $\|V_R\|$ is $\sqrt{\sum_i r_{ij}^2\left\|v_i\right\|^2}$. Successively, we query the QRAM for the vectors v_i and we obtain the following state:

$$\frac{1}{V_R}\sum_i r_{ij}\left\|v_i\right\||i\rangle|v_i\rangle \tag{11.15}$$

On which we can apply quantum linear algebra subroutine, multiplying the first register with the matrix V^T. This will lead us to the desired state $|\Sigma_j'\rangle$, along with an estimate of its norm.

As the runtime for the norm estimation $\frac{\kappa(V)(\mu(V)+T_{R_2,\epsilon_1}))\log(1/\epsilon_{mult})}{\epsilon_{norms}}$ does not depend on d, we consider it smaller than the runtime for performing tomography. Thus, the runtime for this operation is:

$$O(\frac{d^2\log d}{\epsilon_{tom}^2}\kappa(V)(\mu(V)+T_{R_2,\epsilon_1}))\log(1/\epsilon_{mult})).$$

Let's analyze the error of this procedure. We want a matrix $\overline{\Sigma_j'}$ that is $\sqrt{\eta}\delta_\mu$-close to the correct one: $\left\|\overline{\Sigma_j'}-\Sigma_j'\right\|_F=\left\|\text{vec}[\overline{\Sigma_j'}]-\text{vec}[\Sigma_j']\right\|_2<\sqrt{\eta}\delta_\mu$. Again, the error due to matrix multiplication can be taken as small as necessary, since is inside a logarithm. From Claim D.3, we just need to fix the error of tomography and norm estimation such that $\eta(\epsilon_{unit}+\epsilon_{norms})<\sqrt{\eta}\delta_\mu$ where we have used η as an upper bound on $\left\|\Sigma_j\right\|_F$. For the unit vectors, we require $\left\||\Sigma_j'\rangle-\overline{|\Sigma_j'\rangle}\right\|\leq\left\|\overline{|\Sigma_j'\rangle}-\widehat{|\Sigma_j'\rangle}\right\|+\left\|\widehat{|\Sigma_j'\rangle}-|\Sigma_j'\rangle\right\|<\epsilon_4+\epsilon_1\leq\eta\epsilon_{unit}\leq\frac{\delta_\mu\sqrt{\eta}}{2}$, where $\overline{|\Sigma_j'\rangle}$ is the error due to tomography and $\widehat{|\Sigma_j'\rangle}$ is the error due to the responsibilities in lemma 11.3. For this inequality to be true, we choose $\epsilon_4=\epsilon_1<\frac{\delta_\mu/\sqrt{\eta}}{4}$.

The same argument applies to estimating the norm $\left\|\Sigma_j'\right\|$ with relative error : $|\left\|\Sigma_j'\right\|-\overline{\left\|\Sigma_j'\right\|}|\leq|\overline{\left\|\Sigma_j'\right\|}-\widehat{\left\|\Sigma_j'\right\|}|+|\widehat{\left\|\Sigma_j'\right\|}-\left\|\Sigma_j'\right\||<\epsilon+\epsilon_1\leq\delta_\mu/2\sqrt{\eta}$ (where here ϵ is the error of the amplitude estimation step used in theorem 5.11 and ϵ_1 is the error in calling lemma 11.3. Again, we choose $\epsilon=\epsilon_1\leq\frac{\delta_\mu/\sqrt{\eta}}{4}$.

Since the tomography is more costly than the amplitude estimation step, we can disregard the runtime for the norm estimation step. As this operation is repeated k times for the k different covariance matrices, the total runtime of the whole algorithm is given by $\widetilde{O}\Big(\frac{kd^2\eta\kappa(V)(\mu(V)+\eta^2k^{3.5}\kappa(\Sigma)\mu(\Sigma))}{\delta_\mu^3}\Big)$. Let us also recall that for each of new computed covariance matrices, we use lemma @ref(lemma: absolute-error-logdet) to compute an estimate for their log-determinant and this time can be absorbed in the time T_Σ. $\square$

11.3.2.3 Quantum estimation of log-likelihood

Now we are going to show how it is possible to get an estimate of the log-likelihood using a quantum procedure. A good estimate of the log-likelihood is crucial, as it is used as stopping criteria for the quantum algorithm. Recall that the log-likelihood is defined as:

$$\ell(\gamma;V) = \sum_{i=1}^{n} \log \sum_{j\in[k]} \theta_j \phi(v_i;\mu_j,\Sigma_j) = \sum_{i=1}^{n} \log p(v_i;\gamma)$$

Classically, we stop to iterate the EM algorithm when $|\ell(\gamma^t;V) - \ell(\gamma^{t+1};V)| < n\epsilon$, or equivalently, we can set a tolerance on the average increase of the log of the probability: $|\mathbb{E}[\log p(v_i;\gamma^t)] - \mathbb{E}[\log p(v_i;\gamma^{t+1})]| < \epsilon$. In the quantum algorithm it is more practical to estimate $\mathbb{E}[p(v_i;\gamma^t)] = \frac{1}{n}\sum_{i=1}^{n} p(v_i;\gamma)$. From this we can estimate an upper bound on the log-likelihood (with the help of the the Jensen inequality) as:

$$n \log \mathbb{E}[p(v_i)] = \sum_{i=1}^{n} \log \mathbb{E}[p(v_i)] \geq \sum_{i=1}^{n} \log p(v_i) = \ell(\gamma;V)$$

.

Lemma 11.8 (Quantum estimation of likelihood)**.** *We assume we have quantum access to a GMM with parameters γ^t. For $\epsilon_\tau > 0$, there exists a quantum algorithm that estimates $\mathbb{E}[p(v_i;\gamma^t)]$ with absolute error ϵ_τ in time*

$$T_\ell = \widetilde{O}\left(k^{1.5}\eta^{1.5}\frac{\kappa(\Sigma)\mu(\Sigma)}{\epsilon_\tau^2}\right)$$

Proof. We obtain the likelihood from the ability to compute the value of a Gaussian distribution and quantum arithmetic. Using the mapping of lemma 11.1 with precision ϵ_1, we can compute $\phi(v_i|\mu_j,\Sigma_j)$ for all the Gaussians. We can build the state $|i\rangle \bigotimes_{j=0}^{k-1} |j\rangle|\overline{p(v_i|j;\gamma_j)}\rangle$. Then, by knowing θ, and by using quantum arithmetic we can compute in a register the probability of a point belonging to the mixture of Gaussian's: $p(v_i;\gamma) = \sum_{j\in[k]} \theta_j p(v_i|j;\gamma)$ (note that this operation require undoing the previous steps). for simplicity, we now drop the notation for the model γ and write $p(v_i)$ instead of $p(v_i;\gamma)$. Doing the previous calculations in a quantum computer, leads to the creation of the state $|i\rangle|p(v_i)\rangle$. To get an estimate of $\mathbb{E}[p(v_i)]$, we perform the mapping $|i\rangle|p(v_i)\rangle \mapsto |i\rangle\left(\sqrt{p(v_i)}|0\rangle + \sqrt{1-p(v_i)}|1\rangle\right)$ and estimate $p(|0\rangle) \simeq \mathbb{E}[p(v_i)]$ with amplitude estimation on the ancilla qubit being zero.

To get a ϵ_τ-estimate of $p(0)$ we need to decide the precision parameter we use for estimating $\overline{p(v_i|j;\gamma)}$ and the precision required by amplitude estimation. Let $\overline{p(0)}$ be the ϵ_1-error introduced by using lemma 11.1 and $\widehat{p(0)}$ the error introduced by amplitude estimation. Using triangle inequality we set $\left\|p(0) - \widehat{\overline{p(0)}}\right\| < \left\|\widehat{\overline{p(0)}} - \overline{p(0)}\right\| + \left\|\overline{p(0)} - p(0)\right\| < \epsilon_\tau$.

To have $|p(0) - \widehat{p(0)}| < \epsilon_\tau$, we should set ϵ_1 such that $|\overline{p(0)} - p(0)| < \epsilon_\tau/4$, and we set the error in amplitude estimation and in the estimation of the probabilities to be $\epsilon_\tau/2$. The runtime of this procedure is therefore:

$$\widetilde{O}\left(k \cdot T_{G,\epsilon_\tau} \cdot \frac{1}{\epsilon_\tau \sqrt{p(0)}}\right) = \widetilde{O}\left(k^{1.5}\eta^{1.5} \cdot \frac{\kappa(\Sigma)\mu(\Sigma)}{\epsilon_\tau^2}\right)$$

□

11.3.2.4 Analysis of Quantum Expectation-Maximizaiton

Theorem 11.3 (Quantum Expectation-Maximization for Gaussian mixture models)**.** *We assume we have quantum access to a GMM with parameters γ^t. For parameters $\delta_\theta, \delta_\mu, \epsilon_\tau > 0$, the running time of one iteration of the Quantum Expectation-Maximization (QEM) algorithm is*

$$O(T_\theta + T_\mu + T_\Sigma + T_\ell),$$

for

- $T_\theta = \widetilde{O}\left(k^{3.5}\eta^{1.5}\frac{\kappa^2(\Sigma)\mu(\Sigma)}{\delta_\theta^2}\right)$
- $T_\mu = \widetilde{O}\left(\frac{kd\eta\kappa(V)(\mu(V)+k^{3.5}\eta^{1.5}\kappa^2(\Sigma)\mu(\Sigma))}{\delta_\mu^3}\right)$
- $T_\Sigma = \widetilde{O}\left(\frac{kd^2\eta\kappa^2(V)(\mu(V')+\eta^2k^{3.5}\kappa^2(\Sigma)\mu(\Sigma))}{\delta_\mu^3}\right)$
- $T_\ell = \widetilde{O}\left(k^{1.5}\eta^{1.5}\frac{\kappa^2(\Sigma)\mu(\Sigma)}{\epsilon_\tau^2}\right)$

For the range of parameters of interest, the running time is dominated by T_Σ.

The proof follows directly from the previous lemmas. Note that the cost of the whole algorithm is given by repeating the Estimation and the Maximization steps several times, until the threshold on the log-likelihood is reached. Note also that the expression of the runtime can be simplified from the observation that the cost of performing tomography on the covariance matrices Σ_j dominates the cost.

Chapter 12

QML on real datasets

In this chapter, we discuss how to have a better idea on the performance of our algorithms on real dataset. This is very important, as the runtime of the algorithms we discuss here depends on the characteristics of the dataset, which might scale badly by increasing the size of the dataset. It's our duty to see if this is the case. In the worst case, this might be due for each single dataset that we analyze. It this section we discuss two main approaches. We could either start by proving some properties of a dataset that we know is well-suitable to be analyzed by a certain algorithm. The idea is the following: if (quantum or classical) data analysis works well for a certain dataset, it might be because the algorithms that exploits certain structure in the dataset possess. We can exploit this intuition to prove some properties of the dataset that are expected to be analyzed by that algorithm. For example, this is the case for section 12.1.1, where we exploit the "cloud" structure of dataset that we often cluster using k-means (or q-means).

Secondly, we might just verify experimentally that the scaling of these parameters is favourable (they are small, they don't grow by increasing the number of features and the number of samples, and so on..), and that the error introduced by the quantum procedures won't perturb the quality of the data anaysis (in fact we will see that some noise might help regularizing the model, improving the classification accuracy).

12.1 Theoretical considerations

12.1.1 Modelling well-clusterable datasets

In this section, we define a model for the dataset in order to provide a tight analysis on the running time of our clustering algorithm. Thanks to this assumption, we can provide tighter bounds for its running time. Recall that for q-means we consider that the dataset V is normalized so that for all $i \in [N]$, we have $1 \leq \|v_i\|$, and we define the parameter $\eta = \max_i \|v_i\|^2$. We will also assume that the number k is the "right" number of clusters, meaning that we assume each cluster has at least some $\Omega(N/k)$ data points.

We now introduce the notion of a *well-clusterable* dataset. The definition aims to capture some properties that we can expect from datasets that can be clustered efficiently using a k-means algorithm. This notion of a well-clusterable dataset shares some similarity with the assumptions made in (Drineas et al., 2002), but there are also some differences specific to the clustering problem.

Definition 12.1 (Well-clusterable dataset)**.** A data matrix $V \in \mathbb{R}^{n \times d}$ with rows $v_i \in \mathbb{R}^d, i \in [n]$ is said to be well-clusterable if there exist constants $\xi, \beta > 0$, $\lambda \in [0, 1]$, $\eta \leq 1$, and cluster centroids c_i for $i \in [k]$ such that:

- (separation of cluster centroids): \$d(c_i, c_j)   i,j [k] \$
- (proximity to cluster centroid): At least λn points v_i in the dataset satisfy $d(v_i, c_{l(v_i)}) \leq \beta$ where $c_{l(v_i)}$ is the centroid nearest to v_i.
- (Intra-cluster smaller than inter-cluster square distances): The following inequality is satisfied
$$4\sqrt{\eta}\sqrt{\lambda\beta^2 + (1-\lambda)4\eta} \leq \xi^2 - 2\sqrt{\eta}\beta.$$

Intuitively, the assumptions guarantee that most of the data can be easily assigned to one of k clusters, since these points are close to the centroids, and the centroids are sufficiently far from each other. The exact inequality comes from the error analysis, but in spirit it says that ξ^2 should be bigger than a quantity that depends on β and the maximum norm η.

We now show that a well-clusterable dataset has a good rank-k approximation where k is the number of clusters. This result will later be used for giving tight upper bounds on the running time of the quantum algorithm for well-clusterable datasets. As we said, one can easily construct such datasets by picking k well separated vectors to serve as cluster centers and then each point in the cluster is sampled from a Gaussian distribution with small variance centered on the centroid of the cluster.

Lemma 12.1. *Let V_k be the optimal k-rank approximation for a well-clusterable data matrix V, then $\|V - V_k\|_F^2 \leq (\lambda\beta^2 + (1-\lambda)4\eta)\|V\|_F^2$.*

Proof. Let $W \in \mathbb{R}^{n \times d}$ be the matrix with row $w_i = c_{l(v_i)}$, where $c_{l(v_i)}$ is the centroid closest to v_i. The matrix W has rank at most k as it has exactly k distinct rows. As V_k is the optimal rank-k approximation to V, we have $\|V - V_k\|_F^2 \leq \|V - W\|_F^2$. It therefore suffices to upper bound $\|V - W\|_F^2$. Using the fact that V is well-clusterable, we have

$$\|V - W\|_F^2 = \sum_{ij}(v_{ij} - w_{ij})^2 = \sum_i d(v_i, c_{l(v_i)})^2 \leq \lambda n\beta^2 + (1-\lambda)n4\eta,$$

where we used Definition 12.1 to say that for a λn fraction of the points $d(v_i, c_{l(v_i)})^2 \leq \beta^2$ and for the remaining points $d(v_i, c_{l(v_i)})^2 \leq 4\eta$. Also, as all v_i have norm at least 1 we have $n \leq \|V\|_F$, implying that $\|V - V_k\|^2 \leq \|V - W\|_F^2 \leq (\lambda\beta^2 + (1-\lambda)4\eta)\|V\|_F^2$. □

The running time of the quantum linear algebra routines for the data matrix V in Theorem 5.11 depend on the parameters $\mu(V)$ and $\kappa(V)$. We establish bounds on both of these parameters using the fact that V is well-clusterable

Lemma 12.2. *Let V be a well-clusterable data matrix, then $\mu(V) := \frac{\|V\|_F}{\|V\|} = O(\sqrt{k})$.*

Proof. We show that when we rescale V so that $\|V\| = 1$, then we have $\|V\|_F = O(\sqrt{k})$ for the rescaled matrix. From the triangle inequality we have that $\|V\|_F \leq \|V - V_k\|_F + \|V_k\|_F$. Using the fact that $\|V_k\|_F^2 = \sum_{i \in [k]} \sigma_i^2 \leq k$ and lemma 12.1, we have,

$$\|V\|_F \leq \sqrt{(\lambda\beta^2 + (1-\lambda)4\eta)}\,\|V\|_F + \sqrt{k}$$

Rearranging, we have that $\|V\|_F \leq \frac{\sqrt{k}}{1 - \sqrt{(\lambda\beta^2 + (1-\lambda)4\eta)}} = O(\sqrt{k})$. □

We next show that if we use a condition threshold $\kappa_\tau(V)$ instead of the true condition number $\kappa(V)$, that is we consider the matrix $V_{\geq\tau} = \sum_{\sigma_i \geq \tau} \sigma_i u_i v_i^T$ by discarding the smaller singular values $\sigma_i < \tau$, the resulting matrix remains close to the original one, i.e. we have that $\|V - V_{\geq\tau}\|_F$ is bounded.

Lemma 12.3. *Let V be a matrix with a rank-k approximation given by $\|V - V_k\|_F \leq \epsilon' \|V\|_F$ and let $\tau = \frac{\epsilon_\tau}{\sqrt{k}}\|V\|_F$, then $\|V - V_{\geq\tau}\|_F \leq (\epsilon' + \epsilon_\tau)\|V\|_F$.*

Proof. Let l be the smallest index such that $\sigma_l \geq \tau$, so that we have $\|V - V_{\geq\tau}\|_F = \|V - V_l\|_F$. We split the argument into two cases depending on whether l is smaller or greater than k.

- If $l \geq k$ then $\|V - V_l\|_F \leq \|V - V_k\|_F \leq \epsilon' \|V\|_F$.
- If $l < k$ then, $\|V - V_l\|_F \leq \|V - V_k\|_F + \|V_k - V_l\|_F \leq \epsilon' \|V\|_F + \sqrt{\sum_{i=l+1}^{k} \sigma_i^2}$. As each $\sigma_i < \tau$ and the sum is over at most k indices, we have the upper bound $(\epsilon' + \epsilon_\tau)\|V\|_F$.

□

The reason the notion of well-clusterable dataset was defined, was to be able to provide some strong guarantees for the clustering of most points in the dataset. Note that the clustering problem in the worst case is NP-hard and we only expect to have good results for datasets that have some good property. Intuitively, we should only expect k-means to work when the dataset can actually be clusterd in k clusters. We show next that for a well-clusterable dataset V, there is a constant δ that can be computed in terms of the parameters in Definition 12.1 such that the δ-k-means clusters correctly most of the data points.

Lemma 12.4. *Let V be a well-clusterable data matrix. Then, for at least λn data points v_i, we have*

$$\min_{j \neq \ell(i)} (d^2(v_i, c_j) - d^2(v_i, c_{\ell(i)})) \geq \xi^2 - 2\sqrt{\eta}\beta$$

which implies that a δ-k-means algorithm with any $\delta < \xi^2 - 2\sqrt{\eta}\beta$ will cluster these points correctly.

Proof. By Definition 12.1, we know that for a well-clusterable dataset V, we have that $d(v_i, c_{l(v_i)}) \leq \beta$ for at least λn data points and where $c_{l(v_i)}$ is the centroid closest to v_i. Further, the distance between each pair of the k centroids satisfies the bounds $2\sqrt{\eta} \geq d(c_i, c_j) \geq \xi$. By the triangle inequality, we have $d(v_i, c_j) \geq d(c_j, c_{\ell(i)}) - d(v_i, c_{\ell(i)})$. Squaring both sides of the inequality and rearranging,

$$d^2(v_i, c_j) - d^2(v_i, c_{\ell(i)}) \geq d^2(c_j, c_{\ell(i)}) - 2d(c_j, c_{\ell(i)})d(v_i, c_{\ell(i)}))$$

Substituting the bounds on the distances implied by the well-clusterability assumption, we obtain $d^2(v_i, c_j) - d^2(v_i, c_{\ell(i)}) \geq \xi^2 - 2\sqrt{\eta}\beta$. This implies that as long as we pick $\delta < \xi^2 - 2\sqrt{\eta}\beta$, these points are assigned to the correct cluster, since all other centroids are more than δ further away than the correct centroid. □

12.2 Experiments

The experiments bypassed the construction of the quantum circuit and directly performed the noisy linear algebraic operations carried out by the quantum algorithm. The simulations are carried out on datasets that are considered the standard benchmark of new machine learning algorithms, inserting the same kind of errors that we expect to have in the real execution on the quantum hardware. In the experiments, we aim to study the robustness of data analysis to the noise introduced by the quantum algorithm, study the scaling of the runtime algorithm on real data and thus understand which datasets can be analyzed efficiently by quantum computers. The experiments are aimed at finding if the impact of noise in the quantum algorithms decreases significantly the accuracy in the data analysis, and if the impact of the error parameters in the runtime does prevent quantum speedups for large datasets.

12.2.1 Datasets

12.2.1.1 MNIST

MNIST (LeCun, 1998) is probably the most used dataset in image classification. It is a collection of 60000 training plus 10000 testing images of $28 \times 28 = 784$ pixels. Each image is a black and white handwritten digit between 0 and 9 and it is paired with a label that specifies the digit. Since the images are black and white, they are represented as arrays of 784 values that encode the lightness of each pixel. The dataset, excluding the labels, can be encoded in a matrix of size 70000×784.

12.2.1.2 Fashon-MNIST

Fashion MNIST (Xiao et al., 2017) is a recent dataset for benchmarking in image classification. Like the MNIST, it is a collection of 70000 images composed of $28 \times 28 = 784$ pixels. Each image represents a black and white fashion item among {T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot}. Each image is paired with a label that specifies the item represented in the image. Since the images are black and white, they are represented as arrays of 784 values that encode the lightness of each pixel. The dataset, excluding the labels, can be encoded in a matrix of size 70000×784.

12.2.1.3 CIFAR-10

CIFAR-10 (Krizhevsky et al., 2009) is another widely used dataset for benchmarking image classification. It contains 60000 colored images of 32×32 pixel, with the values for each of the 3 RGB colors. Each image represents an object among {airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck} and is paired with the appropriate label. When the images are reshaped to unroll the three channels in a single vector, the resulting size of the dataset is 60000×3072.

12.2.1.4 Research Paper

Research Paper (Harun-Ur-Rashid, 2018) is a dataset for text classification, available on Kaggle. It contains 2507 titles of papers together with the labels of the venue where they have been published. The labels are {WWW, INFOCOM, ISCAS, SIGGRAPH, VLDB}. We pre-process the titles to compute a contingency table of $papers \times words$: the value of the $i^{th} - j^{th}$ cell is the number of times that the j^{th} word is contained in the i^{th} title. We remove the English stop-words, the words that appear in only one document, and the words that appear in more than half the documents. The result is a contingency table of size 2507×2010.

12.2.1.5 VoxForge Dataset

12.2.2 q-means

12.2.2.1 MNIST pre-processing

From this raw data we first performed some dimensionality reduction processing, then we normalized the data such that the minimum norm is one. Note that, if we were doing q-means with a quantum computer, we could use efficient quantum procedures equivalent to Linear Discriminant Analysis, such as (Kerenidis and Luongo, 2020), or other quantum dimensionality reduction algorithms like (Lloyd et al., 2014) (Cong and Duan, 2015).

As preprocessing of the data, we first performed a Principal Component Analysis (PCA), retaining data projected in a subspace of dimension 40. After normalization, the value of η was 8.25 (maximum norm of 2.87), and the condition number was 4.53. Figure **??** represents the evolution of the accuracy during the k-means and δ-k-means for 4 different values of δ. In this numerical experiment, we can see that for values of the parameter η/δ of order 20, both k-means and δ-k-means reached a similar, yet low accuracy in the classification in the same number of steps. It is important to notice that the MNIST dataset, without other preprocessing than dimensionality reduction, is known not to be well-clusterable under the k-means algorithm.

Table 12.1: A sample of results collected from the same experiments as in Figure **??**. Different metrics are presented for the train set and the test set. ACC: accuracy. HOM: homogeneity. COMP: completeness. V-M: v-measure. AMI: adjusted mutual information. ARI: adjusted rand index. RMSEC: Root Mean Square Error of Centroids.

Algo	Dataset	ACC	HOM	COMP	V-M	AMI	ARI	RMSEC
k-means	Train	0.582	0.488	0.523	0.505	0.389	0.488	0
	Test	0.592	0.500	0.535	0.517	0.404	0.499	-
δ-k-means, $\delta = 0.2$	Train	0.580	0.488	0.523	0.505	0.387	0.488	0.009
	Test	0.591	0.499	0.535	0.516	0.404	0.498	-
δ-k-means, $\delta = 0.3$	Train	0.577	0.481	0.517	0.498	0.379	0.481	0.019
	Test	0.589	0.494	0.530	0.511	0.396	0.493	-
δ-k-means, $\delta = 0.4$	Train	0.573	0.464	0.526	0.493	0.377	0.464	0.020
	Test	0.585	0.492	0.527	0.509	0.394	0.491	-
δ-k-means, $\delta = 0.5$	Train	0.573	0.459	0.522	0.488	0.371	0.459	0.034
	Test	0.584	0.487	0.523	0.505	0.389	0.487	-

12.2.3 QSFA

12.2.4 QEM

In this section, we discuss again Quantum Expectation-Maximization algorithm, and we present the results of some experiments on real datasets to estimate its runtime. We will also show some bound on the value of the parameters that governs it, like $\kappa(\Sigma)$, $\kappa(V)$, $\mu(\Sigma)$, $\mu(V)$, δ_θ, and δ_μ, and we give heuristic for dealing with the condition number. As we already have done, we can put a threshold on the condition number of the matrices Σ_j, by discarding singular values which are smaller than a certain threshold. This might decrease the runtime of the algorithm without impacting its performances. This is indeed done often in classical machine learning models, since discarding the eigenvalues smaller than a certain threshold might even improve upon the metric under consideration (i.e. often the accuracy), by acting as a form of regularization (look at Section 6.5 of (Murphy, 2012)). This in practice is equivalent to limiting the eccentricity of the Gaussians. We can do similar considerations for putting a threshold on the condition number of the dataset $\kappa(V)$. Recall that the value of the condition number of the matrix V is approximately $1/\min(\{\theta_1, \cdots, \theta_k\} \cup \{d_{st}(\mathcal{N}(\mu_i, \Sigma_i), \mathcal{N}(\mu_j, \Sigma_j)) | i \neq j \in [k]\})$, where d_{st} is the statistical distance between two Gaussian distributions (Kalai et al., 2012). We have some choice in picking the definition for μ: in previous experiments it has been found that choosing the maximum ℓ_1 norm of the rows of V lead to values of $\mu(V)$ around 10 for the MNIST dataset, as we saw on the experiments of QSFA, PCA and q-means. Because of the way μ is defined, its value will not increase significantly as we add vectors to the training set. In case the matrix V can be clustered with high-enough accuracy by distance-based algorithms like k-means, it has been showed that the Frobenius norm of the matrix is proportional to $\sqrt{k}$, that is, the rank of the matrix depends on the number of different classes contained in the data. Given that EM is just a more powerful extension of k-means, we can rely on similar observations too. Usually, the number of features d is much more than the number of components in the mixture, i.e. $d \gg k$, so we expect d^2 to dominate the $k^{3.5}$ term in the cost needed to estimate the mixing weights, thus making T_Σ the leading term in the runtime. We expect this cost to be be mitigated by using ℓ_∞ form of tomography but we defer further experiment for future research.

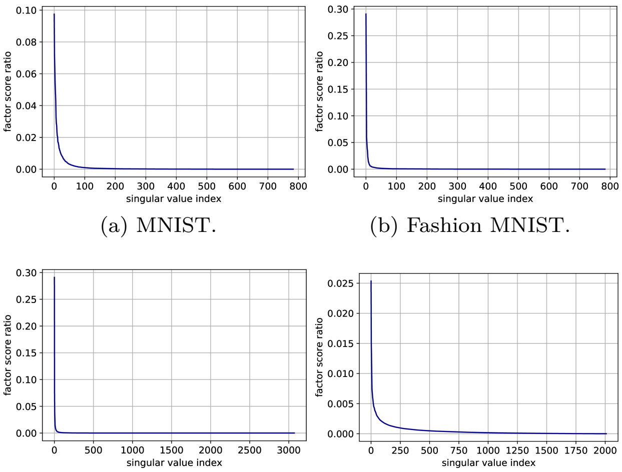
As we said, the quantum running time saves the factor that depends on the number of samples and introduces a number of other parameters. Using our experimental results we can see that when the number of samples is large enough one can expect the quantum running time to be faster than the classical one.

To estimate the runtime of the algorithm, we need to gauge the value of the parameters δ_μ and δ_θ, such that they are small enough so that the likelihood is perturbed less than ϵ_τ, but big enough to have a fast algorithm. We have reasons to believe that on well-clusterable data, the value of these parameters will be large enough, such as not to impact dramatically the runtime. A quantum version of k-means algorithm has already been simulated on the MNIST dataset under similar assumptions (Kerenidis et al., 2019a). The experiment concluded that, for datasets that are expected to be clustered nicely by this kind of clustering algorithms, the value of the parameters δ_μ did not decrease by increasing the number of samples nor the number of features. There, the value of δ_μ (which in their case was called just δ) has been kept between 0.2 and 0.5, while retaining a classification accuracy comparable to the classical k-means algorithm. We expect similar behaviour in the GMM case, namely that for large datasets the impact on the runtime of the errors $(\delta_\mu, \delta_\theta)$ does not cancel out the exponential gain in the dependence on the number of samples, and we discuss more about this in the next paragraph. The value of ϵ_τ is usually (for instance in scikit-learn (Pedregosa et al., 2011)) chosen to be 10^{-3}. We will see that the value of η has always been 10 on average, with a maximum of 105 in the experiments.

12.2.5 QPCA

These experiment are extracted from (Bellante and Zanero, 2022). This section shows several experiments on the MNIST, Fashion MNIST, CIFAR-10 and Research Papers datasets. In all the experiments, the datasets have been shifted to row mean 0 and normalized so that $\sigma_{max} \leq 1$.

12.2.5.0.1 Explained variance distribution First, we explore the distribution of the factor score ratios in the MNIST, Fashion MNIST, CIFAR-10 and Research Papers datasets, showing that a small number of singular values is indeed able to explain a great amount of the variance of the data. For all the four datasets, we compute the singular values and their factor score ratios and we plot them. The results of Figure 12.1 show a rapid decrease of the factor score ratios in all the datasets, confirming the expectations.



(a) MNIST.

(b) Fashion MNIST.

(c) CIFAR 10.

(d) Research Papers.

Figure 12.1: Explained variance distribution in datasets for machine learning. In order: MNIST, Fashion MNIST, CIFAR 10, Research Papers.

12.2.5.1 Image classification with quantum PCA

To provide the reader with a clearer view of the algorithms in Sections 7.2, 9.1.1 and their use in machine learning, we provide experiments on quantum PCA for image classification. We perform PCA on the three datasets for image classification (MNIST, Fashion MNIST and CIFAR 10) and classify them with a K-Nearest Neighbors. First, we simulate the extraction of the singular values and the percentage of variance explained by the principal components (top k factor score ratios' sum) using the procedure from Theorem 7.6. Then, we study the error of the model extraction, using Lemma 9.1, by introducing errors on the Frobenius norm of the representation to see how this affects the accuracy.

12.2.5.1.1 Estimating the number of principal components We simulate Theorem 7.6 to decide the number of principal components needed to retain 0.85 of the total variance. For each dataset, we classically compute the singular values with an exact classical algorithm and simulate the quantum state $\frac{1}{\sqrt{\sum_j^r \sigma_j^2}} \sum_i^r \sigma_i |\sigma_i\rangle$ to emulate the measurement process. After initializing the random object with the correct probabilities, we measure it $\frac{1}{\gamma^2} = 1000$ times and estimate the factor score ratios with a frequentist approach (i.e., dividing the number of measurements of each outcome by the total number of measurements). Measuring 1000 times guarantees us an error of at most $\gamma = 0.03$ on each factor score ratios. To determine the number of principal components to retain, we sum the factor score ratios until the percentage of explained variance becomes greater than 0.85. We report the results of this experiments in Table 12.2. We obtain good results for all the datasets, estimating no more than 3 extra principal components than needed.

Table 12.2: Results of the estimation of the number of principal components to retain. The parameter k is the number of components needed to retain at least $p = 0.85$ of the total variance. p is computed w.r.t. the estimated k.

Parameter	MNIST	F-MNIST	CIFAR-10
Estimated k	62	45	55
Exact k	59	43	55
Estimated p	0.8510	0.8510	0.8510
Exact p	0.8580	0.8543	0.8514
γ	0.0316	0.0316	0.0316

The number of principal components can be further refined using Theorem 7.7. When we increase the percentage of variance to retain, the factor score ratios become smaller and the estimation worsens. When the factor score ratios become too small to perform efficient sampling, it is possible to establish the threshold θ for the smaller singular value to retain using Theorems 7.7 and **??**. If one is interested in refining the exact number k of principal components, rather than θ, it is possible to obtain it using a combination of the algorithms from Theorems 7.7, **??** and the quantum counting algorithm (Brassard et al., 2002) in time that scales with the square root of k. Once that the number of principal components has been set, the next step is to use Theorem 7.8 to extract the top singular vectors. To do so, we can retrieve the threshold θ from the previous step by checking the gap between the last singular value to retain and the first to exclude.

12.2.5.1.1.1 Studying the error in the data representation We continue the experiment by checking how much error in the data representation a classifier can tolerate. We compute the exact PCA's representation for the three datasets and a perform 10-fold Cross-validation error using a k-Nearest Neighbors with 7 neighbors. For each dataset we introduce error in the representation and check how the accuracy decreases. To simulate the error, we perturb the exact representation by adding truncated Gaussian error (zero mean and unit variance, truncated on the interval $[\frac{-\xi}{\sqrt{nm}}, \frac{\xi}{\sqrt{nm}}]$) to each component of the matrix. The graph in Figure @ref{fig:error-matrix} shows the distribution of the effective error on 2000 approximation of a matrix A, such that $\|A - \overline{A}\| \leq 0.1$. The distribution is still Gaussian, centered almost at the half of the bound.

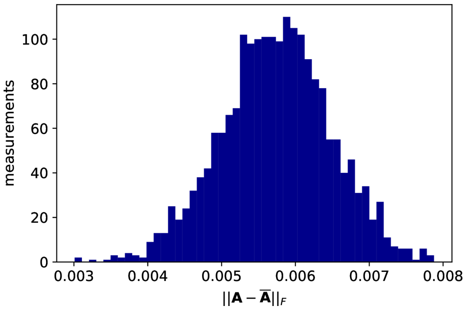
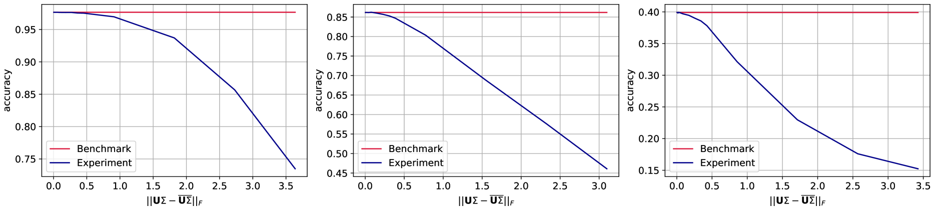


Figure 12.2: Introducing some error in the Frobenius norm of a matrix A. The error was introduced such that $\|A - \overline{A}\| \leq 0.01$. The figure shows the distribution of the error over 2000 measurements.

The results show a reasonable tolerance of the errors, we report them in two set of figures. Figure 12.3 shows the drop of accuracy in classification as the error bound increases. Figure 12.4 shows the trend of the accuracy against the effective error of the approximation.



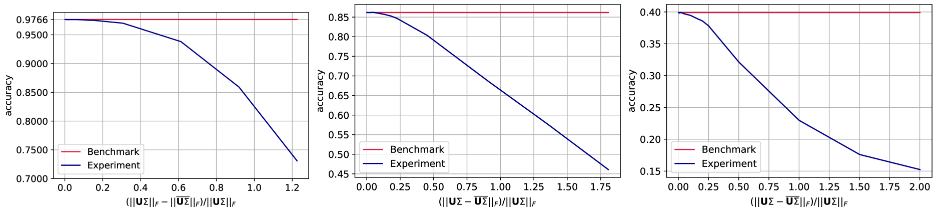
(a) MNIST.

(b) Fashion MNIST.

(c) CIFAR 10.

Figure 12.3: Classification accuracy of 7-Nearest Neighbor on three machine learning datasets after PCA's dimensionality reduction. The drop in accuracy is plotted with respect to the *bound* on the Frobenius norm of the difference between the exact data representation and its approximation. In order: MNIST, Fashion MNIST, CIFAR 10.

12.2.5.1.2 Analyzing the run-time parameters As discussed in Section 9.1.1, the model extraction's run-time is $\widetilde{O}\left(\left(\frac{1}{\gamma^2} + \frac{kz}{\theta\sqrt{p}\delta^2}\right)\frac{\mu(A)}{\epsilon}\right)$, where $A \in$



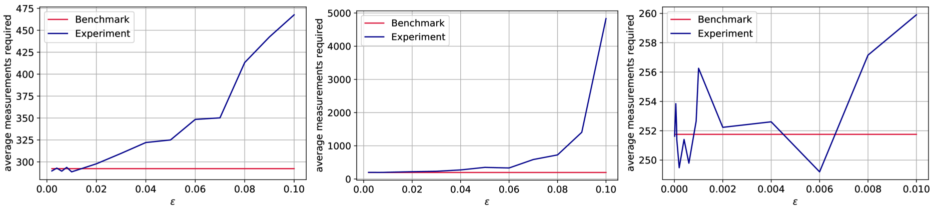
(a) MNIST.

(b) Fashion MNIST.

(c) CIFAR 10.

Figure 12.4: Classification accuracy of 7-Nearest Neighbor on three machine learning datasets after PCA's dimensionality reduction. The drop in accuracy is plotted with respect to the *effective* Frobenius norm of the difference between the exact data representation and its approximation. In order: MNIST, Fashion MNIST, CIFAR 10.

$\mathbb{R}^{n\times m}$ is PCA's input matrix, $\mu(A)$ is a parameter bounded by $\min(\|A\|_F, \|A\|_\infty)$, k is the number of principal components retained, θ is the value of the last singular value retained, γ is the precision to estimate the factor score ratios, ϵ bounds the absolute error on the estimation of the singular values, δ bounds the ℓ_2 norm of the distance between the singular vectors and their approximation, and z is either n, m depending on whether we extract the left singular vectors, to compute the classical representation, or the right ones, to retrieve the model and allow for further quantum/classical computation. This run-time can be further lowered using Theorem **??** if we are not interested in the factor score ratios. The aim of this paragraph is to show how to determine the run-time parameters for a specific dataset. We enrich the parameters of Table 12.2 with the ones in Table 12.3 and we discuss how to compute them. From the previous paragraphs it should be clear how to determine k, θ, γ and p, and it is worth noticing again that $1/\sqrt{p} \simeq 1$. To bound $\mu(A)$ we have computed $\|A\|_F$ and $\|A\|_\infty$. To compute the parameter ϵ we have considered two situations: we need to allow for a correct ordering of the singular values; we need to allow for a correct thresholding. We refer to the first as the ordering ϵ and to the second as the thresholding ϵ. To compute the first one it is sufficient to check the smallest gap between the first k singular values of the dataset. For the last one, one should check the difference between the last retained singular value and the first that is excluded. It follows that the thresholding ϵ is always smaller than the ordering ϵ. For sake of completeness, we have run experiments to check how the Coupon Collector's problem changes as ϵ increases. Recall that in the proof of Theorem @ref(thm:top-k_sv_extraction) we use that $\frac{1}{\sqrt{\sum_i^k \frac{\sigma_i^2}{\overline{\sigma}_i^2}}}\sum_i^k \frac{\sigma_i}{\overline{\sigma}_i}|u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle \sim \frac{1}{\sqrt{k}}\sum_i^k |u_i\rangle|v_i\rangle|\overline{\sigma}_i\rangle$ to say that the number of measurements needed to observe all the singular values is $O(k\log(k))$, and this is true only if ϵ is small enough to let the singular values distribute uniformly. We observe that the thresholding ϵ always satisfy the Coupon Collector's scaling, and we have plotted the results of our tests in Figure 12.5.



(a) MNIST. (b) Fashion MNIST. (c) CIFAR 10.

Figure 12.5: Number of measurements needed to obtain all the k singular values from the quantum state $\frac{1}{\sqrt{\sum_i^r \sigma_i^2}}\sum_i^k \frac{\sigma_i}{\overline{\sigma}_i}|\overline{\sigma}_i\rangle$, where $\|\sigma_i - \overline{\sigma}_i\| \leq \epsilon$, as ϵ increases. The benchmark line is $k\log_{2.4}(k)$. In order: MNIST, Fashion MNIST, CIFAR 10.

Furthermore, we have computed δ by using the fact that $\|A - \overline{A}\| \leq \sqrt{k}(\epsilon + \delta)$ (Lemma 9.1). By inverting the equation and considering the thresholding ϵ we have computed an estimate for δ. In particular, we have fixed $\|A - \overline{A}\|$ to the biggest value in our experiments so that the accuracy doesn't drop more than 1%.

Since we have considered the values of the effective errors instead of the bounds, our estimates are pessimistic.

Table 12.3: Run-time parameters.

Parameter	MNIST	F-MNIST	CIFAR-10
Ordr. ϵ	0.0003	0.0003	0.0002
Thrs. ϵ	0.0030	0.0009	0.0006
$\|A\|_F$	3.2032	1.8551	1.8540
$\|A\|_\infty$	0.3730	0.3207	0.8710
θ	0.1564	0.0776	0.0746
δ	0.1124	0.0106	0.0340

These results show that Theorem 7.6, 7.7 and **??** can already provide speed-ups on datasets as small as the MNIST. Even though their speed-up is not exponential, they still run sub-linearly on the number of elements of the matrix even though all the elements are taken into account during the computation, offering a polynomial speed-up with respect to their traditional classical counterparts. On the other hand, Theorem 7.8 requires bigger datasets. On big low-rank datasets that maintain a good distribution of singular values, these algorithms are expected to show their full speed-up. As a final remark, note that the parameters have similar orders of magnitude.

12.2.5.1.3 Projecting the quantum data in the new quantum feature space To end with, we have tested the value of α (Definition 9.1, Lemma 9.3) for the MNIST dataset, fixing $\varepsilon = 0$ and trying $p \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. We have observed that $\alpha = 0.96 \pm 0.02$, confirming that the run-time of Corollary 9.1 can be assumed constant for the majority of the data points of a PCA-representable dataset.

Chapter 13

Quantum algorithms for graph problems

In this chapter we are going to discuss quantum algorithms for graph theoretical problems. Initial focus of this chapter revolves around the work of (Dürr et al., 2006), which investigated the query complexity of problems like MINIMUM SPANNING TREE, CONNECTIVITY, STRONG CONNECTIVITY, and SINGLE SOURCE SHORTEST PATH. Classically, these problems can be solved efficiently i.e. in polynomial number of queries to the graph. In this chapter, we will see how to decrease the query complexity of the quantum algorithm further, by applying in a shrewd way amplitude amplification and related algorithms. More specifically, in the work of (Dürr et al., 2006), they used three different versions of the Grover's algorithm. In particular, we state a slightly improved version here, with a quadratic improvement also in the runtime dependence on the probability of failure.

Theorem 13.1 (Grover's search algorithm, version of (Buhrman et al., 1999))**.** *Let $N = 2^n$ for $n > 0$. Given quantum oracle access $O_x : |i\rangle \mapsto |i\rangle|x_i\rangle$ to a vector $x \in [k]^N$ (for a fixed k) and access to an oracle $O_f|x\rangle = (-1)^{f(x)}|x\rangle$ for a function $f : [k] \mapsto \{0,1\}$, If m is the number of elements of the vector x that are evaluated to 1 (called marked elements), there is a quantum algorithm that succeed with probability greater than $1-\delta$ and finds an index of a marked element using O_x only $O(\sqrt{N/m \log(1/\delta)})$ times.*

In fact, note that the "standard" version of bounding the probability of failure of a quantum or classical randomized algorithm consist in repeating the algorithm a certain number of time, and use the "powering lemma" C.1. This will result in an increase of the runtime that is logarithmic in δ. This version of Grover's algorithm achieves a quadratic speedup in the failure probability.

Further discussions of some of the quantum algorithms for graphs can also be found in (Dürr et al., 2006) and also in (Dörn, 2008).

As usual, we assume two different access to the graph: the "adjacecny matrix model" and the "adjacency array" (or list) model.

- In the adjacecy matrix model we assume to have query access to the entries of an adjacecny matrix of a graph, as in definition 3.12.
- In the adjacency list model we assume to have query access to the an oracle that tells us the number of adjacent nodes, and an oracle that gives us the index of the adjacent nodes, as in 3.13. Note that sometimes this goes under the name "adjacency array", as in the quantum case we don't have to go through the whole list of adjacent nodes, but we can index them as an array.

13.1 Connectivity

The problem of connectivity, as stated below can be seen as a special case of them minimum spanning tree problem, where all edges of G carries equal weight.

Definition 13.1 (GRAPH CONNECTIVITY problem)**.** Given an undirected graph $G = (V, E)$, decide if G is connected.

The following algorithm has first been proposed in (Dürr et al., 2006), and reelaborated in (Dörn, 2008).

Algorithm 1

Require: Quantum query access to A, the adjacecny matrix of an undirected, unweighted graph $G = (V, E)$.

Ensure: A MST T if the graph is connected, otherwise return "not connected".

```
Create a set $A = \emptyset$.
Create a spanning tree $T = (V, A)$
while $c(T) > 1$ do
    Find $e$ using quantum search algorithm using quantum access to $A$ ($U_A$),
and the oracle $U_{f_T}$
    $A := A \cup \{e\}$
end while
```

Figure 13.1: Algorithm for graph connectivity of undirected graph in the adjacecy model

Theorem 13.2 (Quantum algorithm for graph conectivity (adjacency matrix model))**.** *Assume that U_A is a unitary that gives you query access to the adjacency matrix M of an undirected graph $G = (V, E)$. Then, deciding if a graph is connected has an expected number of queries to U_A of $\widetilde{O}(n^{3/2})$. In particular,* *if the graph is connected, algorithm in figure 13.1 returns a spanning tree for G with probability greater than* 2/3.

Proof. The whole algorithm tries to build a spanning tree for the graph. If we succeed, then the graph is connected. The algorithm start by creating a data structure that holds n different connected components, one for each vertex. Then, we construct a spanning tree by finding an edge that connects any two of the connected components.

Initialize the algorithm with an empty edge set A for the spanning tree $T = (V, A)$. We use theorem 13.1 on the operator U_A and the oracle U_{f_T}. As usual, U_A is defined as $U_A : |i, j, c\rangle \mapsto |i, j, c \oplus A_{ij}\rangle$. We ecapsulate into the oracle U_{f_T} the data structure that stores the connected components, so we can have a unitary implementing the function $f_T : E \mapsto \{0, 1\}$:

$$f_T(e) \begin{cases} 1 & \text{if } c(T \cup e) < c(T) \\ 0 & \text{otherwise} \end{cases}$$

Where $c(G)$ is the number of connected components of the graph G. It is basically a unitary that checks if a given edge has endpoints of 2 different connected component. Note that U_{f_T} needs to compare a given edge with a whole list of edges that are currently in the list of connected components. Note that in order to work, this oracle should compare a given edge with the list of edges that are part of the spanning tree T. The spanning tree can grow up to size $O(n)$, so the depth of the oracle is at worst $O(n)$ (up to a polylogarithmic factors).

The runtime analysis is concluded by noting that we need to repeat the search procedure of theorem 13.1 up to n times (because when we obtain n nodes in the MST we stop the algorithm). Suppose that the graph is connected. The main loop of the algorithm is repeated exactly $n - 1$ times, and each search withint the loop can be done in $O(\sqrt{n^2/k})$, where k is the number of valid solutions to the search problem. These solutions correspond to the edges e of G that are linking any two connected components. It is simple to observe that at the first iteration we have at least $k = n - 1$ solutions (i.e. any edge is a good solution), and the number of solutions decreases at each iteration. The number of queries to the oracle is:

$$\sum_{k=2}^{n} \sqrt{\frac{n^2}{(k-1)}} = n \sum_{k=2}^{n} \frac{1}{\sqrt{k-1}}$$

With Cauchy-Schwartz we can see that:

$$\sum_{k=2}^{n} \frac{1}{\sqrt{k-1}} = \sum_{k=1}^{n-1} \frac{1}{\sqrt{k}} \leq \sqrt{n-1} \left(\sum_{k=1}^{n-1} \frac{1}{k} \right)^{1/2} = \sqrt{n-1} \left(\gamma + \log(n-1) + \frac{1}{2(n-1)} \right)^{1/2}$$

where γ is the Bonferroni constant, and we just interpret the second norm as a truncated Harmonic series approximated by Taylor expansion. Thus, overall we get

$$\sum_{k=2}^{n}\sqrt{\frac{n^2}{k-1}} \leq n\sqrt{n-1}\left(\gamma+\log(n-1)+\frac{1}{2(n-1)}\right)^{1/2} = O(n^{1.5}\log(n))$$

If the graph is not connected, at some point we will not be able to find any new edges, and the procedure of theorem 13.1 will fail (we can repeat this procedure a certain number of times to be sure that there are indeed no more valid edges, leveraging the powering lemma, i.e. lemma C.1).

We need to set the failure probability of each run of theorem 13.1. It is simple to check that if we want the probability of failure to be bounded by 2/3 we need to se the probability of failure for a single run of the algorithm as $\delta \geq \frac{1}{3n}$. This is relatively simple to obatin from the union bound (see exercise C.2). □

Exercise 13.1 (Improve bound of number of queries)**.** Can you show that $\sum_{k=2}^{n}\sqrt{\frac{n^2}{(k-1)}} = O(n^{3/2})$, i.e. without the polylogarithmic factor $\log(n)$. Or can you prove that it is not possible to remove it? Hint.

For the array model, we report the theorem of (Dürr et al., 2006).

Theorem 13.3 (Quantum algorithm for graph conectivity (array model))**.** *Assume that U_A is a unitary that gives you query access to the array model of an undirected graph $G=(V,E)$. Then, deciding if a graph is connected has an expected number of queries to U_M of $O(n)$. In particular, algorithm 13.1 returns a spanning tree for G if G is connected, otherwise runs forever.*

13.2 Summary of results

A summary of the query complexities is stated below.

Problem	Adj. Matrix	Array
Minimum Spanning Tree	$O(n^{3/2})$	$O(\sqrt{nm})$
Connectivity	$O(n^{3/2})$	$O(n)$
Strong Connectivity	$O(n^{3/2})$	$\Omega(\sqrt{nm})$, $O(\sqrt{nmlog(n)})$
Single Sourced Shortest Path	$\Omega(n^{3/2})$, $O(n^{3/2}log^2(n))$	$\Omega(\sqrt{nm}log^2(n))$, $O(\sqrt{nm})$

Chapter 14

Lower bounds on query complexity of quantum algorithms

Contributors: Trong Duong

We have discussed about several important quantum algorithms. A large portion of them is query-based, i.e. the input is given as a quantum black box $O : |\mathbf{x}\rangle|y\rangle \mapsto |\mathbf{x}\rangle|y \oplus f(x)\rangle$. In that setting, query complexity plays a key role in determining the advantage and limitation of an algorithm. This chapter will discuss how to obtain a lower bound on query complexity, i.e. the minimal number of queries an algorithm needs to make for a desired output (probably up to some error bound). In particular, we are going to look into **Polynomial method** and **Adversary method**, along with some simple applications.

14.1 Polynomial method

The goal of a quantum query algorithm usually involves determining entirely or partially a function $f : \{0,1\}^n \to \{0,1\}$ given via an oracle $O_f : O_f|\mathbf{x}, a\rangle = |\mathbf{x}, a \oplus y\rangle$, where $y \in \{0,1\}$ is the evaluation of f at the input $\mathbf{x} \in \{0,1\}^n$. The oracle basically swaps $|\mathbf{x}, 0\rangle$ and $|\mathbf{x}, 1\rangle$ when $y = 1$, and does nothing when $y = 0$. For a general superposition of $|\psi\rangle = \sum_{i\in[N]} \alpha_{i,0}(\mathbf{x})|i,0\rangle + \alpha_{i,1}(\mathbf{x})|i,1\rangle$, where $\alpha_{i,a}(\mathbf{x})$ is an N-variate polynomial in terms of x_i's, the degree of $\alpha_{i,a}(\mathbf{x})$ is increased by O_f by at most 1.

$$O_f|\psi\rangle = \sum_{i\in[N]} \left[(1-y_i)\alpha_{i,0}(\mathbf{x}) + y_i\alpha_{\mathbf{x},1}(\mathbf{x})\right]|i,0\rangle + \left[y_i\alpha_{i,0}(\mathbf{x}) + (1-y_i)\alpha_{i,1}(\mathbf{x})\right]|i,1\rangle$$

A quantum algorithm typically calls an interleaved chain of unitary operators and the oracles. Those unitary operators do not depend of $\mathbf{x}$, so O_f's are the only contributors to any increase in the degree of $\alpha_{i,a}(\mathbf{x})$, with an increase of at most 1 for each call to the oracle.

The following lemmas come as consequences of the preceeding observation and the fact that $y_i^2 = y_i$.

Lemma 14.1 (Degree of amplitude polynomials)**.** *Suppose a quantum query algorithm makes T call to O_f. Then the amplitude of every basis state is a multilinear polynomial in terms of x_i's of degree at most T.*

In a decision problem, one define acceptance rate $p(x)$ as the probability of obtaining the ancilla in state 1 at the final measurement. In particular. $p(x) = \sum_{i \in [N]} |\alpha_{i,1}|^2$. By the previous lemma, one can bound the degree of the acceptance rate as a polynomial.

Lemma 14.2 (Degree of acceptance polynomials)**.** *Acceptance rate of the algorithm is a multilinear polynomial in x_i's of degree at most $2T$.*

For an error-bounded algorithm that computes f with error at most ε, the acceptance polynomial $p(\mathbf{x})$ is called an **approximating polynomial** of f if

$$\forall \mathbf{x} \in \{0,1\}^n, |p(\mathbf{x}) - f(\mathbf{x})| \leq \varepsilon$$

As a result, if we can show a approximating polynomial of f has degree at least d, then every quantum algorithm that computes f with error at most ε must make at least $d/2$ queries to the oracle. In practice we typically choose $\varepsilon = 1/3$.

To shed light on the use of polynomial method, we shall look into one of its particular applications when $f(\mathbf{x})$ is symmetric, i.e. f only depends on the Hamming weight $|\mathbf{x}|$, which the number of bits 1 in $\mathbf{x}$. The value of such f is invariant under permutations of $(x_1, x_2, \dots, x_n)$. Suppose we have an approximating polynomial p of f, consider the average $\bar{p}$ of p over all input permutations $\pi(\mathbf{x})$.

$$\bar{p}(\mathbf{x}) = \frac{1}{N!} \sum_{\pi \in S_n} p(x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)})$$

A nice property of this symmetrized polynomial is that it is a uni-variate polynomial in $|x|$ that approximates f. Moreover, the uni-variate polynomial has a degree $d \leq \deg(p)$. To see why this is the case, we observe that

$\bar{p}(\mathbf{x}) = c_0 + c_1 Q_1 + c_2 Q_2 + \dots + c_d Q_d$, where $Q_1 = x_1 + x_2 + \dots + x_n$, $Q_2 = x_1 x_2 + \dots + x_1 x_n + x_2 x_1 + \dots + x_n x_{n-1}$, etc. Consider

$$Q_j = \sum_{\substack{S \subset [N] \\ |S| = j}} \prod_{i \in S} x_i$$

If $j > |x|$, each product within the representation of Q_j is equal to zero, so $Q_j = 0$. If $j \leq |x|$, there are exactly $\binom{|x|}{j}$ nonzero products within Q_j. Those nonzero products equal 1, so $Q_j = \binom{|x|}{j}$. Therefore the symmetrized polynomial is

$$\bar{p}(\mathbf{x}) = c_0 + c_1 \binom{|x|}{1} + \cdots + c_d \binom{|x|}{d} = r(|x|)$$

Also, since p approximates f that only depends on $|\mathbf{x}|$, the sum over all input permutations $\bar{p}$ also approximates f. From all of the above, one has $\deg(\bar{p}) \leq \deg(p) \leq 2T$. The best lower bound for T can be obtained if we find the smallest-degree $\bar{p}$. Sometimes it is not easy to access to the degree of the symmetrized polynomial without having the explicit form of the initial approximating polynomial. Rather, we can relax the lower bound for T relying on the smallest-degree p. The following powerful theorem characterizes the degree of the smallest-degree approximating polynomial.

Theorem 14.1 (Paturi theorem (Paturi, 1992))**.** *Suppose f is a non-constant symmetric boolean function on {0,1}^n and p is some approximating polynomial of f.*

Let $f_k \equiv f(\mathbf{x})$ when $|\mathbf{x}| = k$, and $\Gamma(f) \equiv \min\{|2k - n + 1| : f_k \neq f_{k+1}$ for $0 \leq k \leq n-1\}$.

Then $\min deg(p) \in \Theta(\sqrt{n(n - \Gamma(f))})$

A simple example is the black-box PARITY function $f(\mathbf{x}) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$. Symmetrizing f induces a zero-error polynomial $r(x)$ such that

$$r(k) = \begin{cases} 1, & k \text{ is even} \\ 0, & k \text{ is odd} \end{cases}$$

Note that $r(x)$ changes direction at least n times, so $\deg(r) \geq n$. Note that, for PARITY function, we have $\Gamma(f) = 1$ and $\min\deg(p) \in \Theta(n)$. So $r(x)$ offers the optimal lower bound. Thus the computation of f requires at least $n/2$ queries. According to this, Deutsch's algorithm is the optimal algorithm even in no-error case for $n = 2$.

We will look at another example of OR function. One way to learn the function from the corresponding oracle is using the Grover's algorithm. If the input has at least one $x_i = 1$, the algorithm can find some index j such that $x_j = 1$ with high probability using $\Theta(\sqrt{n})$ queries. As a result, one can compute the OR function with bounded error with $\Theta(\sqrt{n})$ queries. This result agrees with Paturi's theorem with $\Gamma(f) = n - 1$ and $\min\deg(p) \in \Theta(\sqrt{n})$. We can also prove the minimum number of query calls required independently: Computing the symmetric OR function with error $\leq 1/3$ induces a uni-variate approximating polynomial r such that

$$r(0) \in [0, 1/3] \quad \text{and} \quad r(k) \in [2/3, 1]\ \forall k \in \{1, 2, \ldots, n\}$$

A bound for derivatives of a polynomial can be obtained as a function of its degree according to the Markov brothers' inequality, whose statement is given as follows:

Theorem 14.2 (Markov brothers' inequality (Markov, 1890))**.** *Given a polynomial p of degree d such that $|p(x)| \leq L$ for $x \in [a, b]$. Then within the domain, the first derivative of p is bounded.*

$$|p'(x)| \leq \frac{2d^2 L}{b-a}$$

Applying the inequality to r, one have $|r'(x)| \leq \frac{2d^2}{n}$. On the other hand, $r'(x) \geq 1/3$ at some point between 0 and 1 by the Mean Value Theorem. Hence, $T \geq d/2 \geq \Omega(\sqrt{n})$ that gives the same lower bound for number of queries required as the previous method.

However, a similar quadratic speedup is not feasible for an exact algorithm for OR function. Assume such an algorithm exists, then it induces $r(0) = 0$ and $r(k) = 1$, $k \in \{1, 2, \dots, n\}$. The univariate polynomial r then has to change direction at least $n-1$ times to satisfy the constraint, making its degree at least n. Thus $T \geq n/2$, so the exact algorithm cannot achieve a quadratic speedup.

14.2 Quantum adversary method

Consider a decision problem given by a function $f : \{0,1\}^n \to \{0,1\}$. An input $\mathbf{w} = (w_1, \dots, w_n) \in \{0,1\}^n$ to the function can be accessed via a phase oracle $O_{\mathbf{w}} : |i\rangle \mapsto (-1)^{w_i}|i\rangle$. This type of oracle can be obtained from the general oracle $O'_{\mathbf{w}} : |i\rangle|a\rangle \mapsto |i\rangle|a \oplus g(i)\rangle$ by setting the ancilla state $|a\rangle = |-\rangle$ and $g(i) = w_i$. This trick is often referred as *phase kickback.*

$$\begin{aligned} |i\rangle\left(\tfrac{|0\rangle-|1\rangle}{\sqrt{2}}\right) &\mapsto |i\rangle\left(\tfrac{|w_i\rangle-|1\oplus w_i\rangle}{\sqrt{2}}\right) \\ &= \begin{cases} |i\rangle\left(\tfrac{|0\rangle-|1\rangle}{\sqrt{2}}\right), & w_i = 0 \\ |i\rangle\left(\tfrac{|1\rangle-|0\rangle}{\sqrt{2}}\right), & w_i = 1 \end{cases} \\ &= (-1)^{w_i}|i\rangle\left(\tfrac{|0\rangle-|1\rangle}{\sqrt{2}}\right) \end{aligned} \tag{14.1}$$

Our goal is to determine $f(\mathbf{x})$ with high probability, say at least $2/3$, for a given $\mathbf{x}$ using as few $O_{\mathbf{x}}$ queries as possible. An error-bounded algorithm must accept any $\mathbf{x} \in f^{-1}(0)$ with probability $\leq 1/3$ and accept any $\mathbf{y} \in f^{-1}(1)$ with probability $\geq 2/3$.

For an arbitrary input $\mathbf{w}$, a general algorithm can be formulated as a sequence

$$U_T O_{\mathbf{w}} U_{T-1} O_{\mathbf{w}} \dots O_{\mathbf{w}} U_1 O_{\mathbf{w}} U_0$$

Denote $|\psi^t_{\mathbf{w}}\rangle$ be the state of the system after the t-th query to the oracle. Let $\mathbf{x} \in X \equiv \{\mathbf{x} | f(\mathbf{x}) = 0\} = f^{-1}(0)$ and $\mathbf{y} \in Y = f^{-1}(1)$. Define the *progress* *function* of $\mathbf{x}$ and $\mathbf{y}$ as the inner product of their corresponding quantum state though the time period $\langle\psi^t_{\mathbf{x}}|\psi^t_{\mathbf{y}}\rangle$. Define the *progress measure* over a subset R of $X \times Y$ as

$$S(t) = \sum_{(x,y)\in R} |\langle\psi^t_{\mathbf{x}}|\psi^t_{\mathbf{y}}\rangle|$$

Observe that $\langle\psi^0_{\mathbf{x}}|\psi^0_{\mathbf{y}}\rangle = 1$ as the initial states for all inputs are necessarily the same. Also, non-oracle unitary operators U_i do not alter $\langle\psi^t_{\mathbf{x}}|\psi^t_{\mathbf{y}}\rangle$. One can show $|\langle\psi^T_{\mathbf{x}}|\psi^T_{\mathbf{y}}\rangle| \leq \frac{17}{18}$ for an error-bounded algorithm. The proof is based on an inequality between total variation distance and L_2 norm distance.

Definition 14.1 (Total variation distance)**.** The total variation between two probability distributions P and Q over a countable sample space Ω is given by

$$\begin{array}{rcl} d(P,Q) & \equiv & \sup_{A\subseteq\Omega} |P(A) - Q(A)| \\ & = & \frac{1}{2}\sum_{\omega\in\Omega} |P(\omega) - Q(\omega)| \end{array}$$

In the context of quantum measurement, the decision problem involves a 2-outcome measurement at time $t = T$. The probability of obtaining the outcome $m \in \{0,1\}$ by a measurement of quantum states ϕ and τ are $|\phi_m|^2$ and $|\tau_m|^2$ respectively.

$$\begin{array}{rcl} \frac{1}{2}\sum_m ||\phi_m|^2 - |\tau_m|^2| & = & \frac{1}{2}\sum_m ||\phi_m| - |\tau_m|| \cdot ||\phi_m| + |\tau_m|| \\ & \leq & \frac{1}{2}\sum_m |\phi_m - \tau_m| \cdot ||\phi_m| + |\tau_m|| \\ & \leq & \frac{1}{2}\sqrt{\sum_m |\phi_m - \tau_m|^2}\sqrt{\sum_m \left(|\phi_m| + |\tau_m|\right)^2} \\ & \leq & \sqrt{\sum_m |\phi_m - \tau_m|^2} = \|\phi - \tau\|_2 \end{array}$$

The first and second inequality come as a result of triangle inequality and Cauchy-Schwarz inequality respective. The third inequality comes from $(a+b)^2 \leq 2(a^2+b^2)$ and the fact that $\sum_m |\phi_m|^2 = \sum_m |\tau_m|^2 = 1$. Consider an error-bounded algorithm with tolerance of 1/3, i.e. $|\langle 1|\psi^T_{\mathbf{x}}\rangle|^2 < 1/3$ and $|\langle 1|\psi^T_{\mathbf{y}}\rangle|^2 \geq 2/3$. The total variation distance, and therefore $\|\psi^T_{\mathbf{x}} - \psi^T_{\mathbf{y}}\|_2$, is at least 1/3. Notice that the L_2-norm distance between two states can be written in terms of their inner product

$$\|\psi^T_{\mathbf{x}} - \psi^T_{\mathbf{y}}\|^2_2 = \langle\psi^T_{\mathbf{x}} - \psi^T_{\mathbf{y}}|\psi^T_{\mathbf{x}} - \psi^T_{\mathbf{y}}\rangle = 2 - 2\Re\langle\psi^T_{\mathbf{x}}|\psi^T_{\mathbf{y}}\rangle. \tag{14.2}$$

We can assume that $\langle\psi^T_x|\psi^T_y\rangle$ is real, otherwise multiply $|\psi^T_y\rangle$ by some scalar of norm 1 to make the inner product real. Then it follows that $|\langle\psi^T_x|\psi^T_y\rangle| \leq 17/18$. Intuitively, as the inner product is bounded above, the measurement statistics can distinguish $\mathbf{x} \in f^{-1}(0)$ and $\mathbf{y} \in f^{-1}(1)$.

However we should rather look at more than a fixed pair $\mathbf{x}, \mathbf{y}$ to obtain a meaningful result. This suggests us to look at some particular subset $R \subseteq X \times Y$. Observe that the progress measure before and after the algorithm are $S(0) = |R|$ and $S(T) \leq \frac{17}{18}|R|$. So, if one can come up with an upper bound Δ for $|S(t) - S(t-1)|$, then the number of queries needed is $T \geq |S(T) - S(0)|/\Delta = |R|/18\Delta$. We prove the following theorem based on that idea.

Theorem 14.3 (Basic Adversary Method (Ambainis, 2002))**.** *Let f be a decision problem, $X \subseteq f^{-1}(0), Y \subseteq f^{-1}(1)$, and a* binary relation $R \subseteq X \times Y$. *Suppose that*

1. *$\forall \mathbf{x} \in X$, there are at least m_0 distinct $\mathbf{y} \in Y$ such that $(\mathbf{x}, \mathbf{y}) \in R$*
2. *$\forall \mathbf{y} \in Y$, there are at least m_1 distinct $\mathbf{x} \in X$ such that $(\mathbf{x}, \mathbf{y}) \in R$*
3. *$\forall \mathbf{x} \in X$ and $\forall i \in \{0, 1, \dots, n\}$, there are at most l_0 distinct $\mathbf{y} \in Y$ such that $x_i \neq y_i$ and $(\mathbf{x}, \mathbf{y}) \in R$*
4. *$\forall \mathbf{y} \in Y$ and $\forall i \in \{0, 1, \dots, n\}$, there are at most l_1 distinct $\mathbf{x} \in X$ such that $x_i \neq y_i$ and $(\mathbf{x}, \mathbf{y}) \in R$.*

Then the quantum query complexity of f is $\Omega\sqrt{\frac{m_0 m_1}{l_0 l_1}}$

The following proof is modified from a proof for the case $l_0 = l_1 = 1$ given in the lecture note (O'Donnell, 2015).

For each $(\mathbf{x}, \mathbf{y}) \in R$, let $J_{\mathbf{xy}} = \{(j_1, j_2, \dots, j_k) | x_{j_1} \neq y_{j_1}, x_{j_2} \neq y_{j_2}, \dots, x_{j_k} \neq y_{j_k}\}$. From the first assumption, $|R| \geq m_0|X|$. The quantum states corresponding to the initial inputs right before they pass through the t-th oracle can be represented by

$$|\psi_{\mathbf{x}}^{t-1}\rangle = \sum_i a_i |i\rangle \otimes |\phi_i\rangle \tag{14.3}$$

$$|\psi_{\mathbf{y}}^{t-1}\rangle = \sum_i b_i |i\rangle \otimes |\chi_i\rangle \tag{14.4}$$

$$\Rightarrow \langle\psi_{\mathbf{x}}^{t-1}|\psi_{\mathbf{y}}^{t-1}\rangle = \sum_i a_i^* b_i \langle\phi_i|\chi_i\rangle \tag{14.5}$$

When the states pass through the t-th oracle, it flips the sign on each a_i, b_i whenever $x_i = 1$ or $y_i = 1$ respectively. The overall effect on the inner product is flipping the sign of the coefficient corresponding to $\langle\phi_i|\chi_i\rangle$ when $x_i \neq y_i$. We can express the inner product at time t as follows

$$\langle\psi_{\mathbf{x}}^t|\psi_{\mathbf{y}}^t\rangle = \sum_{i \notin J_{\mathbf{xy}}} a_i^* b_i \langle\phi_i|\chi_i\rangle - \sum_{j \in J_{\mathbf{xy}}} a_j^* b_j \langle\phi_i|\chi_i\rangle$$

The change in the inner product at two successive time is

$$\langle \psi_{\mathbf{x}}^t | \psi_{\mathbf{y}}^t \rangle - \langle \psi_{\mathbf{x}}^{t-1} | \psi_{\mathbf{y}}^{t-1} \rangle = 2 \sum_{j \in J_{\mathbf{xy}}} a_i^* b_i \langle \phi_i | \chi_i \rangle$$

$$\begin{array}{rcc} |S(t) - S(t-1)| & = & \left| \sum_{(\mathbf{x},\mathbf{y}) \in R} |\langle \psi_{\mathbf{x}}^t | \psi_{\mathbf{y}}^t \rangle| - \sum_{(\mathbf{x},\mathbf{y}) \in R} |\langle \psi_{\mathbf{x}}^{t-1} | \psi_{\mathbf{y}}^{t-1} \rangle| \right| \\ & \leq & \sum_{(\mathbf{x},\mathbf{y}) \in R} \left| \langle \psi_{\mathbf{x}}^t | \psi_{\mathbf{y}}^t \rangle - \langle \psi_{\mathbf{x}}^{t-1} | \psi_{\mathbf{y}}^{t-1} \rangle \right| \\ & = & \sum_{(\mathbf{x},\mathbf{y}) \in R} \left| 2 \sum_{j \in J_{\mathbf{xy}}} a_{j(\mathbf{x},\mathbf{y})}^* b_{j(\mathbf{x},\mathbf{y})} \langle \phi_i | \chi_i \rangle \right| \\ & \leq & \sum_{(\mathbf{x},\mathbf{y}) \in R} \sum_{j \in J_{\mathbf{xy}}} 2 \left| a_{j(\mathbf{x},\mathbf{y})}^* b_{j(\mathbf{x},\mathbf{y})} \right| \\ & \leq & \sum_{(\mathbf{x},\mathbf{y}) \in R} \sum_{j \in J_{\mathbf{xy}}} \sqrt{\frac{m_0 l_1}{m_1 l_0}} \left| a_{j(\mathbf{x},\mathbf{y})} \right|^2 + \sqrt{\frac{m_1 l_0}{m_0 l_1}} \left| b_{j(\mathbf{x},\mathbf{y})} \right|^2 \end{array}$$

The last line comes from the simple inequality $|a|^2 + |b|^2 \geq 2|ab|$. Consider the first summand in the expression above.

$$\begin{array}{rcc} \sum_{(\mathbf{x},\mathbf{y}) \in R} \sum_{j \in J_{\mathbf{xy}}} \sqrt{\frac{m_0 l_1}{m_1 l_0}} \left| a_{j(\mathbf{x},\mathbf{y})} \right|^2 & = & \sqrt{\frac{m_0 l_1}{m_1 l_0}} \sum_{\mathbf{x} \in X} \sum_{i \in [N]} \sum_{\substack{\mathbf{y} \in Y \\ y_i \neq x_i}} |a_{i(\mathbf{x},\mathbf{y})}|^2 \\ & \leq & \sqrt{\frac{m_0 l_1}{m_1 l_0}} \sum_{\mathbf{x} \in X} l_0 \\ & \leq & \sqrt{\frac{m_0 l_1}{m_1 l_0}} \frac{|R|}{m_0} l_0 = \sqrt{\frac{l_0 l_1}{m_0 m_1}} |R| \end{array}$$

The second line comes from the third assumption and the fact that $\sum_{i \in [N]} |a_{i(\mathbf{x},\mathbf{y})}|^2 = 1$ for every $(\mathbf{x}, \mathbf{y}) \in R$. We can also derive the same bound for the second summand. The upper bound for $|S(t) - S(t-1)|$ is $\Delta = 2\sqrt{\frac{l_0 l_1}{m_0 m_1}} |R|$. Hence $T \geq \frac{|R|}{18\Delta} \in \Omega\left(\sqrt{\frac{m_0 m_1}{l_0 l_1}}\right)$, which is the conclusion of the theorem.

We revisit two examples of PARITY and OR functions and show the adversary method and the polynomial method give the same lower bound. Recall from the Polynomial method that one needs at least $n/2$ queries for PARITY and $\Theta(\sqrt{n})$ for OR. Let's see if the same result can be obtained with the Adversary method. The PARITY function f maps binary strings with odd Hamming weight to 1 and even Hamming weight to 0. Let $X = f^{-1}(0), Y = f^{-1}(1)$, and $R = \{(\mathbf{x}, \mathbf{y}) | d(\mathbf{x}, \mathbf{y}) = 1\}$. It not difficult to see that $m_0 = m_1 = n, l_0 = l_1 = 1$ so that the function requires $\Omega(n)$ queries. The OR function g maps only the all-zero sequence to 0 and other inputs to 1. Let $X = \{00 \ldots 00\} = g^{-1}(0)$ and $Y = \{\mathbf{y} | \mathbf{y} \text{ contains exactly one bit } 1\} \subset g^{-1}(1)$. Then, $m_0 = n, m_1 = 1, l_0 = l_1 = 1$, so we have a lower bound of $\Omega(\sqrt{n})$.

Part III

Everything else

Chapter 15

Selected works on quantum algorithms

This is a work in progress, as the vast majority of works are not present here, yet. Obviously, feel free to write at “scinawa [at] luongo . pro” for suggestions, or open an issue on github. Please understand that the aim of this section if to select relevant *quantum algorithms*. Special interested is devoted to works that can be applied for data analysis or used as other subroutines for other QML algorithms.

2022

- An efficient quantum algorithm for lattice problems achieving subexponential approximation factor `#algo #crypto`
- Improved quantum algorithms for linear and nonlinear differential equations `#algo`
- New Quantum Algorithms for Computing Quantum Entropies and Distances `#algo`
- Quantum machine learning with subspace states `#algo`
- A quantum algorithm for solving eigenproblem of the Laplacian matrix of a fully connected graph `#algo`
- Quantum State Preparation with Optimal Circuit Depth: Implementations and Applications `#algo, #theory`
- Quantum Meets Fine-Grained Complexity: Sublinear Time Quantum Algorithms for String Problems `#algo`
- Two-level Quantum Walkers on Directed Graphs II: An Application to qRAM `#algo`
- Memory Compression with Quantum Random-Access Gates
- Mean estimation when you have the source code; or, quantum Monte Carlo methods

- Exact and efficient Lanczos method on a quantum computer
- On establishing learning separations between classical and quantum machine learning with classical data
- Matching Triangles and Triangle Collection: Hardness based on a Weak Quantum Conjecture
- Partition Function Estimation: Quantum and Quantum-Inspired Algorithms
- A Faster Quantum Algorithm for Semidefinite Programming via Robust IPM Framework
- Quantum Subroutine Composition #algo
- Complexity-Theoretic Limitations on Quantum Algorithms for Topological Data Analysis #theory
- Quantum Algorithms for Sampling Log-Concave Distributions and Estimating Normalizing Constants #algo
- Quantum divide and conquer

2021

- Information-theoretic bounds on quantum advantage in machine learning #theory
- Noisy intermediate-scale quantum (NISQ) algorithms #review, #variational A massive review on the state-of-the-art quantum algorithms for NISQ architectures. It highlights the limitations, but also the wins of the variational paradigm.
- Parallel Quantum Algorithm for Hamiltonian Simulation #algo
- Quantum Perceptron Revisited: Computational-Statistical Tradeoffs #algo
- Lower bounds for monotone arithmetic circuits via communication complexity #theory
- Fast algorithm for quantum polar decomposition, pretty-goodmeasurements, and the Procrustes problem #algo
- Quantum Algorithms based on the Block-Encoding Framework for Matrix Functions by Contour Integrals #algo
- Classical and Quantum Algorithms for Orthogonal Neural Networks #algo
- Quantum Semi Non-negative Matrix Factorization #algo
- Quantum Algorithms based on the Block-Encoding Framework for Matrix Functions by Contour Integrals
- Quantum Alphatron #algo
- Quantum SubGaussian Mean Estimator #algo
- A randomized quantum algorithm for statistical phase estimation #algo
- Near-Optimal Quantum Algorithms for String Problems #algo
- Quantum Algorithms and Lower Bounds for Linear Regression with Norm Constraints #algo
- Dequantizing the Quantum Singular Value Transformation: Hardness and Applications to Quantum Chemistry and the Quantum PCP Conjecture #algo

- A randomized quantum algorithm for statistical phase estimation #algo
- Nearly Optimal Quantum Algorithm for Estimating Multiple Expectation Values #algo
- Quantum Algorithms for Reinforcement Learning with a Generative Model #algo #qrl #qmc
- Quantum Monte-Carlo Integration: The Full Advantage in Minimal Circuit Depth #algo #qmc
- Near-Optimal Quantum Algorithms for Multivariate Mean Estimation #algo #qmc
- Quantum algorithms for multivariate Monte Carlo estimation #algo #qmc
- Quantum Sub-Gaussian Mean Estimator #algo #qmc
- Quantum algorithm for stochastic optimal stopping problems #qfinance #algo
- Quantum Machine Learning For Classical Data #thesis
- Quantum algorithms for anomaly detection using amplitude estimation #algo
- Two-level Quantum Walkers on Directed Graphs I: Universal Quantum Computing #algo
- Quantum algorithms for learning a hidden graph and beyond #algo
- Asymptotically Optimal Circuit Depth for Quantum State Preparation and General Unitary Synthesis

2020

- Variational Quantum Algorithms #review
- Circuit-centric Quantum Classifier #variational
- Quantum polar decomposition algorithm #algo
- The power of quantum neural networks #variational
- Robust quantum minimum finding with an application to hypothesis selection #algo
- Quantum exploration algorithms for multi-armed bandits #algo
- Sublinear classical and quantum algorithms for general matrix games #algo

2019

- Quantum Language Processing #NLP
- A Quantum Search Decoder for Natural Language Processing #NLP
- Quantum and Classical Algorithms for Approximate Submodular Function Minimization #algo
- Quantum algorithms for zero-sum games #algo
- Practical implementation of a quantum backtracking algorithm #experiment
- Quantum speedup of branch-and-bound algorithms #algo
- The Quantum Version Of Classification Decision Tree Constructing Algorithm C5.0 #algo

- Sublinear quantum algorithms for training linear and kernel-based classifiers `#algo`
- Quantum Algorithms for Classical Probability Distributions `#algo` `#qmc`

2018

- Continuous-variable quantum neural networks A work presented at TQC2018 that exploit deep similarities between the mathematical formulation of NN and photinics
- Classification with quantum neural networks on near term processors `#variational`
- Artificial Quantum Neural Network: quantum neurons, logical elements and tests of convolutional nets. A new approach to qnn ø/. This skips complitely the unitary and gate based quantum computation Also here the model is mean to be trained by classical optimization.
- Optimizing quantum optimization algorithmsvia faster quantum gradient computation `#algo`
- Quantum Statistical Inference `#phdthesis, #algo` A PhD thesis on QML and other aspects of quantum information. With focus on Gaussian Processes, Quantum Bayesian Deep Learning (and other resources about causality and correlations..).
- Troubling Trends in Machine Learning Scholarship `#opinion-paper` Is a self-autocritic of the ML community on the way they are doing science now. I think this might be relevant as well for the QML practicioner.
- Quantum machine learning for data scientits `#review` `#tutorial` This is a very nice review of some of the most known qml algorithms.
- Quantum algorithm implementations for beginners `#review` `#tutorial`
- Quantum linear systems algorithms: a primer `#review`
- Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics `#algo`
- The power of block-encoded matrix powers: improved regression techniques via faster Hamiltonian simulation `#algo`
- Applying quantum algorithms to constraint satisfaction problems `#resource-estimation`
- Quantum Chebyshev's Inequality and Applications `#qmc`
- From linear combination of quantum states to Grover's searching algorithm

2017

- Implementing a distance based classifier with a quantum interference circuit `#algo`
- Quantum SDP solvers: Large speed-ups, optimality, and applications to quantum learning `#algo`
- Quantum machine learning for quantum anomaly detection `#algo` Here the authors used previous technique to perform anomaly detection. Basi-

cally they project the data on the 1-dimensional subspace of the covariance matrix of the data. In this way anomalies are supposed to lie furhter away from the rest of the dataset.

- Quantum machine learning: a classical perspective: `#review #quantum learning theory`
- Quantum Neuron: an elementary building block for machine learning on quantum computers
- Quantum speedup of Monte Carlo methods `#algo`
- Improved quantum backtracking algorithms using effective resistance estimates `#algo`

2016

- Quantum Discriminant Analysis for Dimensionality Reduction and Classification `#algo`
- An efficient quantum algorithm for spectral estimation `#algo`
- Quantum Recommendation Systems `#algo`

2015

- Advances in quantum machine learning `#implementations`, `#review` It cover things up to 2015, so here you can find descriptions of Neural Networks, Bayesian Networks, HHL, PCA, Quantum Nearest Centroid, Quantum k-Nearest Neighbour, and others. -Quantum walk speedup of backtracking algorithms `#algo`
- Quantum algorithms for topological and geometric analysis of data `#algo`

2014

- Quantum Algorithms for Nearest-Neighbor Methods for Supervised and Unsupervised Learning `#algo`
- Quantum support vector machine for big data classification `#algo` This was one of the first example on how to use HHL-like algorithms in order to get something useful out of them.
- Improved Quantum Algorithm for Triangle Finding via Combinatorial Arguments `#algo`
- Fixed-point quantum search with an optimal number of queries `#algo`
- Quantum Principal Component Analysis `#algo`

2013

- Quantum algorithms for supervised and unsupervised machine learning `#algo`
- Exponential improvement in precision for simulating sparse Hamiltonians `#algo`

2010

- Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations `#algo`
- Any and-or formula of size n can be evaluated in time $n^{1/2+o(1)}$ on a quantum computer

2009

- Quantum algorithms for linear systems of equations `#algo`

2007

- A Quantum Algorithm for the Hamiltonian NAND Tree

2005

- Fast quantum algorithm for numerical gradient estimation `#algo`

1999

- The quantum query complexity of approximating the median and related statistics `#algo`

1996

- A fast quantum mechanical algorithm for estimating the median `#algo`

Chapter 16

Solutions to exercises

Appendix A

Math and linear algebra

A.1 Norms, distances, trace, inequalities

Take a look at this, and this.

Just remember that for a complez number z, $z\overline{z} = a^2 + b^2 = |z|^2$, and $\frac{1}{z} = \frac{\overline{z}}{\overline{z}z}$, and the conjugation is the mapping $z = (a + ib) \mapsto \overline{z} = (a - ib)$.

Definition A.1 (Inner product)**.** A function $f : A \times B \mapsto \mathbb{C}$ is called an inner product $(a, b) \mapsto z$ if:

- $(a, b + c) = (a, b) + (a, c)$ (and respectively for $(a + c, b)$)
- $(a, \alpha b) = \alpha(a, b)$

Definition A.2 (Norm)**.** A function $f : \mathbb{R}^d \mapsto \mathbb{R}$ is called a norm if:

- $\|x\| \geq 0 \forall x \in \mathbb{R}^d$, also $\|x\| = 0$ iff $x = 0$ (positive definiteness)
- $\|\alpha x\| = \alpha\|x\| \forall x \in \mathbb{R}^d$ and $\forall \alpha \in \mathbb{R}$ (positively homogeneous)
- $\|x\| - \|y\| \leq \|x + y\| \leq \|x\| + \|y\|$ (triangle inequality).

Note that along with triangle inequality you might also need to know the reverse triangle inequality

The triangle inequality is basically the subadditivity property of the norm. It is simople to see that norms are **not** linear operators.

Theorem A.1 (Cauchy(-Bunyakovsky)-Schwarz)**.**

$$|(x, y)| \leq \|x\|\|y\|$$

Proof. Note that by taking the square on both sides we get: $(x, y)^2 \leq (x, x)(y, y)$. Substituting $(x, y) = \|x\|\|y\|cos(\theta)$, we get:

$$|\|x\|^2\|y\|^2 cos^2(\theta)| \leq (x, x)(y, y)$$

The inequality follows from noting that $|cos(\theta)|$ is always ≤ 1. □

Remark. It is simple to see - using Cauchy-Schwarz - that for a vector x we have that:

$$\|x\|_1 \leq \sqrt{n}\|x\|_2$$

We will use the following matrix norms:

- $\|A\|_0$ as the number of non-zero elements of the matrix A,
- $\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=0}^{n} |a_{ij}|$ is the maximum among the sum of the absolute value along the columns of the matrix,
- $\|A\|_2 = \|A\| = \sigma_1$ is the biggest singular value of the matrix,
- $\|A\|_\infty = \max_{1\leq i\leq m} \sum_{j=0}^{n} |a_{ij}|$ is the maximum among the sum of the absolute values along the rows of the matrix,
- $\|A\|_{\max}$ is the maximal element of the matrix in absolute value.
- $\|A\|_F$ is the Frobenius norm of the matrix, defined as $\sqrt{\sum_{ij} a_{ij}^2}$

Note that for symmetric matrices, $\|A\|_\infty = \|A\|_1$.

Exercise A.1 (bound error on product of matrices)**.** Suppose that $\|A-\overline{A}\|_F \leq \epsilon\|A\|_F$. Bound $\|A^T A - \overline{A}^T\overline{A}\|_F$

Definition A.3 (Distance)**.** A function $f : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is called a distance if:

- $d(x, y) \geq 0$
- $d(x, y) = 0$ iif $x = y$
- $d(x, y) = d(y, x)$
- $d(x, z) \leq d(x, y) + d(y, z)$

Definition A.4 (Convex and concave function)**.** A function f defined on a convex vector space D is said to be *concave* if, $\forall\lambda \in [0, 1]$, and $\forall x, y \in D$:

$$f((1-\alpha)x + \alpha y) \geq (1-\alpha)f(x) + \alpha f(y)$$

Conversely, a function f defined on a convex vector space D is said to be *convex* if, $\forall\lambda \in [0, 1]$, and $\forall x, y \in D$:

$$f((1-\alpha)x + \alpha y) \leq (1-\alpha)f(x) + \alpha f(y)$$

A.1.0.1 Properties of the trace operator

- $Tr[A + B] = Tr[A] + Tr[B]$
- $Tr[A \otimes B] = Tr[A]Tr[B]$
- $Tr_1[A \otimes B] = Tr[A]B$
- $Tr[|a\rangle\langle b|] = \langle a|b\rangle$
- $Tr[AB] = \langle A, B\rangle$

where inner product between matrices is basically defined pointwise as $\sum_{ij} a_{ij}b_{ij}$

Exercise A.2. Can you show that the last identity is true?

A.1.0.2 Properties of tensor product

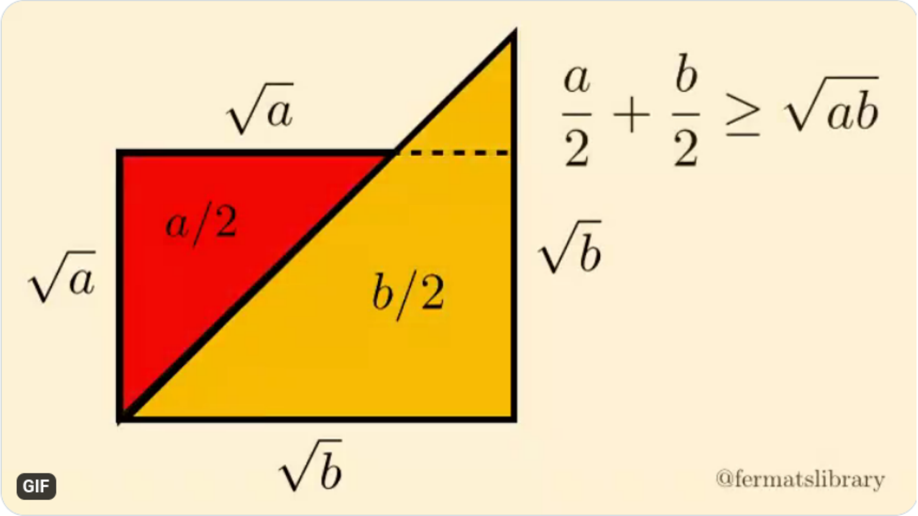
Given two liner maps $V_1 : W_1 \mapsto V_1$ and $V_2 : W_2 \mapsto V_2$ we define the tensor product as the linear map:

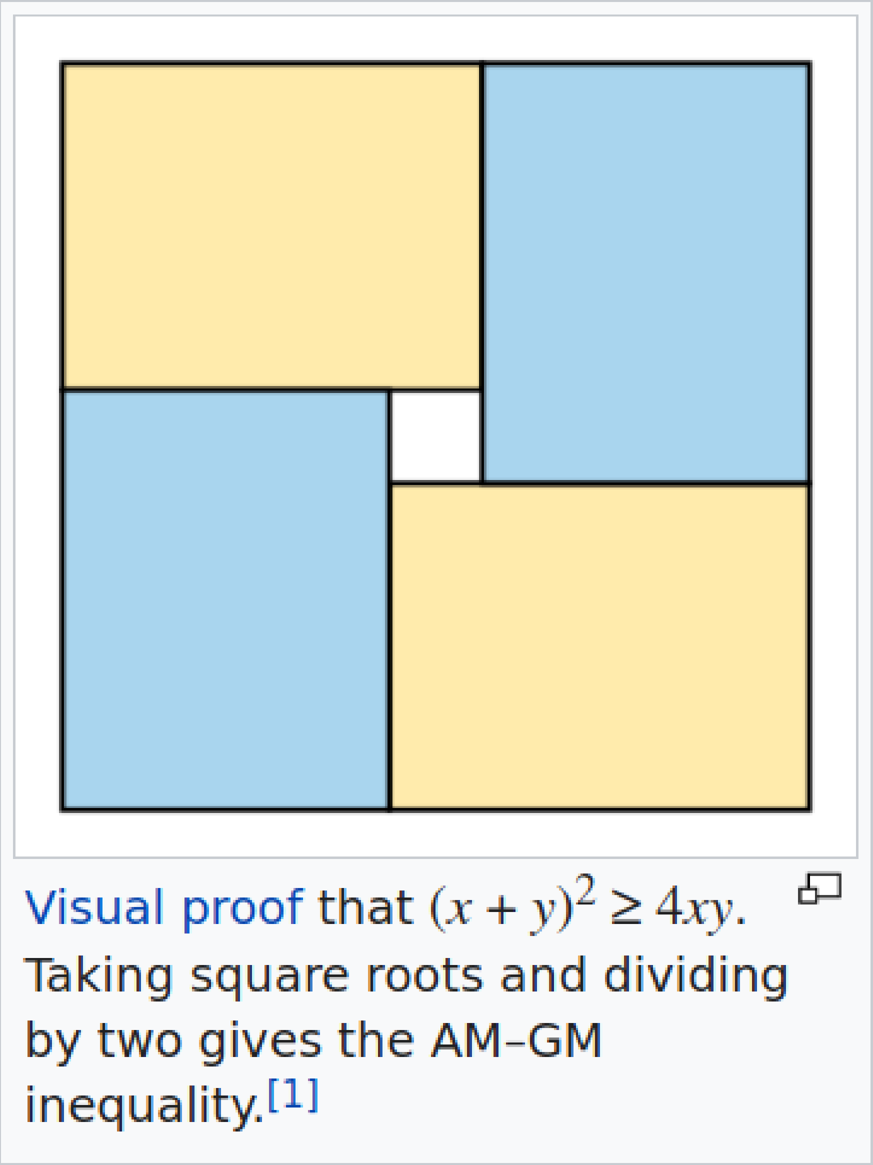
$$V_1 \otimes V_2 : V_1 \otimes V_2 \mapsto W_1 \otimes W_2$$

- $\alpha v \otimes w = v \otimes \alpha w = \alpha(v \otimes w)$
- $(v_1 + v_2) \otimes w = (v_1 \otimes w) + (v_2 \otimes w)$ (and the symmetric of it)
- $|\psi_1\rangle\langle\phi_1| \otimes |\psi_2\rangle\langle\phi_2| = |\psi_1\rangle|\psi_2\rangle \otimes \langle\phi_1|\langle\phi_2|$

When a basis is decided for representing linear maps between vector spaces, the tensor product becomes the Kroeneker product.

A.1.0.3 Useful inequalities





Visual proof that $(x + y)^2 \geq 4xy$. Taking square roots and dividing by two gives the AM–GM inequality.[1]

Theorem A.2 (Binomial theorem)**.**

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$

A.2 Linear algebra

A.2.1 Eigenvalues, eigenvectors and eigendecomposition of a matrix

Real matrices are important tools in Machine Learning as they allow to comfortably represent data and describe the operations to perform during an algorithm. Eigenvectors and eigenvalues are fundamental linear algebra concepts that provide important information about a matrix.

Definition A.5 (Eigenvalues and Eigenvectors (Section 6.1 page 289 from (Strang, 2016)))**.** Let A be a $\mathbb{R}^{n\times n}$ square matrix, $q \in \mathbb{R}^n$ a non-zero vector and λ a scalar. If the following equation is satisfied

$$Aq = \lambda q,$$

then q is said to be an eigenvector of matrix A and λ is its associated eigenvalue.

To have a geometric insight into what eigenvectors and eigenvalues are, we can think of any matrix as a linear transformation in the $\mathbb{R}^n$ space. Under this light, we can say that the eigenvectors of a matrix are those vectors of the space that, after the transformation, lie on their original direction and only get their magnitude scaled by a certain factor: the eigenvalue.

The eigenvalues reveal interesting properties of a matrix. For example, the trace of a matrix (i.e. the sum of the element along the main diagonal of a square matrix) is the sum of its eigenvalues

$$Tr[A] = \sum_{i}^{n} \lambda_i,$$

and its determinant is equal to the product of the eigenvalues (Section 6.1 page 294 from (Strang, 2016))

$$det(A) = \prod_{i}^{n} \lambda_i.$$

Moreover, a matrix A with eigenvalues $\{\lambda_1, ..., \lambda_k\}$ has an inverse only if all the eigenvalues are not zero. The inverse has eigenvalues $\{\frac{1}{\lambda_1}, ..., \frac{1}{\lambda_k}\}$.

Generally, one eigenvalue can be associated with multiple eigenvectors. There might be a set of vectors $E(\lambda) \subseteq \mathbb{R}^n$ such that for all those vectors $q \in E(\lambda) : Aq = \lambda q$. That is why for each eigenvalue we talk about an eigenspace.

Definition A.6 (Eigenspace (Definition 7.1.5 page 108 (Manara et al., 2007)))**.** Let A be a $\mathbb{R}^{n \times n}$ square matrix and λ be an eigenvalue of A. The eigenspace of A related to λ is the space defined over the set of vectors $E(\lambda) = \{x : Ax = \lambda x\}$.

For each eigenspace, through the Gram-Schmidt procedure, starting from linearly independent vectors it is possible to identify a set of orthogonal eigenvectors that constitute a basis for the space. The basis that spans the space where all the eigenvectors of a matrix lie is called eigenbasis.

Definition A.7 (Eigenbasis)**.** A basis for the space where all the eigenvectors of a matrix lie is called eigenbasis.

An important result is that vectors in different eigenspaces are linearly independent.

Lemma A.1 (Linear independence of eigenvectors (Lemma 7.2.3 page 112 from (Manara et al., 2007)))**.** *The set of vectors obtained by the union of the bases of the eigenspaces of a matrix is linearly independent.*

This means that if the sum of the dimensions of the eigenspaces $\sum_i dim(E(\lambda_i))$ equals n, it is possible to find n eigenvectors of A that form a basis for the $\mathbb{R}^n$ space. If that is the case, each vector that lies in $\mathbb{R}^n$ can be written as a linear combination of the eigenvectors of A. Interestingly, matrices that have n linearly independent eigenvectors can be decomposed in terms of their eigenvalues and eigenvectors.

Theorem A.3 (Eigendecomposition or Diagonalization)**.** *(Strang, 2016, Section 6.2 page 304) Let $A \in \mathbb{R}^{n \times n}$ be a square matrix with n linearly independent eigenvectors. Then, it is possible to decompose the matrix as*

$$A = Q \Lambda Q^{-1}.$$

Where $Q \in \mathbb{R}^{n \times n}$ is an square matrix and $\Lambda \in \mathbb{R}^{n \times n}$ is a diagonal matrix. In particular, each i^{th} column of Q is an eigenvector of A and the i^{th} entry of Λ is its associated eigenvalue.

The matrices that can be eigendecomposed are also said diagonalizable, as in practice the theorem above states that such matrices are similar to diagonal matrices. Unfortunately, not all the square matrices have enough independent eigenvectors to be diagonalized. The Spectral Theorem provides us with a set of matrices that can always be eigendecomposed.

Theorem A.4 (Spectral theorem)**.** *(Strang, 2016, Spectral Theorem page 339) Every symmetric matrix is diagonalizable $A = Q \Lambda Q^{-1}$. Furthermore, its eigenvalues are real and it is possible to choose the columns of Q so that it is an orthogonal matrix.*

Recall that a matrix Q is said to be orthogonal if $QQ^T = Q^T Q = \mathbb{I}$, therefore $Q^{-1} = Q^T$. The Spectral theorem, together with the fact that matrices like $A^T A$ and AA^T are symmetric, will come in handy in later discussions.

Being able to eigendecompose a matrix allows performing some computations faster than otherwise. Some examples of operations that gain speed-ups from the eigendecomposed representation are matrix inversion and matrix exponentiation. Indeed, if we have a matrix $A = Q\Lambda Q^{-1}$ its inverse can be computed as $A^{-1} = Q\Lambda^{-1}Q^{-1}$ where $\Lambda^{-1} = diag([\frac{1}{\lambda_1}, \dots \frac{1}{\lambda_n}])$. It is easy to check that this matrix is the inverse of A:

$$AA^{-1} = (Q\Lambda Q^{-1})(Q\Lambda^{-1}Q^{-1}) = Q\Lambda\Lambda^{-1}Q^{-1} = QQ^{-1} = \mathbb{I}$$

$$A^{-1}A = (Q\Lambda^{-1}Q^{-1})(Q\Lambda Q^{-1}) = Q\Lambda^{-1}\Lambda Q^{-1} = QQ^{-1} = \mathbb{I}.$$

At the same time, the eigendecomposition of a matrix allows performing matrix exponentiation much faster than through the usual matrix multiplication. In fact, it is true that $A^p = Q\Lambda^p Q^{-1}$. For instance,

$$A^3 = (Q\Lambda Q^{-1})(Q\Lambda Q^{-1})(Q\Lambda Q^{-1}) = Q\Lambda(Q^{-1}Q)\Lambda(Q^{-1}Q)\Lambda Q^{-1} = Q\Lambda\Lambda\Lambda Q^{-1} = Q\Lambda^3 Q^{-1}.$$

Computing big matrix powers such as A^{100}, with its eigendecomposed representation, only takes two matrix multiplications instead of a hundred.

Traditionally, the computational effort of performing the eigendecomposition of a $\mathbb{R}^{n\times n}$ matrix is in the order of $O(n^3)$ and may become prohibitive for large matrices (Partridge and Calvo, 1997).

A.2.2 Singular value decomposition

Eigenvalues and eigenvectors can be computed only on square matrices. Moreover, not all matrices can be eigendecomposed. For this reason, we introduce the concepts of singular values and singular vectors, that are closely related to the ones of eigenvalues and eigenvectors, and offer a decomposition for all the kind of matrices.

Theorem A.5 (Singular Value Decomposition (Sections 7.1, 7.2 from [@linear-algebragilbert)**.** *)] (Strang, 2016, Sections 7.1, 7.2) Any matrix $A \in \mathbb{R}^{n\times n}$ can be decomposed as*

$$A = U\Sigma V^T$$

where $U \in \mathbb{R}^{n\times r}$ and $V \in \mathbb{R}^{m\times r}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{r\times r}$ is a diagonal matrix. In particular, each i^{th} column of U and V are respectively called left and right singular vectors of A and the i^{th} entry of Σ is their associated singular value. Furthermore, r is a natural number smaller then m and n.

Another (equivaloent) definition of SVD is the following:

$$A = (U, U_0)\begin{pmatrix}\Sigma & 0\\ 0 & 0\end{pmatrix}(V, V_0)^T.$$

The matrix Σ is a diagonal matrix with $\Sigma_{ii} = \sigma_i$ being the singular values (which we assume to be sorted $\sigma_1 \geq \dots \geq \sigma_n$).

The matrices (U, U_0) and (V, V_0) are unitary matrices, which contain a basis for the column and the row space (respectively U and V) and the left null-space and right null-space (respectively U_0 and V_0). Oftentimes, it is simpler to define the SVD of a matrix by simply discarding the left and right null spaces, as $A = U\Sigma V^T$, where U, V are orthogonal matrices and $\Sigma \in \mathbb{R}^{r\times r}$ is a diagonal matrix with real elements, as we did in Theorem A.5.

Similarly to how eigenvalues and eigenvectors have been defined previously, for each pair of left-right singular vector, and the associated singular value, the following equation stands:

$$Av = \sigma u.$$

If we consider the Singular Value Decomposition (SVD) under a geometric perspective, we can see any linear transformation as the result of a rotation, a scaling, and another rotation. Indeed, if we compute the product between a matrix $A \in \mathbb{R}^{n\times m}$ and a vector $x \in \mathbb{R}^m$

$$Ax = U\Sigma V^T x = (U(\Sigma(V^T x))).$$

U and V^T, being orthogonal matrices, only rotate the vector without changing its magnitude, while Σ, being a diagonal matrix, alters its length.

It is interesting to note that the singular values of A - denoted as $\{\sigma_1, ..., \sigma_r\}$ - are the square roots $\{\sqrt{\lambda_1}, ..., \sqrt{\lambda_r}\}$ of the eigenvalues of AA^T (or A^TA) and that the left and right singular vectors of A - denoted as $\{u_1, ..., u_r\}$ and $\{v_1, ..., v_r\}$ - are respectively the eigenvectors of AA^T and A^TA.

The fact that each matrix can be decomposed in terms of its singular vectors and singular values, as in the theorem above, makes the relationship between singular values - singular vectors of a matrix and eigenvalues - eigenvectors of its products with the transpose clearer:

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T;$$

$$A^TA = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T.$$

Note that the matrices AA^T and A^TA are symmetric matrices and so, for the Spectral theorem, we can always find an eigendecomposition. Moreover, note that they have positive eigenvalues: being the square roots of real positive eigenvalues, the singular values of a real matrix are always real positive numbers.

As the left and right singular vectors are eigenvectors of symmetric matrices, they can be chosen to be orthogonal as well. In particular, the left singular vectors of a matrix span the row space of the matrix, and the right singular vectors span the column space.

Definition A.8 (Column (row) Space (Definition 8.1 page 192 [@schlesinger)**.**)] (Schlesinger, 2011, Definition 8.1 page 192) Let A be a $\mathbb{R}^{n\times m}$ matrix. The column (row) space of A is the space spanned by the column (row) vectors of

A. Its dimension is equal to the number of linearly independent columns (rows) of A.

The number r of singular values and singular vectors of a matrix is its rank.

Definition A.9 (Rank of a matrix (Definition 8.3, Proposition 8.4 page 193-194 from [@schlesinger)**.**)] The rank of a matrix is the number of linearly independent rows/columns of the matrix. If the matrix belongs to the $\mathbb{R}^{n\times m}$ space, the rank is less or equal than $min(n, m)$. A matrix is said to be full rank if its rank is equal to $min(n, m)$.

The dimension of the null-space is the number of linearly-dependent columns. For a rank k matrix, the Moore-Penrose pseudo-inverse is defined as $\sum_i^k \frac{1}{\sigma_i} u_i v_i^T$. Another relevant property of SVD is that the nonzero singular values and the corresponding singular vectors are the nonzero eigenvalues and eigenvectors of the matrix $\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$:

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} u_i \\ v_i \end{pmatrix} . = s_i \begin{pmatrix} u_i \\ v_i \end{pmatrix}$$

With $s(A)$ or simply with s we denote the sparsity, that is, the maximum number of non-zero elements of the rows.

A.2.3 Singular vectors for data representation

Singular values and singular vectors provide important information about matrices and allow to speed up certain kind of calculations. Many data analysis algorithms, such as Principal Component Analysis, Correspondence Analysis, and Latent Semantic Analysis that will be further investigated in the following sections, are based on the singular value decomposition of a matrix.

To begin with, the SVD representation of a matrix allows us to better understand some matrix norms, like the spectral norm and the Frobenius norm (Strang, 2016).

Definition A.10 (l2 (or Spectral) norm)**.** Let $A \in \mathbb{R}^{n\times m}$ be a matrix. The l_2 norm of A is defined as $\|A\|_2 = max_x \frac{\|Ax\|}{\|x\|}$.

It is pretty easy to see that $\|A\|_2 = \sigma_{max}$, where σ_{max} is the greatest singular value of A. In particular, if we consider again the matrix $A = U\Sigma V^T$ as a linear transformation, we see that U and V^T only rotate vectors $||Ux|| = ||x||$, $||Vx|| = ||x||$ while Σ changes their magnitude $||\Sigma x|| \leq \sigma_{max}||x||$. For this reason, the l_2 Norm of a matrix is also referred to as the Spectral Norm. During the rest of the work we will also use the notation $||A||$ to refer to the Spectral Norm.

Another important matrix norm that benefits from SVD is the Frobenius norm, defined in the following way.

Definition A.11 (Frobenius norm)**.** Let $A \in \mathbb{R}^{n \times m}$ be a matrix. The Frobenius norm of A is defined as $\|A\|_F = \sqrt{\sum_i^n \sum_j^m a_{ij}^2}$.

It can be shown that also this norm is related to the singular values. ::: {.proposition} The Frobenius norm of a matrix $A \in \mathbb{R}^{n \times m}$ is equal to the square root of the sum of squares of its singular values.

$$\|A\|_F = \sqrt{\sum_i^r \sigma_i^2}$$

:::

Proof.

$$||A||_F = \sqrt{\sum_i^n \sum_j^n a_{ij}^2} = \sqrt{Tr[AA^T]} = \sqrt{Tr[(U\Sigma V^T)(U\Sigma V)^T]} =$$

$$\sqrt{Tr[U\Sigma V^T V \Sigma U^T]} = \sqrt{Tr[U\Sigma\Sigma U^T]} = \sqrt{Tr[U\Sigma^2 U^T]} = \sqrt{\sum_{i=1}^n \sigma^2}$$

From the cyclic property of the trace $Tr[AB] = Tr[BA]$ it follows that $Tr[U\Sigma^2 U^T] = Tr[U^T U \Sigma^2] = Tr[\Sigma^2]$, which is the sum of the squares of the singular values $\sum_{i=1}^n \sigma^2$. □

Another interesting result about the SVD of a matrix is known as the Eckart–Young–Mirsky theorem. ::: {.theorem #eckart-young-mirsky name="Best F-Norm Low Rank Approximation"} Eckart and Young (1936)**?** Let $A \in \mathbb{R}^{n \times m}$ be a matrix of rank r and singular value decomposition $A = U\Sigma V^T$. The matrix $A^{(k)} = U^{(k)}\Sigma^{(k)}V^{(k)T}$ of rank $k \leq r$, obtained by zeroing the smallest $r-k$ singular values of A, is the best rank-k approximation of A. Equivalently, $A_k = argmin_{B:rank(B)=k}(\|A-B\|_F)$. Furthermore, $min_{B:rank(B)=k}(\|A-B\|_F) = \sqrt{\sum_{i=k+1}^r \sigma_i}$. :::

To get a clearer understanding of this result, we could rewrite $A = U\Sigma V^T = \sum_i^r \sigma_i u_i v_i^T$ and notice that matrix A is the sum of r matrices $u_i v_i^T$ each scaled by a scalar σ_i.

In practice, SVD decomposes matrix A in matrices of rank one, ordered by importance according to the magnitude of the singular values: the smaller the σ_i, the smaller is the contribution that the rank-1 matrix gives to the reconstruction of A. When the smallest singular values are set to 0, we still reconstruct a big part of the original matrix, and in practical cases, we will see that matrices can be approximated with a relatively small number of singular values.

Unfortunately though, calculating the singular vectors and singular values of a matrix is a computationally intensive task. Indeed, for a matrix $A \in \mathbb{R}^{n \times m}$

the cost of the exact SVD is $O\left(min(n^2m, nm^2)\right)$. Recently, there have been developed approximate methods that compute the Eckart-Young-Mirsky approximations of matrices in time $O(knm)$, where k is the rank of the output matrix (Partridge and Calvo, 1997), or in times that scale super-linearly on the desired rank and one dimension of the input matrix **??**.

A.3 Useful theorems around linear algebra

- Gershgorin circle theorem
- Perron-Frobenius theorem
- Sherman-Morrison formula
-

A.4 Inequalities

From here.

Theorem A.6 (Bernoulli inequalities)**.**

- *Bernoulli inequality: for* $\forall n \in \mathbb{N}, x \geq -1$

  $$(1+x)^n \geq 1 + nx$$

  (Reader, check what happens if n *even or odd!)*

- *Generalized bernoully inequality:* $\forall r \in \mathbb{R} \geq 1$ *or* $r \leq 0$*, and* $x \geq -1$

  $$(1+x)^r \geq 1 + rx$$

  . For $0 \leq r \leq 1$*,*

  $$(1+x)^r \leq 1 + rx$$

- *Related inequality: for any* $x, r \in \mathbb{R}$*, and* $r > 0 >$*:*

  $$(1+x)^r \leq e^{rx}$$

Theorem A.7 (Jensen inequality)**.** *Let* X *be a random variable with* $\mathbb{E}|X| \leq \infty$ *and* $g : \mathbb{R} \to \mathbb{R}$ *a real continious convex function. Then*

$$g(\mathbb{E}[X]) \leq \mathbb{E}[g(X)].$$

A mnemonic trick to remember the difference between convex and concave. Conv*EX* ends with *EX*, as the word *EX*ponential, which is convex.

Theorem A.8 (Ljapunov's inequality)**.** *For any real* $0 \leq p \leq q$ *and any real random variable,* $\|X\|_p \leq \|X\|_q$*.*

Theorem A.9 (Hölder's inequality)**.** *Let $p, q > 1$ that satisfy $\frac{1}{p} + \frac{1}{q} = 1$. If $\|X\|_p \leq \infty$ and $\|X\|_q$ then*

$$\mathbb{E}[|XY|] \leq \|X\|_p \dot{\|} X\|_q.$$

Theorem A.10 (Minkowski's inequality)**.** *Let $p > 1$, $\|X\|_p \leq \infty$ and $\|Y\|_p \leq \infty$. Then:*

$$\|X + Y\|_p \leq \|X\|_p + \|Y\|_p.$$

A.5 Trigonometry

Always have in mind the following Taylor expansion:

Theorem A.11 (Taylor expansion of exponential function)**.**

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Note that this series is convergent for all x

From that, it is easy to derive that:

$$e^{\pm ix} = cos(x) \pm isin(x)$$

Theorem A.12 (Nunzio's version of Euler's identity)**.** *For $\tau = 2\pi$,*

$$e^{i\tau} = 1$$

Because of this, we can rewrite sin and cos as:

- $\cos(x) = \frac{e^{ix}+e^{-ix}}{2}$
- $\sin(x) = \frac{e^{ix}-e^{-ix}}{2i}$

Note that we can do a similar thing of A.11 for matrices. In this case, we *define* the exponential of a matrix via it's Taylor expansion:

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}$$

The matrix exponential has the following nice properties (Walter, 2018):

- $(e^A)^\dagger = e^{A^\dagger}$
- $e^{A \otimes I} = e^A \otimes I$
- if $[A, B] = 0$, then $e^A e^B = e^{A+B}$.
- $Ue^AU^\dagger = e^{UAU\dagger}$
- $det(e^A) = e^{Tr[A]}$

A.5.0.1 Useful equalities in trigonometry

- $\sin(a)\cos(b) = \frac{1}{2}(\sin(a+b)+\sin(a-b)) = 1/2(\sin(a+b)-\sin(b-a))$
- $2\cos x\cos y = \cos(x+y)+\cos(x-y)$

Exercise A.3. Derive an expression for $\cos(a+b)$.

Proof. Recall that $e^{ix} = cos(x) + isin(x)$,

$$e^{i(A+B)} = cos(a+b)+isin(a+b) = e^{iA}e^{iB} = (cos(a)+isin(a))(cos(b)+isin(b))$$

$$cos(a+b)+isin(a+b) = cos(a)cos(b)+cos(a)+isin(b)+isin(a)cos(b)-sin(b)sin(a)$$

$$cos(a+b)+isin(a+b) = cos(a)cos(b)+cos(a)isin(b)+isin(a)cos(b)-sin(b)sin(a)$$

From this, it follows □

Appendix B

Series

Appendix C

Probability

C.1 Measure theory

Definition C.1 (Sigma algebra)**.** Let Ω be a set, and Σ be a subset of the power set of Ω (or equivalently a collection of subsets of X). Then Σ is a σ-algebra if:

- $\emptyset \in \Sigma$,
- Σ is closed under countable union,
- $\forall S \in \Sigma, \overline{S} \in \Sigma$.

Observe that thanks to de Morgan's law, we can equivalently define the sigma algebra to be closed under countable intersection. Oftentimes, it's common to conflate Σ and (Ω, Σ), and call both σ-algebra.

Definition C.2 (Measurable space)**.** Let Ω be a set, and Σ a σ-algebra. The tuple (Ω, Σ) is a measurable space (or Borel space).

Definition C.3 (Measurable function)**.** Let (Ω, Σ), and (Y, T) two different measurable space. A function $f : \Omega \mapsto Y$ is said to be measurable if for every $E \in T$:

$$f^{-1}(E) := \{x \in \Omega | f(x) \in E\} \in \Sigma$$

A measurable function is a function between the underlying sets of two measurable spaces that preserves the structure of the spaces: the preimage of any measurable set is measurable. This is in direct analogy to the definition that a continuous function between topological spaces preserves the topological structure: the preimage of any open set is open.

Definition C.4 (Continious function)**.** Let $(X, \mathbb{X}), (Y, \mathbb{Y})$ two topological spaces. Let f be a function between two topological spaces $f : X \mapsto Y$ is said to be continious if the inverse image of every open subset of Y is open in X. In other words, if $V \in \mathbb{Y}$, then its inverse image $f^{-1}(V) \in \mathbb{X}$

Definition C.5 (Measure space)**.** The tuple $(\Omega, \Sigma, \mathbb{P})$ is a measure space if:

- (Ω, Σ) is a measurable space.
- $\mu(E)$ is a measure on (Ω, Σ):
  - $\mu : \Sigma \mapsto \mathbb{R} + \{-\infty, +\infty\}$
  - non-negativity: $\mu(E) \geq 0 \forall E \in \Sigma$
  - Null empty set $\mu(\emptyset) = 0$
  - Coutable additivity (or σ-additivity): for all countable collections $\{E_k\}_{k=1}^{\infty}$ of pariwise disjoint sets in Σ,

$$\mu\left(\cup_{k=1}^{\infty} E_k\right) = \sum_{k=1}^{\infty} \mu(E_k)$$

Definition C.6 (Probability space)**.** The tuple $(\Omega, \Sigma, \mathbb{P})$ is a probability space if:

- (Ω, Σ) is a σ-algebra (Ω is the set of *outcomes* of the experiment, and Σ is the set of *events*)
- $\mathbb{P}$ is a measurable function:
  - $\mathbb{P} : \Sigma \mapsto [0, 1]$
  - Null empty set.
  - μ is countably additive.
- $\mathbb{P}(\Omega) = 1$

I.e. a probability space is a measure space where the measurable function on Ω is 1.

Definition C.7 (Complete probability space)**.** For $B \subset \Sigma$ s.t. $\mathbb{P}(B) = 0$, a $(\Omega, \Sigma, \mathbb{P})$ probability space is complete if $\forall A \subset B$, $A \in \Sigma$.

Definition C.8 (Equivalence between probability measures)**.** Let $(\Omega, \Sigma, \mathbb{P})$, $(\Omega, \Sigma, \mathbb{Q})$ two probability space with the same Ω and Σ. We say that $\mathbb{P}$ and $\mathbb{Q}$ are equivalent iif for every $A \in \Sigma$, $\mathbb{P}(A) = 0 \Leftrightarrow \mathbb{Q}(A) = 0$.

It basically means that the two measures agree on the possible and impossible events (even if it is pretty strange to call them equivalent).

Definition C.9 (Random variable)**.** A (real-valued) random variable on a probability space $(\Omega, \Sigma, \mathbb{P})$ is a measurable function $X : \Omega \mapsto \mathbb{R}$.

Remember that, for a list of numbers x_1, x_n,

- The mode can be defined as $\arg\min_x \sum_i |x_i - x|^0$
- The median can be defined as $\arg\min_x \sum_i |x_i - x|^1$
- The mean can be defined as $\arg\min_x \sum_i |x_i - x|^2$.

C.1.0.1 Union bound

The union bound is used to show that the probability of union (i.e. at least one of them happens) of a finite or countable set of events is less than or equal to the sum of the probabilities of the events.

Theorem C.1 (Union bound)**.** $\forall$ *events* $A_1 \dots A_n \in \Sigma$*:*

$$P(\cup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} P(A_i)$$

Exercise C.1. In Erdős–Rényi graphs $G(n, p)$, (that is, a graph with n nodes with probability p that each of the two nodes are connected). We define the event B_n as the event where a graph $G(n, p)$ has at least one isolated node. Show that $P(B_n) \leq n(1-p)^{n-1}$.

Proof. Let $A_i, i \in [n]$ the event that node i is isoldated. Its probability, from the definition of $G(n, p)$ is $(1-p)^{n-1}$, because there might be an edge with probability p with other $n-1$ nods. From this, applying directly the union bonund we obtain an upper bound on the probability that there is at least one isoldated node is in the graph:

$$P(B_n) = P(\cup_{i=1}^{n} A_i) \leq \sum_{i} P(A_i) \leq nP(A_i) = n(1-p)^{n-1}$$

□

Exercise C.2. Suppose we run 4 times a randomized algorithm, with success probability $1-\delta$. Can you bound the probability that we never fail using the union bound?

Proof. Let f_i the event that we fail running our algorithm at time i. We know that the failure probability f_i is δ for all $i \in [4]$. Thanks to the union bound we can bound the probability that we fail at least once: $P(\cup_i^k f_i) \leq \sum_i^4 \delta = 4\delta$. It follows that the have 4 success in a row is *lower* bounded by $1 - 4\delta$.

Note that we could have also bypassed the union bound and compute this quantity analitically, as the probability of getting 4 success in a row would be $(1-\delta)^4$, which we can compute with the binomial theorem A.2. □

Definition C.10 (Variance)**.**

$$\begin{aligned} \mathrm{Var}(X) &= \mathrm{E}\left[(X - \mathrm{E}[X])^2\right] && \text{(C.1)} \\ &= \mathrm{E}\left[X^2 - 2X\,\mathrm{E}[X] + \mathrm{E}[X]^2\right] && \text{(C.2)} \\ &= \mathrm{E}\left[X^2\right] - 2\,\mathrm{E}[X]\,\mathrm{E}[X] + \mathrm{E}[X]^2 && \text{(C.3)} \\ &= \mathrm{E}\left[X^2\right] - \mathrm{E}[X]^2 && \text{(C.4)} \end{aligned}$$

Exercise C.3. How can we express the variance as expectation of quantum states? What quantum algorithm might we run to estimate the variance of a random variable M?

$$\langle\psi|M^2|\psi\rangle - (\langle\psi|M|\psi\rangle)^2$$

Discuss.

Definition C.11 (Exponential Family [@murphy2012machine)**.**]

A probability density function or probability mass function $p(v|\nu)$ for $v = (v_1, \cdots, v_m) \in \mathcal{V}^m$, where $\mathcal{V} \subseteq \mathbb{R}$, is a σ-algebra over a set X $\nu \in \mathbb{R}^p$ is said to be in the exponential family if it can be written as:

$$p(v|\nu) := h(v) \exp\{o(\nu)^T T(v) - A(\nu)\}$$

where:

- $\nu \in \mathbb{R}^p$ is called the *canonical or natural* parameter of the family,
- $o(\nu)$ is a function of ν (which often is just the identity function),
- $T(v)$ is the vector of sufficient statistics: a function that holds all the information the data v holds with respect to the unknown parameters,
- $A(\nu)$ is the cumulant generating function, or log-partition function, which acts as a normalization factor,
- $h(v) > 0$ is the *base measure* which is a non-informative prior and de-facto is scaling constant.

C.1.0.2 Bias-variance tradeoff

Here is a nice reference to understand the bias-variance tradeoff

C.1.1 Boosting probabilities with "median lemma" (or powering lemma)

In this section we discuss the following, widely known result in CS. It's used not only in writing algorithms, but also in complexity theory.

Lemma C.1 (Powering lemma [@jerrum1986random)**.** *] Let $\mathcal{A}$ be a quantum or classical algorithm which aims to estimate some quantity μ, and whose output $\tilde{\mu}$ satisfies $|\mu - \tilde{\mu}| \leq \epsilon$ except with probability γ, for some fixed $\gamma \leq 1/2$. Then, for any $\gamma > 0$ it suffices to repeat $\mathcal{A}$ $O(\log 1/\delta)$ times and take the median to obtain an estimate which is accurate to within ϵ with probability at least $1 - \delta$.*

C.2 Markov chains

Useful resources: here, here.

Definition C.12 (Markov chain (Serfozo, 2009))**.** Let $(X_t)_{t \in I}$ be a stochastic process defined over a probability space $(\Omega, \Sigma, \mathbb{P})$, for a countable set I, where X_t are random variables on a set $\mathcal{S}$ (called state space). Then $(X_t)_{t \in I}$ is a Markov chain if, for any $j \in \mathcal{S}$ and $t \geq 0$, it holds that

$$\mathbb{P}[X_{t+1} = j | X_0, X_1, \dots X_t] = \mathbb{P}[X_{t+1} = j | X_t]$$

and for all $j, i \in \mathcal{S}$, it holds that

$$\mathbb{P}[X_{t+1} = j | X_t = i] = p_{ij}$$

, where p_{ij} is the transition probability for the Markov chain to go from state i to state j.

Less formally, a Markov chain is a stochastic process with the Markov property, for which we can just use a matrix P to identify its transition probability. Most of the time, we will discretize the state space $\mathcal{S}$, so we can label elements of $\mathcal{S}$ with integers $i \in [|\mathcal{S}|]$. This fact will allow us to conflate the (push-forward) measure $\mathcal{P}$ on $\mathcal{S}$ and the matrix P.

A state j is said to be *accessible* from i (written as $i \mapsto j$) if $P^t_{ij} > 0$ for some t, where P^t is the t-th power of the transition matrix P. A communication class is an equivalence releation between states (relatively simple to prove) where two states j, i are said to communicate if they are mutually accessible.

Definition C.13 (Irreducible markov chain)**.** A Markov Chain $(X_t)_{t \in I}$ is irreducible if and only if

- there exist some integer $t \in I$ such that $p^t_{ij} > 0$ for all $i, j \in \mathcal{S}$ there exist some integer $t \in I$ such that $P[X_t = j | X_0 = i] > 0$, for all $i, j \in \mathcal{S}$
- there is only one communication class.

The previous conditions are equivalent.

In terms of random walks, irreducibility means that: if the graph is undirected, the graph is has only one connected component (i.e. is connected), and if the graph is directed, the graph is strongly connected.

C.3 Distributions

This is a beautiful guide that shows you how to draw samples from a probability distribution.

- Binomial distribution
-

C.4 Concentration inequalities

Take a look at this and this. Also recall that the Union bound was presented in the section dedicated for probability theory, i.e. Theorem C.1.

C.4.1 Markov inequality

The Markov inequality is an *upper bound for the probability that a non-negative function of a random variable*, that is greater than or equal to a positive constant. Especially in analysis, people refer to it as Chebyshev's inequality (sometimes, calling it the *first* Chebyshev inequality, while referring to the "usual" Chebyshev's inequality as the second Chebyshev inequality or Bienaymé–Chebyshev inequality).

Theorem C.2 (Markov inequality)**.** *For all non-negative random variable, and* $a > 0$*, we have that:*

- $Pr(X \geq a) \leq \frac{E[X]}{a}$
- $Pr(X \geq aE[X]) \leq \frac{1}{a}$

Proof. Observe that :

$$E[X] = P(X < a) \cdot E[X|X < a] + P(X > a) \cdot E[X|X > a]$$

As both of these expected values are bigger than zero, (using the nonnegativity hypothesis) we have that

$$E[X] \geq P(X > a)\dot{E}[X|X > a]$$

Now is easy to observe that $E[X|X > a]$ is at least a, and by rearranging we obtain that:

$$\frac{E[X]}{a} \geq P(X > a)$$

The second statement of the theorem follows from substitution, i.e. setting $b = aE[X]$ and using the previous statement on $Pr(X \geq b)$. □

A very useful corollary of Markov inequality is the following.

Theorem C.3 (Corollary of Markov inequality)**.** *Let* f *be a monotone increasing (or noll) function on a space* I*, and define the random variable on* Y*.*

$$P(Y \geq b) \leq \frac{E[f(Y)]}{f(b)}$$

C.4.2 Chebyshev inequality

This inequality tells us about the probability of finding the random variable X away from the mean $\mathbb{E}[X]$ is bounded by the variance of X.

Theorem C.4 (Chebyshev inequality)**.** *Let* X *be a random variable with finite mean* μ *and variance* σ*. For* $\epsilon > 0$*:*

$$Pr[|X - \mathbb{E}[X]| \geq \epsilon] \leq \frac{\sigma^2}{\epsilon^2}$$

Moreover, if $k = \epsilon/\sigma$ *we can replace* ϵ *with* $k\sigma$ *and obtain:*

$$Pr[|X - \mathbb{E}[X]| \geq k\sigma] \leq \frac{1}{k^2}$$

Proof. Observe that $(X-\mu)^2$ is a non-negative random variable. Therefore,

$$P(|X-\mu| \geq \epsilon) = P((X-\mu)^2 \geq \epsilon^2)$$

Now since $(X-\mu)^2$ is a non-negative random variable, we can apply Markov inequality to get :

$$P(|X-\mu| \geq \epsilon) = P((X-\mu)^2 \geq \epsilon^2) \leq \frac{E[(X-\mu)^2]}{\epsilon^2}$$

$$(|X-\mu| \geq \epsilon) = P((X-\mu)^2 \geq \epsilon^2) \leq \frac{[Var(X)]}{\epsilon^2}$$

□

It is very useful to see what happen when we define a new random variable Y as the sample mean of $X_1 \dots X_n$ other random variables (iid) indipendent and identically distributed: $Y = \frac{1}{n}\sum_i^n X_i$. The expected value of Y is the same as the expected value of X, but the variance is now linearly smaller in the number of samples:

$$E[Y] = \frac{1}{n}\sum_i^n E[X_i] = \mathbb{E}[X_i] \text{ for any } i$$

$$Var[Y] = \frac{1}{n^2}\sum_i^n \mathrm{Var}[X_i] = \frac{Var[X_i]}{n} \text{ for any } i$$

This allows us to obtain the following bound:

Theorem C.5 (Chebyshev inequality for sample mean)**.** *Let* $Y = \frac{1}{n}\sum_i^n X_i$. *Then,*

$$Pr[|Y - E[Y]| \geq \epsilon] \leq \frac{\sigma^2}{n\epsilon^2}$$

C.4.3 Weak Law of large numbers

Theorem C.6 (Weak Law of large numbers)**.** *Let* $X_1, X_2, \dots, X_n$ *be i.i.d random variables with a finite expected value* $\mathbb{E}[X_i] = \mu \leq \infty$, *and let* $\overline{X}$ *be the average* $\frac{1}{n}\sum_i^n X_i$. *Then, for any* $\epsilon > 0$, *we have that:*

$$\lim_{n\to+\infty} P\left(|\overline{X} - \mu| \geq \epsilon\right) = 0$$

Proof. We know that $E[\overline{X}] = \mu$ and $Var(\overline{X}) = \frac{\sigma^2}{n}$. By Chebyshev Inequality for the sample mean (Theorem C.5):

$$P(|\overline{X} - \mu| > \epsilon) \leq \frac{Var(\overline{X})}{\epsilon^2} = \frac{\sigma^2}{n\epsilon^2}$$

Trivially, $\lim_{n\to\infty} \frac{\sigma^2}{n\epsilon^2} = 0$, concluding the proof. □

C.4.4 Strong Law of Large Numbers

Theorem C.7 ((Strong) Law of large numbers)**.** *Let $X_1, X_2, X_3, \dots, X_n$ be i.i.d random variables with mean μ. Let $\overline{X} = \sum_{i=1}^n X_i$ be the sample mean. Then, $\overline{X}$ converges almost surely to μ:*

$$P(\lim_{x\to-\infty} \overline{X}_n = \mu) = 1$$

Remark. The SLLN implies WLLN but not vice-versa.

C.4.5 Chernoff bound

From here and here, here, and here

We focus on a restricted class of random variables, i.e. the case when our random variable is obtained as the sum of *indipendent* other random variables. Central limit theorem says that, as $n \to \infty$, the value $\frac{X-\mu}{\sigma}$ approaches the standard normal distribution $N(0, 1)$. Hoewver, it does not tell any information on the rate of convergence.

Theorem C.8 (Chernoff bound)**.** *Let $X = \sum_i^n X_i$ where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $(1 - p_i)$, and all X_i are independent. Let $\mu = E[X] = \sum_i^n p_i$. Then:*

- Upper tail*: $P(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2}{2+\delta}}\mu$ for all $\delta > 0$*
- Lower tail*: $P(X \leq (1-\delta)\mu) \leq e^{\mu\delta^2/2}$ for all $0 \leq \delta \leq 1$*

You can find a nice proof here.

Theorem C.9 (Chernoff bound)**.** *Suppose $X_1, \dots, X_t$ are independent random variables taking values in $\{0, 1\}$. Let $M_t = (X_1 + \dots X_t)/t$ denote their average value. Then for any $0 < \epsilon < 1$,*

- *(Multiplicative) $Pr[M_t - \mu \leq -\epsilon\mu] \leq \exp^{-\frac{t\mu\epsilon^2}{2}}$ and $Pr[M_t - \mu \geq \epsilon\mu] \leq \exp^{-\frac{t\mu\epsilon^2}{3}}$*
- *(Additive) $Pr[M_t - \mu \leq -\epsilon] \leq exp^{-2t\epsilon^2}$ and $Pr[M_t - \mu \geq \epsilon] \leq \exp^{-2t\epsilon^2}$*

Remark. Trick: if our random variables are not between 0 and 1, we can define $Y_i = X_i/max(X_i)$

C.4.6 Hoeffding inequality

Lemma C.2 (Hoeffding inequality)**.** *Let $X_1, \dots, X_k$ be independent random variables bounded by the interval $[a, b]$. Define the empirical mean of these variables by $\overline{X} = \frac{1}{k}(X_1 + \dots + X_k)$, then*

$$Pr(|\overline{X} - \mathbb{E}[X]| \le \epsilon) \ge 1 - 2\exp\left(-\frac{2k\epsilon^2}{b-a}\right).$$

Consequently, if $k \ge (b-a)\epsilon^{-2}\log(2/\eta)$, then $\overline{X}$ provides an ϵ-approximation of $\mathbb{E}[X]$ with probability at least $1-\eta$.

Exercise C.4. Suppose the number of red lights Alex encounters each day to work is on average 4.8 (according to historical trips to work). Alex really will be late if he encounters 8 or more red lights. Let X be the number of lights he gets on a given day.

1. Give a bound for $P(X \ge 8)$ using Markov's inequality.
2. Give a bound for $P(X \ge 8)$ using Chebyshev's inequality, if we also assume $Var(X) = 2.88$.
3. Give a bound for $P(X \ge 8)$ using the Chernoff bound. Assume that $X \sim Bin(12, 0.4)$ - that there are 12 traffic lights, and each is independently red with probability 0.4.
4. Compute $P(X \ge 8)$ exactly using the assumption from the previous part.
5. Compare the three bounds and their assumptions.

Proof. We apply all the inequalities learned in this section and discuss them at the end.

1. Since X is nonnegative and we know its expectation, we can apply Markov's inequality:

$$P(X \ge 8) \le \frac{\mathbb{E}[X]}{8} = \frac{4.8}{8} = 0.6$$

2. Since we know $X's$ variance, we can apply Chebyshevs inequality after some manipulation. We have to do this to match the form required:

$$P(X \ge 8) \le P(X \ge 8) + P(X \le 1.6) = P(|X - 4.8| \ge 3.2)$$

The reason we chose $X \le 1.6$ is so it looks like $P(|X - | \ge \alpha)$. Now, applying Chebyshev's gives:

$$\le \frac{Var(X)}{3.2^2} = \frac{2.88}{3.2^2} = 0.28125$$

3. Actually, $X \sim Bin(12, 0.4)$ also has $E[X] = np = 4.8$ and $Var(X) = np(1-p) = 2.88$ (what a coincidence). The Chernoff bound requires

something of the form $P(X \geq (1+\delta)\mu)$, so we first need to solve for $\delta : (1+\delta)4.8 = 8$ so that $\delta = 2/3$. Now,

$$P(X \geq 8) = P(X \geq (1+\frac{2}{3}).4.8) \leq exp(\frac{-\frac{2}{3}^2 4.8}{3}) \approx 0.4991$$

4. The exact probabiltity can be found summing the Binomial probability mass function:

$$P(X \geq 8) = \sum_{k=8}^{1} 2\binom{12}{k} 0.4^k 0.6^{(12-k)} \approx 0.0573$$

5. Usually the bounds are tighter as we move down the list Markov, Chebyshev, Chernoff. But in this case Chebyshev's gave us the tightest bound, even after being weakened by including some additional $P(X \leq 1.6)$. Chernoff bounds will typically be better for farther tails - 8 isn't considered too far from the mean 4.8. It's also important to note that we found out more information progressively - we can't blindly apply all these inequalities every time. We need to make sure the conditions are satisfied.

Remarkably, note that even our best bound of 0.28125 was $5-6$ times larger than the true probability of 0.0573. □

Appendix D

Error propagation and approximation

This part is based on many different sources, like (Hogan, 2006), (Ku et al., 1966). In the following, let A be the quantity that we want to estimate, and $\overline{A}$ our estimate. We have the definition of absolute error and relative error.

Definition D.1 (Absolute error)**.**

$$|A - \overline{A}| = \epsilon_{Abs}$$

Definition D.2 (Relative error)**.**

$$\frac{|A - \overline{A}|}{A} = \epsilon_R$$

or equivalently

$$A(1 - \epsilon_R) \leq \overline{A} \leq A(1 + \epsilon_R)$$

Thus observe that:

- If (and only if) $|A| < 1$, then, $\epsilon_{Abs} \leq \epsilon_R$
- If (and only if) $|A| > 1$, then, $\epsilon_{Abs} \geq \epsilon_R$

We will study the relation between the two errors, often leveraging the trick setting $\epsilon_{Abs} = \epsilon_R A$. Oftentimes, we would like to move from a relative to absolute precision, or vice versa.

D.0.0.0.1 From absolute to relative precision Suppose that we have an algorithm that in time $O(f(\frac{1}{\epsilon_{Abs}}))$ gives us $|A - \overline{A}| \leq \epsilon_{Abs}$ for $\epsilon_{Abs} \in (0, 1]$ and we want a relative error $\epsilon_R > 0$:

- If $|A| < 1$, then we want to obtain an error ϵ_{Abs} such that $\epsilon_{Abs} = \epsilon_R A$. For this, we need to have a lower bound λ^{-1} on A. If we have it, we can just set $\epsilon_{Abs} = \epsilon_R \lambda^{-1}$ and run our algorithm in time $O(f(\frac{\lambda}{\epsilon_{Abs}}))$
- If $|A| > 1$, If we want a relative error ϵ_R, then by running the algorithm with $\epsilon_{Abs} = \epsilon_R$ we have already a relative error bound, as $\frac{|A-\overline{A}|}{A} \leq |A - \overline{A}| \leq \epsilon_{Abs}$Note that we ϵ_{Abs} is meant to stay $\in (0, 1]$ as it wouldn't make sense to have a runtime of $O(\frac{1}{\epsilon_{abs}})$ for $\epsilon_{abs} > 1$.

Exercise D.1. Are there cases of algorithms with $\epsilon_{abs} > 1$? Does it make sense? Can you make examples?

D.0.0.0.2 From relative to absolute precision If we have an algorithm that in time $O(f(\frac{1}{\epsilon_R}))$ gives us $|A - \overline{A}| \leq A\epsilon_R$ and we want an absolute ϵ_{Abs}:

- IF $A \leq 1$, we could just call the algorithm with error $\epsilon_R = \epsilon_{Abs}$, and thus obtain
$$|A - \overline{A}| \leq \epsilon_R A \Rightarrow |A - \overline{A}| \leq \epsilon_R = \epsilon_{Abs},$$
as the absolute error is an upper bound of the relative error. Note that the runtime of the algorithm might (should!) depend on the quantity A that we want to estimate, so we could improve upon this, by trying to *not* pay a price that depends on A in the runtime.
- IF $A > 1$,
$$|A - \overline{A}| \leq \epsilon_R A$$
we want
$$|A - \overline{A}| \leq \epsilon_{Abs} \text{by setting } \epsilon_R = \frac{\epsilon_{Abs}}{\overline{A}}$$
By running algorithm $\mathcal{A}$ with error $\epsilon' = \frac{\epsilon}{A}$, i.e. we run it once with $\epsilon_R = 1/4$ error, and than. We run it again with the improved $\epsilon_R = \frac{1}{\lambda}$, and we have a runtime of $O(\text{f}(\frac{A}{\epsilon^{-1}}))$.

Example D.1. Amplitude estimation output a scalar $0 \leq \tilde{p} \leq 1$ which equal some probability p, such that $|p - \tilde{p}| \leq \epsilon p$ in time $O\left(\frac{1}{\epsilon p}\right)$. We have directly an absolute error of ϵ in this estimate (which we will rarely use, as oftenwe would like to multiply this estimate, so that the error scales proportionately).

–> –> –>

–> –> –> –> –> –>

D.0.1 Propagation of error in functions of one variable

Check this, this, this, and (Hogan, 2006).

–>

–> –> –> –> –> –> –> –> –>

–>

–>

–> –> –>

–>

–> –> –>

–>

–>

–>

–> –> –> –> –>

–> –> –> –>

–> –> –> –> –> –> –>

–>

–> –> –> –> –>

–> –>

–> –> –> –> –>

–> –> –>

–> –> –> –> –>

–>

–> –> –> –> –>

–>

–>

–> –> –>

Lemma D.1 ((Hamoudi et al., 2020))**.** *Let $\tilde{a}$ be an estimate of $a > 0$ such that $|\tilde{a} - a| \leq \epsilon_a a$. with $\epsilon_a \in (0, 1)$. Similarly, let $\tilde{b}$ be an estimate of $b > 0$ and $\epsilon_b \in (0, 1)$ such that $|\tilde{b} - b| \leq \epsilon_b b$. Then the ratio a/b is estimated to relative error $\left|\frac{\tilde{a}}{\tilde{b}} - \frac{a}{b}\right| \leq \left(\frac{\epsilon_a + \epsilon_b}{1 - \epsilon_b}\right) \frac{a}{b}$.*

The proof comes directly from their work ::: {.proof} Note that $b - \tilde{b} \leq |\tilde{b} - b| \leq \epsilon_b b$, so as we said before, deduce $\frac{1}{\tilde{b}} \leq \frac{1}{b(1-\epsilon_b)}$.

Now we can combine the previous observation:

$$\left|\frac{\tilde{a}}{\tilde{b}} - \frac{a}{b}\right| = \left|\frac{\tilde{a}b - a\tilde{b}}{\tilde{b}b}\right| = \left|\frac{\tilde{a}b - ab + ab - a\tilde{b}}{\tilde{b}b}\right| = \left|\frac{\tilde{a} - a}{\tilde{b}} + \frac{a}{\tilde{b}}\frac{b - \tilde{b}}{b}\right| \tag{D.1}$$

$$\leq \left|\frac{\tilde{a} - a}{\tilde{b}}\right| + \frac{a}{\tilde{b}}\left|\frac{b - \tilde{b}}{b}\right| \leq \frac{\epsilon_a a + \epsilon_b a}{\tilde{b}} \leq \frac{a}{b}\frac{\epsilon_a + \epsilon_b}{(1 - \epsilon_b)}. \tag{D.2}$$

:::

–>

–> –> –>

–> –> –>

D.1 Useful quantum subroutines and folklore results

We will often make use of a tool developed in (Wiebe et al., 2018). It is standard technique in classical computer science to boost the success probability of a randomized algorithm by repeating it and computing some statistics in the results. For the case of quantum algorithms, in high level, we take multiple copies of the output of the amplitude estimation procedure, compute the median, and reverse the circuit in order to get rid of the garbage.

Lemma D.2 (Median evaluation [@wiebe2018quantum)**.** *] Let $\mathcal{U}$ be a unitary operation that maps*

$$\mathcal{U} : |0^{\otimes n}\rangle \mapsto \sqrt{a}|x, 1\rangle + \sqrt{1-a}|G, 0\rangle$$

for some $1/2 < a \leq 1$ in time T. Then there exists a quantum algorithm that, for any $\Delta > 0$ and for any $1/2 < a_0 \leq a$, produces a state $|\Psi\rangle$ such that $\||\Psi\rangle - |0^{\otimes nL}\rangle|x\rangle\| \leq \sqrt{2\Delta}$ for some integer L, in time

$$2T\left\lceil\frac{\ln(1/\Delta)}{2\left(|a_0| - \frac{1}{2}\right)^2}\right\rceil.$$

We will report here some simple statements from literature which now are folklore.

Lemma D.3 ([@kerenidis2019qmeans)**.** *] Let ϵ_b be the error we commit in estimating $|c\rangle$ such that $\||c\rangle - |\overline{c}\rangle\| < \epsilon_b$, and ϵ_a the error we commit in the estimating the norms, $|\,\|c\| - \|\overline{c}\|\,| \leq \epsilon_a \|c\|$. Then $\|\overline{c} - c\| \leq \sqrt{\eta}(\epsilon_a + \epsilon_b)$.*

Lemma D.4 ([@kerenidis2017quantumsquares)**.** *] Let θ be the angle between vectors x, y, and assume that $\theta < \pi/2$. Then, $\|x - y\| \leq \epsilon$ implies $\||x\rangle - |y\rangle\| \leq \frac{\sqrt{2}\epsilon}{\|x\|}$. Where $|x\rangle$ and $|y\rangle$ are two unit vectors in ℓ_2 norm.*

Appendix E

Approximation theory

In this section we collect some polynomial approximation of useful functions that can be used in your quantum algorithms. As we learned in Section 5.4.3, in order to obtain efficient quantum algorithms we often need a low-degree polynomial approximation of certain functions. This is an interesting link between approximation theory (where it's relatively easy to obtain some lower bounds) and quantum algorithms. We will start by recalling an important tool in approximation theory.

Chebyshev polynomials play an important role in approximation theory (see, for example, (Iske, 2018)).

Definition E.1 (Chebyshev polynomial)**.** For $n \in \mathbb{N}_0$, the Chebyshev polynomial T_n of degree n is the function defined on the interval $[-1, 1]$ by

$$T_n(x) := \cos(n \arccos(x)).$$

E.1 Polynomial approximation of $\log(x)$

Lemma E.1 (Polynomial approximations of logarithm (Gilyén and Li, 2019))**.** *Let $\beta \in (0,1]$, $\epsilon \in (0,1/6]$. Then there exists a polynomial $\tilde{S}$ of degree $O(\frac{1}{\beta}\log(\frac{1}{\epsilon}))$ such that $|\tilde{S}(x) - \frac{\log_b(x)}{3\log_b(2/\beta)}| \leq \epsilon$ for all $x \in [\beta, 1]$ and base $b \in \mathbb{N}$, and for all $x \in [-1,1]$ $1/2 \leq \tilde{S}(x) = \tilde{S}(-x) \leq 1/2$.*

E.2 Polynomial approximation of $1/x$

We are going to approximate $1/x$ using Chebychev polynomial, with some additional tricks, which will be used to decrease the degree of the polynomial approximation. As

The function $1/x$ has an essential discontinuity in $x = 0$ since

$$\lim_{x\to 0-} \frac{1}{x} = -\infty \text{ and } \lim_{x\to 0+} \frac{1}{x} = +\infty.$$

In this section we will follow (Childs et al., 2017) and show that $1/x$ can be approximated arbitrarily closely on the set $[-1,\, -\delta] \cup [\delta,\, 1]$, where $0 < \delta < 1$, by a linear combination of Chebyshev polynomials. We start by approximating $1/x$ with the following function:

$$g_b(x) := \begin{cases} (1 - (1 - x^2)^b)/x, & \text{if } x \in \mathbb{R} \setminus \{0\} \\ 0, & \text{if } x = 0 \end{cases} \tag{E.1}$$

where $b > 0$ is a positive constant.

Lemma E.2. *The function* (E.1) *is continuous in* $\mathbb{R}$.

Proof. We need to show the continuity of g_b in $x = 0$. This follows by an application of L'Hôpital's rule:

$$\lim_{x\to 0} \frac{1 - (1 - x^2)^b}{x} = \lim_{x\to 0} \frac{2bx(1 - x^2)^{b-1}}{1} = 0 = g_b(0).$$

□

The following lemma shows that g_b approximates $1/x$ arbitrarily closely on the set $[-1,\, -\delta] \cup [\delta,\, 1]$, $0 < \delta < 1$, if b is large enough.

Lemma E.3. *Let* $\epsilon > 0$ *and* $0 < \delta < 1$. *If* $b \geq \max\{1, \delta^{-2}\log(1/(\epsilon\delta))\}$, *then* $|g_b(x) - 1/x| < \epsilon$ *for all* $x \in [-1,\, -\delta] \cup [\delta,\, 1]$.

Proof. From the inequality $e^a \geq 1 + a$, $a \in \mathbb{R}$, and $b > 1$ it follows with $a := -\delta^2$

$$(1 - \delta^2)^b \leq e^{-\delta^2 b}.$$

Therefore, for $x \in [-1,\, -\delta] \cup [\delta,\, 1]$ we have

$$\left| g_b(x) - \frac{1}{x} \right| = \frac{(1 - x^2)^b}{|x|} \leq \frac{(1 - \delta^2)^b}{\delta} \leq \frac{e^{-\delta^2 b}}{\delta} \leq \epsilon.$$

□

The following lemma shows that g_m, $m \in \mathbb{N}$, is on the interval $[-1,\, 1]$ equal to a linear combination of Chebyshev polynomials of degree at most $2m - 1$.

Lemma E.4. *Let* $m \in \mathbb{N}$ *be a positive integer. Then,*

$$g_m(x) = 4 \sum_{n=0}^{m-1} (-1)^n \left(\frac{\sum_{k=n+1}^{m} \binom{2m}{m+k}}{2^{2m}} \right) T_{2n+1}(x) \tag{E.2}$$

for all $x \in [-1,\, 1]$.

Proof. For $x = 0$ the equality (E.2) follows from the definitions of Chebyshev polynomials and g_m. We are left with the task to prove

$$\frac{1-(1-x^2)^m}{x} = 4 \sum_{n=0}^{m-1} (-1)^n \left(\frac{\sum_{k=n+1}^{m} \binom{2m}{m+k}}{2^{2m}} \right) T_{2n+1}(x)$$

for all $x \in [-1,\, 1] \setminus \{0\}$. Choose $\theta \in \mathbb{R}$ such that $x = \cos(\theta)$. Because $\sin^2(\theta) + \cos^2(\theta) = 1$ and $T_n(\cos(\theta)) = \cos(n\theta)$, we need to prove that

$$1 - \sin^{2m}(\theta) = 4 \sum_{n=0}^{m-1} (-1)^n \left(\frac{\sum_{k=n+1}^{m} \binom{2m}{m+k}}{2^{2m}} \right) \cos((2n+1)\theta)\cos(\theta), \tag{E.3}$$

where in the previous equation we moved $\cos(\theta)$ from the denominator of the l.h.s. to the r.h.s..

In order to complete the proof we can proceed in two different ways. Either we will write the left and right side of Equation (E.3) respectively in the form

$$\sum_{j=0}^{m} a_j \cos(2j\theta) \text{ and } \sum_{j=0}^{m} b_j \cos(2j\theta), \tag{E.4}$$

and, finally, verify that $a_j = b_j$ for all $j \in \{0,\, \dots,\, m\}$. This approach follows the original proof of (Childs et al., 2017). Another way is to just convert the l.h.s. of Equation (E.3) so it matches the coefficients of the r.h.s. We will start with the second approach, as it is the canonical one, but we discuss the second one later, as it is the one presented in the original proof.

Using the binomial formula for complex numbers $a,\, b \in \mathbb{C}$ and $p \in \mathbb{N}_0$, which is stated in Theorem A.6

$$(a+b)^p = \sum_{j=0}^{p} \binom{p}{j} a^{p-j} b^j,$$

we obtain a nice expansion of the $\sin(\theta)^{2m}$ as:

$$
\begin{aligned}
\sin^{2m}(\theta) &= \left(\frac{e^{i\theta} - e^{-i\theta}}{2i}\right)^{2m} \\
&= \frac{1}{2^{2m} i^{2m}} \sum_{j=0}^{2m} \binom{2m}{j} (-e^{-i\theta})^{2m-j} (e^{i\theta})^j \\
&= \frac{(-1)^m}{2^{2m}} \sum_{j=0}^{2m} \binom{2m}{j} (-1)^j e^{i(2j-2m)\theta} \\
&= \frac{1}{2^{2m}} \binom{2m}{m} + \frac{(-1)^m}{2^{2m}} \sum_{j=0}^{m-1} \left(\binom{2m}{j} (-1)^j e^{i(2j-2m)\theta} + \binom{2m}{2m-j} (-1)^{2m-j} e^{-i(2j-2m)\theta} \right) \\
&= \frac{1}{2^{2m}} \binom{2m}{m} + \frac{2}{2^{2m}} \sum_{j=0}^{m-1} \binom{2m}{j} (-1)^{m-j} \frac{e^{i(2j-2m)\theta} + e^{-i(2j-2m)\theta}}{2} \\
&= \frac{1}{2^{2m}} \binom{2m}{m} + \frac{2}{2^{2m}} \sum_{j=0}^{m-1} \binom{2m}{j} (-1)^{m-j} \cos(2(j-m)\theta).
\end{aligned}
$$

To give more context on some of the steps in the previous series of equations, we have done the following:

- we wrote the sin using the euler formula.
- we used the binomial theorem
- we used that $(-1)^j = (-1)^{2m-j}$ to factor out the minus sign
- we exploited the symmetry of the binomial, and removed the middle term $(\frac{1}{2^{2m}}\binom{2m}{m})$ from the summation.
- we collect back the $(-1)^m$ in the summation
- we used again Euler formula (multiplying and dividing everything by 2) to obtain a cos in the summation.

Thus, we have

$$
1 - \sin^{2m}(\theta) = \left(1 - \frac{1}{2^{2m}} \binom{2m}{m}\right) + \frac{2}{2^{2m}} \sum_{j=0}^{m-1} (-1)^{m-j-1} \binom{2m}{j} \cos(2(j-m)\theta). \tag{E.5}
$$

Note that we factored the $-$ sign in the third term in the exponent of the (-1) factor. Now we perform a substitution, and set $k = m - j$. Recall that cosine is an even function, and thus $\cos(2(j-m)\theta) = \cos(2(m-j)\theta) = \cos(2k\theta)$. Also recalling again the symmetry of the binomial coefficients, i.e. $\binom{2m}{j} = \binom{2m}{2m-j} = \binom{2m}{m+k}$, we can rewrite our function as:

$$
1 - \sin^{2m}(\theta) = 1 - \frac{1}{2^{2m}} \binom{2m}{m} + \frac{2}{2^{2m}} \sum_{k=1}^{m} \binom{2m}{m+k} (-1)^{k-1} \cos(2k\theta)\,. \tag{E.6}
$$

Note also that because of the substition step, now we have the index of k that goes from 1 to m. Now we have a first trick: $2^{2m} = (1+1)^{2m} = \sum_{i=0}^{2m} \binom{2m}{i}$.This allows to rewrite the first two terms of the summation as:

$$1-\frac{1}{2^{2m}}\binom{2m}{m} = \frac{2^{2m} - \binom{2m}{m}}{2^{2m}} = \frac{1}{2^{2m}}\sum_{k=1}^{m}\left(\binom{2m}{m+k} + \binom{2m}{m-k}\right) = \frac{2}{2^{2m}}\sum_{k=1}^{m}\binom{2m}{m+k}. \tag{E.7}$$

Where we used the fact that $\binom{2m}{m}$ is the central term in $\sum_{i=0}^{2m}\binom{2m}{i}$, and its symmetry again. Thus, the whole equation can be rewritten by factoring $\binom{2m}{m+k}$ as:

$$1 - \sin^{2m}(\theta) = \frac{2}{2^{2m}}\sum_{k=1}^{m}\binom{2m}{m+k} + \frac{2}{2^{2m}}\sum_{k=1}^{m}\binom{2m}{m+k}(-1)^{k-1}\cos(2k\theta) \tag{E.8}$$

$$=1 - \sin^{2m}(\theta) = \frac{2}{2^{2m}}\sum_{k=1}^{m}\binom{2m}{m+k} \times \left(1 + (-1)^{k-1}\cos(2k\theta)\right), \tag{E.9}$$

Now we rewrite the last factor as a telescoping sum as:

$$1 + (-1)^{k-1}\cos(2k\theta) = \sum_{n=0}^{k-1}(-1)^n\left(\cos(2\theta(n+1)) + \cos(2\theta n)\right). \tag{E.10}$$

Now, we want to obtain a product of cosines, as Equation (E.3), we use the standard formula (Section A.5.0.1) of $\cos(x+y) + \cos(x-y) = 2\cos(x)\cos(y)$. Specifically, we set $x + y = 2\theta(n+1)$ and $x - y = 2\theta n$, so we obtain the value of $x = (2n+1)\theta$ and $y = \theta$. Thanks to these two passages we can rewrite

$$1 - \sin^{2m}(\theta) = \frac{2}{2^{2m}}\sum_{k=1}^{m}\binom{2m}{m+k} \times \left(2\sum_{n=0}^{k-1}(-1)^n\cos((2n+1)\theta)\cos\theta\right). \tag{E.11}$$

We need one more step to obtain the coefficients of the Chebychev polynomial from the previous equation, which is a simple change in the order of the summations. To be very explicit, we are just using the following observation:

$$\sum_{k=1}^{m} f_k(\sum_{n=0}^{k-1} c_n) = \sum_{k=1}^{m}\sum_{n=0}^{k-1} f_k c_n = \sum_{n=1}^{m-1} c_n(\sum_{k=n+1}^{m} f_k).$$

Thus, we obtain

$$1-\sin^{2m}(\theta) = \frac{4}{2^{2m}} \sum_{n=0}^{m-1} ((-1)^n \left(\sum_{k=n+1}^{m} \binom{2m}{m+k} \right) \cos((2n+1)\theta) \cos(\theta)). \quad \text{(E.12)}$$

The proof ends here, but for completeness, we report the other approach here. We can start by comparing comparing the first equation of (E.4) with (E.5), we obtain the following formulas for a_j:

$$a_j = \begin{cases} 1 - \frac{1}{2^{2m}}\binom{2m}{m}, \text{ if } j = 0 \\ (-1)^{j-1}\frac{2}{2^{2m}}\binom{2m}{m-j}, \text{ if } j \in \{1, \dots, m\}. \end{cases}$$

Using the trigonometric identity $2\cos(\theta_1)\cos(\theta_2) = \cos(\theta_1 - \theta_2) + \cos(\theta_1 + \theta_2)$, the right side of (E.3) simplifies to

$$2\sum_{n=0}^{m-1} (-1)^n \left(\frac{\sum_{k=n+1}^{m} \binom{2m}{m+k}}{2^{2m}} \right) (\cos((2n+2)\theta) + \cos((2n)\theta)). \quad \text{(E.13)}$$

Comparing the second equation of (E.4) with (E.13), we obtain the following formulas for b_k:

$$\begin{aligned} b_0 &= \frac{2}{2^{2m}} \sum_{k=1}^{m} \binom{2m}{m+k} = \frac{1}{2^{2m}} \left(\sum_{k=1}^{m} \binom{2m}{m+k} + \sum_{k=1}^{m} \binom{2m}{m+k} \right) \\ &= \frac{1}{2^{2m}} \left(\sum_{k=1}^{m} \binom{2m}{m-k} + \sum_{k=m+1}^{2m} \binom{2m}{k} \right) \\ &= \frac{1}{2^{2m}} \left(\sum_{k=0}^{m-1} \binom{2m}{k} + \sum_{k=m+1}^{2m} \binom{2m}{k} \right) \\ &= \frac{1}{2^{2m}} \left(\sum_{k=0}^{2m} \binom{2m}{k} - \binom{2m}{m} \right) \\ &= 1 - \frac{1}{2^{2m}} \binom{2m}{m}, \end{aligned}$$

and, for $j \in \{1, \dots, m-1\}$,

$$\begin{aligned} b_j &= \frac{2}{2^{2m}}(-1)^{j-1} \sum_{l=j}^{m} \binom{2m}{m+l} + \frac{2}{2^{2m}}(-1)^{j} \sum_{l=j+1}^{m} \binom{2m}{m+l} \\ &= \frac{2}{2^{2m}}(-1)^{j-1} \binom{2m}{m+j} \\ &= \frac{2}{2^{2m}}(-1)^{j-1} \binom{2m}{m-j}, \end{aligned}$$

and, for $j = m$,

$$b_m = \frac{1}{2^{2m}}(-1)^{m-1}\binom{2m}{2m} = \frac{(-1)^{m-1}}{2^{2m}}.$$

Thus, $a_j = b_j$ for all $j \in \{0, \dots, m\}$. □

Although g_m is the sum of m Chebyshev polynomials, the following lemma shows that for m large enough the sum can be truncated while still remaining close to g_m.

Lemma E.5. *Let $0 < \epsilon < 1$ and $m \in \mathbb{N} \cap [2, \infty[$. Set $k_0 := \min\{m - 1, \lfloor\sqrt{3m \log(4(m-1)/\epsilon)}\rfloor\}$. Then,*

$$\left| g_m(x) - 4\sum_{n=0}^{k_0}(-1)^n \left(\frac{\sum_{k=n+1}^{m}\binom{2m}{m+k}}{2^{2m}}\right) T_{2n+1}(x)\right| \le \epsilon \tag{E.14}$$

for all $x \in [-1, 1]$.

Proof. If $k_0 = m - 1$ then we know from Lemma E.4 that the left side of Inequality (E.14) is 0. Next, assume $k_0 < m-1$. Consider a random variable X that follows the binomial distribution with parameters $2m$ and $1/2$. It is very simple to see that the expected value of this random variable is just m. The probability of getting at least $m + n + 1$, $n < m$, successes in $2m$ independent Bernoulli trials is given by

$$\Pr(X \ge m + n + 1) = \frac{1}{2^{2m}}\sum_{k=n+1}^{m}\binom{2m}{m+k}.$$

On the other hand, the Chernoff bound of Theorem C.8 gives

$$\Pr(X \ge m + k + 1) = \Pr\left(X \ge \left(1 + \frac{k+1}{m}\right)m\right) \le e^{\frac{-((k+1)/m)^2 m}{2+(k+1)/m}} \le e^{-\frac{k^2}{3m}}.$$

Therefore, we have

$$\begin{aligned}\left| g_m(x) - 4\sum_{n=0}^{k_0}(-1)^n \left(\frac{\sum_{k=n+1}^{m}\binom{2m}{m+k}}{2^{2m}}\right) T_{2n+1}(x)\right| &= \left|4\sum_{n=k_0+1}^{m-1}\left(\frac{\sum_{k=n+1}^{m}\binom{2m}{m+k}}{2^{2m}}\right) T_{2n+1}(x)\right| \\ &\le \left|4\sum_{n=k_0+1}^{m-1} e^{-\frac{k^2}{3m}} T_{2n+1}(x)\right| \\ &\le 4(m-1)e^{-\frac{k_0^2}{3m}} \\ &\le \epsilon,\end{aligned}$$

where we used the fact the $T_{2n+1}(x)| \le 1$ for $x \in [-1, 1]$. □

We are now ready to state the main result of this section that expresses the fact that the function $1/x$ can be approximated arbitrarily closely by linear combination of Chebyshev polynomials with a minimal number of terms.

Corollary E.1 (Low degree polynomial approximation of 1/x)**.** *Let* $0 < \epsilon < 1$ *and* $0 < \delta < 1$*. Set* $m := \lceil \delta^{-2} \log(2/(\epsilon\delta)) \rceil$ *and* $k_0 := 0$ *if* $m = 1$ *or* $k_0 := \min\{m-1, \lfloor \sqrt{3m \log(8(m-1)/\epsilon)} \rfloor\}$ *otherwise. Then,*

$$\left| \frac{1}{x} - 4 \sum_{n=0}^{k_0} (-1)^n \left(\frac{\sum_{k=n+1}^{m} \binom{2m}{m+k}}{2^{2m}} \right) T_{2n+1}(x) \right| \leq \epsilon \tag{E.15}$$

for all $x \in [-1, -\delta] \cup [\delta, 1]$.

Proof. This is an immediate consequence of Lemmas E.3, E.4, and E.5. □

Interestingly, there is another way of obtaining the decomposition of $\frac{1-(1-x^2)^{2m}}{x}$ with Chebychev polynomials, which is more akin to the canonical way one would approach the problem, so we report it here as an exercise, as it might be more mathematically challenging to solve than what we presented before.

Exercise E.1. Can you prove Lemma E.4 using the orthogonality of Chebychev polynomials?

Proof. We give just a hint for the proof.

$$\frac{1}{(1-x^2)^{2m}} x = \sum_{n=0}^{\infty} a_n T_n(x) \tag{E.16}$$

what we want is an explicit expression for a_n, so we exploit the orthogonality of Chebychev polynomials (which are orthogonal under the measure $\mu(x) = \frac{1}{\sqrt{1-x^2}}$ on the interval $[1, 1]$, i.e. we have:

$$\int_{-1}^{1} T_n(x)\, T_m(x) \, \frac{\mathrm{d}x}{\sqrt{1-x^2}} = \begin{cases} 0 & ifn \neq m, \\ \pi & ifn = m = 0, \\ \frac{\pi}{2} & ifn = m \neq 0. \end{cases} \tag{E.17}$$

Therefore, multiplying by a polynomial $T_l(x)$ and the measure $\mu(x)$ on both sides, we can read

$$\int_{-1}^{1} \frac{1-(1-x^2)^{2m}}{x} \frac{T_l(x)}{\sqrt{1-x^2}} = \int_{-1}^{1} \sum_{n=0}^{\infty} a_n \frac{T_n T_l}{\sqrt{1-x^2}} = \sum_{n=0}^{\infty} a_n \frac{\pi}{2} dx = \frac{\pi}{2} a_n. \tag{E.18}$$

The task now is to compute the integral on the l.h.s.. □

E.3 Polynomial approximation of other functions

Appendix F

Contributions and acknowledgements

Hello! I am Alessandro Luongo, and these are my first lecture notes in quantum algorithms! They spurred out from my old blog, back in 2016/2017. Then, they took a more concrete form out of my Ph.D. thesis (which I made at IRIF with the support of Atos, which I thank), and now are in this extended form with the hope to serve the future researchers in QML. While I strive to be as precise as the lecture notes of Ronald de Wolf and Andrew Childs, I know this work is still far from the level of quality I aspire to. If you want to give me any feedback, feel free to write me at "scinawa - at - luongo - dot - pro". Or contact me on Twitter.

Ciao, Ale.

Core team

This work has been made possible only thanks to the help of the team of the Open-Source project of quantumalgorithms.org:

- Armando 'ikiga1' Bellante

- Hue Jun Hao Alexander

Contributors

The contributors to the project are:

- Patrick Rebentrost
- Yassine Hamoudi
- Martin Plávala
- Trong Duong
- Filippo Miatto
- Jinge Bao
- Michele Vischi
- Samantha Buck
- Adrian Lee
- Ethan Hansen
- Lei Fan
- Giacomo De Leva
- Pablo Rotondo
- João Doriguello
- Avhijit_Nair
- Marco Caselli

Funding

This website is supported by:

- unitary.fund.
- Centre for Quantum Technologies

Suppliers

A big thanks to:

- 42LF for the legal support,
- Lorenzo Gecchelin for the graphics.

In sparse order, I would like to thank Dong Ping Zhang, Mehdi Mhalla , Simon Perdrix, Tommaso Fontana, and Nicola Vitucci for the initial help with the previous version of this project, and the helpful words of encouragement.

F.1 License and citation

The website quantumalgorithms.org by Alessandro Luongo is licensed under CC BY-NC-SA 4.0

```
@misc{quantumalgorithms,
title={Quantum algorithms for data analysis},
author={Luongo, Alessandro},
url={https://quantumalgorithms.org},
year={2020}
}
```

The CSS style file comes from here.

F.2 Cookie Policy

The website https://quantumalgorithms.org (the "Website") uses cookies for the following purposes: allowing online authentication, monitoring sessions and memorising information on specific configurations of users accessing the server. This document sets out detailed information on the use of cookies and similar technology, how they are used by the Website and how to manage them. Visitors are able to configure their own browsers so that they are alerted to the use of cookies or they may otherwise refuse them. Browser acceptance of cookies can be disabled by changing your settings.

F.2.0.1 DEFINITIONS

Cookies are small strings of text (letters or numbers) that allow a web server to memorise browser information that can be used during the same session (session cookies) or at a later stage, even days later (persistent cookies). Cookies are memorised in accordance with the user settings by the individual browser on the device being used (computer, tablet, smartphone).

F.2.0.2 TYPES OF COOKIES

There are different categories of cookies and each has its own characteristics and uses:

- Technical cookies: this type of cookie is essential for a website to function properly and they are only used to the extent required for the transmission of communications over an electronic communication network, or to the extent strictly necessary for the supplier of an information service explicitly requested by the subscriber or by the user to supply that service;

- Analytical cookies: this type of cookie is used to anonymously collect and analyse the traffic to and use of a website. Without identifying the user, they make it possible for example, to detect whether said user has subsequently accessed the website. They also make it possible to monitor the system and enhance the services and user experience. These cookies may be disabled without affecting the functioning of a website.
- Profiling cookies: these are persistent cookies used to identify (anonymously and not) user preferences and to enhance the navigation experience.
- Third-party cookies (analytical and/or profiling): these are generated by companies other than the actual website and integrated into a website's pages, for example Google widgets (such as Google Maps) or social plugins (Facebook, Twitter, LinkedIn, Google+, etc.). The management of information that is collected by a "third party" is regulated by the relevant privacy statement, which you are requested to read. For ease of reference, they are indicated in the links set out below.

F.2.0.3 TYPES OF COOKIES USED

The Website uses the following type of cookies:

- Third-party analytical cookies: Google Analytics, a web traffic analysis service provided by Google Inc. ("Google"), which makes it possible to access and analyse detailed statistics on website visitors. The Google Analytics service has been designed to use pre-anonymised data so as to conceal the last part of the visitor's IP address. For further information, please see https://www.google.it/policies/privacy/partners/. Users may disable Google Analytics by installing on their browser the opt-out add-on tool provided by Google (please see https://tools.google.com/dlpage/gaoptout)

F.2.0.4 DURATION

Some cookies (called session cookies) remain active until a user closes their browser. Other cookies (called persistent cookies) "survive" the closure of the browser and are available in subsequent user visits. Their duration is set by the server when they are created: in some cases, there is a set expiry date whereas in other cases their duration is unlimited. However, they may always be deleted using browser settings. The majority of the cookies we use are persistent and expire 2 years from the date when they are downloaded onto the Visitor's device.

F.2.0.5 MANAGEMENT

Visitors may accept or refuse cookies via their browser settings. Content may be accessed even if cookies are completely disabled and disabling "third–party technical" cookies will not prevent a visitor from using a website. It could however adversely impact the User's experience (insofar as it is not possible to memorise their data for future use). Settings can be changed for different websites and/or website applications. Moreover, the leading browsers allow users to change their settings depending on the type of cookie:

- Firefox: https://support.mozilla.org/it/kb/Gestione%20dei%20cookie
- Internet Explorer: https://support.microsoft.com/it-it/help/17442/windows-internet-explorer-delete-manage-cookies
- Chrome: https://support.google.com/chrome/answer/95647?hl=it
- Opera: http://help.opera.com/Windows/10.00/it/cookies.html
- Safari for Mac: https://support.apple.com/kb/PH21411?viewlocale=it_IT&locale=it_IT
- Safari for iOS: http://support.apple.com/kb/HT1677?viewlocale=it_IT

Third parties are hereby informed that the use of this policy, even partial, for other websites shall be subject to sanctions by the Italian Data Protection Authority. This page may be accessed via the link set out in the footer of all the website's pages, pursuant to Article 122(2) of Legislative Decree 196/2003 and the simplified process for privacy information and the acquisition of consent to the use of cookies published on the Italian Official Journal no. 126 of 3 June 2014 and the relevant register of measures 229 dated 8 May 2014.

Appendix G

References

Bibliography

Aaronson, S. and Rall, P. (2020). Quantum approximate counting, simplified. In *Symposium on Simplicity in Algorithms*, pages 24–32. SIAM.

Aharonov, D. and Touati, Y. (2018). Quantum circuit depth lower bounds for homological codes. *arXiv preprint arXiv:1810.03912*.

Ahmadi, H. and Chiang, C.-F. (2010). Quantum phase estimation with arbitrary constant-precision phase shift operators. *arXiv preprint arXiv:1012.4727*.

Allcock, J., Bao, J., Doriguello, J. F., Luongo, A., and Santha, M. (2023). Constant-depth circuits for uniformly controlled gates and boolean functions with application to quantum memory circuits.

Ambainis, A. (2002). Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767.

Ambainis, A. (2007). Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239.

Ambainis, A. (2012a). Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations 29th int. In *Symp. Theoretical Aspects of Computer Science (STACS 2012)*, volume 14, pages 636–47.

Ambainis, A. (2012b). Variable time amplitude amplification and quantum algorithms for linear algebra problems. In *STACS'12 (29th Symposium on Theoretical Aspects of Computer Science)*, volume 14, pages 636–647. LIPIcs.

Ambainis, A., Buhrman, H., Leijnse, K., Patro, S., and Speelman, F. (2022). Matching triangles and triangle collection: Hardness based on a weak quantum conjecture. *arXiv preprint arXiv:2207.11068*.

An, D. and Lin, L. (2022). Quantum linear system solver based on time-optimal adiabatic quantum computing and quantum approximate optimization algorithm. *ACM Transactions on Quantum Computing*, 3(2):1–28.

Andrew, C. (2017). Lecture notes on quantum algorithms.

Araujo, I. F., Park, D. K., Petruccione, F., and da Silva, A. J. (2021). A divide-and-conquer algorithm for quantum state preparation. *Scientific Reports*, 11(1):6329.

Arrazola, J. M., Delgado, A., Bardhan, B. R., and Lloyd, S. (2020). Quantum-inspired algorithms in practice. *Quantum*, 4:307.

Arthur, D. and Vassilvitskii, S. (2006). How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153. ACM.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

Arunachalam, S., Gheorghiu, V., Jochym-O'Connor, T., Mosca, M., and Srinivasan, P. V. (2015). On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 17(12):123010.

Babbush, R., Gidney, C., Berry, D. W., Wiebe, N., McClean, J., Paler, A., Fowler, A., and Neven, H. (2018). Encoding electronic spectra in quantum circuits with linear t complexity. *Physical Review X*, 8(4):041015.

Bausch, J. (2022). Fast black-box quantum state preparation. *Quantum*, 6:773.

Bausch, J., Subramanian, S., and Piddock, S. (2021). A quantum search decoder for natural language processing. *Quantum Machine Intelligence*, 3(1):1–24.

Beals, R., Brierley, S., Gray, O., Harrow, A. W., Kutin, S., Linden, N., Shepherd, D., and Stather, M. (2013). Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120686.

Bellante, A. and Zanero, S. (2022). Quantum matching pursuit: A quantum algorithm for sparse representations. *Physical Review A*, 105(2):022414.

Bergholm, V., Vartiainen, J. J., Möttönen, M., and Salomaa, M. M. (2005). Quantum circuits with uniformly controlled one-qubit gates. *Phys. Rev. A*, 71:052330.

Berkes, P. (2005). Pattern Recognition with Slow Feature Analysis. *Cognitive Sciences EPrint Archive (CogPrints)*, 4104.

Berkes, P. and Wiskott, L. (2005). Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of Vision*, 5(6).

Bernstein, D. J., Jeffery, S., Lange, T., and Meurer, A. (2013). Quantum algorithms for the subset-sum problem. In *International Workshop on Post-Quantum Cryptography*, pages 16–33. Springer.

Berry, D. W., Childs, A. M., Cleve, R., Kothari, R., and Somma, R. D. (2015a). Simulating hamiltonian dynamics with a truncated taylor series. *Physical review letters*, 114(9):090502.

Berry, D. W., Childs, A. M., and Kothari, R. (2015b). Hamiltonian simulation with nearly optimal dependence on all parameters. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 792–809. IEEE.

Biernacki, C., Celeux, G., and Govaert, G. (2003). Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate gaussian mixture models. *Computational Statistics & Data Analysis*, 41(3-4):561–575.

Blaschke, T. and Wiskott, L. (2004). Independent slow feature analysis and nonlinear blind source separation. In *International Conference on Independent Component Analysis and Signal Separation*, pages 742–749. Springer.

Blömer, J. and Bujna, K. (2013). Simple methods for initializing the EM algorithm for gaussian mixture models. *CoRR*.

Borga, M., Landelius, T., and Knutsson, H. (1997). *A unified approach to pca, pls, mlr and cca.* Linköping University, Department of Electrical Engineering.

Boyer, M., Brassard, G., Høyer, P., and Tapp, A. (1998). Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505.

Brassard, G., Dupuis, F., Gambs, S., and Tapp, A. (2011). An optimal quantum algorithm to approximate the mean and its application for approximating the median of a set of points over an arbitrary distance. *arXiv preprint arXiv:1106.4267*.

Brassard, G., Hoyer, P., Mosca, M., and Tapp, A. (2002). Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74.

Buhrman, H., Cleve, R., de Wolf, R., and Zalka, C. (1999). Bounds for small-error and zero-error quantum algorithms. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 358–368. IEEE.

Buhrman, H., Cleve, R., Watrous, J., and de Wolf, R. (2001a). Quantum fingerprinting. *Physical Review Letters*, 87(16):167902.

Buhrman, H., Folkertsma, M., Loff, B., and Neumann, N. M. (2023a). State preparation by shallow circuits using feed forward. *arXiv preprint arXiv:2307.14840*.

Buhrman, H., Folkertsma, M., Loff, B., and Neumann, N. M. P. (2023b). State preparation by shallow circuits using feed forward. *arXiv preprint arXiv:2307.14840*.

Buhrman, H., Loff, B., Patro, S., and Speelman, F. (2022). Memory compression with quantum random-access gates. *arXiv preprint arXiv:2203.05599*.

Buhrman, H., Tromp, J., and Vitányi, P. (2001b). Time and space bounds for reversible simulation. In *International Colloquium on Automata, Languages, and Programming*, pages 1017–1027. Springer.

Cade, C. and Montanaro, A. (2017). The quantum complexity of computing schatten p-norms. *arXiv preprint arXiv:1706.09279*.

Camps, D., Lin, L., Van Beeumen, R., and Yang, C. (2024). Explicit quantum circuits for block encodings of certain sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 45(1):801–827.

Celeux, G. and Govaert, G. (1992). A classification EM algorithm for clustering and two stochastic versions. *Computational statistics & Data analysis*, 14(3):315–332.

Chakrabarti, S., Krishnakumar, R., Mazzola, G., Stamatopoulos, N., Woerner, S., and Zeng, W. J. (2021). A threshold for quantum advantage in derivative pricing. *Quantum*, 5:463.

Chakraborty, S., Gilyén, A., and Jeffery, S. (2019). The power of block-encoded matrix powers: Improved regression techniques via faster hamiltonian simulation. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Chakraborty, S., Morolia, A., and Peduri, A. (2022). Quantum regularized least squares. *arXiv preprint arXiv:2206.13143*.

Childs, A. M., Kothari, R., and Somma, R. D. (2015). Quantum linear systems algorithm with exponentially improved dependence on precision.

Childs, A. M., Kothari, R., and Somma, R. D. (2017). Quantum Algorithm for Systems of Linear Equations with Exponentially Improved Dependence on Precision. *SIAM Journal on Computing*, 46(6):1920–1950.

Childs, A. M. and Wiebe, N. (2012). Hamiltonian simulation using linear combinations of unitary operations. *arXiv preprint arXiv:1202.5822*.

Church, K. W. and Gale, W. A. (1995). Poisson mixtures. *Natural Language Engineering*, 1(2):163–190.

Cong, I. and Duan, L. (2015). Quantum discriminant analysis for dimensionality reduction and classification. *arXiv preprint arXiv:1510.00113*.

Costa, P., An, D., Sanders, Y. R., Su, Y., Babbush, R., and Berry, D. W. (2021). Optimal scaling quantum linear systems solver via discrete adiabatic theorem. *arXiv preprint arXiv:2111.08152*.

Cuccaro, S. A., Draper, T. G., Kutin, S. A., and Moulton, D. P. (2004). A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*.

De Bie, T., Cristianini, N., and Rosipal, R. (2005). Eigenproblems in pattern recognition. In *Handbook of Geometric Computing*, pages 129–167. Springer.

de Brugière, T. G. (2020). *Methods for optimizing the synthesis of quantum circuits*. PhD thesis, Université Paris-Saclay.

de Veras, T. M., da Silva, L. D., and da Silva, A. J. (2022). Double sparse quantum state preparation. *Quantum Information Processing*, 21(6):204.

de Wolf, R. (2019). Quantum computing: Lecture notes. *arXiv:1907.09415*.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

Dervovic, D., Herbster, M., Mountney, P., Severini, S., Usher, N., and Wossnig, L. (2018). Quantum linear systems algorithms: a primer. *arXiv preprint arXiv:1802.08227*.

Deutsch, D. and Jozsa, R. (1992). Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558.

Dexter, A. and Tanner, D. (1972). Packing densities of mixtures of spheres with log-normal size distributions. *Nature physical science*, 238(80):31.

Di Matteo, O., Gheorghiu, V., and Mosca, M. (2020). Fault-tolerant resource estimation of quantum random-access memories. *IEEE Transactions on Quantum Engineering*, 1:1–13.

Doriguello, J. F., Giapitzakis, G., Luongo, A., and Morolia, A. (2024). On the practicality of quantum sieving algorithms for the shortest vector problem. *arXiv preprint arXiv:2410.13759*.

Doriguello, J. F., Luongo, A., Bao, J., Rebentrost, P., and Santha, M. (2022). Quantum algorithm for stochastic optimal stopping problems with applications in finance. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Dörn, S. (2008). *Quantum complexity of graph and algebraic problems*. PhD thesis, Universität Ulm.

Drineas, P., Kerenidis, I., and Raghavan, P. (2002). Competitive recommendation systems. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 82–90. ACM.

Duan, B., Yuan, J., Yu, C.-H., Huang, J., and Hsieh, C.-Y. (2020). A survey on hhl algorithm: From theory to application in quantum machine learning. *Physics Letters A*, 384(24):126595.

Dürr, C., Heiligman, M., Høyer, P., and Mhalla, M. (2004). Quantum query complexity of some graph problems *.

Dürr, C., Heiligman, M., HOyer, P., and Mhalla, M. (2006). Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328.

Durr, C. and Hoyer, P. (1996). A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*.

Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218.

Escalante-B, A. N. and Wiskott, L. (2012). Slow feature analysis: Perspectives for technical applications of a versatile learning algorithm. *KI-Künstliche Intelligenz*, 26(4):341–348.

Ghitany, M., Maller, R. A., and Zhou, S. (1994). Exponential mixture models with long-term survivors and covariates. *Journal of multivariate Analysis*, 49(2):218–241.

Ghojogh, B., Karray, F., and Crowley, M. (2019). Eigenvalue and generalized eigenvalue problems: Tutorial. *arXiv preprint arXiv:1903.11240*.

Gidney, C. (2018). Halving the cost of quantum addition. *Quantum*, 2:74.

Gidney, C. and Ekerå, M. (2021). How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433.

Gilyén, A. and Li, T. (2019). Distributional property testing in a quantum world. *arXiv preprint arXiv:1902.00814*.

Gilyén, A., Su, Y., Low, G. H., and Wiebe, N. (2019). Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 193–204.

Giovannetti, V., Lloyd, S., and Maccone, L. (2008a). Architectures for a quantum random access memory. *Physical Review A*, 78(5):052310.

Giovannetti, V., Lloyd, S., and Maccone, L. (2008b). Quantum random access memory. *Physical review letters*, 100(16):160501.

Gleinig, N. and Hoefler, T. (2021). An efficient algorithm for sparse quantum state preparation. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 433–438. IEEE.

Gosset, D., Kothari, R., and Wu, K. (2024). Quantum state preparation with optimal t-count. *arXiv preprint arXiv:2411.04790*.

Greenacre, M. J. (1984). Theory and applications of correspondence analysis.

Gribling, S., Kerenidis, I., and Szilágyi, D. (2021). Improving quantum linear system solvers via a gradient descent perspective. *arXiv preprint arXiv:2109.04248*.

Grinko, D., Gacon, J., Zoufal, C., and Woerner, S. (2019). Iterative quantum amplitude estimation. *arXiv preprint arXiv:1912.05559*.

Grover, L. and Rudolph, T. (2002). Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint quant-ph/0208112*.

Grover, L. K. (2000). Synthesis of quantum superpositions by quantum computation. *Physical review letters*, 85(6):1334.

Grover, L. K. (2005). Fixed-point quantum search. *Physical Review Letters*, 95(15):150501.

Gu, X., Liu, C., and Wang, S. (2013). Supervised Slow Feature Analysis for Face Recognition. pages 178–184.

Gur, T., Hsieh, M.-H., and Subramanian, S. (2021). Sublinear quantum algorithms for estimating von neumann entropy. *arXiv preprint arXiv:2111.11139*.

Gyurik, C., Cade, C., and Dunjko, V. (2020). Towards quantum advantage for topological data analysis. *arXiv preprint arXiv:2005.02607*.

Hamoudi, Y. and Magniez, F. (2018). Quantum chebyshev's inequality and applications. *arXiv preprint arXiv:1807.06456*.

Hamoudi, Y., Ray, M., Rebentrost, P., Santha, M., Wang, X., and Yang, S. (2020). Quantum algorithms for hedging and the sparsitron. *arXiv preprint arXiv:2002.06003*.

Hann, C. T. (2021). *Practicality of Quantum Random Access Memory*. PhD thesis, Yale University.

Hann, C. T., Lee, G., Girvin, S., and Jiang, L. (2021). Resilience of quantum random access memory to generic noise. *PRX Quantum*, 2(2):020311.

Harrow, A. W., Hassidim, A., and Lloyd, S. (2009). Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*, 103(15):150502.

Harun-Ur-Rashid (2018). Research paper dataset.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*, volume 1 of *Springer Series in Statistics*. Springer New York, New York, NY.

Heinrich, S. (2002). Quantum summation with an application to integration. *Journal of Complexity*, 18(1):1–50.

Herbert, S. (2021). No quantum speedup with grover-rudolph state preparation for quantum monte carlo integration. *Physical Review E*, 103(6):063302.

Hogan, R. (2006). How to combine errors.

Holmes, A. and Matsuura, A. Y. (2020). Efficient quantum circuits for accurate state preparation of smooth, differentiable functions. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 169–179. IEEE.

Hsu, H., Salamatian, S., and Calmon, F. P. (2019). Correspondence analysis using neural networks. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2671–2680.

Huang, H.-Y., Bharti, K., and Rebentrost, P. (2019). Near-term quantum algorithms for linear systems of equations. *arXiv preprint arXiv:1909.07344*.

Iske, A. (2018). *Approximation Theory and Algorithms for Data Analysis*. Springer.

Jeffery, S. (2014). Frameworks for quantum algorithms.

Kalai, A. T., Moitra, A., and Valiant, G. (2012). Disentangling gaussians. *Communications of the ACM*, 55(2):113–120.

Kapoor, A., Wiebe, N., and Svore, K. (2016). Quantum perceptron models. In *Advances in Neural Information Processing Systems*, pages 3999–4007.

Kennedy, T. (2016). Chapter2: Basics of direct Monte Carlo.

Kerenidis, I., Landman, J., Luongo, A., and Prakash, A. (2019a). q-means: A quantum algorithm for unsupervised machine learning. In *Advances in Neural Information Processing Systems*, pages 4136–4146.

Kerenidis, I., Landman, J., and Prakash, A. (2019b). Quantum algorithms for deep convolutional neural networks. *arXiv preprint arXiv:1911.01117*.

Kerenidis, I. and Luongo, A. (2020). Classification of the mnist data set with quantum slow feature analysis. *Physical Review A*, 101(6):062327.

Kerenidis, I. and Prakash, A. (2017). Quantum recommendation systems. *Proceedings of the 8th Innovations in Theoretical Computer Science Conference*.

Kerenidis, I. and Prakash, A. (2018). A quantum interior point method for LPs and SDPs. *arXiv:1808.09266*.

Kerenidis, I. and Prakash, A. (2020). Quantum gradient descent for linear systems and least squares. *Physical Review A*, 101(2):022316.

Kitaev, A. Y. (1996). Quantum measurements and the abelian stabilizer problem. In *Electronic Colloq. on Computational Complexity*.

Krishnakumar, R., Soeken, M., Roetteler, M., and Zeng, W. (2022). Aq# implementation of a quantum lookup table for quantum arithmetic functions. In *2022 IEEE/ACM Third International Workshop on Quantum Computing Software (QCS)*, pages 75–82. IEEE.

Krizhevsky, A. et al. (2009). Learning multiple layers of features from tiny images.

Ku, H. H. et al. (1966). Notes on the use of propagation of error formulas. *Journal of Research of the National Bureau of Standards*, 70(4):263–273.

Kuperberg, G. (2011). Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *arXiv preprint arXiv:1112.3333*.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Lin, L. and Tong, Y. (2020). Optimal polynomial based quantum eigenstate filtering with application to solving quantum linear systems. *Quantum*, 4:361.

Litinski, D. and Nickerson, N. (2022). Active volume: An architecture for efficient fault-tolerant quantum computers with limited non-local connections. *arXiv preprint arXiv:2211.15465*.

Liu, C. and Rubin, D. B. (1995). ML estimation of the t distribution using EM and its extensions, ECM and ECME. *Statistica Sinica*, pages 19–39.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

Lloyd, S., Mohseni, M., and Rebentrost, P. (2013). Quantum principal component analysis. *Nature Physics*, 10(9):631–633.

Lloyd, S., Mohseni, M., and Rebentrost, P. (2014). Quantum principal component analysis. *Nature Physics*, 10(9):631.

Low, G. H. and Chuang, I. L. (2017). Hamiltonian simulation by uniform spectral amplification. *arXiv preprint arXiv:1707.05391*.

Low, G. H. and Chuang, I. L. (2019). Hamiltonian simulation by qubitization. *Quantum*, 3:163.

Low, G. H., Kliuchnikov, V., and Schaeffer, L. (2018). Trading t-gates for dirty qubits in state preparation and unitary synthesis. *arXiv preprint arXiv:1812.00954*.

Luongo, A., Miti, A. M., Narasimhachar, V., and Sireesh, A. (2024). Measurement-based uncomputation of quantum circuits for modular arithmetic. *arXiv preprint arXiv:2407.20167*.

Manara, M., Perotti, A., and Scapellato, R. (2007). *Geometria e algebra lineare*. Esculapio.

Markov (1890). On a question by d. i. mendeleev. *Zap. Imp. Akad. Nauk. St. Petersburg*.

Mathur, N., Landman, J., Li, Y. Y., Strahm, M., Kazdaghli, S., Prakash, A., and Kerenidis, I. (2022). Medical image classification via quantum neural networks.

McArdle, S., Gilyén, A., and Berta, M. (2022). Quantum state preparation without coherent arithmetic. *arXiv preprint arXiv:2210.14892*.

Metger, T. and Yuen, H. (2023). stateqip= statepspace. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1349–1356. IEEE.

Mitchell, T. M. et al. (1997). Machine learning.

Miyahara, H., Aihara, K., and Lechner, W. (2020). Quantum expectation-maximization algorithm. *Physical Review A*, 101(1):012326.

Moitra, A. (2018). *Algorithmic aspects of machine learning*. Cambridge University Press.

Montanaro, A. (2015). Quantum speedup of monte carlo methods. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2181):20150301.

Moosa, M., Watts, T. W., Chen, Y., Sarma, A., and McMahon, P. L. (2023). Linear-depth quantum circuits for loading fourier approximations of arbitrary functions. *Quantum Science and Technology*, 9(1):015002.

Mori, H., Mitarai, K., and Fujii, K. (2024). Efficient state preparation for multivariate monte carlo simulation. *arXiv preprint arXiv:2409.07336*.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nannicini, G. (2019). Fast quantum subroutines for the simplex method. *arXiv preprint arXiv:1910.10649*.

Ng, A. (2012). Cs229 lecture notes - machine learning. Lecture notes CS229 Stanford.

Nielsen, M. A. and Chuang, I. (2002). Quantum computation and quantum information.

O'Donnell, R. and Wright, J. (2016). Efficient quantum tomography. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 899–912.

Otterbach, J., Manenti, R., Alidoust, N., Bestwick, A., Block, M., Bloom, B., Caldwell, S., Didier, N., Fried, E. S., Hong, S., et al. (2017). Unsupervised machine learning on a hybrid quantum computer. *arXiv preprint arXiv:1712.05771*.

O'Donnell, R. (2015). Lecture 13: Lower bounds using the adversary method.

Partridge, M. and Calvo, R. (1997). Fast dimensionality reduction and simple pca. *Intelligent data analysis*, 2(3):292–298.

Paturi, R. (1992). On the degree of polynomials that approximate symmetric boolean functions (preliminary version). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 468–474, New York, NY, USA. Association for Computing Machinery.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Plesch, M. and Brukner, v. (2011). Quantum-state preparation with universal gate decompositions. *Phys. Rev. A*, 83:032302.

Prakash, A. (2014). *Quantum Algorithms for Linear Algebra and Machine Learning.* PhD thesis, EECS Department, University of California, Berkeley.

Rattew, A. G. and Koczor, B. (2022). Preparing arbitrary continuous functions in quantum registers with logarithmic complexity. *arXiv preprint arXiv:2205.00519*.

Rebentrost, P., Gupt, B., and Bromley, T. R. (2018). Quantum computational finance: Monte carlo pricing of financial derivatives. *Physical Review A*, 98(2):022321.

Rebentrost, P. and Lloyd, S. (2018). Quantum computational finance: quantum algorithm for portfolio optimization. *arXiv preprint arXiv:1811.03975*, 98(4):042308.

Rebentrost, P., Santha, M., and Yang, S. (2021). Quantum alphatron. *arXiv preprint arXiv:2108.11670*.

Rosenkranz, M., Brunner, E., Marin-Sanchez, G., Fitzpatrick, N., Dilkes, S., Tang, Y., Kikuchi, Y., and Benedetti, M. (2024). Quantum state preparation for multivariate functions. *arXiv preprint arXiv:2405.21058*.

Rosenthal, G. (2021). Query and depth upper bounds for quantum unitaries via Grover search. *arXiv preprint arXiv:2111.07992*.

Rosenthal, G. (2024). Efficient quantum state synthesis with one query. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2508–2534.

Rosenthal, G. and Yuen, H. (2021). Interactive proofs for synthesizing quantum states and unitaries. *arXiv preprint arXiv:2108.07192*.

Rudin, W. et al. (1964). *Principles of mathematical analysis*, volume 3. McGraw-hill New York.

Sanders, Y. R., Low, G. H., Scherer, A., and Berry, D. W. (2019). Black-box quantum state preparation without arithmetic. *Physical review letters*, 122(2):020502.

Schlesinger, E. (2011). *Algebra lineare e geometria.* Zanichelli.

Schmitt, B., Mozafari, F., Meuli, G., Riener, H., and De Micheli, G. (2021). From boolean functions to quantum circuits: A scalable quantum compilation flow in c++. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1044–1049. IEEE.

Schuld, M. and Petruccione, F. (2018). *Supervised learning with quantum computers*, volume 17. Springer.

Schuld, M. and Petruccione, F. (2021). *Machine Learning with Quantum Computers*. Quantum Science and Technology. Springer International Publishing.

Schuld, M., Sinayskiy, I., and Petruccione, F. (2015). An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185.

Serfozo, R. (2009). *Basics of applied stochastic processes*. Springer Science & Business Media.

Shende, V. V., Bullock, S. S., and Markov, I. L. (2006). Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010.

Shende, V. V., Markov, I. L., and Bullock, S. S. (2004). Minimal universal two-qubit controlled-not-based circuits. *Physical Review A—Atomic, Molecular, and Optical Physics*, 69(6):062321.

Soeken, M., Riener, H., Haaswijk, W., Testa, E., Schmitt, B., Meuli, G., Mozafari, F., and De Micheli, G. (2018). The epfl logic synthesis libraries. *arXiv preprint arXiv:1805.05121*.

Sprekeler, H. and Wiskott, L. (2008). Understanding Slow Feature Analysis: A Mathematical Framework. *Cognitive Sciences EPrint Archive (CogPrints)*, 6223.

Strang, G. (2016). *Introduction to linear algebra*. Wellesley - Cambridge Press.

Subaşı, Y., Somma, R. D., and Orsucci, D. (2019). Quantum algorithms for systems of linear equations inspired by adiabatic quantum computing. *Physical review letters*, 122(6):060504.

Subramanian, S., Brierley, S., and Jozsa, R. (2019). Implementing smooth functions of a hermitian matrix on a quantum computer. *Journal of Physics Communications*, 3(6):065002.

Sun, L., Jia, K., Chan, T.-H., Fang, Y., Wang, G., and Yan, S. (2014). Dl-sfa: deeply-learned slow feature analysis for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2632.

Sun, X., Tian, G., Yang, S., Yuan, P., and Zhang, S. (2023). Asymptotically optimal circuit depth for quantum state preparation and general unitary synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(10):3301–3314.

Ta-Shma, A. (2013). Inverting well conditioned matrices in quantum logspace. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 881–890.

Tang, E. (2018). Quantum-inspired classical algorithms for principal component analysis and supervised clustering. *arXiv preprint arXiv:1811.00414*.

van Apeldoorn, J., Gilyén, A., Gribling, S., and de Wolf, R. (2020). Quantum sdp-solvers: Better upper and lower bounds. *Quantum*, 4:230.

Walter, M. (2018). Symmetry and quantum information.

Wiebe, N., Kapoor, A., and Svore, K. M. (2018). Quantum nearest-neighbor algorithms for machine learning. *Quantum information and computation*, 15.

Wiskott, L., Berkes, P., Franzius, M., Sprekeler, H., and Wilbert, N. (2011). Slow feature analysis. *Scholarpedia*, 6(4):5282. revision #137965.

Wiskott Laurenz and Wiskott, L. (1999). Learning invariance manifolds. *Neurocomputing*, 26-27:925–932.

Wocjan, P., Chiang, C.-F., Nagaj, D., and Abeyesinghe, A. (2009). Quantum algorithm for approximating partition functions. *Physical Review A*, 80(2):022340.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint cs.LG/1708.07747*.

Yin, J. and Wang, J. (2014). A dirichlet multinomial mixture model-based approach for short text clustering. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 233–242. ACM.

Yoder, T. J., Low, G. H., and Chuang, I. L. (2014). Fixed-point quantum search with an optimal number of queries. *Physical review letters*, 113(21):210501.

Yu, C.-H., Gao, F., Lin, S., and Wang, J. (2019). Quantum data compression by principal component analysis. *Quantum Information Processing*, 18(8):249.

Yuan, P. and Zhang, S. (2023). Optimal (controlled) quantum state preparation and improved unitary synthesis by quantum circuits with any number of ancillary qubits. *Quantum*, 7:956.

Zhang, K., Hsieh, M.-H., Liu, L., and Tao, D. (2020). Efficient state read-out for quantum machine learning algorithms. *arXiv preprint arXiv:2004.06421*.

Zhang, X.-M., Li, T., and Yuan, X. (2022). Quantum state preparation with optimal circuit depth: Implementations and applications. *arXiv preprint arXiv:2201.11495*.

Zhang, X.-M., Yung, M.-H., and Yuan, X. (2021). Low-depth quantum state preparation. *Phys. Rev. Res.*, 3:043200.

Zhang, Z., Wang, Q., and Ying, M. (2024). Parallel quantum algorithm for hamiltonian simulation. *Quantum*, 8:1228.

Zhang Zhang and Dacheng Tao (2012). Slow Feature Analysis for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):436–450.

Zhao, L., Pérez-Delgado, C. A., and Fitzsimons, J. F. (2016). Fast graph operations in quantum computation. *Physical Review A*, 93(3).

Zhao, Z., Fitzsimons, J. K., Rebentrost, P., Dunjko, V., and Fitzsimons, J. F. (2021). Smooth input preparation for quantum and quantum-inspired machine learning. *Quantum Machine Intelligence*, 3(1):14.

Zhu, S., Sundaram, A., and Low, G. H. (2024). Unified architecture for a quantum lookup table. *arXiv preprint arXiv:2406.18030*.