Quantum algorithms for unsupervised machine learning and neural networks

Jonas Landman

Université de Paris

École Doctorale Sciences Mathématiques de Paris Centre

Institut de Recherche en Informatique Fondamentale (IRIF)

Quantum Algorithms for Unsupervised Machine Learning and Neural Networks

---

Par **Jonas Landman**

Thèse de doctorat de Mathématiques et Informatique

Dirigée par **Iordanis Kerenidis**

Présentée et soutenue publiquement le 22 Octobre 2021
Devant un jury composé de :

Dir. de recherche	Elham Kashefi	Sorbonne Université	Présidente du Jury
Dir. de recherche	Fréderic Magniez	Université de Paris	Examinateur
Professeur	Ashwin Nayak	University of Waterloo	Rapporteur
Professeur associé	Xiaodi Wu	University of Maryland	Rapporteur
Dir. de recherche	Iordanis Kerenidis	Université de Paris	Directeur de thèse

Abstract

Combining two great scientific revolutions of the XXth century, quantum computing uses the strange properties of quantum physics to redefine the notion of information processing to solve computational problems. From the initial proposal to the present day, much work has been done to develop quantum algorithms for these machines to push the limits of what was thought achievable in the fields of computational physics, chemistry, optimisation, communication, cryptography, and many more.

In this thesis, we investigate whether quantum algorithms can be used in the field of artificial intelligence, or machine learning. In the last decade, this field has been revolutionizing our ability to predict, classify, and learn, for many applications.

We will first recall the fundamentals of machine learning and quantum computing, and then describe more precisely how to link them through linear algebra. By encoding vectors in the form of quantum states, we will present and introduce quantum algorithms to efficiently solve tasks such as matrix product or distance estimation.

These results are then used to develop new quantum algorithms for unsupervised machine learning, such as k-means and spectral clustering. This allows us to define many fundamental procedures, in particular in vector and graph analysis. We will also present new quantum algorithms for artificial neural networks, or deep learning. For this we will introduce an algorithm to perform a quantum convolution product on images, as well as a new way to perform fast tomography on quantum states.

We prove that these quantum algorithms are faster compared to their classical version, but exhibit random effects due to the quantum nature of the computation. Many simulations have been carried out to study these effects and measure their learning accuracy on real data.

Finally, we will present a quantum orthogonal neural network circuit adapted to the currently available small and imperfect quantum computers. This allows us to perform real experiments to test our theory.

The quantum algorithms presented in this thesis give hope for the utility of an ideal quantum computer in the future. Indeed, we prove an asymptotic advantage for each algorithm in terms of complexity or running time, compared to the classical case. That being said, for this hope to become reality, many efforts remain to be realized in practice, from error correction to quantum data access.

Keywords: Quantum computing, quantum algorithms, quantum information, quantum computers, quantum machine learning, machine learning, unsupervised learning, clustering, neural networks, deep learning.

Résumé

Combinant deux grandes révolutions scientifiques du XX$^{\text{ème}}$ siècle, l'ordinateur quantique utilise les étranges propriétés de la physique quantique pour redéfinir la notion d'ordinateur afin de résoudre des problèmes calculatoires. Depuis l'imagination du concept d'ordinateur quantique jusqu'à nos jours, de nombreux travaux ont été réalisés pour développer des algorithmes quantiques afin de repousser les limites de ce que l'on pensait faisable par les ordinateurs dans les domaines de la physique, de la chimie, de l'optimisation, de la communication, de la cryptographie, et bien d'autres encore. Cette thèse questionne quant à elle la capacité théorique des ordinateurs quantiques dans le domaine des algorithmes d'apprentissage automatisé, ou *machine learning*.

Nous proposons de nouveaux algorithmes quantiques étant capable d'améliorer asymptotiquement la résolution de problèmes tels le *k-means* et le *spectral clustering*, tous deux visant à assembler de nombreuses données en groupe de similarité dans des espaces à haute dimension. D'autre part, nous présentons de nouveaux algorithmes quantiques pour les réseaux de neurones artificiels.

Plus globalement, ces travaux nous ont amené à créer les liens entre *machine learning* et information quantique à travers des résultats d'algèbre linéaire, en particulier l'encodage de données vectorielles sous forme d'états quantiques. Cela nous permet de créer des algorithmes quantiques permettant de résoudre rapidement des tâches telles que le produit matriciel ou l'estimation de distance entre vecteurs. Nous développons aussi un algorithme quantique pour réaliser un produit de convolution sur des images, ainsi qu'une nouvelle façon de réaliser une tomographie rapide sur les états quantiques.

Nous prouvons que ces algorithmes sont des équivalents plus rapides que leur version classique, et étudions les effets aléatoires induits par la nature quantique du calcul à l'aide de nombreuses simulations. Enfin, nous présentons un circuit quantique pour les réseaux de neurones orthogonaux, adapté aux ordinateurs quantiques petits et imparfaits actuellement disponibles. Cela nous permet de réaliser de vraies expériences afin de tester notre théorie.

Les algorithmes quantiques présentés dans cette thèse donnent espoir quant à l'utilité d'un ordinateur quantique idéal dans le futur. Ceci étant dit, pour que ces espoirs deviennent réalité, de nombreux efforts resteront à réaliser en pratique dans le domaine de la correction d'erreur et de l'accès aux données.

Mots clés : Informatique quantique, algorithmes quantiques, information quantique, ordinateurs quantiques, intelligence artificielle, apprentissage automatique, apprentissage non supervisé, réseaux neuronaux, apprentissage profond.

Détails

L'informatique quantique aborde la manipulation de l'information (bits dans l'état 0 ou 1) en y appliquant les lois de la physique quantique. Ceci faisant, nous autorisons de nouvelles possibilités et ouvrons des chemins jusqu'alors considérés impossibles. En particulier, la propriété de la superposition quantique stipule qu'un objet quantique peut être simultanément dans plusieurs états à la fois, préalablement à sa mesure. Si l'on manipule ainsi l'information, les *qubits* peuvent être dans l'état 0 et 1 à la fois. Ainsi, avec n qubits, ce sont 2^n états, ou combinaisons binaires, qui peuvent être en superposition. Ces états superposés ont chacun leur probabilité d'être observé une fois la mesure faite. Ces probabilités découlent de l'*amplitude quantique* associée à chaque état. Comme pour un ordinateur classique, nous devons appliquer des séquences logiques, appelés aussi circuits ou algorithmes quantiques, qui nous permettent de manipuler ces amplitudes, d'introduire ou de détruire des qubits, et d'obtenir un résultat. Le recherche des 30 dernières années a permis de trouver de nombreux algorithmes quantiques qui présentent des avantages considérables comparés aux solutions classiques équivalentes.

Les états quantiques peuvent être décrits à l'aide de l'algèbre linéaire, un système en superposition quantique étant considéré comme une combinaison linéaire d'états propres dans une certaine base, les seuls que l'ont peut observer lorsque l'on mesure le système. Un système quantique à n qubit est un vecteur dans un espace de Hilbert de dimension 2^n. À titre d'exemple, pour le chat de Schrödinger ces états seraient "vivant" et "mort", définissant un espace vectoriel bidimensionnel. Toute évolution d'un système quantique n'impliquant pas de mesure est ainsi décrite par un opérateur linéaire (unitaire), c'est à dire une matrice que l'on multiplie au vecteur du système.

Cette description algébrique nous est utile pour faire le lien avec la théorie de l'apprentissage automatisé ou *machine learning*. Ce domaine vise à trouver des méthodes algorithmiques pour qu'un ordinateur puisse apprendre à résoudre un problème donné: la classification d'image, la détection de maladies, la prédiction de phénomènes physiques ou sociaux, la complétion de données astronomiques, et bien d'autres. Ce sujet recouvre de nombreux champs théoriques et applicatifs, et la dernière décennie a prouvé l'efficacité incontournable de ces méthodes, en particulier concernant les réseaux de neurones artificiels. Le formalisme du *machine learning* emploie aussi l'utilisation de vecteurs et de matrices, nécessite de calculer des distances, des produits de convolution, d'extraire des vecteurs propres, et bien d'autres opérations communes en algébre linéaire.

Cette thèse aborde donc le lien entre ces deux champs afin de développer des algorithmes quantiques en *machine learning*. Après avoir rappelé les notions essentielles en informatique quantique et machine learning dans la Partie I, nous aborderons les premiers algorithmes quantiques d'algèbre linéaire dans la Partie II. Nous y verrons en détails les façons de convertir des données de la forme classique à la forme d'états quantiques, en introduisant les notions de mémoires quantiques et de tomorgraphie. Il existe plusieurs méthodes pour encoder un vecteur classique en superposition quantique, et nous décrirons l'*amplitude encoding*, le *bit encoding* et l'*unary encoding*, ainsi que leur liens et différences pratiques. En particulier, la méthode de l'*amplitude encoding* consiste à utiliser chaque composant d'un vecteur

en tant qu'amplitude de l'état quantique correspondant. Chaque partie de la superposition quantique correspond à un état de la *base computationelle*, soit une combinaison de 0 et de 1. Enfin, pour être conforme aux lois de probabilités, les vecteurs sont naturellements normalisés quand ils sont sous forme quantique. Il est clef de noter que pour un vecteur de dimension d, seul $\log_2(d)$ qubits sont nécessaires à l'encoder puisqu'à eux seuls ils définissent un espace de Hilbert à d dimensions, soit d amplitudes à atitrer.

À l'inverse, la tomographie correspond à retrouver le vecteur classique en extrayant chaque amplitude de la superposition quantique. Cette tâche semble difficile pour atteindre une précision idéale, c'est pourquoi nous utilisons des méthodes approximatives. En fonction du type et de la quantité d'erreur que nous autorisons, des résulats peuvent être inférés quant au nombre de fois où il faut mesurer l'état quantique pour en récupérer une description suffisante. Dans cette thèse nous présentons la tomographie ℓ_∞, qui correspond à un type d'erreur suivant la norme ℓ_∞, que nous justifions utile dans certains cas en *machine learning*, et qui nécessite substantiellement moins de mesures que la tomographie ℓ_2.

Nous présenterons aussi des méthodes quantiques pour calculer la distance ou le produit sclaire entre deux vecteurs classiques avec un avantage exponentiel par rapport à la dimension de ces vecteurs. Notre circuit quantique résulte en l'apparition du produit scalaire désiré dans l'amplitude d'un qubit en particulier selon qu'il est mesuré dans l'état 1 ou 0. Cela nous permet par la suite d'appliquer des algorithmes fondamentaux tels que l'*amplitude estimation* pour inscrire la valeur cherchée dans d'autres qubits.

Dans la Partie III, nous nous concentrerons sur le développement d'algorithmes quantiques spécifiques aux problèmes d'apprentissage non supervisé, pour lesquels la donnée vient sous forme de nuage de points non labelisés, et pour lesquels il faut détecter automatiquement des emsembles pouvant être assimilés à différentes classes. Ces problèmes, pour la pluparts considérés comme NP-complets, peuvent être résolus par des algorithmes approximatifs non déterministes. Le plus fondamental de ces algorithmes étant le k-means, nous proposons le q-means, une version quantique potentiellement exponentiellement plus rapide par rapport au nombre de points dans le jeu de données. Il s'agira d'utiliser le circuit de calcul de distance de la partie précédente et de l'appliquer en parallèle à tous les vecteurs, grâce à l'*amplitude encoding* de ces derniers. Grâce à une mesure quantique, il nous est alors possible de sélectionner les vecteurs les plus proches de certains points dits *centroids* et d'itérativement mettre à jour ces derniers. La complexité algorithmique du q-means dépend aussi de la dimenison des vecteurs, du nombre de *centroids*, ainsi que de certains paramètres non classiques qui dépendent de la distribution et de la valeurs des vecteurs.

Ensuite, nous généralisons cette méthode à l'algorithme de *spectral clsutering*, qui considère cette fois les points comme les noeuds d'un graphe dont il faut distinguer les sous-parties. Notre version quantique de cet algorithme utilise à nouveau le calcul de distances en superposition de toutes les paires de points du graphe. Nous développons aussi l'extraction de vecteurs propres de la matrice Laplacienne associée au graphe. Tout en conservant la cohérence quantique au long de l'algorithme, nous projettons les données dans l'espace défini par les vecteurs propres aux valeurs propres les plus petite. Dans cet espace, il est alors possible d'utiliser l'algorithme

q-means ci-dessus. Ce résultat est aussi analysé d'un point de vue de la complexité et nous prouvons un avantage comparé à la version classique.

Dans les deux cas, des simulations sont faites sur des jeux de données réels afin de tester l'impact de l'aléas quantique, ou de la faible précision dûe au nombre limité de qubits disponibles.

Dans la Partie IV, nous irons voir du côtés des réseaux de neurones artificiels. Ces architectures sont des algorithmes enchaînant produits matriciels et fonctions non linéaires. Les produits matriciels pouvant être vus comme une série de produit scalaire, nous avons déjà une méthode quantique afin de les accélérer. Cependant, il est complexe d'appliquer des transformations non linéaires sur un état quantique sans devoir en mesurer une partie. Dans cette thèse nous présentons le cas des réseaux de neurones convolutionnels, spécialisés dans le traitement du signal comme la classification d'image ou de signaux audio. Nous introduisons pour cela un des premiers algorithmes quantiques pour le produit de convolution ayant un avantage sur l'équivalent classique. Enfin nous utilisons une nouvelle méthode de tomographie d'états quantiques. Nous avons simulé notre algorithme sur un cas de classification d'images et avons obtenus des résultats comparables à l'équivalent classique.

Pour finir, nous changerons de paradygme pour étudier les algorithmes dits *Noisy Intermediate Scale Quantum* ou NISQ, c'est-à-dire destinés aux petits ordinateurs quantiques imparfaits actuellement disponibles. Nous proposons un circuit quantique ayant une forme pyramidale dont nous prouvons l'équivalence avec un réseau de neurone dit orthogonal. Ce circuit utilise cette fois-ci l'*unary encoding*. Il nous a été possible de tester notre circuit sur un ordinateur quantique à base de qubits supraconducteurs et d'obtenir des résultats satisfaisants malgré la présence de bruits. Ces circuits pyramidaux ont l'originalité d'utiliser un seul type de portes quantiques, et nécessitent une connectivité simple entre les qubits de la puce quantique. Une optimisation classique est faite en parallèle pour ajuster les portes quantiques en question, permettant un apprentissage via descente de gradient dans un espace alternatif comparé aux réseaux de neurones orthogonaux classiques.

Pour conclure, le développement de cette thèse nous a permis d'enrichir la panoplie d'outils et d'algorithmes pouvant être utilisé par les ordinateurs quantiques idéaux (long terme) mais aussi bruités (court terme). Nous espérons que ces outils pourront être adaptés à encore d'autres algorithmes dans le domaine du *machine learning* mais aussi ailleurs. De nombreux efforts sont encore à faire pour trouver des applications où l'ordinateur quantique pourra proposer un avantantage par rapport à un calculateur classique. Ces efforts seront à faire en parallèle de ceux qu'occupent les phycisiens et ingénieurs qui essayent de fabriquer de telles machines dont le but est de dompter l'état quantique de la matière afin de manipuler l'information d'une nouvelle façon. Nous attendons donc avec impatience l'arrivée des premiers ordinateurs quantiques à grande échelle pour tester nos théories et espérons qu'ils aideront la communauté scientifique à résoudre des problèmes pour l'intérêt général.

Remerciements

This Ph.D. was a long and wonderfully rewarding experience, half of which happened during the Covid-19 global pandemic. Despite this, I was able to travel, learn something new every day and meet many inspiring people.

I would like to thank Iordanis Kerenidis, my Ph.D. supervisor, who gave me this opportunity after we met at a NASA facility in California. I knew immediately that he would be an ideal supervisor and mentor, both scientifically and on a human level. His constant support and guidance over the years allowed me to go through this Ph.D. with confidence, determination, and joy. I look forward to working with Iordanis again and learn more from him.

I would like to thank the Jury, in particular Ashwin Nayak and Xiaodi Wu for kindly accepting to review and provide insights for this manuscript.

I have been fortunate to study, work, or just interact with great researchers who have truly inspired me, notably Frederic Magniez, Elham Kashefi, Sophie Laplante, Andre Chailloux, Ashwin Nayak, Ashley Montanaro, Seth Lloyd, Eleni Diamanti, Pascale Senellart, Philippe Grangier, Umesh Vazirani and John Preskill.

These years were enlightened by many new collaborations and friendships around the world, including Alessandro Luongo, Daniel Szilagyi, Anupam Prakash, Amine Cherrat, Sander Gribling, Alex Grillo, Yassine Hamoudi, and Yixin Shen from our research group at IRIF. All the LIP6-Edinburgh team including Niraj Kumar, Brian Coyle, Slimane Thabet, Constantin Dalyac and Pierre-Emmanuel Emeriau. And Natansh Mathur, Vincent Fortuin, Noah Berner, Juan Ignacio Adame, Federico Centrone, Alex Singh, Avinash Mocherla, Adam Bouland, Chris Cade, and Charles Hadfield. I would like to thank all members of PCQC and IRIF for their support, in particular Etienne Mallet and Eva Ryckelynck.

The quantum computing ecosystem has expended a lot since I started this Ph.D., and I am very glad to have met inspiring actors of the field such as Christophe Jurczak, Olivier Ezratty, Matt Johson, Loïc Henriet, Théau Peronnin, Matthieu Desjardins, Elvira Shishenina, and many more people that will shape the future of quantum computing in France and abroad.

À mes amis qui pourront enfin arrêter de me demander “c’est quoi ta thèse déjà ?”, merci pour tout. En particulier ceux qui m’ont convaincu de démarrer ce doctorat, Alexis Léautier et Pierre Fredenucci (2728 Grant St, Berkeley), Antoine Michon et Reda Agoumi (the Board), Ainsi que ceux dont les conseils sur le doc-

torat furent essentiels, Batiste Le Bars, Maxence Ernoult, Baptiste Louf et Geoffrey Negiar. Merci à tous les Pototunes, à mes trois citrons de Polycool (et La Crampe) qui rendent ma vie si musicale et riche en couleur, en particulier mon niéseux Léon Vidal pour nos discussions du midi sur les ondes graviationnelles et les synthétiseurs analogiques. Merci à Romain Palmieri, Rafael Cohen, Dorian Perron et à toute l'équipe Groover, je suis fier de ce qu'est devenu notre projet aujourd'hui. À ceux et celles que je n'ai pas mentionnés, la thèse fait déjà presque 200 pages donc je vous remercie par télépathie, vous vous reconnaitrez.

Je dois tant à ma famille, mon père qui m'a donné le goût de la science pour le bien commun, ma mère qui m'a sans cesse répété "*quand on cherche on trouve*" (en parlant de mes affaires perdues, mais ça marche aussi pour le reste), Julia, Gabriel et Miko le chat de Schrödinger. Je suis si fier de rejoindre enfin le clan des "Docteurs" Landman, même si je ne pourrai pas prescrire d'ordonnances. Une pensée pour mes grands parents qui m'accompagnent où qu'ils soient, Moshé qui m'a donné le goût du jeu d'échec, Maurice pour le goût de la musique, Adèle qui rigole toujours en m'appelant "*Zveinstein*" ou "*Knobel Price*", et Monique que je connais si bien sans l'avoir connue.

Enfin, Solène, merci et merci. Une thèse entière serait nécessaire pour exprimer à quel point la vie est belle avec toi.

Ci-dessous le ticket boisson de la NASA (Ames Research Center California), que j'avais sur moi le jour où j'ai su que je voulais faire une thèse en quantum machine learning.

Avant-Propos

"La science à tout moment recule les limites du merveilleux."

Guy De Maupassant
La Peur (1882)

Je me permets d'introduire cette thèse de *Doctor of Philosophy* (Ph.D.) par une touche de philosophie, une série de pensées qui m'ont accompagné durant ces années de Doctorat.

J'aime à croire que l'émerveillement scientifique qu'il arrive à certaines et certains de ressentir dans leur vie est en partie dû à notre relation avec les *limites*. Parmi les plus communes, celles qui hantent l'humanité depuis toujours, on peut citer "d'où vient-on ?", "qu'il y a-t-il au plus loin ?", ou bien "quelle est la plus petite chose ?". De ces questions enfantines a jailli l'exploration du réel, emmenant nos connaissances toujours plus loin. Je ne sais pas ce qui est le plus surprenant entre les nouveaux savoirs acquis ou le fait même que nous ayons pu les acquérir. D'ailleurs, les questions les plus élégantes sont souvent celles qui portent sur les limites du savoir lui-même: "pourra-t-on savoir un jour pourquoi l'univers existe ?", "puis-je me prononcer sur la conscience, étant moi-même conscient ?". D'un degré supérieur, ces limites sont la passion communes des philosophes, logiciens, mathématiciens et physiciens.

La *beauté* des mathématiques, c'est ce que je ressens devant la preuve d'une limite sur la connaissance elle-même, ou la démonstration habile de son absence. Les limites elles-mêmes sont devenues des objets mathématiques. D'ailleurs, une limite fait-elle partie de ce qu'elle délimite ? Qu'en est-il du Big Bang ? On voit que les limites ne sont pas des murs, mais bien des portes.

Si la quête des limites est l'archétype d'un *hybris*, elle n'est pas forcément à confondre avec le désir d'utiliser ou de contrôler. Chercher les limites, c'est avant tout une méthode pour interroger notre compréhension du monde. La théorie de la physique quantique n'a pas seulement mieux délimité l'infiniment petit mais en a redéfini la notion même. Le principe d'incertitude de Heisenberg en est un bon exemple: il ne s'agit pas, au contraire de ce que l'on entend souvent, d'une limite de ce que l'on peut *savoir* conjointement sur la position et la vitesse d'une particule, mais plutôt la découverte que ces notions ont un sens physique différent à cette échelle. Ainsi la quête de limite a modifié notre conception de la réalité.

Un sudoku à résoudre, l'équation du mouvement des planètes, la configuration d'une molécule. À partir de nombreux problèmes émergent des interrogations sim-

ilaires sur les limites du possible: Est-ce que ce problème a une solution ? Si oui, puis-je la trouver avec une feuille et un stylo ? Est-ce qu'un ordinateur peut la trouver ? Si oui, peut-il la trouver rapidement, disons avant que le Soleil n'ait englouti la Terre ? Plus généralement, y a-t-il une limite à ce qu'un ordinateur peut résoudre rapidement ? Est-ce qu'un ordinateur peut simuler une conscience, ou un univers ? Les limites qui s'appliqueraient à ces résultats nous informent-elles sur la nature de l'information dans l'univers lui-même ? Par exemple, si j'autorise mon ordinateur à manipuler l'information sous forme quantique, possède-t-il les mêmes limites ? Si non, pourquoi ?

Ainsi, des sciences de l'ordinateur ou *Computer Science*, de nombreuses questions fondamentales émergent et passionnent les chercheurs. Se demander quelles sont les limites de ce qui peut être résolu ou encore *résolu efficacement* permet de voir d'un autre angle de nombreuses questions philosophiques [Aar13]. Il en va de même avec l'intelligence artificielle ou *Machine Learning*. Se demander jusqu'à quel point une machine peut apprendre à différencier des images d'animaux, résoudre un problème de mécanique des fluides, ou détecter l'ironie dans un texte, c'est un chemin alternatif pour comprendre la nature de l'apprentissage, de la physique, ou du language.

Cette thèse, portant sur ce qu'il est théoriquement possible ou impossible de faire avec un ordinateur quantique dans le champ de l'intelligence artificielle, est d'une certaine façon motivée par ces considérations, ces émerveillements.

Enfin, il faut dire que sur le chemin de comprendre les limites se trouve assez fréquemment le désir de dépasser nos propres limites à travers la technique. Ces dernières années, j'ai pu assister en temps réel à l'émergence de l'intelligence artificielle et à l'apparition des premiers ordinateurs quantiques. Ces nouvelles technologies, part leur élégante universalité, sont ou seront probablement capables du meilleur comme du moins meilleur. J'espère que l'on fera attention à ne pas ériger le dépassement des limites en un principe supérieur aux principes naturels et humains. Car si nous sommes ici flottant dans l'espace, le principal est peut être simplement de comprendre et de prendre soin.

Jonas Landman
Juin 2021

Contents

Part I

Introduction

Chapter 1

Introduction

"A mathematician is a blind man in a dark room looking for a black hat which isn't there."

Charles Darwin

1.1 Context and Motivation

Quantum Physics

Quantum physics is often considered as the most wonderful intellectual adventure of modern science. As Einstein, Bohr, Schrödinger, Dirac and others taught us, this theory is a new paradigm to our comprehension of the world. Small objects behave differently, by following specific equations and having the ability to be in multiple states before we look at them. Even though it concerns the tiniest objects such as atoms, electrons, or photons, the consequences are indeed macroscopic: without it, we wouldn't understand the Cosmic Microwave Background or photosynthesis, and we wouldn't have atomic clocks, lasers, computers and smartphones. Since the 1980s and the experimental realization of *entangled* particles by Alain Aspect [ADR82], solving the Einstein-Podolsky-Rosen (EPR) paradox, we even started to manipulate information at a quantum level. We tend to forget it, but information is physical, and therefore it could also be *quantum physical*. This second quantum revolution, paved the way to counterintuitive applications such as quantum teleportation [BBC$^+$93]. At the same time emerged the idea that handling quantum systems could help us performing computation, as Richard Feynman's famous quote [Fey82] puts it:

> "Trying to find a computer simulation of physics seems to me to be an excellent program to follow out [...] the real use of it would be with quantum mechanics [...] Nature isn't classical dammit, and if you want to make a simulation of Nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy."

On year later, inspired by the work of Bennet and Fredkin, he added [Fey87]:

> "We can in principle make a computing device in which the numbers are represented by a row of atoms with each atom in either of the two

> states. That's our input. The Hamiltonian starts "Hamiltonianizing" the wave function [...] The ones move around, the zeros move around [...] Finally, along a particular bunch of atoms, ones and zeros [...] occur that represent the answer."

In addition, using quantum information processing to create a computer seemed to solve an impending problem faced by traditional classical computers: Moore's law. First stated in 1965, this empirical rule claims that the transistors, building blocks of computers and physical embodiment of bits (0's and 1's), will be twice smaller every 18 months. As transistors reach a size of few nanometers with few atoms per unit, quantum and thermodynamical effects will disturb their properties. Shrinking has its limits [Wal16] and experts expect the end of Moore's law in the current decade.

Quantum Computing

So, what is a quantum computer, and why it may surpass classical computers? First of all it is a computer. It manipulates quantum objects, such as photons, electrons, or ions, as bits of information (see Section 3.1 for mathematical formalism). Therefore the quantum bits or *qubits*, representing the 0's and 1's, would inherit a quantum nature. It allows them to be in both states a the same time or to be *entangled* with each other. But these properties are only available before any measurement is made, after which everything becomes classical again. In the meantime, the quantum computer would apply logical operations to the qubits so that the measurements would give the desired answer with less resource globally.

Intuitively, the key difference lies in the exponential superposition of binary inputs. Indeed, one classical bit can be either in state 0 or 1 as a transistor can be opened or closed, but a qubit can simultaneously be in a combination of both states, informally:

$$|0\rangle + |1\rangle \tag{1.1}$$

where the Dirac notation $|\cdot\rangle$ reminds us that the bit is a quantum system. Similarly, two qubits can be in four states simultaneously, informally:

$$|00\rangle + |01\rangle + |10\rangle + |11\rangle \tag{1.2}$$

It follows that n qubits can be in a superposition of 2^n states. Since 2^n classical bits would be necessary to encode the same amount of state. This gives the intuition of the exponential advantage quantum could offer, informally:

$$|0\cdots00\rangle + |0\cdots01\rangle + |0\cdots010\rangle + \cdots + |10\cdots0\rangle \tag{1.3}$$

Let's consider a simplified, informal, and intuitive reasoning. We are required to solve the task of checking if a given name is "Albert", among a random list of $N = 10^9$ names (a billion). The best thing a classical computer can do is to repetitively instantiate bits to be each name of the list, one by one, and check if the name is "Albert" until it finds it. The time to find the right name, called the *complexity* of the algorithm, would be on average $N/2$. We see that this algorithm

would be *linear* in N, and doubling N would double the time.

$$\begin{aligned} \text{Niels} &\mapsto 0 \\ \text{Marie} &\mapsto 0 \\ &\vdots \\ \text{Albert} &\mapsto 1 \\ &\vdots \\ \text{Erwin} &\mapsto 0 \end{aligned} \tag{1.4}$$

However, if a quantum computer could instantiate all names in superposition using only 30 qubits ($2^{30} \approx 10^9$), it would need to check only *once* if the quantum state is "Albert" and get a superposition of all answers:

$$\left.\begin{aligned} &|\text{Niels}\rangle \\ &+ |\text{Marie}\rangle \\ &\vdots \\ &+ |\text{Albert}\rangle \\ &\vdots \\ &+ |\text{Erwin}\rangle \end{aligned}\right\} \mapsto \left\{\begin{aligned} &|0\rangle \\ &+ |0\rangle \\ &\vdots \\ &+ |1\rangle \\ &\vdots \\ &+ |0\rangle \end{aligned}\right. \tag{1.5}$$

One computation instead of one billion seems astonishing. Note however that the output in Eq.1.5 is still in a quantum state before any measurement. Therefore the answer is not directly accessible to us, classical beings. One would have to measure and therefore destroy this state to get only one of the output, most probably a $|0\rangle$. In fact, the effective method is called Grover's algorithm [Gro96] and is quite different. It has the benefit of a complexity of $\sim \sqrt{N}$ instead of $\sim N$.

In fact, during the 1990s, computer scientists and physicists tried to develop a theory on quantum computing and find specific problems where a quantum computer would be beneficial. The first algorithms made by Deutsch-Josza [DJ92], Bernstein–Vazirani [BV97], and Simon [Sim97], where simple but already showed provable exponential speedups. Later, the development of Phase Estimation, Quantum Fourier Transform led to the famous Shor's algorithm [Sho99] for solving prime number factoring in 1994.

Behind these specific algorithms hides the field of *Complexity theory*, and the question of what nature is able to compute efficiently, and what happens if we add quantum physics to it? The discovery of efficient quantum algorithms would invalidate the Church-Turing thesis which, in its modern complexity theoretical formulation, states:

> "A probabilistic Turing machine can efficiently simulate any realistic model of computation."

Consider a yes-no problem with an input of n bits. We call **P** the class of problems *solvable* in time polynomial in n by a computer, or more precisely a Turing Machine. **NP** is the class of problems where, if a solution is given, we can *verify* it in polynomial time in n. Some problems that are in **NP** but not in **P** are called

NP-complete problems. Notably, the question of proving or not that **P**≠**NP** is one of the million-dollar problems of the Clay Math Institute. The class of problems efficiently solvable by a quantum computer with some constant error allowed is called **BQP** for Bounded-Error Quantum Polynomial-Time. We could compare it to **P** but since quantum measurements are probabilistic, it is fairer to compare it to **BPP** for Bounded-error Probabilistic Polynomial time, the equivalent of **P** with the ability to give a solution with some constant probability. It is easy to show **BQP** contains **BPP**, meaning that every (probabilistic) classical circuit can be simulated by a quantum circuit. But is the reciprocal true? In fact, proving that **BQP**≠**BPP** would invalidate the Church-Turing thesis cited above, and be key to understand the power of quantum computing. However, it is believed that **NP** $\not\subseteq$ **BQP**, meaning that some important problems hard to solve but easily checkable, would eventually not be solvable by a quantum computer. In conclusion, **BQP** is something else, a complexity class made stranger due to quantum nature.

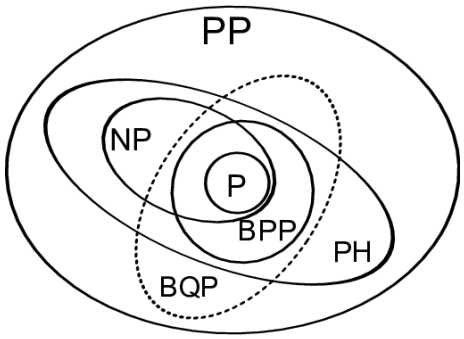


Figure 1.1: A map of fundamental complexity classes. Source: [NM14].

Researchers realized quickly that qubit errors, due to various quantum effects such as decoherence or uncontrolled state perturbation, would be a major flaw for quantum computers. As for classical computing, a theory of *error correction* has been developed [KL97, NC02]. In error correcting codes, a *logical* qubit in state $|0\rangle$ or $|1\rangle$, is in fact composed of many *physical* qubits. Current error correcting codes imply a strong overhead in the number of qubits required for a denoised device.

We call *universal fault tolerant* quantum computers (FTQC) the ideal quantum computers, with a universal set of gates, and a high number of logical qubits. Although they seem far away, conceptualizing and working on these ideal computers help us to understand theoretically what are the hopes and the limits. Proving a serious limitation could thus call into question the efforts currently deployed, or on the contrary provide even more excitement as Shor's algorithm did in 1994.

Throughout this thesis, the majority of the quantum machine learning algorithms will be suited for such FTQC devices, except for Chapter 12 where we propose a quantum circuit that we effectively implement on a real quantum computer.

Machine Learning

Machine learning is a subfield of artificial intelligence. Its specificity is to perform tasks in a radically different way than what is usually considered as algorithms. Indeed, these algorithms are made to progressively learn how to solve a problem instead of being the most efficient solution by design. There exist plenty of algorithm

families which all have their properties, their formalism and applications, while remaining very general. Modern developments include deep learning, or artificial neural networks, which are allegedly built to mimic neurons connectivity in the brain. These methods have become essential in all domains of science.

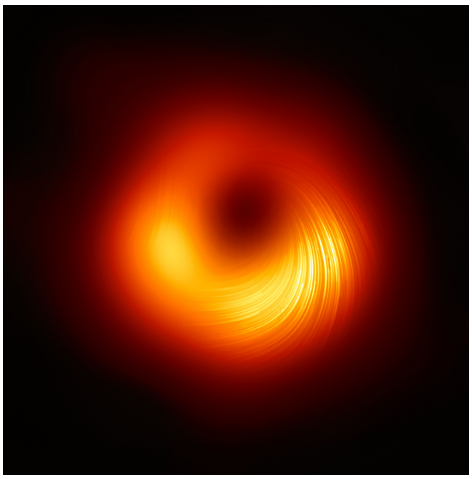


Figure 1.2: On April 2019, the first image of the blackhole M87* was produced using petabytes of data and machine learning algorithms [AAA+19]. Updated image from March 2021.

In the last decade, machine learning algorithms have pushed the boundaries of science and information processing more than we could have imagined. Among the most exciting and recent discoveries, in 2019, the Event Horizon Telescope (EHT) reconstructed the first image of a black hole in the galaxy M87 using a machine learning algorithm [AAA+19]. It used a supercomputer to process an enormous amount of data, gathered from 10 telescopes around the world: more than 10 petabytes, equivalent to the number of pictures 100.000 people would take in their entire life. In 2021, 1200 gravitational lenses, another phenomenon predicted by Einstein (again) in the theory of general relativity, have been discovered thanks to a machine learning algorithm. This algorithm found them in an image of Space containing 10 trillion pixels, or 1 petabyte [HSG+21]. This discovery could help us understand the expansion of the universe and discover new galaxies. Discoveries also concern biology since in 2020 researchers used machine learning algorithms [SEJ+20] to solve the problem of protein folding for the first time at the CASP competition [Cal20]. This opens new paths for understanding life or discovering new medical treatments.

All these developments come at a cost of a tremendous amount of data processing, requiring the most powerful supercomputers available. Machine learning may reach its limit with the never-ending global data growth, associated with the increasing complexity of the algorithms used. Efficient computing will become mandatory since machine learning is now helping in many domains, from medical applications, image processing, social networks, experimental science, safety systems, and may even help for fighting against climate change [RDK+19].

Recent Developments

In recent years, a lot of efforts have been done by physicists all over the world to make the first experimental realizations of qubits, quantum logical gates, and

now quantum computers. In 2017, anyone could access IBM's superconducting qubit quantum computer and launch a circuit. More recently, in 2019, Google demonstrated the first quantum "supremacy" experiment [AAB+19]. Their 53 qubit quantum chip was processed random quantum logical gates to create a complex quantum state and sample random outputs. Despite the noise of their device, they showed that for a classical computer to output random results following the same distribution it would require allegedly 10,000 years, versus in 200 seconds for the quantum computer. Note that this result was recently nuanced by [PZ21] who were able to classically simulate the same quantum circuit in 149 days. However, the "supremacy" experiment remains a great achievement that proves the reality of exponential Hilbert spaces.

Since this news made the front page, quantum computing has became more realistic and triggered a lot of hope. Many countries, universities, big and small private companies have started the race of building a fault tolerant universal quantum computer. The expectations are high, and the pressure on the achievements is rising. Due to the universality of quantum computing, every domain is now interested, including optimization, machine learning, chemistry, material science, health, and of course quantum physics in general.

But one question remains uncertain: will a fault tolerant quantum computer be useful? Despite the difficulty of physically building one, which would already be a fantastic scientific achievement for mankind, what useful task would we do with such a machine? And isn't there any fundamental limits to their power, and why? In the rest of this thesis, we will focus these questions on the field of unsupervised machine learning and neural networks and try to answer the following:

> Can a fault tolerant universal quantum computer provide an advantage in machine learning over classical computing?

Both quantum computing and machine learning have universal properties, and the future will probably allow us to find unexpected results by combining these two fields.

1.2 Quantum Machine Learning

Combining quantum computing and machine learning is audacious but justified as they share a deep connection. Both theories are based on a common mathematical formalism, linear algebra, which makes a certain translation possible. Indeed, as we will see later, all machine learning can be written as vectors, matrices, vector spaces and transformations. Algorithms rely on linear algebra properties and theorems to classify, modify or create data points, seen as vectors (Chapter 2 for details). They can also play with representations and map points from a vector space to another where the task is efficient. On the other hand, quantum physics formalism was built around the mathematical description of quantum states, represented as vectors in a complex vector space called the Hilbert space. Therefore, a set of qubits can always be seen as a vector in a high dimensional space, and any quantum gate or circuit as a matrix or linear operation in that space (see Section 3.1 for details).

In short, both theories speak the same language, but they also differ in many aspects. To cite a few: quantum Hilbert spaces are exponentially bigger but don't

allow non-linear transformations, which are common in machine learning. quantum Hilbert spaces are complex, whereas data in machine learning is mostly real numbers. quantum algorithms deal with quantum states but machine learning requires classical inputs and classical outputs. quantum vectors are normalized, which can be undesirable for representing data in machine learning.

The goal of quantum machine learning (QML) [BWP+17] is to find a common theory for developing quantum algorithms that implement known or unknown machine learning tasks. It is also about using the differences between the two fields to propose new algorithms or to enhance the existing ones.

This deep connection between the two fields become real in 2009 with the HHL algorithm [HHL09] that solve linear systems and matrix inversion with a proven exponential speedup on a quantum computer. Given an N-dimensional input vector $b \in \mathbb{R}^N$ and a Hermitian matrix $A \in \mathbb{R}^{N \times N}$, the task is to find a vector $x \in \mathbb{R}^N$ such that:

$$Ax = b \tag{1.6}$$

Solving this problem comes down to find the inverse A^{-1} of the matrix A, as $x = A^{-1}b$. On a classical computer, this requires in general $\sim N$ iterations, but the HHL allows to solve it in only $\sim \log(N)$ steps. This represents an exponential speedup that could be a practical game changer, as this computational task appears all over science including fluid mechanics, optimization, physics in general, but also machine learning. This breakthrough also questioned our abilities to convert the inputs b and A in quantum states to be further processed by a quantum circuit, as well as the way of recovering a classical output from it [Aar15].

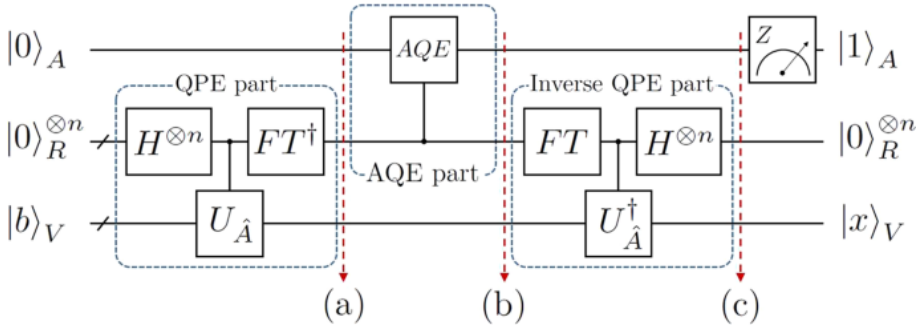


Figure 1.3: Quantum circuit for the HHL algorithm. Source: [LJL19]

Using the HHL algorithm, the first proposals for precise quantum machine learning algorithms appeared a few years later. They concerned various tasks including simple linear regressions, the dimensionality reduction called Principal Component Analysis (PCA) [LMR14], Nearest Neighbors algorithm [WKS14a], topological data analysis [LGZ16], recommendation systems [KP16], classification with Support Vector Machines (SVM) [RSML18], unsupervised learning and clustering [ABG13, LMR13]. Later on, attempts to provide quantum algorithms for neural networks and deep learning were proposed [WKS14b, RBWL18, FN18].

In parallel to quantum machine learning, a concerted research effort has been made to find quantum algorithms for optimization problems. As we will see in Chapter 2, machine learning is intimately linked to optimization, in particular concerning the various way of performing gradient descent. Recent results include a quantum gradient descent algorithm [KP20a], as well for the interior point

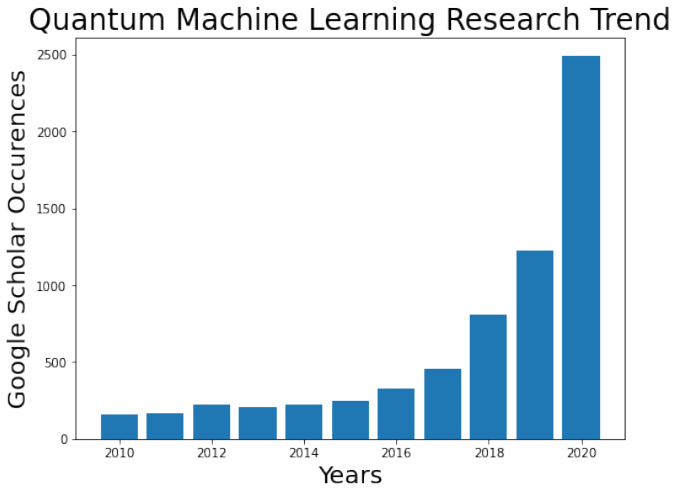


Figure 1.4: Occurrences of the basic keywords "quantum machine learning", "quantum neural network(s)", and "quantum deep learning" in Google Scholar's articles 2010–2020.

method [KP20b] and more generally algorithms for solving LP and SDP problems [VAGGdW17, BS17, vAG18]. For instance, a quantum solver for second order cone programming is directly applicable to support vector machine [KPS21], a popular machine learning algorithm.

It is worth noticing that since the start of this thesis, an impressive amount of new QML algorithms were proposed. Some of them followed the initial works presented above. Others shifted to a new paradigm called variational quantum circuits, where ideas of machine learning (tunable parameters) were directly applied to quantum circuits themselves. In addition, the first implementations on actual quantum hardware were made possible by various institutions and companies allowing for real experimentation: IonQ, IBM, Google, Xanadu, Rigetti, Pasqal, and many more. Besides, many software projects were made to program quantum circuits easily: Qiskit, Pennylane, Cirq, Forge, Q# and plenty more. In only three years, the evolution is noticeable at all levels and brings great hope for the future of quantum computing.

1.3 Contributions

The approach of this thesis is to pursue these works and find new quantum algorithms that correspond to existing machine learning methods that are used in practice. We focus our scope to clustering or unsupervised learning algorithms, as well as on neural networks or deep learning methods. Moreover, we always prove that using a quantum computer would benefit in some manner. Defining and proving a quantum *advantage* is the key difficulty in most cases. It can be a theoretical complexity result for the running time, a quantum circuit with shorter depth, or with few qubits. It can also relate to the final accuracy of the algorithm, in theory with the control on the errors, but also practice on real datasets.

Complexity or running time results are presented with the $O(\cdot)$ notation, indi-

cating asymptotic growth with the size of the problem. For instance, problems of size N that require N, $1/2N+3$, or $100N$ steps to be solved, have each a complexity of $O(N)$, indicating proportionality to N. This notation always emphasizes on most dominant asymptotic terms, and for instance we have $N^3 + 10\sqrt{N} + 4\log(N) = O(N^3)$. We also use the $\widetilde{O}(\cdot)$ symbol to discard the terms that grow logarithmically slow. Therefore, $O(\log(N)N^2)$ can be written $\widetilde{O}(N^2)$. See Section 1.4 for more details.

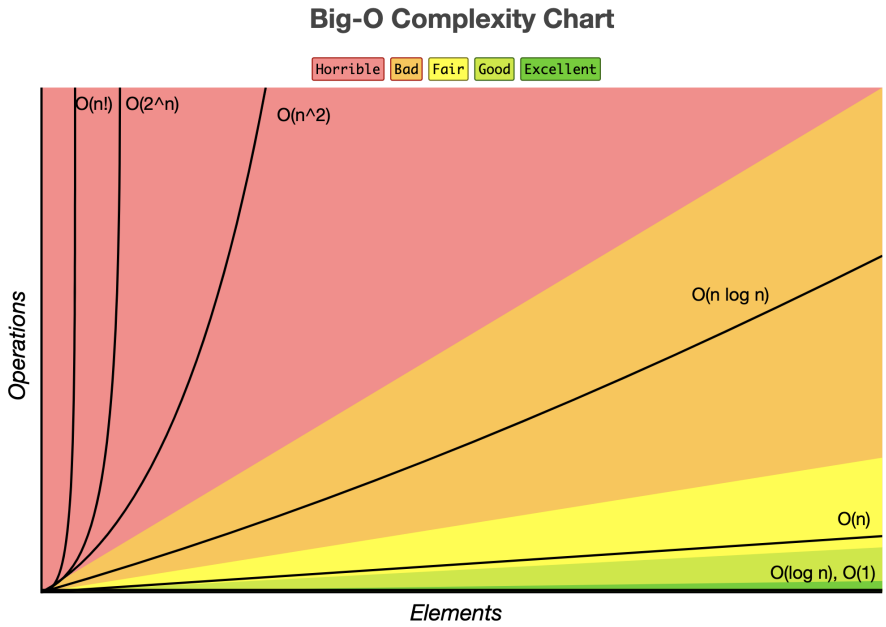


Figure 1.5: Representation of Big-O notation for different asymptotic growths. Source: Bigocheatsheet

To prove a speedup over a classical algorithm, the quantum algorithm must be comparable in some sort. To do so, we require it to be end-to-end and comparable: It should start from classical input, follow similar steps and returning a classical answer. With rigor, we tried to analyze any source of error due to quantum randomness during measurement, and include it in our final running time for a fair comparison. We also have simulated our quantum algorithms on real datasets to compare them in practice with their classical counterparts.

We now present the structure of this thesis and the corresponding results. for each result, we compare the complexity of the previous classical or quantum algorithm and the one from this thesis. Note that they often depend on some parameters define as

- N: usually the number of points, size of the problem, input size, etc.
- d: dimension of the points or vectors for the problem to solve.
- k: number of classes or clusters to find in a dataset.
- ϵ, δ: error or precision parameters due to quantum effects. There is most of the time a trade-off between speed and accuracy.

- μ, η: these are data dependant parameters specific to quantum linear algebra. See Definitions 5.1 and 6.1. In a nutshell, μ is derived from a data matrix norm, and η is the maximum norm of the vectors in the dataset.
- Other parameters should be defined in Section 1.4 or in the corresponding theorems.

Part II To create our quantum algorithms, we used existing quantum linear algebra tools and developed new ones as well. These tools were general enough to be reused across algorithms and are the common thread of most of this thesis. Among others, we propose quantum algorithms for inner product and distance estimation (Section 6.2, Theorem 6.1). As we will see, they allow quantum computing to speak the same language as machine learning. Using them, we also provide an algorithm for a quantum convolution product (Section 11.2.1, Theorem 11.1). As well as a quantum processing routine for graph-based machine learning including the fast creation of an adjacency graph and its Laplacian matrix (Section 9.2, Theorem 9.3). Finally, a new quantum tomography procedure with ℓ_∞-norm error bounds is introduced (Section 4.2.1, Theorem 4.3), to retrieve a classical description of a quantum state faster, while keeping the meaningful information in the context of neural networks.

Algorithm	Type	Running Time
Inner product or distance estimation (IPE)	Classical	$O(Nd)$
Quantum IPE [KLLP19]	Quantum	$\widetilde{O}(\eta/\epsilon)$
Tensor convolution product	Classical	$O(N_o K)$
Quantum tensor convolution product [KLP20a]	Quantum	$\widetilde{O}(\eta/\epsilon)$
Projected Laplacian matrix creation	Classical	$O(N^3)$
Quantum projected Laplacian matrix [KL21]	Quantum	$\widetilde{O}(\mu\kappa/\epsilon)$

Table 1.1: Summary of contributions for fundamental linear algebra routines. In the classical convolution product, N_o, and K are respectively the output size and the kernel size. The quantum convolution product returns a quantum state.

Algorithm	Type	Running Time
ℓ_2 tomography [KP20b]	Quantum	$O(d\log(d)/\epsilon^2)$
ℓ_∞ **tomography [KLP20a]**	Quantum	$O(\log(d)/\epsilon^2)$

Table 1.2: Summary of contributions for quantum tomography. d is the dimension or the number of elements in the quantum vector.

Part III We then focus on quantum algorithms for unsupervised machine learning. We propose q-means, a new quantum algorithm [KLLP19] providing a potential exponential speedup to one of the most basic and widely used clustering algorithms, the k-means algorithm (Chapter 8, Theorem 8.3). We build upon this result by introducing another quantum algorithm [KL21], an analog of the spectral clustering algorithm, which uses the k-means method on top of graph-based machine learning. (Chapter 9, Theorem 9.4).

Algorithm	Type	Running Time
k-means	Classical	$O(Nkd)$
Quantum k-means [LMR13]	Quantum	$O(Nk\log(d)/\epsilon)$
q-means [KLLP19]	Quantum	$\widetilde{O}(\log(N)k^2d\eta^{1.5}/\epsilon^3)$

Table 1.3: Summary of contributions for the k-means algorithms. N is the size of the dataset, d the dimension of each vector, and k the number of clusters. The runtime for q-means is simplified and for the case of *well-clusterable* datasets. The result from [LMR13] outputs a quantum state and would become linear in d to produce a classical output as in our work [KLLP19].

Algorithm	Type	Running Time
Spectral clustering [NJW02]	Classical	$O(N^3)$
Quantum spectral clustering [KL21]	Quantum	$O(\log(N)\mu)$

Table 1.4: Summary of contributions for the spectral clustering algorithms. μ in the quantum algorithm is $O(N)$ in the worst case and in our numerical experiments.

Part IV Next, the same tools are adapted to develop a framework for quantum neural networks, also called quantum deep learning. In particular, we introduce an algorithm for quantum convolution neural network [KLP20a] (Chapter 11, Theorem 11.1).

We also propose a different type of quantum circuit, suited for Noisy Intermediate Scale Quantum computers, or NISQ [Pre18] (see Section 3.3), currently available. These quantum circuits have a specific pyramid shape and data encoding, allowing them to implement a neural network with orthogonal properties (Chapter 12).

Algorithm	Type	Running Time
Convolutional CNN layer [LBBH98]	Classical	$O(N_oK)$
Quantum CNN [KLP20a]	Quantum	$O(\sigma N_o\eta/\epsilon)$
Orthogonal NN inference [JLW+19]	Classical	$O(N^2)$
Orthogonal NN training [JLW+19]	Classical	$O(N^3)$
Pyramidal OrthoNN [KLM21] inference	Quantum	$O(N/\delta^2)$
Pyramidal OrthoNN [KLM21] training	Classical	$O(N^2)$

Table 1.5: Summary of contributions for neural networks algorithms. For CNN, N_o and K are respectively the output size and the kernel size. σ is a ratio in [0,1]. *OrthoNN* stands for Orthogonal Neural Network and N is both the input and output size of a single layer.

These contributions were the subject of scientific publications, which are listed below:

- [KLLP19] "q-means: A quantum algorithm for unsupervised machine learning". Published in *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS) - 2019, pp.4136-4146.* By I. Kerenidis, J. Landman, A. Luongo, A. Prakash.
- [KLP20a] "Quantum Algorithms for Deep Convolutional Neural Networks" Published in *Proceedings of the 8th International Conference on Learning Representation (ICLR) - 2020.* By I. Kerenidis, J. Landman, A. Prakash
- [KL21] "Quantum Spectral Clustering". Published in *Physical Review A 103, 042415 - April 2021.* By I. Kerenidis, J. Landman.
- [KLM21] "Classical and Quantum Algorithms for Orthogonal Neural Networks". By I. Kerenidis, J. Landman, N. Mathur. *(Under submission)*
- [BFL21] "Quantum Inference Algorithm for Bayesian Neural Networks". By N. Berner, V. Fortuin, J. Landman. (Under submission)
- [MLL+21] "Medical Image Classification Via Quantum Neural Networks". By N. Mathur, J. Landman, Y. Li, M. Strahm, S. Kazdaghli, A. Prakash, I. Kerenidis. (Under submission)

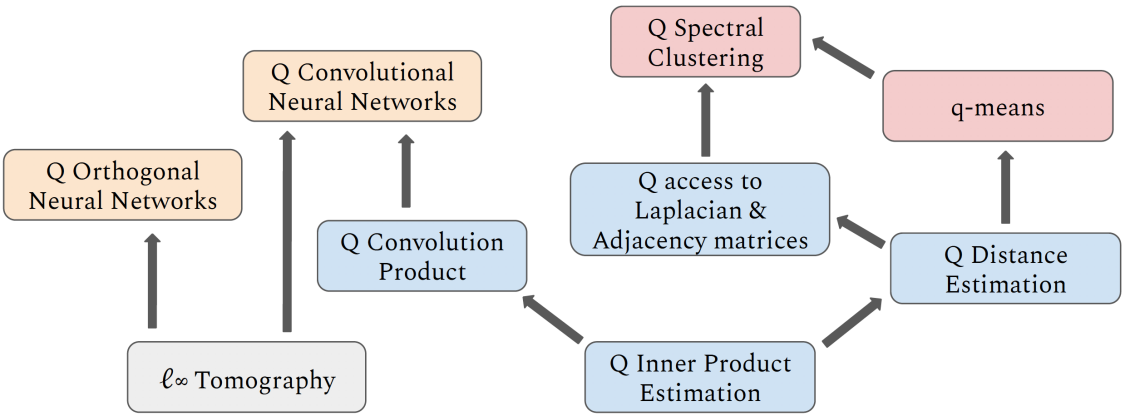


Figure 1.6: Diagram representation of the main contributions. Arrows denote dependencies between linear algebra (blue), tomography (grey), unsupervised machine learning (red), and neural networks (orange) algorithms. Q stands for "Quantum".

1.4 Mathematical Notations

We introduce basic notations and definitions for the understanding of this dissertation.

$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{R}^+$, and $\mathbb{C}$ are respectively the integers, natural numbers, real numbers, positive real numbers, and complex numbers. For an integer $N > 0$, $[N]$ denotes the set of integers between 1 and N included. i is the imaginary number such that $i^2 = -1$, but is also often used as the index of numbered elements. A Hilbert space is a real or complex vector space with an inner product.

Vectors are often written as x, v, s, or y and are elements of (a subspace of) $\mathbb{R}^d$ (or $\mathbb{C}^d$), where $d > 0$ is the dimension. Therefore, a vector $x \in \mathbb{R}^d$ vectors have d real components, each in $\mathbb{R}$ (or $\mathbb{C}$). We write $x = (x_1, x_2, \cdots, x_d)$ or $x = \sum_{i=1}^{d} x_i e_i$ where e_i is the i^{th} vector of the standard or canonical basis. Note that in some cases, the vector's components will be indexed from 0 to $d-1$ instead. x or v will often denote the input vector of an algorithm, y the output vector. c often stands for a centroid vector (center of a cluster).

The ℓ_p norm of a vector is $\|x\|_p = (\sum_{i=1}^{d} |x_i|^p)^{1/p}$. In particular, we use the ℓ_2 norm $\|x\|_2 = \sqrt{\sum_{i=1}^{d} |x_i|^2}$, and we define the ℓ_∞ norm as $\|x\|_\infty = \max_{i \in [d]} |x_i|$. If the subscript is not specified, $\|x\|$ usually represents the ℓ_2 norm.

The inner product between two d-dimensional real vectors x and y is written (x, y), $x \cdot y$ or $x^T y$ and is equal to $\sum_{i \in [d]} x_i y_i$. We have $\|x\|_2 = \sqrt{(x, x)}$ and the euclidean distance between x and y is:

$$d(x, y) = \|x - y\|_2 = \sqrt{\|x\|_2^2 + \|y\|_2^2 - 2(x, y)} \tag{1.7}$$

The normalized inner product is $\langle x|y\rangle$ such that $(x, y) = \|x\|_2 \|y\|_2 \langle x|y\rangle$. Two vectors x and y are orthogonal if $(x, y) = 0$. We use $x^\perp$ to denote a vector orthogonal to x.

For a collection of N vectors, also called *dataset* of size N, we can number each vector in the set $\{x^i\}_{i \in [N]}$. Therefore, the j^{th} component of the i^{th} vector is written x_j^i. However, we often used $\{x_i\}_{i \in [N]}$ with each $x_i \in \mathbb{R}^d$ to denote the i^{th} vector and not its component, which could lead to confusion. A dataset can also be represented as a matrix $A \in \mathbb{R}^{N \times d}$ (also V, X, Y, M or S). Indeed, N vectors of d dimensions can compose the N rows of a matrix. Therefore the i^{th} vector can be written A_i and its j^{th} component is A_{ij} or $A_{i,j}$. We denote its transpose $A^T \in \mathbb{R}^{d \times N}$, with elements A_{ji}.

Let A be a *square* matrix in $\mathbb{C}^{N \times N}$. If A is said *invertible*, we denote A^{-1} the inverse of A such that $AA^{-1} = I$ where I is the identity matrix. The adjoint of A is $A^\dagger = \overline{A}^T$, where $\overline{A}$ is the complex conjugate of A. The matrix A is Hermitian if $A = A^\dagger$ and unitary if $AA^\dagger = A^\dagger A = I$. Note that for real matrices, being unitary is equivalent to being orthogonal.

The Singular Value Decomposition (SVD) of a *rectangular* matrix $A \in \mathbb{R}^{N \times d}$ is of the form $A = U\Sigma V$, where $U \in \mathbb{R}^{N \times N}$, $V \in \mathbb{R}^{d \times d}$ and Σ is a rectangular diagonal matrix with non negative elements σ_i called the *singular values*. If $r \leq \min(N, d)$ is the rank of A, we can write:

$$A = \sum_{i \in [r]} \sigma_i u_i v_i^T \tag{1.8}$$

where u_i and v_i are respective columns of U and V. We can define the pseudo inverse of A as $A^+ = \sum_{i \in [r]} \frac{1}{\sigma_i} u_i v_i^T$. The condition number $\kappa(A)$ is the ratio between the biggest and the smallest singular values $\kappa(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$.

Let $A \in \mathbb{C}^{N \times N}$ be a *diagonalizable* matrix. Then A has N eigenvectors v_i and eigenvalues λ_i such that $Av_i = \lambda_i v_i$. The vectors v_i form a basis in $\mathbb{R}^N$. An unitary matrix is diagonalizable and its eigenvalues are such that $|\lambda_i| = 1$. The sparsity s of A is the maximum number of non zero elements in a row of A.

A symmetric matrix $A \in \mathbb{R}^{N \times N}$ is said to be *positive semidefinite* if, for any vector $x \in \mathbb{R}^N$, we have $x^T Ax \geq 0$. Then all eigenvalues of A are non negative.

We define possible norms for a matrix $A \in \mathbb{C}^{N \times d}$ with elements A_{ij}. the Frobenius norm $\|A\|_F$ if the generalization of the ℓ_2 norm, $\|A\|_F = \sqrt{\sum_{i,j} |A_{ij}|^2} = \sqrt{\sum_i \|A_i\|_2^2}$. For a *square* matrix we have $\|A\|_F = \sqrt{\sum_i \lambda_i^2}$. The spectral norm of A is written $\|A\|_2$ or $\|A\|$ and is the biggest singular value of A.

Tensors are the generalization of matrices with more than two dimensions. 3D tensors are indexed by i, j and d.

For an algorithm depending on the variable N, its running time is stated in the standard asymptotic notation $O(f(N))$ which indicates a running time upper bounded by $cf(N)$ for a fixed $c \in \mathbb{R}^+$ and sufficiently large $N > 0$. The notation $\widetilde{O}()$ hides polylogarithmic factors (e.g. $\log^2(N)$), that is $O(f(N)polylog(N))$ is represented by $\widetilde{O}(f(N))$.

Note finally that across Chapters and algorithms, it is possible that notations switch due to the context e.g. in Chapter 9 singular values are also written λ_i. For neural networks in Chapter 12, the input vectors are of size n and the output of size d.

Chapter 2

Classical Machine Learning

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."

Alan Turing
Computing Machinery and Intelligence (1950)

2.1 Introduction

The growing importance of machine learning in Science, but also in industry and in our society, is undeniable. Recent advances in signal processing, time series forecasting, medical predictions, image recognition, anomaly detection, or generative data, have surpassed most expectations. In 2019 deep learning inventors [LBH15] were awarded the Turing medal, and together with Learning Theory and Neuroscience, the scientific community is working to understand how the brain learns.

The sophistication of the algorithms used, and the amount of data necessary to train them seems staggering. The quantity of data generated by our society, from the web, various captors, open medical data, is expected to grow beyond comprehension. Most of this data will be multimodal, complex, and unlabelled. Therefore the help of unsupervised machine learning and neural networks (or deep learning) will become more necessary, but more powerful ways of making sense of this amount of data will be necessary as well.

In the following, we will introduce basic concepts and notations in machine learning, required for this thesis. Complete courses can be found in [Bis06] and [GBCB16] for deep learning.

A machine learning task aims to extract information from data. It usually consists of a parametrized function (or model). Its parameters are is progressively tuned (or trained) such that the task is done as efficiently as possible. We then say that the model has *learned* to predict using the data provided.

Data points could represent points in space, a set of numbers, or even images decomposed as pixels. In the general case, a dataset $\mathcal{D}$ consists of N vectors $\{x_i\}_{i\in[N]}$,

where each vector lies in a subspace of $\mathbb{R}^d$. Said differently, each vector has d features, or is d-dimensional. The model to train can be written as a function $f(x|\theta)$, where θ are the parameters to tune, that should map each input $x_i \in \mathbb{R}^d$ to an output $y_i \in \mathbb{R}^{d'}$. The outputs can be of any sort as well. In the context of regression, it usually consists of a number, and $d' = 1$. In classification, where the goal is to put on each input a *label* (or *class*), we have $d' = k$, where k is the number of classes.

The most common machine learning branch is supervised learning, where the dataset is provided with labelled data. Namely we are given a dataset $\mathcal{D} = x_i, y_{ii\in[N]}$. Training such algorithms boils down to being able to predict the right y_i for each x_i with a *training set*, and then ensure that the model can also predict the right result for a *testing set* of pairs (x_i, y_i) that haven't been seen during the training. Supervised learning are usually trained by adjusting the parameters $\theta = (\theta_1, \cdots, \theta_m)$ for a chosen number m, such that a *loss* or *cost* function $\mathcal{C}$ decreases. This loss is calculated from the accuracy of the predictions made on the training set. Then, we perform a *gradient descent* to update each parameters θ_j with a *learning rate* $\lambda > 0$:

$$\theta_i \leftarrow \theta_i - \lambda \frac{\partial \mathcal{C}}{\partial \theta_i} \tag{2.1}$$

In this thesis, supervised learning will be used as the framework for neural networks (see Section 2.3). Before that, in the next section, we will introduce unsupervised machine learning and two specific algorithms.

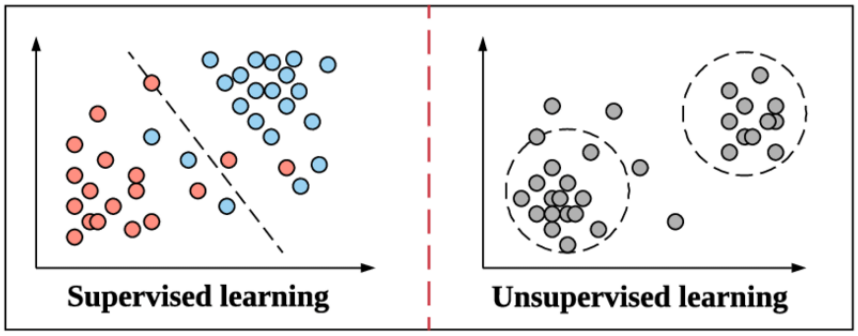


Figure 2.1: Schematic differences between supervised and unsupervised machine learning. Source: [QSW+20].

Note that, for each algorithm presented, notations for inputs, outputs, parameters, matrices, and different numbers, may change but will remain consistent between classical and quantum versions.

2.2 Unsupervised Learning

As shown in Fig.2.1, unsupervised learning deals only with unlabelled dataset $\mathcal{D} = \{x_i\}_{i\in[N]}$. The algorithms must find by themselves the labels y_i to assign at each input x_i. In the context of classification or segmentation, this automatic process is often referred to as *clustering*. It aims to find clusters among data points, each

cluster can then be converted as a certain class or label y_i. Note that unsupervised learning can also include generative tasks such as generative adversarial neural networks [GPAM$^+$14], which have been studied as quantum algorithms as well [DDK18, LW18].

The next two algorithms are the k-means algorithm and the spectral clustering algorithm. They are closely linked, as the latter relies on the former. Given a distribution of points in a vector space, their goal is to identify clusters among them and further classify new points. This problem is known to be NP-complete [Vat09] (see Section 1.1). Both algorithms are iterative, non deterministic algorithms, or heuristic, that solve this problem with good accuracy on simple cases. k-means clustering has a complexity of $O(N)$ per iteration, where N is the number of points, but suffers from poor flexibility and requires well-shaped datasets. Spectral clustering however uses properties of graph theory to distinguish complex data, at the cost of a higher complexity of $O(N^3)$ per iteration. This can be seen in simple examples showed in Fig.2.2.

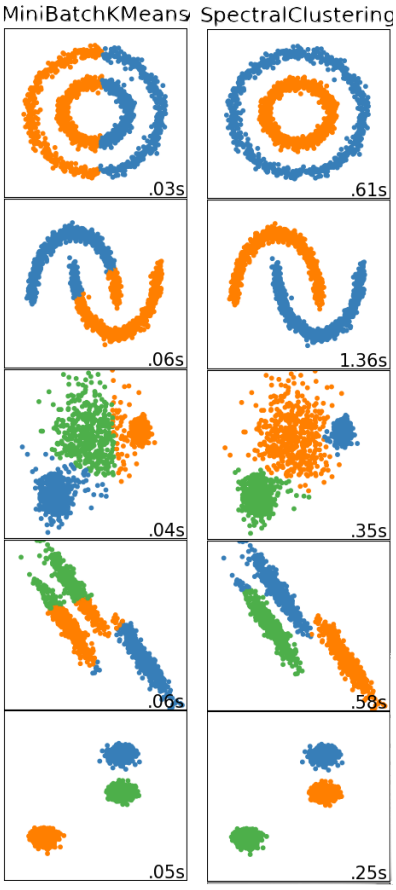


Figure 2.2: A comparison between k-means and spectral clustering on different types of toy datasets. We can see that spectral clustering distinguishes nested datasets with more accuracy, but is unfortunately slower. Source: Scikit Learn [PVG$^+$11].

2.2.1 k-means Clustering

The k-means algorithm was introduced in 1982 [Llo82], and is extensively used for unsupervised problems. The inputs to k-means algorithm are vectors $v_i \in \mathbb{R}^d$ 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++ [AV07]. 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.

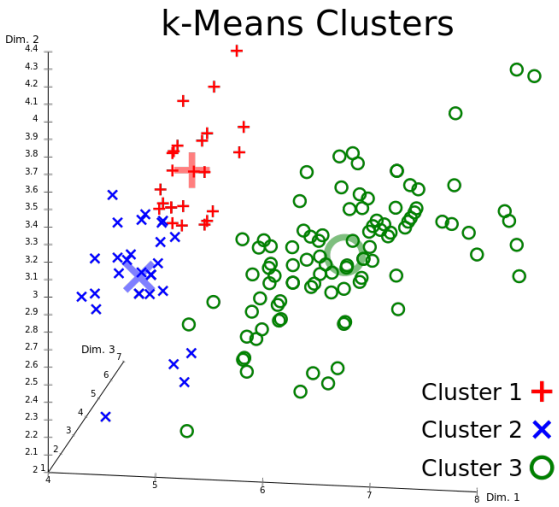


Figure 2.3: 3D representation of the k-means clustering applied on the IRIS dataset of 3 types of flowers. Source: Wikipedia.

More precisely, we are given a dataset V of vectors $v_i \in \mathbb{R}^d$ 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 v_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 $d(v_i, c_j^t)$ be the Euclidean distance between vectors v_i and c_j^t. The first step of the algorithm assigns each v_i a label $\ell(v_i)^t$ corresponding to the closest centroid, that is

$$\ell(v_i)^t = \text{argmin}_{j\in[k]}(d(v_i, c_j^t)) \tag{2.2}$$

The centroids are then updated, $c_j^{t+1} = \frac{1}{|C_j^t|}\sum_{i\in C_j^t} v_i$, 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 we have

$$\frac{1}{k}\sum_{j=1}^{k} d(c_j^t, c_j^{t-1}) \leqslant \tau \tag{2.3}$$

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.

$$\text{RSS} := \sum_{j\in[k]}\sum_{i\in C_j} d(c_j, v_i)^2 \tag{2.4}$$

The RSS decreases 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 algorithm can be super-polynomial in the worst case (the number of iterations is $2^{\omega(\sqrt{N})}$ [AV06]), but the number of iterations is usually small in practice. The k-means algorithm with a suitable heuristic like k-means++ 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 [AV07].

In Section 8.1.2, we will introduce a slightly different version of the algorithm, named δ-k-means, which includes some noise and randomness to be fairly comparable to the quantum algorithm q-means presented in Section 8.2.

2.2.2 Spectral Clustering

A summary of all variables along with their definition is given in Chapter 9, Table 9.1.

Notations and Definitions

Let $S \in \mathbb{R}^{N\times d}$ be the input of our clustering task. S is the data matrix composed of N vectors $s_i \in \mathbb{R}^d$, for $i \in [N]$. The spectral clustering method uses a graph derived from the data S, where similar points are connected. We define the distance between two points by $d_{ij} = \|s_i - s_j\|$.

We consider the undirected graph for which each of the N nodes corresponds to a data point. The value of the edge connecting two nodes i and j is 1 if the two nodes are connected and 0 otherwise. More generally we will denote by $a_{ij} \in \{0, 1\}$ the value of this edge. By convention we have $a_{ii} = 0$. We define the Adjacency matrix $A \in \mathbb{R}^{N\times N}$ as the symmetric matrix with elements a_{ij}.

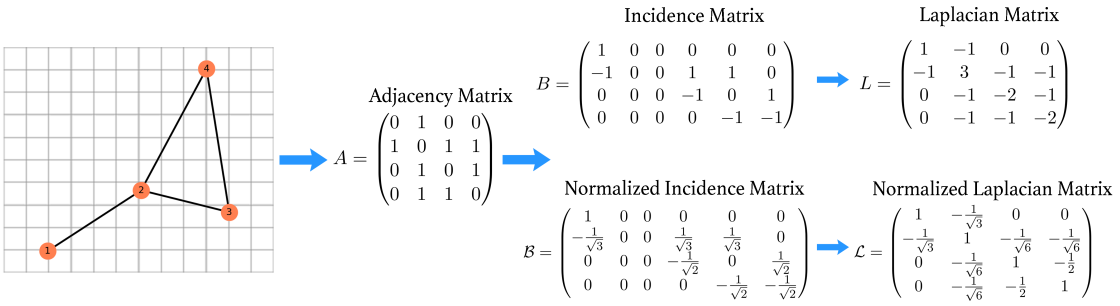


Figure 2.4: Example of the Adjacency, Incidence, and Laplacian matrices of a N=4 nodes graph.

We will use the following construction rule for the graph: the value of an edge between two points s_i and s_j is equal to 1 if their distance satisfies $d_{ij} \leq d_{min}$ and 0 otherwise, for a given threshold $d_{min} > 0$. This choice has been made for simplicity

and to take into account constraints from quantum circuits that will be detailed later.

The Incidence matrix B is another way of representing the graph. Each row of B represents a node whereas a column represents a possible edge. An element of B indicates if an edge is incident to a node. B is not symmetric and has size $N \times \frac{N(N-1)}{2}$. We index the elements of B by three numbers $B_{i,(p,q)}$ where i is the node and (p,q) represents the edge connecting the nodes p and q ordered so that $p < q$. Even though the graph is undirected, the values of B must follow an oriented convention. Therefore the rule for constructing B is the following:

$$B_{i,(p,q)} = \begin{cases} a_{pq} & \text{if } i = p \\ -a_{pq} & \text{if } i = q \\ 0 & \text{if } i \notin \{p,q\} \end{cases} \tag{2.5}$$

We introduce the *normalized* incidence matrix $\mathcal{B}$, with elements defined by $\mathcal{B}_{i,(p,q)} = \frac{B_{i,(p,q)}}{\|B_i\|}$, where B_i is the i^{th} row of B. Therefore each row $\mathcal{B}_i$ has unit norm.

The Laplacian matrix is defined by $L = BB^T$. We introduce the normalized Laplacian matrix as $\mathcal{L} = \mathcal{B}\mathcal{B}^T$. It inherits the properties of the Laplacian matrix and will be used for classification. Note that the usual definition $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$, with D the Degree matrix, coincides if the edges are either 0 or 1.

$\mathcal{L}$ is a symmetric and positive semidefinite matrix in $\mathbb{R}^{N\times N}$. The n eigenvalues of $\mathcal{L}$ are real and positive. We denote them $\{\lambda_1, \cdots, \lambda_N\}$, and their corresponding eigenvectors are $\{u_1, \cdots, u_N\}$. The eigenvalues are ordered such that $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$. For a given integer $k \in [N]$, we will denote by $\widetilde{\mathcal{L}}^{(k)}$ the projection of $\mathcal{L}$ on its k lowest eigenvalues.

Since $\mathcal{L} = \mathcal{B}\mathcal{B}^T$, the N singular values $\lambda_j^{\mathcal{B}}$ of $\mathcal{B}$ are such that $\lambda_j = (\lambda_j^{\mathcal{B}})^2$. Indeed, using the singular value decomposition (SVD), there exist two orthonormal matrices U and V and the diagonal matrix $\Sigma \in \mathbb{R}^{N\times N}$ with elements $(\lambda_1^{\mathcal{B}}, \cdots, \lambda_N^{\mathcal{B}})$, such that $\mathcal{B} = U\Sigma V^T$. Therefore $\mathcal{L} = U\Sigma^2 U^T$, and the eigenvectors u_j of $\mathcal{L}$ are the left singular vectors of $\mathcal{B}$.

In Chapter 9, to ensure a better running time for the quantum algorithm, we will slightly modify the incidence matrix B by replacing the "0" elements with a small parameter $\epsilon_B > 0$. We will see in the experiments that it does not affect the accuracy of the clustering.

Partitioning the Graph into k Clusters

Once the graph's normalized incidence matrix $\mathcal{B}$ is computed, we can calculate its normalized Laplacian and find its eigenvalues and eigenvectors.

Let $\widetilde{\mathcal{L}}^{(k)} \in \mathbb{R}^{N\times k}$ be the *projected* normalized Laplacian matrix on its k lowest eigenvectors, i.e. The j^{th} column of $\widetilde{\mathcal{L}}^{(k)}$ is u_j, the j^{th} eigenvector of $\mathcal{L}$, for $j = 1, \cdots, k$. The method developed by [NJW02] consists in applying the clustering algorithm k-means (see previous Section 2.2.1) with input the N rows $\widetilde{\mathcal{L}}_i^{(k)}$ of the projected normalized Laplacian $\widetilde{\mathcal{L}}^{(k)}$. Each row $\widetilde{\mathcal{L}}_i^{(k)}$ is a vector of dimension k, corresponding to an input vector s_i in the initial input space.

With this procedure, the k-means clustering takes place in a low-dimensional and appropriate space, ensuring an efficient clustering (see Fig.2.5). The output of

the algorithm could be the label of each point (for example the label corresponding to the nearest centroid) or the k centroids in the spectral space.

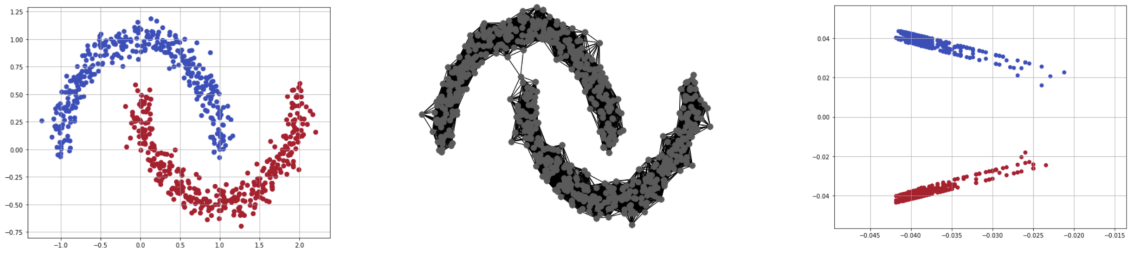


Figure 2.5: Example of the spectral clustering algorithm on two nested half moons. (left) The two classes of N points in two dimensions. (center) The similarity graph. (right) The points projected into the space spanned by the first two eigenvectors of the Laplacian matrix of the graph. In the spectral space (right) we see that the point are easy to separate.

Classical Running Time

The classical algorithm can be decomposed into several steps: the distance calculation between points runs in time $O(dN^2)$, and the creation of the Laplacian matrix in $O(Nm)$ where m is the number of edges in the graph, which is $O(N^2)$ in the worst case. Then, the extraction of eigenvalue and eigenvectors of the Laplacian matrix is done in $O(N^3)$. Finally, the k-means clustering runs in $O(Nk^2)$. The dominant term is in practice therefore $O(N^3)$ (we assume we have more points than dimensions), and the impractical running time of spectral clustering is due to the need for diagonalization of the Laplacian matrix [LLKL11].

Alternative Classical Algorithms

Randomness naturally occurs during our quantum spectral clustering algorithm (see Chapter 9), and one may wonder for fair comparison if an alternative classical algorithm can efficiently spectral clustering with noisy or sampled methods as well. To circumvent the prohibitive running time of the classical algorithm, several approximations have indeed been proposed on different steps. A recent review of these techniques [TL20] concludes that despite many efforts, methods with provable scalability are found limited or worse in practice, whereas other good empirical methods have no provable guarantees.

Some methods aim to build the similarity graph using sampling [CJK+13, RR08, LLKL11]. They present limitations [WGM19] and act by sampling partially the input data, which is not the case of our quantum algorithm. Their running time is often proportional to $O(Nm)$ or $O(Nm^2)$ where m is the number of edges, which is $O(N^2)$ in the worst case. If one of these methods was empirically efficient, we could actually adapt it to our quantum algorithm, by first applying a similar sparsification and then using it as input in our quantum algorithm. Recently such techniques have been done in the quantum setting [AdW20].

At the next step, it is possible to use Lanczos methods to compute the k lowest eigenvalues and eigenvectors, with a running time of $O(N^2)$ for a fully connected

graph. However, they seem to suffer from poor efficiency in practice since they strongly rely on the distribution of the eigenvalues and can require many iterations that would ruin the advantage [BDD+00]. Note that even with an effective application of this method, our quantum algorithm would still be advantageous. We can also cite the use of power methods [BKG15] to solve clustering using approximated eigenvectors.

Some methods try to improve the clustering step itself, by modifying the k-means algorithm [HD15]. One should compare these methods directly with the quantum k-means [KLLP19] (Chapter 8). In most cases, such variations carry over to the quantum case as well. Finally, other attempts use solely preprocessing techniques [YHJ09] on the initial dataset. Again, one could simply use them before the quantum algorithm to similarly improve its practical efficiency.

2.3 Neural Networks

Artificial neural networks may be the most impressive advance in contemporary computing. They were imagined to roughly mimic neuron connectivity in the brain. While being far from complete, this imitation already allowed for impressive advances in machine learning. Today, deep learning has become a state-of-the-art standard in most cases, be it speech recognition, image or video processing, disease detection, etc.

The first attempts to create artificial neural networks go back to the mid XX^{th} century, but the key paradigm for training them efficiently was developed in the 1980s with the *backpropagation* algorithm [RHW86, LBBH98]. Neural networks had to wait until the late 2000s to achieve their worldwide success, thanks to the impressive development of GPUs (Graphics Processing Units) allowing fast implementation of linear algebra routines.

To continue to improve, today's architectures are becoming increasingly complex, deep, and resource-intensive. Training deep networks requires large clusters of GPUs and an excessive amount of time and energy.

For this thesis, we will only review few basic types of neural networks for which quantum algorithms are proposed in Part IV. Emphasis is made on mathematical formalism which will be helpful for the quantum versions of these algorithms. Fully connected neural networks (FCNN) are the original and most basic ones, followed by the backpropagation algorithm, necessary to train all neural networks. We then outline the recent proposal of orthogonal neural networks (OrthoNN). OrthoNNs show special abilities for learning, but that is most interest is their orthogonality constraint that arises naturally in quantum computing. We then present the widely used convolutional neural networks (CNN) which are specialized in signal or image processing. Finally, we detail the backpropagation algorithm in the context of CNN.

2.3.1 Fully Connected Neural Networks

A neural network usually consists of layers of neurons or *nodes*, each being a numerical value. Adjacent layers are connected through *weights*. A network is said to be fully connected if all nodes of a layer are connected to all nodes of the next layer,

as in Fig.2.6. The input layer has as many nodes as the input vector has dimensions. The output layer size is also the dimension of the output. A key feature of neural networks, that gives them the expressive power and universal abilities, is the presence of non-linear *activation functions* at each layer.

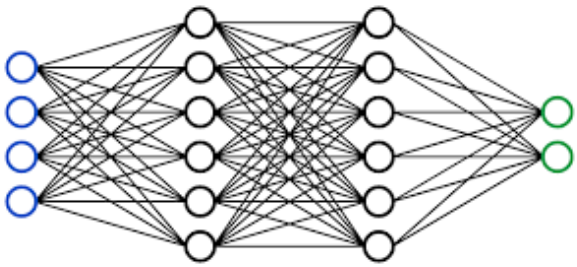


Figure 2.6: A fully connected neural network for 4-dimensional inputs, 2 classes outputs, and two hidden layers. Each line represents a tunable weight.

Layers are numbered $\ell = 1, \cdots, L$, from input to output. Between two layers ℓ and $\ell + 1$, respectively of size n_ℓ and $n_{\ell+1}$, the weights can be embedded in a matrix $W^\ell \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$. For instance, the first column of W^ℓ will correspond to the weights connecting each node of layer ℓ to the first node of layer $\ell + 1$, that is $(w_{00}, w_{10}, \cdots, w_{n^{\ell+1}0})$.

We denote by $a^\ell \in \mathbb{R}^{n_\ell}$ the vector of layer ℓ. The *feedforward* procedure consists of creating the next layer $a^{\ell+1}$ by first doing a matrix product with the weight matrix, and then applying a non-linear function σ. Usually, an extra parameter b^ℓ called *bias* is added to the layer to ensure flexibility to the model. Note that this bias can be discarded in the formalism, as it is equivalent to adding an extra dimension to the value 1 at each layer.

$$z^{\ell+1} = W^\ell a^\ell + b^\ell \tag{2.6}$$

$$a^{\ell+1} = \sigma(z^{\ell+1}) \tag{2.7}$$

This procedure is continued until we obtain the last layer a^L. As we will see in the rest of this thesis, it is important to notice that Eq.(2.6) can be decomposed as several inner products between the input vector a^ℓ and the rows of W^ℓ.

The non-linearity σ is usually taken to be the *sigmoid* function, which has the property of pushing positive and negative values respectively towards $+1$ and 0.

$$\sigma : x \mapsto \frac{1}{1 + e^{-x}} \tag{2.8}$$

It follows that the running time of a single fully connected is dominated by the matrix-vector multiplication at its core, which takes $O(n^\ell n^{\ell+1})$, or $O(n^2)$ in the case of *square* layers.

2.3.2 Backpropagation

The backpropagation algorithm in a fully connected neural network is a well know and efficient procedure to update the weight matrix at each layer [HN92, Roj96].

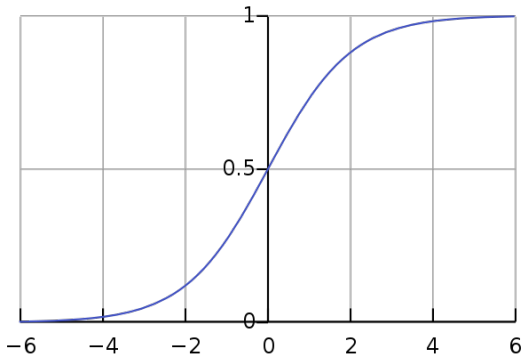


Figure 2.7: The sigmoid function. Source: Wikipedia.

After the last layer a^L, one can define the cost function $\mathcal{C}$ that compares the output to the ground truth during the supervised training. The goal is to calculate the gradient of $\mathcal{C}$ with respect to each weight and bias, namely $\frac{\partial \mathcal{C}}{\partial W^\ell}$ and $\frac{\partial \mathcal{C}}{\partial b^\ell}$. In the backpropagation, we start by calculating these gradients for the last layer, then propagate back to the first layer.

We will require to obtain the *error* vector at layer ℓ defined by $\Delta^\ell = \frac{\partial \mathcal{C}}{\partial z^\ell}$. One can show the backward recursive relation

$$\Delta^\ell = (W^{\ell+1})^T \cdot \Delta^{\ell+1} \odot \sigma'(z^\ell), \tag{2.9}$$

where $\odot$ symbolizes the Hadamard product, or entry-wise multiplication. If using the sigmoid function, we also have the property $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. Note that the previous computation requires simply to apply the layer (i.e. apply matrix multiplication) in reverse. We can then show that each element of the weight gradient matrix at layer ℓ is given by $\frac{\partial \mathcal{C}}{\partial W^\ell_{jk}} = \Delta^\ell_j \cdot a^{\ell-1}_k$. Similarly, the gradient with respect to the biases is easily defined as $\frac{\partial \mathcal{C}}{\partial b^\ell_j} = \Delta^\ell_j$.

Once these gradients are computed, we update the parameters using the gradient descent rule, with learning rate λ :

$$W^\ell_{jk} \leftarrow W^\ell_{jk} - \lambda \frac{\partial \mathcal{C}}{\partial W^\ell_{jk}} \quad ; \quad b^\ell_j \leftarrow b^\ell_j - \lambda \frac{\partial \mathcal{C}}{\partial b^\ell_j} \tag{2.10}$$

The task is repeated until the cost function stops decreasing, indicating a local minimum has been reached. We also refer as *stochastic* gradient descent (SGD) when the loss and its gradients are estimated with one or few samples only, and not on the entire dataset. SGD allows for faster iterations in big datasets, and the imperfect gradient estimations along with good learning rate can help escape from local minima.

The complexity of a single gradient descent update of one layer only is dominated by the time to compute all gradients. Since these are made using a similar matrix-vector multiplication as in the forward pass (see Eq.(2.9)), the complexity is also $O(n^\ell n^{\ell+1})$ for a layer with input size n^ℓ and output size $n^{\ell+1}$. This becomes $O(n^2)$ for a square layer.

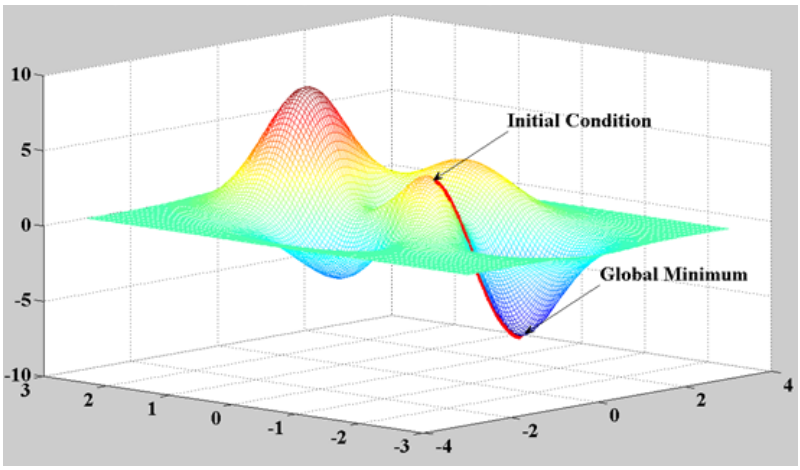


Figure 2.8: Representation of gradient descent on two-dimensional parameter space. The z-axis is the cost function. Source: Matlab.

2.3.3 Orthogonal Neural Networks

The idea behind Orthogonal Neural Networks (OrthoNNs) is to add a constraint to the weight matrices corresponding to the layers of a neural network. Imposing orthogonality to these matrices has theoretical and practical benefits in the generalization error [JLW+19]. Orthogonality ensures a low weight redundancy and preserves the magnitude of the weight matrix's eigenvalues to avoid vanishing gradients. In terms of complexity, for a single layer, the feedforward pass of an OrthoNN is simply a matrix multiplication, hence has a running time of $O(n^2)$ if $n \times n$ is the size of the orthogonal matrix (input and output layers of size n). It is also interesting to note that OrthoNNs have been generalized to convolutional neural networks [WCCY20].

The main drawback of OrthoNNs is to preserve the orthogonality of the matrices while updating them during gradient descent. Several algorithms have been proposed to this end [WCCY20, BCW18, LCMR19], but they all point that pure orthogonality is computationally hard to conserve. Therefore, previous works allow for approximations: strict orthogonality is no longer required, and the matrices are often pushed toward orthogonality using regularization techniques during weights update.

We present two algorithms from [JLW+19] for updating orthogonal matrices.

The first algorithm is an approximated one, called *Singular Value Bounding* (SVB). It starts by applying the usual gradient descent update on the matrix, therefore making it not orthogonal anymore. Then, the singular values of the new matrix are extracted using Singular Value Decomposition (SVD), their values are manually pushed to be close to 1, and the matrix is recomposed hence enforcing orthogonality. This method shows less advantage on practical experiments [JLW+19]. It has a complexity of $O(n^3)$ due to the SVD, which in practice is better than the next algorithm. Note that this running time is still longer than $O(n^2)$, the running time to perform standard gradient descent.

The second algorithm can be considered perfect since it ensures strict orthogonality by performing the gradient descent in the manifold of orthogonal matrices, called the Stiefel Manifold. In practice [JLW+19], this method showed advantageous clas-

sification results on standard datasets. This algorithm requires $O(n^3)$ operations, but is very prohibitive in practice. We give a very informal step-by-step detail of this algorithm:

1. Compute the gradient G of the weight matrix W.
2. Project the gradient matrix G in the tangent space, (The space tangent to the manifold at this point W): multiply G by some other matrices based on W:

$$(I - WW^T)G + \frac{1}{2}W(W^TG - G^TW) \tag{2.11}$$

   This requires several matrix-matrix multiplications. In the case of square $n \times n$ matrices, each has complexity $O(n^3)$. the result of this projection is called the *manifold gradient* Ω.
3. update $W' = W - \eta\Omega$, where η is the chosen learning rate.
4. Perform a *retraction* from the tangent space to the manifold. To do so we multiply W' by Q factor of the *QR decomposition*, obtained using Gram Schmidt orthonormalization, which has complexity $O(2n^3)$.

2.3.4 Convolutional Neural Networks

Convolutional neural networks (CNN) are a specific type of neural networks, designed in particular for image processing or time series. They use the *convolution product* as the main procedure for each layer. They were originally developed by Yann LeCun and others [LBBH98] in the 1980s. They are now the most widely used algorithms for image recognition tasks [KSH12]. Their capacities have been used in various domains such as autonomous vision [BCC+16] or gravitational wave detection [GH18]. Despite these successes, CNNs suffer from a computational bottleneck that makes deep CNNs resource expensive in practice.

In the following, we will focus on image processing with a tensor framework for all elements of the network. Our goal is to explicitly describe the CNN procedures in a form that can be translated in the context of quantum algorithms. As a regular neural network, a CNN should learn how to classify any input, in our case images. The training consists of optimizing parameters learned on the inputs and their corresponding labels.

Tensor representation

Images, or more generally layers of the network, can be seen as tensors. A tensor is a generalization of a matrix to higher dimensions. For instance, an image of height H and width W can be seen as a matrix in $\mathbb{R}^{H\times W}$, where every pixel is a greyscale value between 0 and 255 (8 bit). However, the three channels of color (RGB: Red Green Blue) must be taken into account, by stacking three times the matrix for each color. The whole image is then seen as a 3 dimensional tensor in $\mathbb{R}^{H\times W\times D}$ where D is the number of channels. We will see that the Convolution Product in the CNN can be expressed between 3-tensors (input) and 4-tensors (convolution *filters* or *kernels*), the output being a 3-tensor of different dimensions (spatial size and number of channels).

Figure 2.9: RGB decomposition, a colored image is a 3-tensor.

Architecture

A CNN is composed of 4 main procedures, compiled and repeated in any order: Convolution layers, most often followed by an Activation Function, Pooling Layers, and some Fully Connected layers at the end. We will denote by ℓ the current layer.

Convolution Layer : The ℓ^{th} layer is convolved by a set of filters called *kernels*. The output of this operation is the $(\ell+1)^{th}$ layer. A convolution by a single kernel can be seen as a feature detector, that will screen over all regions of the input. If the feature represented by the kernel, for instance a vertical edge, is present in some part of the input, there will be a high value at the corresponding position of the output. The output is called the *feature map* of this convolution.

Activation Function : As in regular neural networks, we insert some non-linearities also called *activation functions*. These are mandatory for a neural network to be able to learn any function. In the case of a CNN, each convolution is often followed by a Rectified Linear Unit function, or *ReLu*. This is a simple function that puts all negative values of the output to zero, and lets the positive values as they are.

Pooling Layer : This downsampling technique reduces the dimensionality of the layer, in order to improve the computation. Moreover, it gives the CNN the ability to learn a representation invariant to small translations. Most of the time, we apply a Maximum Pooling or an Average Pooling. The first one consists of replacing a subregion of $P \times P$ elements only by the one with the maximum value. The second does the same by averaging all values. Recall that the value of a pixel corresponds to how much a particular feature was present in the previous convolution layer.

Fully Connected Layer : After a certain number of convolution layers, the input has been sufficiently processed so that we can apply a fully connected network. Weights connect each input to each output, where inputs are all elements of the previous layer. The last layer should have one node per possible label. Each node value can be interpreted as the probability of the initial image to belonging to the corresponding class.

Convolution Product as a Tensor Operation

Most of the following mathematical formulations have been very well detailed in [Wu17]. At layer ℓ, we consider the convolution of a multiple channels image, seen as a 3-tensor $X^\ell \in \mathbb{R}^{H^\ell \times W^\ell \times D^\ell}$. Let's consider a single kernel in $\mathbb{R}^{H \times W \times D^\ell}$. Note that its third dimension must match the number of channels of the input, as in

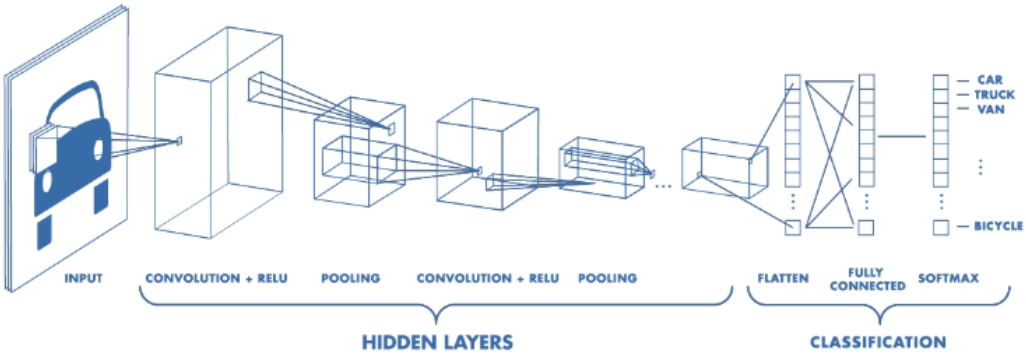


Figure 2.10: Representation of a CNN's layers and operations. Source: Mathworks

Fig.2.11. The kernel passes over all possible regions of the input and outputs a value for each region, stored in the corresponding element of the output. Therefore the output is 2 dimensional, in $\mathbb{R}^{H^{\ell+1} \times W^{\ell+1}}$.

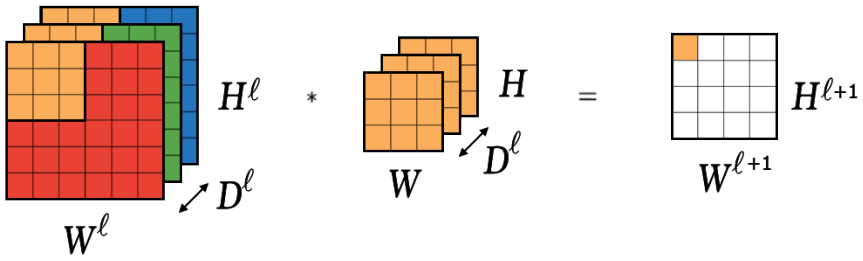


Figure 2.11: Convolution of a 3-tensor input (Left) by one 3-tensor kernel (Center). The ouput (Right) is a matrix for which each entry is a inner product between the kernel and the corresponding overlapping region of the input.

In a CNN, the most general case is to apply several convolution products to the input, each one with a different 3-tensor kernel. Let's consider an input convolved by $D^{\ell+1}$ kernels. We can globally see this process as a whole, represented by one 4-tensor kernel $K^\ell \in \mathbb{R}^{H \times W \times D^\ell \times D^{\ell+1}}$. As $D^{\ell+1}$ convolutions are applied, there are $D^{\ell+1}$ outputs of 2 dimensions, equivalent to a 3-tensor $X^{\ell+1} \in \mathbb{R}^{H^{\ell+1} \times W^{\ell+1} \times D^{\ell+1}}$

This tensor convention explains why Fig.2.10 is represented with layers as volumes of different shapes. Indeed we can see in Fig.2.12 that the output's dimensions are modified given the following rule:

$$\begin{cases} H^{\ell+1} = H^\ell - H + 1 \\ W^{\ell+1} = W^\ell - W + 1 \end{cases} \tag{2.12}$$

We omit to detail the use of *Padding* and *Stride*, two parameters that control how the kernel moves through the input, but these can easily be incorporated in the algorithms.

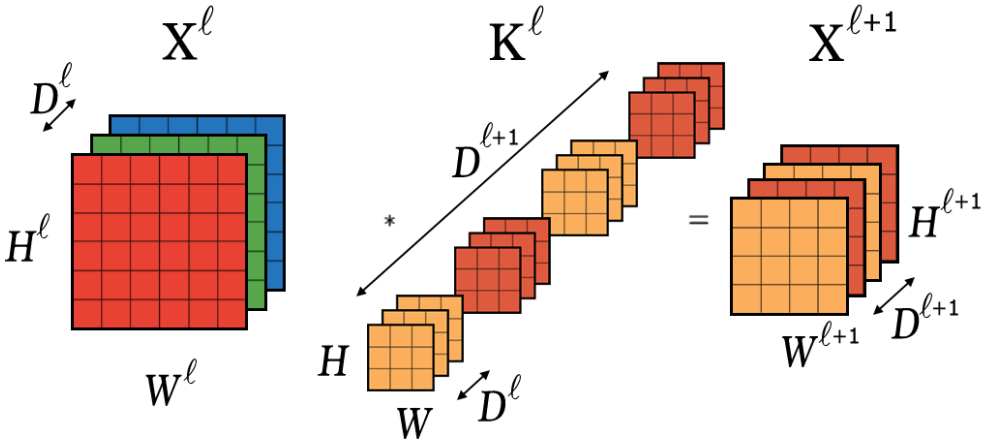


Figure 2.12: Convolutions of the 3-tensor input X^ℓ (Left) by one 4-tensor kernel K^ℓ (Center). Each channel of the output $X^{\ell+1}$ (Right) corresponds to the output matrix of the convolution with one of the 3-tensor kernel.

An element of X^ℓ is determined by 3 indices (i^ℓ, j^ℓ, d^ℓ), while an element of the kernel K^ℓ is determined by 4 indices (i, j, d, d'). For an element of $X^{\ell+1}$ we use 3 indices $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$. We can express the value of each element of the output $X^{\ell+1}$ with the relation

$$X^{\ell+1}_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} = \sum_{i=0}^{H}\sum_{j=0}^{W}\sum_{d=0}^{D^\ell} K^\ell_{i,j,d,d^{\ell+1}} X^\ell_{i^{\ell+1}+i,j^{\ell+1}+j,d} \tag{2.13}$$

Matrix Expression

It is possible to reformulate Eq.(2.13) as a matrix product. For this we have to reshape our objects. We expand the input X^ℓ into a matrix $A^\ell \in \mathbb{R}^{(H^{\ell+1}W^{\ell+1})\times(HWD^\ell)}$. Each row of A^ℓ is a vectorized version of a subregion of X^ℓ. This subregion is a volume of the same size as a single kernel volume $H \times W \times D^\ell$. Hence each of the $H^{\ell+1} \times W^{\ell+1}$ rows of A^ℓ is used for creating one value in $X^{\ell+1}$. Given such a subregion of X^ℓ, the rule for creating the row of A^ℓ is to stack, channel by channel, a column first vectorized form of each matrix. Then, we reshape the kernel tensor K^ℓ into a matrix $F^\ell \in \mathbb{R}^{(HWD^\ell)\times D^{\ell+1}}$, such that each column of F^ℓ is a column first vectorized version of one of the $D^{\ell+1}$ kernels.

As proved in [Wu17], the convolution operation $X^\ell * K^\ell = X^{\ell+1}$ is equivalent to the following matrix multiplication

$$A^\ell F^\ell = Y^{\ell+1}, \tag{2.14}$$

where each column of $Y^{\ell+1} \in \mathbb{R}^{(H^{\ell+1}W^{\ell+1})\times D^{\ell+1}}$ is a column first vectorized form of one of the $D^{\ell+1}$ channels of $X^{\ell+1}$. Note that an element $Y^{\ell+1}_{p,q}$ is the inner product between the p^{th} row of A^ℓ and the q^{th} column of F^ℓ. It is then simple to convert $Y^{\ell+1}$ into $X^{\ell+1}$ The indices relation between the elements $Y^{\ell+1}_{p,q}$ and $X^{\ell+1}_{i^{\ell+1},j^{\ell+1},d^{\ell+1}}$ is

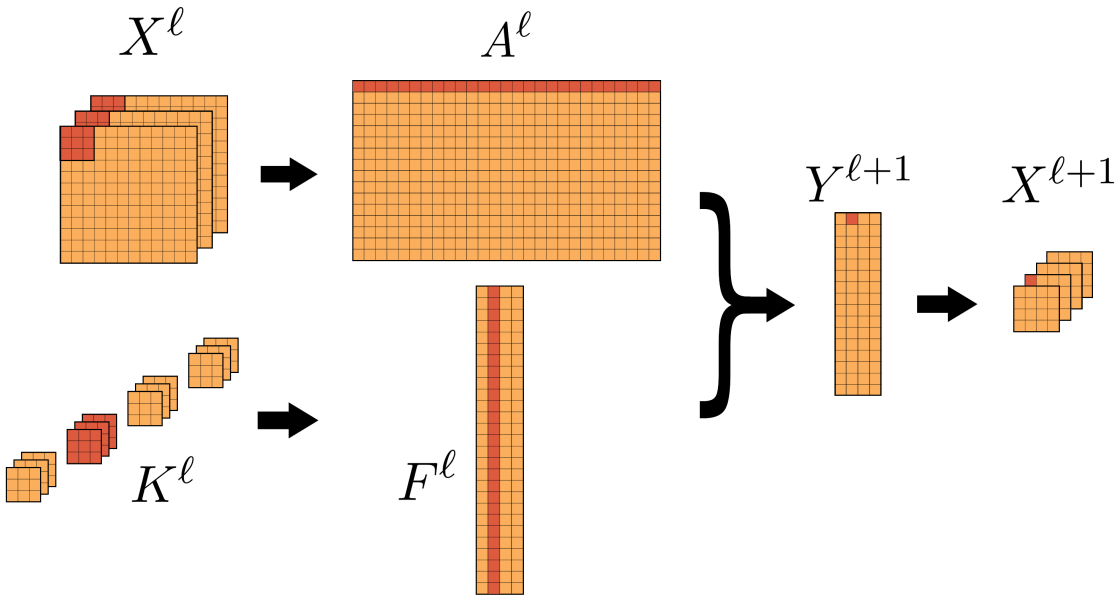


Figure 2.13: A convolution product is equivalent to a matrix-matrix multiplication.

given by:

$$\begin{cases} d^{\ell+1} = q \\ j^{\ell+1} = \lfloor \frac{p}{H^{\ell+1}} \rfloor \\ i^{\ell+1} = p - H^{\ell+1} \lfloor \frac{p}{H^{\ell+1}} \rfloor \end{cases} \tag{2.15}$$

A summary of all variables along with their meaning and dimensions is given in Chapter 11, Table 11.1.

Finally, we can give a running time for one single convolutional layer. From all the routines, the convolution product is the most costly and dominates the rest. With an input tensor of size $H^\ell W^\ell D^\ell$ and $D^{\ell+1}$ kernels of size HWD^ℓ, we produce an output of size $H^{\ell+1}W^{\ell+1}D^{\ell+1}$. We have seen that each pixel of the output was created by applying an inner product between one kernel and a same-size part of the input. Each output's pixel is therefore created in $O(HWD^\ell)$, and the whole convolution product takes:

$$O(H^{\ell+1}W^{\ell+1}D^{\ell+1} \cdot HWD^\ell) \tag{2.16}$$

We can summarize this complexity as:

$$O(\text{output size} \cdot \text{kernel size}) \tag{2.17}$$

2.3.5 Backpropagation for Convolutional Neural Networks

After each forward pass, the outcome is compared to the true labels and a suitable loss function is computed. We can update our weights by gradient descent to minimize this loss, and iterate. The main idea behind the backpropagation is to compute the derivatives of the loss $\mathcal{L}$, layer by layer, starting from the last one.

At layer ℓ, the derivatives needed to perform the gradient descent are $\frac{\partial \mathcal{L}}{\partial F^\ell}$ and $\frac{\partial \mathcal{L}}{\partial Y^\ell}$. The first one represents the gradient of the final loss $\mathcal{L}$ with respect to each kernel element, a matrix of values that we will use to update the kernel weights $F^\ell_{s,q}$. The second one is the gradient of $\mathcal{L}$ with respect to the layer itself and is only needed to calculate the gradient $\frac{\partial \mathcal{L}}{\partial F^{\ell-1}}$ at layer $\ell - 1$.

Convolution Product

We first consider a classical convolution layer without non-linearity or pooling. Thus the output of layer ℓ is the same tensor as the input of layer $\ell + 1$, namely $X^{\ell+1}$ or equivalently $Y^{\ell+1}$. Assuming we know $\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}$ or equivalently $\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}$, both corresponding to the derivatives of the $(\ell+1)^{th}$ layer's input, we will show how to calculate $\frac{\partial \mathcal{L}}{\partial F^\ell}$, the matrix of derivatives with respect to the elements of the previous kernel matrix F^ℓ. This is the main goal to optimize the kernel's weights.

The details of the following calculations can be found in [Wu17]. We will use the notation $vec(X)$ to represents the vectorized form of any tensor X.

Recall that A^ℓ is the matrix expansion of the tensor X^ℓ, whereas Y^ℓ is a matrix reshaping of X^ℓ. By applying the chain rule $\frac{\partial \mathcal{L}}{\partial vec(F^\ell)^T} = \frac{\partial \mathcal{L}}{\partial vec(X^{\ell+1})^T} \frac{\partial vec(X^{\ell+1})}{\partial vec(F^\ell)^T}$, we can obtain (See [Wu17] for calculations details):

$$\frac{\partial \mathcal{L}}{\partial F^\ell} = (A^\ell)^T \frac{\partial L}{\partial Y^{\ell+1}} \tag{2.18}$$

Eq.(2.18) shows that, to obtain the desired gradient, we can just perform a matrix-matrix multiplication between the transposed layer itself (A^ℓ) and the gradient with respect to the previous layer $(\frac{\partial L}{\partial Y^{\ell+1}})$.

Eq.(2.18) explains also why we will need to calculate $\frac{\partial \mathcal{L}}{\partial Y^\ell}$ in order to back-propagate through layer $\ell - 1$. To calculate it, we use the chain rule again for $\frac{\partial \mathcal{L}}{\partial vec(X^\ell)^T} = \frac{\partial \mathcal{L}}{\partial vec(X^{\ell+1})^T} \frac{\partial vec(X^{\ell+1})}{\partial vec(X^\ell)^T}$. Recall that a point in A^ℓ, indexed by the pair (p, r), can correspond to several triplets (i^ℓ, j^ℓ, d^ℓ) in X^ℓ. We will use the notation $(p, r) \leftrightarrow (i^\ell, j^\ell, d^\ell)$ to express formally this relation. One can show that $\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}(F^\ell)^T$ is a matrix of same shape as A^ℓ, and that the chain rule leads to a simple relation to calculate $\frac{\partial \mathcal{L}}{\partial Y^\ell}$ (See [Wu17] for calculations details):

$$\left[\frac{\partial \mathcal{L}}{\partial X^\ell}\right]_{i^\ell, j^\ell, d^\ell} = \sum_{(p,r)\leftrightarrow(i^\ell,j^\ell,d^\ell)} \left[\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}(F^\ell)^T\right]_{p,r} \tag{2.19}$$

We have shown how to obtain the gradients with respect to the kernels F^ℓ and to the layer itself Y^ℓ (or equivalently X^ℓ).

Non Linearity

The activation function has also an impact on the gradient. In the case of the ReLu, we should only cancel the gradient for points with negative values. For points with a positive value, the derivatives remain the same since the function is the identity. A formal relation can be given by

$$\left[\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}\right]_{i^{\ell+1}, j^{\ell+1}, d^{\ell+1}} = \begin{cases} \left[\frac{\partial \mathcal{L}}{\partial f(X^{\ell+1})}\right]_{i^{\ell+1}, j^{\ell+1}, d^{\ell+1}} & \text{if } X^{\ell+1}_{i^{\ell+1}, j^{\ell+1}, d^{\ell+1}} \geq 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.20}$$

Pooling

If we take into account the pooling operation, we must change some of the gradients. Indeed, a pixel that hasn't been selected during pooling has no impact on the final loss, thus should have a gradient equal to 0. We will focus on the case of Max Pooling (Average Pooling relies on similar idea). To state a formal relation, we will use the notations of Section 11.2.3: an element in the output of the layer, the tensor $f(X^{\ell+1})$, is located by the triplet $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$. The tensor after pooling is denoted by $\tilde{X}^{\ell+1}$ and its points are located by the triplet $(\tilde{i}^{\ell+1}, \tilde{j}^{\ell+1}, \tilde{d}^{\ell+1})$. During backpropagation, after the calculation of $\frac{\partial \mathcal{L}}{\partial \tilde{X}^{\ell+1}}$, some of the derivatives of $f(X^{\ell+1})$ should be set to zero with the following rule:

$$\left[\frac{\partial \mathcal{L}}{\partial f(X^{\ell+1})}\right]_{i^{\ell+1}, j^{\ell+1}, d^{\ell+1}} = \begin{cases} \left[\frac{\partial \mathcal{L}}{\partial \tilde{X}^{\ell+1}}\right]_{\tilde{i}^{\ell+1}, \tilde{j}^{\ell+1}, \tilde{d}^{\ell+1}} & \text{if } (i^{\ell+1}, j^{\ell+1}, d^{\ell+1}) \in \mathcal{P} \\ 0 \text{ otherwise} \end{cases} \tag{2.21}$$

where $\mathcal{P}$ is the set of indices selected during pooling.

Chapter 3

Quantum Computing

"Où finit le télescope, le microscope commence. Lequel des deux a la vue la plus grande? Choisissez."

Victor Hugo
Les Misérables (1862)

3.1 Preliminaries in Quantum Computing

We introduce a basic and succinct quantum information background necessary for this thesis. For a more detailed introduction we recommend [NC02, KLM$^+$07, DW19, Chi17].

3.1.1 Quantum Bits and Quantum Registers

The bit is the basic unit of classical information. It can be either in state 0 or 1. Similarly, a quantum bit or *qubit*, is a quantum system that can be in state $|0\rangle$, $|1\rangle$ (the *braket* notation $|\cdot\rangle$ is a reminder that the bit considered is a quantum system) or in a superposition of both states

$$\alpha |0\rangle + \beta |1\rangle \tag{3.1}$$

The coefficients $\alpha, \beta \in \mathbb{C}$, named *amplitudes*, are such that $|\alpha|^2 + |\beta|^2 = 1$. It is also convenient to see this qubit as a unit norm, complex, vector of dimension two, in the *computational basis* (see Fig.4.1a):

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \alpha |0\rangle + \beta |1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{3.2}$$

The amplitudes are linked to the probabilities of observing either 0 or 1 when *measuring* the qubit, since

$$P(0) = |\alpha|^2, \quad P(1) = |\beta|^2. \tag{3.3}$$

Before the measurement, any superposition is possible, which gives quantum information special abilities in terms of computation. With n qubits, the 2^n possible

binary combinations can exist simultaneously, each with a specific amplitude. For instance we can consider an uniform distribution $\frac{1}{\sqrt{n}}\sum_{i=0}^{2^n-1}|i\rangle$ where $|i\rangle$ represents the i^{th} binary combination (e.g. $|01\cdots1001\rangle$). Multiple qubits together are often called a *quantum register*.

In its most general formulation, a quantum state with n qubits can be seen as a vector in a complex Hilbert space of dimension 2^n. This vector must be normalized under ℓ_2-norm, to guarantee that the squared amplitudes sum to 1, to respect the probabilities of measuring each possible state.

With two quantum states or quantum registers $|p\rangle$ and $|q\rangle$, the whole system is written as a tensor product $|p\rangle \otimes |q\rangle$, often simplified as $|p\rangle\,|q\rangle$ or $|p,q\rangle$.

3.1.2 Quantum Computation

To process qubits and therefore quantum registers, we use quantum gates. These gates are *unitary operators* in the Hilbert space as they should map unit-norm vectors to unit-norm vectors. Formally, we can see a quantum gate acting on n qubits as a Hermitian matrix $U \in \mathbb{C}^{2^n}$ such that $UU^\dagger = U^\dagger U = I$, where $U^\dagger$ is the conjugate transpose of U.

There exist plenty of quantum logical gates. For a single qubit, living in a complex 2-dimension space, it is worth mentioning first the Pauli matrices σ_x, σ_y, and σ_z. They have core importance in quantum physics, and together with the identity matrix, they form a basis for all single qubit quantum gates:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{3.4}$$

σ_x is often referred to as the NOT gate, also written X, that inverts $|0\rangle$ and $|1\rangle$. The Hadamard gate, written H, which truly captures the nature of quantum information processing:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.5}$$

Indeed, we can see that H, applied to the computational basis, creates the uniform quantum superposition:

$$H\,|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{3.6}$$

$$H\,|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{3.7}$$

Rotation gates rotate a qubit vector inside the Bloch sphere (see Fig.4.1a), each on a respective plan. These gates take a real value angle as parameter:

$$R_x(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \quad R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix},$$
$$R_z(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{3.8}$$

Multiple qubit gates exist, such as the Controlled-NOT, or $CNOT$, that applies a NOT gate on a target qubit conditioned on the state of a control qubit. As well, the Controlled-Z or CZ gate, flips the phase of the amplitude of the target qubit (σ_z), if the controlled qubit is in state $|1\rangle$:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{3.9}$$

The same controlled gates exist for Rotations and other gates. They can be controlled by more qubits as well. If these gates are not native to a quantum device, they often come at the cost of being decomposed in practice into other gates, adding some depth to the circuit.

An other fundamental and useful gate is the $SWAP$ gate, that swaps two qubits, such that $SWAP\,|p\rangle\,|q\rangle = |q\rangle\,|p\rangle$:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.10}$$

The main advantage of quantum gates is their ability to be applied to a superposition of inputs. Indeed, given a gate U on a quantum state $|x\rangle$, such that:

$$U\,|x\rangle \mapsto |f(x)\rangle \tag{3.11}$$

we can apply it to all possible combinations of $|x\rangle$ at once:

$$U\left(\frac{1}{C}\sum_x |x\rangle\right) \mapsto \frac{1}{C}\sum_x |f(x)\rangle \tag{3.12}$$

where C is is a normalization factor to respect the fact that quantum states must be unit vectors.

3.1.3 Quantum Measurements

Before any intervention, quantum states evolve according to unitary transformations. But there is a moment where one needs to get a result in the classical world. Measuring a quantum state is an action that transfers some information from quantum to classical, but the output can remain quantum if the measurement was made on a part of the state. Measuring a quantum state $|\phi\rangle$ can often result in different outcomes indexed m, symbolized by the set of measurement *operators* $\{M_m\}$. These operators act on the quantum Hilbert space and the probability of obtaining the outcome m is given by:

$$p(m) = \langle\phi|\, M_m^\dagger M_m\,|\phi\rangle \tag{3.13}$$

and the quantum state of the system *after* the measurement is:

$$\frac{M_m\,|\phi\rangle}{\sqrt{\langle\phi|\, M_m^\dagger M_m\,|\phi\rangle}} \tag{3.14}$$

The denominator of this fraction appears to renormalize the output state. A *complete* set of measurement operators is such that $\sum_m M_m^\dagger M_m = I$, where I is the identity operator.

The most common measurement operators are the ones in the *computational basis*, where we want to know in which computational state qubits are. For a single qubit and its basis $\{|0\rangle,|1\rangle\}$, we write $M_0 = |0\rangle\langle 0|$, $M_1 = |1\rangle\langle 1|$, and the associated probability is the respective square of the amplitude.

More generally one can define *POVM* (Positive Operator Valued Measure) and extend their applications to mixtures of quantum states known as density matrices.

And in practice, we most commonly use *Projective* measurements, which are derived from *observables*. An Hermitian measurement operator M can be decomposed in m projections P_m. Each P_m is a projection into an eigenspace of M with eigenvalue m. The projectors are orthogonal between each other, complete, positive definite, and Hermitian.

$$M = \sum_m mP_m \tag{3.15}$$

Since for a projector P we have $P = P^\dagger$ and $P^2 = P$, the probability of obtaining the state m is now given by:

$$p(m) = \langle\phi| P_m |\phi\rangle \tag{3.16}$$

and the remaining state is:

$$\frac{P_m |\phi\rangle}{\sqrt{\langle\phi| P_m |\phi\rangle}} \tag{3.17}$$

Notably, the average outcome of a projective measurement M is the expectation value:

$$\begin{aligned}\mathbb{E}(M) &= \sum_m mp(m) \\ &= \sum_m m \langle\phi| P_m |\phi\rangle \\ &= \langle\phi| (\sum_m mP_m) |\phi\rangle \\ &= \langle\phi| M |\phi\rangle\end{aligned} \tag{3.18}$$

Finally, we will use the following facts:

- When asking the question " what is the probability of measuring $|\psi\rangle$ from quantum state $|\phi\rangle$?" we use $p(\psi) = |\langle\phi|\psi\rangle|^2$. To prove this, we use the projector $|\psi\rangle\langle\psi|$.
- Two quantum states $|\phi\rangle$ and $|\psi\rangle$ are said *orthogonal* is $\langle\phi|\psi\rangle = 0$.
- For tensor product of quantum states, linearity prevails: $(\langle\phi| \otimes \langle\phi'|)(|\psi\rangle \otimes |\psi'\rangle) = \langle\phi|\psi\rangle\langle\phi'|\psi'\rangle$. We wan write $|\psi^\perp\rangle$ to denote a quantum state orthogonal to $|\psi\rangle$.

- We will often have state in the form $|\phi\rangle = \alpha |y\rangle |0\rangle + \beta |y^{\perp}\rangle |1\rangle$. The probability of measuring $|0\rangle$ on the last qubit is then given by:

$$\langle\phi| (|0\rangle \langle 0|) |\phi\rangle = \alpha^2 \tag{3.19}$$

  and the remaining state after the measurement is simply $|y\rangle$.

3.2 Quantum Algorithms

Joined together, quantum gates form quantum circuits, also called quantum algorithms, and are often represented as in Fig.3.1. The complexity of a quantum circuit can be expressed as the relationship between its depth and the size of the problem. The complexity also takes into account the number of time the circuit must be run to obtain the desired output.

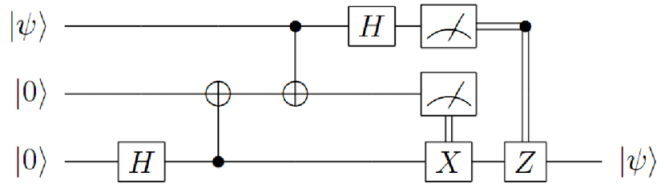


Figure 3.1: Quantum circuit for the teleportation of any quantum state $|\psi\rangle$ [BBC+93]. Each wire corresponds to one or several qubits. The input on the left is processed through the circuit. The circuit uses Hadamard, CNOT, Pauli σ_x and σ_z gates. It also requires measurements and classical information flow.

We already presented the first meaningful quantum algorithms in Section 1.1. Among them, the Deutsch-Josza algorithm [DJ92], the Bernstein-Vazirani algorithm [BV97], Simon's algorithm [Sim97], and the famous Grover [Gro96] and Shor's algorithms [Sho99]. In this section, we will introduce a few more quantum algorithms that will be used as subroutines in this thesis. The algorithms are presented in the form of theorems specifying the error and running time guarantees. We omit the proof of the theorems of this section (see [NC02, KLM+07, DW19, Chi17] for details).

3.2.1 Phase Estimation

Phase estimation [Kit95] is an important quantum algorithm that creates the link with linear algebra. It is also a subroutine used in Shor's algorithm [Sho99], the HHL algorithm[HHL09], and amplitude estimation (see Section 3.2.2). Phase estimation is itself based on the Quantum Fourier Transform (QFT) algorithm. The QFT provides an exponential speedup in the task of mapping a vector, encoded as a quantum state, into the Fourier space.

Basically, the goal of phase estimation is to extract the eigenvalues of a matrix U. In fact, this matrix must be a quantum circuit U, therefore it should be an unitary operator. We denote its eigenvectors by $|u_j\rangle$ and eigenvalues $e^{i\theta_j}$, hence $U |u_j\rangle = e^{i\theta_j} |u_j\rangle$. Given a eigenvector and an extra register $|u_j\rangle |0\rangle$ as input, the

algorithm should return $|u_j\rangle |\theta_j\rangle$. Since the eigenvectors compose a basis, any state $|\psi\rangle$ can be written as $|\psi\rangle = \sum_{j\in[n]} \alpha_j |u_j\rangle$. Therefore, the interesting feature of phase estimation is to apply in superposition. the circuit is shown in Fig.3.2.

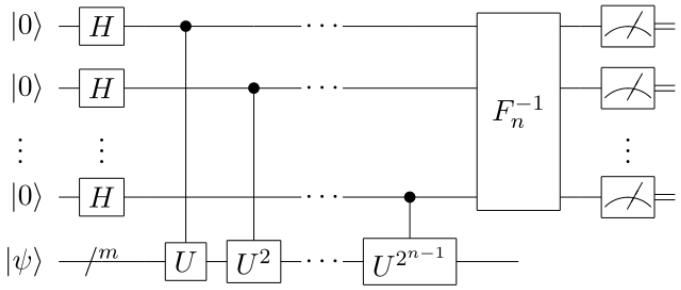


Figure 3.2: The phase estimation circuit for a unitary U and any input $|\psi\rangle$, with measurements at the end. F_n^{-1} is the inverse QFT. Source: Wikipedia

Theorem 3.1: Phase Estimation

Let U be a unitary operator than runs in time $T(U)$, with eigenvectors $|u_j\rangle$ and eigenvalues $e^{i\theta_j}$ for $\theta_j \in [-\pi, \pi]$, for $j \in [N]$. For a precision parameter $\epsilon > 0$, there exists a quantum algorithm that runs in time $O(T(U)\log(N)/\epsilon)$ and with probability $1 - 1/poly(N)$ performs the mapping

$$|\psi\rangle = \sum_{j\in[N]} \alpha_j |u_j\rangle |0\rangle \mapsto \sum_{j\in[N]} \alpha_j |u_j\rangle |\overline{\theta_j}\rangle \tag{3.20}$$

where $\overline{\theta_j}$ is approximating θ_j with guarantee $|\overline{\theta_j} - \theta_j| \leq \epsilon$ for all $j \in [N]$.

One can decide to measure the result at the end to recover a classical description of the eigenvalues, or to leave the output as a quantum state for further processing. Note however that this circuit requires to apply sequentially the unitary U in a controlled fashion. For certain circuits U, this can become a costly operation.

3.2.2 Amplitude Amplification and Amplitude Estimation

As stated before, one of the major difficulties in quantum computing is to manipulate the amplitudes of the quantum states. These amplitudes often carry important information, in particular in quantum linear algebra (see Chapter 4). It also happens that a part of the quantum superposition is considered as "garbage" and that one wants to discard it. Finally, one may need to recover a classical value of a target amplitude.

Following the Grover algorithm [Gro96], a generalisation was proposed in [BH97] and then [BHMT02] to amplify a target amplitude of a quantum state. This has proven to be very useful in many cases. It can also output an amplitude or writing it in a quantum register, and become very powerful when applied in superposition. As Grover algorithm, these algorithms usually provides a quadratic speedup (see Table 3.1).

Theorem 3.2: Amplitude Amplification

Given the ability to implement a quantum unitary U and U^{-1}, such that $U|0\rangle = \sin(\theta)|x,1\rangle + \cos(\theta)|G,0\rangle$, where $|G\rangle$ is a garbage state, then we can instead create the state $|x\rangle$ in time $\widetilde{O}(\frac{T(U)}{\sin(\theta)})$, where $T(U)$ is the time to implement U and U^{-1}.

Theorem 3.3: Amplitude Estimation (1)

Given the ability to implement a quantum unitary U and U^{-1}, such that $U|0\rangle = \sin(\theta)|x,1\rangle + \cos(\theta)|G,0\rangle$, where $|G\rangle$ is a garbage state, then $\sin(\theta)$ can be estimated to multiplicative error $\epsilon > 0$ in time $\widetilde{O}(\frac{T(U)}{\epsilon\sin(\theta)})$, or to additive error in $\widetilde{O}(\frac{T(U)}{\epsilon})$, where $T(U)$ is the time to implement U and U^{-1}.

In this thesis, we will also use a specific version of this algorithm [Gro05, YLC14] where the amplitudes don't have to be known in advance to be estimated.

Theorem 3.4: Amplitude Estimation (2)

Given the ability to implement a quantum unitary U and U^{-1}, such that $U : |0\rangle \to \sqrt{p}|y,1\rangle + \sqrt{1-p}|G,0\rangle$ where $|G\rangle$ is a garbage state, then for any positive integer P, the amplitude estimation algorithm outputs $\tilde{p}$ $(0 \leq \tilde{p} \leq 1)$ such that

$$|\tilde{p} - p| \leq 2\pi\frac{\sqrt{p(1-p)}}{P} + \left(\frac{\pi}{P}\right)^2 \tag{3.21}$$

with probability at least $8/\pi^2$. It uses exactly P iterations of the algorithm U and U^{-1}. If $p = 0$ then $\tilde{p} = 0$ with certainty, and if $p = 1$ and P is even, then $\tilde{p} = 1$ with certainty.

Proper proofs of these theorems are given in [KLM+07]. Briefly, let U be the unitary that creates the state $\sqrt{p}|y,1\rangle + \sqrt{1-p}|G,0\rangle$. Amplitude amplification or estimation is phase estimation (Section 3.2.1) applied on a the unitary $Q = U^{-1}O_\perp U O_f$ where $O_\perp$ and O_f are the Grover phase shift operators. One can show that the eigenvalues of the operator Q which are estimated by phase estimation are linked to the desired amplitudes.

It is also possible to obtain the amplitude as a quantum state, written in binary with some precision ϵ, as $|\sqrt{p}\rangle$ or $|\sin(\theta)\rangle$ with some amplitude $\sqrt{\alpha}$ such that $\alpha > 8/\pi^2$. For this it suffices to not perform the measurement at the end of the phase estimation (see Section 3.2.1). We will refer indistinctly to "amplitude estimation" and Theorem 3.4 for both usages, and often for the additive error case (see Table 3.1). Later on, we will also use Theorem 6.2 to boost the amplitude $\sqrt{\alpha}$ and have a state arbitrary close to $|\sqrt{p}\rangle$ or $|\sin(\theta)\rangle$.

Note that both amplitude amplification and estimation rely on phase estimation (see Section 3.2.1) and therefore can suffer from the same constraints. However recently, proposals for amplitude estimation without phase estimation have been made [AR20, SUR+20] and could better suit short term implementations for which shallow circuits are required [GTKL+20].

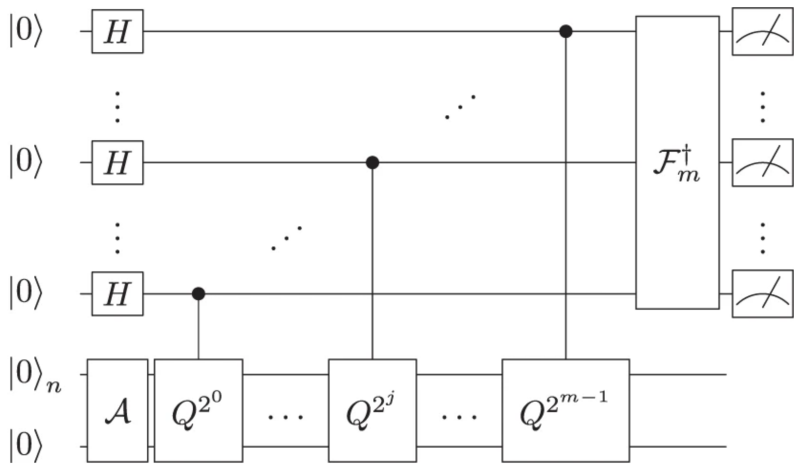


Figure 3.3: The amplitude estimation circuit, where $\mathcal{F}$ is the Quantum Fourier Transform, and $\mathcal{A}$ is the initial unitary (denoted U in above Theorems). Source: [GGZW21].

Type	Guarantee	Quantum	Classical
additive error	$\lvert\tilde{p}-p\rvert < \epsilon$	$O\left(1/\epsilon\right)$	$O\left(1/\epsilon^2\right)$
relative error	$\lvert\tilde{p}-p\rvert < \epsilon p$	$O\left(1/\epsilon\sqrt{P(0)}\right)$	$O\left(1/\epsilon^2 P(0)\right)$

Table 3.1: Comparison of classical and quantum amplitude estimation in the additive and relative cases.

3.2.3 Other Subroutines

Classical Boolean Circuits

In the following claim we state some primitive quantum circuits, which we will use in our algorithm. They are basically quantum circuits with a reversible version of the classical boolean ones.

Using quantum circuits, one can perform the following operations in time linear in the number of qubits used to encode the input values :

Claim 3.1: Classical Boolean Circuits

- For two integers i and j, we can check their equality with the mapping $|i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle|[i=j]\rangle$.
- For two real numbers $a > 0$ and $b > 0$, we can compare them using $|a\rangle|b\rangle|0\rangle \mapsto |a\rangle|b\rangle|[a \leq b]\rangle$.
- For a real number $a > 0$, we can obtain its square $|a\rangle|0\rangle \mapsto |a\rangle|a^2\rangle$.

This can be extended since any classical boolean function can be embedded in a quantum circuit (see Section 3). In particular, non linear functions ($\arcsin(x)$, $\sqrt{x}$, $\text{sigmoid}(x)$ etc.) can be applied to the value encoded in binary quantum registers, as we will do in Part IV. These functions can be implemented using Taylor decomposition or any other technique. Note however that non linear transformations are impossible on quantum amplitudes directly, which require unitary, thus linear, transformations.

Conditional rotation

Conditional rotation is a convenient and short procedure, used in the HHL algorithm [HHL09] for instance, and throughout this thesis. In contrast to amplitude estimation (Theorem 3.4), the goal is to map a value, binary encoded in a quantum register, to the amplitude of an extra qubit. Therefore, this value should be in $[-1, 1]$.

Theorem 3.5: Conditional Rotation

Given the quantum state $|a\rangle$ encoded in q qubits, with $a \in [-1, 1]$, There is a quantum circuit to perform $|a\rangle|0\rangle \mapsto |a\rangle\left(a|0\rangle + \sqrt{1-a^2}|1\rangle\right)$.

Proof. Let $\gamma = \arcsin(a)$. The controlled rotation starts by writing the state $|\gamma\rangle$ in a q qubits register. This can be done using a quantum implementation of the arcsin function, as in Claim 3.1 or any other [HRS18]. Quantum circuits are classical boolean operations, which usually apply part of the arcsine polynomial decomposition from the Taylor's series. This can require $O(poly(q))$ or less, depending on the solution adopted.

$$|a\rangle|0\rangle|0\rangle \mapsto |a\rangle|0\rangle|\gamma\rangle \tag{3.22}$$

The second step is the controlled rotation itself by performing a series of controlled rotation gates along the y-axis, for each one of the q qubits of $|\gamma\rangle$. If we write the binary expansion $\gamma = 0.\gamma_1 \cdots .\gamma_q$, we can write the unitary that performs the rotation (see Fig.3.4) $R_y(2\gamma) = \prod_{j=1}^{q} R_y^{\gamma_j}(2^{1-j})$:

$$|a\rangle|0\rangle|\gamma\rangle \mapsto |a\rangle\left(\sqrt{1-a^2}|0\rangle + a|1\rangle\right)|\gamma\rangle \tag{3.23}$$

Finally, we can get rid of $|\gamma\rangle$ by reverting the circuit, and we can switch $|0\rangle$ and $|1\rangle$ using a NOT gate. □

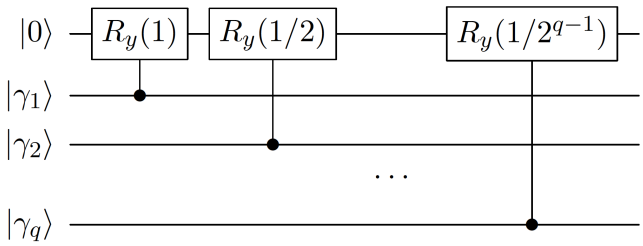


Figure 3.4: First half of a circuit implementing conditional rotation.

In addition, knowing in advance or computing an upper bound $\max(a)$ for the value of a would allow applying the conditional rotation to values whose absolute value is bigger than 1, that is:

$$|a\rangle|0\rangle \mapsto |a\rangle\left(\frac{a}{\max(a)}|0\rangle + \sqrt{1-\frac{a}{\max(a)}}|1\rangle\right) \tag{3.24}$$

Using Theorem 3.5 followed by Theorem 3.4, it then possible to transform the state $\frac{1}{\sqrt{d}}\sum_{j=0}^{d-1}|x_j\rangle$ into $\frac{1}{\|x\|}\sum_{j=0}^{d-1}x_j|x_j\rangle$ and therefore alternate between the encoding types of a vector $x \in \mathbb{R}^d$ (see Chapter 4).

3.3 Noisy Intermediate Scale Quantum Computing (NISQ)

The algorithms presented in the previous Section are the continuation of the first results proving the theoretical superiority of quantum computing. Therefore, they all assume access to an ideal quantum computer, without decoherence, gate noise, and qubit errors.

In recent years, we witnessed the advent of the first noisy quantum computers, up to the first quantum *supremacy* experiment [AAB+19]. Computer scientists and physicists tried to develop quantum algorithms that would suit these "noisy intermediate scale quantum" devices, or NISQ for short [Pre18].

Several approaches exist, but the one that has attracted the most attention of researchers is called *variational* quantum circuits (VQC) [CAB+20, BCLK+21]. Inspired by classical machine learning, it was proposed for quantum chemistry with the variational quantum eigensolver (VQE) algorithm [PMS+14], and for optimization with the quantum approximate optimization algorithm (QAOA) [FGG14]. Later, a lot of derived applications in machine learning [BWP+17, CCL19, CMDK20].

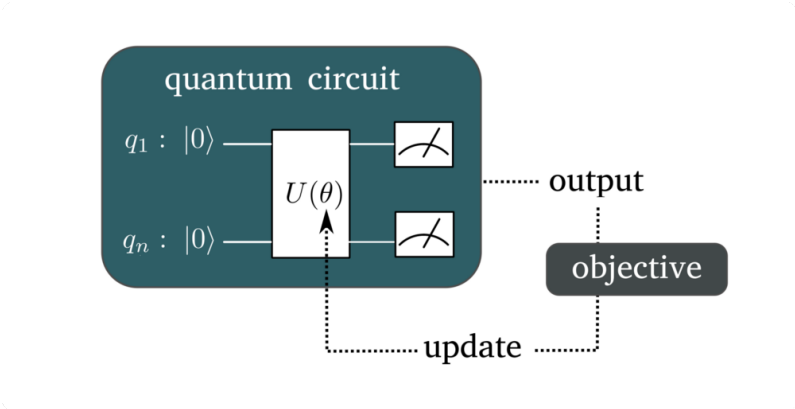


Figure 3.5: Representation of variational quantum circuit optimization scheme. Source: Xanadu

VQC have universal properties [Bia21] and already show encouraging results on real experiments [ASZ+20], however they are very different in nature. They are based on the following scheme (see Fig.3.5): One defines a small circuit, called the *ansatz*, made of many gates with tunable parameters, such as the angle of a rotation gate. Then, measurements of the resulting quantum state are performed and should give the right answers to the desired task (classification, regression). At first, the results are bad because the parameters are almost random. This metric is called the Objective Function or the Loss. Finally, optimization is done on a classical computer to propose a new and hopefully better set of parameters to try. And we repeat this loop until the circuit gives good results.

The main difference with previous quantum algorithms is that the circuit is not implementing a known classical ML algorithm. One would simply hope that the heuristic will converge and successfully classify data or predict values, and even more when quantum machines will become larger.

Researchers hope that VQC would project data in large enough Hilbert space,

to perform classically inaccessible correlations or separations. Notably, research on variational quantum machine learning is less focused on proving computational speedups. The main interest is to reach a more expressive or complex state of information processing. Despite the excitement, VQC also suffers from theoretical disturbance. It is proven that when the number of qubits or the number of gates becomes too big, the optimization landscape will be flat and hinder the ability to optimize the circuit. Many efforts are made to circumvent this issue, called barren plateaus [MBS+18], by using specific circuits [PCW+20] or smart initialization of the parameters [GWOB19]. These barren plateaus may be very fundamental in quantum information, as a deep link has been recently proven with quantum information scrambling and limitations for the Hayden-Preskill thought experiment on black holes information loss [HAY+20].

in a VQC, the gradients of a cost function with respect to each parameter have to be estimated. In classical neural networks, this is usually done using the backpropagation algorithm 2.3.2 over analytic operations. With VQC, operations become too complex, and we cannot access intermediate quantum states, without measuring them. The current state-of-the-art solution is called the parameter shift rule [MNKF18, SBG+19] and requires applying the circuit and measure its result 2 times for each parameter. By comparison, in classical deep learning, the network is applied just once forward and once backward to obtain all thousand or millions gradients. Hopefully, we could parallelize the parameter shift rule on many simulators or quantum devices, but this could be limited for a large number of parameters.

Finally, researchers tend to focus more and more on the importance of data loading into a quantum state [JDM+20], also called feature map [Sch21]. Without the ideal amplitude encoding obtained with the QRAM (see Chapter 4), there are doubts that we will be able to load and process high dimensional classical data with an exponential or high polynomial factor.

Note that the expression "Quantum Neural Networks" has been used to show the similarities with classical Neural Networks (NN) training. However they are not equivalent, since the VQC don't have the same hidden layers architecture, and neither have natural non-linearities, unless a measurement is performed. And there's no simple rule to convert any neural network to a VQC or vice versa.

In Chapter 12, we will propose an alternative NISQ algorithm for neural network implementation, and backpropagation, with exact equivalence.

3.4 How to Test a Quantum Algorithm?

Throughout this thesis, quantum algorithms will be developed theoretically. But actual experiments or simulations are necessary to judge, or at least gain intuition about the results of the algorithms. This becomes even more crucial when comparing to equivalent classical algorithms. Indeed, by the nature of quantum information, precision, noise, and randomness often arise, which can provide undesirable effects.

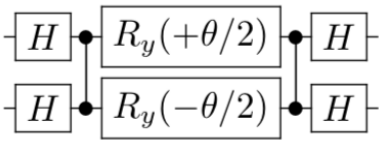
However, as detailed in Section 3.2, we will often consider algorithms that current and near term quantum computers would not support, due to the lack of error correction and qubit number. In the next sections, we will explain the three methods

we used for testing our quantum algorithms and provide meaningful results.

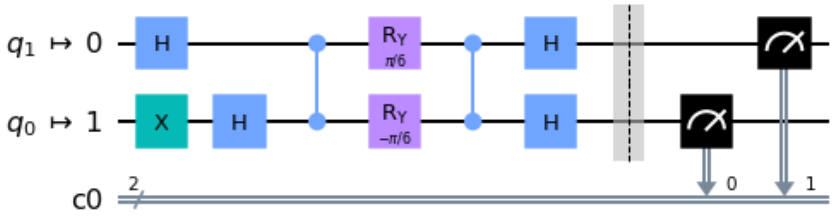
3.4.1 Real Quantum Computers and Emulators

The most conclusive experiment will always be to run the quantum circuit on actual hardware. For a few years, it becomes possible via cloud access to run quantum circuits on real quantum computers. The emergence of open-source quantum software from various institutions and companies allow to program easily quantum circuit and launch experiments.

In Chapter 12, we used several quantum computers made by IBM, ranging from 5 to 16 qubits (see Fig.3.7), for our orthogonal neural network algorithm. Indeed, this algorithm has the advantage of being shallow, repetitive, and requires only adjacent connectivity between qubits.



(a)



(b)

Figure 3.6: (a) Decomposition of the $RBS(\theta)$ gate (see Chapter 12) with Hadamard, y-axis Rotation, and CZ gates. (b) Quantum circuit implementation using Qiskit [A+19], with additional X gate to start in '10' and measurement on both qubits at the end.

However, the current state of these quantum computers makes them only interesting for proof of concepts and reality check. In practice, the qubits are noisy and prone to errors when applying gates and measuring qubit states. Still, it allows to learn the real constraints of quantum computing and to take into account the connectivity between the qubits, their quality, the noxious depth, and of course the monetary cost of such experiments. Moreover, current standard access is limited to few qubits, which is certainly too little to achieve quantum advantage.

It is however possible to move to a more ideal world, using emulators. Emulators are classical computers implementing the quantum circuit, by actually storing the exponentially large number of amplitudes and transforming it gate by gate. The main advantage of this method is to get rid of real hardware noise and augment the number of qubits. However, this number is often limited to 30 or 40 qubits for a general use case, since 2^{40} amplitudes with 32 bit floating point precision is already several Terabytes of data to process. Note that for some specific quantum circuits, efficient classical emulations can be found, which increases the number of

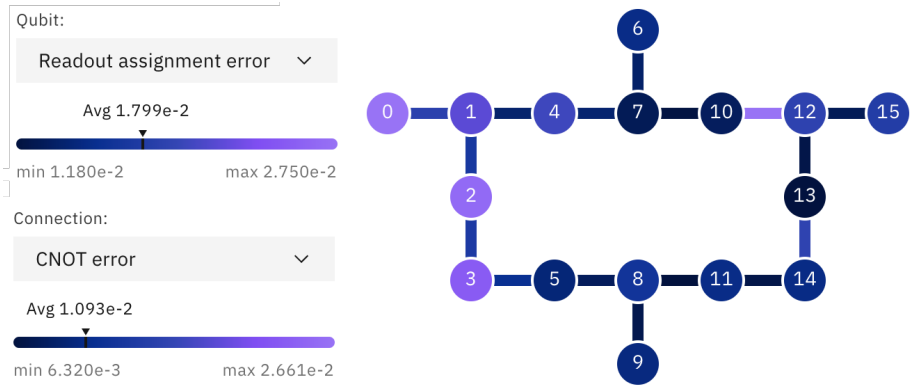


Figure 3.7: The *IBM Guadalupe* quantum computer of 16 qubits. Visualization of the connectivity (right), and the calibrated error parameters(left). Qubits are connected with their adjacent neighbors, and only 3 qubits have 3 connections. May 2021.

qubits emulated.

In Fig.3.8, we see the results of 8192 measurements of the circuit shown in Fig.3.6b. We see in Fig.3.8a, the theoretical emulated result, that states '00' and '11' are not present in the quantum superposition and therefore should not be measured. However, on the real 5 qubits *IBM Santiago* quantum computer, these states are effectively measured because of errors in gates, qubits, or readout.

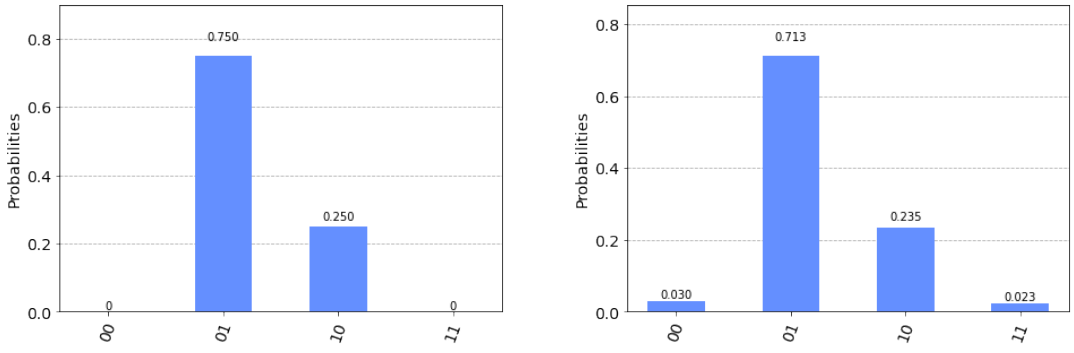
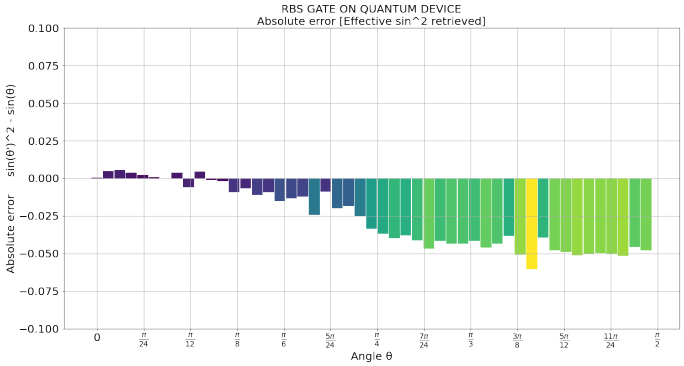


Figure 3.8: Measurements results between the ideal (a) and the real (b) experiment of the RBS gate. This experiment took place on the 5 qubits *IBM Santiago* quantum computer in May 2021.

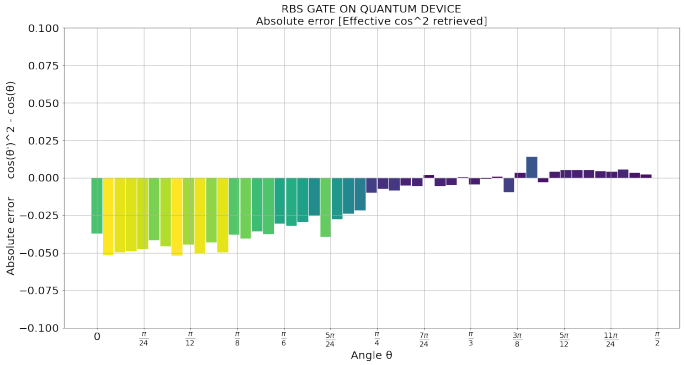
As a result, it adds imprecision to the estimation of the amplitudes. In the case of the $RBS(\theta)$ gate, one expects to recover respectively $\cos(\theta)$ and $\sin(\theta)$ by measuring the relative size of the histogram bar '01' and '10'. In Fig.3.9, we have computed this error, for different angles $\theta \in [0, \pi/2]$,

3.4.2 Classically Simulating Quantum Algorithms

Whether it is with 16 qubits on real hardware or 40 qubits on an emulator, we often require more. Indeed, for complex algorithms as most of those presented in this



(a)



(b)

Figure 3.9: Absolute error in the estimation of $\cos^2(\theta)$ and $\sin^2(\theta)$ from the measurements of the output of the $RBS(\theta)$ gate. We see that this error is correlated to the parameter θ, making error mitigation more complex. This experiment took place on the 5 qubits *IBM Santiago* quantum computer in May 2021.

thesis, or for bigger size problems, a small number of qubits is often far from being useful.

The advantage of the quantum machine learning algorithms we propose is their equivalence with their classical version. Indeed, as we will see, our algorithms follow the same steps, use the same inputs and outputs. The quantum circuits differ a lot, but we can mathematically describe their effects and we control the error committed, or the randomness during measurement.

It is then possible to simply adapt programs, for instance in Python, that implement the classical machine learning algorithms. Since we control how and where the differences will occur, we can as well modify the classical programs to include them. With this, we can simulate our ideal fault-tolerant quantum algorithms in Python.

Part II

Quantum Linear Algebra For Machine Learning

Chapter 4

Quantum Data

"There is no difference between Theory and Practice, except in Practice."

Benjamin Brewster (1882)

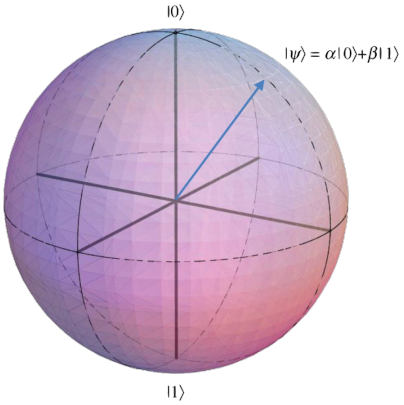
In this chapter, we will cover the numerous techniques to interface quantum algorithms with classical data. As seen in Chapter 2, most machine learning relies on datasets containing samples $\{x_i\}_i \in [N]$, with $x_i \in \mathbb{R}^d$ for all i. These samples come from external experiments or data mining. As well, one needs classical outputs such as classes or labels $\{y_i\}_i \in [N]$, with $y_i \in \mathbb{R}^{d'}$ for all i. It seems that, for most *real life* applications of quantum machine learning or optimization, it will be mandatory to have a way to handle classical data [CB18].

In Section 4.1, we will see different propositions for encoding classical data as quantum states, and how we can generate them efficiently using *quantum memory models*. Then, in Section 4.2, we will present the inverse task: recovering classical data from a quantum state.

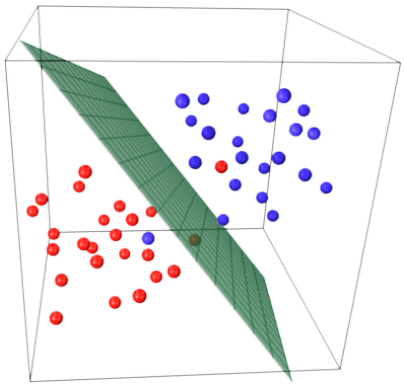
Being able to propose such methods, even though some are only suited for perfect quantum computers (FTQC), is key to understand the potential benefit from near term and long term quantum computers.

Note that we will not cover the field on quantum machine learning on *quantum data*. This is the case when the input of an algorithm is already a quantum state, as it could be in quantum communication or cryptography [CDKK20]. As well, efforts are made for developing classical machine learning for quantum data [DB18]. We will not cover Grover based quantum memory models, such as the *Quantum Associative Memory* [dPNdSdOL19].

Finally, as the translation between classical and quantum data remains a challenge in practice, there is interest in looking for "dataless" problems. They could be problems that only involve an environment, such as solving partial differential equations, training a reinforcement learning algorithm, generate data or sampling from peculiar distributions, or even some chemistry applications.



(a)



(b)

Figure 4.1: (a) A quantum state (blue vector) of one qubit is a unit vector in the Bloch sphere, embedded in a 2 dimensional Hilbert Space. Source: [DGV12] (b) Classical machine learning also manipulates data as vectors (blue and red dots) in vector spaces to classify or transform them. Source: *Machine Learning in Action*.

4.1 From Classical to Quantum Data

4.1.1 Quantum Encodings

In the following, numbers are given in their decimal (N_{10}) or binary (N_2) basis. If not specified, the basis shall be the decimal one. Unless otherwise specified, we will denote by $|i\rangle$ the i^{th} quantum state in the computational basis, e.g. the state $|0110\cdots10\rangle$ that corresponds to the binary representation of the number i.

Note that in the rest of this thesis, except Chapter 12, the *amplitude encoding* described below will be used by default.

Bit Encoding

Classical data is naturally encoded as bits, e.g. $x = 5_{10}$ and $y = 3.25_{10}$ can be respectively written in binary as $x = 101_2$ and $y = 11.01_2$. Therefore, it seems natural to first propose a similar simple *bit encoding* using qubits, e.g. $|x\rangle = |101\rangle$. Similarly, a vector $x = \begin{pmatrix} 3_{10} \\ 2_{10} \end{pmatrix} = \begin{pmatrix} 11_2 \\ 10_2 \end{pmatrix}$, can be encoded as the quantum state $|x\rangle = |11\rangle \otimes |10\rangle$ or $|1110\rangle$.

For a small number of data points, or for low precision values, the loading of classical data known in advance is simple, using simple NOT gates on qubits where $|1\rangle$ should be. This encoding is however poorly efficient as it requires as many qubits as bits, and has a limited precision.

In the quantum regime, it possible to use quantum superposition to handle multiple numbers at the same time. We can then propose the following bit encoding for a vector:

Definition 4.1: Bit Encoding

$m + \lceil \log(d) \rceil$ qubits can encode a vector $x = (x_1, \cdots, x_d) \in \mathbb{R}^d$ as a quantum superposition of bit strings:

$$\frac{1}{\sqrt{d}} \sum_{j=1}^{d} |j\rangle |x_j\rangle \tag{4.1}$$

where m is the number of qubits used for the precision, and $|j\rangle$ the j^{th} quantum state in the computational basis.

Amplitude Encoding

To use qubits to represent classical vectors or matrices, the most efficient encoding is by far the amplitude encoding scheme. It is the theoretical link between quantum computing and linear algebra that exploits quantum properties to the maximum.

We will see that obtaining the amplitude encoding is usually the main bottleneck to our quantum algorithms (see Sections 4.1.2 and 4.2.1), while the rest consists in playing with the amplitudes. In fact, amplitude encoding was used in the pioneering work of [HHL09], and later in many quantum machine learning and linear algebra works.

Definition 4.2: Amplitude Encoding

$\lceil \log(d) \rceil$ qubits can encode a vector $x = (x_1, \cdots, x_d) \in \mathbb{R}^d$ using the amplitudes of the quantum state:

$$|x\rangle = \frac{1}{\|x\|_2} \sum_{j=1}^{d} x_j |j\rangle \tag{4.2}$$

where $|j\rangle$ is the j^{th} quantum state in the computational basis

Since the quantum state $|x\rangle$ must be of unit norm in the Hilbert space, we use the normalization factor $1/\|x\|_2$, which is equivalent to having a normalized input such that $\|x\|_2 = 1$. Amplitude encoding uses only $\lceil \log(d) \rceil$ qubits: high dimensional data can be encoded with a small number of qubits, hence the exponential advantage.

To go further, one can encode N such vectors simultaneously, which is equivalent to encoding a matrix $X \in \mathbb{R}^{N \times d}$, as:

$$|X\rangle = \frac{1}{\|X\|_F} \sum_{i=1}^{N} \|X_i\|_2 |X_i\rangle |i\rangle \tag{4.3}$$

As in Definition 4.2, $|X_i\rangle$ is the quantum state of X_i, the i^{th} row of X. $\|X\|_F$ is the Frobenius norm of the matrix X. Note that this state is still normalized as $\|X\|_F = \sqrt{\sum_{i=1}^{N} \|X_i\|_2^2}$, which is permitted since all states in the superposition are orthogonal to each other thanks to the registers $|i\rangle$ at the end.

Finally, we present a methodology to switch from a superposition using bit encoding to a superposition using amplitude encoding.

Claim 4.1: From Bit Encoding to Amplitude Encoding

Given an unitary U which takes a ground state and creates in time T_U the bit encoding quantum state of the vector $x = (x_1, \cdots, x_d)$, we can perform the following mapping:

$$|0\rangle \overset{U}{\mapsto} \frac{1}{\sqrt{d}} \sum_{j=1}^{d} |j\rangle |x_j\rangle \mapsto \frac{1}{\|x\|_2} \sum_{j=1}^{d} x_j |j\rangle \tag{4.4}$$

in time $\widetilde{O}(T_U \eta^2 / \mathbb{E}(x_j^2))$, where $\eta \geq \max(x_j)$, and $\mathbb{E}(x_j^2)$ is the average value of the square components of x.

Proof. We start with a conditional rotation on the bit encoding state, see Theorem 3.5, and obtain:

$$\frac{1}{\sqrt{d}} \sum_{j=1}^{d} |j\rangle |x_j\rangle |0\rangle \mapsto \frac{1}{\sqrt{d}} \sum_{j=1}^{d} |j\rangle |x_j\rangle \left(\frac{x_j}{\eta} |0\rangle + \sqrt{1 - \frac{x_j^2}{\eta^2}} |1\rangle \right) \tag{4.5}$$

where η is an upper bound of $\{x_j\}_{j \in [d]}$, or simply $\eta = 1$ if x is normalized. It would then suffice to measure the ancilla bit in the state $|0\rangle$ to end up with the desired state, with probability $P(0)$. This can also be done using amplitude amplification (Theorem 3.2). This second step has a complexity of $O(1/\sqrt{P(0)})$:

$$P(0) = \frac{1}{d} \sum_{j=1}^{d} \frac{x_j^2}{\eta^2} \leq \frac{\mathbb{E}(x_j^2)}{\eta^2} \tag{4.6}$$

where $\mathbb{E}(x_j^2)$ is the expectation value, or average, of $\{x_j\}_{j \in [d]}$. We finally obtain the state

$$\frac{1}{\sqrt{d}} \sum_{j=1}^{d} \alpha_j |j\rangle |x_j\rangle |0\rangle \tag{4.7}$$

The new amplitudes α_j must be proportional to x_j, i.e. $\alpha_j = c x_j$. To respect the normalization, we must have $\frac{1}{d} \sum_j c^2 x_j^2 = 1$, therefore $c = d / \sum_j x_j^2 = d / \|x\|_2^2$. This shows that the remaining state is the amplitude encoding version of the vector x:

$$\frac{1}{\|x\|_2} \sum_{j=1}^{d} x_j |j\rangle |x_j\rangle \tag{4.8}$$

Note that the output is close but not exactly the amplitude encoding of x as defined in Defintion 4.2, but can be used in a similar manner. The last two registers are entangled and thus the last one cannot be simply discarded. The only solution is to apply $U^\dagger$, the reversed unitary that was used to create the bit encoding state. This adds another time T_U in the computation but allows to obtain the exact amplitude encoding state $\frac{1}{\|x\|_2} \sum_{j=1}^{d} x_j |j\rangle$.

□

Unary Encoding

As the previous encoding, unary encoding takes advantage of the amplitudes of quantum states in superposition to encode the components of a vector. However, a unary encoded quantum vector can be loaded with smaller circuits, as shown in Section 4.1.2.

The key feature of unary encoding is to use only the amplitude of unary states: the states that have one and only $|1\rangle$, e.g. $|100\cdots0\rangle$, $|010\cdots0\rangle$, etc.

Definition 4.3: Unary Encoding

Given a vector $x = (x_0, \cdots, x_d) \in \mathbb{R}^d$, such that $\|x\|_2 = 1$. We can encode it in a superposition of unary states:

$$|x\rangle = x_0 |10\cdots0\rangle + x_1 |010\cdots0\rangle + \cdots + x_{d-1} |0\cdots01\rangle \tag{4.9}$$

We can also rewrite the previous state as:

$$|x\rangle = \sum_{i=0}^{d-1} x_i |e_i\rangle \tag{4.10}$$

where $|e_i\rangle$ is the i^{th} unary state with a $|1\rangle$ in the i^{th} position e.g. $|0\cdots010\cdots0\rangle$.

Note that if x is not normalized, it is still possible to load it and each amplitude will naturally be divided by the norm.

This encoding is suited for short term quantum computers (NISQ) that are prone to errors. A very convenient consequence is the ability to perform *error mitigation* while measuring the quantum states. Indeed, Indeed, as we expect to obtain only quantum superposition of unary states, we can post-process our measurements and discard the ones that present non unary states (i.e. states with more that one qubit $|1\rangle$, or the ground state). The most expected error is a bit-flip between $|1\rangle$ and $|0\rangle$. The case where two bit-flips happened, which would pass through our error mitigation, is even less probable.

Other Encodings

Other encodings exist in the literature but are not used in this thesis. It is worth mentioning the *gate encoding* used in variational quantum circuits (VQC), where the value of the input vector is directly put as the angle of rotation gate (note that it is close to unary loaders, see Section 4.1.2). *Data reuploading*, the fact of repeating the gates that encode a vector, seems to add efficiency to such VQCs [PSCLGFL20]. Trying to understand how vectors are mapped into the high dimensional Bloch sphere, and what can be done to process the data [SSM21].

Finally, there is also the Hamiltonian encoding which differs a lot since it consists of encoding the problem we desire to solve in a Hamiltonian form, and then try to perform the Hamiltonian evolution of an initial quantum state. This method is used in optimization, quantum annealing, QAOA type of VQCs [FGG14], but most

importantly in Hamiltonian simulation, as used originally in the HHL algorithm [HHL09] where the matrix to be inverted is encoded as a Hamiltonian.

4.1.2 Quantum Memory Models

A quantum memory model, or *data loader*, is the link between classical data and quantum states. It is a classical structure, such as a table, a tree, or a list, where classical information is written. For each type of encoding we desire for the quantum state (see Section 4.1.1), there exists one or several quantum circuits associated. In each case, it is important to differentiate the time to create the classical data structure, which should be done only once, and the time to load the quantum state.

Definition 4.4: Quantum memory model

For a given type of quantum encoding, a quantum memory model is a classical data structure that stores vectors $X_i \in \mathbb{R}^d$ for $i \in [N]$. Along with a quantum circuit, it can perform the mapping:

$$|i\rangle |0\rangle \mapsto |i\rangle |X_i\rangle \tag{4.11}$$

Note that starting with $\log(N)$ qubits in uniform superposition, all quantum states can be loaded as:

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle |0\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle |X_i\rangle \tag{4.12}$$

In the following, the number i will be referred as the *index* or sometimes the *address*.

Quantum Random Access Memory (QRAM)

The QRAM, or Quantum Random Access Memory, is often used as a quantum memory model for amplitude encoding (see Definition 4.2), its name is derived from the classical RAM, for their equivalent underlying tree structure.

For instance, an 8-dimensional vector $x = (x_0, \cdots, x_7)$ would be stored using a tree structure, also known as KP-tree [Pra14, KP16], as shown in Fig.4.2. Compared to the number of dimensions, it has a linear number of leaves, but most importantly a logarithmic depth.

The time to create such a data structure is linear in the dimension d, and in the number N of vectors to store, if applicable (see Theorem 4.1).

The interest of this framework is to compute angles that will be sequentially used to create $|x\rangle$, the amplitude encoding of x. Indeed, as shown in Fig.4.3 on a small example, each node of the tree has a relative weight that corresponds to the amount of rotation that should be applied. As a result, due to depth of the tree, the time to create the quantum state $|x\rangle$ is only $O(polylog(d))$ or, for N such vectors, $O(polylog(Nd))$ (see Theorem 4.1). Note that some additional details must be taken care of to handle the signs of the components [Pra14].

For any matrix, we call *quantum access* the ability to prepare, or *load*, the amplitude encoding of the rows of the matrix.

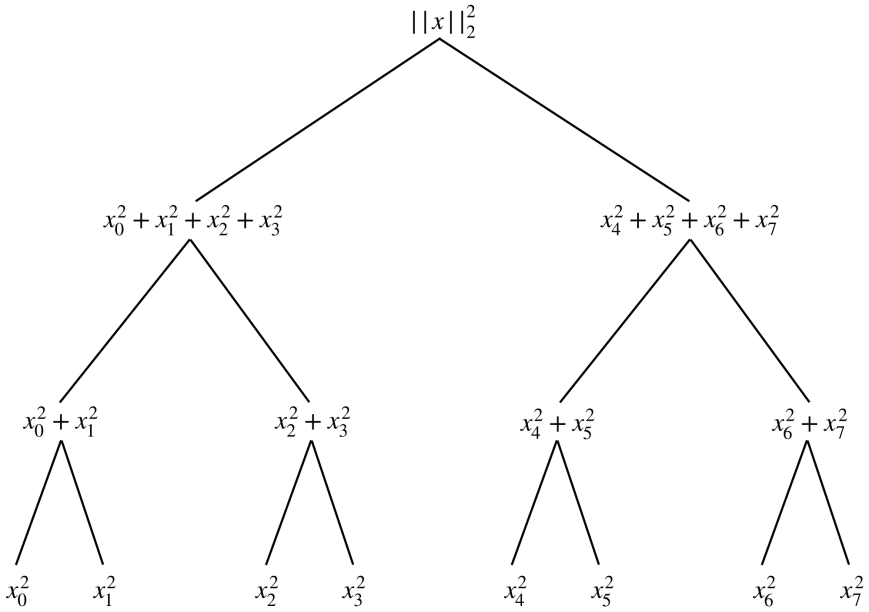


Figure 4.2: KP-tree structure for QRAM model on a 8-dimensional vector x.

Definition 4.5: Quantum Access to Data

We say that we have quantum access to a matrix $X \in \mathbb{R}^{N\times d}$ if there exists a procedure to perform the following mapping, for $i \in [N]$:

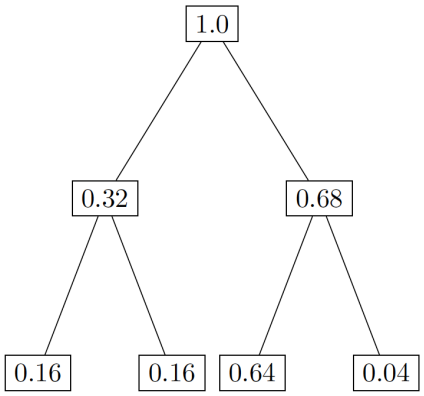
- $|i\rangle|0\rangle \mapsto |i\rangle|X_i\rangle$
- $|0\rangle \mapsto \frac{1}{\|X\|_F}\sum_i \|X_i\|_2 |i\rangle$

Theorem 4.1: QRAM

Let $X \in \mathbb{R}^{N\times d}$ be a matrix N vectors of dimension d. There is a data structure to store the rows of X such that,

1. The size of the structure is $O(Nd\log^2(Nd))$.
2. The time to store a row X_i is $O(d\log^2(Nd))$, and the time to store the whole matrix X is thus $O(Nd\log^2(Nd))$.
3. The time to insert, update or delete a single entry X_{ij} is $O(\log^2(Nd))$.
4. A quantum algorithm with access to the data structure can perform the following unitaries (in superposition if necessary) in time $O(\log^2(Nd))$.
   (a) $|i\rangle|0\rangle \to |i\rangle|X_i\rangle$ for $i \in [n]$.
   (b) $|0\rangle \to \frac{1}{\|X\|_F}\sum_{i\in[n]} \|X_i\|_2 |i\rangle$.

In practice, each component of the vectors X_i will have to be stored using classical bits. The number of bits k used for the precision of each value is not shown in Theorem 4.1 but should appear simply as a multiplicative factor.



Let $|\phi\rangle = 0.4\,|00\rangle + 0.4\,|01\rangle + 0.8\,|10\rangle + 0.2|11\rangle$.

- Rotation on qubit 1:
  $|0\rangle\,|0\rangle \to (\sqrt{0.32}\,|0\rangle + \sqrt{0.68}\,|1\rangle)\,|0\rangle$
- Rotation on qubit 2 conditioned on qubit 1:

$$(\sqrt{0.32}\,|0\rangle + \sqrt{0.68}\,|1\rangle)\,|0\rangle \to$$
$$\sqrt{0.32}\,|0\rangle\,\frac{1}{\sqrt{0.32}}(0.4\,|0\rangle + 0.4\,|1\rangle) +$$
$$\sqrt{0.68}\,|1\rangle\,\frac{1}{\sqrt{0.68}}(0.8\,|0\rangle + 0.2\,|1\rangle)$$

Figure 4.3: Example of sequential rotations to load the vector $\phi = (0.4, 0.4, 0.8, 0.2)$. Source: [Pra14].

Recently, using the QRAM data structure, the *Block Encoding* framework was introduced by [CGJ18, GSLW19], which allows us further improvements on quantum linear algebra subroutines (see Section 5.2).

Definition 4.6: Block Encoding of a Matrix

For a symmetric matrix $X \in \mathbb{R}^{n\times n}$, the q qubits unitary $U \in \mathbb{C}^{2^q\times 2^q}$ is a (ζ, q)-block encoding of M if $U = \begin{pmatrix} X/\zeta & \cdot \\ \cdot & \cdot \end{pmatrix}$. For a general matrix $M \in \mathbb{R}^{n\times m}$, we use a symmetrized version $\overline{P} = \begin{pmatrix} 0 & M \\ M^T & 0 \end{pmatrix}$ to construct a block encoding for it.

QRAM then allows us to store and load such block encoding. In detail, for a matrix $X \in \mathbb{R}^{n\times n}$ it can implement in $\widetilde{O}(\log(n))$ a $(\zeta(X), 2\log(n))$-block encoding with $\zeta(X) = \frac{1}{\|X\|_2}\min(\|X\|_F, s_1(X))$ where $s_1(X) = \max_i(\sum_j |X_{ij}|)$. As before, the storing takes a single pass over the matrix X, but a single update takes only $O(\log^2(n))$.

In the first proposals of a QRAM [GLM08b, GLM08a], the authors assumed access to a hardware platform that could naturally encode data into amplitudes. The circuit requires $O(d)$ qubits and $\log(d)$ depth, but necessitates specific hardware with light-matter interaction. On the other hand, one could also compose a controlled-NOT based multiplexer with only $O(\log(d))$ qubits, which remains impractical since it requires performing a sequence of $d\log(d)$ controlled gates [PPR19]. New ideas were suggested in [AGJO$^+$15] with $O(d)$ depth and $O(d)$ qubits, with strong tolerance to noise and quantum errors. This last proposal encodes the state in $\log(d)$ qubits but requires d additional classical bits to load the d values. Resource estimation and resilience against noise have been studied in depth [DMGM20, HLGJ21, ZDF$^+$18]. They reveal that the task of creating such a circuit and use it efficiently remains a strong challenge. However, in theory, this should be no more difficult than building the fault-tolerant quantum computer itself.

In the next section, we will introduce memory model that require only $\log(d)$ depth for d qubits, but using unary encoding this time.

Finally, note that different methods were proposed for loading amplitude encoding for near term devices, with the help of variational circuits [ZHLT20]. However, they remain imperfect and too costly for now.

Unary Data Loaders

The QRAM model for amplitude encoding might only be available for long term quantum computers. Therefore, shorter term data loaders were proposed in [JDM+20], suited for unary encoding (see Definition 4.3). Since these quantum states are a superposition of unary states, i.e. states with one and only qubit in state $|1\rangle$, circuits with $O(d)$ qubits and depth ranging from $O(\log(d))$ to $O(d)$ can be proposed.

They rely on the Reconfigurable Beam Splitter gate, or RBS gate for short. This two-qubit gate is parametrizable with one angle $\theta \in [0, 2\pi]$. Its matrix representation is given as:

$$RBS(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.13}$$

We note that this gate leaves the states $|00\rangle$ and $|11\rangle$ unaffected. For the two other states, it equivalent to a planar rotation with angle θ:

$$RBS(\theta) : \begin{cases} |01\rangle \mapsto \cos\theta\,|01\rangle - \sin\theta\,|10\rangle \\ |10\rangle \mapsto \sin\theta\,|01\rangle + \cos\theta\,|10\rangle \end{cases} \tag{4.14}$$

We can think of this gate as a rotation in the two-dimensional subspace spanned by the basis $\{|01\rangle, |10\rangle$, while it acts as the identity in the remaining subspace $\{|00\rangle, |11\rangle\}$. Or equivalently, starting with two qubits, one in the $|0\rangle$ state and the other one in the state $|1\rangle$, the qubits can be swapped or not in superposition. The qubit $|1\rangle$ stays on its wire with amplitude $\cos\theta$ or switches with the other qubit with amplitude $+\sin\theta$ if the new wire is below ($|10\rangle \mapsto |01\rangle$) or $-\sin\theta$ if the new wire is above ($|01\rangle \mapsto |10\rangle$). Note that in the two other cases ($|00\rangle$ and $|11\rangle$) the RBS gate acts as identity.

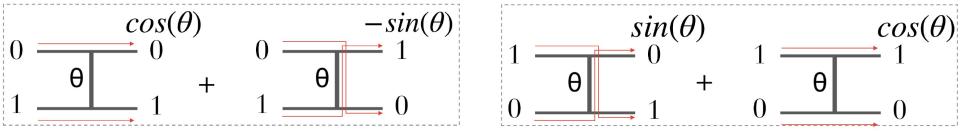
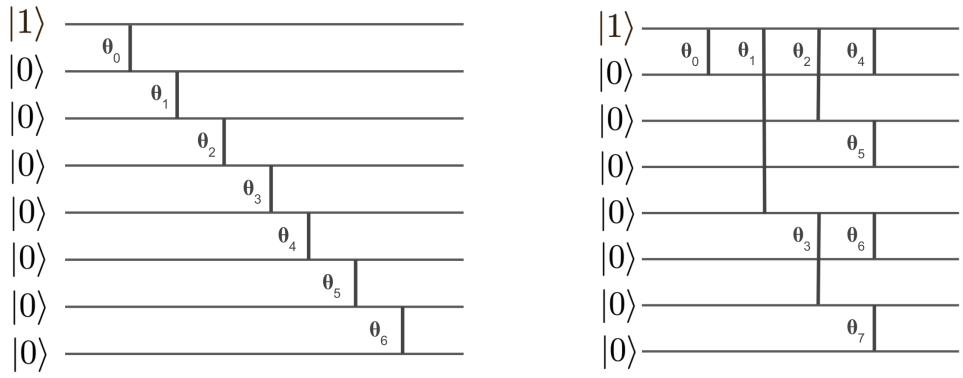


Figure 4.4: Representation of the quantum mapping from Eq.(4.13) on two qubits.

Given a vector $x = (x_1, \cdots, x_d) \in \mathbb{R}^d$, the associated unary data loaders is a simple circuit using d-1 RBS gates, along with the same number of precomputed angles θ_i. In Chapter 12, we will use a diagonal circuit of depth $O(d)$, as shown on Fig.4.5a. This circuit has longer depth but has the property of using only adjacent qubits, which is convenient when using the current quantum computers available. The other circuit, a *parallel* loader, shown on Fig.4.5b, has a depth of $O(\log(d))$. An important advantage compared to the classical inner that uses $O(d)$ steps.

A trade-off is possible between the number of qubits and the depth, the product of the two remaining constant. The optimal solution would be a circuit of $\sqrt{d}\log(d)$



(a) (b)

Figure 4.5: Two quantum circuits for unary data loaders with (a) diagonal and (b) parallel structure. Each vertical bar is a RBS gate applied on the two connected qubits, with angle θ_i.

depth and $2\sqrt{d}$ qubits. This could also be extended to loading matrices. These unary loaders can be combined to apply linear algebra tasks such as inner product [JDM+20] or matrix multiplication (see Chapter 12).

These unary data loaders can be easily implemented by classical emulator as they consist of planar rotations, and are therefore not exploring an exponentially large Hilbert space.

The creation of the angles θ_i for $i \in [d-1]$ is an easy task, requiring $O(d)$ classical precomputations. For instance, for the diagonal unary loader (Fig.4.5a), we recursively obtain d-1 loading angles with:

$$\begin{cases} \theta_0 = \arccos(x_0) \\ \theta_1 = \arccos\left(\frac{x_1}{\sin(\theta_0)}\right) \\ \theta_2 = \arccos\left(\frac{x_2}{\sin(\theta_0)\sin(\theta_1)}\right) \\ \cdots \end{cases} \tag{4.15}$$

Indeed, the diagonal unary loader starts in the all $|0\rangle$ state and flips the first qubit using an x gate, in order to obtain the unary state $|10\cdots0\rangle$ as shown in Fig.12.4. Then a cascade of RBS gates allows creating the state $|x\rangle$. The first gate will propagate the amplitude of the first qubit to the second one:

$$x_0 \,|100\cdots\rangle + \sin(\theta_0)\,|010\cdots\rangle \tag{4.16}$$

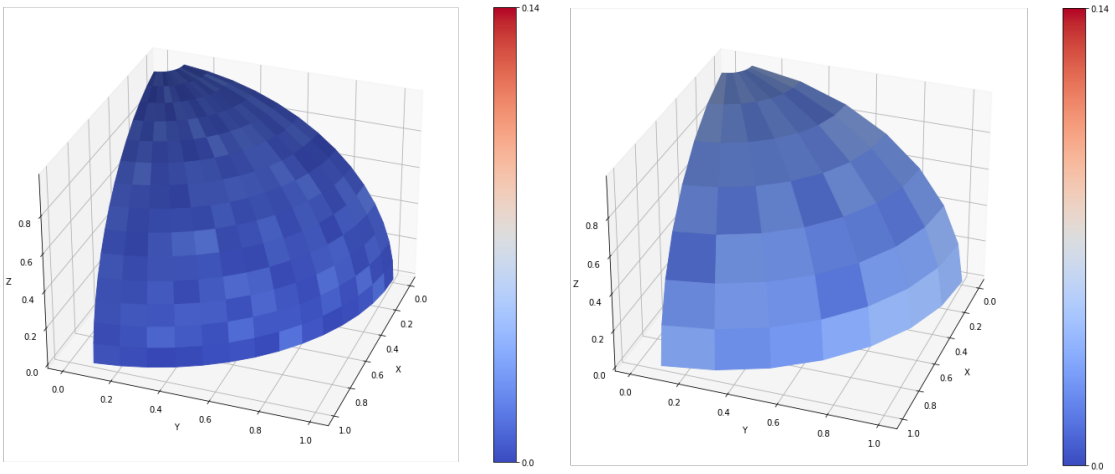
The second gate will create in turn the state :

$$x_0 \,|100\cdots\rangle + x_1\,|010\cdots\rangle + \sin(\theta_0)\sin(\theta_1)\,|001\cdots\rangle \tag{4.17}$$

and so on, until obtaining $|x\rangle$ as in Eq.(12.2).

Finally, to verify the accuracy of these encoding, we performed a real hardware implementation of the unary diagonal loader. We compared the results with quantum circuit simulations. Using a 5 qubits superconducting quantum computer

IBMSantiago, we were able to load unary encoded vectors on the 3-dimensional unit sphere. For this, we used vectors of the form $(x_0, x_1, x_2) \in [0, 1]^3$. For several points, we created the quantum state, measured 8192 samples of it to recover its position, and calculated the euclidean error compared to the actual vector. Results are shown on Fig.4.6. Naturally, we see more errors (light blue coloring) for the real experiment, but the results seem consistent with the simulations.



(a) (b)

Figure 4.6: ℓ_2 norm errors when comparing vectors of the 3D unit sphere and their quantum version by using diagonal unary loaders. (a) Emulated results and (b) Actual hardware experiment on the *IBMSantiago* quantum computer. May 2021.

4.2 From Quantum to Classical data

For quantum linear algebra and machine learning applications, recovering classical data from quantum states is the other side of the coin. This process, called tomography, is usually very costly and is the second bottleneck of quantum algorithms [Aar15], after the data loading (see Section 4.1.1).

In this section we will present two different tomography procedures, both for amplitude encoded quantum vectors, the second one being a contribution of this thesis.

4.2.1 ℓ_2 and ℓ_∞ Tomography

We consider a final state of a quantum circuit, expected to be $|x\rangle = \frac{1}{c}\sum_{i=0}^{d-1} x_i |i\rangle$, the quantum version of an unknown vector $x = (x_0, \cdots, x_{d-1}) \in \mathbb{R}^d$. Measuring this state will randomly result in one of the binary strings $|i\rangle$. Performing a sufficiently large number of measurements would allow us to guess the underlying probability distribution of the binary strings, and therefore the amplitudes $(|x_0|^2, \cdots, |x_{d-1}|^2)$. As the number of measurements is not infinite, the probability distribution is only approximated and therefore an error is committed on the recovered vector, in particular for small value components. Depending on the guarantee we put on this error, the number of queries is modified.

We first present an ℓ_2-norm guarantee tomography, introduced in [KP20b]. Informally, for some parameter $\delta > 0$, we require that the resulting vector $\tilde{x}$ is δ-close to the actual vector x.

Theorem 4.2: ℓ_2 Vector State Tomography

Given access to unitary U such that $U|0\rangle = |x\rangle$ and its controlled version in time $T(U)$, there is a tomography algorithm with time complexity $O(T(U)\frac{d \log d}{\delta^2})$ that produces unit vector $\widetilde{x} \in \mathbb{R}^d$ such that $\|\widetilde{x} - x\|_2 \leq \delta$ with probability at least $(1 - 1/poly(d))$.

Next, we introduce a new procedure, the ℓ_∞-norm guarantee tomography, where now each recovered component $\overline{x}_i$ must be δ-close to the actual vector's component x_i. Noticeably, this tomography requires exponentially fewer resources than the previous one.

Theorem 4.3: ℓ_∞ Vector State Tomography

Given access to unitary U such that $U|0\rangle = |x\rangle$ and its controlled version in time $T(U)$, there is a tomography algorithm with time complexity $O(T(U)\frac{\log d}{\delta^2})$ that produces unit vector $\widetilde{x} \in \mathbb{R}^d$ such that $\|\widetilde{x} - x\|_\infty \leq \delta$ with probability at least $(1 - 1/poly(d))$.

In some contexts, the ℓ_∞ tomography turns out to be more meaningful. In fact, when the quantum state $|x\rangle$ does not represent a vector per se, as a position on a mesh or some precise embedding in a latent space, but rather a collection of values like the pixels of an image or a time series, it is in practice less critical to have low precision on some components. In some cases as well, we only care about the high-value elements of x, for instance in neural networks, where non-linearities are applied to push high values even higher, and downsize small values (see Section 2.3.1).

For instance, in the case of visual neural networks (see Chapter 11), we will use this tomography to recover the highest valued pixels in an image.

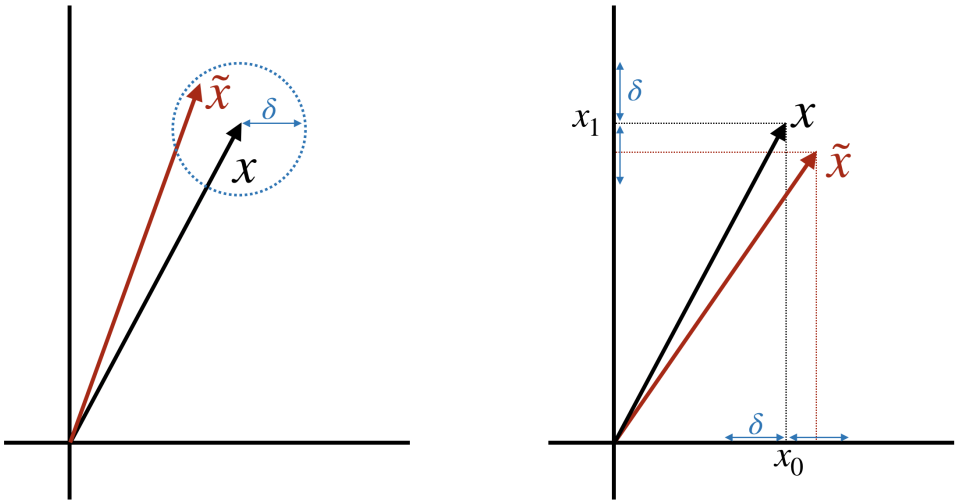


Figure 4.7: Representation of ℓ_2 (left) and ℓ_∞ (right) error guarantees.

4.2.2 ℓ_∞ Tomography Details

To prove the Theorem 4.3 introduced in this thesis, we follow the method from [KP20b]. In the following we consider a quantum state $|x\rangle$ such that $x \in \mathbb{R}^d$ and $\|x\|_2 = 1$.

Algorithm 1 ℓ_∞ norm tomography

Require: Error $\delta > 0$, access to unitary $U : |0\rangle \mapsto |x\rangle = \sum_{i\in[d]} x_i |i\rangle$, the controlled version of U, QRAM access.

Ensure: Classical vector $\widetilde{x} \in \mathbb{R}^d$, such that $\|\widetilde{x}\| = 1$ and $\|\widetilde{x} - x\|_\infty < \delta$.

1: Measure $N = \frac{36 \ln d}{\delta^2}$ copies of $|x\rangle$ in the standard basis and count n_i, the number of times the outcome i is observed. Store $\sqrt{p_i} = \sqrt{n_i/N}$ in QRAM data structure.

2: Create $N = \frac{36 \ln d}{\delta^2}$ copies of the state $\frac{1}{\sqrt{2}} |0\rangle \sum_{i\in[d]} x_i |i\rangle + \frac{1}{\sqrt{2}} |1\rangle \sum_{i\in[d]} \sqrt{p_i} |i\rangle$.

3: Apply an Hadamard gate on the first qubit to obtain

$$|\phi\rangle = \frac{1}{2} \sum_{i\in[d]} ((x_i + \sqrt{p_i}) |0, i\rangle + (x_i - \sqrt{p_i}) |1, i\rangle) \tag{4.18}$$

4: Measure both registers of each copy in the standard basis, and count $n(0, i)$ the number of time the outcome $(0, i)$ is observed.

5: Set $\sigma(i) = +1$ if $n(0, i) > 0.4Np_i$ and $\sigma(i) = -1$ otherwise.

6: Output the unit vector $\widetilde{x}$ such that $\forall i \in [N], \widetilde{x}_i = \sigma_i \sqrt{p_i}$

The following version of the Chernoff Bound will be used for analysis of algorithm 1.

Theorem 4.4: Chernoff Bound

Let X_j, for $j \in [N]$, be independent random variables such that $X_j \in [0, 1]$ and let $X = \sum_{j\in[N]} X_j$. We have the three following inqualities:

1. For $0 < \beta < 1, \mathbb{P}[X < (1 - \beta)\mathbb{E}[X]] \leq e^{-\beta^2 \mathbb{E}[X]/2}$
2. For $\beta > 0, \mathbb{P}[X > (1 + \beta)\mathbb{E}[X]] \leq e^{-\frac{\beta^2}{2+\beta}\mathbb{E}[X]}$
3. For $0 < \beta < 1, \mathbb{P}[|X - \mathbb{E}[X]| \geq \beta\mathbb{E}[X]] \leq e^{-\beta^2 \mathbb{E}[X]/3}$, by composing 1. and 2.

Theorem 4.5

Algorithm 1 produces an estimate $\widetilde{x} \in \mathbb{R}^d$ such that $\|\widetilde{x} - x\|_\infty < (1 + \sqrt{2})\delta$ with probability at least $1 - \frac{1}{d^{0.83}}$.

Proof. Proving $\|x - \widetilde{x}\|_\infty \leq O(\delta)$ is equivalent to showing that for all $i \in [d]$, we have $|x_i - \widetilde{x}_i| = |x_i - \sigma(i)\sqrt{p_i}| \leq O(\delta)$. Let S be the set of indices defined by $S = \{i \in [d]; |x_i| > \delta\}$. We will separate the proof for the two cases where $i \in S$ and $i \notin S$.

Case 1 : $i \in S$.

We will show that if $i \in S$, we correctly have $\sigma(i) = sgn(x_i)$ with high probability. Therefore we will need to bound $|x_i - \sigma(i)\sqrt{p_i}| = ||x_i| - \sqrt{p_i}|$.

We suppose that $x_i > 0$. The value of $\sigma(i)$ correctly determines $sgn(x_i)$ if the number of times we have measured $(0, i)$ at Step 4 is more than half of the measurements, i.e. $n(0,i) > \frac{1}{2}\mathbb{E}[n(0,i)]$. If $x_i < 0$, the same arguments holds for $n(1,i)$. We consider the random variable that represents the outcome of a measurement on state $|\phi\rangle$. The Chernoff Bound (part 1) with $\beta = 1/2$ gives

$$\mathbb{P}[n(0,i) \leq \frac{1}{2}\mathbb{E}[n(0,i)]] \leq e^{-\mathbb{E}[n(0,i)]/8} \tag{4.19}$$

From the definition of $|\phi\rangle$, see Eq.(4.18), we have $\mathbb{E}[n(0,i)] = \frac{N}{4}(x_i + \sqrt{p_i})^2$. We will lower bound this value with the following argument.

For the k^{th} measurement of $|x\rangle$, with $k \in [N]$, let X_k be a random variable such that $X_k = 1$ if the outcome is i, and 0 otherwise. We define $X = \sum_{k\in[N]} X_k$. Note that $X = n_i = Np_i$ and $\mathbb{E}[X] = Nx_i^2$. We can apply the Chernoff Bound, part 3 on X for $\beta = 1/2$ to obtain,

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq \mathbb{E}[X]/2] \leq e^{-\mathbb{E}[X]/12} \tag{4.20}$$

$$\mathbb{P}[|x_i^2 - p_i| \geq x_i^2/2] \leq e^{-Nx_i^2/12} \tag{4.21}$$

We have $N = \frac{36\ln d}{\delta^2}$ and by assumption $x_i^2 > \delta^2$ (since $i \in S$). Therefore,

$$\mathbb{P}[|x_i^2 - p_i| \geq x_i^2/2] \leq e^{-36\ln d/12} = 1/d^3 \tag{4.22}$$

This proves that the event $|x_i^2 - p_i| \leq x_i^2/2$ occurs with probability at least $1 - \frac{1}{d^3}$ if $i \in S$. This previous inequality is equivalent to $\sqrt{2p_i/3} \leq |x_i| \leq \sqrt{2p_i}$. Thus, with high probability we have $\mathbb{E}[n(0,i)] = \frac{N}{4}(x_i + \sqrt{p_i})^2 \geq 0.82Np_i$, since $\sqrt{2p_i/3} \leq |x_i|$. Moreover, since $|p_i| \leq x_i^2/2$, $\mathbb{E}[n(0,i)] \geq 0.82Nx_i^2/2 \geq 14.7\ln d$. Therefore, equation (4.19) becomes

$$\mathbb{P}[n(0,i) \leq 0.41Np_i] \leq e^{-1.83\ln d} = 1/d^{1.83} \tag{4.23}$$

We conclude that for $i \in S$, if $n(0,i) > 0.41Np_i$, the sign of x_i is determined correctly by $\sigma(i)$ with high probability $1 - \frac{1}{d^{1.83}}$, as indicated in Step 5.

We finally show $|x_i - \sigma(i)\sqrt{p_i}| = ||x_i| - \sqrt{p_i}|$ is bounded. Again by the Chernoff Bound (3.) we have, for $0 < \beta < 1$:

$$\mathbb{P}[|x_i^2 - p_i| \geq \beta x_i^2] \leq e^{\beta^2 Nx_i^2/3} \tag{4.24}$$

By the identity $|x_i^2 - p_i| = (|x_i| - \sqrt{p_i})(|x_i| + \sqrt{p_i})$ we have

$$\mathbb{P}\left[\left||x_i| - \sqrt{p_i}\right| \geq \beta\frac{x_i^2}{|x_i| + \sqrt{p_i}}\right] \leq e^{\beta^2 Nx_i^2/3} \tag{4.25}$$

Since $\sqrt{p_i} > 0$, we have $\beta\frac{x_i^2}{|x_i|+\sqrt{p_i}} \leq \beta\frac{x_i^2}{|x_i|} = \beta|x_i|$, therefore $\mathbb{P}\left[\left||x_i| - \sqrt{p_i}\right| \geq \beta|x_i|\right] \leq e^{\beta^2 Nx_i^2/3}$. Finally, by chosing $\beta = \delta/|x_i| < 1$ we have

$$\mathbb{P}\left[\left||x_i| - \sqrt{p_i}\right| \geq \delta\right] \leq e^{36\ln d/3} = 1/d^{12} \tag{4.26}$$

We conclude that, if $i \in S$, we have $|x_i - \tilde{x}_i| \leq \delta$ with high probability.

Since $|S| \leq d$, the probability for this result to be true for all $i \in S$ is $1 - \frac{1}{d^{0.83}}$. This follows from the Union Bound on the correctness of $\sigma(i)$.

Case 2 : $i \notin S$.

If $i \notin S$, we need to separate again in two cases. When the estimated sign is wrong, i.e. $\sigma(i) = -sgn(x_i)$, we have to bound $|x_i - \sigma(i)\sqrt{p_i}| = ||x_i| + \sqrt{p_i}|$. On the contrary, if it is correct, i.e. $\sigma(i) = sgn(x_i)$, we have to bound $|x_i - \sigma(i)\sqrt{p_i}| = ||x_i| - \sqrt{p_i}| \leq ||x_i| + \sqrt{p_i}|$. Therefore only one bound is necessary.

We use Chernoff Bound (2.) on the random variable X with $\beta > 0$ to obtain

$$\mathbb{P}[p_i > (1+\beta)x_i^2] \leq e^{\frac{\beta^2}{2+\beta}Nx_i^2} \tag{4.27}$$

We chose $\beta = \delta^2/x_i^2$ and obtain $\mathbb{P}[p_i > x_i^2 + \delta^2] \leq e^{\frac{\delta^4}{3\delta^2}N} = 1/d^{12}$. Therefore, if $i \notin S$, with very high probability $1 - \frac{1}{d^{12}}$ we have $p_i \leq x_i^2 + \delta^2 \leq 2\delta^2$. We can conclude and bound the error:

$$|x_i - \tilde{X}_i| \leq ||x_i| + \sqrt{p_i}| \leq \delta + \sqrt{2}\delta = (1+\sqrt{2})\delta \tag{4.28}$$

Since $|\overline{S}| \leq d$, the probability for this result to be true for all $i \notin S$ is $1 - \frac{1}{d^{11}}$. This follows from applying the Union Bound on the event $p_i > x_i^2 + \delta^2$. □

Chapter 5

Quantum Linear Algebra

"The purpose of computation is insight, not numbers."

Richard Hamming
Numerical Methods for Scientists and Engineers (1962)

Throughout this thesis, we will use previously developed quantum linear algebra subroutines. These fundamental algorithms harness quantum encoding seen in Chapter 4 to perform their tasks with substantial speedup.

We will present succinctly two of them: the singular value estimation (Section 5.1), and the matrix multiplication (Section 5.2).

5.1 Singular Values Estimation and Projection

Singular Value Decomposition (SVD)

We recall some properties from Section 1.4. For any matrix $A \in \mathbb{R}^{m\times n}$, the singular value decomposition (SVD) of A is

$$A = U\Sigma V^T \tag{5.1}$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are unitary matrices, and $\Sigma \in \mathbb{R}^{m\times n}$ is a rectangular diagonal matrix with non-negative elements σ_i. With r being the rank of A, the SVD can be also expressed as:

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T \tag{5.2}$$

where the left and the right singular vectors u_i and v_i are the columns of U and V.

The singular values σ_i of a matrix A of great importance to understand its properties, especially when the matrix is a transformation from one space to another, or when it is representing a graph (see Section 2.2.2).

The singular values of a $m \times n$ matrix A are the square roots of the eigenvalues of the $n \times n$ matrix $A^T A$. Thus, if A is a $n \times n$, real and positive semidefinite matrix, the singular values and the eigenvalues are the same, which is not generally the case due to negative signs.

Quantum Singular Value Estimation (SVE)

A quantum algorithm for singular value estimation (SVE) was developed in [KP16] for solving the recommendation system problem, and later improved in [KP20a], inspired by the method of [Chi10] for solving eigenvalue estimation. They extended to non unitary matrices the method used in [HHL09] from extracting eigenvalues.

Given a matrix A stored in the appropriate quantum memory model (see Section 4.1.2), the algorithm can map the quantum state of any right singular vector $|v_i\rangle$ to its singular value σ_i, with some precision $\epsilon > 0$. The interesting feature comes when this is applied in superposition over all right singular vectors. And since they form a complete basis, any quantum vector $|x\rangle$ can be written as $|x\rangle = \sum_i \alpha_i |v_i\rangle$, for some coefficients α_i. We begin by introducing the parameter $\mu(A)$ in Definition 5.1.

Definition 5.1: Parameter $\mu(\cdot)$

For a matrix A, the parameter $\mu(A)$ is defined by

$$\mu(A) = \min_{p\in[0,1]} \left(\|A\|_F , \sqrt{s_{2p}(A) s_{2(1-p)}(A^T)} \right) \tag{5.3}$$

where $s_p(A) = \max_i(\|A_i\|_p^p)$.

Theorem 5.1: Quantum Singular Value Estimation

Given quantum access in time T to a matrix $A \in \mathbb{R}^{m\times n}$ with singular value decomposition $A = \sum_i \sigma_i u_i v_i^T$, there is a quantum algorithm that performs the mapping

$$\sum_i \alpha_i |v_i\rangle |0\rangle \mapsto \sum_i \alpha_i |v_i\rangle |\overline{\sigma_i}\rangle \tag{5.4}$$

such that for any precision $\epsilon > 0$, we have for all singular values $|\overline{\sigma_i} - \sigma_i| \leq \epsilon$, in time $\widetilde{O}(T\mu(A)/\epsilon)$, with probability at least $1 - 1/poly(n)$.

The parameter $\mu(A)$ will appear frequently in the running time of algorithms that use quantum linear algebra subroutines. For dense matrices, $\mu(A)$ can be taken to be the ratio Frobenius Norm / Spectral Norm of A. In some sense, it replaces the explicit dependence on the matrix dimension. Note that with $p = 1/2$ we have $\mu(A) \leq s_1(A) = \max_i \|A_i\|_1$. For sparse matrices, $\mu(A)$ can then be seen as the sparsity.

We informally give the details of the initial quantum SVE algorithm [KP16]. One starts by finding two matrices $P \in \mathbb{R}^{mn\times m}$ and $Q \in \mathbb{R}^{mn\times n}$ that form the decomposition $A/\|A\|_F = P^T Q$, such that we have fast quantum access to P and Q. Then we apply phase estimation (Theorem 3.1) on the unitary $W = (2PP^T - I)(2QQ^T - I)$. The input should be $|Qx\rangle$ that can be created from $|x\rangle = \sum_i \alpha_i |v_i\rangle$. Since the eigenvectors of W are Qv_i, with eigenvalues $e^{i\theta_i}$ such that $\cos(\theta_i/2) = \sigma_i/\|A\|_F$, phase estimation allows us to obtain the state $\sum_i \alpha_i |Qv_i\rangle |\overline{\theta_i}\rangle$ and then $\sum_i \alpha_i |Qv_i\rangle |\overline{\sigma_i}\rangle$ with simple processing. Inverting the whole computation will yield to $\sum_i \alpha_i |v_i\rangle |\overline{\sigma_i}\rangle$. We also see that the error in the estimation of σ_i comes directly from the error of phase estimation. The running time at this point is $O(polylog(mn)/\epsilon)$.

In the improved version [KP20a], an additional qubit is used, and instead of choosing P and Q such that $A/\|A\|_F = PQ$, we know require $A/\mu = P \circ Q$ for some parameter μ. It is shown that the best parameter to choose is $\mu(A)$ from Definition 5.1.

Projections using SVE

It is a common task to project vectors in lower-dimensional space. If these space are spanned by a specific basis that results from singular vector or eigenvector decomposition, the projection is meaningful. This technique is widely used in machine learning to reduce the inputs, and quantum algorithms have been proposed to solve this in several ways [LMR14, KP16, KL20]. Often, one wants to select either the highest or lowest singular values and project all vectors in the subspace spanned by the corresponding singular vectors. We will also require such projections in Chapter 9.

Using the quantum algorithm for SVE (Theorem 5.1), it becomes easier since, starting from any vector $|x\rangle$ decomposed in the singular basis, we obtain a superposition of all singular vectors along with their singular values.

$$|x\rangle = \sum_i \alpha_i |v_i\rangle \mapsto \sum_i \alpha_i |v_i\rangle |\sigma_i\rangle \tag{5.5}$$

It then suffices to separate the retained singular values from the others. For instance, if the singular values to select are the ones smaller than a parameter $\delta > 0$, using a boolean comparison circuit (see Claim 3.1) and a marked ancilla qubit, we obtain:

$$\sum_{i|\sigma_i \leq \delta} \alpha_i |v_i\rangle |\sigma_i\rangle + \sum_{i|\sigma_i > \delta} \alpha_i |v_i\rangle |\sigma_i\rangle \mapsto \sum_{i|\sigma_i \leq \delta} \alpha_i |v_i\rangle |\sigma_i\rangle |0\rangle + \sum_{i|\sigma_i > \delta} \alpha_i |v_i\rangle |\sigma_i\rangle |1\rangle \tag{5.6}$$

Then, as in Claim 4.1, we start by applying a conditional rotation (Theorem 3.5) to have:

$$\sum_{i|\sigma_i \leq \delta} \alpha_i |v_i\rangle |\sigma_i\rangle |0\rangle \left(\frac{\sigma_i}{\delta} |0\rangle + \sqrt{1 - \frac{\sigma_i^2}{\delta^2}} |1\rangle \right) + \sum_{i|\sigma_i > \delta} \alpha_i |v_i\rangle |\sigma_i\rangle |1\rangle |0\rangle \tag{5.7}$$

and then we only have to measure the state $|00\rangle$ on the last two qubits. This could also be done using amplitude amplification on the state $|00\rangle$ (Theorem 3.2). In both cases we obtain the desired projection:

$$\sum_{i|\sigma_i \leq \delta} \alpha'_i |v_i\rangle \tag{5.8}$$

The new amplitudes α'_i will correspond to the component of the projection of vector x in the new subspace.

The running time of this operation is driven by the amplitude amplification that involves $O(1/\sqrt{P(00)})$ queries to the SVE circuit itself, where $P(00)$ is the probability to measure '00' in the last two qubits of state (5.7), given by:

$$P(00) = \sum_{i|\sigma_i \leq \delta} \alpha_i^2 \frac{\sigma_i^2}{\delta^2} \tag{5.9}$$

5.2 Matrix Multiplication and Inversion

As mentioned in Section 1.2, the field of quantum machine learning was ignited by the pioneering work of [HHL09] giving exponential speedup for matrix multiplication and inversion. this construction was based on phase estimation (Theorem 3.1) and Hamiltonian simulation. Later, improvements were proposed in [CKS17, KP20a], until [CGJ18, GSLW19] improved it again error wise, by introducing the *Block Encoding* framework (see Definition 4.6). This ensures a complexity sublinear in the dimension. Recall that for a matrix M, $\kappa(M)$ is its condition number (the ratio between the biggest and the smallest singular values), and $\mu(M)$ is defined in Definition 5.1.

Theorem 5.2: Quantum Matrix Multiplication and Inversion

Let $M \in \mathbb{R}^{d\times d}$ and $x \in \mathbb{R}^d$. Let $\delta_1, \delta_2 > 0$. If M is stored in appropriate QRAM data structures and the time to prepare $|x\rangle$ is T_x, then there exist quantum algorithms that with probability at least $1 - 1/poly(d)$ return

1. A state $|z\rangle$ such that $\||z\rangle - |Mx\rangle\|_2 \leq \delta_1$ in time $\widetilde{O}((\kappa(M)\mu(M) + T_x\kappa(M))\log(1/\delta_1))$.
   Note that this also implies $\||z\rangle - |Mx\rangle\|_\infty \leq \delta_1$
2. A state $|z\rangle$ such that $\||z\rangle - |M^{-1}x\rangle\| \leq \epsilon$ in time $\widetilde{O}((\kappa(M)\mu(M) + T_x\kappa(M))\log(1/\epsilon))$.
3. Norm estimate $z \in (1 \pm \delta_2)\|Mx\|_2$, with relative error δ_2, in time $\widetilde{O}(T_x \frac{\kappa(M)\mu(M)}{\delta_2}\log(1/\delta_1))$.

Notably, these routines for quantum linear algebra can be applied to products of matrices. If the matrix M is the product of k matrices, i.e. $M = M_1...M_k$, the resulting factor in the runtime is $\widetilde{O}(\kappa(M)\sum_k \mu(M_k)\log(1/\epsilon))$.

5.3 Quantum Inspired Algorithms

A recent breakthrough by Tang et al. [GLT18, Tan18, Tan19], proposed several classical machine learning algorithms obtained by *dequantizing* the quantum recommendation systems algorithm [KP16] and low rank linear system solvers. Like the quantum algorithms, the running time of these classical algorithms is $O(poly(k)polylog(mn))$, which is polylogarithmic in the dimension of the dataset and polynomial in the rank. However, the polynomial dependence on the rank of the matrices is significantly worse than the quantum algorithms and in fact renders these classical algorithms highly impractical. For example, the classical algorithm for stochastic regression inspired by the HHL algorithm [HHL09] has a running time of $\tilde{O}(\kappa^6 k^{16}\|A\|_F^6/\epsilon^6)$, which is impractical even for a rank-10 matrix. This running time has then been improved by [SM21] under some assumptions, to obtain a running time of $\widetilde{O}(\kappa_F^6\kappa^2/\epsilon^2)$ where $\kappa_F = \|A\|_F\|A^{-1}\|$ and $\kappa = \|A\|_2\|A^{-1}\|$, if A is the matrix to invert or to multiply with.

The extremely high dependence on the rank and the other parameters implies not only that the quantum algorithms are substantially faster, since their dependence

on the rank is sublinear, but more importantly that in practice there exist much faster classical algorithms for these problems. While these new quantum inspired classical algorithms are based on the FKV methods [FKV04], in classical linear algebra, algorithms based on the CUR decomposition (a low rank approximation of the SVD) that have a running time linear in the dimension and quadratic in the rank are preferred to the FKV methods [FKV04, DFK$^+$04, AM01]. Experimental comparisons with the usual classical algorithms have been done in [ADBL19] but don't seem in favor of the new quantum inspired algorithms.

It remains an open question to find classical algorithms for these machine learning problems that are polylogarithmic in the dimension and are competitive with respect to the quantum or the classical algorithms for the same problems. This would involve using significantly different techniques than the ones presently used for these algorithms.

That being said, these results remain theoretically important as they reduce the exponential separation between quantum and classical linear algebra algorithms for low rank problems. In fact, the algorithms presented in this thesis could have their own quantum inspired versions (as we show for the quantum convolutional neural network algorithm in Chapter 11), but would not be usable in practice. This still forces us to find where quantum computations must draw to find an unrivaled advantage. Recent results indicate that sparsity-based algorithm, where the matrices and vectors are high dimensional but sparse, is a better candidate against quantum inspired classical algorithms.

Chapter 6

Inner Product and Distance Estimation

"Quantum mechanics describes nature as absurd from the point of view of common sense. And yet it fully agrees with experiment. So I hope you can accept nature as She is - absurd."

Richard P. Feynman
QED: The Strange Theory of Light and Matter (1985)

6.1 Related Work

6.1.1 SWAP Test

In this section, we detail the seminal idea of [LMR13] of using the SWAP test to compute the distance between two vectors. We assume quantum access to the vectors and their norms, using the amplitude encoding framework (Definition 4.2). For two vectors $v_i \in \mathbb{R}^d$ and $v_j \in \mathbb{R}^d$, respectively indexed by i and j, we can query them in quantum registers in time T using the mapping:

$$|i\rangle |0\rangle \mapsto |i\rangle |v_i\rangle , \quad |j\rangle |0\rangle \mapsto |j\rangle |v_j\rangle \tag{6.1}$$

With a QRAM data structure (Theorem 4.1) the query time is $O(\log d)$ where d if the dimension of the vectors. We can also query their norms in a similar manner,

$$|i\rangle |0\rangle \mapsto |i\rangle |\|v_i\|\rangle , \quad |j\rangle |0\rangle \mapsto |j\rangle |\|v_j\|\rangle \tag{6.2}$$

We can compute the distance $d(v_i, v_j)$ in the amplitude of the ancillary register by performing a Swap Test between two states that were introduced in [LMR13]. Define,

$$|\psi_{ij}\rangle = \frac{1}{\sqrt{2}}(|v_i\rangle |0\rangle - |v_j\rangle |1\rangle) \tag{6.3}$$

$$|\phi_{ij}\rangle = \frac{1}{\sqrt{Z_{ij}}}(\|v_i\| \, |0\rangle + \|v_j\| \, |1\rangle) \tag{6.4}$$

Where $Z_{ij} = \|v_i\|^2 + \|v_j\|^2$. Note that, in Section 6.2, compared to [LMR13], we will exchange the minus sign between $|\psi\rangle$ and $|\phi\rangle$, and the two registers in $|\psi\rangle$, in order to avoid an extra quantum arithmetic operation to the conditional rotation step for $|\phi_{ij}\rangle$. Note that these states have been chosen for retrieving the distance between v_i and v_j, but it suffices to replace $|\phi_{ij}\rangle$ by the $|+\rangle$ state to obtain the inner product $|\langle v_i|v_j\rangle|^2$ instead.

We now describe the preparation procedures for $|\psi_{ij}\rangle$ and $|\phi_{ij}\rangle$. In order to create $|\psi_{ij}\rangle$, we first create $|i\rangle \, |j\rangle \, |-\rangle \, |0\rangle$ and then perform controlled queries as indicated below,

$$\begin{cases} |i\rangle \, |j\rangle \, |0\rangle \, |0\rangle \mapsto |i\rangle \, |j\rangle \, |0\rangle \, |v_i\rangle \\ |i\rangle \, |j\rangle \, |1\rangle \, |0\rangle \mapsto |i\rangle \, |j\rangle \, |1\rangle \, |v_j\rangle \end{cases} \tag{6.5}$$

$$|i\rangle \, |j\rangle \, \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \, |0\rangle \to |i\rangle \, |j\rangle \, \frac{1}{\sqrt{2}}(|0\rangle \, |v_i\rangle - |1\rangle \, |v_j\rangle) = |i\rangle \, |j\rangle \, |\psi_{ij}\rangle \tag{6.6}$$

A different procedure is used for creating $|\phi_{ij}\rangle$. We start with $|i\rangle \, |j\rangle$ as well and query the two norms in ancilla registers to obtain $|i\rangle \, |j\rangle \, |\|v_i\|\rangle \, |\|v_j\|\rangle$. We then add an extra qubit $|0\rangle$ and apply a controlled rotation, to directly create $|i\rangle \, |j\rangle \, |\|v_i\|\rangle \, |\|v_j\|\rangle \, |\phi_{ij}\rangle$. We can then undo the first step to remove the third and fourth registers.

Having prepared the states $|\psi_{ij}\rangle$ and $|\phi_{ij}\rangle$, we can apply the Swap Test circuit given below. The swap test circuit introduces an ancilla qubit on which we first apply a Hadamard gate, we then use the ancilla qubit to perform a controlled swap on two quantum registers, this is followed by a Hadamard gate and a measurement on the ancilla.

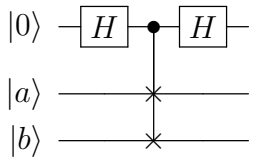


Figure 6.1: Swap Test circuit on two quantum states $|a\rangle$ and $|b\rangle$

The action of the swap test over two registers is:

$$|0\rangle \, |a\rangle \, |b\rangle \mapsto \left(\frac{1}{2} \, |0\rangle \, (|a\rangle \, |b\rangle + |b\rangle \, |a\rangle) + \frac{1}{2} \, |1\rangle \, (|a\rangle \, |b\rangle - |b\rangle \, |a\rangle)\right) \tag{6.7}$$

It follows that after the swap test, the probability of measuring $|0\rangle$ in the ancilla register is equal to $\frac{1+|\langle a|b\rangle|^2}{2}$.

In our case, for the swap test between $|\phi_{ij}\rangle$ and the second register of $|\psi_{ij}\rangle$ which we denote as $\langle\phi_{ij}|\psi_{ij,2}\rangle$. the probability p_{ij} of measuring 0 in the final ancilla register is $\frac{1+|\langle\phi_{ij}|\psi_{ij,2}\rangle|^2}{2}$. In order to compute p_{ij} we first compute $|\langle\phi_{ij}|\psi_{ij,2}\rangle|^2$,

$$
\begin{aligned}
|\langle\phi_{ij}|\psi_{ij,2}\rangle|^2 &= \frac{1}{2Z_{ij}} \left|(\|v_i\| \langle 0| + \|v_j\| \langle 1|)(|v_i\rangle |0\rangle - |v_j\rangle |1\rangle)\right|^2 \\
&= \frac{1}{2Z_{ij}} | \|v_i\| |v_i\rangle - \|v_j\| |v_j\rangle |^2 \\
&= \frac{1}{2Z_{ij}} (\|v_i\|^2 + \|v_j\|^2 - 2 \|v_i\| \|v_j\| \langle v_i|v_j\rangle) \\
&= \frac{d(v_i, v_j)^2}{2Z_{ij}}
\end{aligned} \tag{6.8}
$$

Thus, the probability encodes the squared distance between v_i and v_j, such that:

$$
p_{ij} = \frac{1}{2} + \frac{d(v_i, v_j)^2}{4Z_{ij}} \tag{6.9}
$$

The quantum state before measuring the ancilla qubit in the swap test circuit can be written as,

$$
|i\rangle |j\rangle (\sqrt{p_{ij}} |y_{ij}\rangle + \sqrt{1 - p_{ij}} |y_{ij}^{\perp}\rangle) \tag{6.10}
$$

Finally, we can apply this transformation over a superposition of indices i and j, thus allowing to compute many pairwise distances simultaneously.

In Section 6.2, we will propose a different method to obtain a similar quantum state as Eq.(6.10), and show how to extract the value $d(v_i, v_j)^2$ from the amplitude.

6.1.2 Unary Inner Products

Recently, [JDM+20] used the unary encoding (Definition 4.3) and unary data loaders (Section 4.1.2) to perform the inner product between two vectors.

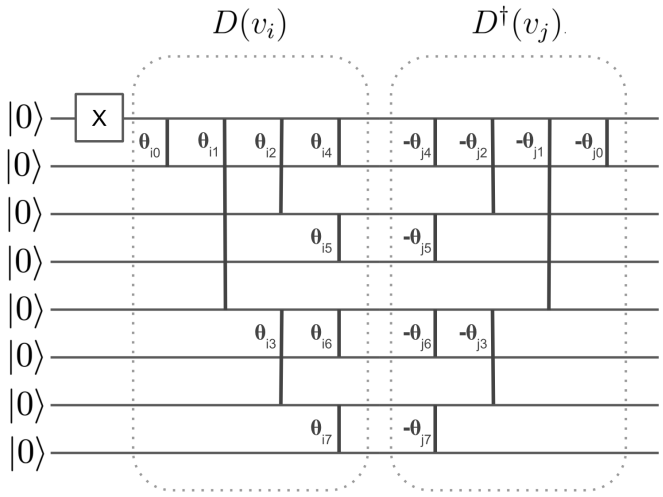


Figure 6.2: The inner product circuit for two unary quantum vectors of dimension 8 is the concatenation of the two data loaders $D^\dagger(v_j)D(v_i)$.

We consider two vectors $v_i \in \mathbb{R}^d$ and $v_j \in \mathbb{R}^d$, with respective address i and j, stored in a unary memory model. We write $D(v_i)$ and $D(v_j)$ their respective loader circuits, for instance the parallel loader from Fig.4.5b.

$$D(v_i)\,|0\rangle = |v_i\rangle = \frac{1}{\sqrt{d}} \sum_{i}^{d} v_i\,|e_i\rangle \tag{6.11}$$

where e_i is the i^{th} unary vector (e.g. $|0\cdots010\cdots0\rangle$). Note that $|v_i\rangle$ can be projected onto the second vector, and thus expressed in the basis $\{v_j, v_j^{\perp}\}$ as

$$|v_i\rangle = \langle v_i|v_j\rangle\,|v_j\rangle + \sqrt{1-\langle v_i|v_j\rangle^2}\,|v_j^{\perp}\rangle \tag{6.12}$$

Recall that the unary data loaders from [JDM+20] are using only $RBS(\theta)$ gates from Eq.4.13 and it is simple to show that $RBS(\theta) = RBS^{\dagger}(-\theta)$. The concatenation of the two circuits $D^{\dagger}(v_j)D(v_i)$ will therefore output the state:

$$\begin{aligned} D^{\dagger}(v_j)D(v_i) &= D^{\dagger}(v_j)\left(\langle v_i|v_j\rangle\,|v_j\rangle + \sqrt{1-\langle v_i|v_j\rangle^2}\,|v_j^{\perp}\rangle\right) \\ &= \langle v_i|v_j\rangle\,|e_1\rangle + \sqrt{1-\langle v_i|v_j\rangle^2}\,|e_1^{\perp}\rangle \end{aligned} \tag{6.13}$$

Using parallel data loaders as shown in Fig.6.2, the depth of the circuit is $O(2\log(d))$, using O(d) qubits. If the goal is to retrieve a classical approximation of $\langle v_i|v_j\rangle$, one can perform several measurements to estimate $|\langle v_i|v_j\rangle|^2$, being the probability to the number of times a '1' is seen on the first qubit.

Notably, we remark that recovering the inner product would be exact only for vectors with the same sign. To generalize and get the sign of the inner product, one can add an extra qubit in superposition to control the whole circuit, which will result in an amplitude of $((1+\langle v_i|v_j\rangle)/2)$.

6.2 Quantum Circuit for Inner Product and Distance Estimation

In this section, we introduce a quantum algorithm for inner product and distance estimation between amplitude encoded vectors, applicable in superposition. Using the work of [WKS14a], we adapted the Frobenius distance estimator from [KL20] that calculates the average square distance between a single point and all set many other points. It allows us to calculate the square distance or inner product (with its sign) between two vectors. This routine becomes very efficient when having quantum access to the vectors (Definition 4.5).

As shown in Fig.1.6, this algorithm will be a core part of many others during this thesis: matrix multiplication, neural network's layer, convolution product, adjacency graph creation, etc.

6.2.1 Quantum Circuit

We first state the Theorem that summarizes the circuit complexity and guarantees, from [KLLP19]:

Theorem 6.1: Distance & Inner Product Estimation

Given quantum access in time T to two data matrices $V \in \mathbb{R}^{N\times d}$ and $C \in \mathbb{R}^{k\times d}$ with rows v_i and v_j. For any $\Delta > 0$ and $\epsilon > 0$, there exists a quantum algorithm that computes

1. The squared distance: $|i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle|\overline{d^2(v_i,v_j)}\rangle$ where $|\overline{d^2(v_i,v_j)} - d^2(v_i,v_j)| \leqslant \epsilon$ with probability at least $1-2\Delta$, in time $O\left(\frac{1}{\epsilon}\|v_i\|\|v_j\| T\log(1/\Delta)\right)$.

2. The unnormalized inner product: $|i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle|\overline{(v_i,v_j)}\rangle$ where $|\overline{(v_i,v_j)} - (v_i,v_j)| \leqslant \epsilon$ with probability at least $1-2\Delta$, in time $O\left(\frac{1}{\epsilon}\|v_i\|\|v_j\| T\log(1/\Delta)\right)$.

Both of these tasks can be applied in superposition over all vectors of V and C, each with a running time upper bounded by $O\left(\frac{1}{\epsilon}\eta T\log(1/\Delta)\right)$ where $\eta = \max_{i,j}(\|v_i\|\|v_j\|)$ with the assumption $\min_i(\|v_i\|) = \min_i(\|v_j\|) = 1$.

Remark that if a QRAM model is used (Theorem 4.1), the complexity becomes simply $\widetilde{O}(\eta/\epsilon)$. Note also that the error guarantees can be reformulate for a relative error $|\overline{(v_i,v_j)} - (v_i,v_j)| \leqslant \epsilon(v_i,v_j)$ which modifies the running time to be $O\left(\frac{1}{\epsilon}\frac{\|v_i\|\|v_j\|}{|(v_i,v_j)|}T\log(1/\Delta)\right)$

The parameter $\eta(\cdot)$ will be present in many running times from now on. To this end, we introduce a proper definition:

Definition 6.1: Parameter $\eta(\cdot)$

For a matrix $V \in \mathbb{R}^{n\times d}$, its parameter $\eta(V)$ is defined as

$$\eta(V) = \frac{\max_i(\|v_i\|^2)}{\min_i(\|v_i\|^2)} \tag{6.14}$$

or as $\max_i(\|v_i\|^2)$ if we assume $\min_i(\|v_i\|) = 1$.

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

$$|\phi_{ij}\rangle := |i\rangle\,|j\rangle\,\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,|0\rangle \tag{6.15}$$

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 $|i\rangle\,|j\rangle\,|1\rangle\,|0\rangle \mapsto |i\rangle\,|j\rangle\,|1\rangle\,|v_j\rangle$. The state after this is given by,

$$\frac{1}{\sqrt{2}}\left(|i\rangle\,|j\rangle\,|0\rangle\,|v_i\rangle + |i\rangle\,|j\rangle\,|1\rangle\,|v_j\rangle\right) \tag{6.16}$$

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

$$|i\rangle\,|j\rangle\left(\frac{1}{2}\,|0\rangle\,(|v_i\rangle + |v_j\rangle) + \frac{1}{2}\,|1\rangle\,(|v_i\rangle - |v_j\rangle)\right) \tag{6.17}$$

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

$$p_{ij} = \frac{1}{4}(2 - 2\langle v_i|v_j\rangle) = \frac{1}{4}d^2(|v_i\rangle, |v_j\rangle) = \frac{1 - \langle v_i|v_j\rangle}{2} \tag{6.18}$$

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

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

$$A : |i\rangle\,|j\rangle\,|0\rangle \mapsto |i\rangle\,|j\rangle\,(\sqrt{p_{ij}}\,|y_{ij}, 1\rangle + \sqrt{1 - p_{ij}}\,|G_{ij}, 0\rangle) \tag{6.19}$$

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 $T_A = \tilde{O}(T)$.

We then use amplitude estimation (Theorem 3.4) on the unitary A defined in Eq.(6.19). This creates an unitary operation that maps

$$\mathcal{U} : |i\rangle\,|j\rangle\,|0\rangle \mapsto |i\rangle\,|j\rangle\left(\sqrt{\alpha}\,|\overline{p_{ij}}, G, 1\rangle + \sqrt{(1-\alpha)}\,|G', 0\rangle\right) \tag{6.20}$$

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 then make use of a tool developed in [WKS14a] to boost the probability and precision for the estimation of the distances or inner products. At high level, it takes

multiple copies of the estimator from the amplitude estimation procedure, and uses them to compute the median value. It finally reverses the circuit to get rid of the garbage states. We provide more details after the end of the current proof. Here we provide a theorem with respect to time and not query complexity.

Theorem 6.2: Median Evaluation

Let U be a unitary operation that maps

$$U : |0\rangle \mapsto \sqrt{a}\,|x,1\rangle + \sqrt{1-a}\,|G,0\rangle \tag{6.21}$$

for some $1/2 < a \leq 1$ in time T. Then there exists a quantum algorithm that uses L copies of the above state, and for any $\Delta > 0$ and for any $1/2 < a_0 \leq a$, produces a state $|\Psi\rangle$ such that $\left\| |\Psi\rangle - |0\rangle^{\otimes L} |x\rangle \right\| \leq \sqrt{2\Delta}$, in time:

$$2T \left\lceil \frac{\ln(1/\Delta)}{2\left(|a_0| - \frac{1}{2}\right)^2} \right\rceil. \tag{6.22}$$

We apply Theorem 6.2 for the unitary $\mathcal{U}$ to obtain a quantum state $|\Psi_{ij}\rangle$ such that,

$$\left\| |\Psi_{ij}\rangle - |0\rangle^{\otimes L} |\overline{p_{ij}}, G\rangle \right\|_2 \leq \sqrt{2\Delta} \tag{6.23}$$

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

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 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 $\epsilon \|v_i\| \|v_j\|$ and hence if we want to have in the end an absolute error ϵ this will incur a factor of $\|v_i\| \|v_j\|$ in the running time. This concludes the proof of the Theorem 6.1. □

Median Evaluation Details

As stated in Theorem 6.2 from [WKS14a], we can boost the probability of measuring the right state during inner product estimation. The improved precision can be arbitrary with only a logarithmic dependence in the runtime. It also requires L new copies of the current state, which fortunately is also logarithmic in the precision.

As before, if we can obtain the state $\sqrt{p}\,|y\rangle + \sqrt{1-p}\,|y^{\perp}\rangle$, with p encoding the meaningful information such as a distance or else, we can apply the amplitude estimation and get in time T:

$$\alpha\,|y..\rangle\,|\overline{p}\rangle + \alpha'\,|y^{\perp}..\rangle\,|\overline{p}^{\perp}\rangle \tag{6.24}$$

with the informal notation $|y..\rangle$ to denote the potential presence of garbage states, and with $\overline{p}$ being an ϵ-close estimate of p using m-qubits for precision. We know from Theorem 3.4 that $|\alpha|^2 \geq 8/\pi^2$.

Then, to increase the probability of measuring $|\overline{p}\rangle$ and not $|\overline{p}^{\perp}\rangle$, we will use L copies of the state in Eq.(6.24), $(\alpha\,|y..\rangle\,|\overline{p}\rangle + \alpha'\,|y^{\perp}..\rangle\,|\overline{p}^{\perp}\rangle)^{\otimes L}$. We will apply a median evaluation unitary on these L states. This operations write in an extra register the median value $|\tilde{P}\rangle$ observed.

$$(\alpha\,|y..\rangle\,|\overline{p}\rangle + \alpha'\,|y^{\perp}..\rangle\,|\overline{p}^{\perp}\rangle)^{\otimes L}\,|0\rangle^{\otimes m} \mapsto (\alpha\,|y..\rangle\,|\overline{p}\rangle + \alpha'\,|y^{\perp}..\rangle\,|\overline{p}^{\perp}\rangle)^{\otimes L}\,|\tilde{P}\rangle \tag{6.25}$$

To understand what is $|\tilde{P}\rangle$, we index each of the L copies gives, such that:

$$\left(\alpha\,|..\overline{p_1}\rangle + \alpha'\,|..\overline{p_1}^{\perp}\rangle\right) \otimes \left(\alpha\,|..\overline{p_2}\rangle + \alpha'\,|..\overline{p_2}^{\perp}\rangle\right) \otimes \cdots \otimes \left(\alpha\,|..\overline{p_L}\rangle + \alpha'\,|..\overline{p_L}^{\perp}\rangle\right) \tag{6.26}$$

This expression, once developed, is informally equivalent to:

$$(|\overline{p_1}\rangle\,|\overline{p_2}\rangle \cdots |\overline{p_L}\rangle) + \left(|\overline{p_1}^{\perp}\rangle\,|\overline{p_2}\rangle \cdots |\overline{p_L}\rangle\right) + \cdots + \left(|\overline{p_1}^{\perp}\rangle\,|\overline{p_2}^{\perp}\rangle \cdots |\overline{p_L}^{\perp}\rangle\right) \tag{6.27}$$

For each term were half of the values are $|\overline{p_i}\rangle$ and not $|\overline{p_i}^{\perp}\rangle$, the median evaluation will gives the right result $|\overline{p}\rangle$. The proof of Theorem 6.2 ensures that the final state $|\tilde{P}\rangle$ is sufficiently close to $\overline{p}$, and that if we reverse the whole computation, we obtain a state $|\Psi\rangle$ such that:

$$\left\| |\Psi\rangle - |0\rangle^{\otimes L}\,|\overline{p}\rangle \right\|_2 \leq \sqrt{2\Delta} \tag{6.28}$$

Where $\Delta > 0$ is an arbitrary precision parameter linked to the number L of copies requires:

$$L = \left\lceil \frac{\ln(\Delta^{-1})}{2(|\alpha| - \frac{1}{2})^2} \right\rceil \tag{6.29}$$

We see that the number L of copies is logarithmic in the precision Δ, which is efficient since the median evaluation unitary has a complexity of $O(L)$. Finally, the overall time is given by $O(2TL)$, where the factor 2 is for reversing the whole computation:

$$O\left(2T\left\lceil \frac{\ln(\Delta^{-1})}{2(|\alpha| - \frac{1}{2})^2} \right\rceil\right) \tag{6.30}$$

Note that Eq.(6.28) can be interpreted as a result on the probability of observing the right state as well. Indeed, if $\||a\rangle - |b\rangle\|_2 \leq \sqrt{\epsilon}$, then we can show that $|a\rangle = \beta\,|b\rangle + \beta'\,|G\rangle$ with $\beta \geq \sqrt{1-\epsilon}$ and $\beta' \leq \sqrt{\epsilon}$. $|G\rangle$ is the undesired state. This implies a probability at least $1 - 2\Delta$ of observing the good state $|0\rangle^{\otimes L}\,|\overline{p}\rangle$ from $|\Psi\rangle$.

Finally, we can extend this demonstration to the case of k tensor products as it will be used in Chapter 8. The logic stays the same, but the error between $|\Psi\rangle$ is now upper bounded by $\sqrt{2\Delta k}$, by a Union bound argument. indeed, following Eq.(6.28), the probability to have one bad results out k is $2\Delta k$.

6.2.2 Error Analysis

Since the quantum inner product or distance estimation algorithm (Theorem 6.1) will be used frequently, it is important to understand exactly the error it implies for the rest of the computations. Since the algorithm relies on amplitude estimation (Theorem 3.4), itself relying on phase estimation (Theorem 3.1), we can cascade back the error.

At first, the phase estimation algorithm [BHMT02] allows us to compute in superposition the phases ω of the eigenvectors $e^{i\pi\omega}$ of a unitary. For one value ω, the phase estimation circuit with n qubits will return the state $|\overline{\omega}\rangle = \sum_x \alpha(k)\,|k\rangle$ where k are series of integers between 0 and 2^n. We define the error $\delta = |\omega - k/2^n|$. The probability of obtaining a specific integer k when measuring the output is given by:

$$p(k) = |\alpha(k)|^2 = \frac{1}{2^{2^n}} \frac{\sin^2(\pi(2^n\delta - k))}{\sin^2(\pi(\delta - k/2^n))} \tag{6.31}$$

The most probable k we would measure is the one such that $k/2^n$ is the closest to ω. Unless $\omega = k/2^n$, in which case the computation is perfect. Note that the error made on the estimation of ω strongly depends on the value of ω itself. See Fig.6.4 for numerical simulations of the distributions.

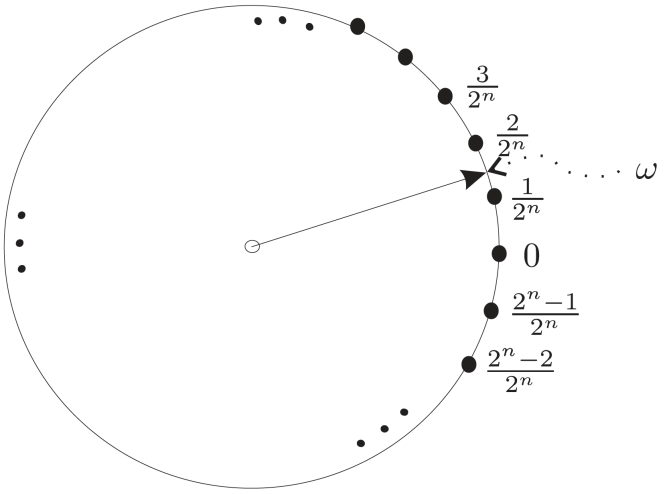


Figure 6.3: We estimate ω with the closest $k/2^n$ for any integer $k \in [0, 2^n]$. Source: [KLM+07].

Later on, this error is modified during amplitude estimation and the quantum circuit from 6.2.1. It is worth noticing that to obtain the unnormalized inner product, or the distance, the error is multiplied by the product of the norm of the two vectors. Therefore, the magnitude of the vectors will increase the absolute error (but not the relative one) as we can see in Fig.6.5.

More precisely, for two vectors v_i and v_j, the amplitude we want to estimate is:

$$a = \frac{1}{2}\left(\frac{\langle v_i|v_j\rangle}{\|v_i\|\,\|v_j\|} + 1\right) \tag{6.32}$$

Since phase estimation allows to obtain the phase ω of an amplitude of the form $sin(\pi\omega)$, we further want to obtain the actual normalized inner product:

$$\omega = \frac{1}{\pi}\arcsin(a) \tag{6.33}$$

As explained above, phase estimation will output, with some probability (see Eq.(6.31)), the closest value to ω of the form $k/2^n$. We write $b = \text{argmin}_{k\in[2^n]}(|k/2^n - \omega|)$. The actual amplitude obtained is an element of the set $\{\overline{\omega}_\ell\}_{\ell\in[2^n]}$ such that:

$$\overline{\omega}_\ell = \frac{1}{2^n}(b + \ell \mod (2^n)) \tag{6.34}$$

Each $\overline{\omega}_\ell$ yields an approximated amplitude $\overline{a}_\ell = \sin(\pi\overline{\omega}_\ell)$ which finally gives the result:

$$\overline{x}_\ell = \|v_i\|\,\|v_j\|\,(2\overline{a}_\ell - 1) \tag{6.35}$$

With probability:

$$p(\ell) = \frac{1}{2^{2^n}}\frac{\sin^2(\pi(2^n\delta - \ell))}{\sin^2(\pi(\delta - \ell/2^n))} \tag{6.36}$$

Note that we don't include median evaluation (Theorem 6.2) in this analysis, which could have improved the precision arbitrarily.

6.2.3 Numerical Simulations

To gain more intuition on the numerical errors made by the quantum algorithms, we perform extensive classical simulations[1] (see Section 3.4.2 for the methodology).

First, we plotted the distribution from Eq.(6.31) along with the inner product estimation that follows, for two particular vectors. The results are given in Fig.6.4. We can see that the more qubits, the precise is the distribution around the exact value of the inner product. Recall from the previous section that the shape of the distribution depends on the value itself. With more qubits, more values with high probabilities are close to the exact value, and therefore the estimation is more and more precise.

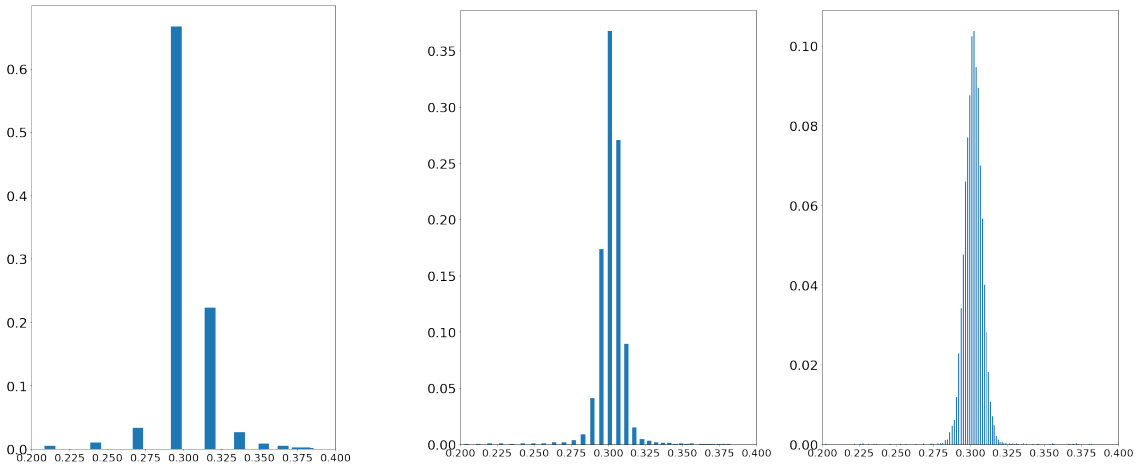
To gain one dimension, we performed the same simulations by varying one of the two input vectors. We chose two vectors of two dimensions but with only one element of one vector is shown in the x-axis, so that the inner product is shown in the y-axis. The first vector was $(0.544, 0.6)$ and the variable vector $(x, 1)$.

The results are given in Figures 6.5 and 6.6 for increasing number of qubits n. For each, we sampled randomly 500000 points x and generate an estimation of the inner product using the simulation. In the ideal case, the result of such a simulation should be a straight line $f(x) = \lambda x$ with $\lambda = 0.6 + 0.544x$. For each graph, we observe a general tendency to follow this line, accompanied by oscillations around it, depending on the x value and the number of qubits.

The parts of the oscillations with high amplitude show where the error is expected to be the highest. As expected, the error increases on average when the norm of the vector increases.

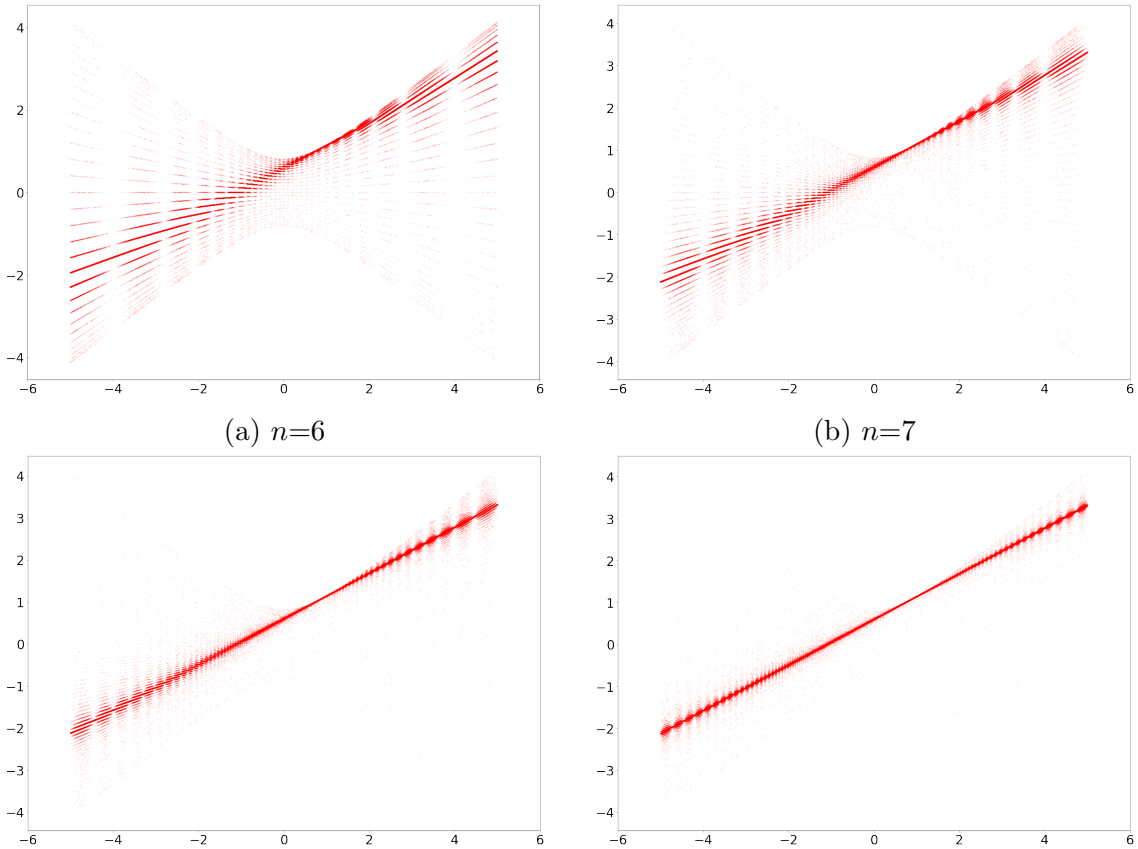
Notably, with a small number of qubits such as $n = 10$, we are already almost reproducing the ideal line with low noise around.

[1]The simulation program was made by Noah Berner



(a) n=6 (b) n=8 (c) n=10

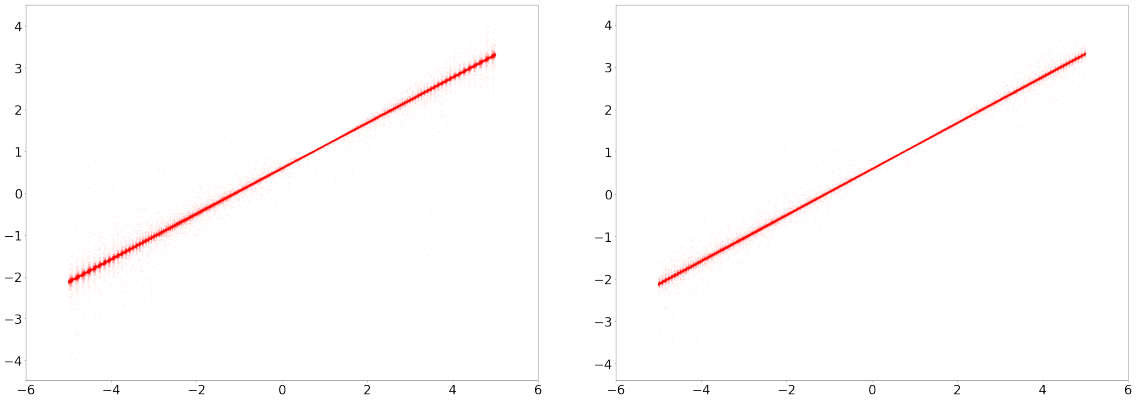
Figure 6.4: Probability distribution for the output of an inner product estimation with $x = (0.25, 0.3, -0.5)$ and $y = (-0.15, +0.3, -0.5)$, $\langle x|y\rangle = 0.302$, for different number of qubits n. The shape of the distribution depends on the value to estimate.



(a) n=6 (b) n=7

(c) n=8 (d) n=9

Figure 6.5: Classical simulation results of the Quantum Inner Product Estimation algorithm. The y-axis is the result of the quantum inner product between a fixed vector $(0.544, 0.6)$ and a variable vector $(x, 1)$, where x is the x-axis. The estimation is repeated over 500000 points, simulated for different number n of qubits.



(a) n=10 (b) n=11

Figure 6.6: Classical simulation results of the Quantum Inner Product Estimation algorithm. The y-axis is the result of the quantum inner product between a fixed vector $(0.544, 0.6)$ and a variable vector $(x, 1)$, where x is the x-axis. The estimation is repeated over 500000 points, simulated for different number n of qubits.

Finally, we couldn't help but think of the famous interference pattern one can see in the diffraction of a laser beam passing through a small slit or obstacle. This pattern is the signature of a wave-like phenomenon such as light, but also matter at its smallest scale, as quantum physics tell us.

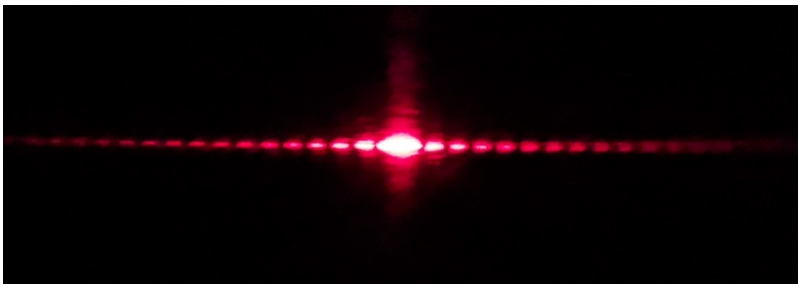


Figure 6.7: A quantum interference pattern produced by the diffraction of a laser beam. Source: Institute of Quantum Computing, University of Waterloo.

Note that in the rest of this thesis, when trying to simulate quantum algorithms classically, we will approximate the quantum noise of inner product or distance estimation by Gaussian noise with mean 0 and variance ϵ, multiplied by the norm of the input vectors. We have seen that this is not exactly what is occurring in theory. The noise at a specific value may seem Gaussian (see Fig.6.4), but in fact it depends on the value itself, which makes it non-isotropic and periodic. That being said, Gaussian noise remains a simpler and fairly accurate approximation of the above effect.

Part III

Quantum Unsupervised Learning

Chapter 7

Introduction

"La science consiste à passer d'un étonnement à un autre."

Aristote
La Métaphysique

In this Part, we are interested in unsupervised learning and in particular in the canonical problem of clustering: given a dataset represented as N vectors, we want to find an assignment of the vectors to one of k labels (for a given k that we assume to know) such that similar vectors are assigned to the same cluster. Often, the Euclidean distance is used to measure the similarity of vectors, but other metrics might be used, according to the problem under consideration. Details of the classical algorithms are given in Chapter 2.

In Chapter 8, we propose q-means [KLLP19], a quantum algorithm for clustering, which can be viewed as a quantum alternative to the classical k-means algorithm. More precisely, q-means is the equivalent of the δ-k-means algorithm, a robust version of k-means that will be defined later. We provide a detailed analysis to show that q-means has an output consistent with the classical δ-k-means algorithm and further has a running time that depends polylogarithmically on N, the number of elements in the dataset. We introduce the notion of *well-clusterable* dataset, allowing us to further refine the final running time. The last part of this work includes simulations which asserts the performance and running time of the q-means algorithm.

In Chapter 9, we propose the quantum spectral clustering algorithm [KL21], a quantum analog of the spectral clustering algorithm. With roots in graph theory, it uses the spectral properties of the Laplacian matrix to project the data in a low dimensional space where clustering is more efficient. Despite its success in clustering tasks, spectral clustering suffers in practice from a fast-growing running time of $O(N^3)$, where N is the number of points in the dataset. In this work we propose an end-to-end quantum algorithm performing spectral clustering, extending several works in quantum machine learning.

The quantum algorithm is composed of two parts: the first is the efficient creation of the quantum state corresponding to the projected Laplacian matrix, and the second consists of applying q-means algorithm presented before. Both steps depend

polynomially on the number of clusters, as well as precision and data parameters arising from quantum procedures, and polylogarithmically on the dimension of the input vectors. Our numerical simulations show an asymptotic linear growth with N when all terms are taken into account, significantly better than the classical cubic growth.

This work opens the path to other graph-based quantum machine learning algorithms, as it provides routines for efficient computation and quantum access to the Incidence, Adjacency, and projected Laplacian matrices of a graph.

Related Work

In this section, we discuss previous work on quantum unsupervised learning and clustering. Aimeur, Brassard and Gambs [ABG13] gave two quantum algorithms for unsupervised learning using the amplification techniques from [DH96]. Specifically, they proposed an algorithm for clustering based on minimum spanning trees that runs in time $\Theta(N^{3/2})$ and a quantum algorithm for k-median (a problem related to k-means) algorithm with complexity time $O(N^{3/2}/\sqrt{k})$.

Lloyd, Mohseni and Rebentrost [LMR13] proposed quantum k-means and nearest centroid algorithms using an efficient subroutine for quantum distance estimation assuming as we do quantum access to the data. Given a dataset of N vectors in a feature space of dimension d, the running time of each iteration of the clustering algorithm (using a distance estimation procedure with error ϵ) is $O(\frac{kN \log d}{\epsilon})$ to produce the quantum state corresponding to the clusters. Note that the time is linear in the number of data points and it will be linear in the dimension of the vectors if the algorithm needs to output the classical description of the clusters.

In the same work, they also proposed an adiabatic algorithm for the assignment step of the k-means algorithm, that can potentially provide an exponential speedup in the number of data points as well, in the case the adiabatic algorithm performs exponentially better than the classical algorithm. The adiabatic algorithm is used in two places for this algorithm, the first to select the initial centroids, and the second to assign data points to the closest cluster. However, while arguments are given for its efficiency, it is left as an open problem to determine how well the adiabatic algorithm performs on average, both in terms of the quality of the solution and the running time.

[WKS14a] applied the minimum finding algorithm [DH96] to obtain nearest-neighbor methods for supervised and unsupervised learning. At a high level, they recovered a Grover-type quadratic speedup with respect to the number of elements in the dataset in finding the k nearest neighbors of a vector. Otterbach et al. [OMA+17] performed clustering by exploiting a well-known reduction from clustering to the Maximum-Cut (MAXCUT) problem; the MAXCUT is then solved using QAOA, a quantum algorithm for performing approximate combinatorial optimization [FGG14].

There is extensive work in quantum computing involving graph problems such as min-cut, max-flow, or the traveling salesman problem [MLM17, CFS+16], but only a few are about graph-based machine learning [OMA+17, DFK+04]. Spectral clustering has been studied in [Das17] but no proven speedups were given. More recently, [AdW20] introduced quantum algorithms using the graph Laplacian for optimization and machine learning applications, including spectral clustering. In

that paper, the starting point was the assumption of having superposition-access to the classically-stored weights of the similarity graph, corresponding to the Laplacian directly, from which the authors performed tasks like sparsification of the graph faster than classical algorithms. In our work, we propose an efficient quantum algorithm to construct the projected Laplacian matrix itself from access to the classical input, before proceeding to the clustering algorithm. Note that one could eventually combine the methods from [AdW20] and our procedure, or any other procedure that provides access to the projected Laplacian matrix, and thus construct different quantum spectral clustering algorithms.

Chapter 8

Q-means

"Mathematics may be defined as the subject in which we never know what we are talking about, nor whether what we are saying is true."

Bertrand Russell
Mysticism and Logic and Other Essays (1910)

8.1 Preliminaries

Preliminaries on the k-means algorithm , along with all notations are given in Section 2.2.1.

8.1.1 Main Results

We define and analyze a new quantum algorithm for clustering, the q-means algorithm, whose performance is similar to that of the classical k-means algorithm defined in Section 2.2.1 and whose running time provides substantial savings, especially for the case of large data sets. To be more precise and to take into account all quantum randomness and noise, we define the classical δ-k-means algorithm in Section 8.1.2, a noisy version of the k-means that is equivalent to our quantum algorithm.

The q-means algorithm combines most of the advantages that quantum machine learning algorithms can offer for clustering. First, the running time is polylogarithmic in the number of elements of the dataset and depends only linearly on the dimension of the feature space. Second, q-means returns explicit classical descriptions of the cluster centroids that are obtained by the δ-k-means algorithm. As the algorithm outputs a classical description of the centroids, it is possible to use them in further (classical or quantum) classification algorithms.

Our q-means algorithm requires that the dataset is stored in a QRAM (see Theorem 4.1), which allows the algorithm to use efficient linear algebra routines (Part II) that have been developed using QRAM data structures. Of course, our algorithm can also be used for clustering datasets for which the data points can be efficiently

prepared even without a QRAM, for example if the data points are the outputs of quantum circuits.

We start by providing a worst case analysis of the running time of our algorithm, which depends on parameters of the data matrix, for example the condition number and the parameter μ that appears in the quantum linear algebra procedures. Note that with $\widetilde{O}$ we hide polylogarithmic factors.

Result 8.1: q-means - General Case

Given dataset $V \in \mathbb{R}^{N\times d}$ stored in QRAM, the q-means algorithm outputs with high probability centroids $c_1, \cdots, c_k$ that are consistent with an output of the δ-k-means algorithm in time

$$\widetilde{O}\left(kd\frac{\eta}{\delta^2}\kappa(V)(\mu(V)+k\frac{\eta}{\delta}) + k^2\frac{\eta^{1.5}}{\delta^2}\kappa(V)\mu(V)\right) \tag{8.1}$$

per iteration, where $\kappa(V)$ is the condition number, $\mu(V)$ is a parameter that appears in quantum linear algebra procedures and $1 \leq \|v_i\|^2 \leq \eta$.

When we say that the q-means output is consistent with the δ-k-means, we mean that with high probability the clusters that the q-means algorithm outputs are also possible outputs of the δ-k-means.

We go further in our analysis and study a well-motivated model for datasets that allows for good clustering. We call these datasets *well-clusterable*. One possible way to think of such datasets is the following: a dataset is well-clusterable when the k clusters arise from picking k well-separated vectors as their centroids, and then each point in the cluster is sampled from a Gaussian distribution with small variance centered on the centroid of the cluster. We provide a rigorous definition in the following sections. For such well-clusterable datasets, we can provide a tighter analysis of the running time and have the following result, whose formal version appears as Theorem 8.3.

Result 8.2: q-means - Well-Clusterable Case

Given a well-clusterable dataset $V \in \mathbb{R}^{N\times d}$ stored in QRAM, the q-means algorithm outputs with high probability k centroids $c_1, \cdots, c_k$ that are consistent with the output of the δ-k-means algorithm in time

$$\widetilde{O}\left(k^2 d\frac{\eta^{2.5}}{\delta^3} + k^{2.5}\frac{\eta^2}{\delta^3}\right) \tag{8.2}$$

per iteration, where $1 \leq \|v_i\|^2 \leq \eta$.

In order to assess the running time and performance of our algorithm, we performed extensive simulations for different datasets. The running time of the q-means algorithm is linear in the dimension d, which is necessary when outputting a classical description of the centroids, and polynomial in the number of clusters k which is typically a small constant. The main advantage of the q-means algorithm is that it provably depends logarithmically on the number of points, which can in many cases provide a substantial speedup. The parameter δ (which plays the same role as in the

δ-k-means) is expected to be a large enough constant that depends on the data and the parameter η is again expected to be a small constant for datasets whose data points have roughly the same norm. For example, for the MNIST dataset, η can be less than 8 and δ can be taken to be equal to 0.5. In Section 8.3 we present the results of our simulations. For different datasets we find parameters δ such that the number of iterations is practically the same as in the k-means, and the δ-k-means algorithm converges to a clustering that achieves an accuracy similar to the k-means algorithm or in times better. We obtained these simulation results by simulating the operations executed by the quantum algorithm adding the appropriate errors in the procedures.

As a side note, a generalization of the q-means algorithms has been proposed to solve the Gaussian mixture model in [KLP20b]. Instead of looking for centroids only, it also finds the covariance matrix for each cluster that approximates the multi dimensional Gaussian distribution of data points around the centroids. This ensures a more precise clustering and allows for clusters to be of different sizes and shapes.

8.1.2 δ-k-means

We now consider a δ-robust version of the k-means defined in Section 2.2.1, 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 us describe the rules for the assignment step of δ-k-means more precisely. Let c_i^* be the closest centroid to the data point v_i. Then, the set of possible labels $L_\delta(v_i)$ for v_i is defined as follows:

$$L_\delta(v_i) = \{c_p : |d^2(c_i^*, v_i) - d^2(c_p, v_i)| \leq \delta \} \tag{8.3}$$

The assignment rule selects arbitrarily a cluster label from the set $L_\delta(v_i)$.

Second, we add $\delta/2$ noise during the calculation of the centroid. Let $\mathcal{C}_j^{t+1}$ be the set of points which has been labeled by j in the previous step. For δ-k-means we pick a centroid c_j^{t+1} with the property

$$\left\| c_j^{t+1} - \frac{1}{|\mathcal{C}_j^{t+1}|} \sum_{v_i \in \mathcal{C}_j^{t+1}} v_i \right\| < \frac{\delta}{2} \tag{8.4}$$

One way to do this is to calculate the centroid exactly and then add some small Gaussian noise to the vector to obtain the robust version of the centroid.

Let us add two remarks on the δ-k-means. First, for a well-clusterable data set 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 $|L_\delta(v_i)| > 1$ 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.

8.1.3 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. Note that we do not need this assumption for our general q-means algorithm, but in this model we can provide tighter bounds for its running time. Without loss of generality we consider in the following 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. Our notion of a well-clusterable dataset shares some similarity with the assumptions made in [DKR02], but there are also some differences specific to the clustering problem.

Definition 8.1: 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:

1. (separation of cluster centroids): $d(c_i, c_j) \geq \xi \quad \forall i, j \in [k]$
2. (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.
3. (Intra-cluster smaller than inter-cluster square distances): The following inequality is satisfied

$$4\sqrt{\eta}\sqrt{\lambda\beta^2 + (1-\lambda)4\eta} \leq \xi^2 - 2\sqrt{\eta}\beta \tag{8.5}$$

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.

Claim 8.1

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

$$\|V - W\|_F^2 = \sum_{ij}(v_{ij} - w_{ij})^2 = \sum_i d(v_i, c_{l(v_i)})^2 \leq \lambda N\beta^2 + (1-\lambda)N4\eta \tag{8.6}$$

where we used Definition 8.1 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 5.2 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

Claim 8.2

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 Claim 8.1, we have,

$$\|V\|_F \leq \sqrt{(\lambda\beta^2 + (1-\lambda)4\eta)}\,\|V\|_F + \sqrt{k} \tag{8.7}$$

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.

Claim 8.3

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 we defined the notion of well-clusterable dataset is 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 clustered in k clusters. We next show that for a well-clusterable dataset V, there is a constant δ that can be computed in terms of the parameters in Definition 8.1 such that the δ-k-means clusters correctly most of the data points.

Claim 8.4

Let V be a well-clusterable data matrix. Then, for at least λN data points v_i, we have

$$\min_{j \neq \ell(i)} (d^2(v_i, c_j) - d^2(v_i, c_{\ell(i)})) \geq \xi^2 - 2\sqrt{\eta}\beta \tag{8.8}$$

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 8.1, 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,

$$d^2(v_i, c_j) - d^2(v_i, c_{\ell(i)}) \geq d^2(c_j, c_{\ell(i)}) - 2d(c_j, c_{\ell(i)})d(v_i, c_{\ell(i)})) \tag{8.9}$$

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.

□

8.2 The q-means Algorithm

8.2.1 Quantum Circuit

The q-means algorithm is given as Algorithm 2. At a high level, it follows the same steps as the classical k-means algorithm described in Section 2.2.1. Compared to k-means, q-means uses 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 some classical techniques, for example k-means++ [AV07]. Then, in Steps 1 and 2 all data points are assigned to clusters, and in Steps 3 and 4 we update the centroids of the clusters. The process is repeated until convergence.

Step 1: Centroid Distance Estimation

The first step of the algorithm estimates the square distance between data points and clusters using a quantum procedure. For this we used the method presented in Section 6.2, assuming quantum access to the vectors $\{v_i\}_{i \in [N}$ and centroids $\{c_j\}_{j \in [k}$.

Algorithm 2 q-means.

Require: Data matrix $V \in \mathbb{R}^{N\times d}$ stored in QRAM data structure. Precision parameters δ for k-means, error parameters ϵ_1 for distance estimation, ϵ_2 and ϵ_3 for matrix multiplication and ϵ_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 δ-k-means algorithm.

1: Select k initial centroids $c_1^0, \cdots, c_k^0$ and store them in QRAM data structure.
2: t=0
3: **repeat**
4: **Step 1: Centroid Distance Estimation**
Perform the mapping (Theorem 8.1)

$$\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]}|j\rangle|0\rangle \mapsto \frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]}|j\rangle|\overline{d^2(v_i,c_j^t)}\rangle \tag{8.10}$$

where $|\overline{d^2(v_i,c_j^t)} - d^2(v_i,c_j^t)| \leq \epsilon_1$.
5: **Step 2: Cluster Assignment**
Find the minimum distance among $\{d^2(v_i,c_j^t)\}_{j\in[k]}$ (Lemma 8.5), then uncompute Step 1 to create the superposition of all points and their labels

$$\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]}|j\rangle|\overline{d^2(v_i,c_j^t)}\rangle \mapsto \frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle|\ell^t(v_i)\rangle \tag{8.11}$$

6: **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 obtain the state $|c_j^{t+1}\rangle$ with error ϵ_2, along with an estimation of $\left\|c_j^{t+1}\right\|$ with relative error ϵ_3 (Theorem 5.2).
7: **Step 4: Centroid Update**
4.1 Perform tomography for the states $|c_j^{t+1}\rangle$ with precision ϵ_4 using the operation from Steps 1-3 (Theorem 4.2) 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 $\overline{c}_0^{t+1}\cdots\overline{c}_k^{t+1}$.
8: t=t+1
9: **until** convergence condition is satisfied.

For q-means, we need to estimate distances or inner products between vectors which have different norms. At a high level, if we first estimate the inner between the quantum states $|v_i\rangle$ and $|c_j\rangle$ corresponding to the normalized vectors and then multiply our estimator by the product of the vector norms we will get an estimator for the inner product of the unnormalized vectors. A similar calculation works for the square distance instead of the inner product. If we have an absolute error ϵ for the square distance estimation of the normalized vectors, then the final error is of the order of $\epsilon \|v_i\| \|c_j\|$.

We now rewrite the Theorem 6.1, suited for the setup and notations of q-means. The proof of the new theorem follows rather straightforwardly. In fact one just needs to apply the distance estimation procedure from Theorem 6.1 k times in parallel. 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.

Theorem 8.1: 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 $|i\rangle|0\rangle \mapsto |i\rangle|v_i\rangle$, 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

$$\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]} (|j\rangle|0\rangle) \mapsto \frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]} (|j\rangle|\overline{d^2(v_i,c_j)}\rangle) \tag{8.12}$$

where $|\overline{d^2(v_i,c_j)} - d^2(v_i,c_j)| \leqslant \epsilon_1$ with probability at least $1-2\Delta$, in time $\widetilde{O}\left(k\frac{\eta}{\epsilon_1}\log(1/\Delta)\right)$ where $\eta = \max_i(\|v_i\|^2)$.

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, as written in Eq.(8.12):

$$\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]} (|j\rangle|\overline{d^2(v_i,c_j)}\rangle) \tag{8.13}$$

We can now select the index j that corresponds to the centroid closest to the given data point, written as $\ell(v_i) = \text{argmin}_{j\in[k]}(d(v_i,c_j))$. As the square is a monotone function, we do not need to compute the square root of the distance in order to find $\ell(v_i)$.

Claim 8.5

Given k different $\log(p)$-bit registers $\otimes_{j\in[k]}|a_j\rangle$, there is a quantum circuit U_{min} that maps $(\otimes_{j\in[p]}|a_j\rangle)|0\rangle \to (\otimes_{j\in[k]}|a_j\rangle)|argmin(a_j)\rangle$ in time $O(k\log p)$.

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 Claim 8.5 and undo the computation we obtain the state

$$|\psi^t\rangle := \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle |\ell^t(v_i)\rangle . \tag{8.14}$$

Remark that if, instead of using the above minimum finding classical circuit, we used the algorithm from [DH96] as in [WKS14a], we could certainly save a factor $\widetilde{O}(\sqrt{k})$ in some part of the running time.

Step 3: Centroid State Creation

The previous step gave us the state $|\psi^t\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle |\ell^t(v_i)\rangle$. 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.

Claim 8.6

Let $\chi_j^t \in \mathbb{R}^N$ be the scaled characteristic vector for $\mathcal{C}_j$ at iteration t and $V \in \mathbb{R}^{N \times d}$ be the data matrix, then $c_j^{t+1} = V^T \chi_j^t$.

Proof. The k-means update rule for the centroids is given by $c_j^{t+1} = \frac{1}{|C_j^t|} \sum_{i \in C_j} v_i$. As the columns of V^T are the vectors v_i, this can be rewritten as $c_j^{t+1} = V^T \chi_j^t$. □

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.

$$|\psi^t\rangle = \sum_{j=1}^{k} \sqrt{\frac{|C_j|}{N}} \left(\frac{1}{\sqrt{|C_j|}} \sum_{i \in \mathcal{C}_j} |i\rangle \right) |j\rangle = \sum_{j=1}^{k} \sqrt{\frac{|C_j|}{N}} |\chi_j^t\rangle |j\rangle \tag{8.15}$$

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 V^T to recover an approximation of the state $|V^T \chi_j\rangle = |c_j^{t+1}\rangle$ with error ϵ_2, as stated in Theorem 5.2. Note that the error ϵ_2 only appears inside a logarithm. The same Theorem allows us to get an estimate of the norm $\left\|V^T \chi_j^t\right\| = \left\|c_j^{t+1}\right\|$ with relative error ϵ_3. For this, we also need

an estimate of the size of each cluster, namely the norms $\|\chi_j\|$. 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 5.2 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.

Step 4: Centroids 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 vector state tomography algorithm, stated in Theorem 4.2, 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 $\||c_j\rangle - |\overline{c_j}\rangle\| < \epsilon_4$. 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 $\||c_j\rangle - |\overline{c_j}\rangle\| < \epsilon_4$. Using the approximation of the norms $\|c_j\|$ 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 claim:

Claim 8.7

Let ϵ_4 be the error we commit in estimating $|c_j\rangle$ such that $\||c_j\rangle - |\overline{c_j}\rangle\| < \epsilon_4$, and ϵ_3 the error we commit in the estimating the norms, $|\,\|c_j\| - \overline{\|c_j\|}\,| \leq \epsilon_3 \|c_j\|$. Then $\|\overline{c_j} - c_j\| \leq \sqrt{\eta}(\epsilon_3 + \epsilon_4) = \epsilon_{centroid}$.

Proof. We can rewrite $\|c_j - \overline{c_j}\|$ as $\left\| \|c_j\| \, |c_j\rangle - \overline{\|c_j\|} \, |\overline{c_j}\rangle \right\|$. It follows from triangle inequality that:

$$\left\| \overline{\|c_j\|} \, |\overline{c_j}\rangle - \|c_j\| \, |c_j\rangle \right\| \leq \left\| \overline{\|c_j\|} \, |\overline{c_j}\rangle - \|c_j\| \, |\overline{c_j}\rangle \right\| + \left\| \|c_j\| \, |\overline{c_j}\rangle - \|c_j\| \, |c_j\rangle \right\| \tag{8.16}$$

We have the upper bound $\|c_j\| \leq \sqrt{\eta}$. 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. □

Let us make a remark about the ability to use Theorem 4.2 to perform tomography in our case. The updated centroids will be recovered in step 4 using the vector state tomography algorithm in Theorem 4.2 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 $\ell(v_i)$ 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 $|x\rangle$.

We therefore need a procedure that finds labels $\ell(v_i)$ that are a deterministic function of v_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 $\ell(v_i) = j$ if $\overline{d(v_i, c_j)} < \overline{d(v_i, c_{j'})} - 2\delta$ 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 [TS13, Amb12]. 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 $|\lambda - \overline{\lambda}| \leq \epsilon$.

We also describe another simple method of getting such consistent phase estimation, which is to combine phase estimation estimates that are obtained for two different precision values. Let us assume that the eigenvalues for the unitary U are $e^{2\pi i\theta_i}$ for $\theta_i \in [0, 1]$. First, we perform phase estimation with precision $\frac{1}{N_1}$ where $N_1 = 2^l$ is a power of 2. We repeat this procedure $O(\log N/\theta^2)$ times and output the median estimate. If the value being estimated is $\frac{\lambda+\alpha}{2^l}$ for $\lambda \in \mathbb{Z}$ and $\alpha \in [0, 1]$ and $|\alpha - 1/2| \geq \theta'$ for an explicit constant θ' (depending on θ) then with probability at least $1 - 1/\mathrm{poly}(N)$ the median estimate will be unique and will equal to $1/2^l$ times the closest integer to $(\lambda + \alpha)$. In order to also produce a consistent estimate for the eigenvalues for the cases where the above procedure fails, we perform a second phase estimation with precision $2/3N_1$. We repeat this procedure as above for $O(\log N/\theta^2)$ iterations and taking the median estimate. The second procedure fails to produce a consistent estimate only for eigenvalues $\frac{\lambda+\alpha}{2^l}$ for $\lambda \in \mathbb{Z}$ and $\alpha \in [0, 1]$ and $|\alpha - 1/3| \leq \theta'$ or $|\alpha - 2/3| \leq \theta'$ for a suitable constant θ'. Since the cases where the two procedures fail are mutually exclusive, one of them succeeds with probability $1 - 1/\mathrm{poly}(N)$. The estimate produced by the phase estimation procedure is therefore deterministic with very high probability. In order to complete this proof sketch, we would have to give explicit values of the constants θ and θ' and the success probability, using the known distribution of outcomes for phase estimation.

For what follows, we assume that indeed the state in Eq.(8.14) is almost a deterministic state, meaning that when we repeat the procedure we get the same state with very high probability.

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.

8.2.2 Analysis

We provide our general theorem about the running time and accuracy of the q-means algorithm.

Theorem 8.2: q-means - General Case

For a data matrix $V \in \mathbb{R}^{N\times d}$ stored in an appropriate QRAM data structure and parameter $\delta > 0$, the q-means algorithm with high probability outputs centroids consistent with the classical δ-k-means algorithm, in time

$$\widetilde{O}\left(kd\frac{\eta}{\delta^2}\kappa(V)(\mu(V)+k\frac{\eta}{\delta})+k^2\frac{\eta^{1.5}}{\delta^2}\kappa(V)\mu(V)\right) \tag{8.17}$$

per iteration, where $\kappa(V)$ is the condition number, $1 \leq \|v_i\|^2 \leq \eta$ and $\mu(M) = \min_{p\in[0,1]}(\|M\|_F, \sqrt{s_{2p}(M)s_{(1-2p)}(M^T)})$.

We prove the theorem in the following Sections and then provide the running time of the algorithm for well-clusterable datasets as Theorem 8.3.

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 so that firstly, 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

$$|\overline{d^2(c_j, v_i)} - d^2(c_j, v_i)| < \epsilon_1 \tag{8.18}$$

for a point v_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:

$$\forall j \in [k], \quad |\overline{d^2(c_j, v_i)} - d^2(c_j, v_i)| \leq \frac{\delta}{2} \tag{8.19}$$

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 8.7, namely

$$\|\bar{c}_j - c_j\| \leq \epsilon_{centroid} = \sqrt{\eta}(\epsilon_3 + \epsilon_4) \tag{8.20}$$

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.

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 5.2 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 Theorem 8.1.

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,

$$\widetilde{O}\left(kd\frac{1}{\epsilon_4^2}\kappa(V)\left(\mu(V) + k\frac{\eta}{\epsilon_1}\right) + k^2\frac{\eta}{\epsilon_3\epsilon_1}\kappa(V)\mu(V)\right) \tag{8.21}$$

The analysis in section 8.2.2 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

$$\widetilde{O}\left(kd\frac{\eta}{\delta^2}\kappa(V)\left(\mu(V) + k\frac{\eta}{\delta}\right) + k^2\frac{\eta^{1.5}}{\delta^2}\kappa(V)\mu(V)\right) \tag{8.22}$$

This completes the proof of Theorem 8.2. We next state our main result when applied to a well-clusterable dataset, as in Definition 8.1.

Theorem 8.3: q-means - Well-Clusterable Data

For a well-clusterable dataset $V \in \mathbb{R}^{N\times d}$ stored in appropriate QRAM, the q-means algorithm returns with high probability the k centroids consistently with the classical δ-k-means algorithm for a constant δ in time $\widetilde{O}\left(k^2 d\frac{\eta^{2.5}}{\delta^3} + k^{2.5}\frac{\eta^2}{\delta^3}\right)$ per iteration, for $1 \leq \|v_i\|^2 \leq \eta$.

Proof. Let $V \in \mathbb{R}^{N\times d}$ be a well-clusterable dataset as in Definition 8.1. In this case, we know by Claim 8.3 that $\kappa(V) = \frac{1}{\sigma_{min}}$ can be replaced by a thresholded condition number $\kappa_\tau(V) = \frac{1}{\tau}$. In practice, this is done by discarding the singular values smaller than a certain threshold during quantum matrix multiplication. Remember that by Claim 8.2 we know that $\|V\|_F = O(\sqrt{k})$. Therefore we need to pick ϵ_τ for a threshold $\tau = \frac{\epsilon_\tau}{\sqrt{k}}\|V\|_F$ such that $\kappa_\tau(V) = O(\frac{1}{\epsilon_\tau})$.

Thresholding the singular values in the matrix multiplication step introduces an additional additive error in $\epsilon_{centroid}$. By Claim 8.3 and Claim 8.7 , we have that the error $\epsilon_{centroid}$ in approximating the true centroids becomes $\sqrt{\eta}(\epsilon_3 + \epsilon_4 + \epsilon' + \epsilon_\tau)$

where $\epsilon' = \sqrt{\lambda\beta^2 + (1-\lambda)4\eta}$ is a dataset dependent parameter computed in Claim 8.1. We can set $\epsilon_\tau = \epsilon_3 = \epsilon_4 = \epsilon'/3$ to obtain $\epsilon_{centroid} = 2\sqrt{\eta}\epsilon'$.

The definition of the δ-k-means update rule requires that $\epsilon_{centroid} \leq \delta/2$. Further, Claim 8.4 shows that if the error δ in the assignment step satsifies $\delta \leq \xi^2 - 2\sqrt{\eta}\beta$, then the δ-k-means algorithm finds the corrects clusters. By Definition 8.1 of a well-clusterable dataset, we can find a suitable constant δ satisfying both these constraints, namely satisfying

$$4\sqrt{\eta}\sqrt{\lambda\beta^2 + (1-\lambda)4\eta} < \delta < \xi^2 - 2\sqrt{\eta}\beta \tag{8.23}$$

Substituting the values $\mu(V) = O(\sqrt{k})$ from Claim 8.2, $\kappa(V) = O(\frac{1}{\epsilon_\tau})$ and $\epsilon_\tau = \epsilon_3 = \epsilon_4 = \epsilon'/3 = O(\sqrt{\eta}/\delta)$ in the running time for the general q-means algorithm, we obtain that the running time for the q-means algorithm on a well-clusterable dataset is $\widetilde{O}\left(k^2 d\frac{\eta^{2.5}}{\delta^3} + k^{2.5}\frac{\eta^2}{\delta^3}\right)$ per iteration.

□

Let us make some 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 has 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.

8.2.3 Initialization: q-means++

We now show that the quantum analogue of the initialization procedure of k-means++ can be implemented efficiently using the square distance subroutine estimation for the q-means algorithm given in Theorem 8.1. Starting with a random index j we compute the state $\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle|j\rangle|d^2(v_i,v_j)\rangle$ in time $\tilde{O}(\frac{\eta}{\epsilon_1})$, where v_j is the initial centroid, using our quantum procedure for distance estimation. By applying some arithmetic preprocessing and a controlled rotation we can transfer the distance information as an amplitude to obtain the following state:

$$\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle|j\rangle|d^2(v_i,v_j)\rangle\left(\frac{d(v_i,v_j)}{2\sqrt{\eta}}|0\rangle + \beta|1\rangle\right) \tag{8.24}$$

Each square distance has been normalized by $2\sqrt{\eta} \geq max_{i,j}(d(v_i,v_j))$ to be a valid amplitude. Note that postselecting on $|0\rangle$ and measuring the register $|i\rangle$ samples exactly from the probability distribution in the k-means++ algorithm as the probability of measuring $(i,0)$ on second and fourth registers is $\frac{d^2(v_i,v_j)}{4\eta N}$.

We can perform amplitude amplification to boost the probability of measuring $|0\rangle$. For this we need to repeat $O(1/\sqrt{P(0)})$ times the previous steps, with $P(0)$ being the probability of measuring $|0\rangle$. Since $P(0) = \frac{1}{N}\left(\sum\frac{d(v_i,v_j)}{2\sqrt{\eta}}\right)^2$, it is simple

to show that $\frac{1}{\sqrt{P(0)}} \leq \frac{2\sqrt{\eta}}{\sqrt{\mathbb{E}(d^2(v_i,v_j))}}$, where $\mathbb{E}(d^2(v_i, v_j))$ is the mean squared distance. Note that for the next steps we can use a tensor product of the squared distance from previous centroids to compute the minimum distance among them, using Lemma 8.5. In the end we repeat $k-1$ times this circuit, for a total time of $\tilde{O}(k^2 \frac{2\eta^{1.5}}{\epsilon_1 \sqrt{\mathbb{E}(d^2(v_i,v_j))}})$. The running time for the q-means++ initialization is smaller than that for the q-means algorithm, showing than q-means++ initialization doesn't cancel any benefit of the q-means algorithm. Thus, we can use the q-means++ algorithm to provide a speedup compared to the classical k-means++.

8.3 Numerical Simulations

We would like to assert the capability of the quantum algorithm to provide accurate classification results, by simulating on several datasets. However, since neither quantum simulators nor quantum computers large enough to test q-means are available currently, we tested the equivalent classical implementation of δ-k-means. For implementing the δ-k-means, we changed the assignment step of the k-means algorithm to select a random centroid among those that are δ-close to the closest centroid and added $\delta/2$ error to the updated clusters.

We benchmarked our q-means algorithm on two datasets: a synthetic dataset of Gaussian clusters, and the well known MNIST dataset of handwritten digits. To measure and compare the accuracy of our clustering algorithm, we ran the k-means and the δ-k-means algorithms for different values of δ on a training dataset and then we compared the accuracy of the classification on a test set, containing data points on which the algorithms have not been trained, using a number of widely-used performance measures.

Gaussian clusters dataset

We describe numerical simulations of the δ-k-means algorithm on a synthetic dataset made of several clusters formed by random Gaussian distributions. These clusters are naturally well suited for clustering by construction, close to what we defined to be a well-clusterable dataset in Definition 8.1 of Section 8.1.3. Doing so, we can start by comparing k-means and δ-k-means algorithms on high accuracy results, even though this may not be the case on real-world datasets. Without loss of generality, we preprocessed the data so that the minimum norm in the dataset is 1, in which case $\eta = 4$. This is why we defined η as a maximum instead of the ratio of the maximum over the minimum which is really the interesting quantity. Note that the running time basically depends on the ratio η/δ. We present a simulation where 20.000 points in a feature space of dimension 10 form 4 Gaussian clusters with standard deviation 2.5, that we can see in Fig.8.1. The condition number of dataset is calculated to be 5.06. We ran k-means and δ-k-means for 7 different values of δ to understand when the δ-k-means becomes less accurate.

In Fig.8.2 we can see that until $\eta/\delta = 3$ (for $\delta = 1.2$), the δ-k-means algorithm converges on this dataset. We can now make some remarks about the impact of δ on the efficiency. It seems natural that for small values of δ both algorithms are equivalent. For higher values of δ, we observed a late start in the evolution of the accuracy, witnessing random assignments for points on the clusters' boundaries.

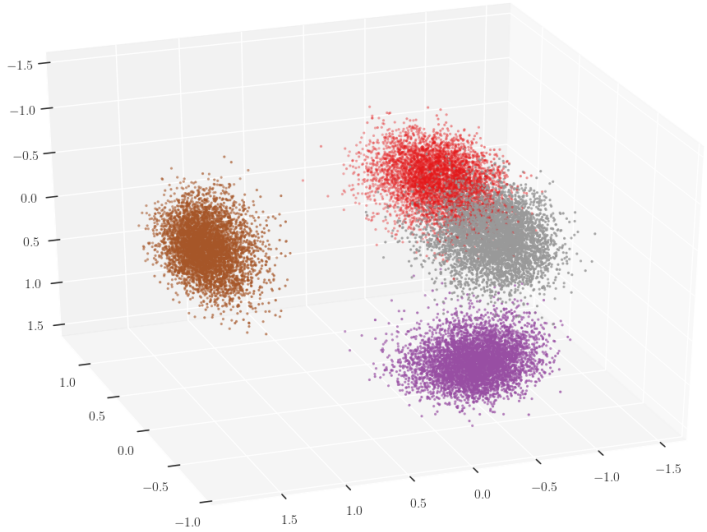


Figure 8.1: Representation of 4 Gaussian clusters of 10 dimensions in a 3D space spanned by the first three PCA dimensions.

However, the accuracy still reaches 100% in a few more steps. The increase in the number of steps is a tradeoff with the parameter η/δ.

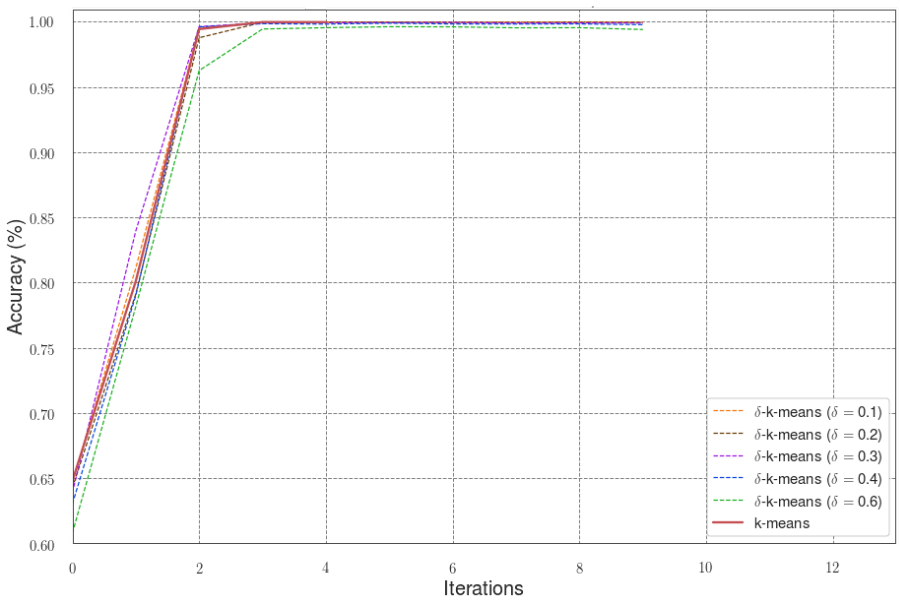


Figure 8.2: Accuracy evolution during k-means and δ-k-means on well-clusterable Gaussians for 5 values of δ. All versions converged to a 100% accuracy in few steps.

MNIST dataset

The MNIST dataset is composed of 60.000 handwritten digits as images of 28x28 pixels (784 dimensions). 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 [KL20], or other quantum dimensionality reduction algorithms like [LMR14, CD15].

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. Fig.8.3 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.

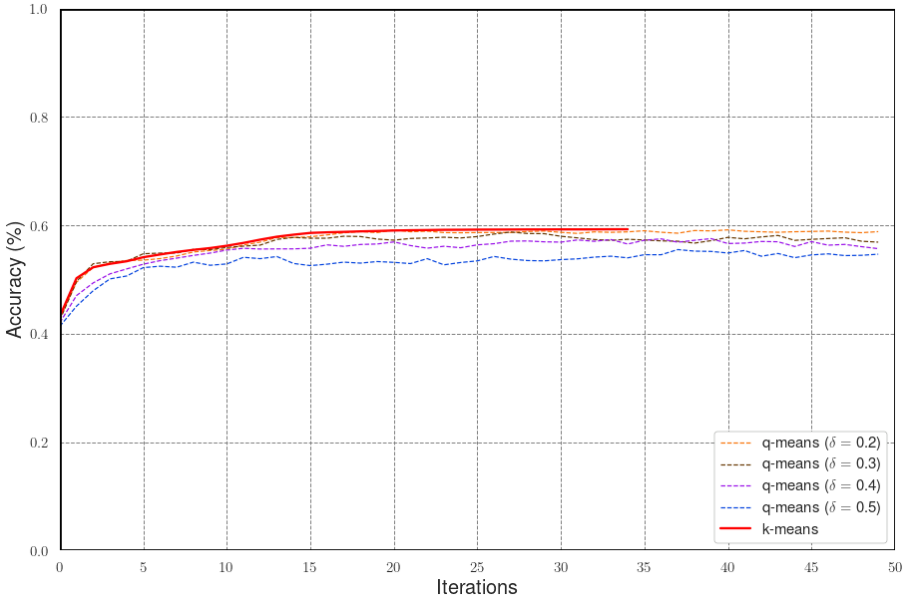


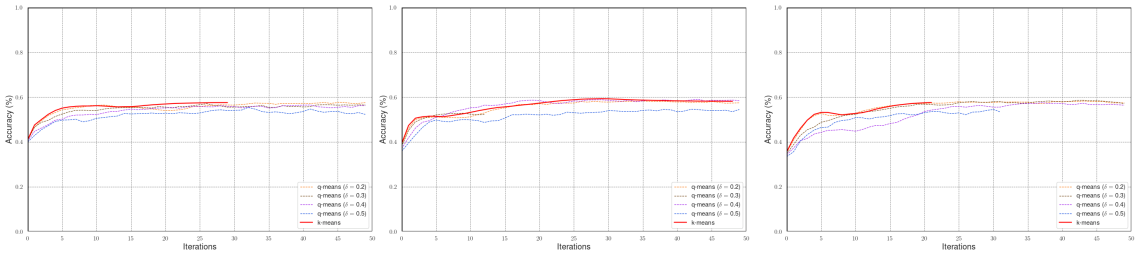
Figure 8.3: Accuracy evolution on the MNIST dataset under k-means and δ-k-means for 4 different values of δ. Data has been preprocessed by a PCA to 40 dimensions. All versions converge in the same number of steps, with a drop in the accuracy while δ increases. The apparent difference in the number of steps until convergence is just due to the stopping condition for k-means and δ-k-means.

On top of the accuracy measure (ACC), we also evaluated the performance of q-means against many other metrics, reported in Table 8.1. More detailed information about these metrics can be found in [FHT01]. We introduce a specific measure of error, the Root Mean Square Error of Centroids (RMSEC), which a direct comparison between the centroids predicted by the k-means algorithm and the ones predicted by the δ-k-means. It is a way to know how far the centroids are predicted. Note that this metric can only be applied to the training set. For all these measures, except RMSEC, a bigger value is better. Our simulations show that δ-k-means, and thus the q-means, even for values of δ (between $0.2 - 0.5$) achieves similar performance to k-means, and in most cases the difference is on the third decimal point.

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	-

Table 8.1: A sample of results collected from the same experiments as in Fig.8.3. 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.

These experiments have been repeated several times and each of them presented a similar behavior despite the random initialization of the centroids.



(a) (b) (c)

Figure 8.4: Three accuracy evolution on the MNIST dataset under k-means and δ-k-means for 4 different values of δ. Each different behavior is due to the random initialization of the centroids

Finally, we present a last experiment with the MNIST dataset with a different data preprocessing. In order to reach higher accuracy in the clustering, we replace the previous dimensionality reduction by a Linear Discriminant Analysis (LDA). Note that a LDA is a supervised process that uses the labels (here, the digits) to project points in a well chosen lower dimensional subspace. Thus this preprocessing cannot be applied in practice in unsupervised machine learning. However, for the sake of benchmarking, by doing so k-means is able to reach a 87% accuracy, therefore it allows us to compare k-means and δ-k-means on a real and almost well-clusterable dataset. In the following, the MNIST dataset is reduced to 9 dimensions. The results in Fig.8.5 and Table 8.2 show that δ-k-means converges to the same accuracy than k-means even for values of η/δ down to 16. In some other cases, δ-k-means shows a faster convergence, due to random fluctuations that can help escape faster from a temporary equilibrium of the clusters.

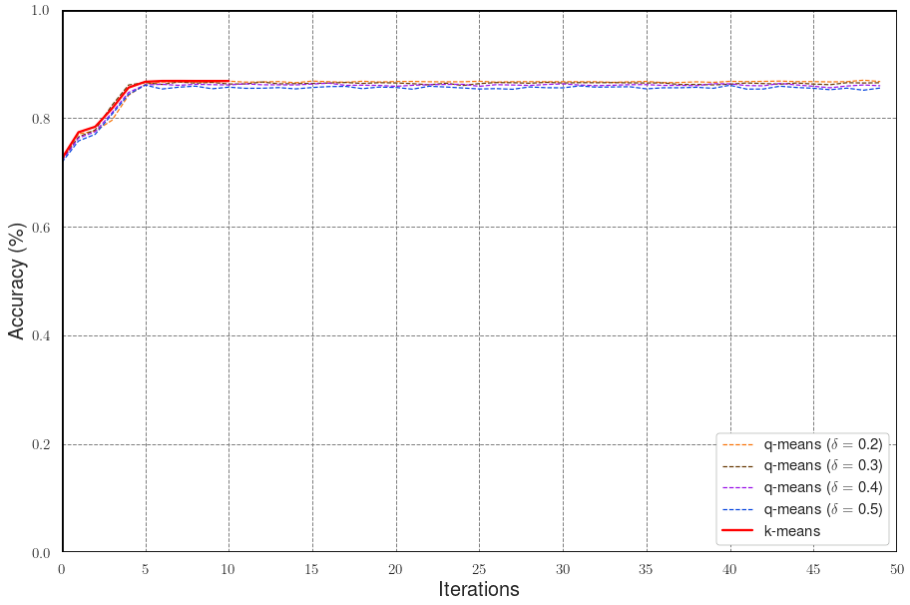


Figure 8.5: Accuracy evolution on the MNIST dataset under k-means and δ-k-means for 4 different values of δ. Data has been preprocessed to 9 dimensions with a LDA reduction. All versions of δ-k-means converge to the same accuracy than k-means in the same number of steps.

Algo	Dataset	ACC	HOM	COMP	V-M	AMI	ARI	RMSEC
k-means	Train	0.868	0.736	0.737	0.737	0.735	0.736	0
	Test	0.891	0.772	0.773	0.773	0.776	0.771	-
q-means ($\delta = 0.2$)	Train	0.868	0.737	0.738	0.738	0.736	0.737	0.031
	Test	0.891	0.774	0.775	0.775	0.777	0.774	-
q-means ($\delta = 0.3$)	Train	0.869	0.737	0.739	0.738	0.736	0.737	0.049
	Test	0.890	0.772	0.774	0.773	0.775	0.772	-
q-means ($\delta = 0.4$)	Train	0.865	0.733	0.735	0.734	0.730	0.733	0.064
	Test	0.889	0.770	0.771	0.770	0.773	0.769	-
q-means ($\delta = 0.5$)	Train	0.866	0.733	0.735	0.734	0.731	0.733	0.079
	Test	0.884	0.764	0.766	0.765	0.764	0.764	-

Table 8.2: A sample of results collected from the same experiments as in Fig.8.5. 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.

Let us remark, that the values of η/δ in our experiment remained between 3 and 20. Moreover, the parameter η, which is the maximum square norm of the points, provides a worst case guarantee for the algorithm, while one can expect that the running time in practice will scale with the average square norm of the points. For the MNIST dataset after PCA, this value is 2.65 whereas $\eta = 8.3$.

In conclusion, our simulations show that the convergence of δ-k-means is almost the same as the regular k-means algorithms for large enough values of δ. This

provides evidence that the q-means algorithm will have as good performance as the classical k-means, and its running time will be significantly lower for large datasets.

Chapter 9

Quantum Spectral Clustering

"All generalisations - perhaps except this one - are false."

Kurt Gödel

Preliminaries on spectral clustering, along with all notations are given in Section 2.2.2. Variables are summarized in Table 9.1.

Variable	Dimension	Remark
Input Matrix: S	$N \times d$	Each input vector is a row $s_i \in \mathbb{R}^d$
Adjacency Matrix: A	$N \times N$	Nodes connectivity. Elements $a_{ij} =$ 1 if $d(s_i, s_j) \leq d_{min}$ and 0 otherwise
Incidence Matrix: B	$N \times \frac{N(N-1)}{2}$	Elements $B_{i,(p,q)} = \pm a_{pq}$ if edge (p, q) is incident to node i
Normalized Incidence Matrix: $\mathcal{B}$	$N \times \frac{N(N-1)}{2}$	Each row is normalized Eigenvalues $\lambda_1^{\mathcal{B}} \leq \cdots \leq \lambda_n^{\mathcal{B}}$
Normalized Laplacian: $\mathcal{L}$	$N \times N$	$\mathcal{L} = \mathcal{B}\mathcal{B}^T$ Eigenvalues $\lambda_j = (\lambda_j^{\mathcal{B}})^2$
Projected Normalized Laplacian: $\widetilde{\mathcal{L}}^{(k)}$	$N \times k$	$\mathcal{L}$ projected on its k lowest eigenvectors

Table 9.1: Summary of variables for Classical and Quantum spectral clustering.

9.1 Main Results

In this work, we develop an end-to-end quantum algorithm for spectral clustering [NJW02]. As detailed in Section 2.2.2, this unsupervised learning algorithm shows great accuracy in identifying complex and interlacing clusters, and allows a high level of explainability. However, it suffers from a fast-growing runtime, namely cubic in the number N of vectors in the dataset, that inhibits its use in practice. The goal of the spectral clustering algorithm is to perform the clustering task in a low dimensional space derived from the data. More precisely, one starts with the set of input vectors and constructs a similarity graph, where the edge between two nodes is built

from the distance between the two associated vectors. From this graph, one can define the Incidence matrix and the Laplacian matrix. By extracting the eigenvectors of the Laplacian matrix and keeping the subspace spanned by the lowest ones, a lower dimensional space is defined. The initial data can then be projected onto this new space, where one can expect the data to be better organized, as clusters. Therefore, to obtain the clusters, the k-means algorithm is applied to the projected vectors.

We provide a quantum algorithm for spectral clustering following a similar methodology, while having to carefully extend or alter the specifics of each step of the algorithm. We first adapt and extend recent and efficient quantum subroutines for linear algebra and distance estimation. These methods are used to create the similarity graph, as well as the Incidence and Laplacian matrices, extract their eigenvectors and project the data points onto the right subspace to finally apply a quantum analog of k-means. While the steps of the algorithm follow the classical ones, we had to adjust most of the definitions, for example the definitions of the Incidence and Laplacian matrices, in order to make the quantum algorithm efficient. We will detail all the needed changes in the following sections.

In high level, the running time of the quantum spectral clustering algorithm reflects the two stages of spectral clustering and is given by

$$O(T_{\widetilde{\mathcal{L}}^{(k)}} T_{qmeans}) \tag{9.1}$$

The first term $T_{\widetilde{\mathcal{L}}^{(k)}}$ is the time to create a quantum state corresponding to the normalized Laplacian matrix of the graph projected on its lowest eigenvectors. The resulting quantum state is the input state to the quantum clustering algorithm whose overall running time contains another multiplicative term that we denote by T_{qmeans}.

For the first part, we will propose an algorithm in time

$$T_{\widetilde{\mathcal{L}}^{(k)}} = \widetilde{O}\left(T_S \frac{\eta(S)}{\epsilon_{dist}\epsilon_B} \frac{\mu(\mathcal{B})\kappa(\widetilde{\mathcal{L}}^{(k)})}{\epsilon_\lambda}\right). \tag{9.2}$$

Here, the term T_S is the time to load the input vectors in a quantum state, which becomes efficient is we assume fast quantum access (see Definition 4.5). The terms $\epsilon_{dist}, \epsilon_B$ and ϵ_λ correspond to error or precision parameters that appear in several quantum routines. The matrices $S, \mathcal{B}$, and $\widetilde{\mathcal{L}}^{(k)}$ are respectively the input data matrix, the normalized incidence matrix, and the projection of the normalized Laplacian matrix. For these matrices, the condition number $\kappa(\cdot)$ is the ratio between the largest and the lowest singular values, and the terms $\mu(\cdot)$ and $\eta(\cdot)$ are two specific norm parameters defined respectively in Definitions 5.1 and 6.1 (see Chapter 5 for more details).

We will show that the term $\mu(B)$ in the above expression is in fact upper bounded by $O(N)$, in the worst case scenario. In our basic numerical experiments (see Section 11.3), we indeed observed a quantum running time scaling linearly with N, when all terms are taken into account. This implies a significant polynomial speedup over the classical algorithm.

The second stage, the quantum clustering, adds to the running time the multiplicative term T_{qmeans}, a rather complex expression detailed in Chapter 8. In the case of well-clusterability, namely when the vectors can effectively be organized in

clusters that can be detected efficiently (See Section 8.1.3 for details), this runtime can be rewritten as follows (see also Theorem 8.3), where k is the number of clusters and δ is a precision parameter :

$$T_{qmeans} = \widetilde{O}\left(\frac{k^3\eta(\widetilde{\mathcal{L}}^{(k)})^{2.5}}{\delta^3}\right) \tag{9.3}$$

We expect our quantum spectral clustering algorithm to be accurate and efficient when the classical spectral clustering algorithm also works well. In fact, the classical algorithm works well in the case when, once projected onto the reduced spectral space, the vectors follow the well-clusterability assumption, allowing for efficient clustering. In particular, in this case, the term $\kappa(\widetilde{\mathcal{L}}^{(k)})$ is close to k, the number of clusters. Note however that our algorithm could work without this well-clusterability assumption, but the theoretical runtime would be bounded differently.

In conclusion, our algorithm provides a considerable theoretical speedup that could allow for new applications of spectral clustering on larger, high-dimensional datasets. The quantum subroutines developed in this section could be useful independently, and we hope for substantial improvements in several other graph based machine learning algorithms.

9.2 Quantum Graph Based Machine Learning

In this part, we will detail the quantum algorithm that performs spectral clustering with similar guarantees and more efficient running time compared to its classical analog. We start by presenting all the theorems that will help us construct the quantum spectral clustering algorithm, then in the following subsections we provide details and proofs.

We first state the theorem that allows to compute the edge's value of the data adjacency graph:

Theorem 9.1: Quantum Algorithm for Data Point Similarity

Given quantum access to the data matrix $S \in \mathbb{R}^{N\times d}$ in time T_S and two indices $p, q \in [N]^2$, we can obtain the following mapping: $|p\rangle|q\rangle|0\rangle \mapsto |p\rangle|q\rangle|a_{pq}\rangle$ in time $O(T_S\frac{\eta(S)}{\epsilon_{dist}})$. The elements a_{pq} correspond to the edge's values of the data adjacency graph, using the rule of construction based on a threshold distance. $\eta(S)$ is a data parameter defined in Definition 6.1, $\epsilon_{dist} > 0$ is the precision parameter in the estimation of the distance between input points.

Using the previous theorem we can have quantum access to the normalized Incidence matrix.

Theorem 9.2: Quantum access to the Normalized Incidence Matrix

Given quantum access to the data matrix $S \in \mathbb{R}^{N\times d}$ in time T_S we can have quantum access to the normalized incidence matrix $\mathcal{B} \in \mathbb{R}^{N\times \frac{N(N-1)}{2}}$ in time

$$T_{\mathcal{B}} = \widetilde{O}\left(T_S \frac{\eta(S)}{\epsilon_{dist}} \frac{1}{\epsilon_B}\right) \tag{9.4}$$

where $\eta(S)$ is defined in Definition 6.1, $\epsilon_{dist} > 0$ is the precision of distance estimation between vectors, and ϵ_B is the substitute of the zeros in B.

Using the previous theorem we can finally have quantum access to the projected Laplacian matrix.

Theorem 9.3: Quantum access to projected Laplacian matrix

Given quantum access to the normalized incidence matrix $\mathcal{B} \in R^{N\times \frac{N(N-1)}{2}}$ in time $T_{\mathcal{B}}$, we can have quantum access to $\widetilde{\mathcal{L}}^{(k)} \in R^{N\times k}$, the Laplacian matrix projected onto its k lowest eigenvalues, in time

$$T_{\widetilde{\mathcal{L}}^{(k)}} = \widetilde{O}\left(T_{\mathcal{B}} \frac{\mu(\mathcal{B})\kappa(\widetilde{\mathcal{L}}^{(k)})}{\epsilon_\lambda}\right) \tag{9.5}$$

where ϵ_λ is the precision parameter for estimating the eigenvalues of $\mathcal{L}$, $\mu(\mathcal{B})$ is a data parameter defined in Definition 5.1, and $\kappa(\widetilde{\mathcal{L}}^{(k)})$ is the condition number of $\widetilde{\mathcal{L}}^{(k)}$.

The quantum spectral clustering algorithm consists then in applying the quantum k-means algorithm (Theorem 8.3) using the fact that we have quantum access to the projected Laplacian matrix.

The main algorithm can thus be summarized in the following theorem.

Theorem 9.4: Quantum Spectral Clustering

Given quantum access to a data matrix $S \in \mathbb{R}^{N\times d}$ in time T_S, there is a quantum algorithm that with high probability outputs k centroids in the Laplacian spectral space, in time

$$\widetilde{O}\left(T_S \frac{\eta(S)}{\epsilon_{dist}\epsilon_B}\frac{\mu(\mathcal{B})\kappa(\widetilde{\mathcal{L}}^{(k)})}{\epsilon_\lambda} T_{qmeans}\right) \tag{9.6}$$

where T_{qmeans} is the multiplicative factor in the running time of the quantum clustering algorithm (Theorem 8.3) on the vectors projected onto the Laplacian spectral space. In the case where the vectors projected onto the spectral space are well-clusterable (see Definition 8.1), the running time becomes

$$\widetilde{O}\left(T_S \frac{\eta(S)}{\epsilon_{dist}\epsilon_B}\frac{\mu(\mathcal{B})\kappa(\widetilde{\mathcal{L}}^{(k)})}{\epsilon_\lambda}\frac{k^3\eta(\widetilde{\mathcal{L}}^{(k)})^{2.5}}{\delta^3}\right) \tag{9.7}$$

In the above formulas, $\mathcal{B}$ refers to the normalized incidence matrix of the data, and $\widetilde{\mathcal{L}}^{(k)}$ to the projection of the Laplacian matrix. ϵ_{dist}, ϵ_B, ϵ_λ and δ are error or precision parameters. $\eta(S)$, $\eta(\widetilde{\mathcal{L}}^{(k)})$, and $\mu(\mathcal{B})$ are data parameters defined in Definition 6.1 and Definition 5.1, and $\kappa(\widetilde{\mathcal{L}}^{(k)})$ is the condition number of $\widetilde{\mathcal{L}}^{(k)}$.

9.2.1 Quantum Circuits

Computing the similarity between two nodes

We now detail the algorithm for Theorem 9.1, which allows to compute the elements a_{pq} of the Adjacency Matrix, corresponding to a pair of input vectors s_p and s_q. This will be used as a subroutine in the algorithm that gives quantum access to the normalized incidence matrix.

For any two quantum states corresponding to the indices $|p\rangle$ and $|q\rangle$, we can use Theorem 6.1 to obtain $|p\rangle|q\rangle|\overline{d(s_p,s_q)^2}\rangle$. The square distance obtained is approximated with a precision $\epsilon_{dist} > 0$ such that $|\overline{d(s_p,s_q)^2} - d(s_p,s_q)^2| \leq \epsilon_{dist}$.

These distances are then converted into the edge values $a_{pq} \in \{0,1\}$ using our modified graph construction rule from Section 2.2.2. For doing so, we can use the comparison operator (see Claim 3.1) to check if $\overline{d(s_p,s_q)^2}$ is smaller than the desired threshold below which we consider that two nodes are connected. Therefore we can easily obtain $|p\rangle|q\rangle|a_{pq}\rangle$.

Note that it was necessary to use this particular definition of the Adjacency Matrix in order to be able to perform this operation efficiently quantumly. Other definitions include the direct use of the distance, or sometimes a mapping of the distance using diverse kernels.

Finally, note that using this theorem, it is possible to have quantum access to a normalized vesrion of the Adjacency Matrix, which could be useful in many applications. In our case, however, it is not clear how to use it to obtain quantum access to the normalized Laplacian, because it would require the Degree matrix and

the norms $\|a_i\|$, which are not accessible efficiently. Therefore, we chose to work with the *normalized* Incidence matrix.

Quantum Access to the Normalized Incidence Matrix $\mathcal{B}$

To obtain quantum access to $\mathcal{B}$ (see Definition 4.5), we start with a quantum state encoding one node's index $i \in [N]$, along with the superposition of all $\frac{N(N-1)}{2}$ possible edges encoded as pairs $(p, q) \in [N]^2$ with $p < q$. We use the following quantum state, whose preparation requires $O(\log(N))$ qubits and a simple quantum circuit.

$$\frac{\sqrt{2}}{\sqrt{N(N-1)}} |i\rangle \sum_{p<q} |p\rangle |q\rangle \tag{9.8}$$

From this state, we first determine which edges are incident using two ancillary qubits as flags, using the equality operator (see Section 3.1). This allows us to separate the cases that appear in Eq.(2.5). For simplicity, we will use the notation $|p\rangle |q\rangle = |p, q\rangle$.

$$\frac{\sqrt{2}}{\sqrt{N(N-1)}} |i\rangle \left(\sum_{\substack{p<q \\ i=p}} |p, q\rangle |11\rangle + \sum_{\substack{p<q \\ i=q}} |p, q\rangle |10\rangle + \sum_{\substack{p<q \\ i\notin\{p,q\}}} |p, q\rangle |00\rangle \right) \tag{9.9}$$

We then use another register to write the values of B, the unormalized incidence matrix, given by Eq.(2.5). For the flagged edges ($i = p$ or $i = q$), using a controlled version of Theorem 9.1, we obtain the superposition of the similarity between all input points and point i, in a quantum register. For the other edges ($i \notin \{p, q\}$), instead of simply writing 0 as in Eq.(2.5) we modify the zero elements of the matrix to a value $\epsilon_B > 0$ in order to retain the efficiency of the quantum algorithm in the next step. The running time for this step is $O(T_S \eta(S)/\epsilon_{dist})$, where $\eta(S)$ is a data parameter defined in Definition 6.1, and $\epsilon_{dist} > 0$ is the precision parameter in the estimation of $d^2(s_i, s_j)$. Finally T_S is the time to have quantum access to the input points $s_1, \cdots, s_N$, which becomes $T_S = O(\log(Nd))$ if we assume QRAM access.

We obtain the following state:

$$\frac{\sqrt{2}}{\sqrt{N(N-1)}} |i\rangle \left(\sum_{\substack{p<q \\ i=p}} |p, q\rangle |11\rangle |a_{pq}\rangle + \sum_{\substack{p<q \\ i=q}} |p, q\rangle |10\rangle |a_{pq}\rangle + \sum_{\substack{p<q \\ i\notin\{p,q\}}} |p, q\rangle |00\rangle |\epsilon_B\rangle \right) \tag{9.10}$$

Which, by Eq.(2.5) and after uncomputing the flags would be equal to

$$\frac{\sqrt{2}}{\sqrt{N(N-1)}} |i\rangle \sum_{p<q} |p, q\rangle |B_{i,(p,q)}\rangle \tag{9.11}$$

From this state, we use a Conditional Rotation (see Theorem 3.5) to encode, in superposition, the values of the B into the amplitude of a new qubit, and after uncomputing the values of the matrix B from the registers, we have the state:

$$\frac{\sqrt{2}}{\sqrt{N(N-1)}} |i\rangle \left(\sum_{p<q} B_{i,(p,q)} |p,q\rangle |0\rangle + \sum_{p<q} \sqrt{1 - B^2_{i,(p,q)}} |p,q\rangle |1\rangle \right) \tag{9.12}$$

Finally, an amplitude amplification (see Theorem 3.2) is performed to select the $|0\rangle$ part of the state, and obtain a valid quantum encoding of the vector B_i. This requires to repeat the previous steps $O(1/\sqrt{P_i(0)})$ times where $P_i(0)$ is the probability of reading "0" in the last register. Since all elements of the matrix B have norm at least ϵ_B we therefore have $O(1/\sqrt{P_i(0)})$ is at least $O(1/\epsilon_B)$. We finally obtain with high probability the state $|i\rangle \frac{1}{\|B_i\|} \sum_{p<q} B_{i,(p,q)} |p,q\rangle = |i\rangle |\mathcal{B}_i\rangle$.

Recall from Section 2.2.2 that $\mathcal{B}_{i,(p,q)} = \frac{B_{i,(p,q)}}{\|B_i\|}$ and that $\|B_i\| = \|a_i\|$. In addition, we have access to the norm of each row of $\mathcal{B}$ since by definition they are all equal to 1. We can therefore conclude that we have quantum access to $\mathcal{B}$, the normalized incidence matrix, according to Definition (4.5). The global running time to have quantum access to $\mathcal{B}$ is given by $O(T_S \frac{\eta(S)}{\epsilon_{dist}} \frac{1}{\epsilon_B}))$. This proves Theorem 9.2.

Quantum Access to the Projected Normalized Laplacian Matrix $\widetilde{\mathcal{L}}^{(k)}$

With the quantum access to the normalized incidence matrix $\mathcal{B}$ in time $T_{\mathcal{B}}$, we will use the fact that the i^{th} row of $\mathcal{L} = \mathcal{B}\mathcal{B}^T$ can be written as $\mathcal{L}_i = \mathcal{L} \cdot e_i$ where e_i represents the i^{th} vector of the standard basis, for which the corresponding quantum state is simply $|i\rangle$. We also use the fact that this state can be naturally expressed as $|i\rangle = \sum_j \sigma_{ij} |u_j\rangle$ in the basis made of the left singular vectors u_j of $\mathcal{B}$, with unknown coefficients σ_{ij}, such that $\sum_j \sigma^2_{ij} = 1$.

On this initial state $|i\rangle$ we apply the SVE algorithm[1] (Theorem 5.1) to estimate the singular values of $\mathcal{B}$ in superposition and obtain $\sum_j \sigma_{ij} |u_j\rangle |\lambda^{\mathcal{B}}_j\rangle$. The running time for this step is $O(T_{\mathcal{B}}\mu(\mathcal{B})/\epsilon_\lambda)$, where $\mu(\mathcal{B})$ is a data parameter defined in Definition 5.1, and $\epsilon_\lambda > 0$ is the desired precision in the estimation of the singular values. We then square these values to obtain the state $\sum_j \sigma_{ij} |u_j\rangle |\lambda_j\rangle$, with the eigenvalues of $\mathcal{L}$.

Note that $\mu(\mathcal{B})$ is upper bounded by N. Indeed, from Definition 5.1 we have $\mu(\mathcal{B}) \leq \|\mathcal{B}\|_F$. Recall that $\mathcal{B} \in \mathbb{R}^{N \times N(N-1)/2}$. Since by construction each $\|B_i\|_2 = 1$, and since $N(N-1)/2 < N^2$, we have finally:

$$\mu(\mathcal{B}) \leq \|\mathcal{B}\|_F = \sqrt{\sum_i^{\frac{N(N-1)}{2}} \|B_i\|_2^2} = \sqrt{\frac{N(N-1)}{2}} \leq N \tag{9.13}$$

At this point, we can prepare the projection on the k lowest eigenvectors of $\mathcal{L}$, as in Section 5.1. We first separate the eigenvalues lower that a threshold $\nu > 0$ with an ancillary qubit, such that the k lowest eigenvalues are flagged by "0" :

$$\sum_{j|\lambda_j \leq \nu} \sigma_{ij} |u_j\rangle |\lambda_j\rangle |0\rangle + \sum_{j|\lambda_j > \nu} \sigma_{ij} |u_j\rangle |\lambda_j\rangle |1\rangle \tag{9.14}$$

[1]Since SVE must be applied to a square matrix, we use in fact $\begin{pmatrix} 0 & \mathcal{B} \\ \mathcal{B}^T & 0 \end{pmatrix}$ see [KP16].

If the flag qubit is "0", we perform a conditional rotation on the eigenvalue register using Theorem 3.5 in a controlled fashion. For this we introduce again an ancillary qubit. Since the amplitude of the rotation must be inferior to 1, we first divide each eigenvalue by ν, which is higher than the largest eigenvalue of $\widetilde{\mathcal{L}}^{(k)}$:

$$\sum_{j|\lambda_j\leq\nu} \sigma_{ij} |u_j\rangle |\lambda_j\rangle |0\rangle \left(\frac{\lambda_j}{\nu}|0\rangle + \sqrt{1-\frac{\lambda_j^2}{\nu^2}}|1\rangle\right) + \sum_{j|\lambda_j>\nu} \sigma_{ij} |u_j\rangle |\lambda_j\rangle |1\rangle |0\rangle \tag{9.15}$$

From this quantum state, we will show how to obtain quantum access to $|\widetilde{\mathcal{L}}^{(k)}\rangle$ using Amplitude Estimation and amplitude amplification on the "00" value on the last two registers. We recall that quantum access is guaranteed if we can recover, for each row $\widetilde{\mathcal{L}}_i^{(k)}$, its norm and the corresponding quantum state.
Let $P_i(00)$ be the probability of reading "00" on the last two registers. It is easy to show that $P_i(00) = \frac{1}{\nu^2}\sum_{j|\lambda_j\leq\nu}\sigma_{ij}^2\lambda_j^2$. We will prove that $\left\|\widetilde{\mathcal{L}}_i^{(k)}\right\| = \nu\sqrt{P_i(00)}$, and that amplifying the state will yield to $|\widetilde{\mathcal{L}}_i^{(k)}\rangle$.
The normalized Laplacian matrix can be written in its eigenbasis using the outer product $\mathcal{L} = \sum_j \lambda_j |u_j\rangle\langle u_j|$. Similarly, recall that we used $\sum_j \sigma_{ij}|u_j\rangle$ to encode e_i, the i^{th} vector in the standard basis. Therefore, each row $\mathcal{L}_i = \mathcal{L}\cdot e_i$ can be written as $|\mathcal{L}_i\rangle = \sum_j \lambda_j |u_j\rangle\langle u_j|\cdot\sum_j \sigma_{ij}|u_j\rangle = \sum_j \sigma_{ij}\lambda_j|u_j\rangle$. We thus have a formula for the norm of the rows $\|\mathcal{L}_i\| = \sqrt{\sum_j \sigma_{ij}^2\lambda_j^2}$. The same idea holds for $\widetilde{\mathcal{L}}^{(k)}$, the projection of the normalized Laplacian, and we obtain $\left\|\widetilde{\mathcal{L}}_i^{(k)}\right\| = \sqrt{\sum_{j=1}^k \sigma_{ij}^2\lambda_j^2} = \nu\sqrt{P_i(00)}$.
Therefore, using Amplitude Estimation (Theorem 3.2), we can estimate $P(00)$ and have access to the norms of $\widetilde{\mathcal{L}}^{(k)}$ to a multiplicative constant. Similarly, using amplitude amplification (Theorem 3.2 also), we can amplify the $|00\rangle$ state and obtain the state $\sum_{\lambda_j\leq\nu}\sigma_{ij}\lambda_j|u_j\rangle = |\widetilde{\mathcal{L}}_i^{(k)}\rangle$.
The number of iterations for Amplitude Estimation and Amplification is $\widetilde{O}(1/\sqrt{P_i(00)})$. Let λ_k be the k^{th} eigenvalue of $\mathcal{L}$ and therefore the largest eigenvalue of $\widetilde{\mathcal{L}}^{(k)}$. By correctly choosing the threshold ν close to the largest eigenvalue, for instance $\nu\leq\gamma\lambda_k$ with $\gamma>1$ a small positive constance, we can write:

$$P_i(00) = \sum_{j|\lambda_j\leq\nu}\sigma_{ij}^2\frac{\lambda_j^2}{\nu^2} \geq \sum_{j|\lambda_j\leq\lambda_k}\sigma_{ij}^2\frac{\lambda_{min}^2}{\gamma^2\lambda_k^2} = \frac{1}{\gamma^2}\frac{\lambda_{min}^2}{\lambda_k^2} = \frac{1}{\gamma^2}\frac{1}{\kappa(\widetilde{\mathcal{L}}^{(k)})^2} = O\left(1/\kappa(\widetilde{\mathcal{L}}^{(k)})^2\right) \tag{9.16}$$

Therefore the number of iterations for this step can be upper bounded by $\widetilde{O}(\kappa(\widetilde{\mathcal{L}}^{(k)}))$. Overall, the running time is given by $O(T_{\mathcal{B}}\frac{\mu(\mathcal{B})}{\epsilon_\lambda}\kappa(\widetilde{\mathcal{L}}^{(k)}))$. This concludes the algorithm that gives quantum access to the projected Laplacian of a graph and proves Theorem 9.3.

9.2.2 Quantum Clustering in the Spectral Space

Having quantum access to the projected normalized Laplacian $\widetilde{\mathcal{L}}^{(k)}$, we possess all requirements to apply the quantum k-means algorithm, or q-means (Theorem 8.3). We initialize k centroids at random or using q-means++ (see Section 8.2.3), equivalent to the k-means++ initialization. The q-means algorithm first consists in constructing a state where all distances between rows $\widetilde{\mathcal{L}}_i^{(k)}$ and the current centroids are

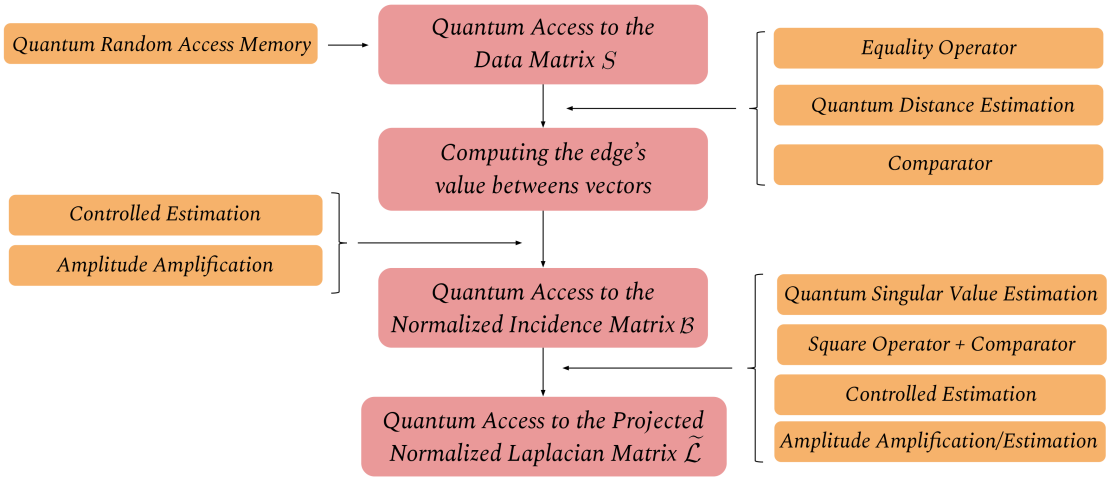


Figure 9.1: Diagram for Sections 9.2.1 to 9.2.1

computed in parallel. The rest of the algorithm consists in finding the label for each row, and updating the centroids as the average of the vectors composing the cluster. After several iterations, the centroids should have converged and can be retrieved.

Several approximations are necessary during the steps, hence the presence of precision parameter $\delta > 0$. It expresses the approximation error committed during the distance estimation, and during the classical description of the new centroids at each step. Therefore, to be more specific, q-means is a noisy version of k-means.

In high level, denoting with $T_{\widetilde{\mathcal{L}}^{(k)}}$ the time to have quantum access to $\widetilde{\mathcal{L}}^{(k)} \in \mathbb{R}^{N\times k}$, which plays the role of the initial state for the clustering algorithm, and T_{qmeans} the remaining multiplicative factor in the running time of the quantum clustering algorithm (see Theorem 8.3), the overall running time of our algorithm is given by

$$T_{\widetilde{\mathcal{L}}^{(k)}} T_{qmeans} \tag{9.17}$$

To go further, we can assume that the vectors are effectively made of well separated clusters. Indeed, it should be the case in the spectral clustering method, once projected onto the spectral space (see Section 11.3 for numerical simulations). This well-clusterability assumption (See Definition 8.1) ensures the classical spectral clustering to classify the data accurately, and the quantum algorithm to work efficiently. Indeed, the running time of the q-means algorithm is bounded with better guarantees in this case (Chapter 8). With this assumption, and with input dimension k for the spectral space of $\widetilde{\mathcal{L}}^{(k)}$, the running time to update the k centroids in the case of well-clusterable data is given by

$$\widetilde{O}\left(T_{\widetilde{\mathcal{L}}^{(k)}} \frac{k^3 \eta (\widetilde{\mathcal{L}}^{(k)})^{2.5}}{\delta^3}\right) \tag{9.18}$$

Again, it is important to note that our algorithm could work without this well-clusterability assumption, the only difference would be a different bound on the theoretical runtime (See Theorem 8.3 for the general formulation).

9.2.3 Running Time

Using the running times obtained in Theorems 9.2 and 9.3, as well as result (9.18) for the well-clusterable case, we can conclude that our quantum algorithm for spectral clustering has the following running time :

$$\widetilde{O}\left(T_S \frac{\eta(S)}{\epsilon_{dist}\epsilon_B} \frac{\mu(\mathcal{B})\kappa(\widetilde{\mathcal{L}}^{(k)})}{\epsilon_\lambda} \frac{k^3\eta(\widetilde{\mathcal{L}}^{(k)})^{2.5}}{\delta^3}\right) \tag{9.19}$$

T_S is the time to have *quantum access* (see Definition 4.5) to the input vectors $S \in \mathbb{R}^{N\times d}$ which becomes $O(polylog(N,d))$ if we assume access to the QRAM. k is the number of clusters. $\kappa(\widetilde{\mathcal{L}}^{(k)})$ is the condition number of $\widetilde{\mathcal{L}}^{(k)}$. $\mu(\mathcal{B})$, $\eta(S)$, and $\eta(\widetilde{\mathcal{L}}^{(k)})$ are data parameters defined in Definitions 5.1 and 6.1. ϵ_B is the chosen minimum value in the incidence matrix. ϵ_{dist} is the precision in the distance calculation between input points. δ is the precision of the q-means algorithm.

9.3 Numerical Simulations

The quantum algorithm performs the same steps as the classical one while introducing noise or randomness along the way. We present numerical simulations on simple synthetic datasets made of two concentric circles, as in the original work on spectral clustering [NJW02], in order to benchmark the quality of the quantum algorithm. In this way, we see how the quantum effects are impacting the graph and the spectral space, and we obtain numerical estimates on the quantum running time.

These simulations are made with a classical computer that simulates the quantum steps and introduces equivalent noise and randomness. It would be very interesting to perform the same experiments using a real quantum computer, alas such computers are not yet available. While simulating the quantum steps, the computation becomes very soon impractical, in fact the simulations we present already take several hours to execute, and thus we leave numerical simulations on larger datasets as future work. Our goal was to design a quantum spectral clustering algorithm with a rigorous theoretical analysis of its running time and provide initial evidence of its practical efficiency and accuracy on a canonical dataset.

The precision parameters used in the quantum case were: $\epsilon_{dist} = 0.1$ for the creation of edges, $\epsilon_B = 0.1$ for the creation of the incidence matrix, $\epsilon_\lambda = 0.9$ during the estimation of the eigenvalues of $\mathcal{L}$ and finally the precision parameter in q-means $\delta = 0.9$. The quantum algorithm was able to classify the two sets with high accuracy (Table 9.2). The clustering was simulated for 300 to 1000 points, repeated 10 times each. In Fig.9.2 we observe clearly the impact of the quantum effects in the graph, where the edges are different, and in the spectral space, where the clusters are more spread out. It is surprising that the quantum algorithm is already faster for small values of N, below 1000 points (Fig.9.3). This difference should substantially increase as N grows. Indeed, compared to the classical algorithm used in practice which scales as $O(N^3)$, the quantum running time is advantageous as its scaling appears to be linear in N. In fact, the scaling comes from the factor $\mu(B)$ which is upper bounded by N (see Section 9.2.1). Note, of course, that both for the classical and quantum running time we used as a proxy the order of steps in the theoretical analysis, disregarding questions of clock time or error correction. Our results show

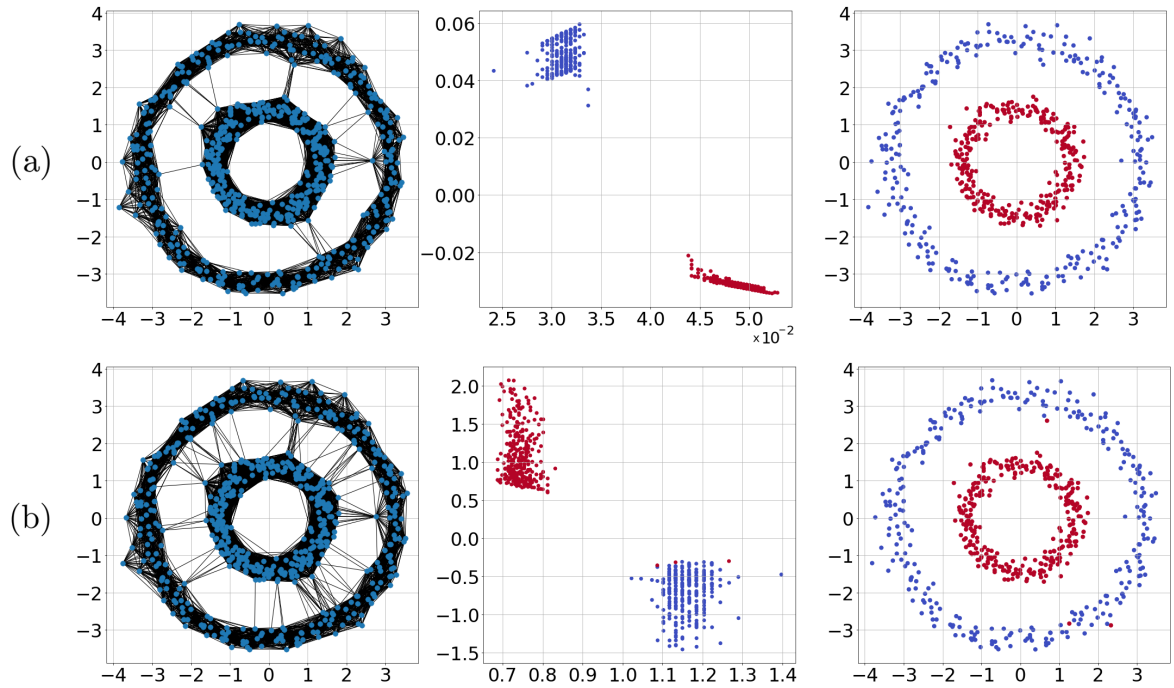


Figure 9.2: (a) Classical and (b) Quantum spectral clustering algorithms on two non linearly separable sets of 600 input vectors. The Laplacian is derived from the adjacency graph *(left)*, itself constructed from the data points. The clustering is shown in both spectral *(center)* and input *(right)* space domains. Three points were misclassified in the quantum case.

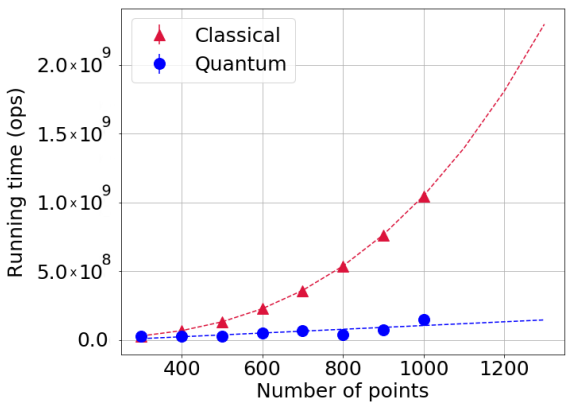


Figure 9.3: Running times for quantum and classical spectral clustering. Error bars are present.

Numbers of points	Accuracy	
	Classical	Quantum
300	100%	$99.6\% \pm 1.05\%$
400	100%	$99.5\% \pm 0.83\%$
500	100%	$99.3\% \pm 1.52\%$
600	100%	$99.9\% \pm 0.31\%$
700	100%	$100.0\% \pm 0.0\%$
800	100%	$99.6\% \pm 0.62\%$
900	100%	$99.6\% \pm 0.68\%$
1000	100%	$99.9\% \pm 0.19\%$

Table 9.2: Our quantum algorithm finds the clusters with a similar high accuracy.

more than anything that it is certainly worth pursuing quantum algorithms for spectral clustering and other graph based machine learning algorithms, since at least at a first level of comparison they can offer considerable advantages compared to the classical algorithms. It remains an open question to see when and if quantum hardware can become good enough to offer such advantages in practice.

Conclusions

In this work, we described a new quantum machine learning algorithm, inspired by the classical spectral clustering algorithm. While introducing a number of modifications, approximations, and randomness in the process, the quantum algorithm can still perform clustering tasks with similar very good accuracy, and with a more efficient running time thanks to a weaker dependence on the number of input points: at most linear in our preliminary experiments. This could allow quantum spectral clustering to be applied on datasets that are now considered infeasible in practice. Our quantum algorithm is end-to-end, from classical input to classical output, and could pave the way to other graph based methods in machine learning and optimization, for example using our methods for obtaining access to the normalized Adjacency, Incident, and Laplacian matrices.

Part IV

Quantum Neural Networks

Chapter 10

Introduction

"Les machines un jour pourront résoudre tous les problèmes, mais jamais aucune d'entre elles ne pourra en poser un !"

Albert Einstein
Comment je vois le monde
(1934)

We refer the reader to Section 2.3 for a basic introduction to classical neural network theory and notations.

10.1 Why is it Hard to Implement a Neural Network on a Quantum Computer?

In recent years, many attempts have been made to provide quantum algorithms for neural networks. this proliferation is due to the significance that it would represent but also to the fact that it is a problem difficult to solve [SSP14].

The main issues in creating quantum neural networks are non-linearities, modularity and diversity:

Non-linearities As we have seen in Section 2.3.1, non-linear functions are systematically present at the end of each neural network layer. They allow the model to reach universality by approximating any functions [Cyb89, LLPS93]. However, quantum computing is the realm of unitary, therefore linear, transformations. In particular, when the amplitudes of a quantum state encode the meaningful data, one cannot apply an arbitrary non-linear function on it. This represents a strong limit on the ability of quantum computers to enhance deep learning by directly converting them as quantum circuits and harnessing quantum superposition across layers.

To deal with this paradox, several attempts have been made [TMGB19, BBF$^+$20, WKS14b, CGAG17]. Most use measurements to trigger non-linear behaviors, others transfer the data in binary encoding for classical applications of non-linearities [AHKZ20]. Finally, it is often the case that quantum neural networks are proposed without any non-linearity [RBWL18]. In this thesis, we will pursue the approach of

[AHKZ20], which comes at the cost of performing amplitude estimation and thus limits the speedup to a quadratic one at the end.

Modular approach The success of deep learning relies on the infinite number of architectures that are possible to produce in order to solve one particular task. It allows us to adapt to any input or output size, or to test different numbers and sizes of hidden layers (often called *hyper parameter optimization*). Recent architectures such as ResNet [TYL17] or LSTM [GSC00] even require connections between input layer non adjacent hidden layers. In deep learning for recommendation systems [ZYST19] several inputs are processed on their own neural network, then branched later on.

The way we define quantum neural networks should include this modularity and offer similar handling. This would require a certain architecture for the quantum circuit. First, the quantum circuit should correspond to one layer, and potentially be end-to-end (from classical input to classical output).

Diversity of modules Even though quantum deep learning should start at the basic, namely fully connected neural networks (see Section 2.3.1), there is a large variety of layers that exist in the literature. Different layer types are combined in a single network, including convolutional layer, but also pooling (see Section 2.3.4), averaging, dropout. A zoo of non-linearities are also used at different steps (ReLu, Sigmoid, tanh, etc.). More recently, additional constraints have been introduced on the layer such as orthogonality, isometry, hyperbolic geometry [GBH18] and more.

Quantum neural networks don't have to mimic them all at first, but we should bear in mind that this is the level of diversity we could aim for. Quantum computing may allow for certain properties easily (see Chapter 12).

Training Finally, we must not overlook the difficulty of training the weights of the neural networks, which is most commonly done using backpropagation (see Section 2.3.2). This also represents a challenge for quantum computing since it imposes to access the state of each layer. It is then a strong argument for modularity and the classical inputs and outputs. Having only a quantum state for each layer and pursuing the computation would present to do the backpropagation unless we destroy the quantum state by measuring it. for instance, faced with this problem, variational quantum circuits have chosen to abandon backpropagation for less efficient techniques (see Section 3.3).

Before going into the details of quantum neural networks, it is useful to remind that there exist different approaches for short term or NISQ applications. As detailed in Section 3.3, variational quantum circuits don't properly have weights but gate parameters that are tunable as well. As in [CCL19, VBB17], the appellation *quantum neural networks* is used for their conceptual similarities. In Chapter 12, we will propose a similar NISQ neural network that has a specific and fast training paradigm, and that is equivalent to its classical counterpart. There also exist recent proposals for fault tolerant quantum computers using the *Neural Tangent Kernel* approach and the HHL algorithm [ZNL21].

10.2 Quantum Algorithm for Fully Connected Neural Networks

In [AHKZ20], the authors introduced a quantum algorithm for fully connected neural network defined in Section 2.3.1, based on the quantum inner product estimation (QIPE) from [KLLP19] (Theorem 6.1).

As stated in Eq.(2.6) and Eq.(2.7), the ℓ^{th} layer of a fully connected neural network's with input vector a^ℓ, weight matrix W^ℓ and bias vector b^ℓ can be written as

$$z^{\ell+1} = W^\ell a^\ell + b^\ell \tag{10.1}$$

$$a^{\ell+1} = \sigma(z^{\ell+1}) \tag{10.2}$$

We see that the matrix product between W^ℓ and a^ℓ is the core of this operation, followed by a non-linear function σ. This matrix multiplication can be replaced by a series of inner product between the input a^ℓ and each row of the weight matrix W_j^ℓ for $j \in [n_\ell]$, where n_ℓ is the number of input nodes or the dimension of a^ℓ.

$$z_j^{\ell+1} = (W_j^\ell, a^\ell) + b_j^\ell \tag{10.3}$$

where (x, y) is the inner product between vectors x and y.

Using the QIPE algorithm from Theorem 6.1, with relative error parameters ϵ and probability parameter Δ, the authors defined a (ϵ, Δ)-feedforward neural network. They achieve a running time of:

$$\widetilde{O}\left(N \frac{\log(1/\Delta)}{\epsilon} R\right) \tag{10.4}$$

where N is the number of neurons in the whole neural network, and R is an aggregate of data dependant parameters (based on the norms of the weights and inputs).

To this end, they apply the QIPE in superposition over all the rows of the weights matrix, assuming quantum access to W^ℓ and a^ℓ.

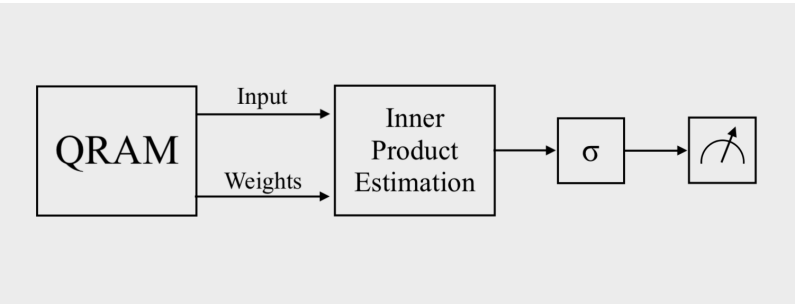


Figure 10.1: Diagram representation of a fully connected neural network's layer using the quantum inner product estimation from Theorem 6.1. It performs the matrix multiplication and the following non-linearity σ.

Further, a quantum backpropagation algorithm was introduced. Indeed, as explained in Section 2.3.2, backpropagation is also a series of matrix multiplication

performed layer by layer backward. Using again the QIPE, the authors proposed a quantum algorithm for training the networks in time

$$\widetilde{O}\left((TM)^{1.5}N\frac{\log(1/\Delta)}{\epsilon}R'\right) \tag{10.5}$$

where T is the number of update iterations, M the size of the mini batches, and R' another aggregate of data dependant parameters. Classical simulations were conducted and showed that the numerical parameters R and R' were small in practice.

In addition, the difficulties presented in Section 10.1 forced the authors to be subtle with the weight matrix. They first imagined a low rank initialization, to avoid having too many values stored in the QRAM. Then, at each iteration, instead of updating the whole matrix, which would have ruined the benefit of previous quantum computations, they store instead the layers and a derivative of the error vectors, which allow for a fast recomputation of the weight matrix.

The authors suggest that the noise and errors caused by the quantum nature of this algorithm could be beneficial for neural networks. They argue that training a neural network while performing noisy computations will help it generalize correctly, preventing the weights to adjust too well to the training set also known as *overfitting*. This *robust* training using quantum effects is interesting, and could be further developed in terms of privacy preserving, data enhancement, and more.

Note finally that it is also possible to perform fully connected neural networks in a different setting, using inner products from unary data loaders [MLL$^+$21] (see Section 6.2).

Chapter 11

Quantum Convolutional Neural Networks

"L'ignorance rend plus sûr de soi que la connaissance."

Etienne Klein
Le goût du vrai (2020)

Preliminaries on convolutional neural networks, along with all notations are given in Section 2.3.4. Variables are summarized in Section 11.2.3.

11.1 Main Results

We design a quantum algorithm for a complete CNN, with a modular architecture that allows any number of layers, any number and size of kernels, and that includes a large variety of non-linearity and pooling methods. We introduce a quantum convolution product and a specific quantum sampling technique well suited for information recovery in the context of CNN. We also propose a quantum algorithm for backpropagation that allows an efficient training of our quantum CNN.

As explained in Section 2.3.4, a single layer ℓ of the classical CNN does the following operations: from an input image X^ℓ seen as a 3D tensor, and a kernel K^ℓ seen as a 4D tensor, it performs a convolution $X^{\ell+1} = X^\ell * K^\ell$, followed by a non-linear function, and followed by pooling. In the quantum case, we will obtain a quantum state corresponding to this output, approximated with error $\epsilon > 0$. To retrieve a classical description from this quantum state, we will apply an ℓ_∞ tomography (see Theorem 4.3) and sample the elements with high values, in order to reduce the computation while still getting the information that matters (see Section 11.2.1).

The QCNN can be directly compared to the classical CNN as it has the same inputs and outputs. We show that it offers a speedup compared to the classical CNN for both the forward pass and for training using backpropagation in certain cases. For each layer, on the forward pass (Algorithm 3), the speedup is exponential in the size of the layer (number of kernels) and almost quadratic on the spatial dimension of the input. We next state informally the speedup for the forward pass, the formal version appears as Theorem 11.1.

Result 11.1: Quantum Convolution Layer

Let X^ℓ be the input and K^ℓ be the kernel for layer ℓ of a convolutional neural network, and $f : \mathbb{R} \mapsto [0, C]$ with $C > 0$ be a non-linear function so that $f(X^{\ell+1}) := f(X^\ell * K^\ell)$ is the output for layer ℓ.
Given X^ℓ and K^ℓ stored in QRAM, there is a quantum algorithm that, for any precision parameters $\epsilon > 0$ and $\nu > 0$, creates quantum state $|f(\overline{X}^{\ell+1})\rangle$ such that $\left\| f(\overline{X}^{\ell+1}) - f(X^{\ell+1}) \right\|_\infty \leq 2M\epsilon$ and retrieves classical tensor $\mathcal{X}^{\ell+1}$ such that for each pixel j,

$$\begin{cases} |\mathcal{X}_j^{\ell+1} - f(X_j^{\ell+1})| \leq 2\epsilon & \text{if} \quad f(\overline{X}_j^{\ell+1}) \geq \nu \\ \mathcal{X}_j^{\ell+1} = 0 & \text{if} \quad f(\overline{X}_j^{\ell+1}) < \nu \end{cases} \tag{11.1}$$

This algorithm runs in time

$$\widetilde{O}\left(\frac{1}{\epsilon\nu^2} \frac{M\sqrt{C}}{\sqrt{\mathbb{E}(f(\overline{X}^{\ell+1}))}} \right) \tag{11.2}$$

where $\mathbb{E}(f(\overline{X}^{\ell+1}))$ represents the average value of $f(\overline{X}^{\ell+1})$, and $\widetilde{O}$ hides factors polylogarithmic in the size of X^ℓ and K^ℓ and the parameter M (defined in Eq.(11.22)) is the maximum product of norms from subregions of X^ℓ and K^ℓ.

We see that the number of kernels contributes only polylogarithmically to the running time, allowing the QCNN to work with larger and in particular exponentially deeper kernels. The contribution from the input size is hidden in the precision parameter ν such that (see also Eq.(11.34)):

$$\nu = \frac{1}{\sqrt{\sigma \cdot H^{\ell+1} W^{\ell+1} D^{\ell+1}}} \tag{11.3}$$

Therefore the running time has a factor $O(\sigma \cdot H^{\ell+1} W^{\ell+1} D^{\ell+1})$, which is the number of pixels classically retrieved from the quantum state. Indeed, a sufficiently large fraction of pixels must be sampled from the output of the quantum convolution to retrieve meaningful information. In the Numerical Simulations (Section 11.3) we give estimates for ν. The cost of generating the output $\mathcal{X}^{\ell+1}$ of the quantum convolutional layer can thus be much smaller than that for the classical CNN in certain cases, Section 11.2.3 provides a detailed comparison to the classical running time.

Following the forward pass, a loss function $\mathcal{L}$ is computed for the output of a classical CNN. The backpropagation algorithm is then used to calculate, layer by layer, the gradient of this loss with respect to the elements of the kernels K^ℓ, in order to update them through gradient descent. The formal version of our quantum backpropagation algorithm is given as Theorem 11.2

Result 11.2: Quantum Backpropagation for Quantum CNN

Given the forward pass quantum algorithm in Result 11.1, and given the kernel matrix F^ℓ, the input matrices A^ℓ and Y^ℓ, stored in the QRAM for each layer ℓ, and a loss function $\mathcal{L}$, there is a quantum backpropagation algorithm that estimates each element of the gradient tensor $\frac{\partial \mathcal{L}}{\partial F^\ell}$ within additive error $\delta \left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|$, and updates them to perform a gradient descent. The running time of a single layer ℓ for quantum backpropagation is given by

$$O\left(\left(\left(\mu(A^\ell)+\mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}})\right)\kappa(\frac{\partial \mathcal{L}}{\partial F^\ell})+\left(\mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}})+\mu(F^\ell)\right)\kappa(\frac{\partial \mathcal{L}}{\partial Y^\ell})\right)\frac{\log(1/\delta)}{\delta^2}\right) \tag{11.4}$$

where for a matrix $V \in \mathbb{R}^{n\times n}$, $\kappa(V)$ is the condition number and $\mu(V) \leq \sqrt{n}$ is a matrix dependent parameter defined in Definition 5.1.

Details concerning the tensors and their matrix expansion or reshaping are given in Section 2.3.4, and a summary of all variables with their meaning and dimension is given in Section 11.2.3. Note that X^ℓ, Y^ℓ and A^ℓ are different forms of the same input. Similarly K^ℓ and F^ℓ both refer to the kernels.

For the quantum backpropagation algorithm, we introduce a quantum tomography algorithm with ℓ_∞ norm guarantees, that could be of independent interest. It is exponentially faster than the tomography with ℓ_2 norm guarantees and is given as Theorem 4.2. Numerical simulations on classifying the MNIST dataset show that our quantum CNN achieves a similar classification accuracy as the classical CNN.

The rest of the chapter is organized as follows: we explain our quantum algorithm in two parts: the forward quantum convolution layer (Section 11.2.1) and the quantum backpropagation (Section 11.2.4). The final part presents the results of our numerical simulations (Section 11.3) and our conclusions (Section 11.4). A summary of the variables is given as Section 11.2.3, and the two algorithms for the forward and backpropagation phase of the QCNN are given as Algorithm 3 and Algorithm 4.

11.2 Quantum Algorithm

11.2.1 Quantum Feedforward Algorithm

In this section we will design quantum procedures for the usual operations in a CNN layer. We start by describing the main ideas before providing the details. The forward pass algorithm for the QCNN is given as Algorithm 3.

First, to perform a convolution product between an input and a kernel, we use the mapping between convolution of tensors and matrix multiplication from Section 2.3.4, which can further be reduced to inner product estimation between vectors. The output will be a quantum state representing the result of the convolution product, from which we can sample to retrieve classical information to feed the next layer. This is stated in the following Theorem:

Theorem 11.1: Quantum Convolution Layer

Given 3D tensor input $X^\ell \in \mathbb{R}^{H^\ell \times W^\ell \times D^\ell}$ and 4D tensor kernel $K^\ell \in \mathbb{R}^{H \times W \times D^\ell \times D^{\ell+1}}$ stored in QRAM, and precision parameters $\epsilon, \Delta > 0$, there is a quantum algorithm that computes a quantum states Δ-close to $|f(\overline{X}^{\ell+1})\rangle$ where $X^{\ell+1} = X^\ell * K^\ell$ and $f : \mathbb{R} \mapsto [0, C]$ is a non-linear function. A classical approximation such that

$$\left\| f(\overline{X}^{\ell+1}) - f(X^{\ell+1}) \right\|_\infty \leq \epsilon \tag{11.5}$$

The time complexity of this procedure is given by $\widetilde{O}\left(M/\epsilon\right)$, where M is the maximum norm of a product between one of the $D^{\ell+1}$ kernels, and one of the regions of X^ℓ of size HWD^ℓ and $\widetilde{O}$ hides factors polylogarithmic in Δ and in the size of X^ℓ and K^ℓ.

Recall that a convolution product can be seen as a pattern detection on the input image, where the pattern is the kernel. The output values correspond to "how much" the pattern was present in the corresponding region of the input. Low value pixels in the output indicate the absence of the pattern in the input at the corresponding regions. Therefore, by sampling according to these output values, where the high value pixels are sampled with more probability, we could retrieve less but only meaningful information for the neural network to learn. It is a singular use case where amplitudes of a quantum state are proportional to the importance of the information they carry, giving a new utility to the probabilistic nature of quantum sampling. Numerical simulations in Section 11.3 provide an empirical estimate of the sampling rate to achieve good classification accuracy.

11.2.2 Single Quantum Convolution Layer

In order to develop a quantum algorithm to perform the convolution as described above, we will make use of quantum linear algebra procedures. We will use quantum states proportional to the rows of A^ℓ, denoted $|A_p\rangle$, and the columns of F^ℓ, denoted $|F_q\rangle$ (we omit the ℓ exponent in the quantum states to simplify the notation). These states are given by

$$|A_p\rangle = \frac{1}{\|A_p\|} \sum_{r=0}^{HWD^\ell - 1} A_{pr} |r\rangle \tag{11.9}$$

$$|F_q\rangle = \frac{1}{\|F_q\|} \sum_{s=0}^{D^{\ell+1}-1} F_{sq} |s\rangle \tag{11.10}$$

We suppose we can load these vectors in quantum states by performing the following queries:

$$\begin{cases} |p\rangle |0\rangle \mapsto |p\rangle |A_p\rangle \\ |q\rangle |0\rangle \mapsto |q\rangle |F_q\rangle \end{cases} \tag{11.11}$$

Such queries, in time polylogarithmic in the dimension of the vector, can be implemented with a Quantum Random Access Memory (QRAM). See Section 11.2.3 for more details on the QRAM update rules and its integration layer by layer.

Algorithm 3 QCNN Layer

Require: Data input matrix A^ℓ and kernel matrix F^ℓ stored in QRAM. Precision parameters ϵ and ν, a non-linearity function $f : \mathbb{R} \mapsto [0, C]$.

Ensure: Outputs the data matrix $A^{\ell+1}$ for the next layer, result of the convolution between the input and the kernel, followed by a non-linearity and pooling.

1: **Step 1: Quantum Convolution**

1.1: Inner product estimation

Perform the following mapping, using QRAM queries on rows A_p^ℓ and columns F_q^ℓ, along with Theorems 3.2 and 6.2 to obtain

$$\frac{1}{K}\sum_{p,q}|p\rangle|q\rangle \mapsto \frac{1}{K}\sum_{p,q}|p\rangle|q\rangle|\overline{P}_{pq}\rangle|g_{pq}\rangle\,, \tag{11.6}$$

where $\overline{P}_{pq}$ is ϵ-close to $P_{pq} = \frac{1+\langle A_p^\ell | F_q^\ell\rangle}{2}$ and $K = \sqrt{H^{\ell+1}W^{\ell+1}D^{\ell+1}}$ is a normalisation factor. $|g_{pq}\rangle$ is some garbage quantum state.

1.2: Non-linearity

Use an arithmetic circuit and two QRAM queries to obtain $\overline{Y}^{\ell+1}$, an ϵ-approximation of the convolution output $Y_{p,q}^{\ell+1} = (A_p^\ell, F_q^\ell)$ and apply the non-linear function f as a boolean circuit to obatin

$$\frac{1}{K}\sum_{p,q}|p\rangle|q\rangle|f(\overline{Y}_{p,q}^{\ell+1})\rangle|g_{pq}\rangle\,. \tag{11.7}$$

2: **Step 2: Quantum Sampling**

Use Conditional Rotation and Amplitude Amplification to obtain the state

$$\frac{1}{K}\sum_{p,q}\alpha'_{pq}|p\rangle|q\rangle|f(\overline{Y}_{pq}^{\ell+1})\rangle|g_{pq}\rangle\,. \tag{11.8}$$

Perform ℓ_∞ tomography from Theorem 4.3 with precision ν, and obtain classically all positions and values $(p, q, f(\overline{Y}_{pq}^{\ell+1}))$ such that, with high probability, values above ν are known exactly, while others are set to 0.

3: **Step 3: QRAM Update and Pooling**

Update the QRAM for the next layer $A^{\ell+1}$ while sampling. The implementation of pooling (Max, Average, etc.) can be done by a specific update or the QRAM data structure described in Section 11.2.3.

Inner Product Estimation

The following method to estimate inner products is derived from previous work [KLLP19]. With the initial state $|p\rangle|q\rangle\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)|0\rangle$ we apply the queries detailed above in a controlled fashion, followed simply by a Hadamard gate to extract the inner product $\langle A_p|F_q\rangle$ in an amplitude.

$$\frac{1}{\sqrt{2}}\left(|p\rangle|q\rangle|0\rangle|0\rangle+|p\rangle|q\rangle|1\rangle|0\rangle\right)\mapsto\frac{1}{\sqrt{2}}\left(|p\rangle|q\rangle|0\rangle|A_p\rangle+|p\rangle|q\rangle|1\rangle|F_q\rangle\right) \quad (11.12)$$

By applying a Hadamard gate on the third register we obtain the following state,

$$\frac{1}{2}|p\rangle|q\rangle\Big(|0\rangle(|A_p\rangle+|F_q\rangle)+|1\rangle(|A_p\rangle-|F_q\rangle)\Big) \quad (11.13)$$

The probability of measuring 0 on the third register is given by $P_{pq}=\frac{1+\langle A_p|F_q\rangle}{2}$. Thus we can rewrite the previous state as

$$|p\rangle|q\rangle\left(\sqrt{P_{pq}}|0,y_{pq}\rangle+\sqrt{1-P_{pq}}|1,y'_{pq}\rangle\right) \quad (11.14)$$

where $|y_{pq}\rangle$ and $|y'_{pq}\rangle$ are some garbage states.
We can perform the previous circuit in superposition. Since A^ℓ has $H^{\ell+1}W^{\ell+1}$ rows, and F^ℓ has $D^{\ell+1}$ columns, we obtain the state:

$$|u\rangle=\frac{1}{\sqrt{H^{\ell+1}W^{\ell+1}D^{\ell+1}}}\sum_p\sum_q|p\rangle|q\rangle\left(\sqrt{P_{pq}}|0,y_{pq}\rangle+\sqrt{1-P_{pq}}|1,y'_{pq}\rangle\right) \quad (11.15)$$

Therefore the probability of measuring the triplet $(p,q,0)$ in the first three registers is given by

$$P_0(p,q)=\frac{P_{pq}}{H^{\ell+1}W^{\ell+1}D^{\ell+1}}=\frac{1+\langle A_p|F_q\rangle}{2H^{\ell+1}W^{\ell+1}D^{\ell+1}} \quad (11.16)$$

Now we can relate to the Convolution product. Indeed, the triplets $(p,q,0)$ that are the most probable to be measured are the ones for which the value $\langle A_p|F_q\rangle$ is the highest. Recall that each element of $Y^{\ell+1}$ is given by $Y^{\ell+1}_{pq}=(A_p,F_q)$, where "$(\cdot,\cdot)$" denotes the inner product. We see here that we will sample most probably the positions (p,q) for the highest values of $Y^{\ell+1}$, that corresponds to the most important points of $X^{\ell+1}$, by the Eq.(2.15). Note that the values of $Y^{\ell+1}$ can be either positive or negative, which is not an issue thanks to the positiveness of $P_0(p,q)$.

A first approach could be to measure indices (p,q) and rely on the fact that pixels with high values, hence a high amplitude, would have a higher probability to be measured. However we have not exactly the final result, since $\langle A_p|F_q\rangle\neq(A_p,F_q)=\|A_p\|\,\|F_q\|\,\langle A_p|F_q\rangle$. Most importantly we then want to apply a non-linearity $f(Y^{\ell+1}_{pq})$ to each pixel, for instance the ReLu function, which seems not possible with unitary quantum gates if the data is encoded in the amplitudes only. Moreover, due to the normalization of the quantum amplitudes and the high dimension of the Hilbert space of the input, the probability of measuring each pixel is roughly the same, making the sampling inefficient. Given these facts, we have added steps to the circuit, in order to measure $(p,q,f(Y^{\ell+1}_{pq}))$, therefore know the value of a pixel when measuring it, while still measuring the most important points in priority.

Encoding the amplitude in a register

Let $\mathcal{U}$ be the unitary that map $|0\rangle$ to $|u\rangle$

$$|u\rangle = \frac{1}{\sqrt{H^{\ell+1}W^{\ell+1}D^{\ell+1}}} \sum_{p,q} |p\rangle |q\rangle \left(\sqrt{P_{pq}} |0, y_{pq}\rangle + \sqrt{1-P_{pq}} |1, y_{pq}^{'}\rangle \right) \tag{11.17}$$

The amplitude $\sqrt{P_{pq}}$ can be encoded in an ancillary register by using Amplitude Estimation (Theorem 3.2) followed by a Median Evaluation (Theorem 6.2).

For any $\Delta > 0$ and $\epsilon > 0$, we can have a state Δ-close to

$$|u^{'}\rangle = \frac{1}{\sqrt{H^{\ell+1}W^{\ell+1}D^{\ell+1}}} \sum_{p,q} |p\rangle |q\rangle |0\rangle |\overline{P}_{pq}\rangle |g_{pq}\rangle \tag{11.18}$$

with probability at least $1 - 2\Delta$, where $|P_{pq} - \overline{P}_{pq}| \leq \epsilon$ and $|g_{pq}\rangle$ is a garbage state. This requires $O(\frac{\ln(1/\Delta)}{\epsilon})$ queries of $\mathcal{U}$. In the following we discard the third register $|0\rangle$ for simplicity.

The benefit of having $\overline{P}_{pq}$ in a register is to be able to perform operations on it (arithmetic or even non-linear). Therefore we can simply obtain a state corresponding to the exact value of the convolution product. Since we've built a circuit such that $P_{pq} = \frac{1+\langle A_p|F_q\rangle}{2}$, with two QRAM calls, we can retrieve the norm of the vectors by applying the following unitary:

$$|p\rangle |q\rangle |\overline{P}_{pq}\rangle |g_{pq}\rangle |0\rangle |0\rangle \mapsto |p\rangle |q\rangle |\overline{P}_{pq}\rangle |g_{pq}\rangle |\|A_p\|\rangle |\|F_q\|\rangle \tag{11.19}$$

On the fourth register, we can then write $Y_{pq}^{\ell+1} = \|A_p\| \|F_q\| \langle A_p|F_q\rangle$ using some arithmetic circuits (addition, multiplication by a scalar, multiplication between registers). We then apply a boolean circuit that implements the ReLu function on the same register, to obtain an estimate of $f(Y_{pq}^{\ell+1})$ in the fourth register. We finish by inverting the previous computations and obtain the final state

$$|f(\overline{Y}^{\ell+1})\rangle = \frac{1}{\sqrt{H^{\ell+1}W^{\ell+1}D^{\ell+1}}} \sum_{p,q} |p\rangle |q\rangle |f(\overline{Y}_{pq}^{\ell+1})\rangle |g_{pq}\rangle \tag{11.20}$$

Because of the precision ϵ on $|\overline{P}_{pq}\rangle$, our estimation $\overline{Y}_{pq}^{\ell+1} = (2\overline{P}_{pq} - 1) \|A_p\| \|F_q\|$, is obtained with error such that

$$|\overline{Y}_{pq}^{\ell+1} - Y_{pq}^{\ell+1}| \leq 2\epsilon \|A_p\| \|F_q\| \tag{11.21}$$

In superposition, we can bound this error by $|\overline{Y}_{pq}^{\ell+1} - Y_{pq}^{\ell+1}| \leq 2M\epsilon$ where we define

$$M = \max_{p,q} \|A_p\| \|F_q\| \tag{11.22}$$

M is the maximum product between norms of one of the $D^{\ell+1}$ kernels, and one of the regions of X^{ℓ} of size HWD^{ℓ}. Finally, since Eq.(11.21) is valid for all pairs (p, q), the overall error committed on the convolution product can be bounded by $\left\| \overline{Y}^{\ell+1} - Y^{\ell+1} \right\|_{\infty} \leq 2M\epsilon$, where $\|.\|_{\infty}$ denotes the ℓ_{∞} norm. Recall that $Y^{\ell+1}$ is just a reshaped version of $X^{\ell+1}$. Since the non-linearity adds no approximation, we can conclude on the final error committed for a layer of our QCNN

$$\left\| f(\overline{X}^{\ell+1}) - f(X^{\ell+1}) \right\|_{\infty} \leq 2M\epsilon \tag{11.23}$$

At this point, we have established Theorem 11.1 as we have created the quantum state (11.20), with given precision guarantees, in time polylogarithmic in Δ and in the size of X^ℓ and K^ℓ.

We know aim to retrieve classical information from this quantum state. Note that $|Y_{pq}^{\ell+1}\rangle$ is representing a scalar encoded in as many qubits as needed for the precision, whereas $|A_p\rangle$ was representing a vector as a quantum state in superposition, where each element $A_{p,r}$ is encoded in one amplitude (See Section 4.1.1). The next step can be seen as a way to retrieve both encodings at the same time, which will allow an efficient tomography focus on the values of high magnitude.

Conditional rotation

In the following sections, we omit the $\ell+1$ exponent for simplicity. Garbage states are removed as they will not perturb the final measurement. We now aim to modify the amplitudes, such that the highest values of $|f(\overline{Y})\rangle$ are measured with higher probability. As shown in Section 4.1.1, a way to do so consists in applying a conditional rotation on an ancillary qubit, proportionally to $f(\overline{Y}_{pq})$. We will detail the calculation since in the general case $f(\overline{Y}_{pq})$ can be greater than 1. To simplify the notation, we denote it by $x = f(\overline{Y}_{pq})$.

This step consists of applying the following rotation on an ancillary qubit:

$$|x\rangle|0\rangle \mapsto |x\rangle\left(\sqrt{\frac{x}{\max x}}|0\rangle + \beta|1\rangle\right) \tag{11.24}$$

Where $\max x = \max_{p,q} f(\overline{Y}_{pq})$ and $\beta = \sqrt{1-(\frac{x}{\max x})^2}$. Note that in practice it is not possible to have access to $|\max x\rangle$ from the state (11.20), but we will present a method to know *a priori* this value or an upper bound in section 11.2.2.

Let's denote $\alpha_{pq} = \sqrt{\frac{f(\overline{Y}_{pq})}{max_{p,q}(f(\overline{Y}_{pq}))}}$. The ouput of this conditional rotation in superposition on state (11.20) is then

$$\frac{1}{\sqrt{HWD}}\sum_{p,q}|p\rangle|q\rangle|f(\overline{Y}_{pq})\rangle\left(\alpha_{pq}|0\rangle + \sqrt{1-\alpha_{pq}^2}|1\rangle\right) \tag{11.25}$$

Amplitude Amplification

In order to measure $(p,q,f(\overline{Y}_{pq}))$ with higher probability where $f(\overline{Y}_{pq})$ has high value, we could post select on the measurement of $|0\rangle$ on the last register. Otherwise, we can perform an amplitude amplification on this ancillary qubit. Let's rewrite the previous state as

$$\frac{1}{\sqrt{HWD}}\sum_{p,q}\alpha_{pq}|p\rangle|q\rangle|f(\overline{Y}_{pq})\rangle|0\rangle + \sqrt{1-\alpha_{pq}^2}|g'_{pq}\rangle|1\rangle \tag{11.26}$$

Where $|g'_{pq}\rangle$ is another garbage state. The overall probability of measuring $|0\rangle$ on the last register is $P(0) = \frac{1}{HWD}\sum_{pq}|\alpha_{pq}|^2$. The number of queries required to amplify the state $|0\rangle$ is $O(\frac{1}{\sqrt{P(0)}})$ (Theorem 3.2). Since $f(\overline{Y}_{pq}) \in \mathbb{R}^+$, we have $\alpha_{pq}^2 = \frac{f(\overline{Y}_{pq})}{\max_{p,q}(f(\overline{Y}_{pq}))}$. Therefore the number of queries is

$$O\left(\sqrt{\max_{p,q}(f(\overline{Y}_{pq}))}\frac{1}{\sqrt{\frac{1}{HWD}\sum_{p,q}f(\overline{Y}_{pq})}}\right) = O\left(\frac{\sqrt{\max_{p,q}(f(\overline{Y}_{pq}))}}{\sqrt{\mathbb{E}_{p,q}(f(\overline{Y}_{pq}))}}\right) \tag{11.27}$$

Where the notation $\mathbb{E}_{p,q}(f(\overline{Y}_{pq}))$ represents the average value of the matrix $f(\overline{Y})$. It can also be written $\mathbb{E}(f(\overline{X}))$ as in Result 11.1:

$$\mathbb{E}_{p,q}(f(\overline{Y}_{pq})) = \frac{1}{HWD}\sum_{p,q} f(\overline{Y}_{pq}) \tag{11.28}$$

At the end of these iterations, we have modified the state to the following:

$$|f(\overline{Y})\rangle = \frac{1}{\sqrt{HWD}}\sum_{p,q} \alpha'_{pq}\,|p\rangle\,|q\rangle\,|f(\overline{Y}_{pq})\rangle \tag{11.29}$$

Where, to respect the normalization of the quantum state, $\alpha'_{pq} = \frac{\alpha_{pq}}{\sqrt{\sum_{p,q}\frac{\alpha_{pq}^2}{HWD}}}$.

Eventually, the probability of measuring $(p, q, f(\overline{Y}_{pq}))$ is given by

$$p(p,q,f(\overline{Y}_{pq})) = \frac{(\alpha'_{pq})^2}{HWD} = \frac{(\alpha_{pq})^2}{\sum_{p,q}(\alpha_{pq})^2} = \frac{f(\overline{Y}_{pq})}{\sum_{p,q} f(\overline{Y}_{pq})} \tag{11.30}$$

Note that we have used the same type of name $|f(\overline{Y})\rangle$ for both state (11.20) and state (11.29). For now on, this state name will refer only to the latter (11.29).

ℓ_∞ tomography and probabilistic sampling

We can rewrite the final quantum state obtained in (11.29) as

$$|f(\overline{Y}^{\ell+1})\rangle = \frac{1}{\sqrt{\sum_{p,q} f(\overline{Y}^{\ell+1}_{pq})}}\sum_{p,q}\sqrt{f(\overline{Y}^{\ell+1}_{pq})}\,|p\rangle\,|q\rangle\,|f(\overline{Y}^{\ell+1}_{pq})\rangle \tag{11.31}$$

We see here that $f(\overline{Y}^{\ell+1}_{pq})$, the values of each pixel, are encoded in both the last register and in the amplitude. We will use this property to extract efficiently the exact values of high magnitude pixels. For simplicity, we will use instead the notation $f(\overline{X}^{\ell+1}_{n})$ to denote a pixel's value, with $n \in [H^{\ell+1}W^{\ell+1}D^{\ell+1}]$. Recall that $Y^{\ell+1}$ and $X^{\ell+1}$ are reshaped version of the same object.

The pixels with high values will have more probability of being sampled. Specifically, we perform a tomography with ℓ_∞ guarantee and precision parameter $\nu > 0$. See Theorem 4.3 and Section 4.2.1 for details. The ℓ_∞ guarantee allows to obtain each pixel with error at most ν, and require $\widetilde{O}(1/\nu^2)$ samples from the state (11.31). Pixels with low values $f(\overline{X}^{\ell+1}_{n}) < \nu$ will probably not be sampled due to their low amplitude. Therefore the error committed will be significative and we adopt the rule of setting them to 0. Pixels with higher values $f(\overline{X}^{\ell+1}_{n}) \geq \nu$, will be sample with high probability, and only one appearance is enough to get the exact register value $f(\overline{X}^{\ell+1}_{n})$ of the pixel, as is it also written in the last register.

To conclude, let's denote $\mathcal{X}^{\ell+1}_{n}$ the resulting pixel values after the tomography, and compare it to the real classical outputs $f(X^{\ell+1}_{n})$. Recall that the measured values $f(\overline{X}^{\ell+1}_{n})$ are approximated with error at most $2M\epsilon$ with $M = \max_{p,q}\|A_p\|\,\|F_q\|$. The algorithm described above implements the following rules:

$$\begin{cases} |\mathcal{X}^{\ell+1}_{n} - f(X^{\ell+1}_{n})| \leq 2M\epsilon & \text{if} \quad f(\overline{X}^{\ell+1}_{n}) \geq \nu \\ \mathcal{X}^{\ell+1}_{n} = 0 & \text{if} \quad f(\overline{X}^{\ell+1}_{n}) < \nu \end{cases} \tag{11.32}$$

Concerning the running time, one could ask what values of ν are sufficient to obtain enough meaningful pixels. Certainly, this highly depends on the output's size $H^{\ell+1}W^{\ell+1}D^{\ell+1}$ and on the output's content itself. But we can view this question from another perspective, by considering that we sample a constant fraction of pixels given by $\sigma \cdot (H^{\ell+1}W^{\ell+1}D^{\ell+1})$ where $\sigma \in [0, 1]$ is a sampling ratio. Because of the particular amplitudes of state (11.31), the high value pixels will be measured and known with higher probability. The points that are not sampled are being set to 0. We see that this approach is equivalent to the ℓ_∞ tomography, therefore we have

$$\frac{1}{\nu^2} = \sigma \cdot H^{\ell+1}W^{\ell+1}D^{\ell+1} \tag{11.33}$$

$$\nu = \frac{1}{\sqrt{\sigma \cdot H^{\ell+1}W^{\ell+1}D^{\ell+1}}} \tag{11.34}$$

We will use this analogy in the numerical simulations (Section 11.3) to estimate, for a particular QCNN architecture and a particular dataset of images, which values of σ are enough to allow the neural network to learn.

Regularization of the Non Linearity

In the previous steps, we see several appearances of the parameter $\max_{p,q}(f(\overline{Y}^{\ell+1}_{pq}))$. First, for the conditional rotation preprocessing, we need to know this value or an upper bound. Then for the running time, we would like to bound this parameter. Both problems can be solved by replacing the usual ReLu non-linearity with a particular activation function, that we denote by *capReLu*. This function is simply a parametrized ReLu function with an upper threshold, the cap C, after which the function remains constant. The choice of C will be tuned for each particular QCNN, as a tradeoff between accuracy and speed. Otherwise, the only other requirement of the QCNN activation function would be not to allow negative values. This is already often the case for most of the classical CNN. In practice, we expect the capReLu to be as good as a usual ReLu, for convenient values of the cap C (≤ 10). We performed numerical simulations to compare the learning curve of the same CNN with several values of C. See the numerical experiments presented in Section 11.3 for more details.

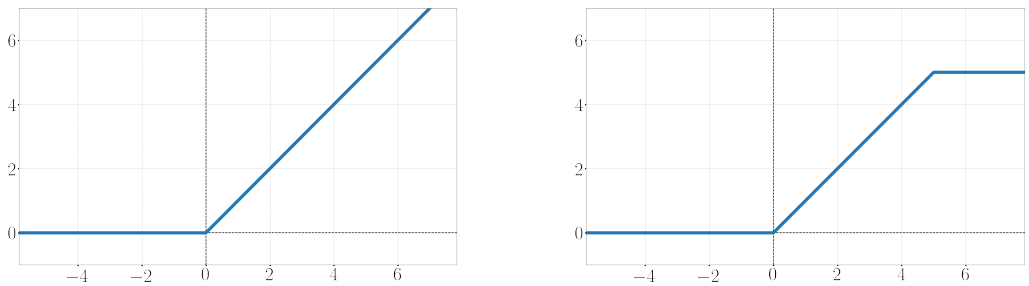


Figure 11.1: Activation functions: ReLu (Left) and capReLu (Right) with a cap C at 5.

11.2.3 Quantum Memory Update

We wish to detail the use of the QRAM between each quantum convolution layer and present how the pooling operation can happen during this phase. General results

about the QRAM are given as Theorem 4.1. Implementation details can be found Section 4.1.2. In this section, we will show how to store samples from the output of the layer ℓ, to create the input of layer $\ell+1$.

Storing the output values during the sampling

At the beginning of layer $\ell+1$, the QRAM must store $A^{\ell+1}$, a matrix where each elements is indexed by (p', r'), and perform $|p'\rangle|0\rangle \mapsto |p'\rangle|A^{\ell+1}_{p'}\rangle$. The data is stored in the QRAM as a tree structure described in Fig.4.2. Each row $A^{\ell+1}_{p'}$ is stored in such a tree $T^{\ell+1}_{p'}$. Each leaf $A^{\ell+1}_{p'r'}$ correspond to a value sampled from the previous quantum state $|f(\overline{Y}^{\ell+1})\rangle$, output of the layer ℓ. The question is to know where to store a sample from $|f(\overline{Y}^{\ell+1})\rangle$ in the tree $T^{\ell+1}_{p'}$.

When a point is sampled from the final state of the quantum convolution, at layer ℓ, as described in Section 11.2.2, we obtain a triplet corresponding to the two positions and the value of a point in the matrix $f(\overline{Y}^{\ell+1})$. We can know where this point belongs in the input of layer $\ell+1$, the tensor $X^{\ell+1}$, by Eq.(2.15), since Y^ℓ is a reshaped version of X^ℓ.

The position in $X^{\ell+1}$, denoted $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$, is then matched to several positions (p', r') in $A^{\ell+1}$. For each p', we write in the tree $T^{\ell+1}_{p'}$ the sampled value at leaf r' and update its parent nodes. Note that leaves that weren't updated will be considered as zeros, corresponding to pixels with too low values, or not selected during pooling (see next section).

Having stored pixels in this way, we can then query $|p'\rangle|0\rangle \mapsto |p'\rangle|A^{\ell}_{p'}\rangle$, using Theorem 4.1, where we correctly have $|A^{\ell+1}_{p'}\rangle = \frac{1}{\left\|A^{\ell+1}_{p'}\right\|}\sum_{r'} A^{\ell+1}_{p'r'}|r'\rangle$. Note that each tree has a logarithmic depth in the number of leaves, hence the running time of writing the output of the quantum convolution layer in the QRAM gives a marginal multiplicative increase, polylogarithmic in the number of points sampled from $|f(\overline{Y}^{\ell+1})\rangle$, namely $O(\log(1/\nu^2))$.

Quantum Pooling

As for the classical CNN, a QCNN should be able to perform pooling operations. We first detail the notations for classical pooling. At the end of layer ℓ, we wish to apply a pooling operation of size P on the output $f(X^{\ell+1})$. We denote by $\tilde{X}^{\ell+1}$ the tensor after the pooling operation. For a point in $f(X^{\ell+1})$ at position $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$, we know to which *pooling region* it belongs, corresponding to a position $(\tilde{i}^{\ell+1}, \tilde{j}^{\ell+1}, \tilde{d}^{\ell+1})$ in $\tilde{X}^{\ell+1}$:

$$\begin{cases} \tilde{d}^{\ell+1} = d^{\ell+1} \\ \tilde{j}^{\ell+1} = \lfloor \frac{j^{\ell+1}}{P} \rfloor \\ \tilde{i}^{\ell+1} = \lfloor \frac{i^{\ell+1}}{P} \rfloor \end{cases} \tag{11.35}$$

We now show how any kind of pooling can be efficiently integrated into our QCNN structure. Indeed the pooling operation will occur during the QRAM update described above, at the end of a convolution layer. At this moment we will store sampled values according to the pooling rules.

In the quantum setting, the output of layer ℓ after tomography is denoted by $\mathcal{X}^{\ell+1}$. After pooling, we will describe it by $\tilde{\mathcal{X}}^{\ell+1}$, which has dimensions $\frac{H^{\ell+1}}{P} \times \frac{W^{\ell+1}}{P} \times$

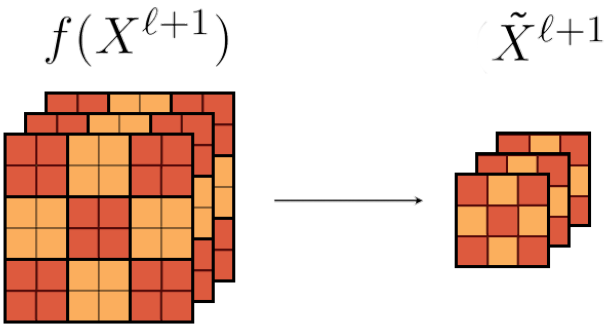


Figure 11.2: A 2×2 tensor pooling. A point in $f(X^{\ell+1})$ (left) is given by its position $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$. A point in $\tilde{X}^{\ell+1}$ (right) is given by its position $(\tilde{i}^{\ell+1}, \tilde{j}^{\ell+1}, \tilde{d}^{\ell+1})$. Different *pooling regions* in $f(X^{\ell+1})$ have separate colours, and each one corresponds to a unique point in $\tilde{X}^{\ell+1}$.

$D^{\ell+1}$. $\tilde{\mathcal{X}}^{\ell+1}$ will be effectively used as input for layer $\ell+1$ and its values should be stored in the QRAM to form the trees $\tilde{T}_{p'}^{\ell+1}$, related to the matrix expansion $\tilde{A}^{\ell+1}$.

However $\mathcal{X}^{\ell+1}$ is not known before the tomography is over. Therefore we have to modify the update rule of the QRAM to implement the pooling in an online fashion, each time a sample from $|f(\overline{X}^{\ell+1})\rangle$ is drawn. Since several sampled values of $|f(\overline{X}^{\ell+1})\rangle$ can correspond to the same leaf $\tilde{A}_{p'r'}^{\ell+1}$ (points in the same *pooling region*), we need an overwrite rule, that will depend on the type of pooling. In the case of Maximum Pooling, we simply update the leaf and the parent nodes if the new sampled value is higher than the one already written. In the case of Average Polling, we replace the actual value with the new averaged value.

In the end, any pooling can be included in the already existing QRAM update. In the worst case, the running time is increased by $\widetilde{O}(P/\nu^2)$, an overhead corresponding to the number of times we need to overwrite existing leaves, with P being a small constant in most cases.

As we will see in Section 11.2.4, the final positions (p, q) that were sampled from $|f(\overline{X}^{\ell+1})\rangle$ and selected after pooling must be stored for further use during the backpropagation phase.

Running Time

We will now summarize the running time for one forward pass of convolution layer ℓ. With $\tilde{O}$ we hide the polylogarithmic factors. We first write the running time of the classical CNN layer, which is given by

$$\widetilde{O}\left(H^{\ell+1}W^{\ell+1}D^{\ell+1} \cdot HWD^{\ell}\right) \tag{11.36}$$

For the QCNN, the previous steps prove Result 11.1 and can be implemented in time

$$\widetilde{O}\left(\frac{1}{\epsilon\nu^2} \cdot \frac{M\sqrt{C}}{\sqrt{\mathbb{E}(f(\overline{X}^{\ell+1}))}}\right) \tag{11.37}$$

Note that, as explain in Section 11.2.2, the quantum running time can also be written

$$\widetilde{O}\left(\sigma H^{\ell+1}W^{\ell+1}D^{\ell+1}\cdot\frac{M\sqrt{C}}{\epsilon\sqrt{\mathbb{E}(f(\overline{X}^{\ell+1}))}}\right) \tag{11.38}$$

with $\sigma \in [0,1]$ being the fraction of sampled elements among $H^{\ell+1}W^{\ell+1}D^{\ell+1}$ of them.

It is interesting to notice that the one quantum convolution layer can also include the ReLu operation and the Pooling operation in the same circuit, for no significant increase in the running time, whereas in the classical CNN each operation must be done on the whole data again.

Let's go through all the important parameters that appear in the quantum running time:

- The error ϵ committed during the inner product estimation, is an empirical parameter Our simulations tend to show that this error can be high without compromising the learning. Indeed the introduction of noise is sometimes interesting in machine learning applications, providing more robust learning [GBCB16, Bis95].

- The parameter $M = \max_{p,q} \|A_p\| \|F_q\|$ as a worst case upper bound during inner product estimation.

- Precision parameter ν can be related to the fraction of sampled elements in the quantum output $|f(\overline{X}^{\ell+1})\rangle$ of the convolution layer, during ℓ_∞ tomography.

- Amplitude amplification adds a multiplicative term $\sqrt{\max\left(f(\overline{X}^{\ell+1})\right)}$ to the running time, replaced here by $\sqrt{C}$, a constant parameter of order $O(1)$, corresponding to the *cap*, or upper bound, of the activation function. See Section 11.2.2 for details. This parameter appears at the conditional rotation step.

- Similarly, the data related value $\mathbb{E}(f(\overline{X}^{\ell+1}))$, appearing during amplitude amplification, denotes the average value in the tensor $f(\overline{X}^{\ell+1})$, as defined in Eq.(11.28).

Finally, in most cases, to recognize kernel features in the input tensor, the size $H \times W$ of the kernels is a sufficient constant fraction of the input size $H^\ell \times W^\ell$. Since $H^{\ell+1} = H^\ell - H + 1$, the classical running time can be seen as quadratic in the input size, whereas the quantum algorithm is almost linear.

Variable Summary

We recall the most important variables for layer ℓ. They represent tensors, their approximations, and their reshaped versions.

Data	Variable	Dimensions	Indices
Input	X^ℓ	$H^\ell \times W^\ell \times D^\ell$	(i^ℓ, j^ℓ, d^ℓ)
	Y^ℓ	$(H^\ell W^\ell) \times D^\ell$	-
	A^ℓ	$(H^{\ell+1}W^{\ell+1}) \times (HWD^\ell)$	(p, r)
Kernel	K^ℓ	$H \times W \times D^\ell \times D^{\ell+1}$	(i, j, d, d')
	F^ℓ	$(HWD^\ell) \times D^{\ell+1}$	(s, q)

Table 11.1: Summary of input variables for the ℓ^{th} layer, along with their meaning, dimensions and corresponding notations. These variables are common for both *quantum* and *classical* algorithms. We have omitted indices for Y^ℓ which don't appear in our work.

Data	Variable	Dimensions	Indices
Output of Quantum Convolution	$f(\overline{Y}^{\ell+1})$	$(H^{\ell+1}W^{\ell+1}) \times D^{\ell+1}$	(p,q)
	$f(\overline{X}^{\ell+1})$	$H^{\ell+1} \times W^{\ell+1} \times D^{\ell+1}$	$(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$
Output of Quantum Tomography	$\mathcal{X}^{\ell+1}$	$H^{\ell+1} \times W^{\ell+1} \times D^{\ell+1}$	$(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$
Output of Quantum Pooling	$\tilde{\mathcal{X}}^{\ell+1}$	$\frac{H^{\ell+1}}{P} \times \frac{W^{\ell+1}}{P} \times D^{\ell+1}$	$(\tilde{i}^{\ell+1}, \tilde{j}^{\ell+1}, \tilde{d}^{\ell+1})$

Table 11.2: Summary of variables describing outputs of the layer ℓ, with the *quantum* algorithm.

Data	Variable	Dimensions	Indices
Output of Classical Convolution	$f(Y^{\ell+1})$	$(H^{\ell+1}W^{\ell+1}) \times D^{\ell+1}$	(p,q)
	$f(X^{\ell+1})$	$H^{\ell+1} \times W^{\ell+1} \times D^{\ell+1}$	$(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$
Output of Classical Pooling	$\tilde{X}^{\ell+1}$	$\frac{H^{\ell+1}}{P} \times \frac{W^{\ell+1}}{P} \times D^{\ell+1}$	$(\tilde{i}^{\ell+1}, \tilde{j}^{\ell+1}, \tilde{d}^{\ell+1})$

Table 11.3: Summary of variables describing outputs of the layer ℓ, with the *classical* algorithm.

Classical and quantum algorithms can be compared with these two diagrams:

$$\begin{cases} \text{Quantum convolution layer}: \ X^{\ell} \to |\overline{X}^{\ell+1}\rangle \to |f(\overline{X}^{\ell+1})\rangle \to \mathcal{X}^{\ell+1} \to \tilde{\mathcal{X}}^{\ell+1} \\ \text{Classical convolution layer}: \ X^{\ell} \to X^{\ell+1} \to f(X^{\ell+1}) \to \tilde{X}^{\ell+1} \end{cases} \tag{11.39}$$

We finally provide some remarks that could clarify some notations ambiguity:

- Formally, the output of the quantum algorithm is $\tilde{\mathcal{X}}^{\ell+1}$. It is used as input for the next layer $\ell+1$. But we consider that all variables' names are *reset* when starting a new layer: $X^{\ell+1} \leftarrow \tilde{\mathcal{X}}^{\ell+1}$.

- For simplicity, we have sometimes replaced the indices $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$ by n to index the elements of the output.

- In Section 11.2.3, the input for layer $\ell+1$ is stored as $A^{\ell+1}$, for which the elements are indexed by (p', r').

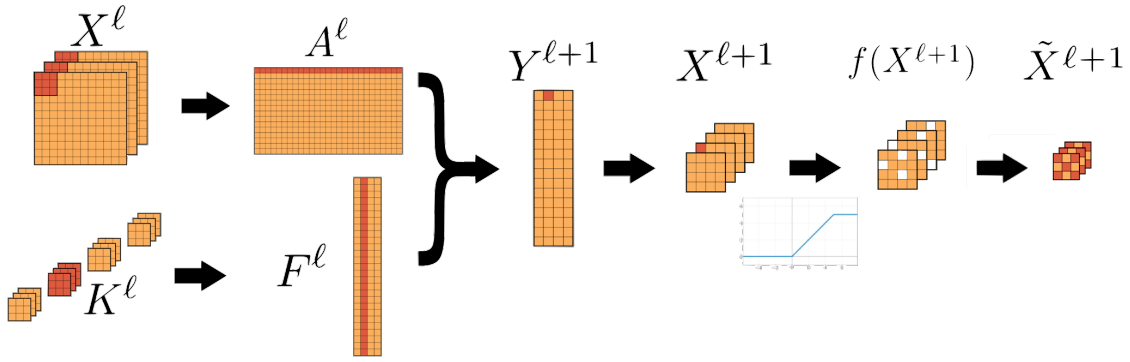


Figure 11.3: Variable summary and representation of a full QCNN layer. From left to right: matrix expansion for input and kernel, parallel matrix product, non-linearity, tomography and pooling.

Quantum-Inspired Classical Algorithm

In Section 5.3, we detailed recent works that have provided quantum inspired classical algorithms for linear algebra that rely on ℓ_2 sampling using classical analogs of the binary search tree (BST) data structure to efficiently estimate inner products. Indeed the inner product can be efficiently approximated classically, analogous to quantum inner product estimation. As shown in [AHKZ20], if $x, y \in \mathbb{R}^n$ are stored in ℓ_2-BST then, with probability at least $1 - \Delta$, a value s that approximates the inner product $\langle x|y\rangle$ can be computed with the following guarantees,

$$|s - \langle x|y\rangle| \leq \begin{cases} \epsilon & \text{in time} \quad \widetilde{O}\left(\frac{\log(1/\Delta)}{\epsilon^2} \frac{\|x\|^2 \|y\|^2}{|\langle x|y\rangle|}\right) \\ \epsilon|\langle x|y\rangle| & \text{in time} \quad \widetilde{O}\left(\frac{\log(1/\Delta)}{\epsilon^2} \|x\|^2 \|y\|^2\right) \end{cases} \tag{11.40}$$

The running time is similar to the quantum inner product estimation presented in Section 11.2.2, but presents a quadratic overhead on the precision ϵ and the norms of the vectors x and y, which in our case would be A_p^ℓ and F_q^ℓ, input and kernel vectors. Similarly, the steps corresponding to the amplitude amplification of Section 11.2.2 can be done classically with a quadratically worse dependence on the parameters.

It is therefore possible to define a classical algorithm inspired by this work if the matrices A^ℓ and F^ℓ are stored in classical ℓ_2 BST. Using the above result, and applying classically non-linearity and pooling, would give a forward pass algorithm with running time,

$$\widetilde{O}\left(H^{\ell+1} W^{\ell+1} D^{\ell+1} \cdot \frac{M^2 C}{\epsilon^2 \mathbb{E}(f(\overline{X}^{\ell+1}))}\right). \tag{11.41}$$

Similar to the quantum algorithm's running time (11.38), we obtain a polylogarithmic dependence on the size of the kernels. We however see a quadratically worse dependence with respect to ϵ, $M = \max_{p,q} \|A_p\| \|F_q\|$, C the upper bound of the non-linearity, and the average value of $f(\overline{X}^{\ell+1})$, too. Recent numerical experiments [ADBL19, AHKZ20] showed that such quantum inspired algorithms were less efficient than the quantum ones, and even than the standard classical algorithms for performing the same tasks.

Finally, the quantum inspired algorithm doesn't provide the speedup characterized by $\sigma \in [0, 1]$, the fraction of sampled elements among $H^{\ell+1} W^{\ell+1} D^{\ell+1}$ of them. Indeed, the probabilistic importance sampling described in Section 11.2.2, that allows sampling the highest values of the convolution product output does not have a classical analog. The importance sampling does not offer an asymptotic speedup, however it could offer constant factor savings that are relevant in practice.

11.2.4 Quantum Backpropagation Algorithm

Preliminaries on classical CNN backpropagation, along with all notations are given in Section Section 2.3.5.

The entire QCNN is made of multiple layers. For the last layer's output, we expect only one possible outcome, or a few in the case of a classification task, which means that the dimension of the quantum output is very small. A full tomography

can be performed on the last layer's output in order to calculate the outcome. The loss $\mathcal{L}$ is then calculated, as a measure of correctness of the predictions compared to the ground truth. As the classical CNN, our QCNN should be able to perform the optimization of its weights (elements of the kernels) to minimize the loss by an iterative method.

Theorem 11.2: Quantum Backpropagation for Quantum CNN

Given the forward pass quantum algorithm in Algorithm 3, the input matrix A^ℓ and the kernel matrix F^ℓ stored in the QRAM for each layer ℓ, and a loss function $\mathcal{L}$, there is a quantum backpropagation algorithm that estimates, for any precision $\delta > 0$, the gradient tensor $\frac{\partial \mathcal{L}}{\partial F^\ell}$ and update each element to perform gradient descent such that

$$\forall (s,q), \left| \frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}} - \overline{\frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}}} \right| \leq 2\delta \left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|_2 \tag{11.42}$$

Let $\frac{\partial \mathcal{L}}{\partial Y^\ell}$ be the gradient with respect to the ℓ^{th} layer. The running time of a single layer ℓ for quantum backpropagation is given by

$$O\left(\left(\left(\mu(A^\ell) + \mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}) \right) \kappa(\frac{\partial \mathcal{L}}{\partial F^\ell}) + \left(\mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}) + \mu(F^\ell) \right) \kappa(\frac{\partial \mathcal{L}}{\partial Y^\ell}) \right) \frac{\log(1/\delta)}{\delta^2} \right) \tag{11.43}$$

where for a matrix V, $\kappa(V)$ is the condition number and $\mu(V)$ is defined in Definition 5.1.

We will detail the quantum algorithm to perform backpropagation on a layer ℓ, and analyze the impact on the derivatives, given by the following diagram:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial X^\ell} \\ \frac{\partial \mathcal{L}}{\partial F^\ell} \end{cases} \leftarrow \frac{\partial \mathcal{L}}{\partial \overline{X}^{\ell+1}} \leftarrow \frac{\partial \mathcal{L}}{\partial f(\overline{X}^{\ell+1})} \leftarrow \frac{\partial \mathcal{L}}{\partial \mathcal{X}^{\ell+1}} \leftarrow \frac{\partial \mathcal{L}}{\partial \tilde{\mathcal{X}}^{\ell+1}} = \frac{\partial \mathcal{L}}{\partial X^{\ell+1}} \tag{11.46}$$

We assume that backpropagation has been done on layer $\ell + 1$. This means in particular that $\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}$ is stored in QRAM. However, as shown on Diagram (11.46), $\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}$ corresponds formally to $\frac{\partial \mathcal{L}}{\partial \tilde{\mathcal{X}}^{\ell+1}}$, and not $\frac{\partial \mathcal{L}}{\partial \overline{X}^{\ell+1}}$. Therefore, we will have to modify the values stored in QRAM to take into account non-linearity, tomography and pooling. We will first consider how to implement $\frac{\partial \mathcal{L}}{\partial X^\ell}$ and $\frac{\partial \mathcal{L}}{\partial F^\ell}$ through backpropagation, considering only convolution product, as if $\frac{\partial \mathcal{L}}{\partial \overline{X}^{\ell+1}}$ and $\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}$ where the same. Then we will detail how to simply modify $\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}$ *a priori*, by setting some of its values to 0.

Quantum Convolution Product

In this section we consider only the quantum convolution product without non-linearity, tomography nor pooling, hence writing its output directly as $X^{\ell+1}$. Regarding derivatives, the quantum convolution product is equivalent to the classical one. Gradient relations (2.18) and (2.19) remain the same. Note that the ϵ-approximation from Section 11.2.2 doesn't participate in gradient considerations.

The gradient relations being the same, we still have to specify the quantum algorithm that implements the backpropagation and outputs classical description

Algorithm 4 Quantum Backpropagation

Require: Precision parameter δ. Data matrices A^ℓ and kernel matrices F^ℓ stored in QRAM for each layer ℓ.

Ensure: Outputs gradient matrices $\frac{\partial \mathcal{L}}{\partial F^\ell}$ and $\frac{\partial \mathcal{L}}{\partial Y^\ell}$ for each layer ℓ.

1: Calculate the gradient for the last layer L using the outputs and the true labels: $\frac{\partial \mathcal{L}}{\partial Y^L}$

2: **for** $\ell = L-1, \cdots, 0$ **do**

3: **Step 1 : Modify the gradient**
With $\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}$ stored in QRAM, set to 0 some of its values to take into account pooling, tomography and non-linearity that occurred in the forward pass of layer ℓ. These values correspond to positions that haven't been sampled nor pooled, since they have no impact on the final loss.

4: **Step 2 : Matrix-matrix multiplications**
With the modified values of $\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}$, use quantum linear algebra (Theorem 5.2) to perform the following matrix-matrix multiplications

$$\begin{cases} (A^\ell)^T \cdot \frac{\partial L}{\partial Y^{\ell+1}} \\ \frac{\partial \mathcal{L}}{\partial Y^{\ell+1}} \cdot (F^\ell)^T \end{cases} \tag{11.44}$$

to obtain quantum states corresponding to $\frac{\partial \mathcal{L}}{\partial F^\ell}$ and $\frac{\partial \mathcal{L}}{\partial Y^\ell}$.

5: **Step 3 : ℓ_∞ tomography**
Using the ℓ_∞ tomography procedure given in Algorithm 1, estimate each entry of $\frac{\partial \mathcal{L}}{\partial F^\ell}$ and $\frac{\partial \mathcal{L}}{\partial Y^\ell}$ with errors $\delta \left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|$ and $\delta \left\| \frac{\partial \mathcal{L}}{\partial Y^\ell} \right\|$ respectively. Store all elements of $\frac{\partial \mathcal{L}}{\partial F^\ell}$ in QRAM.

6: **Step 4 : Gradient descent**
Perform gradient descent using the estimates from step 3 to update the values of F^ℓ in QRAM:

$$F^\ell_{s,q} \leftarrow F^\ell_{s,q} - \lambda \left(\frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}} \pm 2\delta \left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|_2 \right) \tag{11.45}$$

7: **end for**

of $\frac{\partial \mathcal{L}}{\partial X^\ell}$ and $\frac{\partial \mathcal{L}}{\partial F^\ell}$. We have seen that the two main calculations (2.18) and (2.19) are in fact matrix-matrix multiplications both involving $\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}$, the reshaped form of $\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}$. For each, the classical running time is $O(H^{\ell+1}W^{\ell+1}D^{\ell+1}HWD^\ell)$. We know from Theorem 5.2 and Theorem 4.3 a quantum algorithm to perform efficiently a matrix-vector multiplication and return a classical state with ℓ_∞ norm guarantees. For a matrix V and a vector b, both accessible from the QRAM, the running time to perform this operation is

$$O\left(\frac{\mu(V)\kappa(V)\log(1/\delta)}{\delta^2} \right) \tag{11.47}$$

where $\kappa(V)$ is the condition number of the matrix and $\mu(V)$ is a matrix parameter defined in Definition 5.1. Precision parameter $\delta > 0$ is the error committed in the approximation for both Theorems 5.2 and 4.3.

We can therefore apply theses theorems to perform matrix-matrix multiplica-

tions, by simply decomposing them in several matrix-vector multiplications. For instance, in Eq.(2.18), the matrix could be $(A^\ell)^T$ and the different vectors would be each column of $\frac{\partial L}{\partial Y^{\ell+1}}$. We know from Section 5.2 that the global running time to perform quantumly Eq.(2.18) is obtained by replacing $\mu(V)$ by $\mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}) + \mu(A^\ell)$ and $\kappa(V)$ by $\kappa((A^\ell)^T \cdot \frac{\partial \mathcal{L}}{\partial Y^{\ell+1}})$. Likewise, for Eq.(2.19), we have $\mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}) + \mu(F^\ell)$ and $\kappa(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}} \cdot (F^\ell)^T)$.

Note that the dimension of the matrix doesn't appear in the running time since we tolerate a ℓ_∞ norm guarantee for the error, instead of a ℓ_2 guarantee (see Section 4.2.2 for details). The reason why ℓ_∞ tomography is the right approximation here is because the result of these linear algebra operations are rows of the gradient matrices, that are not vectors in an euclidean space, but a series of numbers for which we want to be δ-close to the exact values. See the next section for more details.

It is an open question to see if one can apply the same sub-sampling technique as in the forward pass (Section 11.2.2) and sample only the highest derivatives of $\frac{\partial \mathcal{L}}{\partial X^\ell}$, to reduce the computation cost while maintaining a good optimization.

We then have to understand which elements of $\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}$ must be set to zero to take into account the effects the non-linearity, tomography and pooling.

Quantum Non-Linearity and Tomography

To include the impact of the non-linearity, one could apply the same rule as in (2.20), and simply replace ReLu with capReLu. After the non-linearity, we obtain $f(\overline{X}^{\ell+1})$, and the gradient relation would be given by

$$\left[\frac{\partial \mathcal{L}}{\partial \overline{X}^{\ell+1}}\right]_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} = \begin{cases} \left[\frac{\partial \mathcal{L}}{\partial f(\overline{X}^{\ell+1})}\right]_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} & \text{if } 0 \leq \overline{X}^{\ell+1}_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} \leq C \\ 0 \text{ otherwise} \end{cases} \tag{11.48}$$

If an element of $\overline{X}^{\ell+1}$ was negative or bigger than the cap C, its derivative should be zero during the backpropagation. However, this operation was performed in quantum superposition. In the quantum algorithm, one cannot record at which positions $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$ the activation function was selective or not. The gradient relation (11.48) cannot be implemented *a posteriori*.

We provide a partial solution to this problem, using the fact that quantum tomography must also be taken into account for some derivatives. Indeed, only the points $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$ that have been sampled should have an impact on the gradient of the loss. Therefore we replace the previous relation by

$$\left[\frac{\partial \mathcal{L}}{\partial \overline{X}^{\ell+1}}\right]_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} = \begin{cases} \left[\frac{\partial \mathcal{L}}{\partial \mathcal{X}^{\ell+1}}\right]_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} & \text{if } (i^{\ell+1}, j^{\ell+1}, d^{\ell+1}) \text{ was sampled} \\ 0 \text{ otherwise} \end{cases} \tag{11.49}$$

Nonetheless, we can argue that this approximation will be tolerable:

In the first case where $\overline{X}^{\ell+1}_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} < 0$, the derivatives can not be set to zero as they should. But in practice, their values will be zero after the activation function and such points would not have a chance to be sampled. In conclusion, their derivatives would be zero as required.

In the other case where $\overline{X}^{\ell+1}_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} > C$, the derivatives can not be set to zero as well but the points have a high probability of being sampled. Therefore their derivative will remain unchanged as if we were using a ReLu instead of a capReLu. However in cases where the cap C is high enough, this shouldn't be a source of disadvantage in practice.

Quantum Pooling

From relation (11.49), we can take into account the impact of quantum pooling (see Section 11.2.3) on the derivatives. This case is easier since one can record the selected positions during the QRAM update. Therefore, applying the backpropagation is similar to the classical setting with Eq.(2.21).

$$\left[\frac{\partial \mathcal{L}}{\partial \mathcal{X}^{\ell+1}}\right]_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} = \begin{cases} \left[\frac{\partial \mathcal{L}}{\partial \tilde{\mathcal{X}}^{\ell+1}}\right]_{\tilde{i}^{\ell+1},\tilde{j}^{\ell+1},\tilde{d}^{\ell+1}} & \text{if } (i^{\ell+1},j^{\ell+1},d^{\ell+1}) \in \mathcal{P} \\ 0 \text{ otherwise} \end{cases} \tag{11.50}$$

where $\mathcal{P}$ is the set of indices selected during pooling. Note that we know $\frac{\partial \mathcal{L}}{\partial \tilde{\mathcal{X}}^{\ell+1}}$ as it is equal to $\frac{\partial \mathcal{L}}{\partial X^{\ell+1}}$, the gradient with respect to the input of layer $\ell+1$, known by assumption and stored in the QRAM.

Conclusion and Running Time

In conclusion, given $\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}$ in the QRAM, the quantum backpropagation first consists in applying the relations (11.50) followed by (11.49). The effective gradient now take into account non-linearity, tomography and pooling that occurred during layer ℓ. We can know use apply the quantum algorithm for matrix-matrix multiplication that implements relations (2.19) and (2.18).

Note that the steps in Algorithm 4 could also be reversed: during backpropagation of layer $\ell+1$, when storing values for each elements of $\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}$ in the QRAM, one can already take into account (11.50) and (11.49) of layer ℓ. In this case we directly store $\frac{\partial \mathcal{L}}{\partial \overline{X}^{\ell+1}}$, at no supplementary cost.

Therefore, the running time of the quantum backpropagation for one layer ℓ, given as Algorithm 4, corresponds to the sum of the running times of the circuits for implementing relations (2.18) and (2.19). We finally obtain

$$\begin{aligned} O(((\mu(A^\ell) + \mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}))\kappa((A^\ell)^T \cdot \frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}) \\ + (\mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}) + \mu(F^\ell))\kappa(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}} \cdot (F^\ell)^T))\frac{\log(1/\delta)}{\delta^2}) \end{aligned} \tag{11.51}$$

which can be rewritten as

$$O\left(\left(\left(\mu(A^\ell) + \mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}})\right)\kappa(\frac{\partial \mathcal{L}}{\partial F^\ell}) + \left(\mu(\frac{\partial \mathcal{L}}{\partial Y^{\ell+1}}) + \mu(F^\ell)\right)\kappa(\frac{\partial \mathcal{L}}{\partial Y^\ell})\right)\frac{\log(1/\delta)}{\delta^2}\right). \tag{11.52}$$

Besides storing $\frac{\partial \mathcal{L}}{\partial X^\ell}$, the main output is a classical description of $\frac{\partial \mathcal{L}}{\partial F^\ell}$, necessary to perform gradient descent of the parameters of F^ℓ.

Gradient Descent and Classical equivalence

In this part we will see the impact of the quantum backpropagation compared to the classical case, which can be reduced to a simple noise addition during the gradient descent. Recall that gradient descent, in our case, would consist of applying the following update rule

$$F^\ell \leftarrow F^\ell - \lambda \frac{\partial \mathcal{L}}{\partial F^\ell} \tag{11.53}$$

with the learning rate λ.

Let's denote $x = \frac{\partial \mathcal{L}}{\partial F^\ell}$ and its elements $x_{s,q} = \frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}}$. From the first result of Theorem 5.2 with error $\delta < 0$, and the tomography procedure from Theorem 4.3, with same error δ, we can obtain a classical description of $\frac{\overline{x}}{\|\overline{x}\|_2}$ with ℓ_∞ norm guarantee, such that:

$$\left\| \frac{\overline{x}}{\|\overline{x}\|_2} - \frac{x}{\|x\|_2} \right\|_\infty \leq \delta \tag{11.54}$$

in time $\widetilde{O}(\frac{\kappa(V)\mu(V)\log(\delta)}{\delta^2})$, where we denote V the matrix stored in the QRAM that allows to obtain x, as explained in Section 11.2.4. The ℓ_∞ norm tomography is used so that the error δ is at most the same for each component

$$\forall (s,q), \left| \frac{\overline{x_{s,q}}}{\|\overline{x}\|_2} - \frac{x_{s,q}}{\|x\|_2} \right| \leq \delta \tag{11.55}$$

From the second result of the Theorem 5.2 we can also obtain an estimate $\|\overline{x}\|_2$ of the norm, for the same error δ, such that

$$|\, \|\overline{x}\|_2 - \|x\|_2 \,| \leq \delta \|x\|_2 \tag{11.56}$$

in time $\widetilde{O}(\frac{\kappa(V)\mu(V)}{\delta}\log(\delta))$ (which does not affect the overall asymptotic running time). Using both results we can obtain an unnormalized state close to x such that, by the triangular inequality

$$\begin{aligned}
\|\overline{x} - x\|_\infty &= \left\| \frac{\overline{x}}{\|\overline{x}\|_2} \|\overline{x}\|_2 - \frac{x}{\|x\|_2} \|x\|_2 \right\|_\infty \\
&\leq \left\| \frac{\overline{x}}{\|\overline{x}\|_2} \|\overline{x}\|_2 - \frac{\overline{x}}{\|\overline{x}\|_2} \|x\|_2 \right\|_\infty + \left\| \frac{\overline{x}}{\|\overline{x}\|_2} \|x\|_2 - \frac{x}{\|x\|_2} \|x\|_2 \right\|_\infty \\
&\leq 1 \cdot |\, \|\overline{x}\|_2 - \|x\|_2 \,| + \|x\|_2 \cdot \left\| \frac{\overline{x}}{\|\overline{x}\|_2} - \frac{x}{\|x\|_2} \right\|_\infty \\
&\leq \delta \|x\|_2 + \|\overline{x}\|_2 \delta \leq 2\delta \|x\|_2
\end{aligned} \tag{11.57}$$

in time $\widetilde{O}(\frac{\kappa(V)\mu(V)\log(\delta)}{\delta^2})$. In conclusion, with ℓ_∞ norm guarantee, having also access to the norm of the result is costless.

Finally, the noisy gradient descent update rule, expressed as $F^\ell_{s,q} \leftarrow F^\ell_{s,q} - \lambda \overline{\frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}}}$ can written in the worst case with

$$\overline{\frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}}} = \frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}} \pm 2\delta \left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|_2 \tag{11.58}$$

To summarize, using the quantum linear algebra from Theorem 5.2 with ℓ_∞ norm tomography from Theorem 4.3, both with error δ, along with norm estimation with

relative error δ too, we can obtain classically the unnormalized values $\overline{\frac{\partial \mathcal{L}}{\partial F^\ell}}$ such that $\left\| \overline{\frac{\partial \mathcal{L}}{\partial F^\ell}} - \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|_\infty \leq 2\delta \left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|_2$ or equivalently

$$\forall (s,q), \left| \overline{\frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}}} - \frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}} \right| \leq 2\delta \left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|_2 \tag{11.59}$$

Therefore the gradient descent update rule in the quantum case becomes $F^\ell_{s,q} \leftarrow F^\ell_{s,q} - \lambda \overline{\frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}}}$, which in the worst case becomes

$$F^\ell_{s,q} \leftarrow F^\ell_{s,q} - \lambda \left(\frac{\partial \mathcal{L}}{\partial F^\ell_{s,q}} \pm 2\delta \left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|_2 \right) \tag{11.60}$$

This proves the Theorem 11.2. This update rule can be simulated by the addition of a random relative noise given as a Gaussian centered on 0, with a standard deviation equal to δ. This is how we will simulate quantum backpropagation in the next Section.

Compared to the classical update rule, this corresponds to the addition of noise during the optimization step. This noise decreases as $\left\| \frac{\partial \mathcal{L}}{\partial F^\ell} \right\|_2$, which is expected to happen while converging. Recall that the gradient descent is already a stochastic process. Therefore, we expect that such noise, with acceptable values of δ, will not disturb the convergence of the gradient, as the following numerical simulations tend to confirm.

11.3 Numerical Simulations

As described above, the adaptation of the CNNs to the quantum setting implies some modifications that could alter the efficiency of the learning or classifying phases. We now present some experiments to show that such modified CNNs can converge correctly, as the original ones.

The experiment, using the PyTorch library [PGC+17], consists of training classically a small convolutional neural network for which we have added a "quantum" sampling after each convolution, as in Section 11.2.3. Instead of parameterizing it with the precision ν, we have chosen to use the sampling ratio σ that represents the number of samples drawn during tomography. These two definitions are equivalent, as shown in Section 11.2.2, but the second one is more intuitive regarding the running time and the simulations.

We also add a noise simulating the amplitude estimation (Section 11.2.2, parameter: ϵ), followed by a capReLu instead of the usual ReLu (Section 11.2.2, parameter: C), and a noise during the backpropagation (Section 11.2.4, parameter: δ). In the following results, we observe that our quantum CNN is able to learn and classify visual data from the widely used MNIST dataset. This dataset is made of 60.000 training images and 10.000 testing images of handwritten digits. Each image is a 28x28 grayscale pixels between 0 and 255 (8 bits encoding), before normalization.

Let's first observe the "quantum" effects on an image of the dataset. In particular, the effect of the capped non-linearity, the introduction of noise and the quantum sampling.

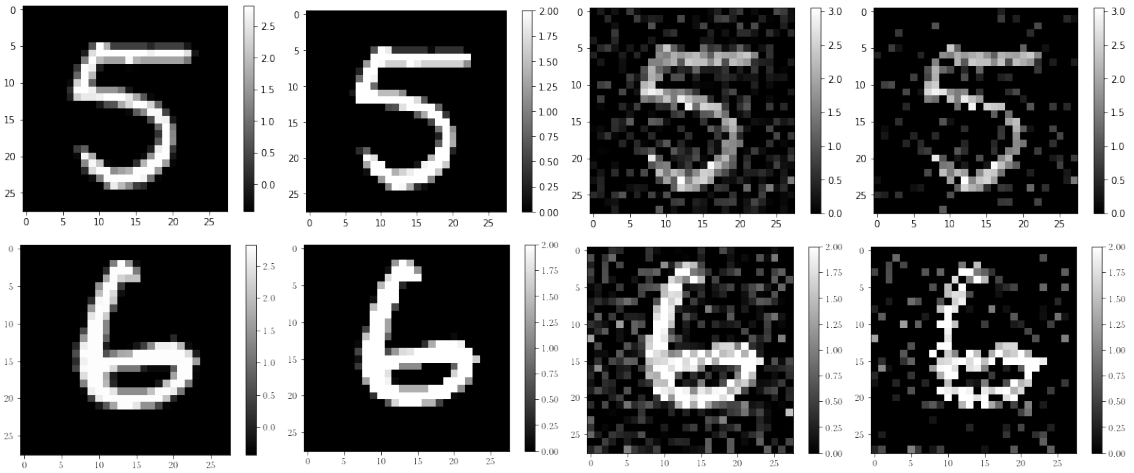


Figure 11.4: Effects of the QCNN on a 28x28 input image. From left to right: original image, image after applying a capReLu activation function with a cap C at 2.0, introduction of a strong noise during amplitude estimation with $\epsilon = 0.5$, quantum sampling with ratio $\sigma = 0.4$ that samples the highest values in priority. The useful information tends to be conserved in this example. The side gray scale indicates the value of each pixel. Note that during the QCNN layer, a convolution is supposed to happen before the last image but we chose not to perform it for better visualization.

We now present the full simulation of our quantum CNN. In the following, we use a simple network made of 2 convolution layers, and compare our quantum CNN to the classical one. The first and second layers are respectively made of 5 and 10 kernels, both of size 7x7. A three-layer fully connected network is applied at the end and a softmax activation function is applied on the last layer to detect the predicted outcome over 10 classes (the ten possible digits). Note that we didn't introduce pooling, being equivalent between quantum and classical algorithms and not improving the results on our CNN. The objective of the learning phase is to minimize the loss function, defined by the negative log-likelihood of the classification on the training set. The optimizer used was a built-in Stochastic Gradient Descent.

Using PyTorch, we have been able to implement the following quantum effects (the first three points are shown in Fig.11.4):

- The addition of noise, to simulate the approximation of amplitude estimation during the forward quantum convolution layer, by adding Gaussian noise centered on 0 and with standard deviation $2M\epsilon$, with $M = \max_{p,q} \|A_p\| \|F_q\|$, as given by Eq.(11.23).

- A modification of the non-linearity: a ReLu function that becomes constant above the value T (the cap).

- A sampling procedure to apply on a tensor with a probability distribution proportional to the tensor itself, reproducing the quantum sampling with ratio σ.

- The addition of noise during the gradient descent, to simulate the quantum backpropagation, by adding a Gaussian noise centered on 0 with standard de-

viation δ, multiplied by the norm of the gradient, as given by Eq.(11.60).

The CNN used for this simulation may seem "small" compared to the standards AlexNet [KSH12] or VGG-16 [SZ14], or those used in industry. However simulating this small QCNN on a classical computer was already very computationally intensive and time consuming, due to the"quantum" sampling task, apparently not optimized for a classical implementation in PyTorch. Every single training curve showed in Fig.11.6 could last for 4 to 8 hours. Hence adding more convolutional layers wasn't convenient. Similarly, we didn't compute the loss on the whole testing set (10.000 images) during the training to plot the testing curve. However we have computed the test losses and accuracies once the model trained (see Table 11.4), in order to detect potential overfitting cases.

We now present the result of the training phase for a quantum version of this CNN, where partial quantum sampling is applied, for different sampling ratio (number of samples taken from the resulting convolution). Since the quantum sampling gives more probability to observe high value pixels, we expect to be able to learn correctly even with a small ratio ($\sigma \leq 0.5$). We compare these training curves to the classical one. The learning has been done on two epochs, meaning that the whole dataset is used twice. The following plots show the evolution of the loss $\mathcal{L}$ during the iterations on batches. This is the standard indicator of the good convergence of a neural network learning phase. We can compare the evolution of the loss between a classical CNN and our QCNN for different parameters.

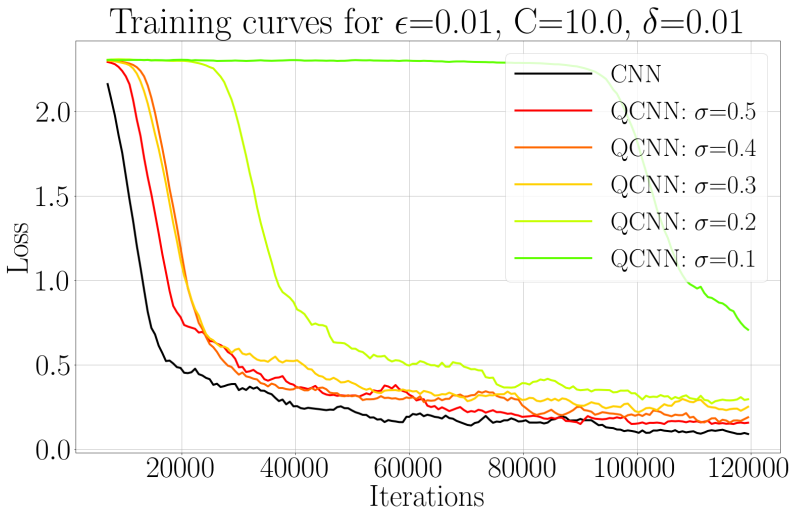


Figure 11.5: Training curves comparison between the classical CNN and the Quantum CNN (QCNN) for $\epsilon = 0.01$, $C = 10$, $\delta = 0.01$ and the sampling ratio σ from 0.1 to 0.5. We can observe a learning phase similar to the classical one, even for a weak sampling of 20% or 30% of each convolution output, which tends to show that the meaningful information is distributed only at certain locations of the images, coherently with the purpose of the convolution layer. Even for a very low sampling ratio of 10%, we observe a convergence despite a late start.

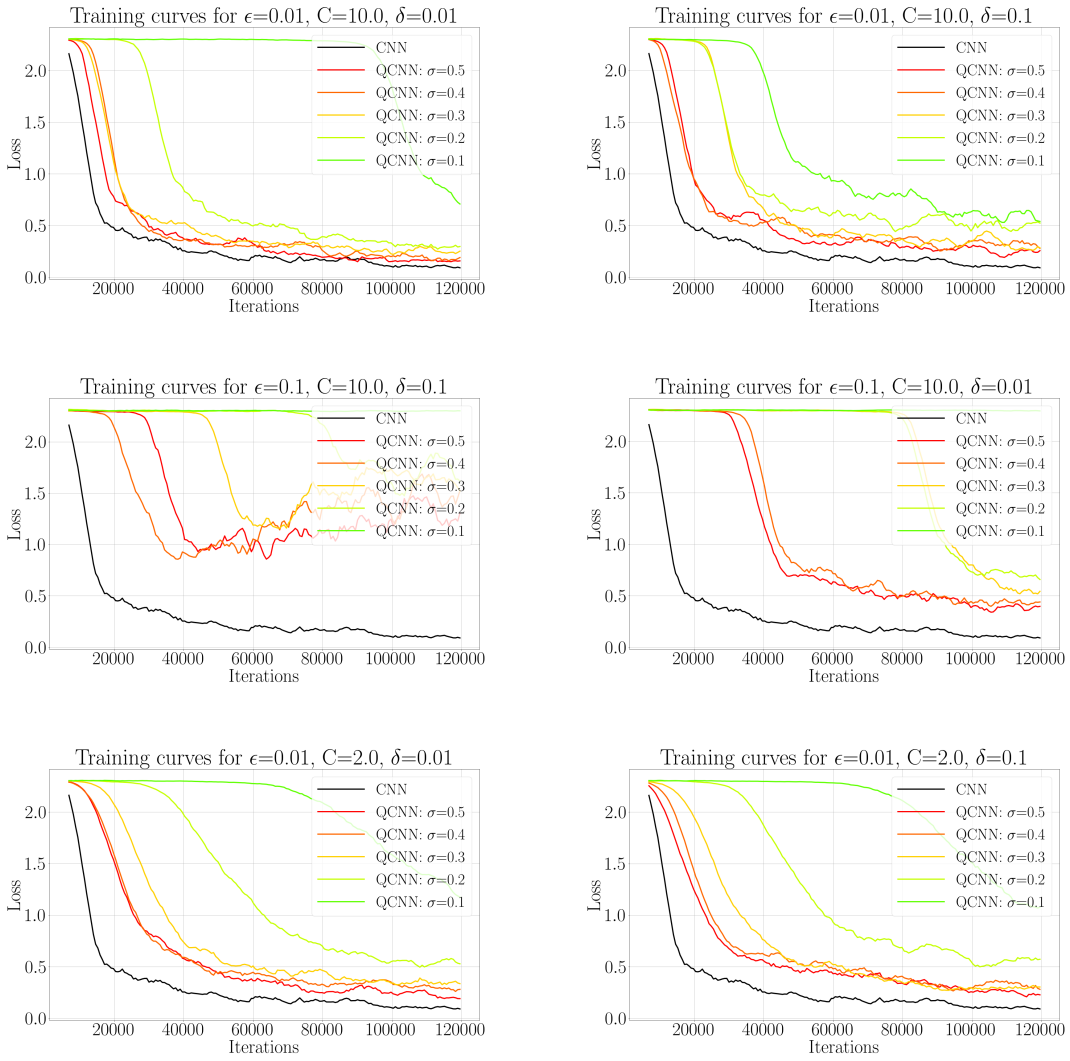


Figure 11.6: Numerical simulations of the training of the QCNN. These training curves represent the evolution of the Loss $\mathcal{L}$ as we iterate through the MNIST dataset. For each graph, the amplitude estimation error ϵ $(0.1, 0.01)$, the non-linearity cap C $(2, 10)$, and the backpropagation error δ $(0.1, 0.01)$ are fixed whereas the quantum sampling ratio σ varies from 0.1 to 0.5. We can compare each training curve to the classical learning (CNN). Note that these training curves are smoothed, over windows of 12 steps, for readability.

In the following we report the classification results of the QCNN when applied on the test set (10.000 images). We distinguish to use cases: in Table 11.4 the QCNN has been trained quantumly as described in this Chapter, whereas in Table 11.5 we first have trained the classical CNN, then transferred the weights to the QCNN only for the classification. This second use case has a global running time worst than the first one, but we see it as another concrete application: quantum machine learning could be used only for faster classification from a classically generated model, which could be the case for high rate classification task (e.g. for autonomous systems, classification over many simultaneous inputs). We report the test loss and accuracy for different values of the sampling ratio σ, the amplitude estimation error ϵ, and for the backpropagation noise δ in the first case. The cap C is fixed at 10. These

values must be compared to the classical CNN classification metrics, for which the loss is 0.129 and the accuracy is 96.1%. Note that we used a relatively small CNN and hence the accuracy is just over 96%, lower than the best possible accuracy with larger CNN.

QCNN Test - Classification					
σ	ϵ	0.01		0.1	
	δ	0.01	0.1	0.01	0.1
0.1	Loss	0.519	0.773	2.30	2.30
	Accuracy	82.8%	74.8%	11.5%	11.7%
0.2	Loss	0.334	0.348	0.439	1.367
	Accuracy	89.5%	89.0%	86.2%	54.1%
0.3	Loss	0.213	0.314	0.381	0.762
	Accuracy	93.4%	90.3%	87.9%	76.8%
0.4	Loss	0.177	0.215	0.263	1.798
	Accuracy	94.7%	93.3%	91.8%	34.9%
0.5	Loss	0.142	0.211	0.337	1.457
	Accuracy	95.4%	93.5%	89.2%	52.8%

Table 11.4: QCNN trained with quantum backpropagation on MNIST dataset. With $C = 10$ fixed.

QCNN Test - Classification			
σ	ϵ	0.01	0.1
0.1	Loss	1.07	1.33
	Accuracy	86.1%	78.6%
0.2	Loss	0.552	0.840
	Accuracy	92.8%	86.5%
0.3	Loss	0.391	0.706
	Accuracy	94,3%	85.8%
0.4	Loss	0.327	0.670
	Accuracy	94.4%	84.0%
0.5	Loss	0.163	0.292
	Accuracy	95.9%	93.5%

Table 11.5: QCNN created from a classical CNN trained on MNIST dataset. With $\delta = 0.01$ and $C = 10$ fixed.

Our simulations show that the QCNN is able to learn despite the introduction of noise, tensor sampling and other modifications. In particular it shows that only a fraction of the information is meaningful for the neural network, and that the quantum algorithm captures this information in priority. This learning can be more or less efficient depending on the choice of the key parameters. For reasonable values of these parameters, the QCNN is able to converge during the training phase. It can then classify correctly on both training and testing set, indicating that it does not overfit the data.

We notice that the learning curves sometimes present a late start before the convergence initializes, in particular for small sampling ratio. This late start can be due to the random initialization of the kernel weights, which performs a meaningless convolution, a case where the quantum sampling of the output is of no interest. However it is very interesting to see that despite this late start, the kernel can start converging once they have found a good combination.

Overall, the QCNN may present some behaviors that do not have a classical equivalent. Understanding their potential effects, positive or negative, is an open question, all the more so as the effects of the classical CNN's hyperparameters are already a topic an active research [SWM17]. Note also that the size of the neural network used in this simulation is rather small. A following experiment would be to simulate a quantum version of a standard deeper CNN (AlexNet or VGG-16), eventually on more complex datasets, such as CIFAR-10 [KH09] or Fashion MNIST [XRV17].

11.4 Conclusions

We have presented a quantum algorithm for evaluating and training convolutional neural networks (CNN). At the core of this algorithm, we have developed the first quantum algorithm for computing a convolution product between two tensors, with a substantial speed up. This technique could be reused in other signal processing tasks that would benefit an enhancement by a quantum computer. Layer by layer, convolutional neural networks process and extract meaningful information. Following this idea of learning foremost important features, we have proposed a new approach of quantum tomography where the most meaningful information is sampled with higher probability, hence reducing the complexity of our algorithm.

Our quantum CNN is complete in the sense that almost all classical architectures can be implemented in a quantum fashion: any (non negative an upper bounded) non-linearity, pooling, number of layers and size of kernels are available. Our circuit is shallow, indeed one could repeat the main loop many times on the same shallow circuit, since performing the convolution product uses shallow linear algebra techniques, and is similar for all layers. The pooling and non-linearity are included in the loop. Our building block approach, layer by layer, allows a high modularity, and can be combined with previous works on quantum feedforward neural network [AHKZ20] (see Section 10.2).

The running time presents a speedup compared to the classical algorithm, due to fast linear algebra when computing the convolution product, and by only sampling the important values from the resulting quantum state. This speedup can be highly significant in cases where the number of channels D^ℓ in the input tensor is high (high dimensional time series, videos sequences, games play) or when the number of kernels $D^{\ell+1}$ is big, allowing deep architectures for CNN, which was the case in the recent breakthrough of DeepMind AlphaGo algorithm [SHM$^+$16]. The quantum CNN also allows larger kernels, that could be used for larger input images, since the size the kernels must be a constant fraction of the input in order to recognize patterns. However, despite our new techniques to reduce the complexity, applying a non-linearity and reusing the result of a layer for the next layer make register encoding and state tomography mandatory, hence preventing from having an exponential speedup on the number of input parameters.

Finally we have presented a backpropagation algorithm that can also be implemented as a quantum circuit. The numerical simulations on a small CNN show that despite the introduction of noise and sampling, the QCNN can efficiently learn to classify visual data from the MNIST dataset, performing a similar accuracy than the classical CNN.

Chapter 12

NISQ Algorithm for Orthogonal Neural Networks

"La vision scientifique et la vision poétique, loin de s'exclure, se rejoignent pour nous faire percevoir le monde dans sa véritable richesse."

Hubert Reeves
Malicorne. Réflexions d'un observateur de la nature (1990)

In this chapter, we present a new training method for neural networks that preserves perfect orthogonality while having the same running time as usual gradient descent methods without the orthogonality condition, thus achieving the best of both worlds, most efficient training and perfect orthogonality.

The main idea comes from the quantum world, where we know that any quantum circuit corresponds to an operation described by a unitary matrix, which if we only use gates with real amplitudes is an orthogonal matrix. In particular, we propose a novel special-architecture quantum circuit, for which there is an efficient way to map the elements of the orthogonal weight matrix to the parameters of the gates of the quantum circuit and vice versa. In other words, while performing a gradient descent on the elements of the weight matrix individually does not preserve orthogonality, performing a gradient descent on the parameters of the quantum circuit preserves orthogonality (since any quantum circuit with real parameters corresponds to an orthogonal matrix) and is equivalent to updating the weight matrix. We also prove that performing gradient descent on the parameters of the quantum circuit can be done efficiently classically (with constant update cost per parameter) thus concluding that there exists a quantum-inspired, but fully classical way of efficiently training perfectly orthogonal neural networks.

Moreover, the special-architecture quantum circuit we defined has many properties that make it a good candidate for NISQ implementation (see Section 3.3): it uses only one type of quantum gates, requires simple connectivity between the qubits, has depth linear in the input and output node sizes, and benefits from powerful error mitigation techniques that make it resilient to noise. This allows us to also propose an inference method running the quantum circuit on data which might

Algorithm	Feedforward Pass	Training
Quantum Pyramidal Circuit (This work)	$2n/\delta^2 = O(n/\delta^2)$	$O(n^2)$
Classical Pyramidal Circuit (This work)	$2n(n-1) = O(n^2)$	
Classical Approximated OrthoNN (SVB) [JLW+19]	$n^2 = O(n^2)$	$O(n^3)$
Classical Strict OrthoNN (Stiefel Manifold) [JLW+19]	$n^2 = O(n^2)$	$O(n^3)$
Standard Neural Network (non orthogonal)	$n^2 = O(n^2)$	$O(n^2)$

Table 12.1: Running times summary. n is the size of the input and output vectors, δ is the error parameter in the quantum implementation. See Section 2.3.3 for details on related work.

offer a faster running time, given the shallow depth of the quantum circuit.

Our main contributions are summarized in Table 12.1, where we have considered the time to perform a feedforward pass, or one gradient descent step. A single neural network layer is considered, with input and output of size n.

12.1 A Parametrized Quantum Circuit for Orthogonal Neural Networks

Preliminaries on orthogonal neural networks, along with all notations are given in Section 2.3.3, as well as Section 4.1.2 for data loaders with unary encoding.

In the following, we will define a special-architecture parametrized quantum circuit that will be useful for performing training and inference on orthogonal neural networks. As we said, the training will be completely classical in the end, but the intuition of the new method comes from this quantum circuit, while the inference can happen both classically or by applying this quantum circuit.

12.1.1 The RBS Gate

The quantum circuit proposed in this work (see Fig.12.1), which implements a fully connected neural network layer with an orthogonal weight matrix, uses only one type of quantum gate, the Reconfigurable Beam Splitter (RBS) gate. This two-qubit gate is parametrizable with one angle $\theta \in [0, 2\pi]$. Its matrix representation is given as:

$$RBS(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad RBS(\theta) : \begin{cases} |01\rangle \mapsto \cos\theta\,|01\rangle - \sin\theta\,|10\rangle \\ |10\rangle \mapsto \sin\theta\,|01\rangle + \cos\theta\,|10\rangle \end{cases} \tag{12.1}$$

We can think of this gate as a rotation in the two-dimensional subspace spanned by the basis $\{|01\rangle, |10\rangle\}$, while it acts as the identity in the remaining subspace $\{|00\rangle, |11\rangle\}$. Or equivalently, starting with two qubits, one in the $|0\rangle$ state and the other one in the state $|1\rangle$, the qubits can be swapped or not in superposition. The qubit $|1\rangle$ stays on its wire with amplitude $\cos\theta$ or switches with the other qubit with amplitude $+\sin\theta$ if the new wire is below ($|10\rangle \mapsto |01\rangle$) or $-\sin\theta$ if the new wire is above ($|01\rangle \mapsto |10\rangle$). Note that in the two other cases ($|00\rangle$ and $|11\rangle$) the RBS gate acts as identity.

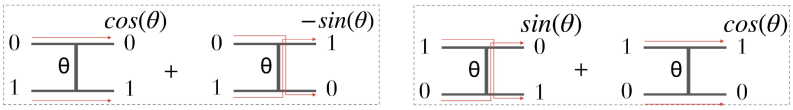
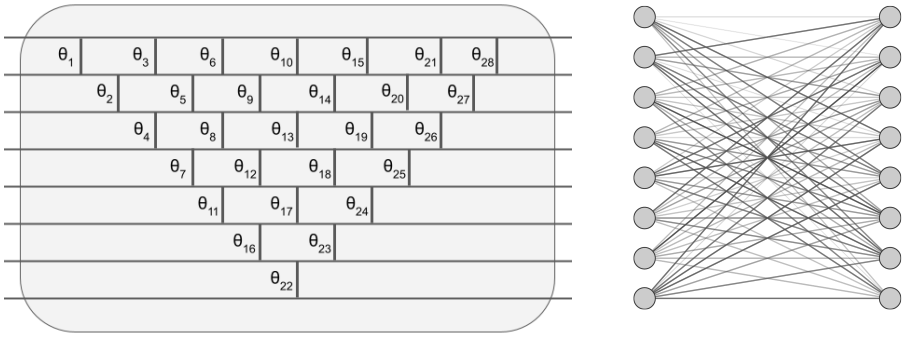


Figure 12.1: Representation of the quantum mapping from Eq.(12.1) on two qubits.

See Fig.3.6a and Section 3.4.1 for practical circuit of the *RBS* gate and implementation with real quantum computers.

12.1.2 Quantum Pyramidal Circuit

We now propose a quantum circuit that implements an orthogonal layer of a neural network [KLM21]. The circuit is a pyramidal structure of *RBS* gates, each with an independent angle, as represented in Fig.12.2a. In Section 12.1.3 and 12.2, more details are provided concerning respectively the input loading, and the equivalence with a neural network's orthogonal layer.



(a) (b)

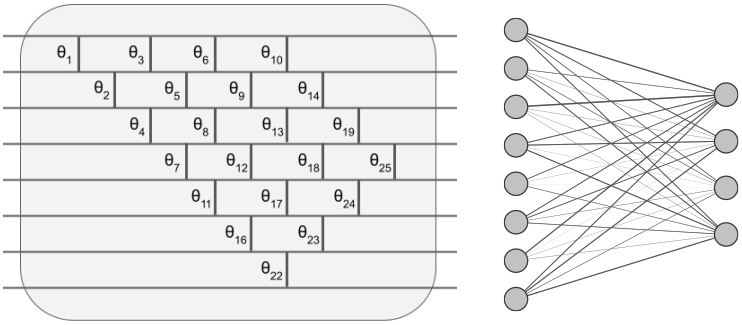
Figure 12.2: (a) Quantum circuit for an 8x8 fully connected, orthogonal layer. Each vertical line corresponds to an *RBS* gate with its angle parameter. And (b), the equivalent classical orthogonal neural network 8x8 layer.

To mimic a given classical layer with a quantum circuit, the number of output qubits should be the size of the classical layer's output. We refer to the *square case* when the input and output sizes are equal, and to the *rectangular case* otherwise (Fig.12.3a).

The important property to note is that the number of parameters of the quantum pyramidal circuit corresponding to a neural network layer of size $n \times d$ is $(2n-1-d) * d/2$ exactly the same as the number of degrees of freedom of an orthogonal matrix of dimension $n \times d$.

For simplicity, we pursue our analysis using only the *square case* but everything can be easily extended to the rectangular case. As we said, the full pyramidal structure of the quantum circuit described above imposes the number of free parameters to be $n(n-1)/2$, the exact number of free parameters to specify a $n \times n$ orthogonal matrix.

In Section 12.2 we will show how the parameters of the gates of this pyramidal circuit can be easily related to the elements of the orthogonal matrix of size $n \times n$ that describes it. We note that alternative architectures can be imagined as long as



(a) (b)

Figure 12.3: (a) Quantum circuit for a *rectangular* 8x4 fully connected orthogonal layer, and (b) the equivalent 8x4 classical orthogonal neural network. They both have 22 free parameters.

the number of gate parameters is equal to the parameters of the orthogonal weight matrix and a simple mapping between them and the elements of the weight matrix can be found.

Note finally that this circuit has linear depth and is convenient for near term quantum hardware platforms with restricted connectivity. Indeed, the distribution of the *RBS* gates requires only nearest neighbor connectivity between qubits.

12.1.3 Loading the Data

Before applying the quantum pyramidal circuit, we will need to upload the classical data into the quantum circuit. As introduced in Section 4.1.1, we will use one qubit per feature of the data. For this, we use a unary amplitude encoding of the input data (see Definition 4.3) that we will recall briefly. Let's consider an input sample $x = (x_0, \cdots, x_{n-1}) \in \mathbb{R}^n$, such that $\|x\|_2 = 1$. We will encode it in a superposition of unary states:

$$|x\rangle = x_0 |10\cdots0\rangle + x_1 |010\cdots0\rangle + \cdots + x_{n-1} |0\cdots01\rangle \tag{12.2}$$

We can also rewrite the previous state as $|x\rangle = \sum_{i=0}^{n-1} x_i |e_i\rangle$, where $|e_i\rangle$ represents the ith unary state with a $|1\rangle$ in the ith position $|0\cdots010\cdots0\rangle$. Recent work [JDM+20] proposed a logarithmic depth data loader circuit for loading such states. Here we will use a much simpler circuit. It is a linear depth cascade of n-1 *RBS* gates which, due to the particular structure of our quantum pyramidal circuit, only adds 2 extra steps to our circuit.

The circuit starts in the all $|0\rangle$ state and flips the first qubit using an X gate, in order to obtain the unary state $|10\cdots0\rangle$ as shown in Fig.12.4. Then a cascade of *RBS* gates allow to create the state $|x\rangle$ using a set of $n-1$ angles $\alpha_0, \cdots, \alpha_{n-2}$. Using Eq.(12.1), we will choose the angles such that, after the first *RBS* gate of the loader, the qubits would be in the state $x_0 |100\cdots\rangle + \sin(\alpha_0) |010\cdots\rangle$ and after the second one in the state $x_0 |100\cdots\rangle + x_1 |010\cdots\rangle + \sin(\alpha_0)\sin(\alpha_1) |001\cdots\rangle$ and so on, until obtaining $|x\rangle$ as in Eq.(12.2). To this end, we simply perform a classical pre-processing to compute recursively the n-1 loading angles, in time $O(n)$. We choose $\alpha_0 = \arccos(x_0)$, $\alpha_1 = \arccos(x_1 \sin^{-1}(\alpha_0))$, $\alpha_2 = \arccos(x_2 \sin^{-1}(\alpha_0)\sin^{-1}(\alpha_1))$ and so on (see Section 4.1.2).

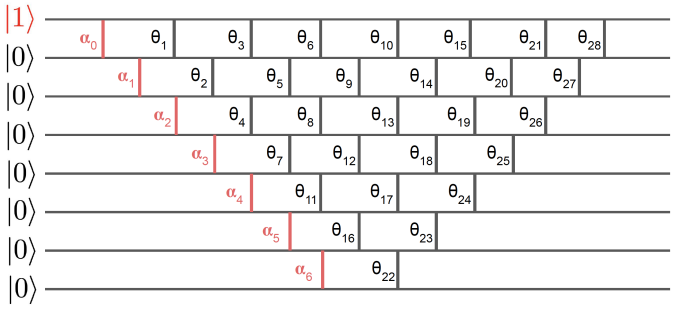


Figure 12.4: The 8 dimensional linear data loader circuit (in red) is efficiently embedded before the pyramidal circuit. The input state is the first unary state. The angles parameters $\alpha_0, \cdots, \alpha_{n-2}$ are classically pre-computed from the input vector.

The ability of loading data in such a way relies on the assumption that each input vector is normalized, i.e. $\|x\|_2 = 1$. This normalization constraint could seem arbitrary and impact the ability to learn from the data. In fact, in the case of an orthogonal neural network, this normalization shouldn't degrade the training because orthogonal weight matrices are in fact orthonormal and thus norm-preserving. Hence, changing the norm of the input vector, by diving each component by $\|x\|_2$, in both classical and quantum settings is not a problem. The normalization would impose that each input has the same norm, or the same "luminosity" in the context of images, which can be helpful or harmful depending on the case.

12.2 Orthogonal Feedforward Pass

In this section, we will detail the effect of the quantum pyramidal circuit on an input encoded in a unary basis, as in Eq.(12.2). We will also see in the end how to simulate this quantum circuit classically with a small overhead and thus be able to provide a fully classical scheme.

Let's first consider one pure unary input, where only the qubit j is in state $|1\rangle$ (e.g. $|00000010\rangle$). This unary input will be transformed into a superposition of unary states, each with an amplitude. If we consider again only one of these possible unary outputs, where only the qubit i is in state $|1\rangle$, its amplitude can be interpreted as a conditional amplitude to transfer the $|1\rangle$ from qubit j to qubit i. Intuitively, this value is the sum of the quantum amplitudes associated with each possible path that *connects* the qubit j to qubit i, as shown in Fig.12.5. Using this image of *connectivity* between input and output qubits, we can construct a matrix $W \in \mathbb{R}^{n \times n}$, where each element W_{ij} is the overall conditional amplitude to transfer the $|1\rangle$ from qubit j to qubit i.

Fig.12.5 shows an example where exactly three paths can be taken to map the input qubit $j = 6$ (the 7^{th} unary state) to the qubit $i = 5$ (the 6^{th} unary state). Each path comes with a certain amplitude. For instance, one of the paths (the red one in Fig.12.5) moves up at the first gate, and then stays put in the next three gates, with a resulting amplitude of $-\sin(\theta_{16})\cos(\theta_{17})\cos(\theta_{23})\cos(\theta_{24})$. The sum of the amplitudes of all possible paths give us the element W_{56} of the matrix W (where,

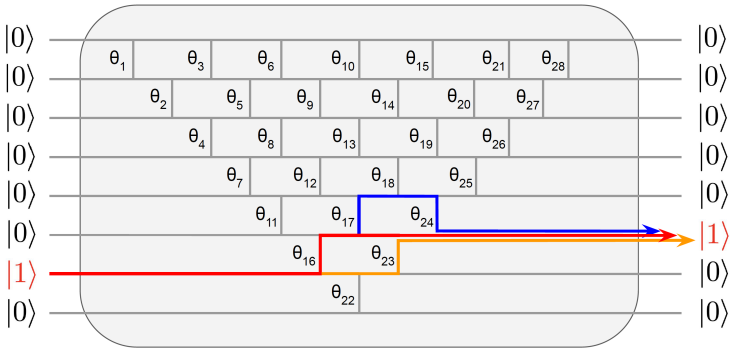


Figure 12.5: The three possibles paths from the 7^{th} unary state to the 6^{th} unary state, on an 8x8 quantum pyramidal circuit.

for simplicity, $s(\theta)$ and $c(\theta)$ respectively stand for $\sin(\theta)$ and $\cos(\theta)$):

$$W_{56} = -c(\theta_{16})c(\theta_{22})s(\theta_{23})c(\theta_{24}) - s(\theta_{16})c(\theta_{17})c(\theta_{23})c(\theta_{24}) + s(\theta_{16})s(\theta_{17})c(\theta_{18})s(\theta_{24}) \tag{12.3}$$

In fact, the $n \times n$ matrix W can be seen as the unitary matrix of our quantum circuit if we solely consider the unary basis, which is specified by the parameters of the quantum gates. A unitary is a complex unitary matrix, but in our case, with only real operations, the matrix is simply orthogonal. This proves the correspondence between any matrix W and the pyramidal quantum circuit.

The full unitary U_W in the Hilbert Space of our n-qubit quantum circuit is a $2^n \times 2^n$ matrix with the $n \times n$ matrix W embedded in it as a submatrix on the unary basis. This is achieved by loading the data as unary states and by using only RBS gates that keep the number of 0s and 1s constant.

For instance, as shown in Fig.12.6, a 3-qubit pyramidal circuit is described as a unique 3×3 matrix, that can be easily verified to be orthogonal.

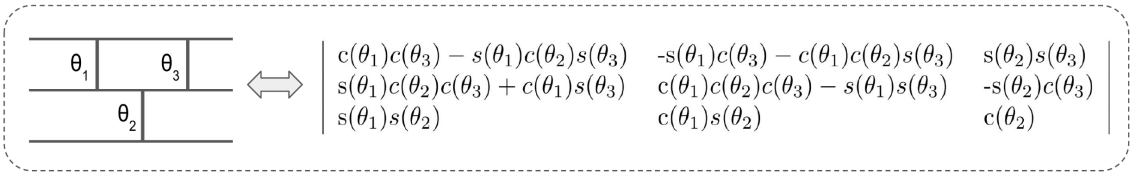


Figure 12.6: Example of a 3 qubits pyramidal circuit and the equivalent orthogonal matrix. $c(\theta)$ and $s(\theta)$ respectively stand for $\cos(\theta)$ and $\sin(\theta)$.

In Fig.12.5, we considered the case of a single unary for both the input and output. But with actual data, as seen in Section 12.1.3, input and output states are in fact a superposition of unary states. Thanks to the linearity of quantum mechanics in absence of measurements, the previous descriptions remain valid and can be applied on a linear combination of unary states.

Let's consider an input vector $x \in \mathbb{R}^n$ encoded as a quantum state $|x\rangle = \sum_{i=0}^{n-1} x_i |e_i\rangle$ where $|e_i\rangle$ represents the i^{th} unary state (see Section 12.1.3). By definition of W, each unary $|e_i\rangle$ will undergo a proper evolution $|e_i\rangle \mapsto \sum_{j=0}^{n-1} W_{ij} |e_j\rangle$. This yields, by linearity, the following mapping

$$|x\rangle \mapsto \sum_{i,j} W_{ij} x_i |e_j\rangle \tag{12.4}$$

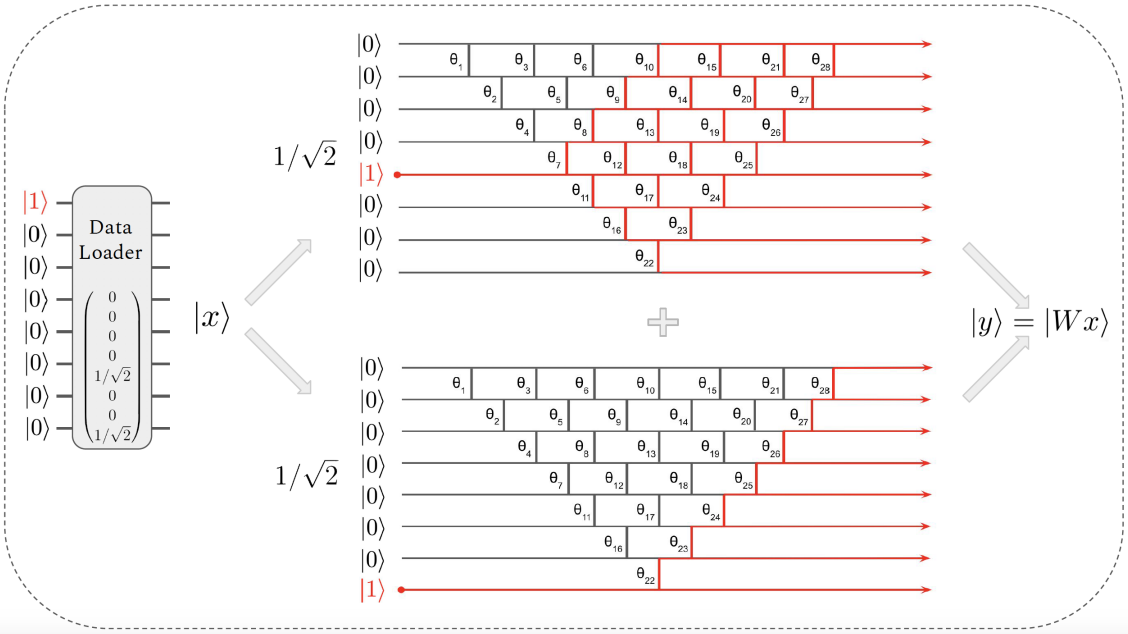


Figure 12.7: Schematic representation of a pyramidal circuit applied on a loaded vector x with two non-zero values. The output is the unary encoding of $y = Wx$ where W is the corresponding orthogonal matrix associated with the circuit.

As explained above, our quantum circuit is equivalently described by the sparse unitary $U_W \in \mathbb{R}^{2^n \times 2^n}$ or on the unary basis by the matrix $W \in \mathbb{R}^{n \times n}$. This can be summarized with

$$U_W |x\rangle = |Wx\rangle \tag{12.5}$$

We see from Eq.(12.4) and Eq.(12.5) that the output is in fact $|y\rangle$, the unary encoding of the vector $y = Wx$, which is the output of a matrix multiplication between the $n \times n$ orthogonal matrix W and the input $x \in \mathbb{R}^n$. As expected, each element of y is given by $y_k = \sum_{i=0}^{n-1} W_{ik} x_i$. See Fig.12.7 for a diagram representation of this mapping.

Therefore, for any given neural network's orthogonal layer, there is a quantum pyramidal circuit that reproduces it. On the other hand, any quantum pyramidal circuit is implementing an orthogonal layer of some sort.

As a side note, we can ask if a circuit with only $\log(n)$ qubits could also implement an orthogonal matrix multiplication of size $n \times n$. Indeed, it would be a unitary matrix in $\mathbb{R}^{n \times n}$, but since the circuit should also have $n(n-1)/2$ free parameters to tune, this would come at a cost of large depth, potentially unsuitable for NISQ devices.

Error Mitigation

It is important to notice that with our restriction to unary states, strong error mitigation techniques become available. Indeed, as we expect to obtain only quantum superposition of unary states at every layer, we can post process our measurements and discard the ones that present non unary states (i.e. states with more than one qubit in state $|1\rangle$, or the ground state). The most expected error is a bit-flip between $|1\rangle$ and $|0\rangle$. The case where two bit-flips happen at the same time, which

would change a unary state to a different unary state and would thus pass through our error mitigation, is even less probable. This error mitigation procedure can be applied efficiently to the results of a hardware demonstration and it has been used in the results presented in this Chapter.

Extracting the classical output

As shown in Fig.12.7, when using the quantum circuit, the output is a quantum state $|y\rangle = |Wx\rangle$. As often in quantum machine learning [Aar15], it is important to consider the cost of retrieving the classical outputs, using a procedure called tomography. In our case, this is even more crucial since, between each layer, the quantum output will be converted into a classical one in order to apply a non-linear function, and then reloaded for the next layer.

Retrieving the amplitudes of a quantum state comes at cost of multiple measurements, which requires running the circuit multiples times, hence adding a multiplicative overhead in the running time. A finite number of samples is also a source of approximation error in the final result. In this work, we will allow for ℓ_∞ errors [KLP20a]. The ℓ_∞ tomography on a quantum state $|y\rangle$ with unary encoding on n qubits requires $O(\log(n)/\delta^2)$ measurements, where $\delta > 0$ is the error threshold allowed. For each $j \in [n]$, $|y_j|$ will be obtained with an absolute error δ, and if $|y_j| < \delta$, it will most probably not be measured, hence set to 0. In practice, one would perform as many measurements as is convenient during the experiment, and deduce the equivalent precision δ from the number of measurements made.

Note that it is important to obtain the amplitudes of the quantum state, which in our case are positive or negative real numbers, and not just the probabilities of the outcomes, which are the squares of the amplitudes. There are different ways of obtaining the sign of the amplitudes and we present two different ways below.

Indeed, a simple measurement in the computational basis will only provide us with estimations of the probabilities that are the squares of the quantum amplitudes (see Section 4.1.1). In the case of neural networks, it is important to obtain the sign of the layer's components in order to apply certain types of non-linearities. For instance, the ReLu activation function is often used to set all negative components to 0.

In Fig.12.8, we propose a specific enhancement to our circuit to obtain the signs of the vector's components at low cost. The sign retrieval procedure consists of three parts.

a) The circuit is first applied as described above, allowing to retrieve each squared amplitude y_j^2 with precision $\delta > 0$ using the ℓ_∞ tomography (Section 4.2.1). The probability of measuring the unary state $|e_1\rangle$ (i.e. $|100...\rangle$), is $p(e_1) = y_1^2$.

b) We apply the same steps a second time on a modified circuit. It has additional *RBS* gates with angle $\pi/4$ at the end, which will mix the amplitudes pair by pair. The probabilities to measure $|e_1\rangle$ and $|e_2\rangle$ are now given by $p(e_1) = (y_1+y_2)^2$ and $p(e_2) = (y_1-y_2)^2$. Therefore if $p(e_1) > p(e_2)$, we have $sign(y_1) \neq sign(y_2)$, and if $p(e_1) < p(e_2)$, we have $sign(y_1) = sign(y_2)$. The same holds for the pairs (y_3, y_4), and so on.

c) We finally perform the same where the *RBS* are shifted by one position below. Then we compare the signs of the pairs (y_2, y_3), (y_4, y_5) and so on.

In the end, we are able to recover each value y_j with its sign, assuming that $y_1 > 0$ for instance. This procedure has the benefit of not adding depth to the original circuit but requires 3 times more runs. The overall cost of the tomography procedure with sign retrieval is given by $\widetilde{O}(n/\delta^2)$.

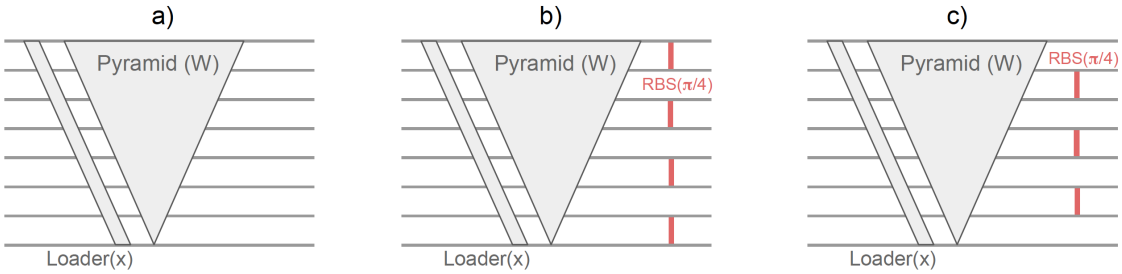


Figure 12.8: First tomography procedure to retrieve the value and the sign of each component of the resulting vector $|y\rangle = |Wx\rangle$. Circuit a) is the original one while circuits b) and c) have additional RBS gates with angle $\pi/4$ at the end to compare the signs between adjacent components. In all three cases an ℓ_∞ tomography is applied.

In Fig.12.9 we propose another method to obtain the values of the amplitudes and their signs, which is in fact what we used for the hardware demonstrations. Compared to the above procedure, it relies on one circuit only but requires an extra qubit and a depth of $3n + O(1)$ instead of $2n + O(1)$.

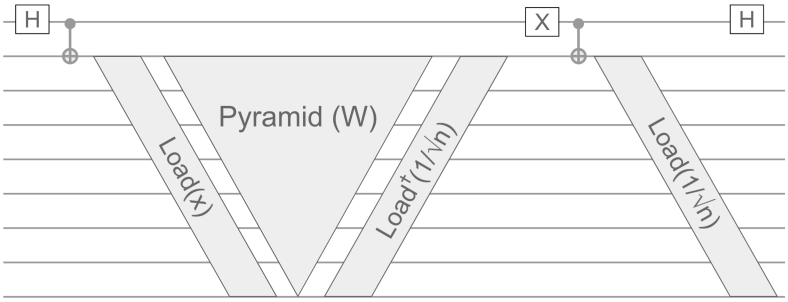


Figure 12.9: Second tomography procedure to retrieve the value and the sign of each component of the resulting vector $|y\rangle = |Wx\rangle$. For a *rectangular* case with output of size m, the two opposite loaders at the end must be on the last m qubits only, and the $CNOT$ gate between them connects the top qubits to the loader's top qubit as well.

This circuit performs a Hadamard and CNOT gate in order to initialize the qubits in the state $\frac{1}{\sqrt{2}}|0\rangle|0\rangle + \frac{1}{\sqrt{2}}|1\rangle|e_1\rangle$, where the second register corresponds to the n qubits that will be processed by the pyramidal circuit and the loaders.

Next, applying the data loader for the normalized input vector x (see Section 12.1.3) and the pyramidal circuit will, according to Eq.(12.4), map the state to

$$\frac{1}{\sqrt{2}}|0\rangle|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\sum_{j=1}^{n} W_j x\,|e_j\rangle \tag{12.6}$$

In other words, we performed the pyramid circuits controlled on the first qubit being in state $|1\rangle$. Then, we flip the fisrt qubit with an X gate and perform a

controlled loading of the uniform norm-1 vector $(\frac{1}{\sqrt{n}}, \cdots, \frac{1}{\sqrt{n}})$. For this, we add the adjoint data loader for the state, a CNOT gate and the data loader a second time. Recall that if a circuit U is followed by $U^\dagger$, it is equivalent to the identity. Therefore, this will load the uniform state only when the first qubit is in state $|1\rangle$:

$$\frac{1}{\sqrt{2}}|0\rangle \sum_{j=1}^{n} W_j x |e_j\rangle + \frac{1}{\sqrt{2}}|1\rangle \sum_{j=1}^{n} \frac{1}{\sqrt{n}} |e_j\rangle \tag{12.7}$$

Finally, a Hadamard gate will mix both parts of the amplitudes on the extra qubit to give us the desired state:

$$\frac{1}{2}|0\rangle \sum_{j=1}^{n} \left(W_j x + \frac{1}{\sqrt{n}}\right) |e_j\rangle + \frac{1}{2}|1\rangle \sum_{j=1}^{n} \left(W_j x - \frac{1}{\sqrt{n}}\right) |e_j\rangle \tag{12.8}$$

On this final state, we can see that the difference in the probabilities of measuring the extra qubit in state 0 or 1 and rest in the unary state e_j is given by $\Pr[0, e_j] - \Pr[1, e_j] = \frac{1}{4}\left(W_j x + \frac{1}{\sqrt{n}}\right)^2 - \frac{1}{4}\left(W_j x - \frac{1}{\sqrt{n}}\right)^2 = W_j x/\sqrt{n}$. Therefore, for each j, we can deduce the sign of $W_j x$ by looking at the most frequent output of the measurement of the first qubit. To deduce as well the value of $W_j x$, we simply use $\Pr[0, e_j]$ or $\Pr[1, e_j]$ depending on the sign found before. For instance, if $W_j x > 0$ we have $W_j x = 2\sqrt{\Pr[0, e_j]} - \frac{1}{\sqrt{n}}$.

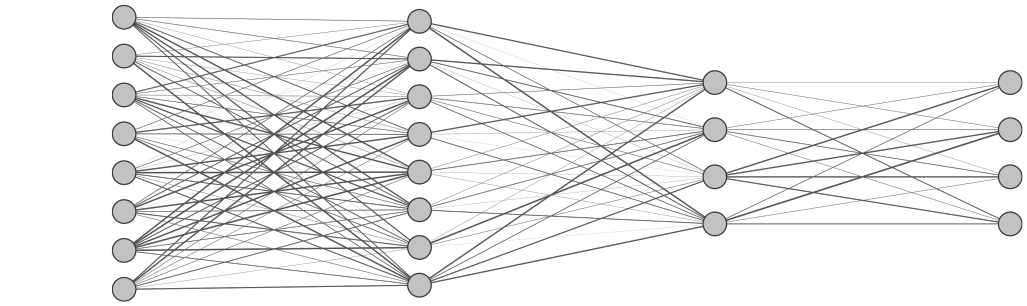
Combining with the ℓ_∞ tomography and the non linearity, the overall cost of this tomography is given by $\widetilde{O}(n/\delta^2)$ as well.

Multiple Quantum Layers

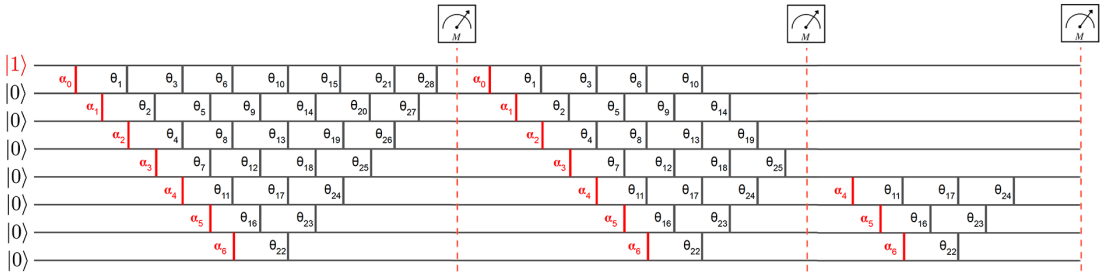
In the previous sections, we have seen how to implement a quantum circuit to perform the evolution of one orthogonal layer. In classical deep learning, such layers are stacked to gain expressivity and accuracy. Between each layer, a non-linear function is applied to the resulting vector. The presence of these non-linearities is key in the ability of the neural network to learn any function [LLPS93].

The benefit of using our quantum pyramidal circuit is the ability to simply concatenate them to mimic a multi layer neural network. After each layer, a tomography of the output state $|z\rangle$ is performed to retrieve each component, corresponding to its quantum amplitudes (see Section 12.2). A non-linear function σ is then applied classically to obtain $a = \sigma(z)$. The next layer starts with a new unary data loader (See Section 12.1.3). This hybrid scheme allows as well to keep the depth of the quantum circuits reasonable for NISQ devices, by applying the neural network layer by layer.

Note that the quantum neural networks we propose here are close to the behavior of classical neural networks and thus we can control and understand the quantum mapping and implement each layer and its non-linearities in a modular way. They are also different regarding the training strategies which are close to the classical ones but use a different optimization landscape that can provide different models (see Section 12.3 for details). It will be interesting to compare our pyramidal circuit to a quantum variational circuit with n qubits and $n(n-1)/2$ gates of any type, as we usually see in the literature. Using such circuits we would explore among all possible $2^n \times 2^n$ matrices instead of $n \times n$ classical orthogonal matrices, but so far there's no theoretical ground to explain why this should provide an advantage.



(a)



(b)

Figure 12.10: A full neural network with layers [8,8,4,4]. (a) Classical representation. (b) The equivalent quantum circuit is a concatenation of multiple pyramidal circuits. Between each layer one performs a measurement and applies a non-linearity. Each layer starts with a new unary data loader.

As an open outlook, one could imagine incorporating additional entangling gates after each pyramid layer (composed, for instance, of $CNOT$ or CZ). This would mark a step out of the unary basis and could effectively allow exploring more interactions in the Hilbert Space.

Classical implementation While we presented the quantum pyramidal circuit as the inspiration of the new methods for orthogonal neural networks, it is not hard to see that these quantum circuits can be simulated classical with a small overhead, thus yielding classical methods for orthogonal neural networks.

This classical algorithm is simply the simulation of the quantum pyramidal circuit, where each RBS gate is replaced by a planar rotation between its two inputs.

As shown in Fig.12.11, we propose a similar classical pyramidal circuit, where each layer is constituted of $\frac{n(n-1)}{2}$ planar rotations, for a total of $4 \times \frac{n(n-1)}{2} = O(n^2)$ basic operations. Therefore our single layer feedforward pass has the same complexity $O(n^2)$ as the usual matrix multiplication.

One may still have an advantage in performing the quantum circuit for inference, since the quantum circuit has depth $O(n)$, instead of the $O(n^2)$ classical complexity of the matrix-vector multiplication. In addition, as we will see below, the main advantage of our method is that we can also now train orthogonal weight matrices classically in time $O(n^2)$, instead of the previously best-known $O(n^3)$.

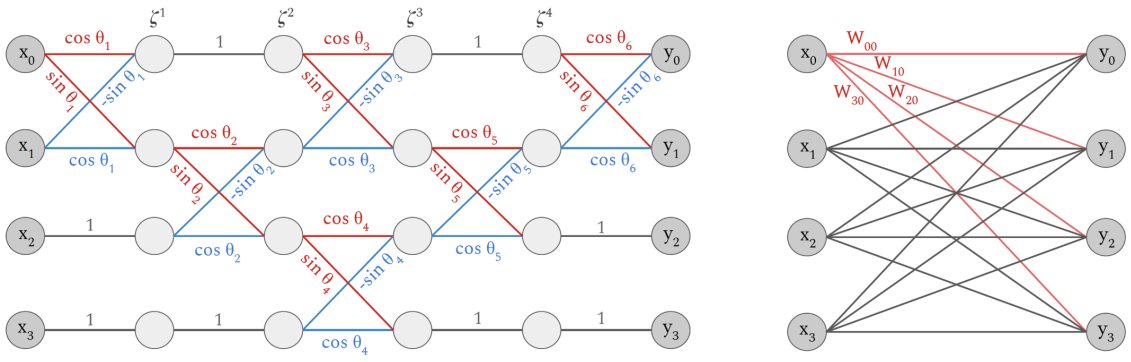


Figure 12.11: Classical representation of a single orthogonal layer on a 4x4 case (n=4) performing $x \mapsto y = Wx$. The angles and the weights can be chosen such that our classical pyramidal circuit (left) and normal classical network (right) are equivalent. Each connecting line represents a scalar multiplication with the value indicated. On the classical pyramidal circuit (left), *inner layers* ζ^λ are displayed. A *timestep* corresponds to the lines in between two *inner layers* (see Section 12.3 for definitions).

12.3 Backpropagation for the Orthogonal Neural Network Circuit

Classical Backpropagation Algorithm

Basic introduction and notation to the backpropagation in fully connected neural networks are given in Section 2.3.2. We recall some facts and notations that will be useful for the following;

The backpropagation in a fully connected neural network is a well known and efficient procedure to update the weight matrix at each layer [HN92, Roj96]. At layer ℓ, we denote its weight matrices by W^ℓ and biases by b^ℓ. Each layer is followed by a non-linear function σ, and can therefore be written as

$$a^\ell = \sigma(W^\ell \cdot a^{\ell-1} + b^\ell) = \sigma(z^\ell) \tag{12.9}$$

After the last layer, one can define a cost function $\mathcal{C}$ that compares the output to the ground truth. The goal is to calculate the gradient of $\mathcal{C}$ with respect to each weight and bias, namely $\frac{\partial \mathcal{C}}{\partial W^\ell}$ and $\frac{\partial \mathcal{C}}{\partial b^\ell}$. In the backpropagation, we start by calculating these gradients for the last layer, then propagate back to the first layer.

We will require to obtain the *error* vector at layer ℓ defined by $\Delta^\ell = \frac{\partial \mathcal{C}}{\partial z^\ell}$. One can show the backward recursive relation $\Delta^\ell = (W^{\ell+1})^T \cdot \Delta^{\ell+1} \odot \sigma'(z^\ell)$, where $\odot$ symbolizes the Hadamard product, or entry-wise multiplication. Note that the previous computation requires simply to apply the layer (i.e. apply matrix multiplication) in reverse. We can then show that each element of the weight gradient matrix at layer ℓ is given by $\frac{\partial \mathcal{C}}{\partial W^\ell_{jk}} = \Delta^\ell_j \cdot a^{\ell-1}_1$. Similarly, the gradient with respect to the biases is easily defined as $\frac{\partial \mathcal{C}}{\partial b^\ell_j} = \Delta^\ell_j$.

Once these gradients are computed, we update the parameters using the gradient descent rule, with learning rate λ:

$$W^\ell_{jk} \leftarrow W^\ell_{jk} - \lambda \frac{\partial \mathcal{C}}{\partial W^\ell_{jk}} \quad ; \quad b^\ell_j \leftarrow b^\ell_j - \lambda \frac{\partial \mathcal{C}}{\partial b^\ell_j} \tag{12.10}$$

Backpropagation for Pyramidal Circuits

Looking through the prism of our pyramidal quantum circuit, the parameters to update are no longer the individual elements of the weight matrices directly, but the angles of the RBS gates that give rise to these matrices. Thus, we need to adapt the backpropagation method to our setting based on the angles. We will start by introducing some notation for a single layer ℓ, which will not be explicit in the notation for simplicity. We assume we have as many output bits as input bits, but this can easily be extended to the *rectangular* case.

We first introduce the notion of *timesteps* inside each layer, which correspond to the computational steps in the pyramidal structure of the circuit (see Fig.12.12). It is easy to show that for n inputs, there will be $2n-3$ such *timesteps*, each one indexed by an integer $\lambda \in [0, \cdots, \lambda_{max}]$. Applying a timestep consists in applying the matrix w^λ, made of all the RBS gates aligned vertically at this timestep (w^λ is the unitary in the unary basis, see Section 12.2 for details). Each time a timestep is applied, the resulting state is a vector in the unary basis named *inner layer* and denoted by ζ^λ. This evolution can be written as $\zeta^{\lambda+1} = w^\lambda \cdot \zeta^\lambda$. We use this notation similar to the real layer ℓ, with the weight matrix W^ℓ and the resulting vector z^ℓ (see Section 12.3).

In fact we have the correspondences $\zeta^0 = a^{\ell-1}$ for the first *inner layer*, which is the input of the actual layer, and $z^\ell = w^{\lambda_{max}} \cdot \zeta^{\lambda_{max}}$ for the last one. We also have $W^\ell = w^{\lambda_{max}} \cdots w^1 w^0$. We use the same kind of notation for the backpropagation

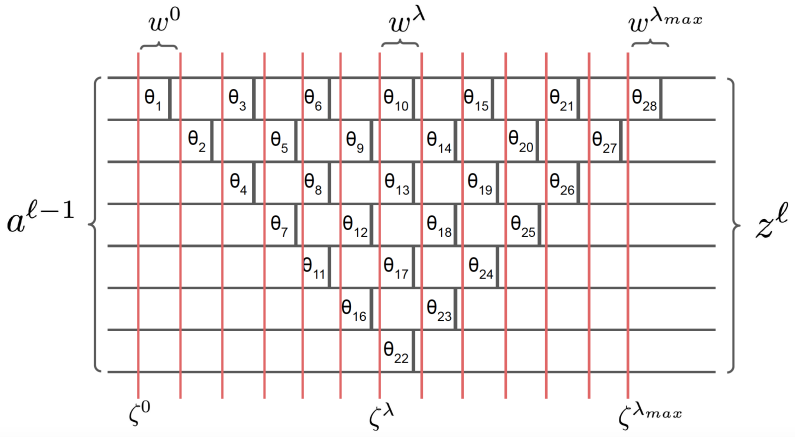


Figure 12.12: Quantum circuit for one neural network layer divided into *timesteps* (red vertical lines) $\lambda \in [0, \cdots, \lambda_{max}]$. Each timestep corresponds to an *inner layer* ζ^λ and an *inner error* δ^λ. The part of the circuit between two timesteps is an unitary matrix w^λ in the unary basis.

errors. At each timestep λ we define an *inner error* $\delta^\lambda = \frac{\partial \mathcal{C}}{\partial \zeta^\lambda}$. This definition is similar to the layer error $\Delta^\ell = \frac{\partial \mathcal{C}}{\partial z^\ell}$. In fact we will use the same backpropagation formulas, without non-linearities, to retrieve each *inner error* vector $\delta^\lambda = (w^\lambda)^T \cdot \delta^{\lambda+1}$. In particular, for the last timestep, the first to be calculated, we have $\delta^{\lambda_{max}} = (w^{\lambda_{max}})^T \cdot \Delta^\ell$. Finally, we can retrieve the error at the previous layer $\ell - 1$ using the correspondence $\Delta^{\ell-1} = \delta^0 \odot \sigma'(z^\ell)$.

The reason for this breakdown into timesteps is the ability to efficiently obtain the gradient with respect to each angle. Let's consider the timestep λ and one of its gate with angle denoted by θ_i acting on qubits i and $i+1$ (note that the numbering is

different from Fig.12.12). We will decompose the gradient $\frac{\partial \mathcal{C}}{\partial \theta_i}$ using each component, indexed by the integer k, of the *inner layer* and *inner error* vectors:

$$\frac{\partial \mathcal{C}}{\partial \theta_i} = \sum_k \frac{\partial \mathcal{C}}{\partial \zeta_k^{\lambda+1}} \frac{\partial \zeta_k^{\lambda+1}}{\partial \theta_i} = \sum_k \delta_k^{\lambda+1} \frac{\partial (w_k^\lambda \cdot \zeta^\lambda)}{\partial \theta_i} \tag{12.11}$$

Where w_k^λ is the kth row of matrix w^λ. Since timestep λ is only composed of separated RBS gates, the matrix w^λ consists of diagonally arranged 2x2 block submatrices given in Eq.(12.1). Only one of these submatrices depends on the angle θ_i considered here, at the position i and $i+1$ in the matrix. We can thus rewrite the above gradient as $\frac{\partial \mathcal{C}}{\partial \theta_i} = \delta_i^{\lambda+1} \frac{\partial}{\partial \theta_i} \left(w_i^\lambda \cdot \zeta^\lambda \right) + \delta_{i+1}^{\lambda+1} \frac{\partial}{\partial \theta_i} \left(w_{i+1}^\lambda \cdot \zeta^\lambda \right)$, or:

$$\frac{\partial \mathcal{C}}{\partial \theta_i} = \delta_i^{\lambda+1} \frac{\partial}{\partial \theta_i} \left(\cos(\theta_i)\zeta_i^\lambda + \sin(\theta_i)\zeta_{i+1}^\lambda \right) + \delta_{i+1}^{\lambda+1} \frac{\partial}{\partial \theta_i} \left(-\sin(\theta_i)\zeta_i^\lambda + \cos(\theta_i)\zeta_{i+1}^\lambda \right) \tag{12.12}$$

$$\frac{\partial \mathcal{C}}{\partial \theta_i} = \delta_i^{\lambda+1}(-\sin(\theta_i)\zeta_i^\lambda + \cos(\theta_i)\zeta_{i+1}^\lambda) + \delta_{i+1}^{\lambda+1}(-\cos(\theta_i)\zeta_i^\lambda - \sin(\theta_i)\zeta_{i+1}^\lambda) \tag{12.13}$$

Therefore we have shown a way to compute each angle gradient: During the feedforward pass, one must apply sequentially each of the $2n-3 = O(n)$ timesteps, and store the resulting vectors, the *inner layers* ζ^λ. During the backpropagation, one obtains the *inner errors* δ^λ by applying the timesteps in reverse. One can finally use a gradient descent on each angle θ_i, while preserving the orthogonality of the overall equivalent weight matrix $\theta_i^\ell \leftarrow \theta_i^\ell - \lambda \frac{\partial \mathcal{C}}{\partial \theta_i^\ell}$. Since the optimization is performed in the angle landscape, and not on the equivalent weight landscape, it can potentially be different and produce different models. We leave open the study of the properties of both landscapes.

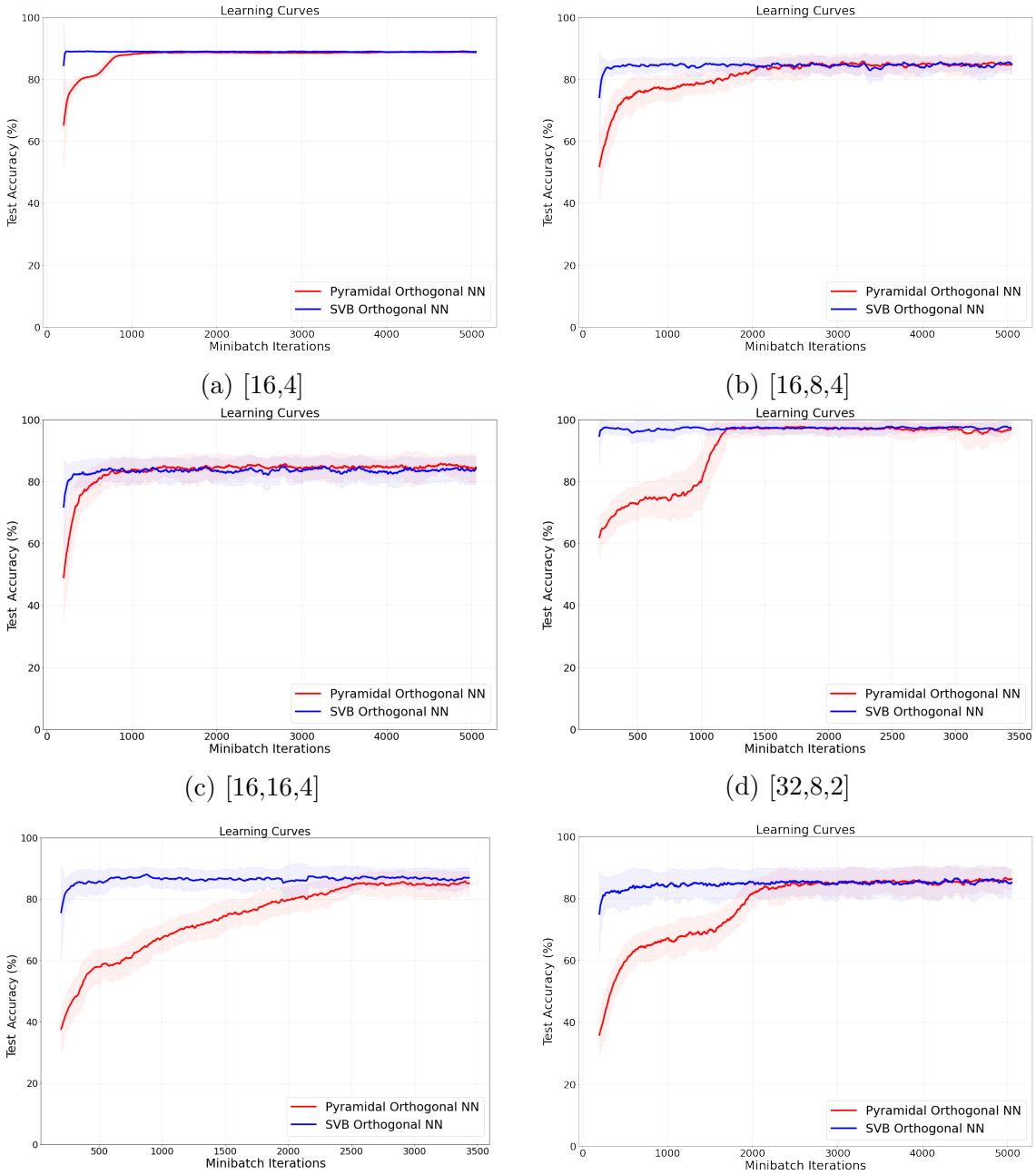
As one can see from the above description, this is in fact a classical algorithm to obtain the angle's gradients, which allows us to train our OrthoNN efficiently classically while preserving the strict orthogonality. To obtain the angle's gradient, one needs to store the $2n$-3 *inner layers* ζ^λ during the feedforward pass. Next, given the error at the following layer, we perform a backward loop on each *timestep* (see Fig.12.11). At each *timestep*, we obtain the gradient for each angle parameter, by simply applying Eq.(12.13). This requires $O(1)$ operations for each angle. Since there are at most $n/2$ angles per *timesteps*, estimating gradients has a complexity of $O(n^2)$. After each *timestep*, the next *inner error* $\delta^{\lambda-1}$ is computed as well, using at most $4n/2$ operations.

In the end, our classical algorithm allows us to compute the gradients of the $n(n-1)/2$ angles in time $O(n^2)$, thus performing a gradient descent respecting the strict orthogonality of the weight matrix in the same time. This is considerably faster than previous methods based on Singular Value Decomposition methods and provides a training method that is asymptotically as fast as for normal neural networks, while providing the extra property of orthogonality.

12.4 Numerical Experiments

In [KLM21], we performed basic numerical experiments to verify the abilities of our pyramidal circuit, on the standard MNIST dataset. Note that current quantum

hardware and software are not yet suited for bigger experiments. We first compared the training of our Classical OrthoNN to the SVB algorithm from [JLW+19] (see Section 2.3.3). Results as reported in Fig.12.13. These small scale tests confirmed that the pyramidal circuits and the corresponding gradient descent on the angles were efficient for learning a classification task.



(a) [16,4]

(b) [16,8,4]

(c) [16,16,4]

(d) [32,8,2]

(e) [32,16,4]

(f) [32,32,4]

Figure 12.13: Training comparison between the SVB OrthoNN from [JLW+19] and our classical pyramidal OrthoNN. Test accuracy on 1000 samples during several epochs of training on the MNIST dataset on 5000 samples. Initial dimensionality reduction (PCA) was on the samples to fit the input layer of the networks. Shaded areas indicate the accuracy variance during minibatch updates of size 50.

Then, we implemented the quantum circuit on a real quantum computer provided by IBM. We used a 16 and 5 qubits device to perform respectively a [8,2] and a [4,2] orthogonal layer. We also branched two layers to perform a [4,4,2] network with non-linearity. A pyramidal OrthoNN was trained classically, and the resulting angles were transferred to test the quantum circuit on a classification task on classes 6 and 9 of the MNIST dataset, over 500 samples. We compared the real experiment with a simulated one, and the classical pyramidal circuit as well. Results are reported in Table 12.2.

Network Architecture	Inference Accuracy		
	Classical Pyramidal Circuit	IBM Simulator	IBM Quantum Computer
$[4, 2]$	98,4%	98,4%	98,0%
$[8, 2]$	97,4%	97,4%	95,0%
$[4, 4, 2]$	98,2%	98,2%	82,8%

Table 12.2: Results of the Pyramidal OrthoNN on classical simulators and real quantum computers. *ibmq_bogota v1.4.32* and *ibmq_guadalupe v1.2.17* are respectively 5 and 16 qubits quantum computers. May 2021.

Finally, in [MLL+21], we performed a benchmark on the MedMNIST dataset for medical imaging. We compared quantum and classical methods for orthogonal neural networks to classify diseases in two datasets: a classification of Chest X-Rays (Pneumonia) and a classification of retina images for retinopathy detection (Retina).

In Fig.12.14 we show our results, where we provide the AUC (area under curve) and ACC (accuracy) for all different types of neural network experiments, for both the training and test sets, for the Pneumonia and Retina datasets. In Fig.12.15 we show similar results for non orthogonal quantum neural networks, also called Quantum assisted Neural Network, that are based on simple inner products using data loaders as explained in Section 6.2.

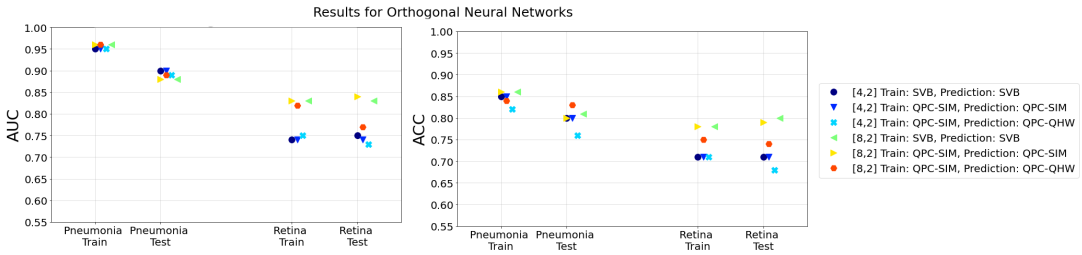


Figure 12.14: Results of experiments for the Orthogonal Neural Network. QPC stands for Quantum Pyramidal Circuit and is the classical algorithm simulating our quantum circuit. QHW is the quantum circuit on the real quantum hardware. SVB stands for the classical Singular Value Bounded algorithm.

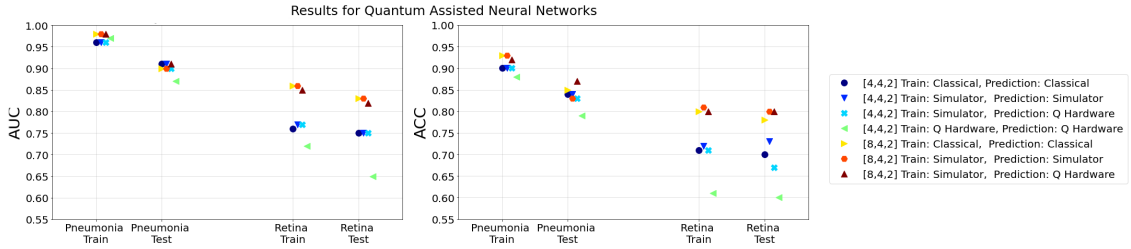


Figure 12.15: Results of experiments for the Quantum assisted Neural Network.

Conclusion and Outlook

In this chapter, we have proposed for the first time training methods for orthogonal neural networks (OrthoNNs) that run in quadratic time, a significant improvement from previous methods based on Singular Value Decomposition. The main idea of our method is to replace usual weights and orthogonal matrices with an equivalent pyramidal circuit made of two-dimensional rotations. Each rotation is parametrizable by an angle, and the gradient descent takes place in the angle's optimization landscape. This unique type of gradient backpropagation ensures perfect orthogonality of the weight matrices while substantially improving the running time compared to previous work. Moreover, we propose both classical and quantum methods for inference, where the forward pass on a near term quantum computer would provide a provable advantage in the running time. This work expands the field of quantum deep learning by introducing new tools, concepts, and equivalences with classical deep learning theory. We have highlighted open questions regarding the construction of such pyramidal circuits for neural networks and their potential new advantages in terms of execution time, accuracy, and learning properties.

Conclusion

"Aux murs de nos laboratoires
les cadrans lumineux remplacent
les ombres de la caverne."

Arthur Koestler
Les Somnambules (1959)

In this thesis, we have proven the existence of new quantum algorithms for machine learning applications in unsupervised learning (k-means, spectral clustering) and in deep learning (convolutional and orthogonal neural networks).

During the development of these algorithms, we have introduced new quantum subroutines for linear algebra, distance estimation, graph analysis, and tomography of quantum states. These subroutines are fundamental enough to be reused across all machine learning and hopefully in other fields as well.

Our algorithms are provably faster, using complexity analysis, often with respect to the number of points in the dataset, or in their dimension. These running times should be interpreted with subtlety as they depend on counter intuitive parameters, that often depends on the data values themselves. This behaviour has no classical counterpart and can help us determine where quantum computing can provide an advantage in machine learning. To go further, we have performed extensive classical simulations in order to test the training abilities of our quantum solutions in practice, as well as the scaling of their running times. For quantum orthogonal neural networks, we even implemented real quantum circuits on 8 and 16 qubits quantum computers.

In the end, we can expect quantum machine learning to be the most advantageous for hard linear algebra problems, involving spectral analysis (eigenvalue decomposition or projection) for which classical complexity are cubic or more. This should also motivate us to look at new problems that are even harder, in different fields such as topological data analysis and graph problems. On the other hand, it will be interesting to see what quantum can offer for machine learning problems that don't involve data, such as reinforcement learning, approximately solving partial differential equations, and even chemistry or many-body physics.

Quantum machine learning is definitely an interesting field, and there is still a lot of work to be done to bring these algorithms closer to what can be done in practice. Many efforts are also expected in the physical realization of qubits quality, error correction, and quantum access to data. We are looking forward to the arrival of the first large scale quantum computers to test our theories in practice and hope they will help people solve problems for the general interest.

Bibliography

[A+19] Héctor Abraham et al. Qiskit: An open-source framework for quantum computing, 2019.

[AAA+19] Kazunori Akiyama, Antxon Alberdi, Walter Alef, Keiichi Asada, Rebecca Azulay, Anne-Kathrin Baczko, David Ball, Mislav Baloković, John Barrett, Dan Bintley, et al. First m87 event horizon telescope results. iv. imaging the central supermassive black hole. *The Astrophysical Journal Letters*, 875(1):L4, 2019.

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[Aar13] Scott Aaronson. Why philosophers should care about computational complexity. *Computability: Turing, Gödel, Church, and Beyond*, 261:327, 2013.

[Aar15] Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291–293, 2015.

[ABG13] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Quantum speed-up for unsupervised learning. *Machine Learning*, 90(2):261–287, 2013.

[ADBL19] Juan Miguel Arrazola, Alain Delgado, Bhaskar Roy Bardhan, and Seth Lloyd. Quantum-inspired algorithms in practice. *arXiv preprint arXiv:1905.10415*, 2019.

[ADR82] Alain Aspect, Jean Dalibard, and Gérard Roger. Experimental test of bell's inequalities using time-varying analyzers. *Physical review letters*, 49(25):1804, 1982.

[AdW20] Simon Apers and Ronald de Wolf. Quantum speedup for graph sparsification, cut approximation and laplacian solving. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 637–648. IEEE, 2020.

[AGJO+15] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 17(12):123010, 2015.

[AHKZ20] Jonathan Allcock, Chang-Yu Hsieh, Iordanis Kerenidis, and Shengyu Zhang. Quantum algorithms for feedforward neural networks. *ACM Transactions on Quantum Computing*, 1(1):1–24, 2020.

[AM01] Dimitris Achlioptas and Frank McSherry. Fast computation of low rank matrix approximations. In *Proceedings of the 33rd Annual Symposium on Theory of Computing*, pages 611–618, 2001.

[Amb12] Andris Ambainis. 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, 2012.

[AR20] Scott Aaronson and Patrick Rall. Quantum approximate counting, simplified. In *Symposium on Simplicity in Algorithms*, pages 24–32. SIAM, 2020.

[ASZ+20] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *arXiv preprint arXiv:2011.00027*, 2020.

[AV06] David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153. ACM, 2006.

[AV07] David Arthur and Sergei Vassilvitskii. 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, 2007.

[BBC+93] Charles H Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical review letters*, 70(13):1895, 1993.

[BBF+20] Kerstin Beer, Dmytro Bondarenko, Terry Farrelly, Tobias J Osborne, Robert Salzmann, Daniel Scheiermann, and Ramona Wolf. Training deep quantum neural networks. *Nature communications*, 11(1):1–6, 2020.

[BCC+16] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry Jackel, Urs Muller, and Karol Zieba. Visualbackprop: efficient visualization of cnns. *arXiv preprint arXiv:1611.05418*, 2016.

[BCLK+21] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S Kottmann, Tim Menke, et al. Noisy intermediate-scale quantum (nisq) algorithms. *arXiv preprint arXiv:2101.08448*, 2021.

[BCW18] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep cnns? *arXiv preprint arXiv:1810.09102*, 2018.

[BDD+00] Zhaojun Bai, James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst. *Templates for the solution of algebraic eigenvalue problems: a practical guide*. SIAM, Philadelphia, PA, USA, 2000.

[BFL21] Noah Berner, Vincent Fortuin, and Jonas Landman. Quantum bayesian neural networks. 2021.

[BH97] Gilles Brassard and Peter Hoyer. An exact quantum polynomial-time algorithm for simon's problem. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*, pages 12–23. IEEE, 1997.

[BHMT02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[Bia21] Jacob Biamonte. Universal variational quantum computation. *Physical Review A*, 103(3):L030401, 2021.

[Bis95] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[BKG15] Christos Boutsidis, Prabhanjan Kambadur, and Alex Gittens. Spectral clustering via the power method-provably. In *International conference on machine learning*, pages 40–48, 2015.

[BS17] Fernando GSL Brandao and Krysta M Svore. Quantum speed-ups for solving semidefinite programs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 415–426. IEEE, 2017.

[BV97] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997.

[BWP+17] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[CAB+20] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *arXiv preprint arXiv:2012.09265*, 2020.

[Cal20] Ewen Callaway. 'it will change everything': Deepmind's ai makes gigantic leap in solving protein structures. *Nature*, 2020.

[CB18] John A Cortese and Timothy M Braje. Loading classical data into a quantum computer. *arXiv preprint arXiv:1803.01958*, 2018.

[CCL19] Iris Cong, Soonwon Choi, and Mikhail D Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, 2019.

[CD15] Iris Cong and Luming Duan. Quantum discriminant analysis for dimensionality reduction and classification. *arXiv preprint arXiv:1510.00113*, 2015.

[CDKK20] Brian Coyle, Mina Doosti, Elham Kashefi, and Niraj Kumar. Variational quantum cloning: Improving practicality for quantum cryptanalysis. *arXiv preprint arXiv:2012.11424*, 2020.

[CFS+16] Shawn X. Cui, Michael H. Freedman, Or Sattath, Richard Stong, and Greg Minton. Quantum max-flow/min-cut. *Journal of Mathematical Physics*, 57(6):062206, 6 2016.

[CGAG17] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Quantum neuron: an elementary building block for machine learning on quantum computers. *arXiv preprint arXiv:1711.11240*, 2017.

[CGJ18] Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The power of block-encoded matrix powers: improved regression techniques via faster hamiltonian simulation. *46th International Colloquium on Automata, Languages, and Programming*, pages Art. No. 33, 14., 2018.

[Chi10] Andrew M Childs. On the relationship between continuous-and discrete-time quantum walk. *Communications in Mathematical Physics*, 294(2):581–603, 2010.

[Chi17] Andrew M Childs. Lecture notes on quantum algorithms. *Lecture notes at University of Maryland*, 2017.

[CJK+13] Anna Choromanska, Tony Jebara, Hyungtae Kim, Mahesh Mohan, and Claire Monteleoni. Fast spectral clustering via the nyström method. In *International Conference on Algorithmic Learning Theory*, pages 367–381. Springer, New York, NY, USA, 2013.

[CKS17] Andrew M Childs, Robin Kothari, and Rolando D Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46(6):1920–1950, 2017.

[CMDK20] Brian Coyle, Daniel Mills, Vincent Danos, and Elham Kashefi. The born supremacy: quantum advantage and training of an ising born machine. *npj Quantum Information*, 6(1):1–11, 2020.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[Das17] Ammar Daskin. Quantum spectral clustering through a biased phase estimation algorithm. *TWMS Journal of Applied and Engineering Mathematics*, 10(1):24–33, 2017.

[DB18] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81(7):074001, 2018.

[DDK18] Pierre-Luc Dallaire-Demers and Nathan Killoran. Quantum generative adversarial networks. *Physical Review A*, 98(1):012324, 2018.

[DFK+04] Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V Vinay. Clustering large graphs via the singular value decomposition. *Machine learning*, 56(1-3):9–33, 2004.

[DGV12] Ross Dorner, John Goold, and Vlatko Vedral. Towards quantum simulations of biological information flow. *Interface focus*, 2(4):522–528, 2012.

[DH96] Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*, 1996.

[DJ92] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[DKR02] Petros Drineas, Iordanis Kerenidis, and Prabhakar Raghavan. Competitive recommendation systems. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 82–90. ACM, 2002.

[DMGM20] Olivia Di Matteo, Vlad Gheorghiu, and Michele Mosca. Fault-tolerant resource estimation of quantum random-access memories. *IEEE Transactions on Quantum Engineering*, 1:1–13, 2020.

[dPNdSdOL19] Fernando M de Paula Neto, Adenilton J da Silva, Wilson R de Oliveira, and Teresa B Ludermir. Quantum probabilistic associative memory architecture. *Neurocomputing*, 351:101–110, 2019.

[DW19] Ronald De Wolf. Quantum computing: Lecture notes. *arXiv preprint arXiv:1907.09415*, 2019.

[Fey82] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[Fey87] Richard P Feynman. Tiny computers obeying quantum mechanical laws. *N. Metropolis, DM Kerr, and G. Rota, editors, New Directions in Physics: The Los Alamos 40th Anniversary Volume*, pages 7–25, 1987.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

[FKV04] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

[FN18] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GBH18] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. *arXiv preprint arXiv:1805.09112*, 2018.

[GGZW21] Dmitry Grinko, Julien Gacon, Christa Zoufal, and Stefan Woerner. Iterative quantum amplitude estimation. *npj Quantum Information*, 7(1):1–6, 2021.

[GH18] Daniel George and EA Huerta. Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced ligo data. *Physics Letters B*, 778:64–70, 2018.

[GLM08a] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Physical Review A*, 78(5):052310, 2008.

[GLM08b] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical review letters*, 100(16):160501, 2008.

[GLT18] András Gilyén, Seth Lloyd, and Ewin Tang. Quantum-inspired low-rank stochastic regression with logarithmic dependence on the dimension. *arXiv preprint arXiv:1811.04909*, 2018.

[GPAM+14] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[Gro05] Lov K Grover. Fixed-point quantum search. *Physical Review Letters*, 95(15):150501, 2005.

[GSC00] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[GSLW19] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. 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, 2019.

[GTKL+20] Tudor Giurgica-Tiron, Iordanis Kerenidis, Farrokh Labib, Anupam Prakash, and William Zeng. Low depth algorithms for quantum amplitude estimation. *arXiv preprint arXiv:2012.03348*, 2020.

[GWOB19] Edward Grant, Leonard Wossnig, Mateusz Ostaszewski, and Marcello Benedetti. An initialization strategy for addressing barren plateaus in parametrized quantum circuits. *Quantum*, 3:214, 2019.

[HAY+20] Zoë Holmes, Andrew Arrasmith, Bin Yan, Patrick J Coles, Andreas Albrecht, and Andrew T Sornborger. Barren plateaus preclude learning scramblers. *arXiv preprint arXiv:2009.14808*, 2020.

[HD15] Greg Hamerly and Jonathan Drake. Accelerating lloyd's algorithm for k-means clustering. In *Partitional clustering algorithms*, pages 41–78. Springer, New York, NY, USA, 2015.

[HHL09] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

[HLGJ21] Connor T Hann, Gideon Lee, SM Girvin, and Liang Jiang. Resilience of quantum random access memory to generic noise. *PRX Quantum*, 2(2):020311, 2021.

[HN92] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[HRS18] Thomas Häner, Martin Roetteler, and Krysta M Svore. Optimizing quantum circuits for arithmetic. *arXiv preprint arXiv:1805.12445*, 2018.

[HSG+21] Xiaosheng Huang, Christopher Storfer, A Gu, V Ravi, A Pilon, W Sheu, R Venguswamy, S Banka, A Dey, M Landriau, et al. Discovering new strong gravitational lenses in the desi legacy imaging surveys. *The Astrophysical Journal*, 909(1):27, 2021.

[JDM+20] Sonika Johri, Shantanu Debnath, Avinash Mocherla, Alexandros Singh, Anupam Prakash, Jungsang Kim, and Iordanis Kerenidis.

Nearest centroid classification on a trapped ion quantum computer. *arXiv preprint arXiv:2012.04145*, 2020.

[JLW+19] Kui Jia, Shuai Li, Yuxin Wen, Tongliang Liu, and Dacheng Tao. Orthogonal deep neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[KH09] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[Kit95] A Yu Kitaev. Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*, 1995.

[KL97] Emanuel Knill and Raymond Laflamme. Theory of quantum error-correcting codes. *Physical Review A*, 55(2):900, 1997.

[KL20] Iordanis Kerenidis and Alessandro Luongo. Classification of the MNIST data set with quantum slow feature analysis. *Physical Review A*, 101(6):062327, 2020.

[KL21] Iordanis Kerenidis and Jonas Landman. Quantum spectral clustering. *Physical Review A*, 103(4):042415, 2021.

[KLLP19] Iordanis Kerenidis, Jonas Landman, Alessandro Luongo, and Anupam Prakash. q-means: A quantum algorithm for unsupervised machine learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[KLM+07] Phillip Kaye, Raymond Laflamme, Michele Mosca, et al. *An introduction to quantum computing*. Oxford University Press on Demand, 2007.

[KLM21] Iordanis Kerenidis, Jonas Landman, and Natansh Mathur. Classical and quantum algorithms for orthogonal neural networks. 2021.

[KLP20a] Iordanis Kerenidis, Jonas Landman, and Anupam Prakash. Quantum algorithms for deep convolutional neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[KLP20b] Iordanis Kerenidis, Alessandro Luongo, and Anupam Prakash. Quantum expectation-maximization for gaussian mixture models. In *International Conference on Machine Learning*, pages 5187–5197. PMLR, 2020.

[KP16] Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. *arXiv preprint arXiv:1603.08675*, 2016.

[KP20a] Iordanis Kerenidis and Anupam Prakash. Quantum gradient descent for linear systems and least squares. *Physical Review A*, 101(2):022316, 2020.

[KP20b] Iordanis Kerenidis and Anupam Prakash. A quantum interior point method for LPs and SDPs. *ACM Transactions on Quantum Computing*, 1(1):1–32, 2020.

[KPS21] Iordanis Kerenidis, Anupam Prakash, and Dániel Szilágyi. Quantum algorithms for second-order cone programming and support vector machines. *Quantum*, 5:427, 2021.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LCMR19] Mario Lezcano-Casado and David Martınez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*, pages 3794–3803. PMLR, 2019.

[LGZ16] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. Quantum algorithms for topological and geometric analysis of data. *Nature communications*, 7(1):1–7, 2016.

[LJL19] Yonghae Lee, Jaewoo Joo, and Soojoon Lee. Hybrid quantum linear equation algorithm and its experimental test on ibm quantum experience. *Scientific reports*, 9(1):1–12, 2019.

[LLKL11] Mu Li, Xiao-Chen Lian, James T Kwok, and Bao-Liang Lu. Time and space efficient spectral clustering via column sampling. In *CVPR 2011*, pages 2297–2304. IEEE, Colorado Springs, CO, USA, 2011.

[Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[LLPS93] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[LMR13] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv*, 1307.0411:1–11, 7 2013.

[LMR14] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014.

[LW18] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Physical review letters*, 121(4):040502, 2018.

[MBS+18] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature communications*, 9(1):1–6, 2018.

[MLL+21] Natansh Mathur, Jonas Landman, Yun Yvonna Li, Martin Strahm, Skander Kazdaghli, Anupam Prakash, and Iordanis Kerenidis. Medical image classification via quantum neural networks. *arXiv preprint arXiv:2109.01831*, 2021.

[MLM17] Dominic J Moylett, Noah Linden, and Ashley Montanaro. Quantum speedup of the traveling-salesman problem for bounded-degree graphs. *Physical Review A*, 95(3):032323, 2017.

[MNKF18] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

[NC02] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[NJW02] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[NM14] Yoshifumi Nakata and Mio Murao. Diagonal quantum circuits: their computational power and applications. *The European Physical Journal Plus*, 129(7):1–14, 2014.

[OMA+17] JS Otterbach, R Manenti, N Alidoust, A Bestwick, M Block, B Bloom, S Caldwell, N Didier, E Schuyler Fried, S Hong, et al. Unsupervised machine learning on a hybrid quantum computer. *arXiv preprint arXiv:1712.05771*, 2017.

[PCW+20] Arthur Pesah, M Cerezo, Samson Wang, Tyler Volkoff, Andrew T Sornborger, and Patrick J Coles. Absence of barren plateaus in quantum convolutional neural networks. *arXiv preprint arXiv:2011.02966*, 2020.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[PMS+14] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):1–7, 2014.

[PPR19] Daniel K Park, Francesco Petruccione, and June-Koo Kevin Rhee. Circuit-based quantum random access memory for classical data. *Scientific reports*, 9(1):1–8, 2019.

[Pra14] Anupam Prakash. *Quantum algorithms for linear algebra and machine learning.* PhD thesis, UC Berkeley, 2014.

[Pre18] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[PSCLGFL20] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, 2020.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PZ21] Feng Pan and Pan Zhang. Simulating the sycamore quantum supremacy circuits. *arXiv preprint arXiv:2103.03074*, 2021.

[QSW+20] Bin Qian, Jie Su, Zhenyu Wen, Devki Nandan Jha, Yinhao Li, Yu Guan, Deepak Puthal, Philip James, Renyu Yang, Albert Y Zomaya, et al. Orchestrating the development lifecycle of machine learning-based iot applications: A taxonomy and survey. *ACM Computing Surveys (CSUR)*, 53(4):1–47, 2020.

[RBWL18] Patrick Rebentrost, Thomas R Bromley, Christian Weedbrook, and Seth Lloyd. Quantum hopfield neural network. *Physical Review A*, 98(4):042308, 2018.

[RDK+19] David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *arXiv preprint arXiv:1906.05433*, 2019.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[Roj96] Raul Rojas. The backpropagation algorithm. In *Neural networks*, pages 149–182. Springer, 1996.

[RR08] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[RSML18] Patrick Rebentrost, Adrian Steffens, Iman Marvian, and Seth Lloyd. Quantum singular-value decomposition of nonsparse low-rank matrices. *Physical review A*, 97(1):012327, 2018.

[SBG+19] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3):032331, 2019.

[Sch21] Maria Schuld. Quantum machine learning models are kernel methods. *arXiv preprint arXiv:2101.11020*, 2021.

[SEJ+20] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[Sim97] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[SM21] Changpeng Shao and Ashley Montanaro. Faster quantum-inspired algorithms for solving linear systems. *arXiv preprint arXiv:2103.10309*, 2021.

[SSM21] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A*, 103(3):032430, 2021.

[SSP14] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, 2014.

[SUR+20] Yohichi Suzuki, Shumpei Uno, Rudy Raymond, Tomoki Tanaka, Tamiya Onodera, and Naoki Yamamoto. Amplitude estimation without phase estimation. *Quantum Information Processing*, 19(2):1–17, 2020.

[SWM17] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Tan18] Ewin Tang. Quantum-inspired classical algorithms for principal component analysis and supervised clustering. *arXiv preprint arXiv:1811.00414*, 2018.

[Tan19] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 217–228, 2019.

[TL20] Nicolas Tremblay and Andreas Loukas. Approximating spectral clustering via sampling: a review. In *Sampling Techniques for Supervised or Unsupervised Tasks*, pages 129–183. Springer, New York, NY, USA, 2020.

[TMGB19] Francesco Tacchino, Chiara Macchiavello, Dario Gerace, and Daniele Bajoni. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1):1–8, 2019.

[TS13] Amnon Ta-Shma. Inverting well conditioned matrices in quantum logspace. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 881–890. ACM, 2013.

[TYL17] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3147–3155, 2017.

[vAG18] Joran van Apeldoorn and András Gilyén. Improvements in quantum sdp-solving with applications. *arXiv preprint arXiv:1804.05058*, 2018.

[VAGGdW17] Joran Van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum sdp-solvers: Better upper and lower bounds. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 403–414. IEEE, 2017.

[Vat09] Andrea Vattani. The hardness of k-means clustering in the plane. *Manuscript, accessible at http://cseweb. ucsd. edu/avattani/papers/kmeans_hardness. pdf*, 617, 2009.

[VBB17] Guillaume Verdon, Michael Broughton, and Jacob Biamonte. A quantum algorithm to train neural networks using low-depth circuits. *arXiv preprint arXiv:1712.05304*, 2017.

[Wal16] M Mitchell Waldrop. The chips are down for moore's law. *Nature News*, 530(7589):144, 2016.

[WCCY20] Jiayun Wang, Yubei Chen, Rudrasis Chakraborty, and Stella X Yu. Orthogonal convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11505–11515, 2020.

[WGM19] Shusen Wang, Alex Gittens, and Michael W Mahoney. Scalable kernel k-means clustering with nyström approximation: relative-error bounds. *The Journal of Machine Learning Research*, 20(1):431–479, 2019.

[WKS14a] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *arXiv preprint arXiv:1401.2142*, 2014.

[WKS14b] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum deep learning. *arXiv preprint arXiv:1412.3489*, 2014.

[Wu17] J Wu. Introduction to convolutional neural networks. 2017.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[YHJ09] Donghui Yan, Ling Huang, and Michael I Jordan. Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 907–916. ACM, New York, NY, USA, 2009.

[YLC14] Theodore J Yoder, Guang Hao Low, and Isaac L Chuang. Fixed-point quantum search with an optimal number of queries. *Physical review letters*, 113(21):210501, 2014.

[ZDF+18] Zhikuan Zhao, Vedran Dunjko, Jack K Fitzsimons, Patrick Rebentrost, and Joseph F Fitzsimons. A note on state preparation for quantum machine learning. *arXiv preprint arXiv:1804.00281*, 2018.

[ZHLT20] Kaining Zhang, Min-Hsiu Hsieh, Liu Liu, and Dacheng Tao. Toward trainability of quantum neural networks. *arXiv preprint arXiv:2011.06258*, 2020.

[ZNL21] Alexander Zlokapa, Hartmut Neven, and Seth Lloyd. A quantum algorithm for training wide and deep classical neural networks. *arXiv preprint arXiv:2107.09200*, 2021.

[ZYST19] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.