Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN

COMPUTER SCIENCE AND ENGINEERING

Ciclo 33

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

A NOVEL FRAMEWORK FOR QUANTUM MACHINE LEARNING

Presentata da: Antonio Macaluso

Coordinatore Dottorato

Davide Sangiorgi

Supervisore

Stefano Lodi

Co-supervisore

Claudio Sartori

Esame finale anno 2021

To my dad

Acknowledgements

I am very grateful to my wonderful family and friends for their unwavering support and kindness. I am particularly thankful to my sister Marilù and my mom Rita, for always believed in me.

During my career, I have met several extraordinary teachers who allowed me to reach important personal goals. I sincerely thank my supervisors Prof. Stefano Lodi and Prof. Claudio Sartori for suggesting to focus my research on such an exciting topic. Their expertise and guidance have been precious in formulating the right research questions. Furthermore, I would like to thank Prof. Daniela G. Calò for being the best teacher I have ever met. She introduced me to the fascinating world of research, and with her passion and competence, she motivated me to consider the possibility of a PhD. I would also like to extend my thanks to Prof. Herbert Wiklicky, who introduced me to the field of quantum computing at the beginning of my PhD program. At that time, the topic was unknown to me, and his lectures have been immensely inspiring to choose quantum computing as the main theme for my PhD.

My deep gratitude is to my closest friends with who I spent wonderful moments. Particularly, I thank Lorenzo for the lengthy discussions about physics, statistics and everything we are passionate about. He always pushed me to improve and instilled in me the curiosity for physics. Also, I thank Elena for helping me during these last three years. Her advice and suggestions have been crucial to me. I will not forget to thank Giuseppe and Silvia, with whom I spent amusing moments and for having survived so many of my oddities.

Finally and foremost, I would like to thank my friend Luca for his patient support and the stimulating discussions, as well as the happy distractions to rest my mind outside of research. Besides a great colleague, he has always been a valuable friend, and without his help, generosity, and extraordinary intellect, I would never have completed my doctorate.

Nothing is more practical than a good theory.

Vladimir N. Vapnik

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

Abstract

Computer Science and its applications had a massive impact on human life in the last decades. Automating repetitive tasks, connecting people and resources thanks to the Internet, and all the consequent services are just a few examples of how the classical computation paradigm has profoundly influenced our perception of the world and our interaction with it. Following this stream, recent developments in machine learning designate artificial intelligence as the next big thing.

Although the math behind these techniques has been around for more than forty years, the field's current success revolves around two crucial factors: data and computational power. Hence, these agents are expected to be the driving forces of the next sector advancements also in the short term. Data availability will continue to grow due to the digital revolution, thus fostering innovative applications in the field. On the other hand, with the cost of technology reaching a plateau (the end of Moore's law) and the physical limitations in building more powerful classical computers, improving existing methods will become more challenging. As a result, the lack of adequate computational power to analyse this ever-growing amount of information seems to be the foremost hamper for future developments. In light of this, quantum computing represents one possibility to bring the field of machine learning to a new era.

Quantum computation is an emerging computing paradigm with the potential to revolutionise the world of information technology. It leverages quantum mechanics laws to endow quantum machines with tremendous computing power, thus enabling the solution of problems impossible to address with classical devices. For this reason, the field is attracting ever-increasing attention from both academic and private sectors, and its full potential is still to be understood.

This dissertation investigates how classical machine learning can benefit from quantum computing and provides several contributions to the emerging field of Quantum Machine Learning.

First and foremost, the idea is to provide a universal, efficient framework that can reproduce the output of a plethora of classical machine learning algorithms exploiting quantum computation's advantages. The proposed framework is named *Multiple*

Aggregator Quantum Algorithm (MAQA) due to its capability to combine multiple functions to solve typical supervised learning tasks. Thanks to this property, in its general formulation MAQA can be potentially adopted as the quantum counterpart of all those models falling into the scheme of aggregation of multiple functions. The theoretical design of the quantum algorithm and the corresponding circuit's implementation are presented. As a second meaningful addition, two practical applications are illustrated: the quantum version of ensemble methods and neural networks.

The final contribution addresses the restriction to linear operations imposed by quantum mechanics. The idea is to exploit a quantum transposition of classical Splines to approximate non-linear functions, thus overcoming this limitation and introducing significant advantages in terms of computational complexity theory.

The thesis is organised as follows. Part I explains the basic elements of quantum computation. After a brief introduction to quantum mechanics' postulates and their implications, the key enabling requirements for a real quantum device are discussed. Special focus is given on the different technologies to build quantum computers and the difference between long-term fault-tolerant and error-affected near-term devices. Furthermore, all the fundamental elements for quantum computation are presented, including Dirac notation, qubits, quantum gates, quantum algorithms and quantum complexity theory.

Part II introduces the emerging field of Quantum Machine Learning. After an overview of classical methods, several proposals of quantum algorithms for machine learning tasks are presented. The problem of state preparation is then examined, and some considerations about complexity theory in Quantum Machine Learning are given.

Part III contains the innovative contributions of the thesis, and it is divided into three parts. Firstly, the idea of a machine learning model as functions aggregator is provided, and some algorithms falling into this framework are considered. Secondly, the MAQA is introduced alongside two possible applications: quantum ensemble methods and quantum neural networks. Thirdly, the idea of quantum splines and its implementation, are illustrated.

Each contribution is corroborated from experiments executed on simulators and real quantum devices that demonstrate in practice what is expressed by the theory, thus showing the merits of the presented algorithms.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

AI	Artificial Intelligence
DL	Deep Learning
HHL	Harrow Hassidim Lloyd
MAQA	Multiple Aggregator Quantum Algorithms
ML	Machine Learning
QC	Quantum Computing
QNN	Quantum Neural Networks
qSLP	quantum Single Layer Perceptron
QSVM	Quantum Support Vector Machine

Part I

Quantum Computing

Chapter 1

Introduction

A computation is essentially a physical process performed on a machine whose operations obey specific laws of physics. The classical theory of computation is based on the model of the Universal Turing Machine, which works according to a set of rules and principles set out in 1936 by Alan Turing and further elaborated by John Von Neumann in the 1940s. The implicit assumption underlying these principles is that a Turing machine idealises a mechanical computation device, with a potentially infinite memory, that obeys the laws of classical physics. Despite enormous progress technologies made in the last decades, these principles have remained essentially unchanged. Although the theory of classical computation provides a potent tool to process and analyse information, there are many limitations regarding what classical computers can do. Thus, the possibility to explore different models of computation has always been taken into consideration to push the Computer Science further.

In 1981, Richard Feynman during the *First Conference on the Physics of Computation* observed that the simulation of quantum physics was impossible to perform using a classical device. In particular, Feynman denoted that an exact simulation of nature was possible only considering a different kind of devices that would work the same as nature itself. The importance in simulating physics stems from the fact that, as Feynman said:

> *"The physical world is quantum mechanical, and therefore the proper problem is the simulation of quantum physics (...) the number of computer elements to simulate a large physical system is proportional to the space-time volume of the physical system (...) If doubling the volume I would need an exponentially larger computer".*

Starting from these assumptions, Feynman proposed, for the first time, a basic model for a quantum computer. A few years later, David Deutsch described the idea of the first universal quantum computer. Just as a Universal Turing machine can simulate any other Turing machine efficiently (Church-Turing thesis), a universal quantum computer can simulate any other quantum system with at most a polynomial overhead.

An important turning point happens in 1994 when Peter Shor proposes a novel algorithm that allows a quantum computer to factor large integers efficiently. For the first time, quantum computing appears as a general, alternative paradigm to overcome the limitation of classical computing and perform tasks that are intractable using the classical approaches, not necessarily related to quantum physics. Shor's algorithm can theoretically break many of the cryptosystems in use today, and its invention sparked a tremendous interest in quantum computers. Two years later, Lov Grover, at Bell Labs, invents the quantum database search algorithm that provides a quadratic speed-up with respect to the best classical counterpart. The algorithm can be applied to a much more all-inclusive variety of problems. Indeed, any problem that has to be solved by random brute-force search can potentially leverage Grover's algorithm.

From the nineties onwards, many quantum algorithms have been developed, and theoretical research has provided many examples of how quantum computing can handle intractable problems even for the best classical supercomputer. However, most of those algorithms assume a perfect working quantum machine, that is something we will not have soon. Besides this, in the last years, there has been tremendous progress in the experimental developments of quantum computers, with the possibility to access small machines that it is reasonable to think will be useful in future for specific cases.

1.1 A Brief Introduction to Quantum Mechanics

Quantum computers apply the laws of quantum mechanics to provide a different mechanism for computation. Fortunately, in-depth knowledge of quantum physics is not a prerequisite for understanding quantum algorithms, as it is not necessary to know the engineering to build a processor to design classical algorithms. However, to be familiar with quantum mechanics' basic concepts allows a better understanding of the mathematics behind quantum computation and the quantum algorithms themselves.

The quantum mechanics theory is ruled by a set of axioms (or postulates), derived after a long experiment-and-failure process, that accurately describes the behaviour of a quantum system. The first postulate defines the state space of quantum objects, the ground in which quantum mechanics takes place:

Postulate 1

Associated to any isolated physical system is a complex vector space with an inner product (i.e., the Hilbert space) known as the state space of the system. The system is completely described by its state vector, a unit vector in the system's state space.

This postulate implies that any computational task that uses quantum mechanics requires the input data to be mapped into the Hilbert space, that is a generalised version of the traditional Euclidean space. Notice that this postulate does not provide specific information about any physical state. It merely circumscribes the region within which movements are allowed when a quantum system is analysed. The simplest quantum mechanical system is the *qubit*, a two-dimensional state space. Suppose $|0\rangle$ and $|1\rangle$ form an orthonormal basis for the state space, then any arbitrary state vector can be written as:

$$|\psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix} = a \begin{bmatrix} 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 1 \end{bmatrix} = a\,|0\rangle + b\,|1\rangle\,, \tag{1.1}$$

where a and b are complex numbers, and the vector $|\psi\rangle$ is subjected to a normalisation condition, which means that the inner product of the state vector with its conjugate transpose is equal to 1 (*i.e.* $\langle\psi|\psi\rangle = 1$).

The second postulate relates the evolution of a quantum system:

Postulate 2

The evolution of a closed quantum system[1] *is described by a unitary transformation. Thus, the state $|\psi\rangle$ of the system at time t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2:*

$$|\psi'\rangle = U\,|\psi\rangle\,. \tag{1.2}$$

where U is a unitary matrix[2]. Just as the first postulate does not provide specific information of a particular quantum state, the second postulate only tells what kind of operation determines the evolution of a quantum system but does not give any information about the construction of U. In quantum computing, a unitary matrix U is called *quantum gate*, and it represents the basic object that allows performing a generic computation on a given quantum state. As a consequence of the unitarity

[1]The postulate 2 describe the evolution of a system at two different times, but it can be revised to describe the evolution in a *continuous time* by using the Schrödinger equation.

[2]A unitary matrix is a matrix whose inverse U^{-1} is equal to its conjugate transpose $U^\dagger$. Therefore, the product between U and $U^\dagger$ is equal to the identity matrix I, i.e. $U \cdot U^\dagger = I$

constraint on U, the *No-cloning theorem* forbids to create identical copies of an arbitrary unknown quantum state. In practice, the *No-cloning theorem* argues that, given any arbitrary quantum state $|\psi\rangle$, it doesn't exist a unitary transformation U such that $U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle$ (proof in Appendix A.1).

When considering the problem of function approximation, the constraint about unitarity represents a big limitation for quantum computation, since it implies that it is impossible to compute an output state vector that is a non-linear function of the input. Actually, as shown in Chapter 7, there are (classical) methods that allow approximating non-linear functions by solving a linear system in the space of predetermined basis functions. The quantum transposition of these methods can help to overcome the constraint about the unitarity of quantum operations for non-linear function approximations.

Importantly, Postulate 2 requires the system to be *closed*, which means there is no interaction with any other quantum system. However, there must also be times when the system has to be observed to read out the information of interest. In this regard, the third postulate considers the effect of measurement on a quantum system:

Postulate 3

Quantum measurements are described by a collection $\{M_m\}$ of measurement operators, where the index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle, \tag{1.3}$$

and the state of the system after the measurement is

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}}, \tag{1.4}$$

Notice that the measurement operator satisfies the *completeness equation*

$$\sum_m M_m^\dagger M_m = I. \tag{1.5}$$

An important example of measurement of a quantum system is the measurement of a qubit in the computational basis, which provides two possible outcomes defined by the two measurement operators $M_0 = |0\rangle\langle 0|$, $M_1 = |1\rangle\langle 1|$[3]. Notice that every observable

[3] $\langle 0|$ indicates the row vector of $|0\rangle$ then M_0 is a 2×2 matrix

of a physical system is associated with a self-adjoint (or Hermitian) operator allowing a complete set of eigenfunctions. In other words, the measurement operation allows to retrieve a specific eigenvalue of the chosen Hermitian matrix, that is associated with a specific computational basis of the measured state vector.

The fourth postulate describes the relationship between the state spaces of component systems and the overall state space:

Postulate 4

The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have n systems, and i-th is prepared in the state $|\psi_i\rangle$ (for $0 \leq i \leq n$), the joint state of the total system is $|\psi_1\rangle \otimes |\psi_2\rangle \cdots \otimes |\psi_n\rangle$.

Since the tensor product of (finite-dimensional) vector spaces has dimension equal to the product of the dimensions of the factors, if the component systems are single qubits, the whole state space is described by a 2^n possible basis states.

As a consequence of these postulates, quantum computation has three main features that enable quantum algorithms to solve intractable problems for classical algorithms. These features are *superposition*, *entanglement*, and $interference$.

The superposition means that a quantum system exists in all of its possible basis states at the same time. Differently from classical bits, that can be 0 *or* 1, the ability of a qubit to be in a linear combination of the $|0\rangle$ *and* $|1\rangle$ allows to evaluate a function on many inputs simultaneously. In practice, the superposition is the mathematical interpretation of the *particle-wave duality* that is a phenomenon which allows matter to exist simultaneously as particles (because it can be assigned definite quantities like location and momentum) and as wave (as a propagating disturbance in some underlying field). Particle-wave duality has been proved in many experiments, though this behaviour has not been predicted by the theory but it has been observed experimentally and then led to the definition of the theory.

Another important property is the entanglement, that produces correlation between the measurements of two distinct quantum systems. For instance, two qubits can be entangled in such a way that acting on one will affect the other. As a result of the entanglement, it is possible to perform the quantum teleportation of a quantum state, where two or more particles are inextricably linked to each other. If two particles are entangled and shared between two separate locations, the encoded information is teleported, no matter the distance between them.

Interference is a phenomenon derived by the ability of a quantum system to behave as wave. It usually refers to the interaction of waves that are correlated or coherent with each other, either because they come from the same source or because they have nearly the same frequency.

All these three features will be employed to propose a novel quantum framework that allows to perform machine learning tasks efficiently.

1.2 Requirements for Quantum Computers

Quantum computers are machines that use the properties of quantum mechanics to store and process information. The most critical characteristic is the closed box requirement: it is impossible to observe a quantum system without producing an uncontrollable disturbance, and it is necessary to keep that system perfectly isolated from the rest of the universe to perform computation. Also, we should be able to control the quantum system from outside and eventually to read out the result, while the qubits have to interact with each other to process information. All these aspects make the construction of a quantum device a highly challenging task.

There are five requirements (DiVincenzo, 1997) to build an efficient and reliable quantum device: (i) the degrees of freedom of the quantum system must be precisely delineated, which means that the dimensions of the correspondent Hilbert space should be precisely enumerable in terms of the number of possible states allowed; (ii) it must be possible to initialise the quantum system in a fiducial starting quantum state (e.g. the all-zero state); (iii) a quantum system has to be, as much as possible, isolated from the rest of the environment. In particular, if the state of the computer at instant t_1 is ideally supposed to be $|\psi\rangle$, then the same state at t_2 should differ from $|\psi\rangle$ by only a small amount; (iv) it must be possible to control the quantum system by a sequence of unitary transformations. In fact, any quantum algorithm is expressed in terms of such sequences; (v) it is necessary to act on the quantum system with a strong form of measurement. *Strong* means that the measurement determines which orthogonal eigenstate of some particular Hermitian operator the quantum state belongs to.

All these requirements need to be fulfilled in order to make a quantum system usable for computation. The most challenging one is probably the ability to isolate the system, since quantum systems are by nature very fragile and sensitive to small amounts of information leakage that can disturb quantum mechanical waves. This destructive process is known as *decoherence* and no system is fully free of it. However, small amounts of decoherence can be mitigated using Quantum Error Correction (QEC).

In classical computing, the basic principle of error correction is that the number of bits used to encode a given amount of information is increased. Starting from this redundant encoding, a set of instructions, known as an *error correction code* are specified, to obtain the information of interest (Devitt et al., 2013; Hamming, 1950; MacKay and Mac Kay, 2003). The same idea applies to quantum computing, where the number of qubits is increased to correct the noise due to the effect of decoherence. The use of QEC makes necessary the distinction between logical and physical qubits. A logical qubit is the one used for programming where an n-qubit quantum state is represented using $2^n - 1$ complex numbers. A physical qubit is the actual implementation of a logical qubit in a real quantum device. Real physical qubits suffer from decoherence. Usually, logical qubits are implemented on top of multiple physical qubits to get redundancy. Realistically, most of the resources used in a fault-tolerant quantum computer will be used to correct its own errors.

QEC is the most promising approach to protect quantum computation and scale efficiently quantum computers. Unfortunately, there is a significant overhead cost for doing it, so reliable quantum computers are not likely to be available in the near future. However, recent developments in building small scale quantum devices pushed towards the search for algorithms capable of exploiting such devices that, despite being noisy, can help to deal with difficult problems.

1.3 Building a Quantum Computer

There are many different ways in which a qubit can be generated: as the two different polarisations of a photon, the alignment of a nuclear spin in a uniform magnetic field, the two states of an electron orbiting a single atom. Each possible realisation of a qubit leads to building a different quantum device, and each technology presents specific challenges. In this section, the main technologies are discussed, with particular focus on those that represent the most promising approaches to building a full-scale quantum computer.

Superconductors

The quantum architecture based on superconducting qubits is the leading candidate for scalable quantum computing platform. Superconductivity is the property of some materials to conduct electricity and shield magnetic fields perfectly if they are cooled down to low temperatures. It is a macroscopic quantum phenomenon since the carriers of electric charge in a superconductor first pair up and then condense into a single quantum state as a single large atom. Using small contacts between superconductors (so-called Josephson junctions), it is possible to engineer a variety of quantum circuits and run quantum algorithms on them. The advantages of superconducting qubits are the following (Huang et al., 2020). *High designability:* different types of qubits can be designed and adjusted based on different requirements. *Scalability*: superconductors are based on the existing processes, and high-quality devices can be built by leveraging advanced chip-making technologies to improve the scalability of the quantum system. *Easy to couple:* superconducting qubits can be coupled by capacitance and inductance, this makes relatively easy to couple multiple qubits together. *Easy to control:* commercial microwave device can be used in superconducting quantum computing experiments to perform operations and measurement. All these features make such technology the most promising approach towards fault-tolerant quantum computers.

Recently, most of the efforts to build a quantum processor have been devoted to developing superconductor technology and the milestone of demonstrating quantum supremacy has been achieved using a Quantum Processing Unit (QPU) of 53 superconducting qubits (Arute et al., 2019).

Photons

Realising a qubit as the polarisation state of a photon is appealing because photons are relatively free of decoherence, that is the main issue of other quantum technologies. Photonics is currently the only architecture that enables building a room-temperature and easily-networked quantum computer. The qubits usually are encoded into the states of light using a method called GKP (Gottesman et al., 2001) that allows implementing them using standard integrated photonic devices. Moreover, photonic quantum computers allow efficient implementation of one-qubit gates that can be easily performed with incredibly high fidelity. Also, they are not confined to a unit of quantum information described by a superposition of two states (qubits), but they potentially allow to encode multilevel qudits (generalisation to a superposition of d possible states). However, photons do not easily interact, making deterministic two-qubit gates as the biggest hurdle to overcome. Furthermore, although an optical quantum computer can potentially work at room temperature, it requires costly dedicated hardware.

Recently, the company Xanadu released the first photonic quantum computing platform[4], describing the photonic quantum computing as the most viable approach towards universal fault-tolerant quantum computation.

Trapped Ions

An ion is defined as an electrically charged molecular entity that results from the

[4]https://www.xanadu.ai/cloud-platform

transfer of one or more electrons by an atom, a molecule or a group of atoms linked together. The first proposal of a quantum computer based on cold trapped ions appeared in (Cirac and Zoller, 1995). In this scheme, the qubits are realised by ions confined in radiofrequency traps.

In quantum computers based on trapped ions, single-qubit gates (Brown et al., 2011), two-qubit gates (Benhelm et al., 2008), state preparation routine and readout (Myerson et al., 2008), can be performed with a fidelity exceeding that required for fault-tolerant quantum computation using high-threshold quantum error correction codes. However, despite the promise shown by trapped ions, there are still many challenges that must be addressed in order to realise a practically useful quantum device. Main among these is increasing the number of qubits while maintaining the ability to control and measure them individually with high fidelity.

To summarise, there is no single way to achieve full-scale quantum device and the roadmap to a universal quantum computer is still evolving. Thus, the best way to accomplish a computational task depends on the strengths and weaknesses of the physical system at hand. Even so, the essential ingredients remain a robust, high fidelity execution of quantum operations, which is the goal of all physical implementations of quantum computers. A large number of other technologies exhibiting quantum coherence have been proposed and tested for quantum computers. For a complete review see Ladd et al. (2010).

1.4 Fault-Tolerant Computation

The number of qubits is essential for building a full-scale quantum computer, but that is not the only requirement. What it is necessary to care about is also the *quality* of those qubits and, in particular, the accuracy in transforming them[5]. In fact, the central challenge in building quantum computers is maintaining the simultaneous ability to control the quantum system and to preserve its strong isolation. These two characteristics lead to the main plague for a fault-tolerant quantum architecture, that is known as *quantum decoherence*. Technically, the decoherence is measured by two parameters: the energy relaxation (T_1), which is the time taken for the excited state $|1\rangle$ decays toward the ground state $|0\rangle$, and the coherence time (T_2), that describes how long a phase of qubit stays intact.

[5]with the current best hardware, it is possible to control trapped ions or superconducting circuits with an error rate for two-qubits gates above .1% level.

Another important metric to consider is the time to execution for setting the time scale needed for a quantum device to solve a problem. In this sense, it is notable that superconducting circuits are about a thousand times faster than ion trap quantum processors. In this regards, IBM developed a metric, known as *QuantumVolume* (Cross et al., 2019) that can be measured using a concrete protocol on near-term quantum computers of modest size. In particular, this measure quantifies the largest random circuit of equal width and depth that the computer successfully implements, and it represents a pragmatic way to measure and compare progress toward improved system-wide gate error rates for near-term quantum computation and error-correction experiments.

Besides the incredible advances made in the last few years, the noise produced by the available quantum devices is much larger than the signals and, practically, scaling up using QEC is something far from practice. Therefore, a large-scale, error-corrected quantum computer is still an extremely ambitious goal.

1.5 Near-Term Quantum Computation

As discussed above, the construction of full-scale, error-corrected quantum devices still poses many technical challenges. At the same time, significant progress has been made in the development of small-scale quantum computers, hence giving rise to the so-called *Noisy Intermediate-Scale Quantum* (NISQ) era. The terms intermediate-scale refers to the size of quantum computers that are currently available with a number of qubits ranging from 50 to a few hundred. Noisy emphasises that it is not possible to control over those qubits perfectly; the noise will place severe limitations on what quantum devices can achieve in the near term. Thus NISQ machines are still not sufficiently powerful to be a credible alternative to classical ones and cannot be used to execute many of the algorithms described in the literature[6]. Furthermore, noise in quantum gates limits the size of quantum circuits that can be executed reliably. However, 50 qubits is a significant milestone because it is beyond what can be simulated by brute force using the most powerful existing digital supercomputers. Hence, it is not expected for NISQ devices to change the world by itself. Instead, they should be regarded as a step toward more powerful quantum technologies that will be developed in the future.

For these reasons, many researchers are currently focusing on algorithms for NISQ machines that may have an immediate impact on real-world applications, e.g. chemistry

[6]quantum algorithms that assume accessing a fault-tolerant quantum device are also known as *full-coherent protocols*.

(Lanyon et al., 2010) and optimisation (Farhi et al., 2014; Kandala et al., 2017). In particular, recently the idea of quantum computational supremacy (Harrow and Montanaro, 2017; Preskill, 2012) has been proved in practice by Google (Arute et al., 2019) using a processor with programmable superconducting (physical) 53 qubits. Quantum supremacy argues that, under reasonable assumptions, quantum states which are easy to prepare with a quantum computer have super classical properties; specifically, measuring all the qubits in such a state corresponds to sample from a correlated probability distribution that cannot be sampled from by any efficient classical means. The experiment performed by Google took about 200 seconds to sample one instance of a quantum circuit a million times. The equivalent task for a state-of-the-art classical supercomputer would take approximately 10.000 years[7]. This dramatic increase in speed compared to all known classical algorithms can be considered an experimental realisation of quantum supremacy for this specific computational task.

The realisation of quantum supremacy is an incredible milestone because it allows appreciating the potential of quantum computation. However, it is mostly a worthy goal notable for entrepreneurs and investors to attract attention in the field, rather than a sign of significant progress toward more practical applications.

Although NISQ technology will not be the quantum revolution expected, it is reasonable to think that near-term quantum devices are special-purpose devices that, together with classical methods, will be able to solve problems bounded to specific fields of application.

1.6 Promising Applications and Future Directions

In general, it is possible to think of at least three good reasons to suppose that quantum computers have capabilities surpassing classical computers. First, there are already quantum algorithms that (theoretically) can solve classically intractable problems, e.g. Shor's algorithm for prime factors of large composite integers. Second, complexity theory arguments regarding quantum computers can efficiently prepare quantum states with extremely hard properties to reproduce using classical simulations (e.g. the experiment of quantum supremacy). Third, no classical algorithm can simulate a quantum computer efficiently. This task would require exponentially larger resources while increasing linearly the number of qubits of the quantum system to simulate.

[7]There is a controversial opinion provided by IBM that argues that an ideal simulation of the same task can be performed on a classical system in 2.5 days and with far greater fidelity.

In practice, there are many applications where quantum computation may have an impact in the future. Likely, the best natural way to leverage quantum computation is by quantum simulation. This task was the main goal of Feynman's proposal. The simulation of strongly entangles matter is a challenging computational problem and, in the long term, the simulation of different quantum systems will be the main field of application for quantum computers. This topic also involves quantum chemistry where it has been shown that even NISQ technology can help to solve a difficult problem for a classical algorithm in use.

Another promising field is represented by optimisation. Indeed, there are some combinatorial optimisation problems for which even finding a good approximated solution can be an NP-hard problem (Khot, 2016). In this sense, the most promising approach is the use of *Quantum Approximate Optimization Algorithm* (QAOA) (Farhi et al., 2014) that exploits adiabatic quantum computation[8] to produce approximate solutions for combinatorial optimisation problems.

Given the recent success of Machine learning (ML) and Deep Learning (DL), it is natural to examine whether quantum algorithms will be able to enhance classical methods to improve them even further. In principle, many methodological reasons suggest the ability of quantum phenomena to improve classical ML methods, but the road towards *Quantum Machine Learning* is still long and full of obstacles. However, the history of ML teaches us that when hardware becomes available, that stimulates and accelerates the development of new algorithms. This may be the case also for quantum machine learning algorithms.

In all these fields of application, the distinction between fault-tolerant and near-term is necessary to understand the impact of quantum computation in real-world problems. Also, it is essential to make progress using NISQ devices by developing better methods guided by relatively small-scale experiments that can lead QC communities in the right direction.

[8]Adiabatic quantum computation is a form of quantum computing which relies on the adiabatic theorem to do calculations. The company D-Wave (https://www.dwavesys.com/) use a process called quantum annealing to search for solutions to a problem.

Chapter 2

Basic Principles of Quantum Computing

In this chapter, the basics to understand quantum computation and its potential are provided. The chapter is constructed as follows: Section 2.1 introduces the formalism typically employed in quantum mechanics and quantum computing. This formalism allows manipulating linear algebra objects that describe the properties of quantum systems. Section 2.2 describes qubits and quantum gates that are the two fundamental concepts that permit to move from pure mathematical formalism to design real quantum algorithms. Also, the most important quantum gates are described, with a particular focus on those in use in the proposal of this thesis. Sections 2.3 and 2.4 describe two of the most important properties of quantum computation. The first one is *Entanglement*. It represents a sort of *strong* correlation between quantum systems, and it is something impossible to reproduce using classical computation; the second one is *measurement* that has important implications when applied to quantum computing. Section 2.5 set out the idea of quantum algorithms. The circuit model is introduced, and a few examples of quantum algorithms are provided. Finally, Section 2.6 discusses the computational complexity framework where a quantum algorithm can be analysed, underling the differences with classical computation.

2.1 Hilbert space and Dirac Notation

Before discussing the basics of quantum computation, it is necessary to introduce the mathematical formalism typical of quantum mechanics, which is inherited by quantum computing itself. This formalism is known as Dirac (or Bra-ket) notation, and it is another way of describing vectors.

A column vector $\mathbf{v}$ in Dirac notation is described as:

$$\mathbf{v} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = |\mathbf{v}\rangle\,, \tag{2.1}$$

where $\{v_i\}_{i=0,\ldots,n}$ are complex numbers and $|\mathbf{v}\rangle$ is referred to as "*ket-v*". The correspondent dual vector of $|\mathbf{v}\rangle$ is called "*bra-v*"and has the following notation:

$$\langle\mathbf{v}| = \overline{\mathbf{v}^T} = \begin{bmatrix} \overline{v_0} & \overline{v_1} & \ldots & \overline{v_n} \end{bmatrix}, \tag{2.2}$$

where $\overline{v_i}$ is the complex conjugate of v_i (it is common also the notation v^*). Notice that the vector $\overline{\mathbf{v}^T}$ is also known as the *adjoint* of $\mathbf{v}$ and is usually described using the standard notation $\mathbf{v}^\dagger$.

Dirac notation is convenient when describing vectors in the Hilbert space $\mathbb{C}^n$, which is the vector space to represent a quantum system. In particular, a finite-dimensional Hilbert space is a vector space with an inner product and a norm defined by that inner product[1]. The inner product is an operation that assigns a scalar value to each pair of vectors $\mathbf{u}$ and $\mathbf{v}$ in the vector space and it is denoted as $\langle\mathbf{u}|\mathbf{v}\rangle$. In practice, it is calculated as the dot product between $\mathbf{v}$ and $\overline{\mathbf{u}^T}$ (the conjugate transpose of $\mathbf{u}$):

$$\langle\mathbf{u}|\mathbf{v}\rangle = \overline{\mathbf{u}^T}\mathbf{v} = \begin{bmatrix} \overline{u_0} & \overline{u_1} & \ldots & \overline{u_n} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = \overline{u_0}\cdot v_0 + \overline{u_1}\cdot v_1 + \cdots + \overline{u_n}\cdot v_n. \tag{2.3}$$

By definition, the inner product satisfies the following conditions:

1. $\langle\mathbf{v}|\mathbf{v}\rangle \geq 0$ with $\langle\mathbf{v}|\mathbf{v}\rangle = 0$ if and only if $|\mathbf{v}\rangle = \mathbf{0}$;

2. $\langle\mathbf{u}|\mathbf{v}\rangle = \overline{\langle\mathbf{v}|\mathbf{u}\rangle}$ for all $|\mathbf{u}\rangle$, $|\mathbf{v}\rangle$ in the vector space;

3. $\langle\mathbf{u}|\alpha_0\mathbf{v} + \alpha_1\mathbf{w}\rangle = \alpha_0\langle\mathbf{u}|\mathbf{v}\rangle + \alpha_1\langle\mathbf{u}|\mathbf{w}\rangle$;

   More generally, the inner product of $|\mathbf{u}\rangle$ and $\sum_i \alpha_i |\mathbf{v}_i\rangle$ is equal to $\sum_i \alpha_i \langle\mathbf{u}|\mathbf{v}_i\rangle$ for all scalars α_i and vectors $|\mathbf{u}\rangle$, $|\mathbf{v}\rangle$ in the vector space (this is known as *linearity in the second argument*).

[1] The norm is the length of a vector, or alternatively, the distance from the origin to the point that the vector represents.

Also, the 2-norm (or l^2-norm) of a vector in a Hilbert space is defined using the square root of the inner product of $|\mathbf{v}\rangle$ with itself:

$$|| \, |\mathbf{v}\rangle \, ||_2 = \sqrt{\langle \mathbf{v}|\mathbf{v}\rangle}. \tag{2.4}$$

Another important operation is the *tensor product* (or *Kronecker product*) that is denoted as $\otimes$. Given two column vectors $|\mathbf{u}\rangle$ and $|\mathbf{v}\rangle$ of lengths m and n, the tensor product $|\mathbf{u}\rangle \otimes |\mathbf{v}\rangle$[2] between them is a column vector of length $m \cdot n$ calculated as follows:

$$|\mathbf{u}\rangle \otimes |\mathbf{v}\rangle = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \otimes \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 \cdot v_1 \\ u_1 \cdot v_2 \\ \vdots \\ u_1 \cdot v_n \\ u_2 \cdot v_1 \\ \vdots \\ u_{m-1} \cdot v_n \\ u_m \cdot v_1 \\ \vdots \\ u_m \cdot v_n \end{bmatrix}. \tag{2.5}$$

The tensor product is extremely important because represents the mathematical abstraction to describe the interaction between quantum systems: the vector space describing one quantum system tensored with another vector space of another quantum system is the vector space composed by the linear combinations of all the vectors in the two vector spaces. Also, a vector tensored with itself n times is denoted as $|\mathbf{v}\rangle^{\otimes n}$.

Since the composition of many quantum systems can be described as the tensor product between them, the overall dimension of the Hilbert space grows exponentially, while increasing the number of component systems linearly, thus without an exponential cost in resources.

2.2 Qubits and Quantum Gates

The basic unit of information in classical computing is the bit, that can assume two possible values: 0 or 1. Just as a classical bit, a qubit has two possible basis states $|0\rangle$ and $|1\rangle$ which correspond to the two possible values of a classical bit. The difference

[2]The tensor product is usually simplified as $|\mathbf{u}\rangle |\mathbf{v}\rangle$ or $|\mathbf{uv}\rangle$.

between bits and qubits is that a qubit can be in an additional state that is a *linear combination* of the two basis states (superposition):

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.6}$$

where α and β are complex numbers and are indicated as *amplitudes*. Mathematically, the state of a qubit can be seen as a two-dimensional complex vector space, where $|0\rangle$ and $|1\rangle$ are special cases of the superposition known as *computational basis states*. They form an orthonormal basis for this vector space.

Another difference between bits and qubits regards the operation of measurement. While the value of a bit can be accessed easily, quantum mechanics imposes that much more restricted information about the quantum state of a qubit can be retrieved. When a qubit is measured, the probability to obtain either $|0\rangle$ or $|1\rangle$ are respectively $|\alpha|^2$ and $|\beta|^2$. Since the squared of the amplitudes are probabilities, they must sum to one, then $|\alpha|^2 + |\beta|^2 = 1$. Thus, a state of a qubit is a unit vector in a two-dimensional complex vector space.

In practice, the superposition of states means that the qubit is in $|0\rangle$ and $|1\rangle$ at the same time, this idea runs counter the "common sense" of the physical world around us. Nevertheless, from a statistical perspective, the measurement of a qubit can be considered as a random variable, where the quantum states are the possible outcomes and the amplitudes are the square root of the probabilities associated. Thus, quantum states can be seen as a generalisation of probability distributions and quantum computation is a way to transform them. Furthermore, because of the constraint on the norm, Equation (2.6) can be rewritten as follows:

$$|\psi\rangle = e^{i\gamma}\left(cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}sin\frac{\theta}{2}\,|1\rangle\right), \tag{2.7}$$

where θ, ϕ and γ are real numbers. Notice that it is possible to ignore the factor $e^{i\gamma}$ because it has no observable effects, then a specific quantum state is determined by θ, ϕ. From a geometrical perspective, the *Bloch sphere* (Figure 2.1) provides a useful representation for visualising the state of a single qubit.

Many of the operations on a single qubit are described with a specific transformation in a Bloch sphere. However, it is important to point out that this intuition is limited since there is no simple generalisation of the Bloch sphere when dealing with multiple qubits.

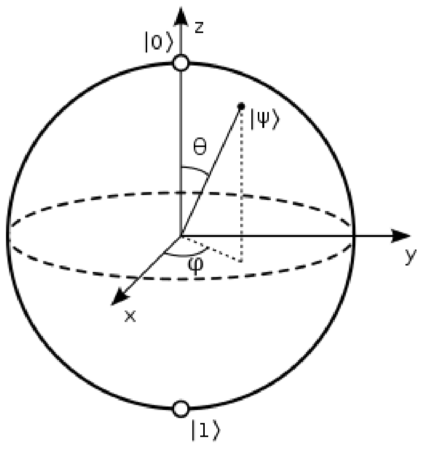


Fig. 2.1 Bloch sphere.

The composition of n qubits[3] generates a state space of dimension 2^n (Postulate 4). Then each normalised vector in such space represents a possible outcome of a measurement. This exponential growth in the state space, suggests the potential capacity of a quantum computer to process information exponentially faster than a classical supercomputer. Observe that for $n = 200$, the total number of basis states is greater than the total number of atoms in the universe.

Formally, a quantum register of n qubits is an element of the 2^n dimensional Hilbert space, $\mathbb{C}^{2^n}$, with computational basis formed by 2^n registers of n qubits:

$$|i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_n\rangle \, , \tag{2.8}$$

where $i_x \in \{0, 1\}$ and $1 \leq x \leq n$.

For instance, a two-qubit system has four *computational basis states* denoted as $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$. A pair of qubits can exist in a superposition of these four states so the quantum state of two qubits associates a complex coefficient (*amplitude*) at each computational basis. The whole quantum system can be described as follows:

$$|\psi\rangle = \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle = \sum_{j=0}^{2^2-1} \alpha_j |j\rangle \, , \tag{2.9}$$

[3]also called n-qubit quantum register.

where

$$|0\rangle = |00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad |1\rangle = |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \quad |2\rangle = |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \quad |3\rangle = |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}. \tag{2.10}$$

The measurement result x comes out with probability $|\alpha_x|^2$, subject to the normalisation condition $\sum_{x\in\{0,1\}^2} |\alpha_x|^2 = 1$.

Since the whole system is described by two qubits that are separate physical objects, one can think to measure just one of the two qubits. In this case, if considering to measure the first qubit, the probability to readout the state $|0\rangle$ is $|\alpha_{00}|^2 + |\alpha_{01}|^2$. Thus, the post measurement state is:

$$|\psi'\rangle = \frac{\alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}, \tag{2.11}$$

Notice that the post-measurement state is re-normalised as expected for a legitimate quantum state.

An important two-qubit state is the EPR pair (Einstein et al., 1935) (or Bell state):

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}, \tag{2.12}$$

that represents the maximal example of quantum entanglement. When measuring a Bell state, the two qubits can be either $|0\rangle$ or $|1\rangle$. However, while the first measurement of one of the two qubits leads to perfectly random results, the measurement of the second qubit is affected by the first measurement and would surely be the same. The two outcomes are then perfectly correlated, and this correlation is something beyond what is possible in the classical world. Indeed, once two qubits are entangled, the possible outcomes in measuring them are correlated independently by their distance.

Entanglement is the key of quantum teleportation and superdense coding, two quantum protocols for communication.

2.2.1 Quantum Gates

The formalism underlying any algorithm based on classical computing is the Boolean circuit model that can be represented by a finite directed acyclic graph with n input nodes, which contains the n input bits ($n \geq 0$). The internal nodes are logic gates (e.g., AND, OR, and NOT) that implement Boolean functions.

The quantum counterpart of classical Boolean circuit families is the quantum circuit model that considers qubits instead of classical bits as input, and replace the AND, OR, and NOT gates by elementary quantum gates, to manipulate quantum information. A quantum gate is a unitary transformation usually performed on a small number of qubits. Mathematically, these gates can be composed by taking tensor products, if gates are applied in parallel to different parts of the register, and ordinary products, if gates are applied sequentially. An example is the quantum *NOT* gate (denoted by X) which is the analogue of classical NOT gate. Given an input qubit, X-gate interchanges the role of the basis state $|0\rangle$ and $|1\rangle$ performing the following transformation:

$$\alpha |0\rangle + \beta |1\rangle \xrightarrow{X} \alpha |1\rangle + \beta |0\rangle . \tag{2.13}$$

It is possible to express the transformation of the quantum *NOT* gate in matrix form:

$$X |\psi\rangle = X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}. \tag{2.14}$$

So single quantum gate can be described by unitary two by two matrices. Notice that the normalisation condition $|\alpha|^2+|\beta|^2 = 1$ for a generic quantum state $|\psi\rangle = \alpha |0\rangle+\beta |1\rangle$ has to be true also for the transformed quantum state after the application of a quantum gate. Furthermore, due to the laws of quantum mechanics, any appropriate gate U can be represented by a unitary matrix, i.e., $U^\dagger U = I$, where $U^\dagger$ is the adjoint of U (obtained by transposing and then complex conjugating U), and I is the identity matrix. Also, since unitary matrices are always invertible, unlike classical computation, any quantum computation is reversible. Importantly, the unitarity constraint is the only constraint on quantum gate, then any unitary matrix represents a valid quantum gate.

We can represent any single qubit gate as a unitary matrix parametrised as follows:

$$U(\theta, \phi, \lambda) = \begin{bmatrix} cos(\theta/2) & -e^{i\lambda} sin(\theta/2) \\ e^{i\phi} sin(\theta/2) & e^{i\lambda+i\phi} cos(\theta/2) \end{bmatrix}, \tag{2.15}$$

where θ, λ and ϕ are real numbers. For example the *NOT* gate can be expressed as $U(\pi, 0, \pi)$.

Among the class of unitary matrices that serve as quantum gates, there are Pauli matrices:

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; \quad Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.16}$$

The Pauli matrices satisfy the condition $X^2 = Y^2 = Z^2 = I$, and when exponentiated, they give rise to three useful classes of unitary matrices:

$$R_x(\theta) \equiv e^{-i\theta X/2} = cos\frac{\theta}{2}I - i\,sin\frac{\theta}{2}X = \begin{bmatrix} cos\,\frac{\theta}{2} & -i\,sin\,\frac{\theta}{2} \\ -i\,sin\,\frac{\theta}{2} & cos\,\frac{\theta}{2} \end{bmatrix}; \tag{2.17}$$

$$R_y(\theta) \equiv e^{-i\theta Y/2} = cos\frac{\theta}{2}I - i\,sin\frac{\theta}{2}Y = \begin{bmatrix} cos\,\frac{\theta}{2} & -sin\,\frac{\theta}{2} \\ sin\,\frac{\theta}{2} & cos\,\frac{\theta}{2} \end{bmatrix}; \tag{2.18}$$

$$R_z(\theta) \equiv e^{-i\theta Z/2} = cos\frac{\theta}{2}I - i\,sin\frac{\theta}{2}Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}. \tag{2.19}$$

These matrices corresponds to *rotation operators* around the three orthogonal axis in a 3-dimensional space (Figure 2.1). Three other important quantum gates are the following:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}; \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}. \tag{2.20}$$

In particular, H is the Hadamard gate that can be used to initialise state $|0\rangle$ into a uniform superposition between $|0\rangle$ and $|1\rangle$.

Any two by two matrix is an example of single-qubit gate, but one can also make use of multi-qubit operations that allows qubits interact with each other. For example, the *controlled*-NOT gate (or CNOT) has two input qubits, known as *control* and *target* qubit respectively. CNOT gate changes the state of the *target* qubit based on the value of the *control* qubit. If the control qubit is set to $|0\rangle$, then the target qubit is left untouched. If the control qubit is set to $|1\rangle$ then the target qubit is flipped:

$$|00\rangle \rightarrow |00\rangle\,; \;\; |01\rangle \rightarrow |01\rangle\,; \;\; |10\rangle \rightarrow |11\rangle\,; \;\; |11\rangle \rightarrow |10\rangle\,; \tag{2.21}$$

Another way of describing the CNOT gate is as generalisation of XOR gate. Given a 2-qubit quantum system, the action of the CNOT on $|\psi\rangle$ can be summarised as:

$$|\psi\rangle = |\psi_1, \psi_2\rangle \xrightarrow{\text{CNOT}} |\psi_1, \psi_2 \oplus \psi_1\rangle \tag{2.22}$$

where $|\psi_1, \psi_2\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$ and $\oplus$ is the addition modulo two, which is exactly what classical XOR gate does. In matrix form:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.23}$$

Another important two-qubit operation is the SWAP gate that swaps the state of the two qubits involved in the operation:

$$|\psi_1, \psi_2\rangle \xrightarrow{\text{SWAP}} |\psi_2, \psi_1\rangle . \tag{2.24}$$

The unitary matrix that represents the SWAP gate is the following:

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.25}$$

An important theoretical result in classical computing is that any function on bits can be computed by the composition of NAND gates which is known as *universal* gate. In the same way, a set of universal quantum gates allows performing any other possible quantum computation, or in other terms, any other unitary operation can be expressed as a finite sequence of gates from the universal set. Thus, a set of gates is *universal*, if any quantum operation can be approximated by a sequence of gates from this finite set[4]. An example of a universal set is the Clifford set (CNOT, H, S) plus the T gate.

Furthermore, one important universal gate is the Toffoli gate (Toffoli, 1980) (also called *controlled-controlled*-NOT gate) which is a 3-qubit gate that acts using two control qubits and one target. Toffoli gate exists in both, classical and quantum

[4]For unitaries on a constant number of qubits, the Solovay–Kitaev theorem guarantees that this can be done efficiently (Dawson and Nielsen, 2006)

computation, this means that quantum computation is at least as powerful as classical computation.

2.3 Entanglement

One of the most counter-intuitive phenomena in quantum mechanics is the entanglement. A pair or group of qubits are entangled when the quantum state of each qubit cannot be described independently of the quantum state of the other. Generally a n-qubit quantum system can be described as the tensor product of the single component:

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle = (a_1 |0\rangle + b_1 |1\rangle) \otimes \cdots \otimes (a_n |0\rangle + b_n |1\rangle). \tag{2.26}$$

Also, it is possible to express the quantum state as a superpositon of 2^N possible basis states:

$$|\psi\rangle = \sum_{i=0}^{2^N-1} \alpha_i |i\rangle , \tag{2.27}$$

where the amplitudes $\{\alpha_i\}_{i=1,\dots 2^N}$ depend linearly on the amplitudes of the single components $\{a_i, b_i\}_{i=1,\dots N}$. If a quantum state of the form in Equation (2.27) cannot be expressed in the form of Equation (2.26), then the state is called *entangled*. One example of such state is the Bell state (Equation (2.12)) which cannot be factorised in the tensor product of two independent qubits. In other terms, it does not exist a_1, a_2, b_1, b_2 such that

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = (a_1 |0\rangle + b_1 |1\rangle) \otimes (a_2 |0\rangle + b_2 |1\rangle). \tag{2.28}$$

In a quantum computer, entanglement translates in the correlation between the probability of observing a given configuration of one qubit to depend on the probability of observing another qubit, entangled with the first one. An especially interesting quality of quantum entanglement is that elements of a quantum system may be entangled even when they are separated by considerable space. The exact physics of quantum entanglement remains elusive even to physicists, but that has not stopped them from applying it to quantum information theory. Quantum teleportation, an important concept in the field of quantum cryptography, relies on entangled quantum states to send quantum information adequately accurately and over relatively long distances.

A common misunderstanding is that entanglement could be used to send information from one point to another instantaneously. This is not possible because the measurement results of the individual particles are random and specified on a predetermined orthonormal basis of the eigenspace. However, the entanglement between qubits makes a quantum computer more powerful than a classical computer.

2.4 Measurement

The *measurement* of a quantum system corresponds to transforming the quantum information into classical information. For example, measuring a qubit typically corresponds to reading out a classical bit, i.e., whether the qubit is 0 or 1. A central principle of quantum mechanics is that measurement outcomes are probabilistic (Postulate 3).

Given a state vector $|\psi\rangle \in \mathbb{C}^n$ we consider a measurement of the observable A. Because observables are Hermitian, their eigenvalues $\lambda_0, \dots, \lambda_{n-1}$ are real values. Then it is possible to express A as:

$$A = \sum_{j=0}^{n-1} \lambda_j |\phi_j\rangle \langle\phi_j| \,. \tag{2.29}$$

Thus the probability p_j of obtaining the outcome λ_j is given by

$$p_j = |\langle\psi|\phi_j\rangle|^2. \tag{2.30}$$

The state vector immediately after the measurement is given by

$$|\psi_j\rangle = \frac{1}{\sqrt{p_j}} \langle\phi_j|\psi\rangle |\phi_j\rangle \,, \tag{2.31}$$

and the actual value that is readout from the quantum device is not the computational basis state itself, but the correspondent eigenvalue of A. Thus, the final result is the eigenvalue, which means that the measurement "collapses" the state vector into one of the eigenvectors of A[5]. As a consequence, a single measurement provides only partial information about the state vector and computing the expectation value of measurement is extremely important, since it allows to retrieve a more complete information about the state vector. In formula, the expectation value of A for $|\psi_j\rangle$ is

[5]Notice that we assumed that none of the eigenvalues is degenerate. This case would require mild modification.

the following:

$$\langle A \rangle = \langle \psi | A | \psi \rangle . \tag{2.32}$$

For example, if we measure a Pauli operator σ_z on a system initially prepared in $|0\rangle$, then the following expectation value is observed:

$$\langle 0 | \sigma_z | 0 \rangle = 1. \tag{2.33}$$

In case of $|1\rangle$:

$$\langle 1 | \sigma_z | 1 \rangle = -1. \tag{2.34}$$

In quantum physics, observables manifest as linear operators on a Hilbert space representing the state space of quantum states. The eigenvalues of observables are the real numbers observed when measuring a quantum system. Hence, the observables in quantum mechanics assign real numbers to outcomes of particular measurements, corresponding to the eigenvalue of the operator with respect to the system. The effect of the measurement is that the new state is exactly the outcome of the measurement, implying that it is impossible to collect any additional information about the system, once it is measured.

The entanglement becomes clear from the results of measurements. The outcome of the measurements on the individual qubits could be $|0\rangle$ or $|1\rangle$. However, the outcome of the measurement on one qubit is always correlated to the measurement on the other qubit, if the two are entangled. This is always the case, even if the particles are separated from each other by a considerable distance.

2.5 Quantum Algorithms

Qubits and quantum gates represent the basic elements needed for practical quantum algorithms. Considering the *Gate-model*[6], a quantum algorithm consists of three basic steps: the first one is the definition of a sequence of operation to transform a set of input qubits from all-zero state to a quantum representation of a classical input data. This step is known as *state preparation*, and it is extremely important when comparing a quantum routine with its classical counterpart. Indeed, state preparation

[6]this terminology refers to quantum states as input/output of the quantum system and quantum gate refers to a unitary operator.

is an additional cost specific of quantum computation. Once data are encoded in our quantum system, the second step consists of applying the quantum gates of interest to obtain the desired quantum state. Finally, the third step regards the measurement of the quantum system and readout a classically interpretable result.

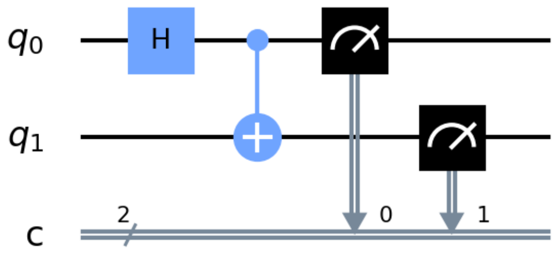


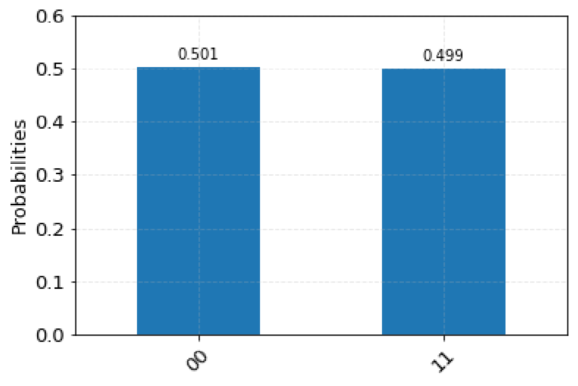
Fig. 2.2 Quantum circuit for Bell state. First, the top qubit q_0 is initialised into uniform superposition. Then, q_0 serves as control qubit (indicated with a dot) of the CNOT. The bottom qubit, labelled as q_1, is the target qubit (indicated as $\oplus$) and is inverted if the control qubit is 1.

An example is the quantum circuit to produce the Bell state given in Figure 2.2 which allows to create a fully entangled two-qubit quantum system. Starting from two qubits to the all-zero state, the quantum operations to produce the Bell state are the following:

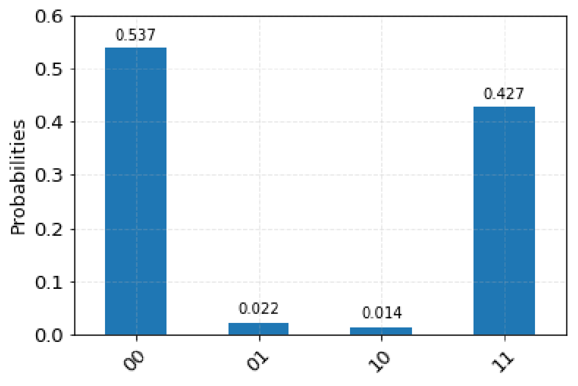
$$(\mathrm{CNOT}_{12})(H \otimes I)\,|0\rangle \otimes |0\rangle = (\mathrm{CNOT}_{12})\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{2.35}$$

First, the Hadamard gate acts on the topmost qubit changing the state from $|0\rangle$ to the uniform superposition of $|0\rangle$ and $|1\rangle$. Then CNOT_{12} acts on both qubits. The blue dot on the first qubit identifies that this qubit is used as control. The $\oplus$ symbol on the second qubit implies that this qubit is the target of the NOT gate (controlled by the state of the first qubit). The measurement of a qubit is also denoted by a special gate with a meter symbol on it (Figure 2.2). The presence of this gate on a qubit means that the qubit must be measured in the computational basis.

Thus, any quantum algorithm has to be defined in terms of qubits and quantum gates and quantum circuit to be implemented in a real quantum device. In the circuit representation, qubits are represented by horizontal lines and gates are then drawn on the qubits they act on. This is done in sequence from left to right. The initial state of the qubit is denoted at the beginning of each of the qubit lines. Importantly, when we



(a) QASM simulator



(b) IBMQ santiago

Fig. 2.3 On the left the execution of the quantum circuit in Figure 2.2 using the QASM simulator, which simulates an ideal, fault-tolerant quantum computer. The distributions of the measurement results is almost perfectly uniform between 00 and 11. The slight fluctuations in the results depend on finite sampling of the output statevector. On the right the same circuit is executed on a real device, the *IBMQ_santiago* (QV32). In this case, the basis states 01 and 10 are also readout due to decoherence that occurs on a real quantum system.

write down a mathematical expression for the circuit, the gates are written down from right to left in the order of their application.

2.6 Quantum Computational Complexity

Computational complexity theory aims to classify the difficulty of computational problems in both classical and quantum computation. A complexity class is a collection of computational problems, that share some common characteristic regarding the computational resources needed to solve them. Two important complexity classes are **P**, which is the class of computational problems that can be solved "efficiently"on a classical computer[7], and **NP** that is the class of problems whose solutions can be checked on a classical computer[8]. Given these definitions, it is straightforward that **P** is a subset of **NP** since the ability to solve an assigned problem implies the capacity to check potential solutions. However, what is not clear is whether there are problems in **NP** that are not in **P** and this is one of the most important problems in computer science (Jaffe, 2006). There exists an important subclass of the **NP** known

[7]P stands for polynomial time, which means that the complexity of the algorithm is polynomial in the input size using a deterministic Turing machine

[8]NP stands for non-deterministic polynomial time

as **NP**-complete, that is important since any algorithm to solve a given **NP**-complete problem can be adapted to solve any other problem in **NP**, with small overhead. This means that, in some sense, any **NP**-complete problem is at least as hard as all other problems in **NP**. This means that if **P**=**NP** it follows that **NP**-complete problems can be efficiently solved on a classical computer.

There is another relevant complexity class called *Buonded-error Probabilistic Polynomial time* (**BPP**), that describes decision problems that can be solved in polynomial time by a *probabilistic* Turing machine, considering that the found solution could be incorrect [9]. A probabilistic Turing machine chooses the next step according to a given probability distribution and not deterministically.

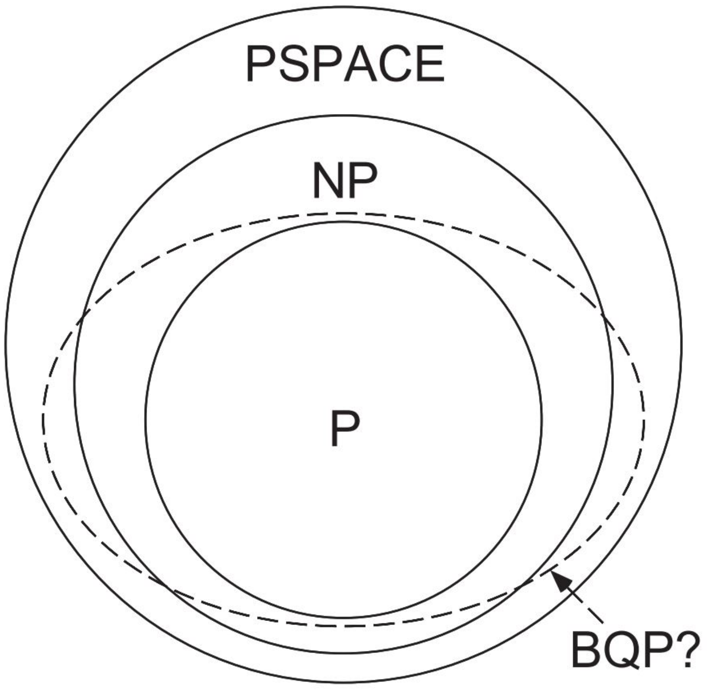


Fig. 2.4 Complexity classes (Nielsen and Chuang, 2011).

Although, it is not yet been proven whether $\mathbf{P} \subset \mathbf{BPP}$, it is conjectured that $\mathbf{P} = \mathbf{BPP}$ (Impagliazzo and Wigderson, 1997). Quantum computing introduces a new complexity class, the *Bounded-error Quantum Polynomial time* (**BQP**), that represents the class of problems solvable in polynomial time by an innately probabilistic quantum Turing machine with the same error constraint defined for **BPP**. Unlike **BPP**, it is presumed that $\mathbf{P} \subset \mathbf{BQP}$ which means that quantum computers are, at least in principle, capable of solving some problems in polynomial time that cannot be solved

[9]In BPP, the probability that the answer is correct must be at least $\frac{2}{3}$

efficiently using a classical Turing machine. The supposed relationships between all of these complexity classes are depicted in Figure 2.4.

2.6.1 Time and Gate Complexity

Different types of complexity must be considered when looking at computational complexity theory. For example, one of the most famous quantum routines is Grover's algorithm (Grover, 1996) that performs a search over an unordered set of $N = 2^n$ items to find a specific element. The best classical algorithm to solve this task requires $\mathcal{O}(N)$ steps, while Grover's algorithm performs the search on a quantum computer in only $\mathcal{O}\left(\sqrt{N}\right)$ operations, which corresponds to a quadratic speedup. However, Grover's algorithm assumes that it is given black-box access to some function f, where the goal is to answer some question about f itself. The quadratic speedup is provided in terms of *query complexity* (Ambainis, 2017), instead of *time complexity*. The idea of query complexity is beyond the scope of this dissertation, but the concept of time complexity is essential to understand the advantages of the thesis proposals, and it is discussed below.

The time complexity of an algorithm is the number of necessary "steps" to transform a given input encoded as a binary string (bits) into the desired output. These steps are local operations limited to a small number of bits (two or three) and the overall cost is usually specified in terms of various well-defined resources that are evaluated in terms of some designated elementary operations and memory usage.

The same approach can be employed to evaluate the time complexity of quantum algorithms. Based on quantum gate-model, an algorithm needs to be defined together with the correspondent quantum circuit, whose time complexity can be defined as the depth of the circuit in terms of quantum gates. The number of gates corresponds to the number of time steps required for the quantum algorithm to be executed on the quantum hardware. Thus, it is necessary to define a set of elementary gates whose cost is equal to 1 in order to compute the total cost of an algorithm. Since the actual implementation of quantum gates is strictly related to the specific quantum hardware, usually the assumption is that any quantum gate that acts on 1 or 2 qubits has a unit cost. Notably, the gate complexity usually refers to an error-corrected quantum system, which means that the realisation of the quantum circuit requires high fidelity, high precision, and high-level control. Hence, the comparison of time complexity between classical and quantum algorithms makes sense only in the context of fault-tolerant quantum computation. Instead, when dealing with NISQ device, the idea is to build a small circuit to solve a specific task that has some advantage with respect to the classical counterpart.

In addition to all these considerations, a further level of complexity exists when the comparison concerns machine learning algorithms. In fact, other than the input size (the size of the training set usually gives), when considering quantum machine algorithms there are other computational considerations related to the *type* of the algorithm in use that depends on a set of parameters. The way an algorithm works and the number of its parameters is a fundamental part of the computational cost, that dominates the correspondent overall time complexity.

In this thesis, a general quantum framework for machine learning is proposed. Such framework is able to reproduce many classical machine learning models with a proper definition of a set of quantum operators. When considering specific algorithms, the use of the quantum counterpart can improve the time complexity sensitively with respect to the parameters of the algorithm, in some case, even exponentially.

Part II

Quantum Algorithms for Machine Learning

Chapter 3

Quantum Machine Learning

Quantum Machine Learning (QML) joins together machine learning (ML) and quantum computing (QC) to improve classical ML methods and overcome their biggest weaknesses. This chapter reviews the main contributions in the field of QML and provides the basic elements to compare classical models with their quantum counterparts. The chapter is organised as follows. Section 3.1 provides an overview of classical methods with a focus on supervised learning and the notion of the computational complexity of learning. Section 3.2 introduces the field of QML, intended as the enhancement of classical methods using quantum computation. Section 3.3 discusses the main QML algorithms proposed in the literature, distinguishing between full-coherent protocols and NISQ algorithms. Section 3.4 discusses the problem of state preparation. More than in other cases, the encoding strategy for data impacts the time complexity of any QML algorithm and can destroy the potential advantage provided by the use of a quantum algorithm. Thus, Section 3.5 describes all the problems related to the use of quantum computation to solve ML tasks. Finally, Section 3.6 summarises the contribution of the thesis in the context of QML.

3.1 Overview on Machine Learning

Machine learning (ML) is the science of making computers to act without being explicitly programmed, but using a learner, an algorithm able to learn patterns from a set of data. In particular, according to Mitchell et al. (1997) "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*". We can imagine a very wide variety of experiences E, tasks T, and performance measures P as well as the application fields of ML itself. In the last years, ML made an impact in many fields, such as self-driving cars, speech recognition, predictive maintenance, and a vastly improved understanding of the human genome. In practice, ML allows tackling tasks that are too difficult to solve with fixed programs written and designed by human beings.

There are mainly three categories of learning:

- **Supervised learning:** the goal is to find a useful approximation $\hat{f}(x)$ to the function $f(x)$ that underlies the predictive relationship between the inputs and outputs. For instance, a supervised learning task is the classification problem: the algorithm is required to learn a function which maps a vector of features into one of several classes by looking at several input-output examples.

- **Unsupervised learning:** only the features are observed, not the outcome, and the task is to describe how the data are organised or can be represented instead of computing specific information. In a sense, unsupervised learning methods regard finding patterns in the data, excluding what can be considered pure unstructured noise. Two examples of unsupervised learning are clustering and dimensionality reduction.

- **Reinforcement learning:** the algorithm learns a policy of how to act given an observation of the world. Every action has some impact on the environment, which in turn provides feedback that guide the learning algorithm. These types of methods focus on finding a balance between exploration of uncharted territory and exploitation of current knowledge.

Any of these methods aims to find a function f that is able to produce a reliable outcome Y for a given input set X[1]. From this point, only the supervised approach is considered where, given a set of input-output pairs, the algorithm tries to estimate an explicit target variable Y given the input X.

3.1.1 Empirical Risk Minimization

Supervised methods aim to learn an unknown target function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are respectively the sets of features and the target variable. Consider $\mathcal{D}$ as the probability distribution over the set of points $\mathcal{X}$. Given a specific learning algorithm, it is associated to a hypothesis class $\mathcal{H}$ of functions $h : \mathcal{X} \to \mathcal{Y}$. $\mathcal{X}$ and $\mathcal{H}$ consist of

[1]Notice that in case of reinforcement learning the outcome is the action to make at time t and the input also contains the action performed at the time $t - 1$.

objects with bounded computational power (e.g. all neural networks with a specific and possibly limited depth and the number of nodes, or all decision trees of at most a certain depth). It is assumed that each $h \in \mathcal{H}$ has a succinct description and that it is possible to evaluate a given h on a given $x \in \mathcal{X}$.

Thus the goal of the learner[2] is to minimise the *generalisation error* of h with respect to the target function f:

$$err(h, f, \mathcal{D}) = \Pr_{x \sim \mathcal{D}} \left[h(x) \neq f(x) \right]. \tag{3.1}$$

Generalisation error, also known as *Expected Prediction Error*(EPE) is minimised by the learner over the class $\mathcal{H}$. However, since the learner does not know what $\mathcal{D}$ and f are, the generalisation error is not directly available.

In practice, the learner has access to a limited training set $S = \{(x_i, y_i)\}_{i=1,\ldots,m}$, where the points x_i are presumed *independent and identically distributed*, and generated according to the unknown distribution $\mathcal{D}$ on $\mathcal{X}$. The target variable can be computed as $f(x_i) = y_i$ and f represents the function to learn. A useful notion of error is the training error which is computed according to the instances in S:

$$\hat{err}(h, f, \mathcal{D}) = \Pr_{x,y \in S} \left[h(x) \neq f(x) \right]. \tag{3.2}$$

This is also known as empirical risk. Since the training sample is a snapshot of the $\mathcal{D}$, it makes sense to search for solutions that work well on the training data. This learning paradigm which aims to find a predictor h that minimises $\hat{err}(h, f, \mathcal{D})$, in the hope that it will be able to generalise over the unknown $\mathcal{D}$, is called *Empirical Risk Minimisation* (ERM). Furthermore, it is possible to show that the *generalisation error* can be decomposed in into three components: noise, bias and variance (Hastie et al., 2001). The *noise* component, also known as *irreducible error*, is the variance of the target variable around its true mean. This error is due to the intrinsic uncertainty of S, and cannot be avoided no matter how well h works. The *bias*, instead, is linked to the particular learning technique h adopted, and it measures how well the method suits the problem. Finally, the *variance* component measures the variability of the learning method around its expected value. In light of this, in order to improve the performance of any ML technique, one has to try to reduce one or more of these components.

[2]The definition refers to classification, but it can be easily generalised to regression.

3.1.2 Computational Complexity of Learning

To evaluate the performance of an algorithm two different factors need to be considered: *learnability* and *time complexity*.

The learnability concerns finding a learner that is able to achieve a good generalisation error with the smallest sample size. This approach regards the framework of *Probably Approximately Correct* (PAC) learning that provides a mathematical analysis of machine learning. In particular, the learner receives samples and must select a generalisation function from a certain class of possible functions. The goal is that, with high probability, the selected function will have low generalisation error. The PAC learning framework is a distribution-free setting, so the idea is to design a learner that works well for every $\mathcal{D}$ in the sense of outputting a hypothesis with low generalisation error.

The time complexity regards the ability of an algorithm to be executed using a number of steps that relies on the input size and the parameters of the algorithm itself. In fact, each possible parametrisation gives rise to a different hypothesis and the computational effort required to train a specific model can be sensitively different when considering different parametrisations. This makes ML tasks different from other typical problems in computer science, in which, time complexity can be entirely expressed as a function of the input.

The big-$\mathcal{O}$ notation is usually employed to express the time complexity, that describe the worst-case scenario as a function of the execution time required by the algorithm in terms of the input size and the parameters.

3.2 Introduction to Quantum Machine Learning

In recent years, machine learning algorithms achieved remarkable successes in many applications, but this revolution is beginning to face increasing challenges. With the ever-increasing size of datasets and Moore's law coming to an end, a point where the current computational tools will no longer be sufficient will be reached soon. Although tailored hardware architectures can significantly improve performance (e.g. graphics processing units and tensor processing units), they do not offer a structural solution to the problem.

An opportunity for ML to overcome its limitations is to leverage quantum computation that exploiting quantum effects can efficiently solve selected problems (Hallgren, 2007; Shor, 1999; Van Dam et al., 2006) that cannot be solved on classical machines. The speed-up of a quantum algorithm usually is an effect of quantum parallelism achieved through the superposition of qubits, that allows the system to exist in all of its possible states and to evaluate a function on many inputs simultaneously.

The intersection between machine learning and quantum computation is known as Quantum Machine Learning (QML); this term is usually employed to denote different paths of research, such as using machine learning techniques to analyse the quantum processes or the design of classical machine learning algorithms inspired by quantum structures. We refer to QML to describe learning models that make use of quantum resources. There are two ways in which ML and QC can be combined: one approach is to run the learning process predominantly with classical computation and only one or multiple subroutines requiring access to a quantum computer. In this case, the potential speed-up comes directly from a part of the process. However, the protocols within this category need to carefully consider the limitations given by read out the data of a quantum computational process.

Another approach concerns methods that use only quantum computations in which the algorithm contains no sub-routine that are performed on classical computers. Likewise, the input and output data can be classical, but the whole computation is performed end-to-end on a quantum device.

The QC community developed a rich collection of quantum algorithms for basic linear algebra subroutines, such as *matrix multiplication*, *matrix inversion*, *singular value decomposition* (Biamonte et al., 2017; d'Alessandro, 2007; Gillespie, 1980; Haag and Kastler, 1964; Nakahara and Ohmi, 2008) that can be used to address several ML tasks. Furthermore, many quantum routines that leverage hybrid quantum-classical computation have been proposed.

3.3 Related works

In QML, algorithms are developed by setting classical algorithms, or their expensive subroutines to run on a potential quantum computer. The expectation is that quantum devices can help to process and analyse the growing amounts of global information.

To date, there is no comprehensive theory of *quantum learning* yet. A theory of quantum learning would refer to methods of quantum information processing that learn input-output relationships from training input, either for the optimisation of system parameters or to find a quantum decision function. There are many open questions of how a quantum algorithm for learning should look like. Thus, in order to exploit the properties of quantum mechanics is necessary finding standard quantum ways of representing data with proper and efficient state preparation routines.

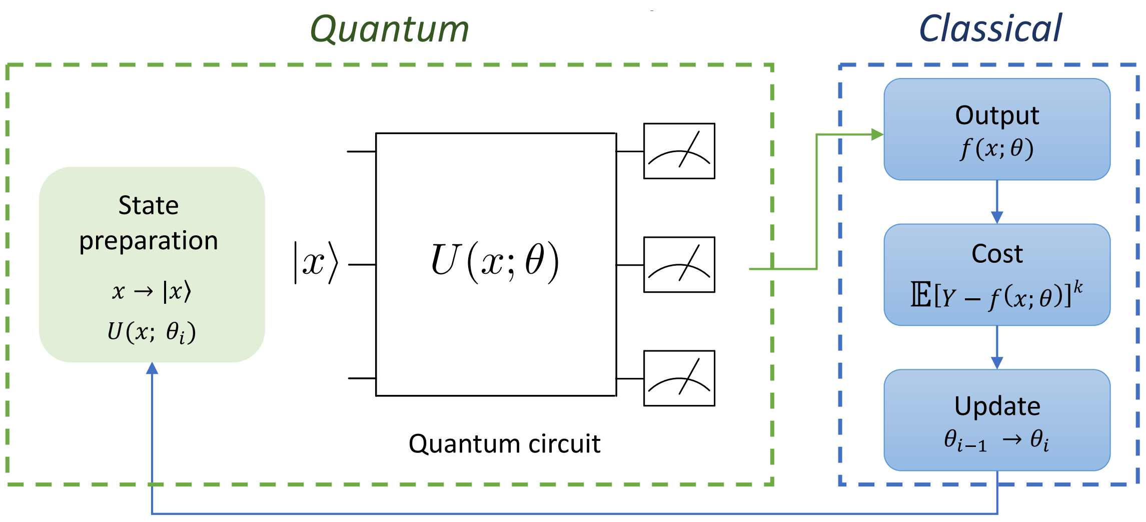


Fig. 3.1 Scheme of a hybrid quantum-classical algorithm for supervised learning. The quantum variational circuit is depicted in green, while the classical component is represented in blue.

In the following sections, the most relevant QML algorithms presented in the literature are reviewed.

3.3.1 Quantum Variational Algorithms

Quantum variational algorithms (Moll et al., 2018; Wecker et al., 2015) represent the most promising attempt to leverage NISQ technology in the context of hybrid quantum-classical computation. They are designed to tackle optimisation problems using both classical and quantum resources for supervised learning tasks where the latter component is referred to as variational circuit, and it presents three ingredients: *i*) a parametrised quantum circuit $U(x;\theta)$, *ii*) a quantum output $f(x;\theta)$ and *iii*) an updating rule for the parameters θ. The general hybrid approach is illustrated in Figure 3.1.

The data, x, are initially preprocessed on a classical device to determine the input quantum state. The quantum hardware then prepares a quantum state $|x\rangle$ and computes $U(x;\theta)$ with randomly initialised parameters θ. After multiple executions of $U(x;\theta)$, the classical component post-processes the measurements and generates a prediction $f(x; \theta)$. Finally, the parameters are updated, and the whole cycle is run multiple times in a closed loop between the classical and quantum hardware.

Interestingly, the first practical demonstration of quantum advantage over classical supercomputers is related precisely to variational algorithms (Peruzzo et al., 2014). Other applications related to Machine Learning (ML) problems are also explored (Biamonte et al., 2017; Ristè et al., 2017). More recently, Schuld et al. (2018) proposed a low-depth variational algorithm for classification. The strengths of this approach are two-fold. On one side, the possibility of learning gate parameters enables the adaptation of the architecture for different use cases. On the other hand, the choice of amplitude encoding allows obtaining model predictions with a single-qubit measurement. Importantly, simulations on standard benchmark datasets showed good performances, with the advantage of requiring fewer parameters than classical alternatives. Another similar framework which discusses the possibility to employ parametrised quantum circuit as ML model is discussed by Benedetti et al. (2019)

Although variational algorithms promise good results in many applications, such as quantum simulation, optimisation, and machine learning, the exponential dimension of Hilbert space and the gradient estimation complexity make them unsuitable for running on more than a few qubits. In fact, for a wide class of reasonable parameterised quantum circuits, the probability that the gradient along any reasonable direction is non-zero is exponentially small as a function of the number of qubits. This is known as the problem of Barren Plateaus (McClean et al., 2018), and the solution to this problem needs to be investigated.

3.3.2 Quantum Artificial Neural Networks

Artificial Neural Networks (NNs) are ML algorithms, whose structure is inspired by biological neural networks, where the inputs are the features of the dataset and the output provides the estimate for a given target variable. Training a NN can be done by selecting weight parameters and an activation function encoding a specific input-output relation. The power of NNs lies in their ability to learn their weights from training data; a fact that neuroscientists believe is the basic principle of how our brain processes information (Dayan and Abbott, 2005). The most popular type of NNs are the feed-forward NNs in which neurons are arranged in layers, and each layer feeds its values into the next layer. Input is presented to a feed-forward neural network by initialising the input layer, and after each layer, successively updates its nodes to produce the final output. Feed-forward NNs often use non-linear activation.

In order to achieve the desired generalisation, the network is initialised with training vectors, the output is compared to the correct output, and the weights adjusted through gradient descent in order to minimise the classification error; this procedure is called *backpropagation* (Rumelhart et al., 1986). A challenge for pattern classification is the computational cost for the backpropagation algorithm, even when considering improved training methods such as deep learning (Hinton et al., 2006).

There are several proposals for quantum versions of neural networks (Faber and Giraldi, 2002; Gupta and Zia, 2001; Schuld et al., 2014, 2015; Schützhold, 2003; Trugenberger, 2002). However, the main challenge remains the implementation in a quantum circuit since the learning algorithms for these models break the postulates of quantum mechanics by the use of non-linear or non-unitary operations. Usually, most of the work considers the so-called Hopfield networks, which are powerful for the related task of associative memory that is derived from neuroscience rather than machine learning (Behrman et al., 1999; Faber and Giraldi, 2002; Tóth et al., 1996).

Recently, many proposals of Quantum Neural Networks (QNNs) that leverage hybrid quantum-classical computation have been presented in the literature. A concrete implementation in near-term processors is illustrated by Tacchino et al. (2019), where the authors introduced a model for binary classification using a modified version of the perceptron updating rule. A key characteristic of their architecture is the theoretical exponential advantage in storage resources over classical alternatives. This constitutes the first step towards the efficient implementation of QNNs on near-term quantum processing hardware.

Another possible approach for QNNs is discussed by Farhi and Neven (2018), where the model is potentially able to represent the label of any Boolean function of n bits. The authors introduce parameter-dependent unitaries that can be adapted by supervised learning of labelled data.

All the proposed approaches do not reproduce precisely the quantum counterpart of the classical NNs, instead, they are quantum algorithm inspired to classical NNs, where parametrised quantum gates implement each layer. To date, there are no trainable quantum algorithms that efficiently reproduce a quantum state encoding the output of a classical NN.

3.3.3 Quantum Algorithms for Ensemble Learning

Ensemble methods are machine learning models that combine the decisions from multiple models to improve the overall performance. The combined prediction is usually calculated by averaging the single models with a set of specific weights.

Recently, the idea of a quantum ensemble has been investigated by Schuld and Petruccione (2018a). In this case, the construction of the ensemble corresponds to three different stages: *(i)* a state preparation routine, *(ii)* the evaluation in parallel of the quantum classifiers and *(iii)* the access to the combined decision. This approach is based on Bayesian Model Averaging (BMA) that exploits many models whose parameters are fixed so as to span a large part of parameters domain. The strength of this approach is that the individual classifiers do not have to be trained. However, the algorithm assumes two oracles whose form is not precisely defined in terms of quantum gates. Furthermore, the BMA approach is not very used in ML because of limited performance in real-world applications (Domingos, 2000, 1997). In fact, it has been shown that models combination works better by enriching the space of hypotheses, not by approximating a Bayesian model average. On the other side, classical ensemble methods (e.g. Random Forest) generate a collection of complementary hypotheses whose predictions are compatible with the data. These hypotheses are induced by fitting the same model under different training conditions.

Another QML algorithm based on ensemble methods is the idea of *Quantum Boosting* investigated by Arunachalam and Maity (2020). In this case, the authors suppose a weak learner $\mathcal{A}$ and try to improve its performance by simulating adaptive boosting procedure (Freund et al., 1999) that allows converting a weak learning algorithm to a strong one. This is done by achieving a quadratic improvement over classical AdaBoost. The main limitation of quantum boosting is the ability to prepare efficiently multiple copies of the same quantum states that encode the training set. Also, it assumes to execute the Quantum Phase Estimation algorithm that is a full-coherent protocol requiring a fault-tolerant quantum computer to be executed.

3.3.4 Quantum k-Nearest Neighbour

A very popular method for pattern classification is the k-nearest neighbour algorithm. The idea behind this method is to choose the class for the new input based on the frequency distribution of often amongst its k nearest neighbour. The assumption is that close feature vectors encode similar examples, which is true for many real-world applications. Distance measures usually employed are the inner product, the Euclidean or the Hamming distance. The parameter k is a hyperparameter of the algorithm, and it can influence the results significantly.

Some efforts to translate this algorithm into its quantum version have been made with particular focus on the efficient evaluation of a classical distance through a quantum algorithm. Aïmeur et al. (2006) introduce the idea of using the overlap (or fidelity) of two quantum states as a similarity measure that can be measured by using a well-known quantum routine, the swap test (Buhrman et al., 2001). Wiebe et al. (2018) also give a scheme for a (weighted) nearest-centroid algorithm based on the Euclidian distance evaluated by well-known algorithms from the toolbox of quantum information, the amplitude estimation algorithm (Brassard et al., 2002; Durr and Hoyer, 1996). Although the evaluation of the distance using a quantum algorithm seems to be reasonable, it is not entirely clear whether these approaches would perform better than their classical counterpart even considering a fault-tolerant quantum device. In fact, repeated state preparation and measurements executions can cancel out the quantum advantages provided by the efficient evaluation of the distance.

3.3.5 Quantum Algorithms for Clustering

Clustering is an unsupervised technique that aims of dividing a set of feature vectors into a fixed number of clusters. It does not require a training set where the target variable is known, or prior examples to generalise the model, but rather it extracts information on structural characteristics from a set of data.

In classical ML, the standard example for clustering is the k-means algorithm, where each feature vector is assigned to its closest current centroid vector that is representative of a specific cluster. The final centroids are calculated after a long iterative process whose complexity depends on the distance metric used, the size of the dataset and the number of clusters.

A quantum proposal for clustering is provided by Rebentrost et al. (2014) which present a full coherent protocol routine to perform cluster analysis. Durr and Hoyer (1996) describe an algorithm to find the smallest distance between a point of a cluster and its centroid. In this case, an oracle is employed, and its definition in terms of quantum gates is not provided. Also, this approach largely depends on a considerable amount of quantum resources. For an in-depth discussion about the quantum version of unsupervised methods, see Aïmeur et al. (2013).

3.3.6 Quantum Linear Models

The basic idea of the linear algebra approach in QML is to use a quantum system for linear algebra calculus, where Hamiltonian of the system represents the design matrix via dynamic encoding.

In classical ML, the matrices involved in the computation are usually constructed from the training set; this means that the dimension of the problem grows with the number of features p and the number of the training vectors N. There are many algorithms which operate using hyperplanes to separate or fit data, therefore, finding the *optimal* hyperplane for a certain problem is one of the most important tasks in ML (Bishop, 2006; Fan et al., 2008). From a statistical point of view, this problem can be reformulated fitting a Linear Regression (LR) that uses least square optimisation, considering the correspondent system of normal equations (MATH, 2007). This approach allows, in theory, to exploit the exponential speed up provided by HHL algorithm (Harrow et al., 2009)

The problem of HHL is that the solution of the linear system is encoded into a quantum state in amplitude encoding, so given the proposed approach, it is necessary to extract the solution and use it in a classical computer. This can be done either reading out the parameters via quantum tomography (D'Ariano et al., 2003) or simulate many times the execution of the algorithm to get a normalised estimation of the parameters[3].

3.3.7 Kernel Methods and Quantum Support Vector Machine

A support vector machine (SVM) is an algorithm that maps input vectors into feature space in such a way that the target variable in that space is easy to compute (Sánchez A, 2003). It operates by constructing the optimal hyperplane dividing the two sets (in case of binary classification), in a higher-dimensional kernel space. The objective function of an SVM can be reformulated as a quadratic programming problem, which can be solved in time proportional to $\mathcal{O}(log(\epsilon^{-1})poly(p, N))$, with p the dimension of the feature space, N the number of training vectors, and ϵ the accuracy. Rebentrost et al. (2014) propose a full-coherent protocol for quantum SVM whose time complexity is $\mathcal{O}(log(pN))$ in both training and testing stages.

While classical SVM do not directly lead to a matrix inversion problem, a version called *least-squares support vector machines* (De Brabanter et al., 2002) transforms the convex quadratic optimisation problem into a least squares problem. In this case, the problem of fitting SVM model becomes:

$$K \begin{pmatrix} w_0 \\ \gamma \end{pmatrix} = \begin{pmatrix} 0 \\ y \end{pmatrix}, \tag{3.3}$$

where K is the kernel Gram matrix (Drineas and Mahoney, 2005), $\gamma = (\gamma_1, \ldots, \gamma_M)^T$ is the set of the Lagrangian parameters, and $w_0 \in \mathbb{R}$ the scalar bias. To obtain the γ_i, one has to solve a linear system by inverting the kernel matrix.

[3]HHL will be discussed in details in Chapter 7.

In quantum setting, one can use the HHL algorithm to apply K^{-1} to quantum state which encodes y, thus computing the quantum state encoding the solution $K^{-1}y$. Once the solution is obtained, it is possible to use it into two different ways: using interference to classify new input or extracting information about parameters via quantum tomography or via multiple executions of the circuit.

Schuld and Killoran (2019) recently explored the relation between quantum states and feature maps. In particular, the authors illustrate that a critical element in both quantum computing and kernel methods is performing computations in a high dimensional (possibly infinite) Hilbert space manipulating the inputs features. In fact, encoding a classical vector x into a quantum state $|\phi(x)\rangle$ is equivalent to define a feature mapping, where ϕ maps classical vectors to the Hilbert space associated with a system of qubits. For a comprehensive review of kernel methods in quantum computing see Mengoni and Di Pierro (2019)

Despite the theoretical proposal, a full quantum algorithm which implements SVM does not exist yet. Usually, the algorithm refers to a specific operation needed to SVM rather than its use dealing with real data. However, the huge computational effort to train a classical SVM has limited its use in the past; therefore, the possibility to define its quantum version exploiting quantum resources may allow extending its use for several machine learning applications.

3.4 State Preparation in QML

Any quantum algorithm for data analysis must address (at least) three issues: the encoding strategy for classical data into a quantum representation, the quantum operation to apply to the input state and the measurement strategy to retrieve the information of interest (which qubits? which computational basis?). Thus, the ability to encode a real vector into a quantum state is mandatory for QML.

There are several ways in which state preparation can be performed, and the nature of data itself influences the proper representation for a given problem. The most used approaches are *basis encoding* and *amplitude encoding*. The first one associates a computational basis state of a n-qubit system with n bits. Amplitude encoding, instead, stores classical vectors as the amplitudes of a quantum state. Usually, state preparation routines scale linearly in the size of the input, and any quantum algorithm that processes classical data needs to consider this additional cost. An overview of the four encoding methods is provided in Table 3.1.

Encoding	**Number of qubits**	**Runtime**	**Input features**
Basis	N	$\mathcal{O}(Np)$	Binary
Amplitude	$log(Np)$	$\mathcal{O}(Np)$	Continuous
Hamiltonian	$log(Np)$	$\mathcal{O}(Np)$	Continuous

Table 3.1 Strategies of data encoding in a quantum system for a dataset of N training points and p features: *Basis* uses a binary representation of numbers, and the computation acts in parallel on all bit sequences in superposition. *Amplitude* associates classical information (e.g. real vector) with quantum amplitudes. *Hamiltonian* associates the Hamiltonian of a system with a matrix that represents some meaningful transformations of the matrix that represents the original data. For more details see Schuld and Petruccione (2018b).

Several quantum algorithms presented in the literature assume to employ a quantum random access memory (QRAM) (Giovannetti et al., 2008b) which is a theoretical device that can store classical vectors as the amplitudes of a quantum state (*Amplitude Encoding*). Assuming to process N p-dimensional classical vectors, QRAM encodes them in superposition into $log(Np)$ qubits in $\mathcal{O}(poly\,(log(Np)))$ steps (Arunachalam et al., 2015). Although possible physical architectures for the QRAM have been discussed (Giovannetti et al., 2008a), its actual feasibility is still not entirely clear, and many caveats remain.

An alternative state preparation approach has been proposed by Mottonen et al. (2004). In this case, the authors consider the reverse problem to map an arbitrary state $|\psi\rangle$ to the ground $|0\dots0\rangle$. Once we get the circuit to map $|\psi\rangle$ in $|0\dots0\rangle$ it is possible to invert all of the operations and apply them in the reversed order. In this case, some preprocessing is needed, since it is necessary to convert the real values in a series of angles to apply different controlled rotations.

Recently, a hybrid quantum-classical algorithm for efficient, approximate quantum state loading has been proposed (Zoufal et al., 2019). The proposed approach is referred to as quantum Generative Adversarial Networks (qGANs) that by leveraging the interplay of a variational quantum circuit, and a classical neural network, is able to learn a probability distribution underlying the data samples and load it into a quantum state. This procedure requires $\mathcal{O}(poly(n))$ gates (n is the total number of qubits) and potentially enables the efficient implementation of a generic probability distribution into quantum states. Although this approach potentially provides a structural solution to the problem of state preparation, it is unclear if would work on real datasets. In fact, the experiments about qGAN usually are based on univariate, well-known probability distributions, while the distributions underlying real-world datasets are usually much more complex.

In general, the cost of a state preparation routine seems unavoidable per algorithm run, even when performing a set of algorithms with an identical input state. Thus, the way data is encoded into a quantum system is part of the algorithm and may be a crucial part of the complexity.

In this dissertation, we will not address the problem of state preparation, and the method employed for experiments is the one proposed by Mottonen et al. (2004). However, the proposed quantum framework allows evaluating different transformations of the same input quantum state limiting the number of state preparation routines.

3.5 The Quest of Quantum Machine Learning

In QML, the algorithms require a state preparation routine for information encoding and a readout step, that in general are non-trivial procedures. Addressing these two tasks is extremely important to enable QML for real-world application, since they can easily predominate the complexity of a quantum algorithm and, thereby, reduce the potential quantum advantage. Thus, in designing a new quantum algorithm is necessary to limit as much as possible, the number of state preparation routines and the number of qubits to measure. Also, in order to be a credible alternative to classical ML, a quantum algorithm needs to be *better* in terms of either *learnability* of intractable patterns or overall time computational complexity. The first case relates to the capability to exploit the mapping into a larger Hilbert space. It seems reasonable to leverage this space to discover patterns on data that are impossible to detect using classical representation. In this regard, it is essential to extract the relevant information about those patterns through a limited number of measurements; indeed the dimension of the Hilbert space grows exponentially and obtaining a complete representation of it would be inefficient even with a few tens of qubits.

When considering the time complexity instead, other than the measurement operations, it is necessary to consider the impact of state preparation routine that is at least linear in the size of the input (Section 3.4). Therefore, it is necessary to trade the potential advantage provided by quantum computation in terms of computational capabilities with the overhead to perform operations that are not contemplated when performing classical ML methods.

To date, considering actual quantum technology and the proposed algorithms in the literature, there is no convincing QML algorithm that can have consistent advantage with respect to its classical ML alternative. However, the field of QML is moving extremely fast, and the belief is that many applications suitable for QML are right around the corner.

3.6 Research Contribution

The main contribution of the thesis is a general framework to perform Machine Learning tasks using Quantum Computing. The high-level idea is to design an algorithm that propagates an input state to multiple quantum trajectories in superposition, in such a way that a sum of individual results from each trajectory is obtained. The algorithm is able to generate an exponentially large number of transformations of the input data in superposition, each entangled with the a basis state of a separate quantum register. By averaging those transformations, the results of the final Quantum Machine Learning (QML) model can be accessed efficiently. The proposed architecture allows to reproduce the mathematical framework in which many classical ML algorithms fall into, that is, the idea of an aggregator of different functions of the same input data. Furthermore, it allows to plug-in different components to obtain a plethora of QML models not yet present in the literature. In addition to the theoretical formulation, the quantum circuit architecture is provided. Thus, the framework is complete in its technical formulation and can be adapted for both NISQ and fault-tolerant quantum devices. Also, an in-depth discussion about time and space complexity of the algorithm, and a comparison with other quantum algorithms are provided.

Besides the general framework, two examples of its use are discussed. Firstly, in the context of fault-tolerant quantum computation, the framework is employed to implement a quantum ensemble (Macaluso et al., 2020a,d). Thanks to the generation of the several quantum trajectories in superposition, we obtain B different predictions in only $log(B)$ operations. Thus, the proposed quantum ensemble allows to increase exponentially the ensemble size with respect to classical methods, while increasing linearly the number of steps required. Furthermore, when considering the algorithm's overall time complexity, we show that the training of a single weak classifier impacts additively rather than multiplicatively, as it usually happens in classical ensemble methods.

Secondly, the proposed quantum framework is employed in the context of quantum variational algorithms, implementing a quantum Single Layer Perceptron (qSLP) (Macaluso et al., 2020c). Thanks to the universal approximation theorem, and given that the number of hidden neurons scales exponentially with the number of qubits, the qSLP algorithm opens the possibility of approximating any continuous bounded function on quantum computers. Thus, the proposed approach produces a model with substantial descriptive power and widens the horizon of potential applications already in the NISQ era, especially the ones related to Quantum Artificial Intelligence. In particular, we design a quantum circuit to perform linear combinations in superposition and discuss adaptations to classification tasks.

Finally, a further contribution of the thesis lies in the proposal of the quantum version of spline functions (Macaluso et al., 2020b), which aims to overcome the limitation to linear operations imposed by quantum mechanics that hampers the representation of complex relationships in data. In particular, we demonstrate how to adopt quantum splines for approximating several popular activation functions commonly employed in Neural Networks following two different strategies. The hybrid approach computes quantum estimates of the spline coefficients via the Harrow Hassidim Lloyd (HHL) quantum algorithm. Then a classical device is used to evaluate the activation functions. Instead, the full quantum approach takes care of the evaluation process end-to-end, with an additional circuit that reads the HHL estimates and evaluates the function. The results show that the quantum splines are able to reproduce the non-linearity of the curves, thus candidating this approach as a building block in the development of a complete model for Quantum Neural Networks.

Practical experiments using benchmark datasets support all the proposed algorithms both in the simulator and on a real device.

Part III

A Novel Quantum Framework for Machine Learning

Chapter 4

MAQA: Multiple Aggregator Quantum Algorithm

In this chapter, the main methodological contribution of the thesis is presented. First, the idea of machine learning model as functions aggregator is discussed, alongside specific algorithms that fall into this structure. A particular focus will be given on Neural Networks (NNs) and ensemble methods. Thus, the quantum algorithm that reproduces this framework is described. The proposal consists of providing a general quantum algorithm that can represent all machine learning methods that are model-based, which means that the final output is given by the additive structure of the base models. We will refer to this quantum framework as *Multiple Aggregator Quantum Algorithm* (MAQA). The proposed approach has two main advantages: from a classical machine learning perspective, it introduces an exponential scaling in the number of the aggregated functions. From a quantum computing perspective instead, the framework opens the possibility to implement a plethora of machine learning models not yet present in the literature.

4.1 Machine Learning Model as Aggregator of Multiple Functions

The goal of supervised ML is to find a useful approximation to the function $f(x;\theta)$ that underlies the predictive relationship between the input x, and output y, for a fixed set of parameters θ. Assuming for simplicity an additive error, the model of interest can be expressed as follows:

$$y = f(x;\theta) + \epsilon, \tag{4.1}$$

where ϵ is a random variable, whose conditioned probability distribution given x is centred in 0. Although Equation (4.1) provides a general mathematical formulation for supervised learning, several methods do not estimate a single function but include multiple and diverse functions which belong to the same family and differ from each other, either a set of parameters or the training data. In all these cases, the final model results from the weighted average of the estimated functions. Thus, it is possible to rewrite the function f as an aggregator of functions of the form:

$$f(x;\theta) = \sum_{h=1}^{H} \beta_h g(x;\theta_h), \tag{4.2}$$

where $f(x;\theta)$ is the final output and g describes the *function component.*

The function $g(x;\cdot)$ corresponds to a specific transformation of data based on θ_h, whose contribution to the final output is weighted by β_h. The estimation of a collection of base functions allows producing an extremely flexible model, which is able to approximate the behaviour of complex functions and solve different pattern recognition problems. Different choices for β, $g(\cdot)$ and θ_h determine different machine learning models, then Equation (4.2) incorporates many models, commonly adopted in real-world applications.

In the following sections, some of the most popular ML algorithms will be presented, describing them in terms of aggregators of functions.

4.1.1 Feedforward Neural Networks

The main idea behind feedforward Neural Networks (or Multilayer Perceptrons - MLP) is to extract linear combinations of the inputs as derived features, and model the target variable as a nonlinear function of these new features. For example, a NN for classification maps an input x to a set of categories y. In particular, a feedforward network defines a mapping $y = f(x;\theta)$ and learns the value of the parameters θ to select the best category for a given input features. These models are called feedforward because information passes through the function being evaluated from x, through the intermediate computations used to define f with no feedback connections (Figure 4.1)[1]. For an in-depth review on Neural Networks see Goodfellow et al. (2016).

Neural Networks also provide a universal approximation framework. Specifically, the universal approximation theorem (Cybenko, 1989; Hornik et al., 1990, 1989) states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units[2]. While the original theorem imposes limitations on the form of activation functions, it has also been proven that universal approximation theorem holds for a wider class of activation functions, including the commonly used rectified linear unit, also known as RELU (Leshno et al., 1993).

Notice that the universal approximation theorem says that it is possible to learn any function using a single, large MLP, but it is not guaranteed that the training algorithm can effectively learn that function. In fact, although the MLP is able to *represent* any function, the learning procedure can fail to achieve the goal for two different reasons. First, the optimisation algorithm used for training cannot find the value of the parameters that correspond to the desired function. Second, the training algorithm chooses the wrong function because of overfitting. In this regard, it is essential to consider the *no-free-lunch theorem* (Wolpert, 1996) which asserts that, given a set of data, even a powerful ML method cannot be totally superior to other algorithms. Thus, there is no universal procedure for testing a training set of specific examples and choosing a function that generalises to points out-of-sample. In other words, the universal approximation theorem says that there exists a network large enough to achieve any desired degree of accuracy, but the theorem does not say how large this network will be. Barron (1993) provides some bounds on the size of a single-layer network needed to approximate a wide class of functions. However, in the worst case, an exponentially large number of hidden units may be required (possibly with one hidden unit corresponding to each input configuration that needs to be distinguished).

In light of these considerations, even a Single Layer Neural Network (also called Single Layer Perceptron-SLP) is an extremely powerful method whose potential is

[1]When feedforward neural networks are extended to include feedback connections, they are called *recurrent neural networks*.

[2]The concept of Borel measurability is beyond the scope of this dissertation for which it suffices to say that any continuous function on a closed and bounded subset of $\mathcal{R}^n$ is Borel measurable and can be approximated by a neural network.

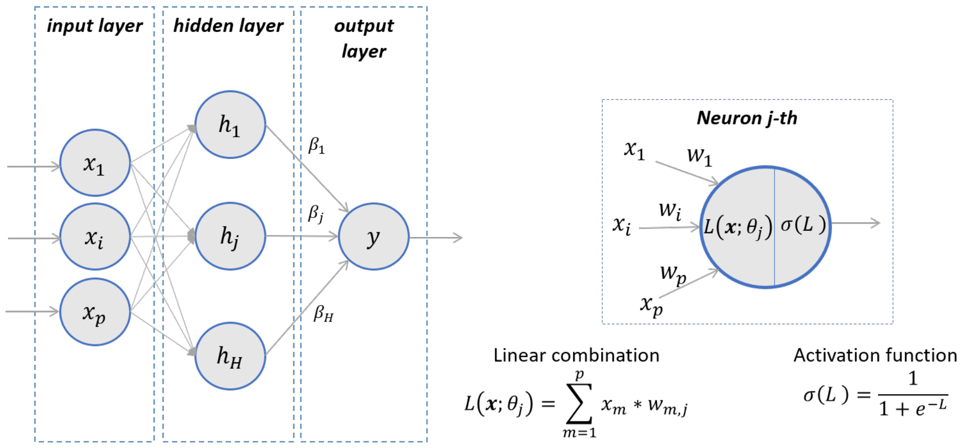


Fig. 4.1 Diagram of a Single hidden Layer Perceptron. The left-hand side represents the graph of an SLP model with p inputs, H hidden neurons and one output node suitable either for regression or binary classification. The right-hand side illustrates the functioning of each neuron after the input layer. It receives a linear combination of outputs of the previous layers and then applies an activation function that determines the actual value that is then propagated further in the network.

limited by the lack of capacity of classical optimisation algorithms to exploit its theoretical properties.

From a statistical point of view, an SLP is a non-linear statistical model similar to other methods, such as the projection pursuit regression (Friedman and Tukey, 1974). In particular, the SLP is a two-stage regression or classification model, typically represented by a network diagram as in Figure 4.1. Formally, given a training point (x_i, y_i), the output of a feedforward NN with a single hidden layer containing H neurons can be expressed as follows:

$$f(x_i) = \sigma_{\text{output}} \left(\sum_{j=1}^{H} \beta_j \sigma_{\text{hidden}} \left(L(x_i; \theta_j) \right) \right), \tag{4.3}$$

where σ_{output} is the identity when the task is the function approximation. Each hidden neuron j computes a linear combination of the input features x_i with coefficients equal to the p-dimensional vector θ_j. This operation is performed for all the neurons, and the results are then individually convoluted with the inner activation function σ_{hidden}.

Regarding the framework discusses in Section 4.1, the SLP assumes as function component $g(x; \cdot)$ the composition of the linear combinations of the input x and the activation function σ_{hidden}.

Notice that when considering a NN with multiple hidden layers, the only difference in Equation (4.3) is that the function component $g(x; \cdot)$ is, in turn, a neural network.

4.1.2 Ensemble Methods

The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models to reduce the generalisation error. A necessary and sufficient condition for an ensemble to outperform any of its members is that the single models are **accurate**, in the sense that they have an error rate better than random guessing, and **diverse**, which means that the individual models make different errors given the same data points (Hansen and Salamon, 1990).

There exist several ways to build ensemble methods, each designed to tackle a specific component of the *EPE* (Section 3.1.1). In *Boosting* (Schapire, 2003), the idea is to exploit a committee of weak learners that evolves over time. In practice, at each iteration a new weak learner is trained with respect to the error of the whole ensemble. This mechanism allows getting closer and closer to the true population values, thus reducing the bias. *Randomisation* (Kwok and Carter, 1990) methods consist in estimating the single base model with a randomly perturbed training algorithm. This alteration worsens the accuracy of the individual learners, but reduces the ensemble variance thanks to the combination of a large number of randomised models. Unlike to other methods, this approach is also applicable to stable learners, thus enlarging the plethora of methods it applies to. Another approach is *Bagging* (Breiman, 1996). In this case, the same model is fitted to different training sets, thus creating a committee of independent weak learners. The individual predictions are then averaged to obtain the ensemble prediction. This approach decreases the *EPE* by reducing the variance component so the more classifiers are included (i.e., the larger the size of the ensemble), the more significant is the reduction.

When considering ensemble methods as aggregators of functions, the component functions $g(x; \cdot)$ are (weak) classification models and the choice of the weights depends on the type of the ensemble in use.

4.1.3 Other Supervised Methods as Multiple Aggregators

In this section, other models that fall into the idea of multiple aggregations are presented. Specifically, the predicted value of these ML models is calculated according to a parametrised linear combination of a specific function $g(x; \cdot)$ that takes as input the features x and transforms them based on a set of parameters θ.

Regression based

The simplest classical supervised method that fits the idea of aggregation of multiple components is the Linear Regression. Given a vector of inputs $X^T = (X_1, X_2, \dots, X_p)$ the final output Y can be computed as:

$$Y = \beta_0 + \sum_{j=1}^{p} X_j \beta_j + \epsilon, \tag{4.4}$$

where ϵ is a not observable random variable. This methods attempts to explicitly model the relationship between inputs X_j and the outputs Y, typically in the form of linear equation in which the parameters $\{\beta_j\}_{j=1,\dots,p}$ are estimated from the data. In this case, each function component $g(x; \cdot)$ in Equation (4.2) selects a single predictor and only the β_h parameters need to be estimated. Also, the number H (number of aggregated functions) is equal to p (number of features). Linear methods can be used for both classification and regression tasks and often provide explicit estimates of measures of association between individual inputs and the outcome, adjusted for other inputs, with standard error estimates provided by the modelling paradigm in use. Usually, the least-square optimisation procedure is employed to find the best set of parameters $\{\beta_j\}_{j=1,\dots,p}$. However, under appropriate conditions[3], Maximum Likelihood Estimator and Mean squared error Estimator coincide (Freedman, 2009), this underlies the robustness of the estimated coefficients when solving a given task using linear regression.

The linear regression is one of the fundamental supervised machine learning algorithms due to its simplicity and well-known properties.

Generalised Additive Models

Although attractively simple, linear methods often fail to approximate complex functions; thus, it is necessary to transform the predictors before fitting the model, in order to incorporate non-linear forms. In this regard, a Generalised Additive Model (GAM) allows extending classical linear models to fit more different and diverse functions. In particular, a GAM has the form:

$$Y = \alpha + \sum_{j=1}^{p} \alpha_j f_j(X_j) + \epsilon, \tag{4.5}$$

where the error term ϵ is centred in zero and f_j is a smooth functions that can be either fixed apriori or estimated flexibly. In this case, the function component $g(x; \cdot)$

[3]The random variable ϵ has the following normal distribution $\epsilon \sim \mathcal{N}(0; \sigma_\epsilon)$ where σ_ϵ is known. Also, data points need to be independent and identically distributed.

selects one or more predictors and transform them according to f_j. As in standard linear regression, $\{\alpha_j\}_{j=1,\ldots,p}$ can be chosen according to least-square optimisation.

Decision Trees

A decision tree is a method for approximating discrete-valued target functions, where a set of if-then decision rules presents the learned function. These learning methods partition the feature space into a set of regions and fit a simple model (like a constant) in each one. In general, the prediction of a decision three is given by:

$$Y = \sum_{m=1}^{M} c_m \boldsymbol{I}\{(X_1, \ldots, X_p) \in R_m\}, \tag{4.6}$$

where $\{R_m\}_{m=1,\ldots,M}$ is a partition into the M regions and c_m is the estimate of the target variable in the m-th region. In this case, the set of parameters that determines the learning algorithm is characterised in terms of split variables, cut-points, terminal-node values. Regarding the framework of aggregation of functions, decision trees determine the $g(x;\cdot)$ as an indicator function and the β_i values aren't parameters to learn but constant values calculated according to the training set (more details about decision trees are provided by Hastie et al. (2001)).

Support Vector Machines

In binary classification problems, Support Vector Machines (SVM) aims to find the *optimal separating hyperplane* that separates the two classes of interest and maximises the distance to the closest from either class[4] (Cortes and Vapnik, 1995). This approach provides a unique solution to the separating hyperplane problem by maximising the margin between the two classes on the training data leads to better classification in performance on test data. In particular, given a training set of N pairs $\{x_i, y_i\}_{i=1,\ldots,N}$ with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$, we define the following hyperplane as:

$$\{x : f(x) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j = 0\}, \tag{4.7}$$

where β is a unit vector: $||\beta|| = 1$. A classification rule induced by $f(x)$ is the following:

$$G(x) = sign[x^T \beta + \beta_0], \tag{4.8}$$

[4]SVM can also be used as a regression method maintaining all the main features that characterise the algorithm

$f(x)$ gives the signed distance from a point x to the hyperplane $f(x) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j = 0$. If the classes are separable, it is possible to find a function $f(x)$ with $y_i f(x_i) > 0 \ \forall_i$. Hence it is possible to find the hyperplane that creates the biggest margin between the training points for class $+1$ and -1 (see Figure 4.2). This formulation of the SVM can

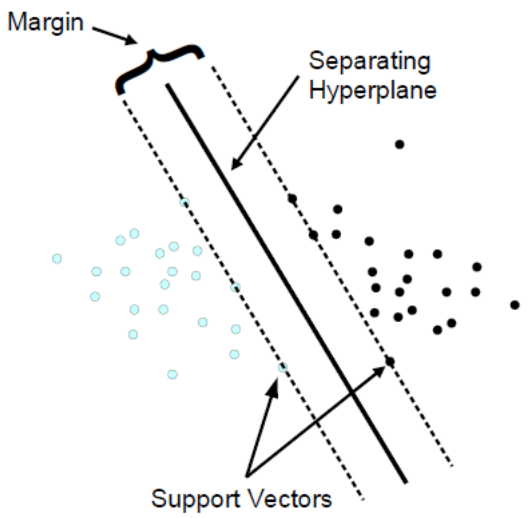


Fig. 4.2 Visual representation of SVM.

find linear boundaries in the input feature space. The extension of this approach for non-linear boundaries consist of enlarge the input space using kernel functions. In fact, generally linear boundaries in the enlarged space translate non-linear boundaries in the original space. We define $h(x)$ as the transformation of the input x into a larger feature space. Thus it is possible to show (Hastie et al., 2001) that the solution function $f(x)$ can be written as:

$$f(x) = h(x)^T + \beta_0 = \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + \beta_0, \tag{4.9}$$

where $K(x, x') = \langle h(x), h(x_i) \rangle$ computes the inner product in the transformed space. K should be a symmetric semi-definite function (Soman et al., 2009) space[5]. Although a kernel mapping is required for non-linear boundaries, what is involved in Equation 4.9 is not $h(x)$ itself but only the inner product. In particular, SVM does not explicitly map each training point into the enlarged space, but it evaluates the inner product directly.

[5]The introduction of the inner product in the objective function depends on the formulation of the optimisation problem using the Lagrangian dual objective function, that is beyond the scope of this thesis.

SVMs differ radically from similar approaches such as neural networks because they find a global minimum, and their simple geometric interpretation provides prolific ground for additional investigations. An SVM is mainly characterised by choice of its kernel, hence link the problems they are designed for with a large body of existing work on kernel-based methods.

As we can see from Equation (4.9), even the decision boundaries determined by an SVM can be considered an aggregator of functions. The difference with other approaches lies in elements of the sum that, in this case, are related to the number of points in the training set, instead of the number of features (as in the previous models).

4.2 Multiple Aggregator Quantum Algorithm (MAQA)

We propose a quantum algorithm that is able to reproduce the classical model expressed in Equation (4.2). The algorithm leverages the three main properties of quantum computing (superposition, entanglement and interference) to encode in a quantum state the sum of different transformations of the input, that is accessible by measuring a single quantum register. The proposed algorithm is potentially able to reproduce all those models that refer to the idea of functions aggregation, and provides interesting computational advantages with respect to the classical counterparts.

The quantum algorithm adopts two quantum registers: data and control. The *data* register encodes the input of the model in one of the encoding strategies detailed in Section 3.4. The *control* register is used to generate multiple trajectories in superposition, where each trajectory represents a different transformation of data.

Starting from a n-qubit *data* register and a d-qubit *control* register the *Multiple Aggregator Quantum Algorithm* (MAQA) involves four main steps: *state preparation*, *multiple trajectories in superposition*, *transformation via interference* and *measurement.*

(Step 1) State Preparation

State preparation consists in encoding the input in the *data* register and the initialisation of *control* register whose amplitudes depend on a set of parameters β:

$$|\Phi_0\rangle = (S_\beta \otimes S_x)\,|0\rangle_{\text{control}} \otimes |0\rangle_{\text{data}} = \frac{1}{\sqrt{2^d}} \sum_{i=0}^{2^d-1} \beta_i\,|i\rangle \otimes |x\rangle\,. \tag{4.10}$$

We refer to S_x as a quantum routine to encode data into a quantum state, and to S_β as a routine that transforms a d-qubit register from all-zero state to a state which depends on a set of parameters $\{\beta_i\}_{i=0,\dots,2^d-1}$. The structure of S_x and S_β will be detailed when discussing the use of this framework for specific algorithms (Ensemble and Neural Networks).

Importantly, the computational cost of this step would not be considered in classical computing. In fact, any classical algorithm assumes that the input x is given and it is directly accessible.

(Step 2) Multiple Trajectories in Superposition

The second step regards the generation of 2^d different transformations of the input data in superposition, each entangled with a possible state of the *control* register. The single quantum state of the superposition encodes a specific transformation of the data and it depends on a set of parameters Θ_k. To this end, a unitary $G(\theta_1, \dots, \theta_{2^d})$ that performs the following operation is assumed[6]:

$$|\Phi_1\rangle = G\left(\theta_1, \dots, \theta_{2^d}\right)|\Phi_0\rangle = \frac{1}{\sqrt{E}}\left(\sum_{k=0}^{2^d-1} \beta_k |k\rangle |g\left(x; \Theta_k\right)\rangle\right), \tag{4.11}$$

where E is a normalisation constant.

The implementation of $G(\theta_1, \dots, \theta_{2^d})$ can be accomplished in only d steps. Each step consists in the entanglement of the i^{th} $(i = 1, \dots d)$ control qubit with two transformations $g\left(x; \theta_{i,1}\right)$ and $g\left(x; \theta_{i,2}\right)$ of $|x\rangle$ based on two sets of parameters, $\theta_{i,1}$ and $\theta_{i,2}$, for $i = 1, \dots, d$.

Let us consider a unitary $G(\theta_{i,j})$ that implements the transformation $g\left(x; \theta_{i,j}\right)$. The most straightforward way to obtain the quantum state in Equation (4.11) is to apply $G\left(\theta_{i,j}\right)$ through controlled operations, using as control state the two basis states of the current control qubit. In particular, the generic i^{th} step involves the following two transformations:

[6]Notice that the definition of $G(\theta_1, \dots, \theta_{2^d})$ unitary in terms of quantum gates depend on the algorithm to implement and will be specified for the use of MAQA in the context of Quantum Ensemble (Chapter 5) and Quantum Neural Networks (Chapter 6).

- First, the controlled-unitary $C^{(1)}G\left(\theta_{i,1}\right)$ is executed to entangle the transformation $G\left(\theta_{i,1}\right)|x\rangle$ with the excited state of the i^{th} control qubit:

$$
\begin{aligned}
|\Phi_{i,1}\rangle &= \Big(C^{(1)}G\left(\theta_{i,1}\right)\Big)|c_i\rangle \otimes |x\rangle \\
&= \Big(C^{(1)}G\left(\theta_{i,1}\right)\Big)\Big(a_i|0\rangle + b_i|1\rangle\Big) \otimes |x\rangle \\
&= \Big(a_i|0\rangle|x\rangle + b_i|1\rangle G\left(\theta_{i,1}\right)|x\rangle\Big),
\end{aligned} \tag{4.12}
$$

  where a_i and b_i are the amplitudes of thei^{th} control qubit and $C^{(1)}G(\theta_{i,1})$ is a controlled operation that entangles the exited state of the control qubit $|c_i\rangle$ to transform the *data* register according to the unitary $G(\theta_{i,1})$.

- Then, a second controlled-unitary $C^{(0)}G\left(\theta_{i,2}\right)$ is executed. This time the control state is the $|0\rangle$ basis state:

$$
\begin{aligned}
|\Phi_{i}\rangle &= \Big(C^{(0)}G\left(\theta_{i,2}\right)\Big)|\Phi_{i,1}\rangle \\
&= \Big(C^{(0)}G\left(\theta_{i,2}\right)\Big)\Big(a_i|0\rangle|x\rangle + b_i|1\rangle G\left(\theta_{i,1}\right)|x\rangle\Big) \\
&= \Big(a_i|0\rangle G\left(\theta_{i,2}\right)|x\rangle + b_i|1\rangle G\left(\theta_{i,1}\right)|x\rangle\Big).
\end{aligned} \tag{4.13}
$$

These two transformations are repeated for each qubit in the *control* register and, at each iteration, two parametrised $G\left(\theta_{i,1}\right)$ and $G\left(\theta_{i,2}\right)$ are applied. After d steps, the *control* and *data* registers are fully entangled and 2^d different quantum trajectories in superposition are generated. The output of this procedure can be expressed as follows:

$$
|\Phi_d\rangle = \frac{1}{\sqrt{E}}\sum_{k=1}^{2^d}\beta_k|k\rangle G\left(\Theta_k\right)|x\rangle = \frac{1}{\sqrt{2^d}}\sum_{k=1}^{2^d}\beta_k|k\rangle|g(x;\Theta_k)\rangle \tag{4.14}
$$

where $G\left(\Theta_k\right)$ results from the product of d unitary matrices $G\left(\theta_{i,j}\right)$ and it represents a single quantum trajectory. Each trajectory differs from the others for, at least, one matrix $G\left(\theta_{i,j}\right)$. The composition of $G\left(\Theta_k\right)$ strictly depends on the learning algorithm and the encoding strategy chosen for data.

When discussing a specific implementation of QML algorithms, we will see that, from a computational point of view, the possibility to generate 2^d different transformations in only d steps potentially lead to an exponential speed-up with respect to classical methods, assuming an efficient implementation for the controlled-$G(\theta_{i,j})$.

(Step 3) Transformation via Interference

Once we generated multiple transformations $g(x;\Theta_k)$ of the input in superposition, the third step consists of transforming the *data* register through a generic quantum gate F that works via interference:

$$\begin{aligned}|\Phi_f\rangle &= \left(\mathbb{1}^{\otimes d} \otimes F\right)|\Phi_d\rangle \\ &= \left(\mathbb{1}^{\otimes d} \otimes F\right)\left[\frac{1}{\sqrt{E}}\sum_{k=1}^{2^d}\beta_k\,|k\rangle\,|g(x;\Theta_k)\rangle\right] \\ &= \frac{1}{\sqrt{E}}\sum_{k=1}^{2^d}\beta_k\,|k\rangle\ |f^*(x;\Theta_k)\rangle = \frac{1}{\sqrt{E}}\sum_{k=1}^{2^d}\beta_k\,|k\rangle\ |f_k^*\rangle\,, \end{aligned} \tag{4.15}$$

where $f^*(x;\Theta_k) = f_k^*$. In Equation (4.15) the assumption is that the application of $G(x;\Theta_k)$ and F on the quantum state $|x\rangle$ is equivalent to apply the target function f_k^* to an input x. At this point, different values of the function f^* are entangled with different states of the *control* register.

It is important to notice that a single execution of F allows to compute the function f^* for all the quantum trajectories in superposition. This is extremely useful when, during the computation, the same operations needs to be applied to multiple inputs (e.g., when the activation function is applied to a huge number of neurons or in case of ensemble learning, the same classifier has to be executed to different sub-samples of the training set).

(Step 4) Measurement

The last step consists of measuring the *data* register, leaving untouched the *control* register:

$$\begin{aligned}\langle M\rangle &= \left\langle \Phi_f|\mathbb{1}^{\otimes d}\otimes M\middle|\Phi_f\right\rangle \\ &= \sum_{k=1}^{2^d}\beta_k^{'}\,\langle k|k\rangle \otimes \langle f_k^*|M|f_k^*\rangle \\ &= \sum_{k=1}^{2^d}\beta_k^{'}\,\langle f_k^*|M|f_k^*\rangle = \sum_{k=1}^{2^d}\beta_k^{'}\,\langle M_k\rangle \\ &= \sum_{k=1}^{2^d}\beta_k^{'}f_k = f_{\text{agg}}, \end{aligned} \tag{4.16}$$

where $f_k = \langle f_k^*|M|f_k^*\rangle$ and $\beta_k^{'} = |\beta_k|^2$ with $\sum_k|\beta_k|^2 = 1$.

The expectation value $\langle M \rangle$ stores the weighted average of the 2^d functions f_k, that is accessible by measuring the *data* register, due to the entanglement with the *control* register. If the goal of the computation is to extract from the quantum system the single contribution f_k, this would require an exponential number of measurements, since those values are in the superposition of 2^d possible basis states. However, as usually happens in machine learning, it is not necessary to know the single contributions but rather the measure of interest is the aggregation of all the functions which provide the estimation of the target variable for a given input.

To summarise, the proposed architecture allows accessing the result of the algorithm by measuring only the *data* register. In particular, we obtain the weighted average of different values of the same function f with different parameters. The quantum state in Equation (5.10) is the quantum version of Equation (4.2) and, specifying properly S_β, S_x, $\{G(\theta_{i,1}), G(\theta_{i,2})\}_{i=1,\dots,d}$ and F allows to reproduce the quantum version of all the machine learning algorithms discussed in Section 4.1. Furthermore, the algorithm is very generic (on purpose) and can be used for NISQ and fault-tolerant computation. This also allows learning the set of parameters using a quantum variational approach in hybrid quantum-classical computation.

The algorithm expressed in pseudo-code is shown below.

Algorithm 1: Multiple Aggregator Quantum Algorithm (MAQA)

```
**Result:** Estimation of a target function as sum of $2^d$ different functions $f$

**Input:**
- $n$–qubit data register, $d$–qubit control register
- quantum gates: $S_\beta$, $S_x$, $\{G(\theta_{i,1}), G(\theta_{i,2})\}_{i=1,\dots,d}$ and $F$
- measurement operator $\langle M \rangle$

# (Step 1) State Preparation
Encode data into the n-qubits data register: $x \xrightarrow{S_x} |x\rangle$
Initialise the d-qubits control register $|0\dots0\rangle \xrightarrow{S_\beta} \sum_{k=0}^{2^d-1} \beta_k |k\rangle$

# (Step 2) Multiple Trajectories in Superposition
for each qubit in the control register $(i = 1, \dots d)$ do
    entangle state $|0\rangle$ with $G(\theta_{i,1})$
    entangle state $|1\rangle$ with $G(\theta_{i,2})$;
end

# (Step 3) Transformation via Interference
Apply the quantum gate $F$ on the data register;

# (Step 4) Measurement
Measure the data register using measurement operation $\langle M \rangle$

**Output:**
```

$$\langle M \rangle = \sum_{k=1}^{2^d} \beta_k^{'} f_k = f_{\text{agg}}$$

4.2.1 Quantum Circuit Architecture

The MAQA described in the previous section can be implemented using the quantum circuit in Figure 4.3.

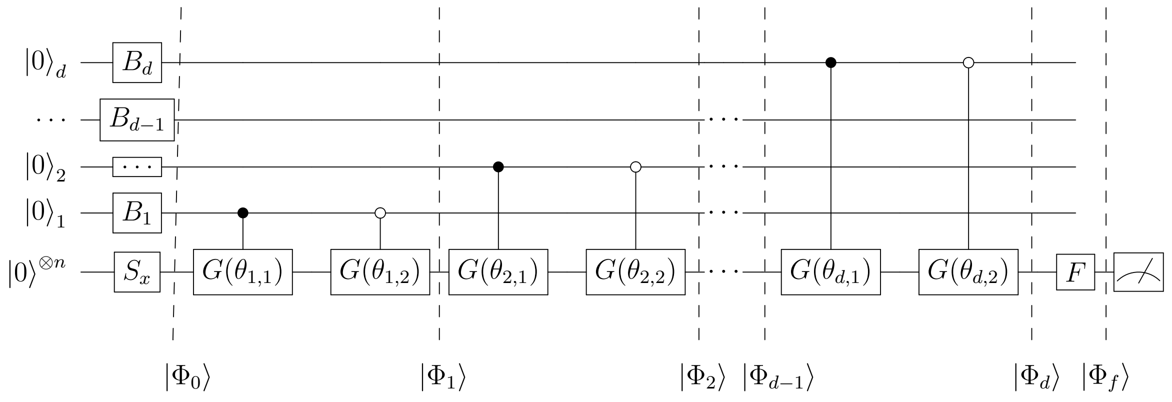


Fig. 4.3 Quantum Circuit for MAQA.

Without loss of generality, we can express the quantum gate S_β as the tensor product of d gates B_i. Then, the state preparation step can be expressed as follows:

$$\begin{aligned} |\Phi_0\rangle &= (S_\beta \otimes S_x)\,|0,\dots,0\rangle_d \otimes |0,\dots,0\rangle_n \\ &= \left(\bigotimes_{i=1}^{d} B_i \otimes S_x\right)|0\rangle^{\otimes d} \otimes |0\rangle^{\otimes n} \\ &= \bigotimes_{i=1}^{d} (a_i\,|0\rangle + b_i\,|1\rangle) \otimes |x\rangle = \bigotimes_{i=1}^{d} |c_i\rangle \otimes |x\rangle\,, \end{aligned} \tag{4.17}$$

where $|c_i\rangle$ is the i^{th} control qubit and a_i and b_i are the parameters that determine the amplitudes:

$$|c_i\rangle = a_i\,|0\rangle + b_i\,|1\rangle\,. \tag{4.18}$$

Notice that the amplitudes β_k in Equation (4.10) can be expressed as the product of d a_i and b_i parameters (Equation (2.5)).

Once the two registers are initialised, each qubit in the *control* register is entangled with two different random transformations of the *data* register. As a result, 2^d different transformations in superposition of the input are generated. Thus, the first step after state preparation is the following:

Step 2.1 $(i = 1)$

- First, the controlled-unitary $C^{(1)}G\left(\theta_{1,1}\right)$ is executed to entangle the transformation $G\left(\theta_{1,1}\right)|x\rangle$ with the excited state of $|c_1\rangle$:

$$\begin{aligned}|\Phi_{1,1}\rangle &= \Big[\mathbb{1}^{\otimes d-1} \otimes C^{(1)}G\left(\theta_{1,1}\right)\Big]|\Phi_0\rangle \\ &= \Big[\mathbb{1}^{\otimes d-1} \otimes C^{(1)}G\left(\theta_{1,1}\right)\Big]\Big(a_1|0\rangle + b_2|1\rangle\Big) \otimes |x\rangle \\ &= \bigotimes_{i=1}^{d-1}|c_i\rangle \otimes \Big(a_1|0\rangle|x\rangle + b_2|1\rangle G\Big(\theta_{1,1}\Big)|x\rangle\Big), \end{aligned} \tag{4.19}$$

- Second, $|c_1\rangle$ is transformed based on Pauli–X gate[7], so that the two basis states are exchanged:

$$\begin{aligned}|\Phi_{1,2}\rangle &= [\mathbb{1} \otimes X \otimes \mathbb{1}]|\Phi_{1,1}\rangle \\ &= \bigotimes_{i=1}^{d-1}|c_i\rangle\left(a_1|1\rangle|x\rangle + b_1|0\rangle G(\theta_{1,1})|x\rangle\right). \end{aligned} \tag{4.20}$$

- Third, a second controlled-unitary $C^{(1)}G\left(\theta_{1,2}\right)$ is executed:

$$\begin{aligned}|\Phi_1\rangle &= \Big[\mathbb{1}^{\otimes d-1} \otimes C^{(1)}G(\theta_{1,2})\Big]\Big(a_1|1\rangle|x\rangle + b_1|0\rangle G\Big(\theta_{1,1}\Big)|x\rangle\Big) \\ &= \bigotimes_{i=1}^{d-1}|c_i\rangle \otimes \Big(a_1|1\rangle G\Big(\theta_{1,2}\Big)|x\rangle + b_2|0\rangle G\Big(\theta_{1,1}\Big)|x\rangle\Big). \end{aligned} \tag{4.21}$$

At this point, two different transformations, $G\left(\theta_{1,1}\right)$ and $G\left(\theta_{1,2}\right)$ of the initial state $|x\rangle$ are generated in superposition and are entangled with the two basis states of the control qubit $|c_1\rangle$.

Step 2.2 $(i = 2)$

The same operations are applied using $|c_2\rangle$ as control qubit and different random matrices, $G\left(\theta_{2,1}\right)$ and $G\left(\theta_{2,2}\right)$.

[7] the controlled operation $C^{(0)}G(\theta_{i,j})$ where $|0\rangle$ is the controlled state to apply $G(\theta_{i,j})$ can be performed by applying the Pauli-X gate to the control qubit and then using standard controlled operation $C^{(1)}G(\theta_{i,j})$.

- First, the controlled-unitary $C^{(1)}G\left(\theta_{2,1}\right)$ is applied to entangle a transformation of $|x\rangle$ with the excited state of $|c_2\rangle$:

$$\begin{aligned}|\Phi_{2,1}\rangle &= \left(C^{(1)} \otimes \mathbb{1} \otimes G(\theta_{2,1})\right)|\Phi_1\rangle \\ &= \frac{1}{\sqrt{E}}\Big[a_2\,|0\rangle \Big(b_1\,|0\rangle\, G(\theta_{1,1})\,|x\rangle + a_1\,|1\rangle\, G(\theta_{1,2})\,|x\rangle\Big)+ \\ &\qquad + b_2\,|1\rangle \Big(b_1\,|0\rangle\, G(\theta_{2,1})G(\theta_{1,1})\,|x\rangle + a_1\,|1\rangle\, G(\theta_{2,1})G(\theta_{1,2})\,|x\rangle\Big)\Big],\end{aligned} \tag{4.22}$$

where the position of the gate $C^{(1)}$ indicates the control qubit used to apply $G(\theta_{2,1})$ and E is a normalisation constant.

- Second, $|c_2\rangle$ is transformed based on Pauli–X gate:

$$\begin{aligned}|\Phi_{2,2}\rangle &= (X \otimes \mathbb{1} \otimes \mathbb{1})\,|\Phi_{2,1}\rangle \\ &= \frac{1}{\sqrt{E}}\Big[a_2\,|1\rangle \Big(b_1\,|0\rangle\, G(\theta_{1,1})\,|x\rangle + a_1\,|1\rangle\, G(\theta_{1,2})\,|x\rangle\Big)+ \\ &\qquad + b_2\,|0\rangle \Big(b_1\,|0\rangle\, G(\theta_{2,1})G(\theta_{1,1})\,|x\rangle + a_1\,|1\rangle\, G(\theta_{2,1})G(\theta_{1,2})\,|x\rangle\Big)\Big].\end{aligned} \tag{4.23}$$

- Third, a second controlled-unitary $C^{(1)}G(\theta_{2,2})$ is executed:

$$\begin{aligned}|\Phi_2\rangle &= \left(C^{(1)} \otimes \mathbb{1} \otimes G(\theta_{2,2})\right)|\Phi_{2,2}\rangle \\ &= \frac{1}{\sqrt{E}}\Big[a_2\,|1\rangle \Big(b_1\,|1\rangle\, G(\theta_{2,2})G(\theta_{1,2})\,|x\rangle + a_1\,|0\rangle\, G(\theta_{2,2})G(\theta_{1,1})\,|x\rangle\Big)+ \\ &\qquad + b_2\,|0\rangle \Big(b_1\,|0\rangle\, G(\theta_{2,1})G(\theta_{1,1})\,|x\rangle + a_1\,|1\rangle\, G(\theta_{2,1})G(\theta_{1,2})\,|x\rangle\Big)\Big].\end{aligned} \tag{4.24}$$

Notice that the entanglement performed in **Step 2.1** influences the entanglement in **Step 2.2**, and each trajectory describes a different transformation of $|x\rangle$. Equation (4.24) can be rewritten expressing the four basis states of the control register using natural numbers:

$$\begin{aligned}
|\Phi_2\rangle = \frac{1}{\sqrt{E}}\Big[& b_2 a_1 |00\rangle\, G(\theta_{2,1})G(\theta_{1,1})\, |x\rangle \\
& + b_2 b_1 |01\rangle\, G(\theta_{2,1})G(\theta_{1,2})\, |x\rangle \\
& + a_2 a_1 |10\rangle\, G(\theta_{2,2})G(\theta_{1,1})\, |x\rangle \\
& + a_2 b_1 |11\rangle\, G(\theta_{2,2})G(\theta_{1,2})\, |x\rangle \Big] \\
= & \frac{1}{\sqrt{E}} \sum_{k=0}^{3} \beta_k |k\rangle\, , G(\Theta_k)\, |x\rangle\, , \qquad (4.25)
\end{aligned}$$

where $G(\Theta_k)$ is the product of $d = 2$ unitaries $G(\theta_{i,j})$, the coefficients β_k result from the product of two coefficients a_i and b_i and the basis states are expressed using integer representation:

$$|00\rangle = |0\rangle\, ; \quad |01\rangle = |1\rangle\, ; \quad |10\rangle = |2\rangle\, ; \quad |11\rangle = |3\rangle\, . \qquad (4.26)$$

Thus, using 2 control qubits 4 different quantum trajectories are generated that correspond to 4 different transformations of data $|x\rangle$.

Extending the same procedure when $d = 3$, the result is the following:

$$\begin{aligned}
|\Phi_3\rangle = \frac{1}{\sqrt{E}}\Big[& \beta_0 |000\rangle\, G(\theta_{3,1})G(\theta_{2,1})G(\theta_{1,1})\, |x\rangle + \beta_1 |001\rangle\, G(\theta_{3,1})G(\theta_{2,1})G(\theta_{1,2})\, |x\rangle \\
& + \beta_2 |010\rangle\, G(\theta_{3,1})G(\theta_{2,2})G(\theta_{1,1})\, |x\rangle + \beta_3 |011\rangle\, G(\theta_{3,1})G(\theta_{2,2})G(\theta_{1,2})\, |x\rangle \\
& + \beta_4 |100\rangle\, G(\theta_{3,2})G(\theta_{2,1})G(\theta_{1,1})\, |x\rangle + \beta_5 |101\rangle\, G(\theta_{3,2})G(\theta_{2,1})G(\theta_{1,2})\, |x\rangle \\
& + \beta_6 |110\rangle\, G(\theta_{3,2})G(\theta_{2,2})G(\theta_{1,1})\, |x\rangle + \beta_7 |111\rangle\, G(\theta_{3,2})G(\theta_{2,2})G(\theta_{1,2})\, |x\rangle \Big] \\
= & \frac{1}{\sqrt{E}} \sum_{k=0}^{7} \beta_k |k\rangle\, G(\Theta_k)\, |x\rangle\, , \qquad (4.27)
\end{aligned}$$

where each $G(\Theta_k)$ is the product of 3 unitaries $G(\theta_{i,j})$ for $i = 1, 2, 3$ and $j = 1, 2$.

Repeating this procedure d times with different control qubits the result is the following quantum state:

$$|\Phi_d\rangle = \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_k |k\rangle\, G(\Theta_k)\, |x\rangle = \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_k |k\rangle\, |g(x; \Theta_k)\rangle\, , \qquad (4.28)$$

where each $G(\Theta_k)$ is the product of d unitaries $G(\theta_{i,j})$ for $i = 1, \cdots, d$ and $j = 1, 2$.

Finally, gate F is applied, as shown in Equation (4.15) and the measurement of the data register is performed.

The underlying idea of this procedure is to initialise the control register according to a set of weights and assign the weight β_k to a component function $g(x; \Theta_k)$. This approach is extremely flexible and allows training all the parameters β_k and Θ_k, that are not directly determined but depend on a_i, b_i and $\theta_{i,j}$.

Therefore, the MAQA method allows to reproduce a weighted average of different transformations of the input; this computation fits the idea function aggregation described in Section 4.2. In particular, by leveraging the entanglement between the *data* and the *control* registers, the number of different transformations increase exponentially at each iteration. Furthermore, the proposed architecture allows propagating the application of the quantum gate F to all the transformations. In the next chapters, it will be shown how to employ the MAQA in the context of the fault-tolerant computation through quantum ensemble (Chapter 5) and in the quantum-classical hybrid computation for Quantum Neural Networks (Chapter 6).

4.2.2 Computational Considerations

As shown in the previous section, the MAQA reproduces the idea of ML model as aggregator of functions using the properties of quantum computing. The advantage of using MAQA in Quantum Machine Learning is the possibility of implementing the quantum counterpart of many Machine Learning algorithms by designing properly just two sets of quantum gates ($G(\cdot)$ and F) based on the task of interest. The architecture is extremely flexible and can be leveraged in both the contexts of fault-tolerant and NISQ computation.

From a classical ML perspective, relevant computational advantage are introduced. Given 2^d base functions, any method that leverages the idea of function aggregation scales linearly in 2^d, because it is necessary to compute explicitly the base functions in order to obtain the overall average (Equation 4.2). Furthermore, in the worst-case scenario, each base function has to process all available data to obtain the final model, this implies a linear cost in the size of the training set multiplied by 2^d. Using big-$\mathcal{O}$ notation, given a dataset (x_i, y_i) for $i = 1, \dots N$, where x_i is a p-dimensional vector and y_i is the target variable of interest, the overall time complexity of any ML model based on the aggregation of 2^d functions is:

$$\mathcal{O}(2^d N^\alpha p^\beta) \qquad \alpha, \beta \geq 1.$$

In contrast, with MAQA it is possible to generate a superposition of 2^d different transformations of the input in only d steps because the single transformations are not computed directly, but they result by the combination of different unitaries $G(\theta_{i,j})$. In particular, each step consists in implementing 2 controlled-$G(\theta_{i,j})$ operations. Then, once the quantum state in Equation (4.28) is generated, any further operation is propagated to all the quantum trajectories with a single execution (unitary F). Using big-$\mathcal{O}$ notation, the cost of implementing the MAQA is the following:

$$\mathcal{O}\left(d \times 2C_G + C_F\right),$$

where C_G is the cost of implementing the controlled operation $C^{(j)}G(\theta)$ $(j = 0, 1)$ and C_F is the cost of implementing F. Note that, assuming a unitary cost for each step, with respect to the parameter d, the number of different functions grows exponentially. This means that it is possible to generate an exponentially large number of different functions of the input while obtaining their average efficiently, simply by measuring the *data* register. Also, the computation allows optimising the application of any function that can be defined as the F gate that works via interference. In the next chapters, it will be shown how to exploit these computational advantaged to train a QML model.

However, all these advantages come with some compromises. First, the assumption about the nature of the operator $G(\theta_{i,j})$; indeed, the algorithm assumes that the product of $G(\theta_{i,j})$, applied to $|x\rangle$ implements the function $g(x; \Theta_k)$, whose parameters Θ_k can be expressed as a function of the $\theta_{i,j}$ explicitly defined:

$$G(\Theta_k) = \prod_{\substack{i=1,\dots,d \\ j=1,2}} G(\theta_{i,j}). \tag{4.29}$$

In practice, this means that multiple applications of the unitary that depends on some set of parameters $\theta_{i,j}$, result in a single transformation of the same nature that depends on a derived set of parameters Θ_k.

Also, we assumed that the composition of the two sets of gates $G(\Theta_k)$ and F is the quantum counterpart of the base function f^* in Equation (4.2).

Finally, as described in Section 2.6, when looking at the complexity of a quantum algorithm, it is necessary to consider its cost in terms of gate complexity. Thus, it is necessary that the exponential scaling introduced with respect to d is maintained when considering a specific implementation of the QML model.

Chapter 5

Quantum Algorithm for Ensemble Learning

A powerful way to improve performance in machine learning is to construct an ensemble that combines the predictions of multiple models. Ensemble methods are often much more accurate and have lower variance than the individual classifiers that make them up, but have high requirements in terms of memory and computational time. In fact, a large number of alternative algorithms is usually adopted, each requiring to query all available data.

In this chapter, we adopt the MAQA framework to implement a new quantum algorithm that exploits quantum superposition, entanglement and interference to build an ensemble of classification models. Thanks to the generation of several quantum trajectories in superposition, we obtain B transformations of the quantum state which encodes the training set in only $log\left(B\right)$ operations. This implies an exponential growth in the size of the ensemble with respect to classical methods. Furthermore, when considering the overall cost of the algorithm, we show that the training of a single weak classifier impacts additively rather than multiplicatively, as it usually happens classically.

We also introduce a new routine for classification, named quantum cosine classifier, that works via interference and allows to implement the quantum ensemble using it as base model. Finally, we present experiments on several real-world datasets to discuss advantages and limitations of the proposed approach, defining a quantum version of the cosine classifier and using the IBM qiskit environment to show how the algorithm works.

5.1 Bagging Strategy for Ensemble Methods

The idea behind ensemble methods based on the bagging strategy is to build a prediction model by combining the strengths of a collection of simpler base models. Ensemble learning can be broken down into two tasks: developing a population of base learners from the training data, and then combining them to form the composite predictor. The crucial element is the instability of the prediction method. In particular, if perturbing the training set can cause significant changes in the predictor constructed, then bagging can improve significantly the accuracy.

In practice, bagging reduces to computing several predictions $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$ using B different training sets, which are then averaged to obtain a single model with lower variance:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x). \tag{5.1}$$

Although this approach guarantees a lower uncertainty in prediction, it is not practical in its theoretical formulation, due to the lack of multiple training sets. To overcome this issue, the bootstrap procedure (Efron and Tibshirani, 1994) can be employed, that takes repeated samples from the available data and generates B different bootstrapped training sets. The learning algorithm is then trained on the b^{th} bootstrapped observations to get B different predictions $\hat{f}_b(x)$. The difference between the bootstrap and the idealised procedure is the way the training sets are derived. Instead of obtaining independent datasets from the domain, the initial training set is perturbed as many times as the number of weak classifiers to aggregate. The generated datasets are certainly not independent because they are all based on the same training set. Nonetheless, empirical findings suggest that bagging is still able to produce combined models that often significantly outperform individual learners, and that are never substantially worse (Tumer and Ghosh, 1996b).

5.2 Quantum Algorithm for Ensemble Learning

In this section we introduce the basic idea of our quantum algorithm for ensemble classification using bagging in the context of binary classification.

The algorithm adopts three quantum registers, the ones required by the MAQA framework described in the Chapter 4 (*data*, *control*) plus an additional register to encode the test set. In particular, the *data* register encodes the whole training set and it is employed together with the d-qubits *control* register to generate 2^d altered copies of the training set in superposition. This step allows to simulate the bootstrap procedure in the context of bagging by executing only once the state preparation routine. The *test* register, instead, encodes unseen observations from the test set. Starting from these three registers, the algorithm involves four main steps: *state preparation*, *sampling in superposition*, *learning via interference* and *measurement*.

(Step 1) State Preparation

State preparation consists in the initialisation of the *control* register into a uniform superposition through a Walsh-Hadamard gate and the encoding of the training set (x, y) in the *data* register:

$$|\Phi_0\rangle = \left(W \otimes S_{(x,y)}\right) \bigotimes_{j=1}^{d} |0\rangle \otimes |0\rangle = \left(H^{\otimes d} \otimes S_{(x,y)}\right) \bigotimes_{j=1}^{d} |0\rangle \otimes |0\rangle = \bigotimes_{j=1}^{d} |c_j\rangle \otimes |x, y\rangle \,, \tag{5.2}$$

where $S_{(x,y)}$ is the state preparation routine for the training set and it strictly depends on the encoding strategy, W is the Walsh-Hadamard gate and $|c_j\rangle$ is the j-th qubit of the *control* register.

(Step 2) Sampling in Superposition

The second step regards the generation of 2^d different transformations of the training set in superposition, each entangled with a state of the *control* register. To this end, d steps are necessary, where each step consists in the entanglement of the i^{th} *control* qubit with two transformations of $|x, y\rangle$ based on two random unitaries, $U_{(i,1)}$ and $U_{(i,2)}$, for $i = 1, \ldots, d$. The most straightforward way to accomplish this is to apply the unitary $U_{(i,j)}$ through controlled operations, using as control state the two basis states of the current *control* qubit. In particular, the generic i^{th} step involves the following three transformations:

- First, the controlled-unitary $CU_{(i,1)}$[1] is executed to entangle the transformation $U_{(i,1)}\left|x,y\right\rangle$ with the excited state of the i^{th} *control* qubit:

$$\begin{aligned}\left|\Phi_{i,1}\right\rangle =&\Big(CU_{(i,1)}\Big)\left|c_i\right\rangle \otimes \left|x,y\right\rangle\\ =&\Big(CU_{(i,1)}\Big)\frac{1}{\sqrt{2}}\Big(\left|0\right\rangle + \left|1\right\rangle\Big)\otimes\left|x,y\right\rangle\\ =&\frac{1}{\sqrt{2}}\Big(\left|0\right\rangle\left|x,y\right\rangle + \left|1\right\rangle U_{(i,1)}\left|x,y\right\rangle\Big).\end{aligned} \tag{5.3}$$

- Second, the i–th *control* qubit is transformed based on Pauli–X gate:

$$\begin{aligned}\left|\Phi_{i,2}\right\rangle =&(X\otimes \mathbb{1})\left|\Phi_{i,1}\right\rangle\\ =&\frac{1}{\sqrt{2}}\Big(\left|1\right\rangle\left|x,y\right\rangle + \left|0\right\rangle U_{(i,1)}\left|x,y\right\rangle\Big).\end{aligned} \tag{5.4}$$

- Third, a second controlled-unitary $CU_{(i,2)}$ is executed:

$$\begin{aligned}\left|\Phi_{i}\right\rangle =&\Big(CU_{(1,2)}\Big)\left|\Phi_{i,2}\right\rangle\\ =&\Big(CU_{(1,2)}\Big)\frac{1}{\sqrt{2}}\Big(\left|1\right\rangle\left|x,y\right\rangle + \left|0\right\rangle U_{(i,1)}\left|x,y\right\rangle\Big)\\ =&\frac{1}{\sqrt{2}}\Big(\left|1\right\rangle U_{(i,2)}\left|x,y\right\rangle + \left|0\right\rangle U_{(i,1)}\left|x,y\right\rangle\Big).\end{aligned} \tag{5.5}$$

These three transformations are repeated for each qubit in the *control* register and, at each iteration, two random unitaries $U_{(i,1)}$ and $U_{(i,2)}$ are applied. After d steps, the *control* and *data* registers are fully entangled and 2^d different quantum trajectories in superposition are generated (as shown in Section 4.2.1). The output of this procedure can be expressed as follows:

$$\left|\Phi_d\right\rangle =\frac{1}{\sqrt{2^d}}\sum_{b=1}^{2^d}\left|b\right\rangle V_b\left|x,y\right\rangle = \frac{1}{\sqrt{2^d}}\sum_{b=1}^{2^d}\left|b\right\rangle\left|x_b,y_b\right\rangle, \tag{5.6}$$

where V_b results from the product of d matrices $U_{(i,j)}$ and represents a single quantum trajectory which differs from the others for at least one matrix $U_{(i,j)}$. In general, it is possible to refer to the unitary V_b as a unitary transformation that allows obtaining a

[1]by default the control state is the excited state $\left|1\right\rangle$.

random sub-sample of the training set:

$$|x, y\rangle \xrightarrow{V_b} |x_b, y_b\rangle\,. \tag{5.7}$$

The composition of V_b strictly depends on the encoding strategy chosen for data. In Section 5.3.1 we introduce a classification routine (*quantum cosine classifier*) based on the qubit encoding strategy, where a single observation is encoded into a qubit. In that case, the implementation of unitaries $U_{(i,j)}$ corresponds to the swap gate applied randomly to the qubits of the *data* register. Notice that the only requirement to perform ensemble learning using bagging effectively is that small changes in the product of the unitaries $U_{(i,j)}$ imply significant differences in $|x_b, y_b\rangle$, since the more independent sub-samples are, the better the ensemble works.

(Step 3) Learning via Interference

The third step of the algorithm is *Learning via Interference*. First, the *test* register is initialised to encode the test set, $x^{(\text{test})}$, considering also an additional register to store the final predictions:

$$(S_{x^{(\text{test})}} \otimes \mathbb{1})\,|0\rangle\,|0\rangle = |x^{(\text{test})}\rangle\,|0\rangle\,. \tag{5.8}$$

Then, the *data* and *test* registers interact via interference to compute the estimates of the target variable. To this end, we define a quantum classifier F that satisfies the necessary conditions described in Section 4.1.2. In particular, F acts on three registers to predict $y^{(\text{test})}$ starting from the training set (x_b, y_b):

$$|x_b, y_b\rangle\,|x^{(\text{test})}\rangle\,|0\rangle \xrightarrow{F} |x_b, y_b\rangle\,|x^{(\text{test})}\rangle\,|\hat{f}_b\rangle\,. \tag{5.9}$$

Thus, F represents the classification function $\hat{f}$ that estimates the value of the target variable of interest. For example, in binary classification problems, the prediction can be encoded into the probability amplitudes of a qubit, where the state $|0\rangle$ encodes one class, and the state $|1\rangle$ the other.

The *Learning via Interference* step leads to:

$$
\begin{aligned}
|\Phi_f\rangle &= \left(\mathbb{1}^{\otimes d} \otimes F\right) |\Phi_d\rangle \\
&= (\mathbb{1}^{\otimes d} \otimes F) \left[\frac{1}{\sqrt{2^d}} \sum_{b=1}^{2^d} |b\rangle \, |x_b, y_b\rangle \right] \otimes |x^{(\text{test})}\rangle \, |0\rangle \\
&= \frac{1}{\sqrt{2^d}} \sum_{b=1}^{2^d} |b\rangle \, |x_b, y_b\rangle \, |x^{(\text{test})}\rangle \, |\hat{f}_b\rangle \,, \qquad (5.10)
\end{aligned}
$$

where $\hat{f}_b$ represents the prediction for $x^{(\text{test})}$ given the b^{th} training set, and it is implemented via F. Notice that expressing the prediction according to Equation 5.10 implies that it is necessary to execute F only once in order to propagate its use to all the quantum trajectories. Furthermore, as consequence of Steps 2 and 3, the b^{th} state of the *control* register is entangled with the b^{th} value of $\hat{f}$.

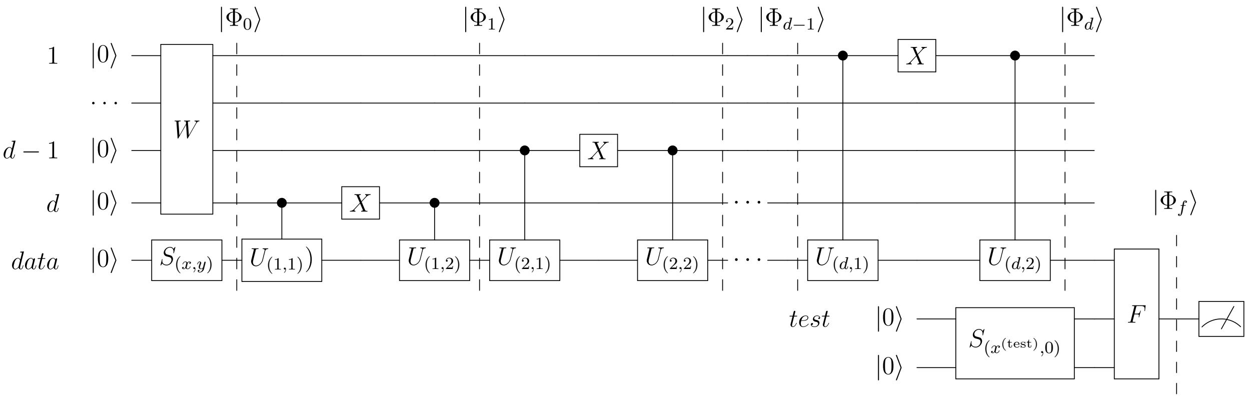


Fig. 5.1 Quantum algorithm for ensemble classification. The circuit contains d pairs of unitaries $U_{(i,1)}$, $U_{(i,2)}$ and d *control* qubits. It produces an ensemble of B classifiers, where $B = 2^d$. The single evaluation of F allows propagating the classification function $\hat{f}$ in all trajectories in superposition. The first d steps allows generating B transformations of the training set (x, y) in superposition, and each transformation is entangled with a quantum state of the *control* register (firsts d qubits). Thus, the test set $x^{(\text{test})}$ is encoded in the *test* register that interferes with all samples in superposition. Finally, the ensemble prediction is obtained as the average of individual results from each trajectory.

(Step 4) Measurement

Measuring the last register allows retrieving the average of the predictions provided by all the classifiers:

$$\begin{aligned}\langle M\rangle &= \left\langle \Phi_f | \mathbb{1}^{\otimes d} \otimes \mathbb{1} \otimes \mathbb{1} \otimes M \middle| \Phi_f \right\rangle \\ &= \frac{1}{2^d}\sum_{b=1}^{2^d} \langle b|b\rangle \otimes \langle (x_b, y_b)|(x_b, y_b)\rangle \otimes \langle x^{(\text{test})}|x^{(\text{test})}\rangle \otimes \langle \hat{f}_b|M|\hat{f}_b\rangle \\ &= \frac{1}{2^d}\sum_{b=1}^{2^d} \langle \hat{f}_b|M|\hat{f}_b\rangle = \frac{1}{2^d}\sum_{b=1}^{2^d}\langle M_b\rangle \\ &= \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b = \hat{f}_{bag}(x^{(\text{test})}|x, y),\end{aligned} \tag{5.11}$$

where $B = 2^d$ and M is a measurement operator (e.g. Pauli-Z gate). The expectation value $\langle M\rangle$ computes the ensemble prediction since it results from the average of the predictions of all the weak learners. Thus, if the two classes of the target variable are encoded in the two basis states of a qubit, it is possible to access to the ensemble prediction by single-qubit measurement:

$$\hat{f}_{bag} = \sqrt{a_0}\,|0\rangle + \sqrt{a_1}\,|1\rangle\,, \tag{5.12}$$

where a_0 and a_1 are the average of the probabilities for $x^{(\text{test})}$ to be classified in class 0 and 1, respectively. The quantum circuit of the quantum ensemble is illustrated in Figure 5.1.

5.2.1 Quantum Algorithm for Boosting and Randomisation

The same framework presented above can be adapted with slight variations to allow also randomisation and boosting.

The main principle of the ensemble based on randomisation consists in the introduction of casual perturbations that decorrelate the predictions of individual classifiers as much as possible. In this case, it is possible to loosen the constraints imposed on the classifier F, which can be generalised beyond weak learners. The procedure described in Step 2 (*Sampling in Superposition*), in fact, can be employed to introduce a random component in the single learner, so to decrease the accuracy of each individual model. As a consequence, the predictions are less correlated and the variance of the final prediction is reduced.

Technically, it is necessary to define a classification routine which can be decomposed in the product of V_b and F. Here, the different trajectories do not simulate the bootstrap procedure as for bagging, but they are part of the classification routine and introduce randomisation in the computation of $\hat{f}$. In practice, we define a unitary G_b that performs the following transformation:

$$|x,y\rangle \, |x^{(\text{test})}\rangle \, |0\rangle \xrightarrow{G_b} |x,y\rangle \, |x^{(\text{test})}\rangle \, |\hat{f}_b\rangle \, , \tag{5.13}$$

where $G_b = V_b F$ is the quantum classifier composed by F – common to all the classifiers – and V_b which is its random component – different for each quantum trajectory. This formulation allows rewriting the quantum state in Equation 5.10 as:

$$|\Phi_f\rangle = \frac{1}{\sqrt{2^d}} \sum_{b=1}^{2^d} |b\rangle \, G_b \, |x,y\rangle \, |x^{(\text{test})}\rangle \, |0\rangle = \frac{1}{\sqrt{2^d}} \sum_{b=1}^{2^d} |b\rangle \, G_b \, |x,y\rangle \, |x^{(\text{test})}\rangle \, |\hat{f}_b\rangle \, . \tag{5.14}$$

Likewise, the proposed framework can also be adapted for boosting, where the estimates provided by the single classifiers are weighted so that individual models do not contribute equally to the final prediction. In practice, the only difference is that the amplitudes of the *control* register need to be updated as the computation evolves. As a result, the output of a quantum ensemble based on boosting can be described as:

$$|\Phi_f\rangle = \frac{1}{\sqrt{2^d}} \sum_{b=1}^{2^d} \alpha_b \, |b\rangle \, |\hat{f}_b\rangle \, , \tag{5.15}$$

where the contribution of $\hat{f}_b$ to the ensemble depends on α_b. However, although in principle this approach fits in the scheme of a boosting ensemble, the difficulty in updating the *control* register is non-trivial.

To summarise, the main difference between quantum bagging and the other approaches is the way we define the unitaries $U_{(i,j)}$ and F. However, the exponential growth in the ensemble size that comes from the advantage of generating an ensemble of $B = 2^d$ classifiers in only d steps still holds.

5.2.2 Aggregation Strategy and Theoretical Performance

When considering classical implementations of ensemble algorithms, it is possible to distinguish two broad families of methods based on the strategy adopted to aggregate the predictions of the individual models. On one hand, the most popular technique used in ensemble classification is *majority voting*, where each classifier votes for a target class and the most frequent is then selected. On the other hand, an alternative strategy is given by *simple averaging*. In this case, the target probability distribution provided by individual models is considered, and the final prediction is computed as follows:

$$f_{\text{avg}}^{(i)}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b^{(i)}(x), \tag{5.16}$$

where B is the ensemble size and $f_b^{(i)}(x)$ is the probability for x to be classified in the i^{th} class provided by the b^{th} classifier. This approach allows a reduction of the estimates variance (Tumer and Ghosh, 1996a) and has shown good performance even for large and complex datasets (Xu et al., 1992). In particular, the error E_{ens} of an ensemble obtained averaging B individual learners can be expressed as (Jacobs, 1995; Oza and Tumer, 2008):

$$E_{\text{ens}} = \frac{1+\rho(B-1)}{B} E_{\text{model}}, \tag{5.17}$$

where E_{model} is the expected error of the single models and ρ is the average correlation among their errors. Hence, the more independent the single classifiers are, the greater the error reduction due to averaging. A graphical illustration of the theoretical performance of an ensemble as a function B, ρ and E_{model} is reported in Figure 5.2.

Coming to our implementation of the quantum ensemble, the prediction of the single classifier is encoded into the probability amplitudes of a quantum state and the final prediction is computed by averaging the results of all quantum trajectories in superposition. Implicitly, this means that the quantum ensemble fits in the simple averaging strategy. Thus, the possibility to generate exponentially large ensembles at the cost of increasing linearly the number of *control* qubits d allows quantum ensemble to improve significantly the performance of the single classifier (Figure 5.2) using relatively small circuits ($d \sim 10$).

5.2.3 Computational Considerations

Classically, given a number B of base learners and a dataset (x_i, y_i) for $i = 1, \dots N$, where x_i is a p-dimensional vector and y_i is the target variable of interest, the overall time complexity for training an ensemble based on randomisation or bagging scales at least linearly with respect to B and polynomially in p and N:

$$\underbrace{\mathcal{O}(BN^{\alpha}p^{\beta})}_{\text{Training}} + \underbrace{\mathcal{O}(Bp)}_{\text{Testing}} \qquad \alpha, \beta \geq 1,$$

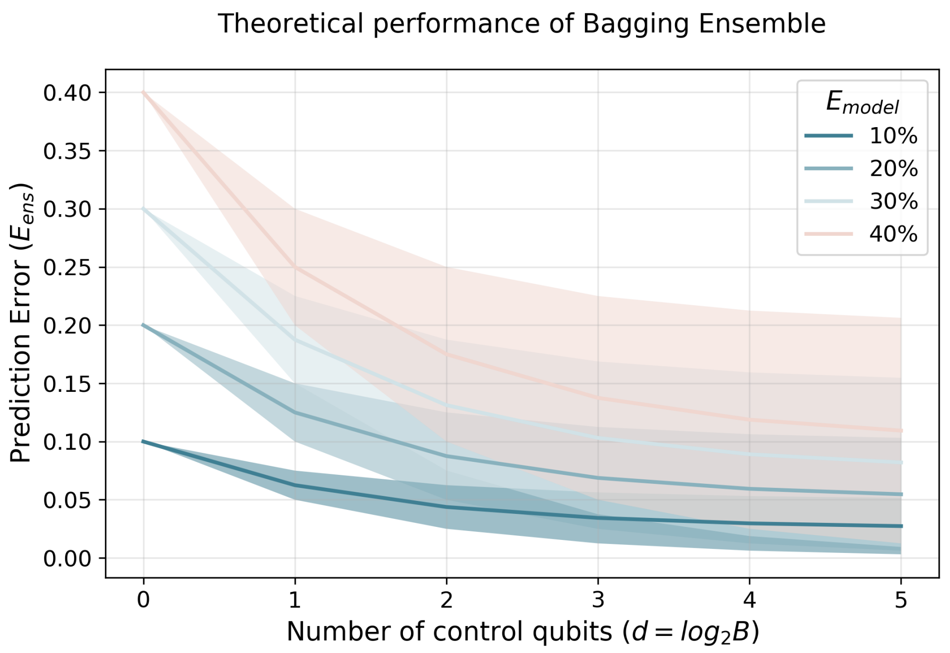


Fig. 5.2 Theoretical performance of the quantum ensemble based on the expected prediction error of the base classifiers (E_{model}) and their average correlation (ρ). The ensemble size depends on the number of qubits d in the *control* register. Each solid line corresponds to an error level, with coloured bands obtained by varying ρ between 0 (lower edge) and 0.5 (upper edge).

where α and β depend on the single base model and $N^{\alpha}p^{\beta}$ is its training cost. In boosting, instead, the model evolves over time and the individual classifiers are not independent. This usually implies higher time complexity and less parallelism.

Despite this clear definition of the computational cost, comparing the classical algorithm to its quantum counterpart is not straightforward since they belong to different classes of complexity. For this reason, we benchmarked the two approaches by looking at how they scale in terms of the parameters of the ensemble, i.e, the ensemble size B and the cost of each base model. In particular, this resolves in considering the Boolean circuit model (Arora and Barak, 2009) for the classical ensemble, and the depth of the corresponding quantum circuit for the quantum algorithm. In light of this definition, the quantum algorithm described in Section 5.2 is able to generate an ensemble of size $B = 2^d$ in only d steps. This means that, assuming a unitary cost for each step, we are able to introduce (potentially) an exponential speed-up with respect to classical ensemble methods in terms of the ensemble size.Furthermore, the cost of the single classifier is additive – instead of multiplicative as in classical ensembles – since it is necessary to execute the quantum classifier F only once to propagate its application to all quantum trajectories in superposition, as shown in Equation (5.10). In addition, the cost of the state preparation routine is equivalent to any other quantum algorithm for processing the same training and test sets. However, this comparison does not take into account the additional cost due to state preparation which is not present in classical ensembles. Also, the quantum ensemble comes with an extra cost related to the implementation of the gates $U_{(i,j)}$, that strictly depends on the encoding strategy chosen for the data and needs to be evaluated for a any specific implementation.

5.3 Experiments

To test how our framework for quantum ensemble works in practice, we implemented the circuit illustrated in Figure 5.1 using IBM Qiskit[2]. Then, we conducted experiments on simulated data to show that (i) one execution of a quantum classifier allows retrieving the ensemble prediction, and that (ii) the ensemble outperforms the single model.

5.3.1 Quantum Cosine Classifier

In order to implement the quantum ensemble, a classifier that fulfils the conditions in Equation (5.9) is necessary. For this purpose, we define a simple routine for classification based on the swap-test (Buhrman et al., 2001) that stores the cosine distance into the amplitudes of a quantum state. This metric describes how similar two vectors are depending on the angle that separates them, irrespectively of their magnitude. The smaller the angle between two objects, the higher the similarity. Starting from this, the high-level idea is predicting a similar target class for similar input features. In particular, for any test observation $(x^{(\text{test})}, y^{(\text{test})})$ we take one training point (x_b, y_b) at random and express the probability of $y^{(\text{test})}$ and y_b being equal as a function of the similarity between $x^{(\text{test})}$ and x_b:

$$Pr\left(y^{(\text{test})} = y_b\right) = \frac{1}{2} + \frac{\left[d\left(x_b, x^{(\text{test})}\right)\right]^2}{2}, \tag{5.18}$$

where $d(\cdot,\cdot)$ is the cosine distance between x_b and $x^{(\text{test})}$. Thus, the final classification rule becomes:

$$y^{(\text{test})} = \begin{cases} y_b, & \text{if } Pr\left(y^{(\text{test})} = y_b\right) > \frac{1}{2} \\ 1 - y_b & \text{otherwise }, \end{cases} \tag{5.19}$$

[2]https://qiskit.org/

Notice that, by definition, $Pr\left(y^{(\text{test})} = y_b\right)$ is bounded in $[\frac{1}{2}, 1]$, which means that Equation (5.19) always estimates the same class as the training point, unless x_b and $x^{(\text{test})}$ are orthogonal. As a consequence, the cosine classifier performs well only if the test and training observations happen to belong to the same target class.

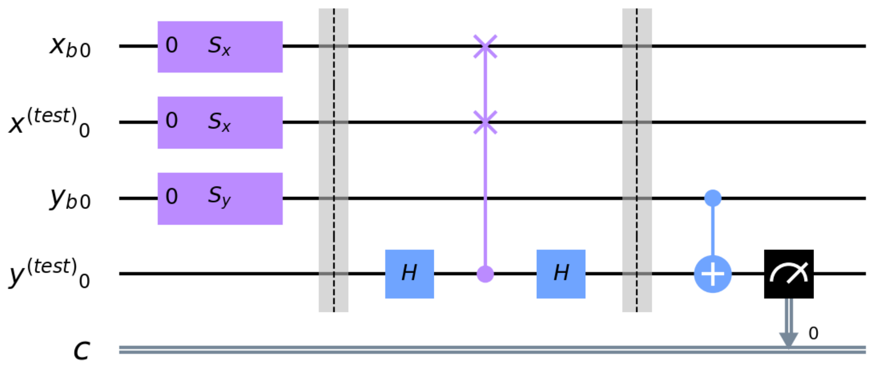


Fig. 5.3 Quantum circuit of the cosine classifier using x_b as training vector and $x^{(\text{test})}$ as test vector. The training label y_b is either $|0\rangle$ or $|1\rangle$ based on the binary target value. The results of the classification based on random generated data points are shown in Figure 5.4.

The quantum circuit to reproduce the cosine classifier (Figure 5.3) encodes data into three different registers: the training vector x_b, the training label y_b and the test point $x^{(\text{test})}$. An additional qubit is then used to store the prediction. The algorithm is made of three steps. First, data are encoded into three different quantum registers through a routine S. Second, the swap-test transforms the amplitudes of the qubit $y^{(\text{test})}$ as a function of the squared cosine distance. In particular, after the execution of the swap-test the probability of getting the basis state $|0\rangle$ is between 1/2 and 1, hence the probability of class 0 is never lower than the probability of class 1. Third, a controlled Pauli-X rotation is applied using as *control* qubit the label of the training vector. This implies that $y^{(\text{test})}$ is left untouched if x_b belongs to the class 0. Otherwise, the amplitudes of the $y^{(\text{test})}$ qubit are inverted, and $Pr(y^{(\text{test})} = 1)$ becomes higher as the similarity between the two vectors increases.

To summarise, the quantum cosine classifier performs classification via interference and allows calculating the probability of belonging to one of the two classes by single-qubit measurement. Furthermore, it is a weak method with high-variance, since it is sensitive to the random choice of the training observation. In addition, it requires data to be encoded using qubit encoding, where a dataset with N 2-dimensional observations x_b is stored into N different qubits. This allows the definition of $U_{(i,j)}$ in terms of random swap gates that move observations from one register to another. All these features make this classifier a good candidate for ensemble methods (more details about the quantum classifier are provided in Appendix B.3).

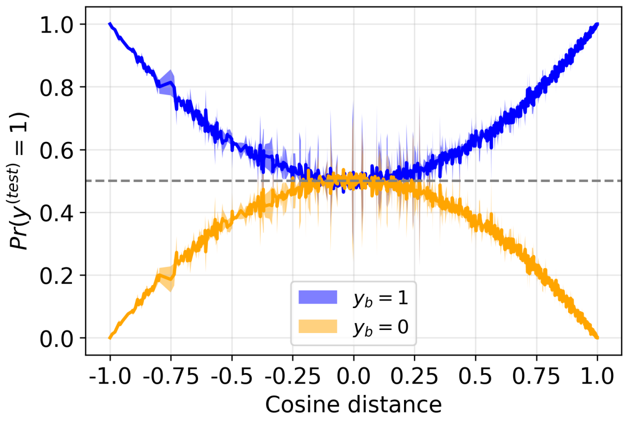


Fig. 5.4 Predictions of the cosine distance classifier based on 10^3 randomly generated datasets per class. The classifier is implemented using the circuit in Figure 5.3.

5.3.2 Quantum Ensemble as Simple Averaging

As a proof-of-concept for the quantum ensemble based on bagging, we consider a dataset with four training points and one test example. We show experimentally that the quantum ensemble prediction is exactly the average of the values of all trajectories in superposition and it can be obtained with just one execution of the classification routine.

The toy dataset used here is reported in Table 5.1. Each training point is fed into a quantum cosine classifier as input so to provide an estimate for a test observation $x^{(\text{test})}$. In practice, the quantum circuit of the ensemble uses two qubits in the *control* register ($d = 2$) and eight in the *data* register, four for the training vectors x_b and four for training labels y_b. Two additional qubits are then used for the test observation, $x^{(\text{test})}$, and the final prediction. Notice that the four matrices $U_{(i,j)}$ need to be fixed to guarantee that each quantum trajectory V_b described in Section 5.2 provides the prediction of different and independent training points (the details about the implementation in terms of quantum gates is reported in Appendix B.2).

The results of the quantum implementation are shown in Figure 5.5. The value $\hat{f}_b$ indicates the output of the quantum cosine classifier using (x_b, y_b) as training set. The experiments using the QASM simulator (top plot) show an equivalence between the probability of $x^{(\text{test})}$ to be classified in class 1 (blue bar) and the same probability

	Dataset				
	X_1	X_2	y	$d(\cdot,\cdot)$	$Pr(y^{(\text{test})}=1)$
x_1	1	3	0	0.89	0.10
x_2	-2	2	1	0	0.50
x_3	3	0	0	0.71	0.25
x_4	3	1	1	0.89	0.90
$x^{(\text{test})}$	2	2	?	1.0	/

Table 5.1 Each row of the table corresponds to a possible training observation. X_1 and X_2 are the features, $d(\cdot,\cdot)$ is the cosine distance of the training point from $x^{(\text{test})}$ and $Pr(y^{(\text{test})}=1)$ is the predicted probability computed classically (see Equation (5.18)).

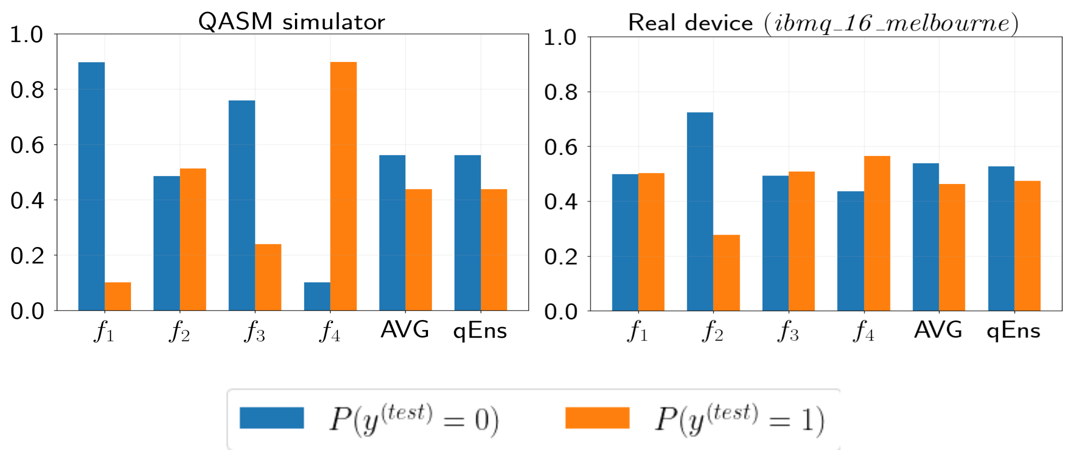


Fig. 5.5 Quantum results based on data in Table 5.1. The labels $\hat{f}_{b=1,\cdots,4}$ indicate the estimated probabilities for $x^{(\text{test})}$ given the b^{th} observation as training set. The AVG bars are obtained by averaging the individual classifiers, while qEnsemble represents the prediction of the quantum ensemble.

computed classically (column $Pr(y^{(\text{test})}=1)$ of Table 5.1). Also, the quantum estimate (qEnsemble) matches perfectly the classical ensemble prediction computed by averaging the four classifications (AVG), as expected. The agreement, however, deteriorates when running on a real quantum device (bottom plot).

In order to generalise the results of the quantum ensemble beyond the dataset in Table 5.1, we performed the same experiment on 20 randomly generated datasets, and we compared the average of the quantum cosine classifiers with the quantum ensemble prediction. Results are shown in Figure 5.6. In this case, the agreement between the quantum ensemble (orange line) and the average (brown dots) is almost perfect, which

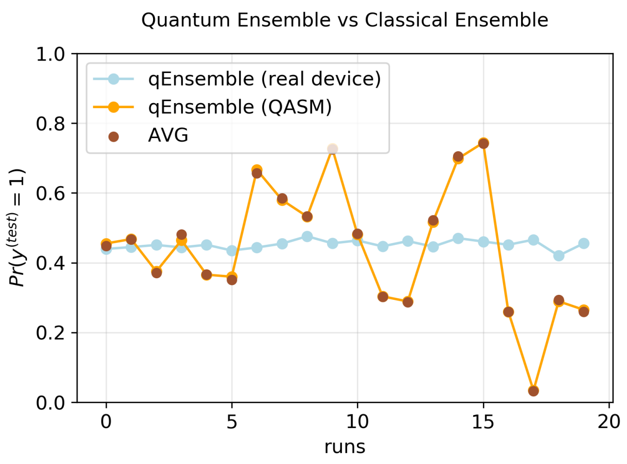


Fig. 5.6 Comparison between the quantum ensemble (qEnsemble) and the average of the four quantum cosine classifiers executed separately (AVG, brown dots), which is computed classically. The simulation of the circuit on the QASM simulator is illustrated in orange, while the light blue line depicts the behaviour on a real device (*ibmq_16_melbourne*).

confirms the possibility to perform quantum ensemble with the advantages described in Section 5.2. Results considering the real device (light blue line) show significant deterioration, this may be due to the depth of the quantum circuit which seems to be prohibitive considering current available quantum technology.

5.3.3 Performance of Quantum Ensemble

To show that the quantum ensemble outperforms the single classifier we generated a simulated dataset and compared the performance of the two models (the pseudo code of the ensemble is described in the Appendix B.4). In particular, we drew a random sample of 200 observations (100 per class) from two independent bivariate Gaussian distributions, with different mean vectors and the same covariance matrix (Figure 5.7). Then, we used the 90% of the data for training and the remaining 10% for testing.

Notice that, given the limitations of the present quantum technology and the definition of the cosine classifier, we need to execute the classification routine once for each test point (See Appendix B.2 for more details). We considered two performance metrics, accuracy and Brier Score. The accuracy is the fraction of labels predicted correctly by the quantum model, and it is evaluated using a set of observations not employed for training (test set). Instead, the Brier Score (BS) measures the difference

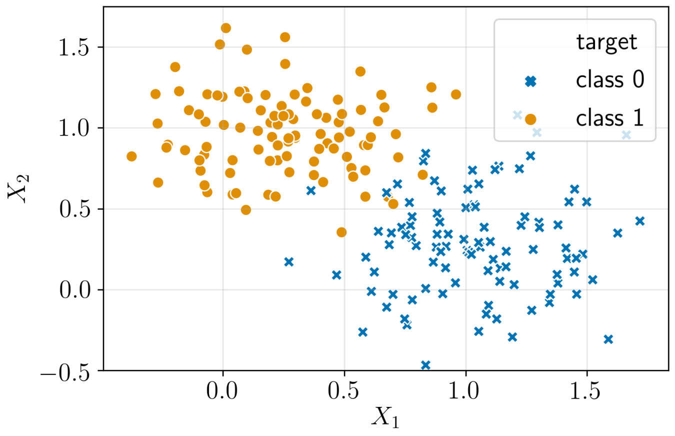


Fig. 5.7 Dataset generated by two independent bivariate Gaussian distributions. Mean vectors for the two classes are $(1, 0.3)$ and $(0.3, 1)$. The two distributions have the same diagonal covariance matrix, with constant value of 0.3.

between the probability estimates and the true label in terms of mean squared error:

$$BS = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left[y_i - f(x_i)\right]^2 , \tag{5.20}$$

where N_{test} is the number of observation in the test set, y_i and $f(x_i)$ are, respectively, the true label and the probability estimates provided by the quantum model for the i^{th} observation (for quantum ensemble see Equation (5.11)). Hence a low BS score implies a good prediction.

Because of the random component in the models due to the choice of training points, we repeated the experiments 10 times and evaluated the classifiers in terms of mean and standard deviation of both accuracy and Brier Score.

Results are shown in Table 5.2. The single quantum cosine classifier performed only slightly better than random guessing, with an average accuracy of 55%. Yet, the quantum ensemble managed to achieve definitely better results, with both metrics improving as the ensemble size grows.

Variance Analysis

In addition, we investigated how the quantum ensemble behaves as the generated distributions get closer and less separated. To this end, we drew multiple samples from the two distributions, each time increasing the common standard deviation so to force reciprocal contamination. Results are reported in Figure 5.8.

			Accuracy		Brier Score	
d	B	N. train	Mean	Std dev	Mean	Std dev
0	1	1	.55	.09	.21	.05
1	2	2	.92	.09	.14	.09
2	4	4	.91	.09	.15	.05
3	8	8	.96	.04	.14	.04
4	16	8	.98	.02	.13	.02

Table 5.2 Performance comparison between quantum cosine classifier and quantum ensemble of different sizes $B = 2^d$. The first row indicates the performance of the single quantum cosine classifier. The column *N. train* indicates the number of training points used to build the ensemble, that is limited to 8 because of limited number of qubits that is possible to simulate using a simulator.

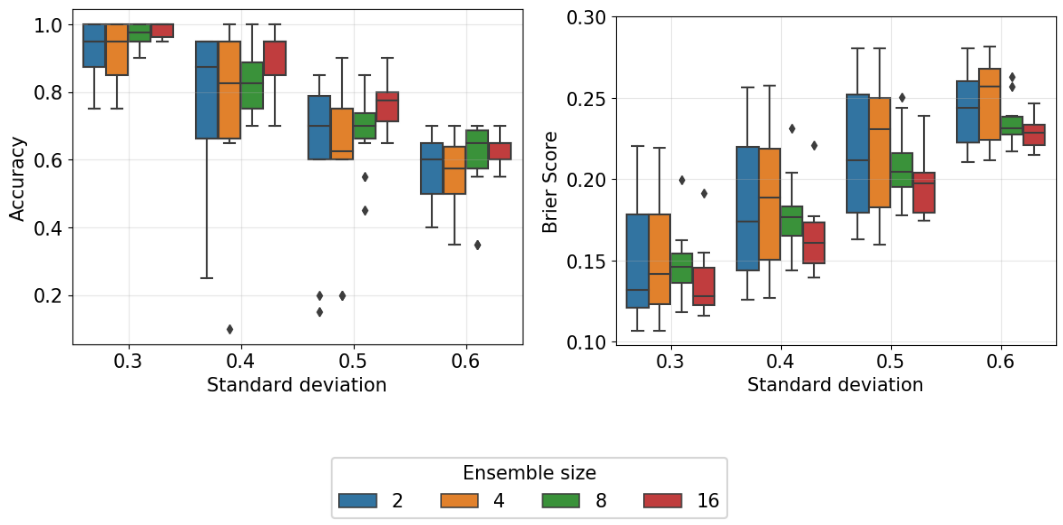


Fig. 5.8 Distribution of the performance metrics as a function of the ensemble size (legend colors) and the separation between the two classes (x axis).

The accuracy showed a decreasing trend as the overlap of the distributions increased. The opposite behaviour is observed for the Brier Score. Also, the shape of the boxplots is much narrower for greater ensemble sizes (green and red boxplots) than for smaller ones (blue and orange). Hence, this confirms that the variability of the ensemble decreases as the number of weak learners adopted grows, as expected.

5.4 Benchmark on real-world datasets

In this section we test the performance of the quantum ensemble on real-world datasets that are usually employed to benchmark classical machine learning algorithms.

5.4.1 Datasets description

As discussed in the previous chapters, the simulation of a quantum system on a classical device is a challenging task, even for systems of moderate size. For this reason, experiments consider only datasets with a relatively small number of observations (100–150) that will be split in training (90%) and test (10%) set. Furthermore, in order to limit the overall number of qubits, the Principal Components Analysis (PCA) (Pearson, 1901) is employed to reduce the number of features to 2. PCA is an unsupervised technique which computes a small number of new variables that, in a certain sense, summarise the original input set. The matrix of correlation coefficients derived from the original data is decomposed to obtain a new set of variables whose composition depends on observed correlations. PCA usually requires the variables to be measured on a continuous scale, but it can also be employed with ordinal features. The performance of PCA is measured by the explained variance of the new set of features. In fact, the total variance of the new set of features is equal to the total variance of the original input features. The difference is that the most important subset of the new features embeds most of the original dataset's variability. Thus, the ratio between the variance explained by the selected principal components and the total variance measures how well the PCA performed.

For all the datasets, the quantum algorithm will be trained on the training set, and the test error[3] (the error measured on the test set) is considered to evaluate the generalisation error of the quantum model.

MNIST

MNIST (Modified National Institute of Standards and Technology) data is a large dataset of handwritten digits that is commonly used to benchmark various image processing systems. In particular, it is usually employed for training and testing different algorithms in the field of machine learning. The dataset contains 60.000 black and white images, each represented by 28×28 pixels. Thus, the single image can be described as a vector of binary 784 features (0 if the pixel is white, 1 if it is black). Also, each image belongs to a class of ten possible, that represents the digit

[3]in terms of Accuracy and Brier Score

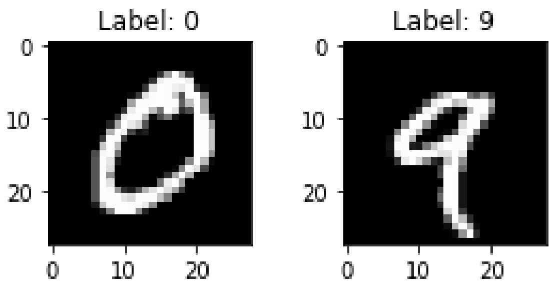


Fig. 5.9 Image representation of the digits 0 and 9.

depicted in the image. As discussed in Section 5.3, the current implementation of the quantum ensemble is arranged to solve a binary classification problem. Hence, only two different classes will be considered, the digits 0 and 9 (Figure 5.9). In order to

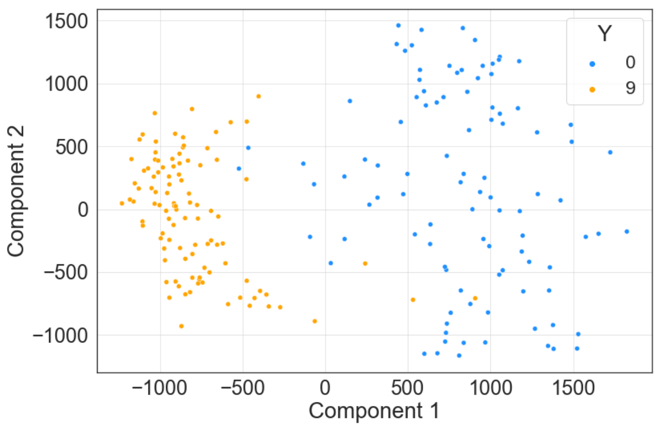


Fig. 5.10 Scatterplot of the two first principal components of the MNIST dataset. The explained variance by these two components is 31%.

reduce the number of features and, consequently, the number of qubits needed for the implementation, the original feature space is transformed using the PCA. This allows generating a new set of features that results from the linear combination of the initial 784 binary features. The coefficients of the linear combinations are calculated according to the spectral decomposition of the correlation matrix of data.

Iris

The Iris flower data set collects the data to quantify the morphologic variation of Iris flowers of three related species (Anderson, 1936). The data set consists of 50 examples from each of three species of Iris (Iris *setosa*, Iris *virginica* and Iris *versicolor*). Four features describe each observation: the length and the width of the sepals and petals, in centimetres (Figure 5.11).

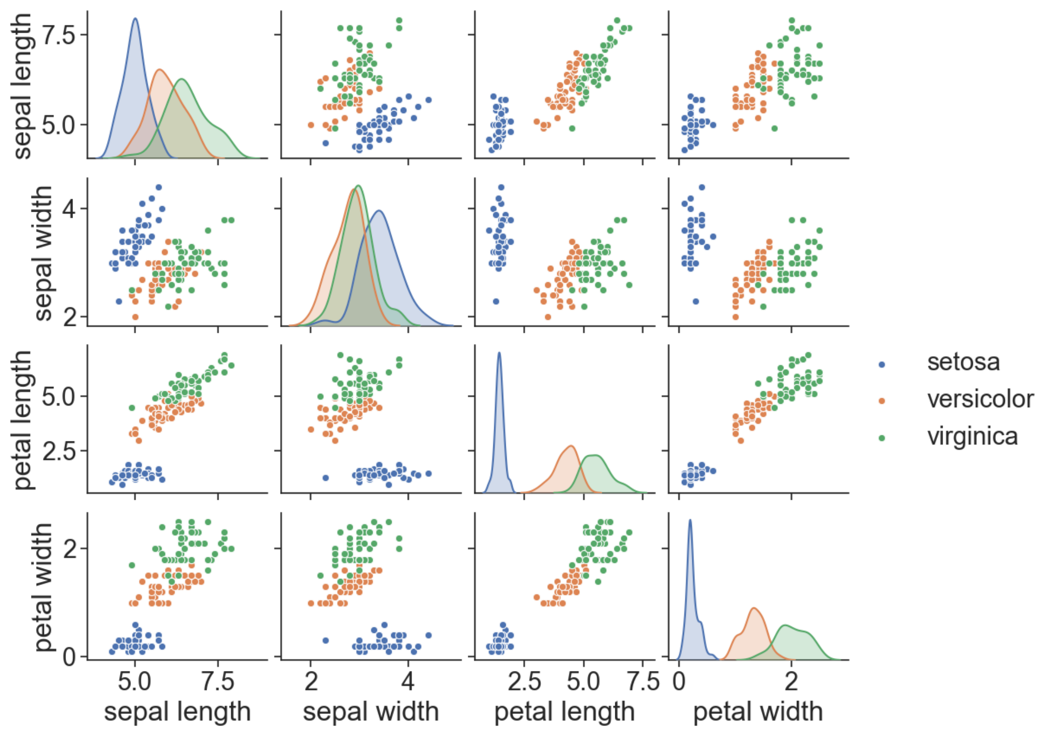


Fig. 5.11 Original scatterplot matrix of the Iris dataset.

The dataset is often used in statistical learning theory as classification and clustering examples and to test algorithms. In order to test the quantum ensemble on iris dataset, it is necessary to employ only two features. Thus the PCA is performed to extract only two principal components as new features. Also, the binary classification problem will be considered, hence, only two classes at time (three different datasets). The explained variance of the PCA performed to separate the classes *virginica* and *versicolor* is 92.3%. In the other cases (*setosa vs versicolor* and *setosa vs virginica*), the explained variance is around 98%.

Breast cancer

Breast cancer data is a dataset containing information about the presence of breast cancer in a given set of patients. The data set contains 569 rows and 32 columns. The target variable says if the cancer is $M = malignant$ or $B = benign$. The value 1 of the target variable means the cancer is malignant, 0 means benign. The features describe the characteristics of the cell nuclei present in the image.

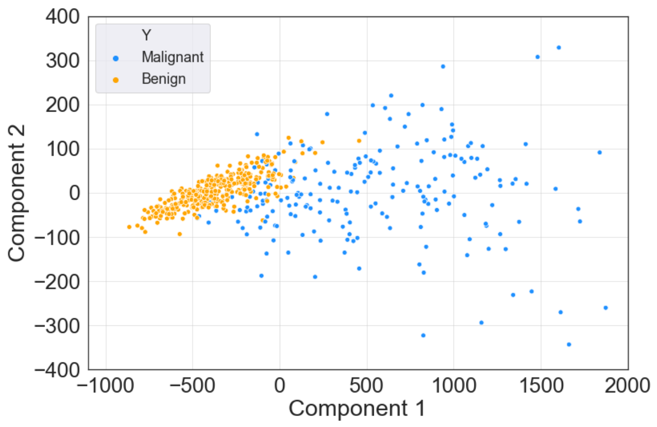


Fig. 5.12 Scatterplot of the two firsts principal components of the reduced Breast dataset. The explained variance is 99.8%.

In order to employ this set of data, PCA will be applied, together with resampling that allows perfectly balancing the observations in the two classes. Data after the preprocessing are depicted in Figure 5.12

5.4.2 Results

The results of quantum ensemble on the real-world datasets are reported in Figure 5.13. As for simulated data, we investigated the behaviour of the quantum ensemble as the number of base models increases.

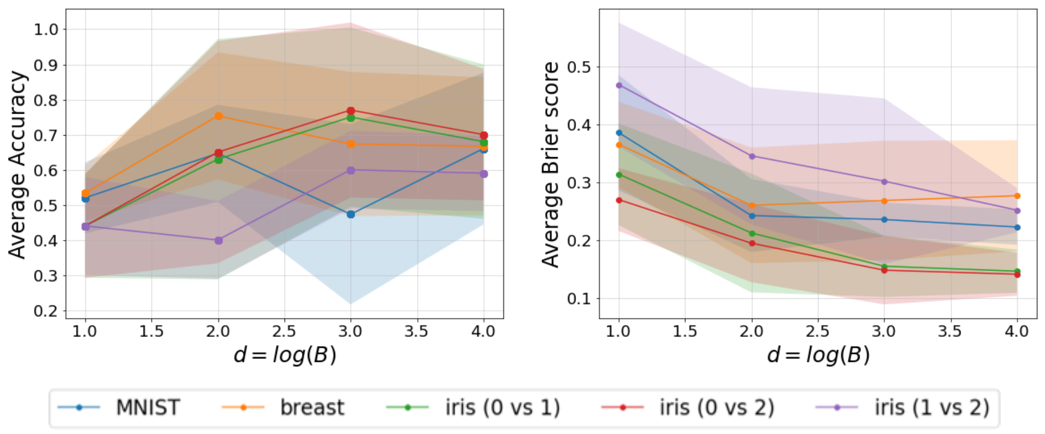


Fig. 5.13 Performance of the quantum ensemble on real-world datasets.

Comparing the results in terms of the ensemble size ($B = 2^d$), it is possible to observe a decreasing trend of the Brier score and an increasing trend for accuracy. This confirms the ability of the quantum ensemble to improve the performance of the single quantum classifier.

5.5 Conclusion

In this chapter, we proposed the use of MAQA to build a quantum algorithm for binary classification based on ensemble learning. The correspondent algorithm allows generating a large number of trajectories in superposition, performing just one state preparation routine. Each trajectory is entangled with a quantum state of the *control* register and represents a single classifier. This convenient design allows scaling the number of base models exponentially with respect to the available qubits in the control register ($B = 2^d$). As a consequence, we introduce an exponential growth in the ensemble size with respect to the classical counterpart. Furthermore, when considering the overall time complexity of the algorithm, the cost of the weak classifier is additive, instead of multiplicative as it usually happens.

In addition, we presented a practical implementation of the quantum ensemble using bagging where the quantum cosine classifier is adopted as base model. In particular, we showed experimentally that the ensemble prediction corresponds to the average of all the probabilities estimated by the single classifiers. Moreover, we tested our algorithm on synthetic and real-world datasets and demonstrated that the quantum ensemble systematically outperforms the single classifier by improving the accuracy and decreasing the prediction variance. Also, the variability decreases as we added more base models to the ensemble.

However, the currently proposed implementation requires the execution of the classifier for just one test point at the time, which is a big limitation for practical applications. In this respect, the main challenge to tackle to make the ensemble effective in the near future is the design of a quantum classifier based on interference that guarantees a more efficient data encoding strategy (e.g. amplitude encoding), and that can process larger datasets. However, these upgrades would imply a different definition of $U_{(i,j)}$ for generating multiple and diverse training sets in superposition.

Another natural follow-up is the implementation of quantum algorithms for randomisation and boosting. In this work, we only referred to an ensemble based on bagging because the learning step was performed independently in each quantum trajectory and the weak classifiers were assumed to be sensitive to perturbations of the training set. However, with appropriate amendments and loosening these constraints, we believe that it is possible to design other types of ensemble techniques.

Chapter 6

qSLP: Quantum Single Layer Perceptron

In this chapter, we propose the adoption of MAQA (Chapter 4) to implement a Quantum Neural Network introducing a novel variational algorithm for quantum Single Layer Perceptron (qSLP). Thanks to the universal approximation theorem, and given that the number of hidden neurons scales exponentially with the number of qubits, the use of MAQA in the context of quantum variational algorithms opens to the possibility of approximating any function on near-term quantum computers.

The proposed approach produces a model with substantial descriptive power, and widens the horizon of potential applications already in the NISQ era, especially the ones related to Quantum Artificial Intelligence. In particular, we design a quantum circuit to perform parametrised linear combinations in superposition where the parameters of the quantum algorithm can be learned using hybrid quantum-classical computation and discuss adaptations to classification tasks. After this theoretical investigation, we also provide practical implementations using various simulated and real-world datasets. Finally, we test the proposed algorithm on both simulators and real quantum devices.

6.1 Motivation

In the Chapter 1.5 we discussed that one of the topics in which QC may have a higher impact is Quantum Machine Learning (QML), a sub-discipline intended as to developing quantum algorithms that learn from data. However, the ability to deliver a significant boost in performance through quantum algorithms on near-term devices is still to be demonstrated. Given these premises, Neural Networks (NNs) are among the most desired targets when coming to transposing classical models into their quantum

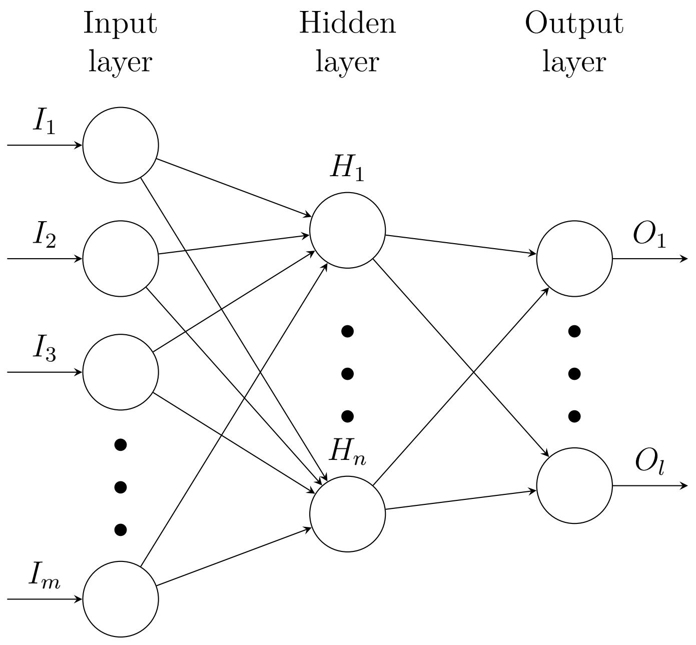


Fig. 6.1 Diagram representation of a single hidden layer neural network.

counterpart. In fact, NNs have shown remarkable performances in many real-world applications and multiple learning tasks, including clustering, classification, regression and pattern recognition.

In the following sections, we introduce a general model framework that reproduces a quantum state equivalent to the output of a classical Single Layer Perceptron (SLP). This is achieved by implementing an efficient variational algorithm based on MAQA that performs linear combinations in superposition. The results are then passed altogether through an activation function with just one application. Importantly, the framework supports pluggable activation function routines, thus allowing an easy way to adapt the approach to different use cases. In Section 6.2, we discuss the first prototype of the quantum SLP (Macaluso et al., 2020c) which produces the output of a two-neuron single layer neural network quantumly. Section 6.2.5 is devoted to practical experiments to test our model as a classifier. Finally, Section 6.4 describes how our approach can be extended to the case of more hidden neurons.

6.1.1 Neural Network as Universal Approximator

A Single Hidden Layer Neural Network (or Single Layer Perceptron - SLP) (Hastie et al., 2001) is a two-stage model that can be used for both classification and regression, typically represented by a network diagram as in Figure 4.1.

Given a training point (x_i, y_i), the output of a feedforward Neural Network with a single hidden layer containing H neurons can be expressed as:

$$f(x_i) = \sigma_{\text{output}} \left(\sum_{j=1}^{H} \beta_j \sigma_{\text{hidden}} \left(L(x_i; \theta_j) \right) \right), \tag{6.1}$$

where σ_{output} is the identity when the task is to approximate a function. Each hidden neuron j computes a linear combination of the input features x_i with coefficients equal to the p-dimensional vector θ_j. This operation is performed for all the neurons, and the results are then individually convoluted with the inner activation function σ_{hidden}. Despite being more straightforward than fancier, deep architectures proposed in recent years, the SLP model is very expressive and capable of mimicking diverse and complex functions. According to the *universal approximation theorem*, in fact, a single hidden layer Neural Network with a non-constant, bounded and continuous activation function can approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$, provided that enough hidden neurons are specified. This result implies that if we manage to build an algorithm that reproduces an SLP on a quantum computer, then we have automatically access to an enormously powerful tool that is potentially able to solve any approximation problem.

6.2 Variational Algorithm for Single Hidden Layer Neural Network

In this section, we propose a new variational algorithm reproducing a quantum Single Layer Perceptron, whose output is equivalent to the classical counterpart. In particular, building on top of the approach described in Section 4.1, we design an algorithm that allows efficient computation using just mild constraints on the input. Also, the flexible architecture enables to plug in custom implementations of the activation function routine, thus adapting to different use cases. Thanks to the possibility of learning the parameters for a given task, the proposed framework allows training models that can potentially approximate any function.

However, we do not address the problem of implementing a non-linear activation function. Our goal is to provide a framework that generates multiple linear combinations in superposition entangled with a control register. In this way, instead of executing a given activation function for each hidden neuron, a single application is needed to propagate it to all of the quantum states. This allows scaling the number of hidden neurons exponentially with the number of qubits, thus enabling the qSLP to be a concrete alternative for approximating complex and diverse functions.

6.2.1 Encode Data in Amplitude encoding

The first issue to address when using a quantum computer for data analysis is *state preparation*, i.e. the design of a process that loads the data from a classical memory to a quantum system. The most general encoding adopted in QML is amplitude encoding (Section 3.4). This strategy associates quantum amplitudes with real vectors of observations at the cost of introducing just normalisation constraints. Formally, a normalised vector $x \in \mathbb{R}^{2^n}$ can be described by the amplitudes of a quantum state $|x\rangle$ as:

$$|x\rangle = \sum_{k=1}^{2^n} x_k \, |k\rangle \longleftrightarrow x = \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix}. \tag{6.2}$$

In this way, it is possible to use the index register to indicate the k-th feature. The main advantage of this encoding is that we only need n qubits for a vector of $p = 2^n$ elements. This means that, if a quantum algorithm is polynomial in n, then it will have a polylogarithmic runtime dependency on the data size. A possible strategy for amplitude encoding has been proposed by Mottonen et al. (2004), which is the one used for experiments in this work. The goal of this approach is to map an arbitrary state $|x\rangle$ to the ground $|0 \dots 0\rangle$. Once the circuit is obtained, then all of the operations are inverted and applied in the reversed order.

6.2.2 Activation function

The implementation of a proper activation function – in the sense of the Universal Approximation Theorem – is one of the major issues for building a complete quantum Neural Network. This is due to the restrictions to linear and unitary operations imposed by the laws of quantum mechanics (Nielsen and Chuang, 2011). One of the most famous attempt to solve this problem is described in (Cao et al., 2017), where the authors use the repeat-until-success technique to achieve non-linearity. However, a significant limitation is the requirement of inputs in the range $\left[0, \frac{\pi}{2}\right]$, which is a severe constraint for real-world problems. Also, the idea of quantum spline (QSpline) (Macaluso et al., 2020b) has recently been proposed to approximate non-linear activation functions via a quantum algorithm. Although the QSpline provides a fitting method to store the value of a non-linear function in the amplitudes of a quantum state, it makes use of the HHL as sub routine, which is a full-coherent protocol and has high computational requirements. In the next chapter a detailed discussion about QSpline will be provided.

In this chapter, we do not discuss how to implement in practice a non-linear activation function. However, we provide a framework that permits to train a quantum SLP for a given activation function Σ. Our architecture is naturally capable of incorporating any implementation of an activation function whose parameters can be learned, like, for instance, the one described in Hu (2018). Indeed, we can think of extending the circuit that trains the qSLP to also learn the activation parameters. For this reason, new implementations of non-linear activation functions are naturally pluggable in the proposed framework as long as they fit in a learning paradigm.

6.2.3 Gates as Linear Operators

A variational circuit $U(\theta)$ is composed of a series of gates, each one possibly parametrised by a set of parameters $\{\theta_l\}_{l=1,\dots,L}$. Formally, $U(\theta)$ is the product of matrices:

$$U(\theta) = U_L \cdots U_l \cdots U_1, \tag{6.3}$$

where each U_l is composed of a single-qubit or a two-qubit quantum gate. In order to make the single-qubit gate trainable it is necessary to formulate U_l in terms of parameters that can be learned. This is possible by adopting a single-qubit gate G which is defined as the following 2×2 unitary matrix (Barenco et al., 1995):

$$G(\alpha, \beta, \gamma) = \begin{pmatrix} e^{i\beta} cos(\alpha/2) & e^{i\gamma} sin(\alpha/2) \\ -e^{-i\gamma} sin(\alpha/2) & e^{-i\beta} cos(\alpha/2) \end{pmatrix}. \tag{6.4}$$

Thus, we can now express each U_l in terms of single-qubit gates, G_i, acting on the i-th qubit:

$$U_l = \mathbb{1}_1 \otimes \cdots \otimes G_i \otimes \cdots \otimes \mathbb{1}_n, \tag{6.5}$$

where n is the total number of qubits of the quantum system. This representation of $U(\theta)$ is convenient since it allows computing the gradient analytically, as shown in Schuld et al. (2018).

Alternatively, we can express Equation (6.4) using complex numbers $z, u \in \mathbb{C}$ instead of trigonometric functions:

$$G(z, v) = \begin{pmatrix} z & v \\ -v^* & z^* \end{pmatrix}, \tag{6.6}$$

where $|z|^2 + |v|^2 = 1$. This parametrisation avoids non-linear dependencies between the circuit parameters and the model output. Notice that the definition of linear operator given in Equation (6.6) involves complex coefficients. Therefore, it describes a more general operation with respect to the classical counterpart adopted in an SLP, that only allows for linear combinations with real-valued coefficient. Nonetheless, one can still parametrise the circuit using Pauli-Y rotation in case one wants to restrict the computation to the real domain.

6.2.4 Original prototype of Quantum Single Hidden Layer Network

In this section we discuss the proposal of a quantum Single Layer Perceptron with two neurons in the hidden layer (Macaluso et al., 2020c). The generalisation of the algorithm which makes use of MAQA framework is then discussed in Section 6.4.

Intuitively, a qSLP can be implemented into a quantum computer in two steps. Firstly, we generate different linear operations in superposition, each one having different parameters θ_j, entangled with a control register. Secondly, we propagate the activation function to all the linear combinations in superposition. Notice that, thanks this approach, instead of executing a given activation function for each hidden neuron, we need only one application to obtain the output of all the neurons in the hidden layer. To this end, three quantum registers are necessary: *control*, *data* (denoted by $|\psi\rangle$) and *temporary* register ($|\phi\rangle$). The latter is responsible for generating the linear combinations of the input data in superposition. Also, it can be in any arbitrary state, possibly even unknown.

The algorithm is composed of five main steps: *state preparation*, *entangled linear operators in superposition*, *application of the activation function*, *read-out step*, *post-processing*.

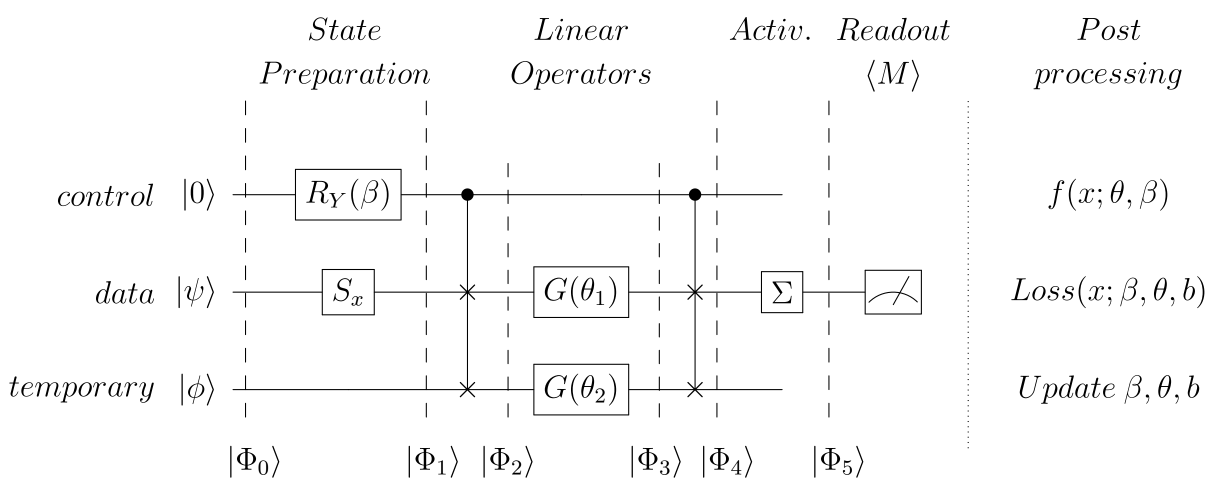


Fig. 6.2 Quantum circuit for training a qSLP.

(Step 1) The state preparation includes encoding the data, x, in the amplitude of $|\psi\rangle$ and applying a parametrised Y-rotation $R_y(\beta)$ to the control qubit:

$$\begin{aligned}|\Phi_1\rangle &= \Big(R_y(\beta)\otimes S_x \otimes \mathbb{1}\Big)\,|\Phi_0\rangle = \Big(R_y(\beta)\otimes S_x \otimes \mathbb{1}\Big)\,|0\rangle\,|0\rangle\,|\phi\rangle \\ &= (\beta_1\,|0\rangle + \beta_2\,|1\rangle)\otimes |x\rangle \otimes |\phi\rangle = \beta_1\,|0\rangle\,|x\rangle\,|\phi\rangle + \beta_2\,|1\rangle\,|x\rangle\,|\phi\rangle), \end{aligned} \tag{6.7}$$

where S_x indicates the routine that encodes the data, $|\beta_1|^2 + |\beta_2|^2 = 1$ and $\beta_1, \beta_2 \in \mathbb{R}$.

(Step 2) We exploit the idea of quantum forking (Park et al., 2019) to generate two different linear operations in superposition, each entangled with the control qubit.

2.1 The first controlled-swap is applied to swap $|x\rangle$ with $|\phi\rangle$ if the control qubit is equal to $|1\rangle$:

$$|\Phi_2\rangle = \frac{1}{\sqrt{E}}\Big(\beta_1\,|0\rangle\,|x\rangle\,|\phi\rangle + \beta_2\,|1\rangle\,|\phi\rangle\,|x\rangle\,\Big), \tag{6.8}$$

where E is a normalisation constant.

2.2 Two linear operations parametrised by two different sets (θ_1 and θ_2) act on $|\psi\rangle$ and $|\phi\rangle$ respectively:

$$\begin{aligned} |\Phi_3\rangle &= \Big(\mathbb{1} \otimes G(\theta_1) \otimes G(\theta_2)\Big) |\Phi_2\rangle \\ &= \frac{1}{\sqrt{E}}\Big(\beta_1 |0\rangle G(\theta_1) |x\rangle |\phi\rangle + \beta_2 |1\rangle |\phi\rangle G(\theta_2) |x\rangle\Big) \\ &= \frac{1}{\sqrt{E}}\Big(\beta_1 |0\rangle |L(x;\theta_1)\rangle |\phi\rangle + \beta_2 |1\rangle |\phi\rangle |L(x;\theta_2)\rangle\Big). \end{aligned} \tag{6.9}$$

2.3 Then, the second controlled-swap is executed to swap $|L(x;\theta_2)\rangle$ with $|\phi\rangle$ if the control qubit is equal to $|1\rangle$:

$$|\Phi_4\rangle = \frac{1}{\sqrt{E}}\Big(\beta_1 |0\rangle |L(x;\theta_1)\rangle |\phi\rangle + \beta_2 |1\rangle |L(x;\theta_2)\rangle |\phi\rangle\Big). \tag{6.10}$$

Finally, the two linear operations are stored in $|\psi\rangle$ and are then entangled with one state of the control qubit. At this point, a routine is necessary to propagate the activation function in both the trajectories of $|\psi\rangle$.

(Step 3) Activation function:

$$\begin{aligned} |\Phi_5\rangle &= \Big(\mathbb{1} \otimes \Sigma \otimes \mathbb{1}\Big) |\Phi_4\rangle \\ &= \frac{1}{\sqrt{E}}\Big(\beta_1 |0\rangle \Sigma |L(x;\theta_1)\rangle |\phi\rangle + \beta_2 |1\rangle \Sigma |L(x;\theta_2)\rangle |\phi\rangle\Big) \\ &= \frac{1}{\sqrt{E}}\Big(\beta_1 |0\rangle \Big|\sigma_{hid}\big[L(x;\theta_1)\big]\Big\rangle |\phi\rangle + \beta_2 |1\rangle \Big|\sigma_{hid}\big[L(x;\theta_2)\big]\Big\rangle |\phi\rangle\Big). \end{aligned} \tag{6.11}$$

At the end of Step 3 the two linear operations, $L(\cdot)$, are put through the same activation function, σ_{hid}, represented by the gate Σ[1]. The results are then encoded in the quantum register $|\psi\rangle$. Each output is finally weighed by the parameters of the control qubit (β), i.e. the coefficients attached to the hidden neurons in the linear combination that produces the output of the NN. This is exactly the quantum version of the two-neurons classical SLP presented in Equation (4.3).

[1]In the current literature, a Σ gate that reproduces non-linear functions quantumly doesn't exist yet. For experiments, the identity function will be employed. However, Chapter 7 introduces the proposal of Quantum Splines, a full-coherent protocol that assumes a perfect quantum device to approximate popular non-linear activation functions usually adopted for classical neural networks.

(Step 4) The measurement of $|\psi\rangle$ can be expressed as the expected value of the Pauli-Z operator acting on the quantum state $|x\rangle$:

$$\langle M\rangle = \langle\Phi_0|\, U^\dagger(\beta,\theta)(\mathbb{1}\otimes\sigma_z\otimes\mathbb{1})U(\beta,\theta)\,|\Phi_0\rangle = \pi(x;\beta,\theta), \tag{6.12}$$

where $U(\beta,\theta)$ represents the qSLP circuit. In order to get an estimate of $\pi(\cdot)$, we have to run the entire circuit multiple times.

(Step 5) The post-processing is performed classically and is task-dependent. For classification models we need four steps: (i) adding a learnable bias term b to produce a continuous output, (ii) applying a thresholding operation, (iii) computing the loss function and (iv) updating the parameters. Notice that all these steps are customisable and can be adapted to the particular needs of the application. In the case of the experiments presented in Section 6.2.5 we adopt the following thresholding operation:

$$f(x_i;\beta,\theta,b) = \begin{cases} 1 & \text{if } \pi(x_i;\beta,\theta)+b > 0.5 \\ 0 & \text{else }, \end{cases} \tag{6.13}$$

where b is the bias term and $f(x_i;\beta,\theta,b)$ gives us the predicted class for observation x. As loss function we choose the *Sum of Squared Errors* (SSE) between the predictions and the true values y:

$$SSE = Loss(\Theta;D) = \sum_{i=1}^{N}\left[y_i - f\left(x_i;\Theta\right)\right]^2, \tag{6.14}$$

where N is the total number of observations in the sample and $\Theta = \{\beta,\theta,b\}$. Finally, we exploit the Nesterov accelerated gradient method for updating the parameters, although many alternative optimisation strategies can be adopted to update the parameters (Ruder, 2016) .

To summarise, the variational algorithm described above allows reproducing a classical Neural Network with one hidden layer on a quantum computer. In particular, it includes a variational circuit adopted for encoding the data, performing the linear combinations of input neurons and applying the same activation function to their results with just one execution. A single iteration during the learning process is then completed using classical resources to measure the output of the network, compute the loss function and update the parameters. The whole process is then repeated iteratively until convergence, as for classical Neural Networks.

As a final remark, notice that having a post-processing step that is extremely flexible enables the adoption of this model both for regression and classification problems, thus enhancing the impact of such algorithm.

6.2.5 Experiments

To test the performances of the qSLP, we implemented the circuit illustrated in Figure 6.2 using Qiskit (Figure 6.3) and Pennylane (Bergholm et al., 2018), two software frameworks for optimisation and quantum computation. These libraries can be used for both quantum and hybrid computations, and allow using quantum objects (e.g. qubits, gates) in conjuction with classical elements (e.g. variables, functions). They can handle many learning tasks such as training a hybrid ML model in a supervised fashion.

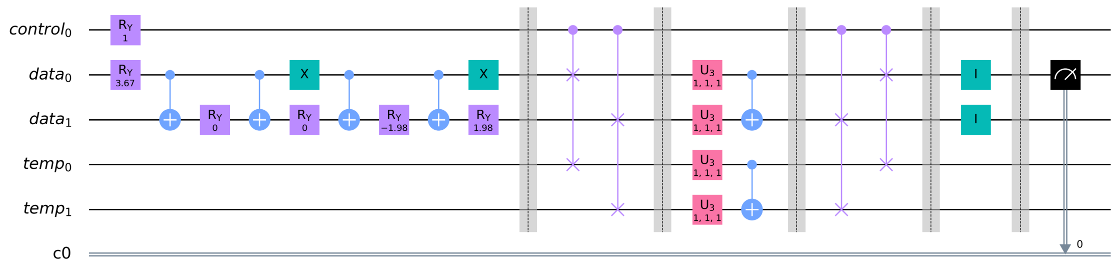


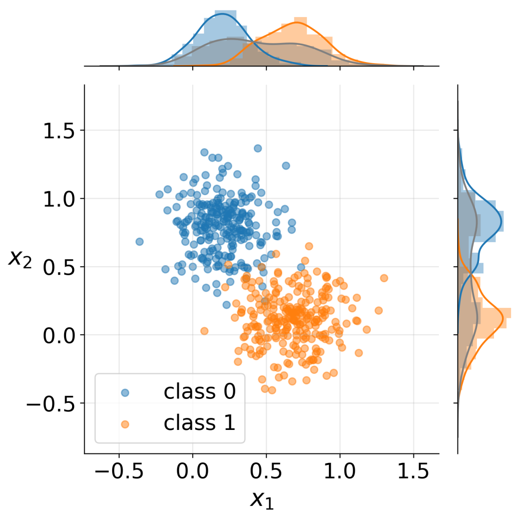
Fig. 6.3 Qiskit implementation of the two-neuron qSLP in Figure 6.2. The first step consists in preparing the *data* and the *control* register. Then the qubits of *data* and temporary (*temp*) registers are swapped through controlled-swap. Two parametrised unitary transformations are applied, each represented by two single-qubit rotation and a CNOT. Finally, the controlled-SWAP is applied a second time, and the sigma function (identity matrix) is implemented to the data register.

In addition, thanks to qiskit, we are able to execute the pre-trained algorithm obtained on a real device[2]. In our case, the goal is to find the parameters of the quantum circuit (β, θ) plus the additional bias term b. In absence of a gate Σ which implements a non-linear activation function, the final quantum state of $|\psi\rangle$ is:

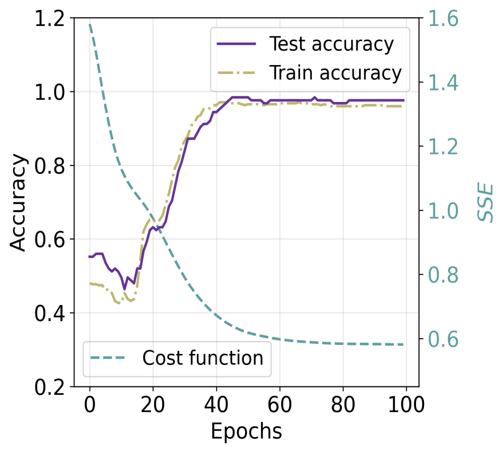
$$|\Phi_5\rangle = \frac{1}{\sqrt{E}}\Big(\beta_1 |0\rangle |L(x;\theta_1)\rangle |\phi\rangle + \beta_2 |1\rangle |L(x;\theta_2)\rangle |\phi\rangle\Big), \tag{6.15}$$

which is a linear transformation of the input data and defines a linear classifier. Notice that $Pr\left[y_i = 1|x_i\right]$ for a given observation x_i corresponds to the square of the linear

[2]The use of a specific device (santiago, vigo) depends on the availability at the time of the algorithm execution.



(a)



(b)

Fig. 6.4 The plot on the left illustrates the distributions of generated data in the two classes $(0, 1)$. The plot on the right shows the trends over training epochs of the cost function and the accuracy.

transformation of hidden neurons with coefficients β_j plus a bias term, b. In practice, we generated linearly separable data to test our classifier. In particular, we drew a random sample of 500 observations (250 per class) from two independent bivariate Gaussian distributions, with different mean vectors and the same covariance matrix (Figure 6.4a). Then, we used the 75% of the data for training and the remaining 25% for testing. The training metrics for the model trained on the PennyLane simulator are illustrated in Figure 6.4b.

The results demonstrate that the quantum SLP is able to classify correctly the observations, as testified by the high classification accuracy in both training and test sets, 0.97 and 0.95 respectively.

After the model was trained, the variational algorithm was also implemented using Qiskit, and its performance was tested on 50 newly-generated observations. In this way, it was possible to test the pre-trained model on both the QASM simulator – which emulates the execution of a quantum circuit on a real device, also including highly configurable noise models – and a real device. Results are reported in Table 6.1. The PennyLane implementation was in line with the training results, and was the most accurate (94% accuracy), as expected since the framework assumes a perfect device. The effects of introducing the intrinsic noise due to quantum computations, instead, can be appreciated in the Qiskit implementations. Both alternatives showed lower performances, although the decrease in accuracy was certainly smaller for QASM. The

PennyLane	**QASM**	**IBMQ Vigo**
94%	90%	64%

Table 6.1 Test accuracy of multiple implementations. The performance deteriorates when using a real device. ($ibmq_vigo$)

real device, instead, presented a significant deterioration. This may be due to the depth of the implemented circuit, especially regarding the encoding part, that seems to be prohibitive considering the actual quantum devices.

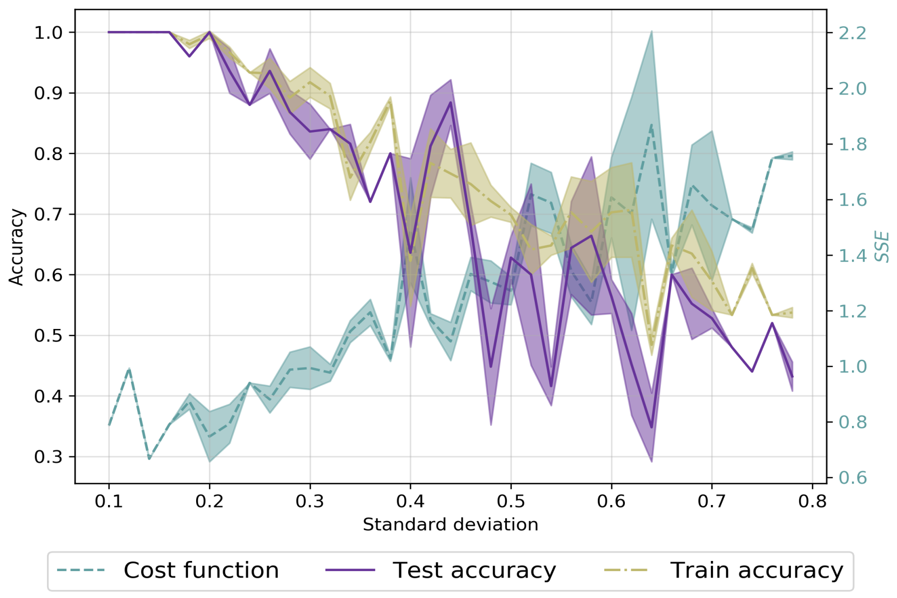


Fig. 6.5 Assessment metrics trend as a function of distributions overlapping. Larger standard deviations cause the two distributions to overlap, so that observations belonging to the two classes are mixed together and, hence, harder to separate. As a consequence, model performances decrease and non-linearity is required.

In addition, we investigated how the performance of the qSLP implemented changes as the generated distributions get closer and less separated. To this end, we drew multiple samples from the two distributions, each time increasing the common standard deviation so to force reciprocal contamination. As expected, the accuracy showed a decreasing trend as the overlap of the distributions increased (Figure 6.5). In conclusion, the experiments show that the proposed architecture works well for linearly separable data. However, performance decreases as we add to the problem a level of complexity that cannot be solved by linear classifiers.

6.3 Experiments on real-world datasets

We tested the performance of the two neurons qSLP on the datasets described in Section 5.4.1. Results are shown in Table 6.2.

	QASM		Real device
Dataset	**Train Accuracy**	**Test Accuracy**	**Test Accuracy**
MNIST (0 vs 9)	.91	.93	.80
Breast	.52	.33	.47
Iris (0 vs 1)	.98	1.0	.60
Iris (0 vs 2)	.99	1.0	.70
Iris (1 vs 2)	.82	.80	.70

Table 6.2 Performance metrics of the qSLP (Figure 6.3) on real-world datasets. The model is trained using QASM simulator and tested on both simulator and real device (*ibmq_santiago*).

We can see that using the QASM simulator as quantum device, the qSLP achieves good performance for all the datasets except the Breast dataset. In particular, considering the Iris dataset, the model is able to distinguish the class 0 (setosa) from the other two almost perfectly, with an accuracy around 99%. Instead, when considering the binary classification problem for separate the classes 1 (versicolor) and 2 (virginica) the model's performance deteriorates with an accuracy of 82%. The image classification achieves good performance with an accuracy of 91%. In all the cases, the test accuracy is in line with the training accuracy, hence the quantum models are not affected by overfitting.

6.4 Generalisation to H hidden neurons

In this section we discuss the generalisation of the quantum SLP to the case of $H > 2$ hidden neurons by using the MAQA.

In order to extend the quantum state in Equation (6.11), two quantum registers are considered: a *data* register whose size depends on the number of input features and a *control* register made by d qubits. The final output of the qSLP will be stored in the *data* register (*output*). Intuitively, the algorithm can be summarised into five steps.

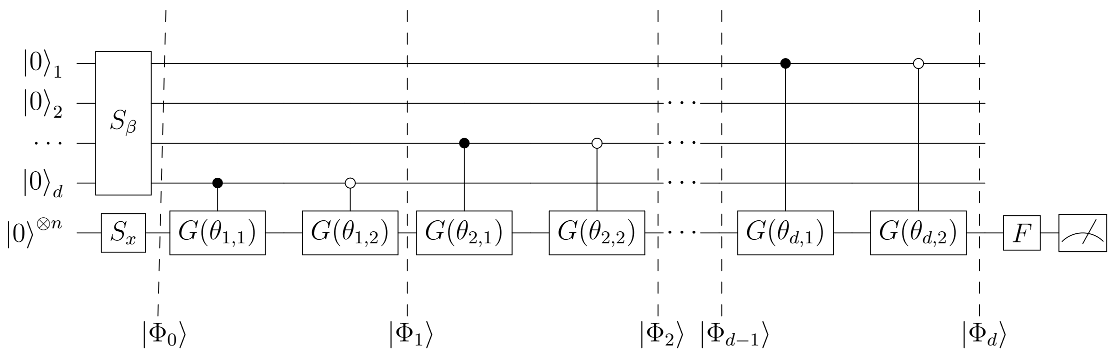


Fig. 6.6 Generalised qSLP that makes use of MAQA.

First, the *control* register is turned into a non-uniform superposition parameterised by the 2^d-dimensional vector β by means of a quantum gate S_β. Also, a single p-dimensional training point is encoded into the n-qubit ($n = log(p)$) *data* register through the quantum gate S_x:

$$|\Phi_0\rangle = (S_\beta \otimes S_x)\,|0\rangle_{\text{data}}\,|0\rangle_{\text{control}} = \frac{1}{\sqrt{E}}\Big(\sum_{j=0}^{2^d-1} \beta_j\,|j\rangle \otimes |x\rangle\Big). \tag{6.16}$$

The second step generates a superposition of the same linear operation with different parameters entangled with the control register. This is possible by means of the Step 2 of the MAQA that leads to the following quantum state:

$$|\Phi_d\rangle = \frac{1}{\sqrt{E}}\sum_{j=0}^{2^d-1} \beta_j\,|j\rangle\,G(\Theta_j)\,|x\rangle = \frac{1}{\sqrt{E}}\sum_{j=0}^{2^d-1} \beta_j\,|j\rangle\,|g(x;\Theta_j)\rangle\,. \tag{6.17}$$

Notice that the parametrised function $g(x;\Theta_b)$ corresponds to the transformation $L(x;\theta_j)$ introduced in the previous section.

The third step applies the Σ gate to the *data* register, thus propagating the activation function in all the basis states of the superposition:

$$|\Phi_\Sigma\rangle = (\mathbb{1}^{\otimes d} \otimes \Sigma)\,|\Phi_d\rangle \to \frac{1}{\sqrt{E}}\left(\sum_{j=0}^{2^d-1} \beta_j\,|j\rangle\,|\sigma[L(x;\theta_j)]\rangle\right), \tag{6.18}$$

where $\mathbb{1}^{\otimes d}$ is the identity matrix. In this way, the result of the algorithm above that corresponds to the output of the SLP with 2^d hidden neurons and can be accessed by measuring the *data* register.

The fourth step consists of measuring the data register:

$$\begin{aligned}\langle M\rangle &= \left\langle \Phi_\Sigma | \mathbb{1}^{\otimes d} \otimes M \middle| \Phi_\Sigma \right\rangle \\ &= \sum_{j=1}^{2^d} \beta_j^{'} \left\langle f_j^* | M \middle| f_j^* \right\rangle = \sum_{j=1}^{2^d} \beta_j^{'} \langle M_j\rangle \\ &= \sum_{j=1}^{2^d} \beta_j^{'} f_j = f_{\text{SLP}},\end{aligned} \tag{6.19}$$

where $f_j^* = \sigma[L(x;\theta_j)]$, f_j is the squared of f_j^*, and and $\beta_j^{'} = |\beta_j|^2$ with $\sum_j |\beta_j|^2 = 1$. Although we do not measure the control register, the j-th transformation of the input is associated to a specific amplitude β_j of the *control* register.

Regarding the parameters, β and $\{\theta_j\}_{j=1,\ldots,H}$ can be randomly initialised and the same hybrid optimisation process presented in Section 6.2.4 can be exploited.

Thus, the MAQA allows extending the proposed approach of the qSLP to an exponentially large number of neurons in the hidden layer. In fact, the entanglement of linear combinations to the basis states of the control register implies that the number of linear combinations that can be performed is equal to the number of basis states of the quantum system. This, in turn, implies that the number of hidden neurons H scales exponentially with the number of states of the control register, 2^d. This is a consequence of each hidden neuron being represented by a trajectory of the of the MAQA. This exponential scaling enables the construction of quantum Neural Networks with an arbitrary large number of hidden neurons as the amount of available qubits increases. In other terms, we can build qSLP with an incredible descriptive power that may be really capable of being an universal approximator.

6.5 Discussion

As shown in Section 6.4, the use of MAQA allows building a generalised qSLP with an exponentially large number of neurons, increasing the number of steps of the quantum algorithm linearly. The qSLP supports amplitude encoding strategy, which means that we have an exponential advantage in terms of space complexity when encoding data into the *data* quantum register. This implies a polylogarithmic advantage in terms of the number of parameters needed of the quantum algorithm with respect to its classical counterpart (Schuld et al., 2018).

Furthermore, there are two main differences between classical and quantum SLP, which derive from the normalisation constraint introduced when dealing with the amplitudes of quantum systems. In fact, the input data vector x and the weight vector $\beta = \{\beta_j\}_{j=1,\dots,2^d}$ in the proposal of the qSLP are normalised to 1. Apparently, this may be a limitation since classical NNs take raw input and freely learn the weight parameters. However, rescaling the inputs and limiting the magnitudes of the weights are two common strategies adopted in classical NNs to avoid overfitting. In particular, these procedures are known as *batch normalisation* and *weight decay*. Thus, quantum mechanics' normalisation constraint allows implementing automatically ad-hoc procedures developed in the context of classical NNs without any additional computational effort.

From a computational point of view, given N observations, p features, H hidden neurons and L training epochs, a classical neural network fit typically requires $\mathcal{O}(NpML)$ operations. Although modern GPU allows speeding up linear algebra computation and parallel computing allows to perform the training of classical deep networks efficiently, the linear cost in the four parameters N, p, H, l represents a lower bound in terms of time complexity. In particular, if H is too large, that is a necessary condition for an SLP to be a universal approximator, it is impossible to train the algorithm in a reasonable time. If considering the qSLP that makes use of MAQA, the cost in terms of time complexity to train the quantum circuit is $\mathcal{O}(N \times n \times d \times L)$ where N is the number of training points, n is equal to $log(p)$ and $d = log(H)$. L is still the number of epochs. Importantly, the cost of state preparation and measurement need to be taken into account. While the cost of measurement is restricted to a low number of qubits (just one in case of binary classification or regression), its cost is supposed to be a multiplication constant that can be ignored when considering the overall time complexity. However, the use of qSLP for large datasets needs an efficient routine for state preparation since the state preparation routine proposed by Mottonen et al. (2004) scales linearly in N and p, Nonetheless, once the optimal set of parameters of the qSLP are obtained for a specific task, the full quantum algorithm can be employed in other quantum algorithms to reproduce the function for which it was trained.

Finally, when comparing MAQA architecture with the first proposal of qSLP, the advantages are significant, mostly considering NISQ devices. First, it does not use an additional *temp* register to generate multiple linear combinations in superposition. Consequently, the use of CSWAP is avoided, which introduces a linear cost in terms of gate complexity, as shown in Smolin and DiVincenzo (1996). Also, MAQA provides a general architecture to design a qSLP with an arbitrary number of neurons in the hidden layer, thus extending its use to solve more complex problems.

6.6 Conclusion

In this chapter, we proposed implementing MAQA to generate the quantum version of the Single Layer Perceptron. The key idea is to use a single state preparation routine and apply different linear combinations in superposition, each entangled with a control register. This allows propagating a generic activation function's routine to all the basis states with only one execution. As a result, a model trained through our algorithm is potentially able to approximate any desired function as long as enough hidden neurons and a non-linear activation function are available.

Furthermore, we provided a practical implementation of our variational algorithm that reproduces a quantum SLP for classification with two hidden neurons and an identity function as activation.

In addition, we tested our algorithm on synthetic data and demonstrated that the model works well in case of linearly separable observations, with a test accuracy of 95%. However, the performance deteriorates when facing the intrinsic noise due to quantum computations and current technology limits. On the other hand, experiments showed how the model's performance deteriorates as the distributions of the two classes overlap so to contaminate each other, thus testifying the necessity of introducing non-linearity into the model. For this reason, the main challenge to tackle in the near future is the design of a routine that reproduces a non-linear activation function.

Another natural follow-up of this work is the practical implementation of a generalisation of the quantum SLP to the case of $H > 2$ hidden neurons. This would be beneficial for more hands-on experimentation, including, for instance, the discussion of a regression task.

Notably, Goto et al. (2020) showed that quantum feature maps alongside functions aggregation is able to achieve universal approximation. Thus, a possible future work consists of studying the qSLP on top of the quantum feature map, enabling the qSLP as a universal functions approximator, without implementing a non-linear quantum activation function.

In conclusion, we are still far from proving that the field of Deep Learning can benefit from Quantum Computation in practice. However, thanks to the flexibility of variational algorithms, the hybrid quantum-classical approach may be the ideal setting to make universal approximation possible in quantum computers.

Chapter 7

Beyond Unitarity - Quantum Splines

The previous chapters showed that quantum computing is hugely appealing for many machine learning algorithms, hence providing a new efficient way to solve many real-world applications. However, the operations on quantum states are required to be linear and unitary under the laws of quantum mechanics. This limitation potentially deprives Quantum Machine Learning techniques to describe complex non-linear relations accurately.

In this chapter, we demonstrate how to adopt the quantum implementation of spline functions for approximating several widespread activation functions commonly employed in Neural Networks. In particular, we propose a fault-tolerant quantum algorithm that employs the Harrow Hassidim Lloyd (HHL) algorithm for matrix inversion as a subroutine to compute the spline coefficients in a quantum computer. Then, a given input $|x\rangle$ interacts with the quantum state encoding these coefficients to store the spline estimates in the amplitude of a quantum register.

In practice, two different strategies are followed. The *hybrid* approach computes quantum estimates of the spline coefficients via the HHL quantum algorithm. Then a classical device is used to evaluate the activation functions. The $full$ quantum approach, instead, takes care of the evaluation process end-to-end, with an additional circuit that reads the HHL estimates and evaluates the function.

The chapter is organised as follows. Section 7.1 reviews the most popular statistical learning methods that use matrix inversion to solve a fitting problem. In Section 7.2 the HHL algorithm is discussed. Thus, in Section 7.3 the proposal of the quantum version of the spline is provided. In particular, it will be shown that it is possible to approximate non-linear activation functions using a full quantum routine. To this end, a specific formulation of the splines is employed, the B-spline. Finally, Section 7.4 treats an in-depth discussion of the HHL algorithm efficiency with respect to classical alternatives.

7.1 Matrix inversion in Pattern Recognition

The matrix inversion is usually employed in many statistical learning methods whose solutions can be computed in a closed form. For instance, when considering the standard linear regression model, the inversion of the design matrix is the key to find the parameters of the model to estimate the target variable of interest. In particular, given an input vector $X = (X_1, X_2, \dots, X_p)$ and a target variable Y, the linear regression model has the form:

$$Y = f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j. \tag{7.1}$$

Thus, the linear regression assumes that the regression function $f(X)$ is linear. Notice that, the variables X are not necessarily the observed features, but they can be generated with augmentation procedures. Typically, given a set of N training points $(x_1, y_1) \dots (x_N, y_N)$, the goal is to estimate the vector of parameters $\beta = (\beta_0, \dots, \beta_p)$. The most popular estimation method is the minimisation of the residual sum of squares:

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 = \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 . \tag{7.2}$$

In matrix form, denote $\boldsymbol{X}$ the $N \times (p+1)$ matrix with each row an input vector (with a 1 in the first position), and let $\boldsymbol{y}$ by the N-vector of outputs in the training set. Then it is possible to rewrite the residual sum of squares as

$$RSS(\beta) = (\boldsymbol{y} - \boldsymbol{X}\beta)^T (\boldsymbol{y} - \boldsymbol{X}\beta), \tag{7.3}$$

which is a quadratic form in the $p+1$ parameters. Differentiating with respect to β and assuming that $\boldsymbol{X}$ has full column rank and hence $\boldsymbol{X}^T\boldsymbol{X}$ is positive definite, the estimate of β that minimise the function $RSS(\beta)$ is the following:

$$\hat{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{7.4}$$

Thus, in order to fit a linear model following the minimisation of the $RSS(\beta)$ criteria, it is necessary to invert the design matrix $\boldsymbol{X}^T\boldsymbol{X}$. More specifically, the estimates of β can be computed by solving the linear system expressed in Equation (7.4).

7.1.1 Regularisation and Ridge Regression

Thanks to its simplicity and interpretability, linear regression is widely used in many real-worlds application. However, it often fails in generalising when applied to out-of-sample examples because of the limited *generalisation capability*, which is a key aspect in prediction problems. Indeed, while data fitting is often approached as an optimisation problem in practice, the focus in machine learning is to design statistical estimators able to fit well future examples. Furthermore, the estimator $\hat{\beta}$ in Equation (7.4) is well-defined only if $\boldsymbol{X}^T\boldsymbol{X}$ exists. This implies that, when $\boldsymbol{X}$ is high-dimensional, β cannot be estimated, because it cannot be uniquely determined. This question is typically addressed with so-called *regularisation techniques* which essentially limit the expressive power of the learned estimator in order to avoid overfitting.

A variety of regularisation strategies have been proposed in the literature, each adopting a different perspective on the problem. From a computational perspective, regularised-based methods leverage optimisation techniques to find a solution for the learning problem and typically consist of a sequence of standard linear algebra operations such as matrix multiplication and inversion.

Ridge Regression (Casella, 1985) is an example of such methods. It was originally formulated with two goals: increase the mean squared error loss and improve numerical stability of the coefficients estimates.

In practice, the idea of Ridge Regression (RR) is to shrink the regression coefficients by imposing a penalty on their size. If $\boldsymbol{X}$ is high-dimensional, its columns are likely supercollinears, which means that the subspace spanned by collinear variables may not be full rank. The supercollinearity of $\boldsymbol{X}$ implies that, when solving a regression problem using *least squares*, the rank of the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is smaller than p (number of features) and then it is singular (non-invertible).

To obtain an estimate of regression parameter β when $\boldsymbol{X}$ is supercollinear Hoerl and Kennard (1970) proposed an ad-hoc fix to resolve the almost singularity of $\boldsymbol{X}^T\boldsymbol{X}$, simply replacing it by $\boldsymbol{X}^T\boldsymbol{X} + \lambda \mathbf{1}_{\mathbb{pp}}$ with $\lambda \in [0, \infty)$. The scalar λ is a tuning parameter and it is called *penalty parameter.*

Furthermore, it is possible to show that the ridge penalty in linear regression shrinks the singular values of the matrix to invert, improving the stability of the linear system (van Wieringen, 2015). The ridge coefficients minimise a penalised residual sum of squares:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\text{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}. \tag{7.5}$$

Writing the criterion in Equation (7.5) in matrix form:

$$RSS(\beta) = (\boldsymbol{y} - \boldsymbol{X}\beta)^T(\boldsymbol{y} - \boldsymbol{X}\beta) + \lambda\beta^T\beta, \tag{7.6}$$

where $\lambda \geq 0$. Thus, the ridge regression coefficients can be easily computed as:

$$\hat{\beta}^{ridge} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda'\mathbf{1})^{-1}\boldsymbol{X}^T\boldsymbol{y}, \tag{7.7}$$

where $\mathbf{1}$ is the $p \times p$ identity matrix. The difference between the standard (7.4) and the ridge estimates (7.5) is that the design matrix is modified with $\lambda\mathbf{1}$ which makes it non-singular. From a computational point of view, the regularisation implies a lower condition number of the linear system, which corresponds to improving its numerical stability. In fact, the condition number of a square matrix A is the ratio of its largest ($\lambda_{max}(A)$) and smallest ($\lambda_{min}(A)$) eigenvalue; in case of supercollinearity, the smallest eigenvalue is zero and the condition number is undefined and so is A^{-1}. If considering the condition number of the regularised matrix, this is always lower than the condition number of the original matrix A:

$$\kappa(A) = \frac{|\lambda_{max}(A)|}{|\lambda_{min}(A)|} \geq \frac{|\lambda_{max}(A) + \lambda'|}{|\lambda_{min}(A) + \lambda'|}, \tag{7.8}$$

where $\lambda \in [0, \infty)$.

Thus, regularisation prevents overfitting and guarantees better numerical stability (the more λ increases, the more $\kappa(A)$ decreases). Furthermore, we will see that a lower condition number reduces the computational complexity of many algorithms for matrix inversion. This makes regularisation techniques an ideal setting to employ quantum algorithms to solve typical pattern recognition problems.

7.1.2 Spline Functions

Spline functions are smoothing methods suitable for modelling the relationships between variables, typically adopted either as a visual aid in data exploration or for estimation purposes (Hastie et al., 2001). The underpinning idea is to use linear models in which the input features are augmented with the so-called *basis expansions*. These consist of transformations of the original variables and serve to introduce non-linearity. Technically, splines are constructed by dividing the sample data into sub-intervals delimited by breakpoints, also referred to as knots. A fixed degree polynomial is then fitted in each of the segments, thus resulting in a piecewise polynomial regression. Formally, in the case of a 1-dimensional input vector $\boldsymbol{x}$, we can express its relationship with a target variable $\boldsymbol{y}$ in terms of an order-M spline with knots $\{\xi_k\}_{k=1,\dots,K}$:

$$\boldsymbol{y}_{n_{\text{obs}}\times 1} = \boldsymbol{N}_{n_{\text{obs}}\times(M+K)}\boldsymbol{\theta}_{(M+K)\times 1} + \boldsymbol{\epsilon}_{n_{\text{obs}}\times 1}, \tag{7.9}$$

where $\boldsymbol{\theta}$ is the vector of coefficients attached to the basis expansions, n_{obs} is the sample size, $\boldsymbol{\epsilon}$ is a random error term and the design matrix $\boldsymbol{N}$ contains $M+K$ basis functions defined as follows:

$$h_j(x) = x^{j-1}, \qquad j = 1, \cdots, M \tag{7.10}$$

$$h_{M+k} = (x-\xi_k)_+^{M-1}, \qquad k = 1, \cdots, K. \tag{7.11}$$

Notice that the formulation above includes M basis functions that determine the order-M polynomial to be fitted in each segment. The additional K basis introduce continuity constraints on the spline and its derivatives up to order $M-2$.

The goal is then to find the optimal set of parameters $\boldsymbol{\theta}$ that minimises the residual sum of squares (RSS), with a *ridge regularisation* of the curvature acting as a roughness penalty:

$$\text{Score}(\boldsymbol{\theta}, \eta) = (\boldsymbol{y} - \boldsymbol{N}\boldsymbol{\theta})^T(\boldsymbol{y} - \boldsymbol{N}\boldsymbol{\theta}) + \eta\boldsymbol{\theta}^T\boldsymbol{\Omega}_{(M+K)\times(M+K)}\boldsymbol{\theta}, \tag{7.12}$$

where $\boldsymbol{\Omega}$ is a diagonal matrix containing the partial second derivatives. The solution to (7.12) can easily seen to be:

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{N}^T\boldsymbol{N} + \eta\boldsymbol{\Omega})^{-1}\boldsymbol{N}^T\boldsymbol{y}. \tag{7.13}$$

From Equation (7.13) we can see that the estimates of the spline coefficients reduces to a ridge regression optimisation where the observed variable are augmented using polynomials. Also in this case, we benefit from the advantages given by the reduction of the condition number due to regularisation.

7.2 Quantum Algorithms for Linear Systems

The basic idea of the linear algebra approach in QML is to use a quantum system for linear algebra calculus, where the design matrix is represented by the Hamiltonian of the system via dynamic encoding. As shown in the previous sections, in classical ML the matrices are usually constructed from the training set, this means that the dimension of the problem grows with the number of features and the number of the training points.

HHL (Harrow et al., 2009) is a quantum algorithm that approximately prepares a quantum superposition of the form $|x\rangle$, where x is the solution to a linear system $Ax = b$, assuming the ability to prepare efficiently the state $|b\rangle$ and to apply the unitary transformation e^{-iAt}. It scales with a time complexity that grows roughly like $\mathcal{O}(s^2\kappa^2 log(N)/\epsilon)$, where n is the system size, κ is the system's condition number, s is its sparsity, and ϵ is the desired error (Wiebe et al., 2012).

Although the exponential advantage over the size of the matrix, the constraints on other parameters (s, k) may be very stringent in real-world applications. Therefore, the best way to look at HHL is as a template for other quantum algorithms where it is possible to find specific cases of interest in which preparing $|b\rangle$, applying e^{-iAt}, and measuring $|x\rangle$ are tasks that can be accomplished efficiently.

For instance, ridge regression (RR) is based on a different formulation of the standard linear system of equations where the design matrix to invert is affected by an ill-conditioned problem. The solution for RR can be computed by regularised least squares, that leads to a lower condition number of the design matrix when compared to the standard one used in a classical linear model.

From a quantum perspective, when considering Hamiltonian simulation, spline functions are more suitable to study for a quantum computer, since it corresponds to a low-rank linear system, whose Hamiltonian simulation can be implemented efficiently.

Thus, spline functions are a fitting method that allow to overcome the caveats of the HHL and perform QML efficiently.

7.2.1 Overview on HHL

Given a $N \times N$ Hermitian matrix A and a unit vector $\vec{b}$, the goal of the HHL algorithm is to find the solution vector $\vec{x}$ satisfying the following equation:

$$A\,|x\rangle = |b\rangle, \tag{7.14}$$

where the two vectors $\vec{b}$ and $\vec{x}$ are mapped into the amplitudes of the respective quantum registers $|b\rangle$ and $|x\rangle$. Since A is Hermitian, it can be decomposed using its spectral decomposition as follows:

$$A = \sum_{j=0}^{N-1} \lambda_j |u_j\rangle \langle u_j| , \quad \lambda_j \in \mathbb{R}, \tag{7.15}$$

where $|u_j\rangle$ is the j^{th} eigenvector of A with respective eigenvalue λ_j. Assuming A invertible, A^{-1} is uniquely determined by A, then the solution of Equation (7.14) can be expressed as $x = A^{-1}b$ where the matrix A^{-1} can be written as:

$$A^{-1} = \sum_{j=0}^{N-1} \lambda_j^{-1} |u_j\rangle \langle u_j| . \tag{7.16}$$

Furthermore, the vector $|b\rangle$ can be expressed as linear combination of the eigenbasis of A:

$$|b\rangle = \sum_{j=0}^{N-1} b_j |u_j\rangle, \quad b_j \in \mathbb{C}. \tag{7.17}$$

The HHL provides an exponentially faster quantum protocol with respect to N for solving a linear system as follows:

$$|x\rangle = A^{-1}|b\rangle = \sum_{j=0}^{N-1} \lambda_j^{-1} b_j |u_j\rangle, \tag{7.18}$$

assuming the ability to prepare efficiently the state $|b\rangle$ and apply the unitary transformation e^{-iAt}.

Although the exponential advantage in the system size, a number of caveats limit its applicability to practical problems (Aaronson, 2015). First, it requires the matrix A to be sparse, because of the polynomial dependency on the level of sparsity s. Second, data must be loaded in quantum superposition efficiently, to preserve the computational advantage when inverting A. Third, the output is encoded in a quantum state, where the entries of x are in superposition and the approximation of all the coefficients may be prohibitive for large linear systems. Fourth, the condition number and the sparsity must scale at most sub-linearly with N. Also, to extract information from a quantum state, one must perform a measurement. Learning all N amplitudes of an N-dimensional state requires a number of measurements at least proportional to N. Thus, if our goal is to completely reconstruct a solution x, the quantum algorithm cannot have a significant advantage over classical methods. In addition, as in classical methods for solving linear equations, the performance depends crucially on the condition number κ, a measure of how close A is to singular.

An interesting problem that satisfies these requirements is discussed by Clader et al. (2013) where a generalised version of the HHL is employed to compute the electromagnetic scattering cross section of an arbitrary target exponentially faster than the best classical algorithm.

7.2.2 Quantum algorithm for HHL

Here, we discuss the quantum implementation of the HHL, summarising the description presented by Lee et al. (2019). Technically, the algorithm assumes b represented as $|b\rangle = \sum_{j=1}^{l} b_j |u_j\rangle$ where $|u_j\rangle$ is the j^{th} eigenstate of A and $b_j \in \mathbb{C}$, such that $\sum_{j=1}^{l} |b_j|^2 = 1$. The solution of the linear system in Equation (7.18) can be expressed by explicitly write the normalisation constraint as:

$$|x\rangle = \frac{A^{-1}|b\rangle}{||A^{-1}|b\rangle||}, \tag{7.19}$$

where A^{-1} is the inverse matrix of A.

In practice, the algorithm uses three quantum registers: the first one is used to store a binary representation of the eigenvalues of A[1], a second register contains the vector solution $|x\rangle$ and includes $n = log(N)$ qubits, where N is the size of the linear system. Another register is auxiliary to the computation. Starting from these three registers, the algorithm involves four main steps: quantum phase estimation (QPE), eigenvalue inversion, uncomputation (or inverting QPE), and Rejections Sampling.

Quantum Phase Estimation

The QPE part assumes the efficient implementation of the unitary U defined as:

$$U = e^{iAt} := \sum_{j=0}^{N-1} e^{i\lambda_j t}|u_j\rangle\langle u_j|, \tag{7.20}$$

and transforms the initial quantum state as follows:

$$|0\rangle \otimes |b\rangle_n \otimes |0\rangle \xrightarrow{QPE} \sum_{j=0}^{N-1} b_j |\lambda_j\rangle |u_j\rangle_n \otimes |0\rangle\,, \tag{7.21}$$

[1]the number of qubits depends on the precision for the binary representation of the eigenvalues

where $|\lambda_j\rangle$ is the binary representation of the j^{th} eigenvalue λ_j.

Eigenvalues Inversion

A conditioned rotation on $|\lambda_j\rangle$ is applied using the auxiliary qubit:

$$\sum_{j=0}^{N-1} b_j \, |\lambda_j\rangle \, |u_j\rangle_n \otimes |0\rangle \xrightarrow{\text{Eigen Inversion}} \sum_{j=0}^{N-1} b_j |\lambda_j\rangle |u_j\rangle_n \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right) \tag{7.22}$$

where C is a normalisation constant.

Uncomputation

At this point, the states of all the registers are entangled and it is not possible to trace out a part of the system which contains the solution of interest. Thus, the uncomputation step performs the inverse QPE to isolate the quantum state that contains the solution of the linear system:

$$\sum_{j=0}^{N-1} b_j \, |\lambda_j\rangle \, |u_j\rangle_n \otimes |0\rangle \xrightarrow{\text{Uncomputation}} \sum_{j=0}^{N-1} b_j |0\rangle |u_j\rangle_n \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right), \tag{7.23}$$

where all the register are reseted in $|0\rangle$. Notice that, because of this step, the circuit is almost twice as deep as it was.

Rejection Sampling

At this point the ancilla register is measured and, if state $|0\rangle$ is readout, the result is discarded. Instead, if the ancilla state is $|1\rangle$, the result is proportional to the inverse of λ and the state describing the qubit system successfully represents the solution of the linear equation as follows:

$$\frac{1}{||A^{-1} \, |b\rangle \, ||} \sum_{j}^{l} \frac{b_j}{\lambda_j} \, |u_j\rangle \,, \tag{7.24}$$

where $||A^{-1} \, |b\rangle \, || = \sum_j^l \frac{|b_j|^2}{\lambda_j^2}$.

7.3 Quantum Activation Functions

As stated in Section 6, one of the major issues for building a quantum Neural Network is the implementation of a non-linear activation function on a quantum system. Indeed, the operations on quantum states are required to be linear and unitary under the laws of quantum mechanics, as discussed in Nielsen and Chuang (2011).

The most popular attempt to achieve non-linearity via a quantum algorithm is described in Cao et al. (2017), where the authors used the repeat-until-success technique to design a quantum circuit for reproducing a non-linear activation function. The biggest limitation is that this function requires input in the range $\left[0, \frac{\pi}{2}\right]$, which is a severe constraint for real-world problems. A step forward was made by Hu (2018), where domain restrictions are removed and the activation gate parameters were learned during training.

However, besides several attempts have been made, up to date, there is no way to approximate non-linear functions using a full quantum routine.

7.3.1 Quantum Spline

Here we investigate non-linear approximation by means of a quantum spline (QSpline). In particular, we want to demonstrate its applicability as an evaluation routine by fitting the spline to some widespread activation functions adopted in Neural Networks. To this end, we consider *relu, elu, tanh* and *sigmoid* and we use a linear spline with no derivability constraints. Also, no roughness penalisation is applied since we would like to mimic the target function as closely as possible. Regarding the choice of knots, 20 equally spaced breakpoints were selected over the interval $(-1, 1)$.

7.3.2 Implementation

While the formulation in terms of truncated basis functions described in Section 7.1.2 is conceptually simple, its numerical and computational properties are not very attractive. For this reason, in practice we adopt a *B-splines* parametrisation (De Boor, 1978). This allows generating a block design matrix where the *sparsity* is constant and depends on the degree of the polynomial fitted in each local interval. Given a sequence of knots $\xi_1, \xi_2, \cdots, \xi_K$, we fit a line in each interval $[\xi_k, \xi_{k+1}]_{k=1,\cdots,K-1}$ without derivability constraints. In matrix form:

$$\tilde{\boldsymbol{y}} = \boldsymbol{S}\boldsymbol{\beta} \rightarrow \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \\ \cdots \\ \tilde{y}_K \end{pmatrix} = \begin{pmatrix} S_1 & 0 & \cdots & 0 \\ 0 & S_2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & S_K \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \cdots \\ \beta_K \end{pmatrix}, \tag{7.25}$$

where $\tilde{y_k}$ contains the activation function evaluations in ξ_k and ξ_{k+1}, β_ks are the spline coefficients and $\boldsymbol{S}_{(2K)\times(2K)}$ is a block diagonal matrix with each block S_k that represents the basis expansions in the k-th interval. Solving the linear system in Equation (7.25) requires using HHL to invert the matrix $\boldsymbol{S}$. Nonetheless, we can exploit the fact that the inverse of a block diagonal matrix is still block diagonal, with the correspondent inverse matrices in each block. This implies we can solve K 2×2 quantum linear systems $S_k \left|\beta_k\right\rangle = \left|\tilde{y_k}\right\rangle$ instead of a single one for the entire function. This little trick permits to overcome the practical limitations of the available quantum simulators, thus enabling the calculation of the spline coefficients through quantum simulations.

In particular, the computation of the full QSpline is performed in three steps. First, the HHL computes the spline coefficients for the k-th interval:

$$S_k \left|\beta_k\right\rangle = \left|\tilde{y_k}\right\rangle \xrightarrow{HHL} \left|\beta_k\right\rangle \simeq S_k^{-1} \left|\tilde{y_k}\right\rangle . \tag{7.26}$$

Second, $\left|\beta_k\right\rangle$ interacts with the quantum state encoding the input in the k-th interval via quantum interference. The scalar product between $\left|\beta_k\right\rangle$ and $\left|x_{i,k}\right\rangle$ is computed using the swap-test (Buhrman et al., 2001):

$$\left|\beta_k\right\rangle \left|x_{i,k}\right\rangle \left|0\right\rangle \xrightarrow{swap-test} \left|e_1\right\rangle \left|e_2\right\rangle \left|f_{i,k}\right\rangle . \tag{7.27}$$

At this point, the amplitudes of the quantum state $\left|f_{i,k}\right\rangle$ embed the estimate of the activation function evaluated in $x_{i,k}$.

Third, $\left|f_{i,k}\right\rangle$ is measured to get the probability of state $\left|0\right\rangle$. This depends on the dot product between β_k and $x_{i,k}$ as follows:

$$\left|f_{i,k}\right\rangle = \sqrt{p_0}\left|0\right\rangle + \sqrt{p_1}\left|1\right\rangle , \tag{7.28}$$

where:

$$p_0 = \frac{1}{2} + \frac{\left|\left\langle\beta_k|x_{i,k}\right\rangle\right|^2}{2} = \frac{1}{2} + \frac{|f_{i,k}|^2}{2}. \tag{7.29}$$

Finally, the activation function estimate in correspondence of $x_{i,k}$ is retrieved by back-transforming Equation (7.29) to get $f_{i,k}$.

Notice that, the estimates are intrinsically bounded in the interval $[0,1]$ since they are encoded as the amplitude of a quantum state. For this reason, the target activation functions are first scaled (i.e. $f_{i,k} \to f^*_{i,k} \subseteq [0,1]$). The QSpline is then trained on $f^*_{i,k}$, and the results are finally back-transformed to get the original $f_{i,k}$.

7.3.3 Results

The results of hybrid QSpline are illustrated in Figure 7.1.

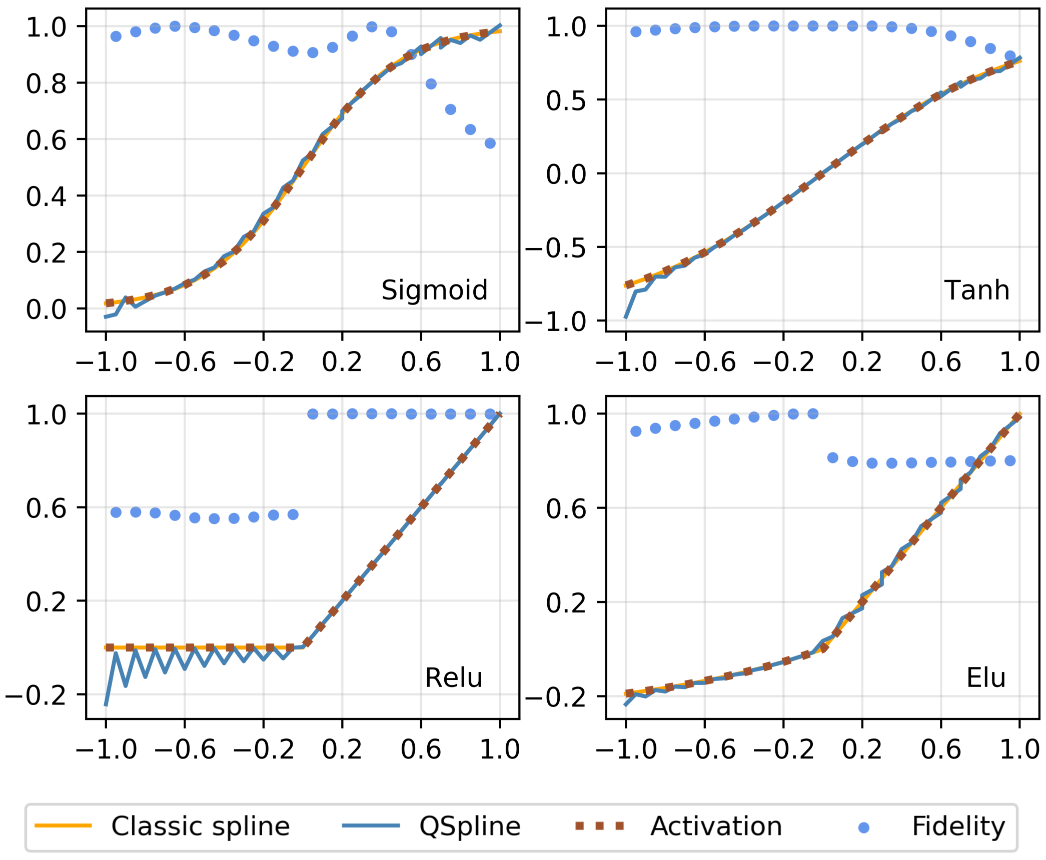


Fig. 7.1 Hybrid QSpline with $k = 20$.

The quantum splines perform very well in reproducing the activation curves. The agreement is almost perfect for the sigmoid and the hyperbolic tangent, with slight deviations at the boundaries of the estimation interval. The situation is similar for Relu and Elu, although apparently some systematic error is introduced in the negative part of the x domain. A possible explanation can be given by looking at the blue dots representing the fidelity of the quantum algorithm, that measures the discrepancy between the HHL output and the real solution of the system. In fact, larger deviations appear in areas where the fidelity is low, thus suggesting the HHL implementation may behave poorly in our setting.

A further attempt was then made exploiting a *full* quantum circuit. The results are reported in Figure 7.2. In this case, the goodness of fit is definitely worse, with an almost systematic overestimate of the target curves. This may be due to the adoption

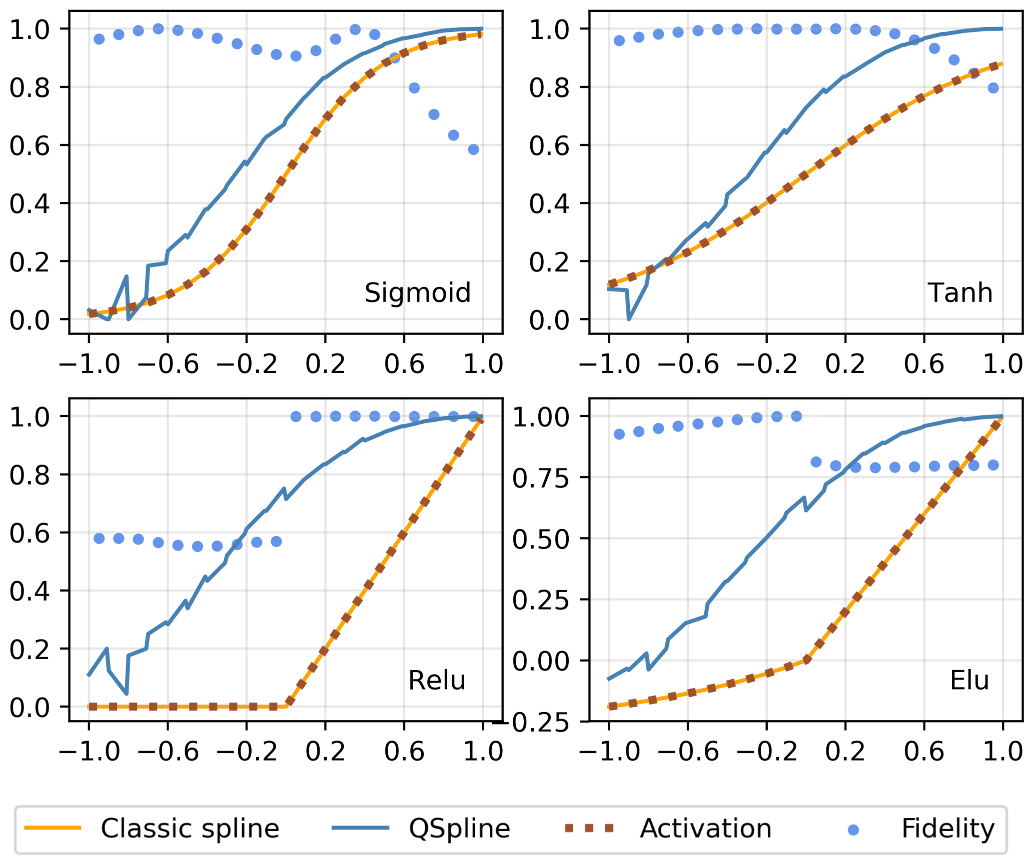


Fig. 7.2 Full QSpline with $k = 20$.

of a second circuit for evaluation on top of the estimation one. Therefore, having two measurement phases introduce more uncertainty which is then propagated and produce a larger error. Nonetheless, notice that the QSpline is likewise capable of reproducing the non-linear behaviour of the curves, which is anyway the most relevant characteristic under investigation. For this reason, we interpret the results as promising, although they necessitate for more tuning. A quantitative assessment of the goodness of fit of both strategies is reported in Table 7.1.

7.4 Computational Efficiency

Here we discuss in details the computational cost of the quantum estimation phase, also drawing a theoretical comparison with the classical baselines. In general, matrix inversion can be accomplished in polynomial time on classical devices (Bürgisser et al., 2013; Coppersmith and Winograd, 1987; Strassen, 1969). However, when several

Activation function	**RSS (classic)**	**RSS (hybrid)**	**RSS (quantum)**	**Average Fidelity**
Sigmoid	3.3×10^{-5}	0.01	0.75	0.90
Tanh	1.2×10^{-5}	0.06	1.12	0.96
Relu	7.6×10^{-31}	0.14	8.16	0.78
Elu	5.9×10^{-7}	0.12	7.06	0.88

Table 7.1 Approximation metrics. The table shows the Residual Sum of Squares (RSS) of both quantum and classical splines with respect to the true activation functions. Quantum approaches are indicated as *hybrid* and *quantum*. The average fidelity of the HHL is also reported.

Gauss elimination	**Strassen**	**Coppersmith**	**Conjugate Gradient**	**HHL**
$\mathcal{O}(n^3)$	$\mathcal{O}(n^{2.8})$	$\mathcal{O}(n^{2.37})$	$\mathcal{O}(sn\sqrt{\kappa}/\log(\epsilon))$	$\mathcal{O}(s^2\kappa^2\log(n)/\epsilon)$

Table 7.2 Comparison of algorithms computational costs.

favourable assumptions hold it is possible to reduce computational costs. In particular, the *Conjugate Gradient* algorithm (Ciliberto et al., 2018; Shewchuk et al., 1994) allows solving a linear system with a complexity equal to $\mathcal{O}(s\sqrt{\kappa}n/\log(\epsilon))$, where n is the system size, κ is the system condition number, s the matrix sparsity (i.e. the maximum number of non-zero matrix elements of $\boldsymbol{A}$ in any given row or column), and ϵ is the desired error tolerance.

The reference quantum technique is the HHL algorithm. HHL is a method for approximately preparing a quantum superposition of the form $|\boldsymbol{x}\rangle$, where $\boldsymbol{x}$ is the solution to a linear system $\mathbf{Ax} = \mathbf{b}$, $\boldsymbol{A}$ is a hermitian design matrix and $\boldsymbol{b}$ is encoded in amplitudes of $|\boldsymbol{b}\rangle$. From a computational point of view, this requires an amount of time that grows roughly like $\mathcal{O}(s^2\kappa^2\log(n)/\epsilon)$ (cfr. Table 7.2 for a comparison between HHL and classical counterparts).

The algorithm scales logarithmically with respect to the size of the matrix, which means it has an exponential advantage when compared to classical alternatives. However, its complexity is polynomial in s and κ, which means we have to introduce constraints on the condition number and the sparsity not to destroy the computational advantage of the HHL. This makes the previous comparison unfair since we cannot make assumptions about the design matrix in general.

In our case, we can observe that an order-M penalised spline has very desirable properties. First of all, the ridge penalisation described in Equation (7.13) guarantees a lower condition number compared to the unconstrained design matrix. Specifically, Casella (1985) showed that the higher the penalisation, the lower the condition number is. In fact, the condition number of a matrix, $\kappa(\boldsymbol{A})$, is equal to $\frac{|\lambda_{max}(\boldsymbol{A})+\eta|}{|\lambda_{min}(\boldsymbol{A})+\eta|}$, where λ_{min} and λ_{max} are the smallest and the biggest eigenvalues respectively, and η is the amount of penalisation. Clearly, the more η increases, the more it dominates the fraction, eventually tending to 1 for large penalisations. Furthermore, regarding splines as piecewise polynomials defined on contiguous segments allows building a sparse design matrix $\boldsymbol{A}$, whose blocks describe the approximation in the corresponding intervals. This implies that the sparsity amounts to the number of basis functions, $M + K$. In light of the two properties above, splines are an ideal setting for an efficient application of the HHL algorithm.

Figure 7.3 illustrates a theoretical comparison of HHL computational costs with respect to the classical counterparts. The green curves illustrate the computational

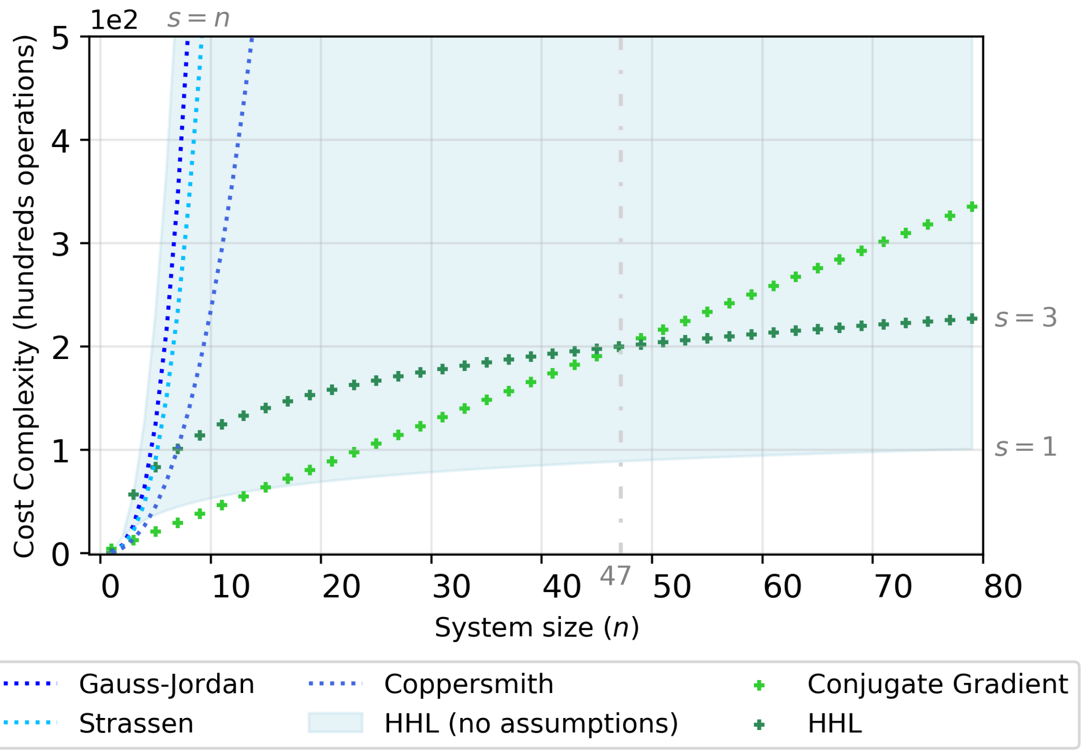


Fig. 7.3 Cost complexity as a function of the size, n, of the $\boldsymbol{A}$ matrix. The green curves represent HHL and Conjugate Gradient for fixed s, κ, while the blue ones are referred to matrix inversion with no assumptions. The light blue shaded area illustrates the performance of HHL as the sparsity varies (κ fixed).

costs (in hundreds of operations) of HHL and Conjugate Gradient for fixed sparsity and condition number. In particular, their values were chosen to reflect the case of natural splines with no intercept coefficient ($s = 3$) and an approximately well-conditioned data matrix ($\kappa = 2$). The HHL outperforms the Conjugate Gradient as the number of features becomes larger than 47, that is quite frequent in *Big Data* and *Artificial Intelligence* applications, e.g. bioinformatics, natural language processing and computer vision. A comparison of HHL with matrix inversion algorithms is also conducted when no assumptions are made. The light blue shaded are depicts the performance of HHL as s varies (κ fixed to 2), while the curves using the blue palette describe classical alternatives. The advantage of HHL is evident as soon as some sparsity is introduced. However, we have also to take into account κ and the higher costs of quantum computation to draw more solid conclusions. Thus the efficiency boost due to the quantum technologies may foster the use of HHL for novel, classically unfeasible applications. For instance, the computational burden of spline functions could be mitigated in this way, paving the way for future studies aimed at performing splines per se (e.g. multidimensional splines), and not just as a mere tool for evaluating non-linear functions.

7.5 Conclusion

In this chapter, we demonstrated the adoption of splines for approximating popular activation functions on a quantum device. The preliminary results were promising, although some tuning of the HHL implementation and the circuit for the full quantum approach is required. The quantum spline was able to reproduce the non-linearity of the curves, thus candidating this approach as a building block in the development of Quantum Neural Networks.

Also, we compared the computational complexity of the HHL algorithm against the most adopted classical counterparts, showing why splines may be an ideal setting for leveraging its advantages.

Future studies will be dedicated to improving the full quantum approach, with the possibility of developing a novel and more stable implementation of the HHL routine and approach the same problem using NISQ algorithms.

Chapter 8

Conclusions and Outlook

Quantum Machine Learning has the potential to improve classical machine learning and overcome some of the main limitations imposed by the classical computing paradigm. However, the practical advantages of using quantum resources to solve machine learning tasks are still to be demonstrated. The limitations are two-fold. From a technical point of view, the main constraint lies in efficiently performing state preparation and applying an arbitrarily long sequence of quantum gates. This limits the use of many QML algorithms presented in the literature. On the other hand, from a methodological perspective, the ground provided by quantum mechanics is extremely appealing since a low number of qubits allows accessing an exponentially large Hilbert space. Though, the potential benefit of machine learning from near-term quantum devices has not yet been proven.

This dissertation tries to make a further step towards the study of how machine learning can benefit from the quantum computation. The proposed quantum framework, the *Multiple Aggregator Quantum Algorithm* (MAQA), is potentially capable of reproducing some of the most important classical machine learning algorithms. In particular, MAQA propagates an input state to multiple quantum trajectories in superposition, and each trajectory describes a specific function $g(x;\cdot)$ that represents the component function of the final QML model. The entanglement between the two quantum registers involved (data and control) allows averaging those transformations efficiently, and the result can be accessed by measuring only a subset of qubits. MAQA is potentially able to improve, in terms of time complexity, all those models that compute multiple and diverse functions to produce a final strong model. The speed-up comes from the efficient aggregation of multiple functions, which classically scales at least linearly in the number of these functions. Instead, the proposed quantum architecture allows aggregating H functions in only $log(H)$ steps, providing an exponential speed-up, under the assumption that the cost in terms of circuit complexity is unitary for each step. Furthermore, quantum interference allows propagating the use of a specific unitary (gate F) to all the quantum trajectories in superposition. Hence, the application of F impacts additively to the overall time complexity and the same operation would require a multiplicative cost in classical computation. In addition, the proposed approach can be adopted both as full-coherent protocol (quantum ensemble) and as NISQ algorithm (quantum Single Layer Perceptron).

The quantum ensemble proposal uses MAQA to design a quantum algorithm that exploits quantum superposition, entanglement, and interference to build an ensemble of classification models. The generation of multiple training sets in superposition allows increasing exponentially the ensemble's size with respect to classical methods. Moreover, when considering the overall cost of the algorithm, the use of the single weak classifier impacts additively rather than multiplicatively, as it happens classically. However, due to limits of available quantum technology, some compromises have to be made when implementing the quantum ensemble. In this respect, the main challenge to tackle to make the ensemble effective in the future is the design of a quantum classifier based on interference that guarantees a more efficient data encoding strategy (e.g. amplitude encoding), and that can process larger datasets.

Furthermore, MAQA has been employed to build the quantum Single Layer Perceptron (qSLP). The idea of qSLP is to design a quantum algorithm to generate different linear combinations in superposition, where the parameters of the quantum algorithm can be learned using hybrid quantum-classical computation. The use of MAQA allows producing an exponentially large number of neurons, increasing the algorithm's number of steps linearly. This allows building a model with an incredible descriptive power capable of being a universal approximator if equipped with a proper activation function. In this respect, the main challenge to tackle in the near future is designing a routine that reproduces a non-linear activation function. Another natural follow-up of the qSLP is the practical implementation of the generalisation with more than two neurons.

Another contribution of this dissertation is the quantum spline (QSpline), which provides a full quantum algorithm that can approximate non-linear functions. Because of the use of the HHL, the QSpline is a full-coherent protocol, thus unfeasible to run on NISQ devices. However, recent developments in the context of variational algorithms have shown the possibility to execute quantum matrix inversion using near-term technology. This opens to the possibility of adopting the QSpline as a quantum activation function.

To conclude, we are still in an early stage for QML, and its contribution in the context of machine learning and artificial intelligence is still to be understood. However, many research findings, including this thesis, suggest that the potential of quantum computing is enormous, and machine learning will likely benefit from it in the future.

References

Aaronson, S. (2015). Read the fine print. *Nature Physics*, 11(4):291.

Aïmeur, E., Brassard, G., and Gambs, S. (2006). Machine learning in a quantum world. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 431–442. Springer.

Aïmeur, E., Brassard, G., and Gambs, S. (2013). Quantum speed-up for unsupervised learning. *Machine Learning*, 90(2):261–287.

Ambainis, A. (2017). Understanding quantum algorithms via query complexity. *arXiv preprint arXiv:1712.06349*, 244.

Anderson, E. (1936). The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457–509.

Arora, S. and Barak, B. (2009). *Computational complexity: a modern approach*. Cambridge University Press.

Arunachalam, S., Gheorghiu, V., Jochym-O'Connor, T., Mosca, M., and Srinivasan, P. V. (2015). On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 17(12):123010.

Arunachalam, S. and Maity, R. (2020). Quantum boosting. *arXiv preprint arXiv:2002.05056*.

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G., Buell, D. A., et al. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510.

Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., Sleator, T., Smolin, J. A., and Weinfurter, H. (1995). Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467.

Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945.

Behrman, E. C., Steck, J. E., and Skinner, S. R. (1999). A spatial quantum neural computer. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, volume 2, pages 874–877. IEEE.

Benedetti, M., Lloyd, E., Sack, S., and Fiorentini, M. (2019). Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001.

Benhelm, J., Kirchmair, G., Roos, C. F., and Blatt, R. (2008). Towards fault-tolerant quantum computing with trapped ions. *Nature Physics*, 4(6):463–466.

Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Blank, C., McKiernan, K., and Killoran, N. (2018). Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*.

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. (2017). Quantum machine learning. *Nature*, 549(7671):195.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Brassard, G., Hoyer, P., Mosca, M., and Tapp, A. (2002). Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Brown, K. R., Wilson, A. C., Colombe, Y., Ospelkaus, C., Meier, A. M., Knill, E., Leibfried, D., and Wineland, D. J. (2011). Single-qubit-gate error below $\mathbf{10^{-4}}$ in a trapped ion. *Phys. Rev. A*, 84:030303.

Buhrman, H., Cleve, R., Watrous, J., and de Wolf, R. (2001). Quantum fingerprinting. *Phys. Rev. Lett.*, 87:167902.

Bürgisser, P., Clausen, M., and Shokrollahi, M. A. (2013). *Algebraic complexity theory*, volume 315. Springer Science & Business Media.

Cao, Y., Guerreschi, G. G., and Aspuru-Guzik, A. (2017). Quantum neuron: an elementary building block for machine learning on quantum computers. *arXiv preprint arXiv:1711.11240*.

Casella, G. (1985). Condition numbers and minimax ridge regression estimators. *Journal of the American Statistical Association*, 80(391):753–758.

Ciliberto, C., Herbster, M., Ialongo, A. D., Pontil, M., Rocchetto, A., Severini, S., and Wossnig, L. (2018). Quantum machine learning: a classical perspective. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2209):20170551.

Cirac, J. I. and Zoller, P. (1995). Quantum computations with cold trapped ions. *Physical review letters*, 74(20):4091.

Clader, B. D., Jacobs, B. C., and Sprouse, C. R. (2013). Preconditioned quantum linear system algorithm. *Physical review letters*, 110(25):250504.

Coppersmith, D. and Winograd, S. (1987). Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Cross, A. W., Bishop, L. S., Sheldon, S., Nation, P. D., and Gambetta, J. M. (2019). Validating quantum computers using randomized model circuits. *Physical Review A*, 100(3):032328.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

d'Alessandro, D. (2007). *Introduction to quantum control and dynamics*. Chapman and Hall/CRC.

D'Ariano, G. M., Paris, M. G., and Sacchi, M. F. (2003). Quantum tomography. *Advances in Imaging and Electron Physics*, 128:206–309.

Dawson, C. M. and Nielsen, M. A. (2006). The solovay-kitaev algorithm. *Quantum Info. Comput.*, 6(1):81–95.

Dayan, P. and Abbott, L. F. (2005). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press.

De Boor, C. (1978). *A practical guide to splines*, volume 27. Springer-Verlag New York.

De Brabanter, J., De Moor, B., Suykens, J. A., Van Gestel, T., and Vandewalle, J. P. (2002). *Least squares support vector machines*. World scientific.

Devitt, S. J., Munro, W. J., and Nemoto, K. (2013). Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001.

DiVincenzo, D. P. (1997). Topics in quantum computers. In *Mesoscopic electron transport*, pages 657–677. Springer.

Domingos, P. (2000). Bayesian averaging of classifiers and the overfitting problem. In *ICML*, volume 2000, pages 223–230.

Domingos, P. M. (1997). Why does bagging work? a bayesian account and its implications. In *KDD*, pages 155–158. Citeseer.

Drineas, P. and Mahoney, M. W. (2005). On the nyström method for approximating a gram matrix for improved kernel-based learning. *journal of machine learning research*, 6(Dec):2153–2175.

Durr, C. and Hoyer, P. (1996). A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*.

Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Einstein, A., Podolsky, B., and Rosen, N. (1935). Can quantum-mechanical description of physical reality be considered complete? *Physical review*, 47(10):777.

Faber, J. and Giraldi, G. A. (2002). Quantum models of artificial neural networks. *Electronically available: http://arquivosweb. lncc. br/pdfs/QNN-Review. pdf*, 5(7.2):5–7.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9(Aug):1871–1874.

Farhi, E., Goldstone, J., and Gutmann, S. (2014). A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*.

Farhi, E. and Neven, H. (2018). Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*.

Freedman, D. A. (2009). *Statistical models: theory and practice*. Cambridge University press.

Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612.

Friedman, J. H. and Tukey, J. W. (1974). A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on computers*, 100(9):881–890.

Gillespie, T. A. (1980). Spectral theory of linear operators. *Proceedings of the Edinburgh Mathematical Society*, 23(3).

Giovannetti, V., Lloyd, S., and Maccone, L. (2008a). Architectures for a quantum random access memory. *Phys. Rev. A*, 78:052310.

Giovannetti, V., Lloyd, S., and Maccone, L. (2008b). Quantum random access memory. *Phys. Rev. Lett.*, 100:160501.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Goto, T., Tran, Q. H., and Nakajima, K. (2020). Universal approximation property of quantum feature map. *arXiv preprint arXiv:2009.00298*.

Gottesman, D., Kitaev, A., and Preskill, J. (2001). Encoding a qubit in an oscillator. *Physical Review A*, 64(1):012310.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. *arXiv preprint quant-ph/9605043*.

Gupta, S. and Zia, R. (2001). Quantum neural networks. *Journal of Computer and System Sciences*, 63(3):355–383.

Haag, R. and Kastler, D. (1964). An algebraic approach to quantum field theory. *Journal of Mathematical Physics*, 5(7):848–861.

Hallgren, S. (2007). Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. *Journal of the ACM (JACM)*, 54(1):4.

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160.

Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001.

Harrow, A. W., Hassidim, A., and Lloyd, S. (2009). Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502.

Harrow, A. W. and Montanaro, A. (2017). Quantum computational supremacy. *Nature*, 549(7671):203–209.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Hornik, K., Stinchcombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560.

Hornik, K., Stinchcombe, M., White, H., et al. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hu, W. (2018). Towards a real quantum neuron. *Natural Science*, 10(3):99–109.

Huang, H.-L., Wu, D., Fan, D., and Zhu, X. (2020). Superconducting quantum computing: a review. *Science China Information Sciences*, 63(8):1–32.

Impagliazzo, R. and Wigderson, A. (1997). P= BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of Computing*, pages 220–229.

Jacobs, R. A. (1995). Methods for combining experts' probability assessments. *Neural computation*, 7(5):867–888.

Jaffe, A. M. (2006). The millennium grand challenge in mathematics. *Notices of the AMS*, 53(6).

Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J. M., and Gambetta, J. M. (2017). Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242.

Khot, S. (2016). Hardness of approximation. In *ICALP*, pages 3–1. Citeseer.

Kwok, S. W. and Carter, C. (1990). Multiple decision trees. In *Machine Intelligence and Pattern Recognition*, volume 9, pages 327–335. Elsevier.

Ladd, T. D., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C., and O'Brien, J. L. (2010). Quantum computers. *Nature*, 464(7285):45–53.

Lanyon, B. P., Whitfield, J. D., Gillett, G. G., Goggin, M. E., Almeida, M. P., Kassal, I., Biamonte, J. D., Mohseni, M., Powell, B. J., Barbieri, M., et al. (2010). Towards quantum chemistry on a quantum computer. *Nature chemistry*, 2(2):106.

Lee, Y., Joo, J., and Lee, S. (2019). Hybrid quantum linear equation algorithm and its experimental test on IBM quantum experience. *Scientific Reports*, 9(1):4778.

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867.

Li, B. and Han, L. (2013). Distance weighted cosine similarity measure for text classification. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 611–618. Springer.

Macaluso, A., Clissa, L., Lodi, S., and Sartori, C. (2020a). Quantum ensemble for classification. *arXiv preprint arXiv:2007.01028*.

Macaluso, A., Clissa, L., Lodi, S., and Sartori, C. (2020b). Quantum splines for non-linear approximations. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 249–252.

Macaluso, A., Clissa, L., Lodi, S., and Sartori, C. (2020c). A variational algorithm for quantum neural networks. In *International Conference on Computational Science*, pages 591–604. Springer.

Macaluso, A., Lodi, S., and Sartori, C. (2020d). Quantum algorithm for ensemble learning. In *Proceedings of the 21st Italian Conference on Theoretical Computer Science*.

MacKay, D. J. and Mac Kay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

MATH, N. (2007). Singular value decomposition and least squares solutions (with c. reinsch). *Milestones in Matrix Computation: The Selected Works of Gene H. Golub with Commentaries*, 14:160.

McClean, J. R., Boixo, S., Smelyanskiy, V. N., Babbush, R., and Neven, H. (2018). Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):1–6.

Mengoni, R. and Di Pierro, A. (2019). Kernel methods in quantum machine learning. *Quantum Machine Intelligence*, pages 1–7.

Mitchell, T. M. et al. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877.

Moll, N., Barkoutsos, P., Bishop, L. S., Chow, J. M., Cross, A., Egger, D. J., Filipp, S., Fuhrer, A., Gambetta, J. M., Ganzhorn, M., et al. (2018). Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503.

Mottonen, M., Vartiainen, J. J., Bergholm, V., and Salomaa, M. M. (2004). Transformation of quantum states using uniformly controlled rotations. *arXiv preprint quant-ph/0407010.*

Myerson, A. H., Szwer, D. J., Webster, S. C., Allcock, D. T. C., Curtis, M. J., Imreh, G., Sherman, J. A., Stacey, D. N., Steane, A. M., and Lucas, D. M. (2008). High-fidelity readout of trapped-ion qubits. *Phys. Rev. Lett.*, 100:200502.

Nakahara, M. and Ohmi, T. (2008). *Quantum computing: from linear algebra to physical realizations.* CRC press.

Nielsen, M. A. and Chuang, I. L. (2011). *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, USA, 10th edition.

Oza, N. C. and Tumer, K. (2008). Classifier ensembles: Select real-world applications. *Information Fusion*, 9(1):4–20.

Park, D. K., Sinayskiy, I., Fingerhuth, M., Petruccione, F., and Rhee, J.-K. K. (2019). Quantum forking for fast weighted power summation. *arXiv preprint arXiv:1902.07959.*

Pearson, K. (1901). On lines and planes of closest fit to system of points in space, philos. *Edinburgh and Dublin Philosophical Magazine and Journal of Science,(2)*, pages 33–41.

Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P. J., Aspuru-Guzik, A., and O'brien, J. L. (2014). A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:4213.

Preskill, J. (2012). Quantum computing and the entanglement frontier. *arXiv preprint arXiv:1203.5813.*

Rebentrost, P., Mohseni, M., and Lloyd, S. (2014). Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503.

Ristè, D., da Silva, M. P., Ryan, C. A., Cross, A. W., Córcoles, A. D., Smolin, J. A., Gambetta, J. M., Chow, J. M., and Johnson, B. R. (2017). Demonstration of quantum advantage in machine learning. *npj Quantum Information*, 3(1):16.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747.*

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Sánchez A, V. D. (2003). Advanced support vector machines and kernel methods. *Neurocomputing*, 55(1-2):5–20.

Schapire, R. E. (2003). The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer.

Schnabel, T., Labutov, I., Mimno, D., and Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307.

Schuld, M., Bocharov, A., Svore, K., and Wiebe, N. (2018). Circuit-centric quantum classifiers. *arXiv preprint arXiv:1804.00633*.

Schuld, M. and Killoran, N. (2019). Quantum machine learning in feature hilbert spaces. *Physical review letters*, 122(4):040504.

Schuld, M. and Petruccione, F. (2018a). Quantum ensembles of quantum classifiers. *Scientific reports*, 8(1):2772.

Schuld, M. and Petruccione, F. (2018b). *Supervised Learning with Quantum Computers*. Springer Publishing Company, Incorporated, 1st edition.

Schuld, M., Sinayskiy, I., and Petruccione, F. (2014). The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586.

Schuld, M., Sinayskiy, I., and Petruccione, F. (2015). Simulating a perceptron on a quantum computer. *Physics Letters A*, 379(7):660–663.

Schützhold, R. (2003). Pattern recognition on a quantum computer. *Physical Review A*, 67(6):062311.

Shewchuk, J. R. et al. (1994). An introduction to the conjugate gradient method without the agonizing pain.

Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332.

Smolin, J. A. and DiVincenzo, D. P. (1996). Five two-bit quantum gates are sufficient to implement the quantum fredkin gate. *Phys. Rev. A*, 53:2855–2856.

Soman, K., Loganathan, R., and Ajay, V. (2009). *Machine learning with SVM and other kernel methods*. PHI Learning Pvt. Ltd.

Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356.

Tacchino, F., Macchiavello, C., Gerace, D., and Bajoni, D. (2019). An artificial neuron implemented on an actual quantum processor, zak1998quantum. *npj Quantum Information*, 5(1):26.

Toffoli, T. (1980). Reversible computing. In *International Colloquium on Automata, Languages, and Programming*, pages 632–644. Springer.

Tóth, G., Lent, C. S., Tougaw, P. D., Brazhnik, Y., Weng, W., Porod, W., Liu, R.-W., and Huang, Y.-F. (1996). Quantum cellular neural networks. *Superlattices and Microstructures*, 20(4):473–478.

Trugenberger, C. A. (2002). Quantum pattern recognition. *Quantum Information Processing*, 1(6):471–493.

Tumer, K. and Ghosh, J. (1996a). Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 29(2):341–348.

Tumer, K. and Ghosh, J. (1996b). Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3-4):385–404.

Van Dam, W., Hallgren, S., and Ip, L. (2006). Quantum algorithms for some hidden shift problems. *SIAM Journal on Computing*, 36(3):763–778.

van Wieringen, W. N. (2015). Lecture notes on ridge regression. *arXiv preprint arXiv:1509.09169.*

Wecker, D., Hastings, M. B., and Troyer, M. (2015). Progress towards practical quantum variational algorithms. *Phys. Rev. A*, 92:042303.

Wiebe, N., Braun, D., and Lloyd, S. (2012). Quantum algorithm for data fitting. *Physical review letters*, 109(5):050505.

Wiebe, N., Kapoor, A., and Svore, K. M. (2018). Quantum nearest-neighbor algorithms for machine learning. *Quantum Information and Computation*, 15.

Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390.

Xu, L., Krzyzak, A., and Suen, C. Y. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE transactions on systems, man, and cybernetics*, 22(3):418–435.

Zoufal, C., Lucchi, A., and Woerner, S. (2019). Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 5(1):1–9.

Appendix A

A Brief Introduction to Quantum Mechanics

A.1 Proof of the no-cloning theorem

Suppose there exists a unitary matrix U such that

$$U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle, \tag{A.1}$$

for any arbitrary state vector $|\psi\rangle$. Thus, we can rewrite the generated 4-dimensional state vector as

$$\begin{aligned} |\psi\rangle|0\rangle \xrightarrow{U} |\psi\rangle|\psi\rangle &= (a|0\rangle + b|1\rangle) \otimes (a|0\rangle + b|1\rangle) \\ &= a^2|00\rangle + ab|01\rangle + ba|10\rangle + b^2|11\rangle. \end{aligned} \tag{A.2}$$

Then, if we apply the same operation U to copy the expansion of $|\psi\rangle$

$$(a|0\rangle + b|1\rangle) \otimes |0\rangle \xrightarrow{U} a|00\rangle + b|10\rangle, \tag{A.3}$$

we end up with a different quantum state with no cross terms, which is different from the quantum state in Equation (A.2). This is a contradiction and therefore such unitary U cannot exist.

Appendix B

Details on Quantum Ensemble

B.1 Swap-test to compute the cosine distance

The swap-test is a procedure to measure the cosine similarity distance between two quantum states. Cosine similarity is a metric used to measure how similar two objects are irrespective of their size, capturing only the orientation (the angle) and not the magnitude. The smaller the angle between two objects, the higher the similarity. Given two p-dimensional vectors $\vec{a}$ and $\vec{b}$, the cosine similarity between them is calculated as follows:

$$cos\left(\vec{a},\vec{b}\right) = \frac{\vec{a}\cdot\vec{b}}{||\vec{a}||\cdot||\vec{b}||} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}},$$

for $i = 1,\dots,p$. As a fundamental component, cosine similarity has been applied for solving different pattern recognition problems, such as text classification, information retrieval, word embedding (Li and Han, 2013; Schnabel et al., 2015). The quantum implementation of the cosine distance is performed through the swap-test where two 2-dimensional vectors ($\vec{a}$, $\vec{b}$) are encoded into the amplitudes of two different qubits ($|a\rangle$, $|b\rangle$). Then using an additional ancillary qubit, the output of the swap-test is function of the distance between the two vectors. The swap-test is performed into three steps.

Step 1: State Preparation

The first step consists of encoding the two classical (normalised) vectors $\vec{a}$ and $\vec{b}$ are encoded into the amplitudes of two different qubits:

$$\left(\mathbb{1}\otimes S_{\vec{a}}\otimes S_{\vec{b}}\right)|0\rangle|0\rangle|0\rangle = |0\rangle|a\rangle|b\rangle\,, \tag{B.1}$$

where $S_{\vec{x}}$ encodes a real vector $\vec{x}$ into the amplitudes of a qubit.

Step 2: Execution of the swap-test

In the second step, the Hadamard gate is applied to the ancilla qubit, then the controlled-SWAP gate, and a second Hadamard gate:

$$\Big(H \otimes \mathbb{1} \otimes \mathbb{1}\Big)\Big(\text{C-SWAP}\Big)\Big(H \otimes \mathbb{1} \otimes \mathbb{1}\Big) |0\rangle |a\rangle |b\rangle = \frac{1}{2}|0\rangle(|a,b\rangle + |b,a\rangle) + \frac{1}{2}|1\rangle(|a,b\rangle - |b,a\rangle), \tag{B.2}$$

where H is the Hadamard gate, C-SWAP is the controlled-swap operation which uses the ancilla qubit as control qubit and swap $|a\rangle$ and $|b\rangle$ if the state of the ancilla is $|1\rangle$.

Step 3: Measurement

Measuring the first qubits produces the state 0 with probability:

$$\Pr\Big(|0\rangle\Big) = \frac{1 + |cos(\vec{a}, \vec{b})|^2}{2}, \tag{B.3}$$

where $cos(\vec{a}, \vec{b})$ is the classical cosine similarity between $\vec{a}$ and $\vec{b}$. If $\vec{a} = \vec{b}$, then the probability of measuring $|0\rangle$ is equal to 1. The quantum circuit to perform the swap-test is depicted in Figure B.1.

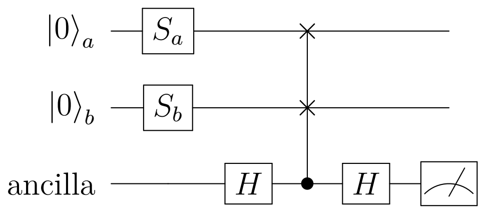


Fig. B.1 Quantum circuit for Swap-test.

B.2 Quantum Ensemble as Simple Averaging

Here we describe the quantum circuit to obtain four *independent* quantum trajectories in superposition considering a quantum ensemble of cosine classifiers (Section 5.3).

(Step 1) State Preparation

For a 2-qubit *control* register ($d = 2$), we can build an ensemble of $B = 2^2$ classifiers. The *data* encodes a single observation using a single qubit. In particular, given a dataset made up of N observations $\{x_i, y_i\}_{i=1,\dots,N}$, where $x_i = (x_{i,1}, x_{i,2})$ is a 2-dimensional vector and $y_i \in \{0, 1\}$ is the binary target variable, the *data* register encodes N training points $2 \times N$ qubits:

$$\text{data register: } \underbrace{\left(\bigotimes_{i=1}^{4} |x_i\rangle\right)}_{features} \otimes \underbrace{\left(\bigotimes_{i=1}^{4} |y_i\rangle\right)}_{labels} = |features\rangle\, |labels\rangle\,, \tag{B.4}$$

where the values $x_{i,1}$ and $x_{i,2}$ are encoded into the amplitudes of a single qubit:

$$|x_i\rangle = x_{i,1}\,|0\rangle + x_{i,2}\,|1\rangle\,, \tag{B.5}$$

and the two classes of the target variable are represented by the two basis states of a single qubit. Thus, if $|y_i\rangle = |0\rangle$ the i-th observation belongs to the class 0. Otherwise, if $|y_i\rangle = |1\rangle$ the i-th observation belongs to the class 1.

Qubit encoding strategy allows to store a training set of 4 observations using an 8-qubit *data* register. In formulas, state preparation step leads to:

$$\begin{aligned}
|\Phi_0\rangle &= \left(H^{\otimes 2} \otimes S_{(x,y)}\right) |0\rangle \otimes |0\rangle \otimes |0\rangle \\
&= |c_1\rangle \otimes |c_2\rangle \otimes |x\rangle\, |y\rangle \\
&= \frac{1}{\sqrt{2}}\Big(|0\rangle + |1\rangle \Big) \otimes \frac{1}{\sqrt{2}}\Big(|0\rangle + |1\rangle \Big) \otimes |x_0, x_1, x_2, x_3\rangle\, |y_0, y_1, y_2, y_3\rangle\,,
\end{aligned}$$

where S_x is the routine which encodes in the amplitudes of a qubit a real vector x and H is the Hadamard transformation.

(Step 2) Sampling in Superposition

The second step regards the generation of 2^d different transformations of the training set in superposition, each entangled with a state of the control register. To this end, d steps are necessary, where each step consists in the entanglement of the i-th control qubit with two transformations of $|x, y\rangle$ based on two random unitaries, $U_{(i,1)}$ and $U_{(i,2)}$, for $i = 1, 2$.

The sampling in superposition step leads to the following quantum state:

$$\begin{aligned}|\Phi_1\rangle = \frac{1}{2}\Big[& |00\rangle\, U_{(2,1)}U_{(1,1)}\, |x_0, x_1, x_2, x_3\rangle\, |y_0, y_1, y_2, y_3\rangle \\ & + |01\rangle\, U_{(2,1)}U_{(1,2)}\, |x_0, x_1, x_2, x_3\rangle\, |y_0, y_1, y_2, y_3\rangle \\ & + |10\rangle\, U_{(2,2)}U_{(1,1)}\, |x_0, x_1, x_2, x_3\rangle\, |y_0, y_1, y_2, y_3\rangle \\ & + |11\rangle\, U_{(2,2)}U_{(1,2)}\, |x_0, x_1, x_2, x_3\rangle\, |y_0, y_1, y_2, y_3\rangle \Big].\end{aligned}$$

In order to obtain independent quantum trajectories, we provide the following definition for $U_{(i,j)}$:

$$U_{(1,1)} = \text{SWAP}(x_0, x_2) \times \text{SWAP}(y_0, y_2); \tag{B.6}$$

$$U_{(1,2)} = \text{SWAP}(x_1, x_3) \times \text{SWAP}(y_1, y_3); \tag{B.7}$$

$$U_{(2,1)} = \mathbb{1}; \tag{B.8}$$

$$U_{(2,2)} = \text{SWAP}(x_2, x_3) \times \text{SWAP}(y_2, y_3); \tag{B.9}$$

where $\mathbb{1}$ is the identity matrix. Thus, we get:

$$\begin{aligned}|\Phi_2\rangle = \frac{1}{2}\Big[& |11\rangle\, |x_0, x_3, x_1, x_2\rangle\, |y_0, y_3, y_1, y_2\rangle \\ & + |10\rangle\, |x_2, x_1, x_3, x_0\rangle\, |y_2, y_1, y_3, y_0\rangle \\ & + |01\rangle\, |x_0, x_3, x_2, x_1\rangle\, |y_0, y_3, y_2, y_1\rangle \\ & + |00\rangle\, |x_2, x_1, x_0, x_3\rangle\, |y_2, y_1, y_0, y_3\rangle \Big].\end{aligned}$$

We can see that swap operations allows to entangle different observations (in terms of the indices of the qubits) to different state of the *control* register. In particular, if considering the last qubit of the *features* and *labels* (sub-)registers, the above choices for $U_{(i,j)}$ guarantee that each quantum state of the control register is entangled with a different training observation. Using a compact representation:

$$\begin{aligned}|\Phi_{2'}\rangle &= \frac{1}{2}\Big[|11\rangle\, |x_2\rangle\, |y_2\rangle + |10\rangle\, |x_0\rangle\, |y_0\rangle + |01\rangle\, |x_1\rangle\, |y_1\rangle + |00\rangle\, |x_3\rangle\, |y_3\rangle \Big] \\ &= \frac{1}{\sqrt{4}} \sum_{i=0}^{3} |i\rangle\, |x_i, y_i\rangle\, .\end{aligned} \tag{B.10}$$

Notice that, in this case the i-th basis state does not correspond to the integer representation of the binary state. Importantly, the only difference to implement

(Step 3) Learning via interference

The *test* register is initialised to encode the test set, $\tilde{x}$, considering also an additional qubit to store the final prediction:

$$(S_{\tilde{x}} \otimes \mathbb{1}) |0\rangle |0\rangle = |x^{(test)}\rangle |0\rangle . \tag{B.11}$$

Then, the *data* and *test* registers interact via interference using the quantum version of the cosine classifier (gate F) to compute the estimates of the target variable:

$$\begin{aligned} |\Phi_f\rangle =& \left(\mathbb{1}^{\otimes 2} \otimes F\right) |\Phi_d\rangle \\ =& \frac{1}{\sqrt{2^d}} \sum_{b=1}^{2^d} |b\rangle |x_b, y_b\rangle \left|x^{(test)}\right\rangle \left|\hat{f}_b\right\rangle . \end{aligned}$$

Since the 4 points of the training set are in superposition, the application of the quantum cosine classifier allows computing 4 different predictions for the test point, $\{\hat{f}_b\}_{b=1,\dots 4}$, executing the classifier only once.

(Step 4) Measurement

Due to the entanglement between the predictions for $\tilde{x}$ and the *control* register the expectation measurement allows retrieving the average of all the predictions, which correspond to the ensemble prediction that uses bagging strategy aggregation:

$$\begin{aligned} \langle M \rangle =& \frac{1}{2^d} \sum_{b=1}^{2^d} \left\langle \hat{f}_b | M | \hat{f}_b \right\rangle = \frac{1}{2^d} \sum_{b=1}^{2^d} \langle M_b \rangle \\ =& \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b = \hat{f}_{bag}(\tilde{x}|x, y). \end{aligned}$$

The implementations of the quantum ensemble to perform simple averaging is depicted in Figure B.2.

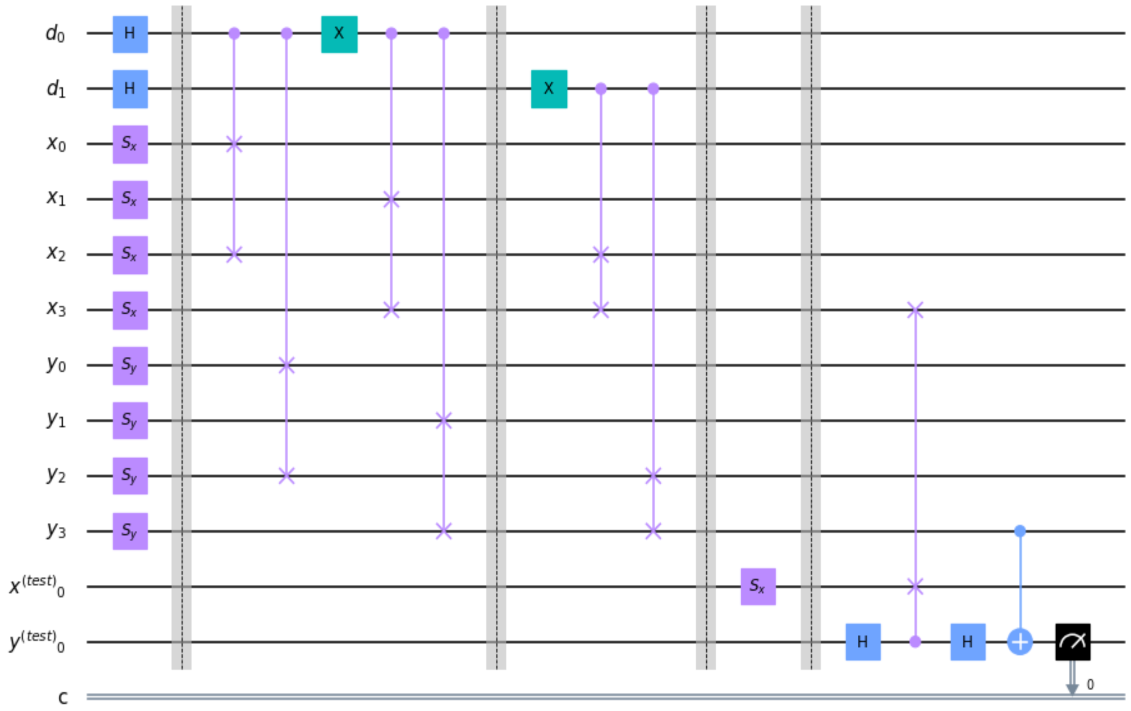


Fig. B.2 Qiskit implementation of the quantum ensemble for independent quantum trajectories.

B.3 Quantum Cosine Classifier

Classically, cosine classifier is defined as follows:

$$Pr\left(y^{(test)} = y_b\right) = \frac{1}{2} + \frac{\left[d\left(x_b, x^{(test)}\right)\right]^2}{2} \tag{B.12}$$

where (x_b, y_b) is a random training example, $x^{(test)}$ the test point and $d(\cdot,\cdot)$ the cosine distance between x_b and $x^{(test)}$. Since the probability of belonging to a class depends on the squared cosine distance between the two vectors, the maximum dissimilarity occurs when training and test observations are orthogonal. In this case, the cosine classifier assigns a uniform probability distribution in the two classes for $y^{(test)}$. This means that the cosine classifier performs well only if the test point belongs to the same class of the training point.

The quantum circuit that implements the cosine classifier (Figure 5.3) encodes data into three different registers: the training vector $x^{(i)}$, the training label $y^{(i)}$ and the test point $x^{(test)}$. One last qubit is used to store the prediction.

The algorithm is made of the following three steps.

Step 1: State Preparation

The state preparation routine can be performed independently for each qubit:

$$|\Phi_1\rangle = \left(S_{x_b} \otimes S_{x^{(test)}} \otimes S_{y_b} \otimes \mathbb{1}\right) |0\rangle\, |0\rangle\, |0\rangle\, |0\rangle = |x_b\rangle\, |x^{(test)}\rangle\, |y_b\rangle\, |0\rangle\,, \tag{B.13}$$

where S_x is the routine which encodes in the amplitudes of a qubit a 2-dimensional, normalised real vector x.

Step 2: Execution of the swap test

In the second step, the swap-test transforms the amplitudes of the qubit $y^{(test)}$ as a function of the squared cosine distance:

$$\begin{aligned}|\Phi_2\rangle =& (\mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} \otimes H)\,(\text{cswap} \otimes \mathbb{1} \otimes C)\,(\mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} \otimes H)\,|x_b\rangle\, |x^{(test)}\rangle\, |y_b\rangle\, |0\rangle \\ =& \frac{1}{2}\left[\left(|x_b\rangle\, |x^{(test)}\rangle + |x^{(test)}\rangle\, |x_b\rangle\right) |y_b\rangle\, |0\rangle + \left(|x_b\rangle\, |x^{(test)}\rangle - |x^{(test)}\rangle\, |x_b\rangle\right) |y_b\rangle\, |1\rangle\right),\end{aligned} \tag{B.14}$$

where H is the Hadamard gate, CSWAP is the controlled-swap operation which uses the last qubit (position of gate C) as control qubit to swap $|x_b\rangle$ and $\left|x^{(test)}\right\rangle$. After the execution of the swap test the probability to readout the basis state $|0\rangle$, that is the probability for the test observation to be classified in class 0 is:

$$P(y^{(test)} = |0\rangle) = \frac{1}{2} + \frac{|\langle x_b|x^{(test)}\rangle|^2}{2}. \tag{B.15}$$

Step 3: Controlled Pauli-X gate

The third step consists of applying a controlled-Pauli-X gate using as control qubit the label of the training vector. This implies that $y^{(test)}$ is left untouched if x_b belongs to the class 0. Otherwise, the amplitudes of the $y^{(test)}$ qubit are exchanged, and the probability $P(y^{(test)} = 1)$ is higher as the similarity between the two vectors increases.

$$|\Phi_3\rangle = (\mathbb{1} \otimes \mathbb{1} \otimes \text{C-X})\,|\Phi_2\rangle\,. \tag{B.16}$$

At this point the expectation measurement provides on the last qubit provides the prediction of interest.

B.4 Algorithm for Quantum Ensemble

In this section is presented the implementation of the quantum ensemble to produce the results shown in Section 4.1.2.

Algorithm 2: Quantum ensemble of quantum cosine classifiers

```
Result: Predictions of the binary target value for all points in the test set
Input:
- $2n$–qubit data register, $d$–qubit control register, 2-qubit test register
- Pauli-Z (measurement) operator $\langle\sigma_z\rangle$

for each point $\tilde{x}$ in the test set do
    # (Step 1) State Preparation
    Encode $n$ random training points into the $n \times 2$ qubits of the data register:
    $(x_1, y_1), \dots, (x_n, y_n) \xrightarrow{S_{(x,y)}} |x_1, \dots, x_n;\ y_1, \dots, y_n\rangle = |features;\ labels\rangle$
    Initialise the $d$ qubits of control register into a uniform superposition:
    $|0 \dots 0\rangle \xrightarrow{W} \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} |k\rangle$
    Initialise the test register: $|0, 0\rangle \xrightarrow{S_{(\tilde{x},0)}} |\tilde{x}, 0\rangle$

    # (Step 2) Sampling in superposition
    for each qubit in the control register $(i = 1, \dots d)$ do
        Select two pairs of random integers $l, m$ and $l', m'$ between 1 and $n$;
        C-SWAP($control(i), features(l), features(m)$);
        C-SWAP($control(i), labels(l), labels(m)$);
        Apply Pauli-X gate to the current control qubit ;
        C-SWAP$\left(control(i), features\left(l'\right), features\left(m'\right)\right)$;
        C-SWAP$\left(control(i), labels\left(l'\right), labels\left(m'\right)\right)$;
    end

    # (Step 3) Learning via Interference
    Apply the quantum cosine classifier (gate $F$) using as training set a random
      pair of qubits (features, labels) of the data register;

    # (Step 4) Measurement
    Measure the test register using $\langle\sigma_z\rangle$ operator
end
Output: Ensemble predictions for all points in the test set;
```