Alma Mater Studiorum · Università di Bologna

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Artificial intelligence

Quantum machine learning: development and evaluation of the Multiple Aggregator Quantum Algorithm

Relatore:
Chiar.mo Prof. C. Sartori

Presentata da:
Filippo Orazi

Correlatori:
Chiar.mo Prof. S. Lodi,
Dott. A. Macaluso

IV Sessione di laurea
Anno Accademico: 2020/2021

Abstract

Human society has always been shaped by its technology, so much that even ages and parts of our history are often named after the discoveries of that time. The growth of modern society is largely derived from the introduction of classical computers that brought us innovations like repeated tasks automatization and long-distance communication. However, this explosive technological advancement could be subjected to a heavy stop when computers reach physical limitations and the empirical law known as Moore Law comes to an end. Foreshadowing these limits and hoping for an even more powerful technology, forty years ago the branch of quantum computation was born. Quantum computation uses at its advantage the same quantum effects that could stop the progress of traditional computation and aim to deliver hardware and software capable of even greater computational power. In this context, this thesis presents the implementation of a quantum variational machine learning algorithm called quantum single-layer perceptron. We start by briefly explaining the foundation of quantum computing and machine learning, to later dive into the theoretical approach of the multiple aggregator quantum algorithms, and finally deliver a versatile implementation of the quantum counterparts of a single hidden layer perceptron. To conclude we train the model to perform binary classification using standard benchmark datasets, alongside three baseline quantum machine learning models taken from the literature. We then perform tests on both simulated quantum hardware and real devices to compare the performances of the various models.

Introduction

The branch of computer science referred to as quantum computation studies how to exploit properties of quantum mechanics for computational purposes. This approach is profoundly different from the classical computation based on classical bits that can assume two possible states 0 or 1. Quantum computation instead, is built upon the existence of a quantum bits or Qubits that can be found in state 1, 0, and in a combination of the two. A more thorough description is presented in section 1.2. While the studies for practical implementation of quantum hardware are fairly recent, the theory begins in the early '80s with Paul Benioff (1980) that defines a quantum Turing machine based on a reversible Turing machine by Bennett (1973), and with Richard P Feynman (1982) that asserted the need for quantum machinery for the study of quantum phenomena in a conference in 1982.

The attention of the scientific community only rose in 1994 when Peter W. Shor (1997) published "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithm on a Quantum Computer", an article where he describes two algorithms that could solve in (error bounded) polynomial-time problems that belong to the NP complexity class for traditional methods. The consequences of this paper were important, but the limited technology available at that time reduced the field to theoretical speculation.

Today different quantum hardware are available in the private sector, but some com-

panies like IBM (2022) allow researchers to use their quantum circuit. Nations and organizations worldwide invest a lot of funds in this field (Temkin, 2021) and more investments will come in the upcoming years.

In this thesis, we implement and test a quantum artificial intelligence model, comparing it to today's standards. In particular, we show the implementation of a single layer perceptron based on the MAQA theoretical framework (Macaluso, 2021) and we perform a comparison between that and the models proposed for artificial intelligence by the Qiskit library by training everything on real-world datasets. We also focus our attention on the problem of state preparation, discussing and comparing the effect that different approaches have on the models.

The Thesis is divided as follows:

- Chapter 1: Brief introduction to the principles that describe quantum mechanics and introduction to quantum computation and its fundamentals properties.
- Chapter 2: Small introduction to machine learning, quantum machine learning algorithms and the problem of state preparation.
- Chapter 3: Exposition of the multiple aggregators quantum algorithm, implementation of the proposed models and discussion about the benchmark models.
- Chapter 4: Discussion on the experiments, description of the standard datasets, and analysis of the results.
- Chapter 5: Conclusions and outlook.

Contents

List of Figures

List of Tables

Chapter 1

Quantum computation principles

This chapter explains the theory needed to understand quantum computation and its fundamentals: Qubit, Quantum gates, and Quantum circuits. The topics are covered mainly from a mathematical point of view without deepening too much in the physical nature of the elements and quantum physics in general. The chapter is proposed as an introduction to the concept of quantum computation, it explains the basic elements allowing the reader to understand what will be presented in the rest of the thesis.

1.1 Quantum mechanics' principles

We start the introduction with a brief overview of quantum mechanics. Quantum mechanics is the branch of physics that studies behaviors and properties of elements at an atomic and subatomic scale. This theory was born in the early 1900 after the so-called *ultraviolet catastrophe*. Following this event, Max Planck derived an equation that described energy as not being continuous but quantized. In 1907 Albert Einstein demonstrated that not only light is quantized but so are atomic vibration Einstein

(1987) leading to the creation of two equivalent quantum mechanics theories in the following years Schrödinger (1926)Heisenberg (1925).
In this section, we will focus on four postulates that describe quantum mechanics in the formulation provided by Nielsen and Chuang Michael A. Nielsen (2010) and on how these can be interpreted in favor of a new way to compute data.

1.1.1 Vector space postulate

Postulate 1. *Associated to any isolated physical system is a complex vector space with an inner product (that is, a Hilbert space) known as the state space of the system. The system is completely described by its state vector, which is a unit vector in the system's state space.*

For our purposes, we will consider a simple quantum system called qubit that is described by two elements: an orthonormal basis in a two-dimensional complex space and two complex coefficients subjected to the normalization condition (in-depth explanation can be found in section 1.2). This postulate also tells us that every quantum computation with classical data will involve a mapping operation to the Hilbert space.

1.1.2 State evolution postulate

Postulate 2. *The time evolution of the state of a closed quantum system is described by the Schrödinger equation:*

$$i\hbar\frac{d\left|\psi\right\rangle}{dt} = H\left|\psi\right\rangle. \tag{1.1}$$

This postulate specifies how a closed quantum system evolves in continuous time. In our experiments we can use a less refined postulate that can be directly derived from postulate 2.

Postulate 2.1. *The evolution of a closed quantum system is described by a unitary transformation. That is, the state $|\psi\rangle$ of the system at time t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2.*

In this, we refer to a discrete time-step that corresponds to the application of a unitary operation. In quantum computation, a unitary operator U, implemented through the use of quantum gates (section 1.3) allows us to utilize quantum hardware for computation.
A direct consequence of postulate 2.1 is the following theorem:

Theorem 1 (**No cloning theorem**). *Quantum mechanics does not allow unknown quantum state to be copied exactly. In quantum information theory it doesn't exist a unitary operation U that given two states $|\psi\rangle$ and $|s\rangle$ can copy $|\psi\rangle$ in $|s\rangle$*

$$\nexists U \mid U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle \,.$$

This theorem is an evolution of the *No-go theorem* by James-Park Park (1970) and it poses significant limitations to quantum computations since it limits our ability to create multiple copies of the same quantum state efficiently.
Both postulates 2 and 2.1 consider a closed quantum system where quantum states can evolve over time without any interaction with the external environment. However such a perfect system cannot exist in reality, with the only exception being the universe as a whole. For the experiment in this thesis, we will assume an error corrected quantum hardware that can deal with external noise and faulty quantum operation by means of quantum error correcting procedures.

1.1.3 Measurement postulate

Even if we could create and maintain a closed system, the final stage of our computation must result in a measurement of the quantum state. In that case, the system will come into contact with the outside world and might undergo non-unitary transformation. Postulate 3 describes what happens when the measurement action is taken upon a closed quantum system.

Postulate 3. *Quantum measurements are described by a collection* $\{M_m\}$ *of measurement operators. These are operators acting on the state space of the measured system. 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*

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle \tag{1.2}$$

and the state of the system after the measurement is

$$\frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}} \tag{1.3}$$

The measurement operators satisfy the completeness equation:

$$\sum_m M_m^\dagger M_m = I \tag{1.4}$$

The completeness equation expresses the fact that probabilities sum to one:

$$1 = \sum_m p(m) = \sum_m \langle\psi| M_m^\dagger M_m |\psi\rangle . \tag{1.5}$$

The measurement utilized in this thesis belongs to the category of *measurement of a qubit in the computational basis.* This means that the measurement on a single qubit is done through the measurement of two different operators $M_0 = |0\rangle\langle 0|$ and

$M_1 = |1\rangle\langle 1|$. These operators are Hermitian and $M_0^2 = M_0$, $M_1^2 = M_1$ complying with the completeness relation $I = M_0^\dagger M_0 + M_1^\dagger M_1 = M_0 + M_1$. Suppose that we have a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ then the probability of obtaining an outcome 0 after a measurement in M_0 is

$$p(0) = \langle\psi| M_0^\dagger M_0 |\psi\rangle = \langle\psi| M_0 |\psi\rangle = |\alpha|^2 \tag{1.6}$$

1.1.4 Composite system postulate

We now want to consider systems made up of more distinct physical systems, this in practice expands the state space following postulate 4

Postulate 4. *The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n, and system number i is prepared in the state $|\phi_i\rangle$ then the joint state of the total system is $|\phi_1\rangle \otimes |\phi_2\rangle \otimes \ldots \otimes |\phi_n\rangle$.*

This postulate enables the superposition principle of quantum mechanics that states: *if $|x\rangle$ and $|y\rangle$ are two different states of a quantum system then their superposition $\alpha|x\rangle + \beta|y\rangle$ should be consider a quantum system where $|\alpha|^2 + |\beta|^2 = 1$.* The possible combination of multiple quantum systems opens up the possibilities for the use of entanglement and interference. The application of postulate 3 and 4 allows us to derive what is called *normalization condition*: for a quantum register $|\psi\rangle$ made of one or more qubits described by means of the normal basis $S = |i_0 i_1 \ldots i_n\rangle$

$$|\psi\rangle = \sum_{j=1}^{n} a_j |i_j\rangle \tag{1.7}$$

the sum of the squared coefficient of each base vector a_j must be 1

$$\sum_{i=1}^{n} |a_j|^2 = 1 \tag{1.8}$$

1.2 Qubit

Qubits are the foundation of quantum computation acting as the counterpart of bits in classical computation. While bits are elements that have two distinct states 0 and 1, a qubit is a more complex element that can be found in both states 0 and 1 at the same time in what is called a superposition of states.

1.2.1 Mathematical representation

In order to obtain a meaningful mathematical description of a qubit we have to describe it using the Dirac notation. The Dirac notation or Bra-Ket notation is used to denote quantum states as vectors: we can denote a column vector as

$$\vec{v} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = |v\rangle \tag{1.9}$$

In quantum mechanics, $|v\rangle$ is a state vector called *"ket-v"* and can be expressed as the linear combination of the basis of its space. The complex conjugate transposed of $|v\rangle$ is called *"bra-v"* and is represented as:

$$\langle v| = \overline{v^T} = \begin{bmatrix} \overline{a_1} & \overline{a_2} & \dots & \overline{a_n} \end{bmatrix} \tag{1.10}$$

Following this definition, we obtain that $\langle v|$ is the adjoint of $|v\rangle$ and vice-versa

$$\begin{aligned} |v\rangle &= \langle v|^\dagger \\ \langle v| &= |v\rangle^\dagger \end{aligned} \tag{1.11}$$

When considering two vectors in a finite-dimensional space represented by fixed orthonormal basis vectors we can represent the inner product of two vectors v_1 and w

as:

$$\langle v|w\rangle = \begin{bmatrix} \overline{v_1} & \overline{v_2} & \dots & \overline{v_n} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \sum_{i=0}^{n} \overline{v_i} w_i \tag{1.12}$$

The scalar product in an Hilbert space owns the following properties:

- $\langle x|x\rangle \geq 0 \wedge \langle x|x\rangle = 0 \implies x = 0$
- $\langle x|y\rangle = \overline{\langle y|x\rangle}$ in complex spaces $\forall x, y \in V$
- $\langle ax|y\rangle = a\langle x|y\rangle \, \forall x, y \in V \wedge a$ scalar

For each vector v we can define its norm as:

$$\| v \| = \sqrt{\langle v|v\rangle} \tag{1.13}$$

Another fundamental operation in the Hilbert space is the Kronecker product (or tensor product). Given two vectors $|v\rangle$, $|w\rangle$ of size n, m the tensor product is computed as:

$$|v\rangle \otimes |w\rangle = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \otimes \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} = \begin{bmatrix} v_1 w_1 \\ v_1 w_2 \\ \vdots \\ v_1 w_m \\ v_2 w_1 \\ \vdots \\ v_n w_m \end{bmatrix} \tag{1.14}$$

In this context, a Qubit can be seen as a normalized vector in the Hilbert space $\mathbb{C}^2$ were vectors $|0\rangle$ and $|1\rangle$, also represented as $(1,0)^T$ and $(0,1)^T$, form an orthogonal basis called standard computational basis S. A vector can be seen as the linear

combination of the basis S in a n-dimensional space:

$$|\psi\rangle = \sum_{i=0}^{n} a_i \, |S_i\rangle$$

In our case being $\mathbb{C}^2$ a two dimensional space we can represent a single qubit as:

$$|\psi\rangle = \alpha \, |0\rangle + \beta \, |1\rangle$$

$|\psi\rangle$ represents a qubit if and only if the amplitudes α and β comply with the normalization condition: $|\alpha|^2 + |\beta|^2 = 1$ with $\alpha, \beta \in \mathbb{C}$. This is necessary because the squared module of the coefficient represents the probability with which the qubit collapses to the relative basis when measured.

1.2.2 Geometric representation

To better understand qubits, we can give a geometrical representation by associating $|\psi\rangle$ with a point on the surface of a Bloch sphere (Poincaré et al., 1892). A Bloch sphere is a sphere with a unitary radius where the two poles represent the two basis states $|0\rangle$ and $|1\rangle$, it will be useful when talking about unitary quantum gates since we can have a visual representation of the transformation applied to the qubit. To further explain the process, we first need to recall some properties of complex numbers. A complex number $z = a + ib$ can be represented by (a, b) in a Cartesian plan where one ax represents the real numbers and the other the imaginary ones. From this representation, we can obtain the polar one by computing the module of the complex vector z as $r = \sqrt{a^2 + b^2}$ and the angle ϕ between the vector and the ax of the real numbers.

z is now identified by the pair (r, ϕ) that can substitute (a, b) in the original formula:

$$z = r(\cos(\phi) + i \sin(\phi)) \tag{1.15}$$

Then we can use Euler's formula to obtain:

$$z = r(\cos(\phi) + i\sin(\phi)) = re^{i\phi}. \tag{1.16}$$

Using the notation in 1.16 to change (a, b) we can write our qubit as $|\phi\rangle = r_0 e^{\phi_0 i} |0\rangle + r_1 e^{\phi_1 i} |1\rangle$. Thanks to the normalization condition we have that $r_0^2 + r_1^2 = 1$, this allows us to write r_0 and r_1 as coordinate of a point belonging to a circumference of unitary radius: $r_0 = \cos(\rho)$ and $r_1 = \sin(\rho)$ with $\rho = \theta/2$. This reduces the modules to be dependent by the same parameter θ with $0 < \theta < \pi$:

$$|\phi\rangle = \cos(\theta/2)e^{i\phi_0} |0\rangle + \sin(\theta/2)e^{i\phi_1} |1\rangle . \tag{1.17}$$

Now we can extract the global phase e^{γ} obtaining:

$$|\phi\rangle = e^{i\gamma}(\cos(\theta/2)e^{i} |0\rangle + sin(\theta/2)e^{i\varphi} |1\rangle) \tag{1.18}$$

where $\varphi = \phi_1 - \phi_0$ and $\gamma = \phi_0$ with $0 < \varphi < 2\pi$.
When applying the measuring operator on a qubit we can see that the global phase $e^{i\gamma}$ doesn't influence the statistics of measurement between the states $|\psi\rangle$ and $e^{i\gamma} |\psi\rangle$:

$$\langle\psi| M_m^\dagger M_m |\psi\rangle = \langle\psi| e^{-i\gamma} M_m^\dagger M_m e^{i\gamma} |\psi\rangle \tag{1.19}$$

for this reason, we can remove $e^{i\gamma}$ from the equation without altering the results. To conclude we now see that the parameter φ can be seen as the angle of the projection of a point on the xy plane while the parameter θ can be seen as the spherical angle of a point for the axis z. The pair (φ, θ) describes now a point on the surface of a unitary sphere where the poles correspond to the basis vectors $|0\rangle$ and $|1\rangle$:

$$\begin{aligned} \theta = 0 &\implies |\psi\rangle = |0\rangle \\ \theta = \pi &\implies |\psi\rangle = |1\rangle \end{aligned} \tag{1.20}$$

1.2.3 Quantum registers

The true advantages of quantum computation lie in the use of multiple qubits at the same time or in the use of what is called a quantum register.

A quantum register is an ordered group of qubits used during a computation, the difference between this and its classical counterpart can be found in the different computational power they provide: a classical register is a group of n bits from which we can identify 2^n different states while a quantum register is made of n qubits and generates a 2^n-dimensional Hilbert space ($\mathbb{C}^{2^n}$). Inside this space every normalized vector is considered a possible state, and we can consider a quantum register $|\psi\rangle$ as the tensor product of n vectors $|i_j\rangle$:

$$|\psi\rangle = \bigotimes_{j=1}^{n} |i_j\rangle \iff |i_1 i_2 ... i_{n-1} i_n\rangle \,. \tag{1.21}$$

This formula is a direct use of postulate 4 and as it was already mentioned, the composition of quantum systems generates interesting and useful properties: *superposition*, *interference* and *entanglement*.

Superposition is the mathematical formulation for the quantum phenomena of particle-wave duality[1].We already saw an example of superposition when we considered a single qubit. It can be described as the contemporary existence of two different states (in our specific case we considered $|0\rangle$ and $|1\rangle$). When considering more than one qubit we can see that the number of dimensions (i.e., the number of possible basis states) of the system increase exponentially allowing us to evaluate many input state at the same time.

Interference is closely related to the wave nature of quantum and describes how two systems can interact depending on their sources or wavelength. One of the consequences of interference could be the damage of the quantum state by external noise,

[1]Physical property of quantum that can be described simultaneously as particle and waves without being fully described by one of the two formulations.

but for our purpose, we consider a perfectly error-corrected system (Section 1.1.2) where this phenomenon has no relevance in practice.

Entanglement is a property of a quantum state that allows a part to influence the whole system, it is the property that allows us to define and perform critical procedures of quantum computation such as quantum teleportation and super-dense coding. A more in-depth description of entanglement can be found in section 1.4

1.3 Quantum circuits and quantum gates

Like in classical computation to change the state of each fundamental unit we utilize different gates depending on the transformation we desire. From postulate 2.1 we know that all transformation U on a quantum system can only be unitary, meaning that for every U the relationship $U^\dagger U = I$ must be true. This kind of transformation also preserve the normalization condition as seen before.

A quantum circuit consists of applications of one or more quantum gates on one or more qubits and it's represented with a structure like the one shown in figure 1.1.

Each line corresponds to the qubits' lifetime, following these lines we can see where

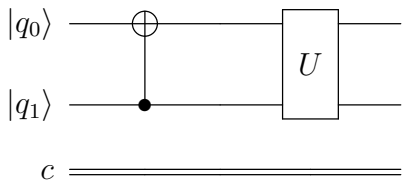


Figure 1.1: Example of a quantum circuit.

and when each gate is applied. Double lines indicate a classical bit used to store the information after the measurement.

It is worth reminding that the symbol $|\cdot\rangle$ corresponds to a column vector and that the

application of a unitary transformation can be seen from a mathematical viewpoint as matrix multiplication.

1.3.1 Unitary gates on single qubits

Single qubit gates are operations that resemble the classical *Not*, they operate on the state of a single qubit, and can be visualized as a shift in the position of the qubit on the Bloch sphere. We will now describe in detail the most useful gates and the class under which they fall: *Pauli gates*, *Rotation gates* and the *common gates* that don't belong to the previous classes.

Pauli gates are the gates that describe rotation with a fixed angle (π or 0) around the axis. They can be written as:

$$\begin{aligned} I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad X = \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ Y = \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \end{aligned} \tag{1.22}$$

and are represented in a quantum circuit as:

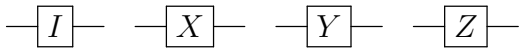


Figure 1.2: Pauli's gate representation in quantum circuits

The identity gate I is just an identity transformation that doesn't influence the state of the qubit. Later we will utilize this gate as a placeholder during the implementation of quantum Single Layer Perceptron (qSLP) placing it instead of an operator that we cannot yet implement (more details in section 3.3).

The Pauli X gate is the quantum equivalent of a classical *NOT* operator, in this

context, since a qubit can be found in a superposition of states, its behavior is more general and if it is applied to a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ it switches the role of the basis states:

$$X|\psi\rangle = X\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \tag{1.23}$$

This means that if $|\psi\rangle$ is in a basis state it goes to the other:

$$X|0\rangle = |1\rangle \quad X|1\rangle = |0\rangle . \tag{1.24}$$

The Pauli Y applies π rotations around the y axis while the Z gate applies a rotation of π around the z axis.

From the Paulis gate, we can derive the second class of gates *rotation gate*. They can be obtained by exponentiating Pauli matrices as described by the equation 1.25.

$$\begin{aligned} R_x(\theta) \equiv e^{-i\theta X/2} &= \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}X = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \\ R_y(\theta) \equiv e^{-i\theta Y/2} &= \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Y = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \\ R_z(\theta) \equiv e^{-i\theta Z/2} &= \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta Z/2} \end{bmatrix}. \end{aligned} \tag{1.25}$$

Lastly, we want to focus our attention on three meaningful gates: Hadamard gate (H), phase gate (S), and $\pi/8$ gate (also T gate):

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{bmatrix}. \tag{1.26}$$

In particular, the Hadamard gate is very important in quantum computation because it allows us to put the qubit in standard superpositions and its application on $|0\rangle$ and $|1\rangle$ produces two important states $|+\rangle$ and $|-\rangle$ that represent another possible basis for a single qubit system. The H gate can be visualized as a rotation of $\frac{\pi}{2}$ around the

y axis and a reflection over the yz plane. Its behavior, when applied on the states $|0\rangle$ and $|1\rangle$, can be seen in the following equation:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle$$
$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle$$

In figure 1.3 we show how the non-parametric gates discussed above are represented in a quantum circuit and their corresponding matrices.

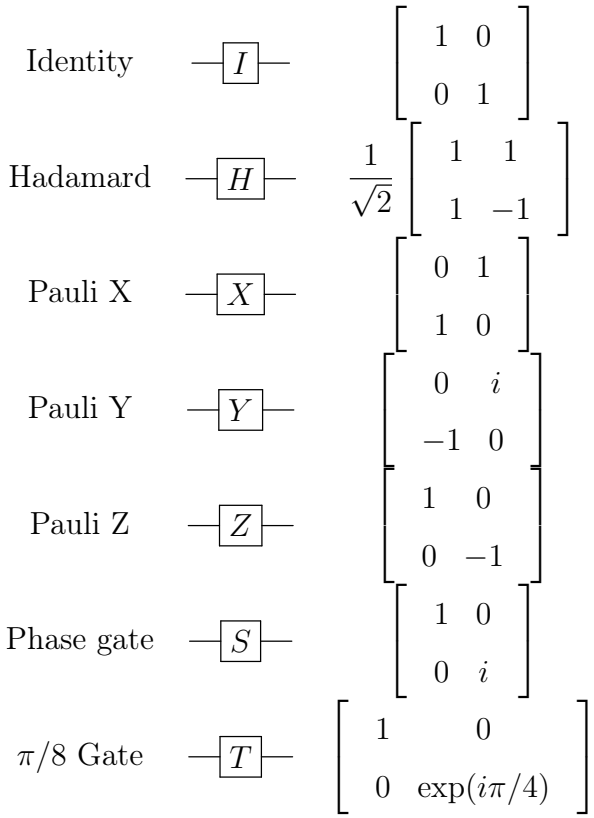


Figure 1.3: Summary of the most important non parametric gates with name, quantum circuit representation and associated matrices.

1.3.2 Controlled gates

We now have a way to operate a single qubit but we still don't know how two or more qubits can interact. In this section, we will discuss the most important multi-qubit gates and how they can exploit quantum effects in computation.

The first operator is the controlled not or CNOT gate: it applies an X gate to a target qubit depending on the state of a control qubit, implementing what we can write as *"if control is true than negate target"*. Mathematically the CNOT gate can be seen as an XOR gate and applies an addition modulo two ($\oplus$) to the qubits:

$$|\psi\rangle = |\psi_1, \psi_2\rangle \xrightarrow{CNOT} |\psi_1, \psi_2 \oplus \psi_1\rangle \tag{1.27}$$

Controlled gates, like single qubits gates, can be written as a matrix, and for the CNOT this matrix is the one we see in equation 1.28

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.28}$$

More generally we can implement a more generic phrase *"if control is true then apply f to target"* with a controlled unitary gate. Suppose we have a unitary gate U with:

$$U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix} \tag{1.29}$$

We can implement a controlled U (cU) gate as follows:

$$CU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix} \tag{1.30}$$

The application of controlled unitary gates is very useful as they allow us to exploit quantum parallelism by applying the same operation to every possible state of the qubit at the same time. As an example suppose we want to apply a generic operation U_f to an input state $|0\rangle$ conditioned by the state $|-\rangle$, we will obtain:

$$|-\rangle \otimes |0\rangle \xrightarrow{cU_f} |-\rangle \otimes |0 \oplus f(-)\rangle = \frac{|0f(0)\rangle + |1f(1)\rangle}{\sqrt{2}} \tag{1.31}$$

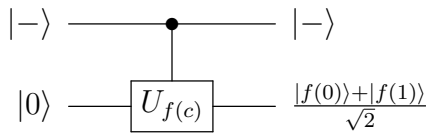
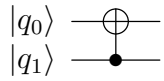


Figure 1.4: Circuit representation of a generic cU operation on the qubits $|-\rangle \otimes |0\rangle$.

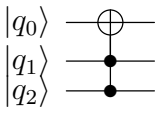
The last gate we present is called CCNOT or Toffoli gate. It is similar to the CNOT gate but considers the state of two control qubits before applying the X gate. Its behavior can be represented in the phrase *"if control 1 is true and control 2 is true then apply X to target"* and the representative matrix is:

$$CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.32}$$

CNOT and CCNOT gates can be seen in figure 1.5



(a) CNOT: $|q_0\rangle$ target and $|q1\rangle$ control



(b) CCNOT: $|q_0\rangle$ target and $|q1\rangle$ and $|q2\rangle$ control

Figure 1.5: Quantum gate representation for the CNOT (a) and CCNOT (b) gates.

1.4 Entanglement

Quantum entanglement is the physical phenomenon that doesn't allow us to describe the state of a single unite of an entangled system without referencing the rest of the system, even when the single parts are separated. This property is the core of quantum mechanics and one of the most powerful tools in quantum computation. Let's consider a practical example of a quantum register:
Given a quantum register $|\psi\rangle$ composed by two qubits, it can be described by the linear combination of the basis vectors $B = \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ of the space $\mathbb{C}^4$

$$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle \tag{1.33}$$

with $|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1$ following the normalization rule. Same as it was with a single Qubit here $|a_{xy}|^2$ represents the probability that upon measurement the system collapses in the state $|xy\rangle$. When we measure the first Qubit the whole system collapses and changes the possible combination and the relative probability. If i_0 is found in the state $|0\rangle$ the system will go from the state at equation 1.33 to:

$$|\psi\rangle = \frac{a_{00}|00\rangle + a_{01}|01\rangle}{\sqrt{|a_{00}|^2 + |a_{01}|^2}} \tag{1.34}$$

Losing $|10\rangle$ and $|11\rangle$ as possible combination and with re-normalized coefficients a_{0y}. In this example, we can see that the measurement of a single Qubit doesn't influence

the probability of the other, but this changes if we entangle the states of the register. Let's consider now the EPR pair (or Bell state) discussed by Einstein, Podolsky, and Rosen. During a mental exercise they hypothesize a pair of entangled particles (qubits in this case) that can be described by the state:

$$|\psi\rangle = \frac{|11\rangle + |00\rangle}{\sqrt{2}} \tag{1.35}$$

This register has the same probability to be found in the two states. The two qubits are then separated and given to two different people, Alice and Bob, that start traveling in opposite directions. As long as both qubits are still part of the same closed system, upon measurement, both Alice and Bob can obtain one of the two states with 50% probability. This changes if, after traveling a great distance, Bob measures its qubit forcing it to assume a specific value and making the whole system collapse. The collapsed register now changes the probability of Alice's qubit that once measured will output the same results as Bob's. In this case, the measurement of one qubit influenced the whole register even at a great distance.

1.5 Measurement

We can describe measurement as the action that recovers the result 0 or 1 with respective probability $|\alpha|^2$ and $|\beta|^2$ from a qubit $|\psi\rangle$ in the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ leaving the qubit in one of the two basis. More generally we can perform measurement with respect to a general $\{|a\rangle, |b\rangle\}$ orthonormal basis state but for the purpose of this thesis we will always refer to the basis $S = \{|0\rangle, |1\rangle\}$.

From postulate 3 in section 1.1.3 we can derive the definition of projective measurement:

Definition 1 (**Projective measurements**). *A projective measurement can be de-*

scribed by an observable M. An observable is a Hermitian operation on the observed space. The observable can be decomposed thanks to the projection P_m onto its eigenspace:

$$M = \sum_m m P_m \tag{1.36}$$

Here m are the eigenvalue and represent the possible outcome of the observation and their probability can be described as

$$p(m) = \langle\psi| P_M |psi\rangle \tag{1.37}$$

After the measurement and the retrieval of the outcome m, the system will collapse into the new state:

$$\frac{P_m |\psi\rangle}{\sqrt{p(m)}} \tag{1.38}$$

We focus on projective measurement because it has some advantageous properties, in particular it is easy to compute average values of measures:

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

This property allows us to simplify many calculation when applying a measurement operation.

Chapter 2

Quantum machine learning

This chapter will provide an in-depth description of the quantum machine learning algorithms that constitute the state of the art. We will begin with a classical machine learning introduction and with the challenge of state preparation. Later our focus will shift onto specific examples of machine learning algorithms and their quantum counterparts, in particular, we will see quantum variational algorithms, quantum neural networks, and quantum kernel methods.

2.1 Classical machine learning

When we talk about machine learning we are referring to a subsection of the more vast artificial intelligence field, where we have an agent that based on observation can improve its performance. The previously mentioned observation can come from three different sources based on the type of learning:

- *Unsupervised learning*: the agent learns pattern in the input even if there is no explicit feedback. The most common problem is clustering, where the agent tries to detect potentially useful cluster from unlabelled input examples.

- *Reinforced learning*: the agent is rewarded or punished based on the outcomes of its decision. A traditional example is an agent in a chess game that is rewarded if its decisions lead to the victory of the match and punished if the match is lost.

- *Supervised learning*: the agent has a set of input-output data and tries to learn a function to map unseen inputs to outputs. An example can be the binary classification of email into spam - not spam categories given some features of the email itself.

In the following sections will focus on the third class of problems as they are the ones that are considered in the experimental part of the thesis.

2.1.1 Supervised learning

We can formally describe it as:

Definition 2 (**Supervised learning**). *Given a training set T of N example input output pairs*

$$T = \{(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)\}$$

Where each y_j was generated by an unknown function $y = f(x)$. The algorithm aims to discover a function h that approximates the true function f.

Here we call h the hypothesis function and we measure its accuracy on a set unseen during training called test set. From this definition, we can distinguish two kinds of problems: classification and regression.
A classification problem is one where $y_i \in Y$ and Y is a finite set of possible answers much like the problem proposed before where email can be classified into one of the two classes $Y = \{\text{spam, not spam}\}$. A regression problem is one where Y is continuous

(E.g. a real number such as temperature).

When evaluating a hypothesis we make what is called a stationary assumption: there is a probability distribution $\mathcal{D}$ over examples that remains stationary over time. each data point is a random variable E_j whose observed value $e_j = (x_j, y_j)$ is sampled from $\mathcal{D}$ and is independent of the previous example with the same prior probability distribution over all variables:

$$\mathbf{P}(E_j|E_{j-1}, E_{j-2}, ...) = \mathbf{P}(E_j)$$
$$\mathbf{P}(E_j) = P(E_j) = P(E_{j-2}) = ...$$

Examples that satisfy these assumption are called independent and identically distributed (**i.i.d.**).

The objective of a supervised ML model 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:

$$y = f(x;\theta) + \epsilon, \tag{2.1}$$

where ϵ is a random variable whose conditioned probability distribution given x is centred in 0.

Although Equation (2.1) provides a general mathematical formulation for supervised learning, several methods do not estimate a single function but explicitly calculate 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 where the target variable is obtained by aggregating them:

$$y = f(x;\theta) = \sum_{h=1}^{H} \beta_h g(x;\theta_h), \tag{2.2}$$

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

The calculation of $g(x;\theta_h)$ corresponds to a specific transformation of data x based on θ_h, whose contribution to the final output is weighted by β_h. The estimation of a collection of functions component allows producing an extremely flexible model, which is able to approximate the behaviour of complex patterns. Different choices for β, $g(x;\cdot)$ and θ_h determine different ML models commonly adopted in real-world applications.

For instance, a single-layer neural network (or Single Layer Perceptron - SLP) with H hidden neurons is a two-stage regression or classification model that takes as input a training data x and $H+1$ sets of linear coefficients, and computes the target variable as follows:

$$f_{\text{SLP}}(x) = \sigma_{\text{output}}\left[\sum_{h=1}^{H} \beta_h \sigma_{\text{hidden}}\left(L(x;\Theta_h)\right)\right], \tag{2.3}$$

where σ_{output} is the identity function when the task is the function approximation. The SLP assumes as function component $g(x;\cdot)$ the activation function σ_{hidden} that takes as input the linear combination $L(x;\theta_h)$ of the input vector x . If considering a neural network with multiple hidden layers, the only difference in Equation (2.3) is that the function component $g(x;\cdot)$ is, in turn, a neural network.

2.1.2 Agent training

Now that we have defined what the agent has to achieve we show how it can improve its hypothesis through the loss function.

We want an agent that obtains the best fit over unseen data, or equivalent, that can well approximate the target function f. To do so we have to define the error rate of the hypothesis as the rate of the wrong prediction it makes. Decreasing the error

rate on the test set is the first way to improve a model, but we know that we cannot evaluate it on the same set on which we perform training since it could become overfitted.
Furthermore, minimizing the error rate is not the best solution to improve performances since often we need to give different weight on different errors (E.g. In a regression with $y = 1$ a $\hat{y} = 0.98$ is better than $\hat{y} = 0.1$) and we need a function that measures the utility in order to maximise it. We define the loss function $\mathcal{L}$ as a function that computes the amount of utility lost when making a prediction $\hat{y}$.

$$\mathcal{L}(x, y, \hat{y}) = Utility(\text{using } y = f(x)) - Utility(\text{using } \hat{y} = h(x))$$

In general we want to minimize the loss function, in doing so we will estimate the best hypothesis

2.2 State preparation

While creating a quantum algorithm the work can be divided into three different parts: state preparation, circuit, and measurement. Measurement description can be found in section 1.5 and subsection 1.1.3, and later we will discuss which qubits will contain the wanted results as to take into consideration only useful outputs. Circuits will be discussed in detail in chapter 3 when an implementation of the Multiple Aggregator Quantum Algorithm will be presented. We focus now on state preparation, the procedure that describes the encoding mechanism of classical data into quantum states.
State preparation is still an open problem in quantum computing, in literature several methodologies have been proposed but without reaching a satisfactory conclusion on which is the best way to encode classical data into a quantum computer. The most utilized techniques are *Basis encoding* and *amplitude encoding,* followed by *Hamilto-*

nian encoding (see table 2.1). The first associates a computational basis state of an n-qubit system with the state of n bits, the second stores classical vectors as quantum state amplitudes, and the third associates the Hamiltonian of a system with a matrix representing meaningful transformation to the original data (for more information see Maria Schuld (2018).

Many quantum algorithms in the literature assume to employ QRAM (quantum

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

Table 2.1: Strategies for encoding N point and p features into a quantum circuit

random access memory), a theoretical device that stores classical information as the amplitudes of a quantum state (Giovannetti et al., 2007), but to this day is not clear if such device can be created. A feasible alternative can be found in Mottonen et al. (2004) that solve the problem of mapping an arbitrary state $|\psi\rangle$ into the state $|0, ..., 0\rangle$, and once the circuit is found it's reversed. In this case, some preprocessing is needed since the real values need to be converted into angles to apply different controlled rotations. This last method is used in practice during the implementation phase for the circuit described in subsection 3.3.2.

The other two state preparation algorithms that will be used in this thesis are by Shende et al. (2004) and Havlicek et al. (2018). The first one focuses on disentangling a quantum register $|\psi\rangle$ bringing it to a basis state and later reversing the circuit similarly as in Mottenen et al.. The second was originally devised as a quantum kernel method (subsection 2.5) and focuses on exploiting the higher dimension of the

quantum space. Here the input data $\vec{x} \in \mathbf{R}^n$ is transformed as

$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S \subseteq [n]} \phi_S(\vec{x}) \prod_{i \in S} P_i\right) \tag{2.4}$$

Where P is a standard Pauli matrix and ϕ_S is the data-mapping function:

$$\phi_S(\vec{x}) = \begin{cases} x_0 \text{ if } k = 1 \\ \prod_{j \in S}(\pi - x_j) \text{ otherwise} \end{cases} \tag{2.5}$$

The circuit can contain many repetitions of this transformation but in this thesis, we will focus on two different variations that we will call ZFeaturemap and ZZFeaturemap following the naming convention from Qiskit (Table 2.2).

Name	Repetition	P	ϕ_S
ZFeaturemap	2	Z Pauli's matrix	x
ZZFeaturemap	2	ZZ Pauli's Matrices	x

Table 2.2: Two specific instances of Pauli feature maps that will be used in this thesis.

2.3 Quantum variational algorithm

Quantum variational algorithms, proposed by Moll et al. (2017) and Wecker et al. (2015) are among the most promising algorithms for the noisy intermediate quantum computing era (NISQ). They consist in exploiting both quantum and classical technology for optimization problems and are constituted by three main parts: a quantum parametric circuit, a quantum measurement operation, and a classical update rule for the parameters (Figure 2.1).

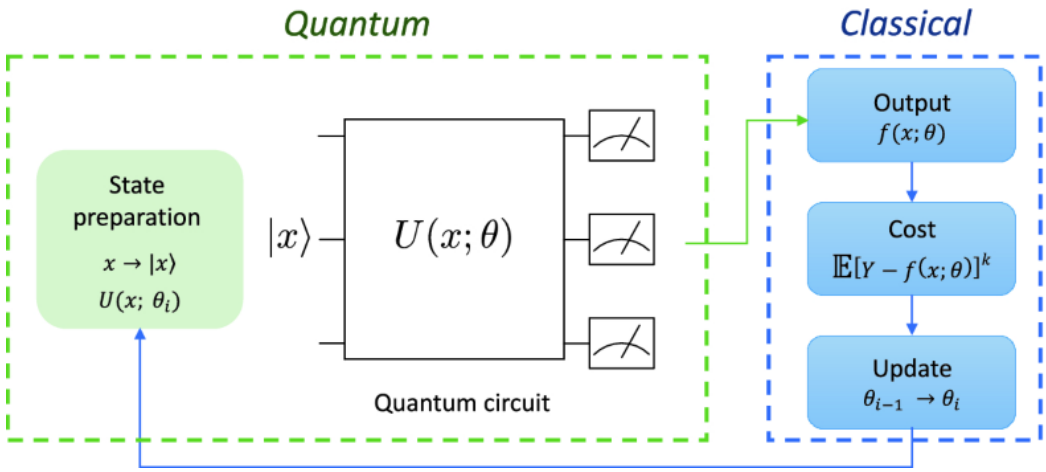


Figure 2.1: Structure of a Quantum variational algorithm. In the image the green highlights the quantum part while the blue the classical.

Source: Macaluso et al. (2020b)

In quantum variational algorithms the data x is preprocessed in a classical environment to be inserted into a quantum state through a state preparation circuit, $|x\rangle$ is then passed in the parameterized circuit $U(x, \theta)$ that starts with randomly initialized parameters. The circuit is run n times until a prediction $f(x, \theta)$ can be computed. The procedure finishes by classically updating the parameters and starting again the cycle (See algorithm 1).

Machine learning applications of variational algorithms can be found in Biamonte et al. (2016), Ristè et al. (2015), and Benedetti et al. (2019), but we will focus on the proposal made by Schuld et al. (2018). They presented a low-depth variational algorithm for classification that exploits amplitude encoding to perform a single-qubit measurement and parametric gates to keep a general use case.

While being very promising, variational algorithms often encounter the problem of Barren Plateaus (McClean et al., 2018): the exponential dimension of the Hilbert space reduces the probability that the gradient along any dimension is non-zero,

Algorithm 1 Quantum variational algorithm

$\theta \leftarrow ran()$

for $x \in X$ **do**

$x' \leftarrow preproces(x)$

$|x\rangle \leftarrow state\ preparation(x)$

$f(x, \theta) \leftarrow execute_n(U(|x\rangle, \theta))$

$\theta \leftarrow update(\theta)$

end for

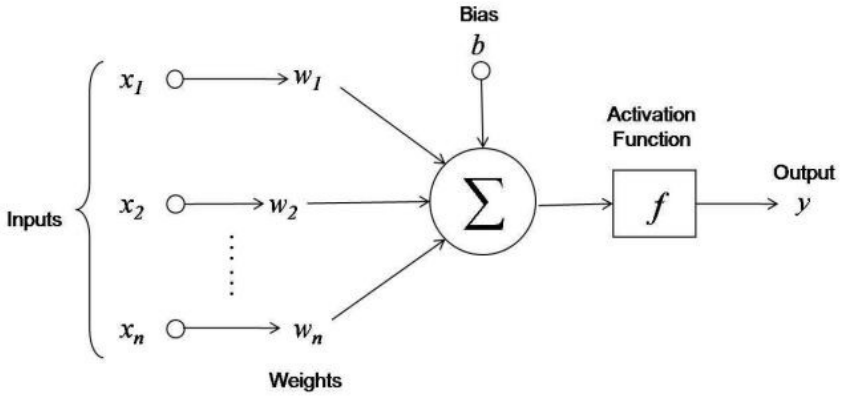
specifically it gets exponentially small as a function of the number of qubits utilized. From a practical viewpoint, it means that quantum variational algorithms are not suitable for running on more than a few qubits.

2.4 Quantum neural networks

With Neural Network (NN) we define a type of Machine learning algorithm inspired by a biological neural network.

2.4.1 Classical NN

Classical NN originated in 1943 with the "neuron" devised by McCulloch and Pitts (1943) (figure 2.2) Here the partial output of a single neuron is computed as the sum of the weighted inputs and the bias. The result is then passed through what is called an activation function (note that most of the time the activation functions are non-linear functions).



i

Figure 2.2: The figure shows the structure of a neuron. Here $y = f(\sum_{i=1}^{n}(x_i w_i) + b)$.

The neuron is the building block of modern neural networks that are built by stacking multiple neurons in layers. We can highlight three types of layers based on their purpose: input, output, and hidden. The relative positions and link between layers depend on the type of neural network and its depth. In a feed-forward network each layer n_i takes in input the layer n_{i-1} and is the input for layer n_{i+1}. Training is done through the computation of the loss and the gradient descent that updates the weights in a procedure called backpropagation.
The most simple neural network is called single hidden layer perceptron (SLP): it is a fully connected feed-forward network where the inputs are connected to a single hidden layer that feeds the outputs. We will see its quantum counterpart in section 3.3.

2.4.2 Quantum architectures

With quantum neural networks, we refer to a class of quantum algorithms that takes inspiration from classical neural networks without producing a precise quan-

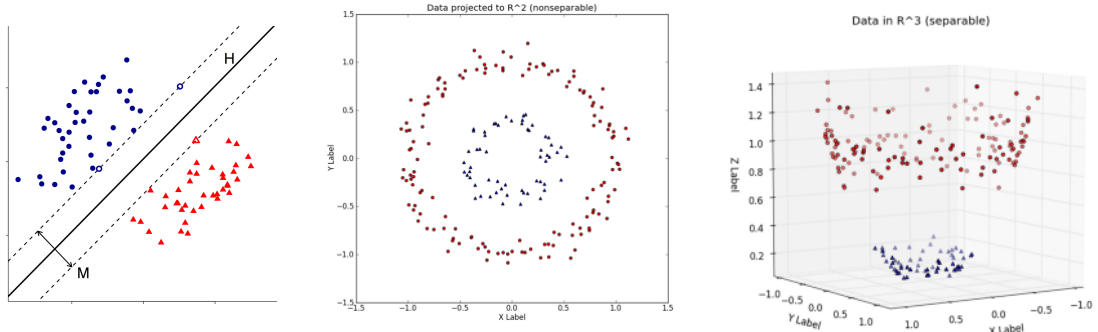
tum counterpart, but simulating layers by using parametrized gates. As of today, no quantum algorithm can encode the output of a classical neural network into a quantum state. Many proposal have been presented in literature (Gupta and Zia, 2002; J. Faber, 2002; Schuld et al., 2014a,b; Schützhold, 2002; Trugenberger, 2002), but all face the same problem: quantum mechanics postulates forbid the use of non-linear operations on quantum states. From a classical machine learning perspective this means that, in principle, it's impossible to embed a non-linear activation function in quantum neural networks Some models are also based on the Hopfield networks (Hopfield, 1982) focusing on associative memory that is derived by neuroscience rather than machine learning (Behrman et al., 1999; J. Faber, 2002; Toth et al., 1996).
Recently many have proposed quantum neural networks that exploit hybrid approaches, one example of a concrete implementation in a near-term processor is given by Tacchino et al. (2018). They introduced a model for binary classification that utilizes a perceptron-like updating rule, revealing exponential advantages in storage resources with respect to classical alternatives.

2.5 Quantum kernel methods

Support vector machines (SVM) is a supervised learning technique useful when there's no prior knowledge about the domain, it creates a maximum margin separator and tries to linearly separate the data with a hyper-plane. Usually, this is not possible so SVM utilizes what is called a kernel trick to add meaningful dimensions that allow for a linear separation. SVM are classified as non-parametric methods even if they retain only a small fraction of examples without the need to preserve all.
The objective of an SVM is to find the hyper-plane that can separate different classes of data with the largest margin as in figure 2.3a, if this is not possible a kernel function can be applied to exploit higher dimensions as we can see in figure 2.3c.

The SVM algorithm has a time complexity of $\mathcal{O}(\log(\epsilon^{-1}\text{poly}(N, M))$ where N is the dimension of the feature space, M the number of training vectors and ϵ the accuracy (Stephen Boyd, 2004).

Quantum SVM exploits the higher dimensional of the Hilbert space to better divide



(a) (b) (c)

Figure 2.3: Different applications of svms: 2.3a shows linearly separable data and the hyperplane that maximise margins, 2.3b shows non separable data that can be moved in higher dimension through a kernel function to make them separable 2.3c

the data with a smaller cost, It was proved by Rebentrost that a quantum support vector machine can be implemented with $\mathcal{O}(\log NM)$ run time in both training and classification (Rebentrost et al., 2013).

Schuld and Killoran (2018) explored the relationship between quantum state and feature maps, highlighting the need for both quantum computing and kernel method of computation in high dimensional Hilbert space. In their paper they state that encoding a vector x into a state $|\psi(x)\rangle$ is equivalent to the feature mapping action.

Despite the predicted advantages, a full quantum algorithm for SVM still doesn't exist but new technologies could create new use-cases for SVM that, as of today, is limited due to computational effort.

For the rest of the thesis when talking about SVM we will refer to the approach

presented by Havlicek et al. (2018) that focuses on creating a variational circuit, that generates a separating hyperplane in the quantum feature space (Farhi et al., 2017; Farhi and Neven, 2018; Kandala et al., 2017; Mitarai et al., 2018) .

2.6 Research contribution

The main contribution of this thesis is the realization of a generalized quantum Single Layer Perceptron (qSLP) by means of the Multiple Aggregator Quantum Algorithm (MAQA), a theoretical framework for quantum machine learning proposed by Macaluso (2021). The MAQA leverages the three main properties of quantum computing (superposition, entanglement, and interference) to reproduce all machine learning models based on the idea of function aggregation. One of such models is the classical Single Layer Perceptron, whose quantum counterpart is implemented in this thesis. Specifically, we produce the source code for a generic qSLP as a quantum variational algorithm.

The original proposal for the qSLP has many theoretical advantages over its classical counterpart thanks to the universal approximation theorem that tells us that SLPs can approximate any continuous bounded function given enough nodes. In the context of quantum computing, we can devise a quantum single-layer perceptron architecture where the number of neurons in the hidden layer scales exponentially with the number of qubits, whereas the classical counterpart scales only linearly. Furthermore, we propose the first-ever realization of the generalized qSLP, built as a parametrized quantum circuit that can be plugged in and trained alongside other quantum architectures belonging to the most popular library for quantum computing (qiskit). This allows us to produce extensive experiments on real-world datasets, with models trained on simulations on classical machines and quantum computers.

Finally, we compare the implementation of the generalized qSLP with the other

three quantum architectures trained: a quantum support vector machine and two types of quantum neural network classifiers. The experiments suggest that our algorithm performs on par or better than the other models at the cost of a higher number of qubits.

Chapter 3

Methodology

In this chapter, we will describe the architecture of the models used for the experiments. For each model, we will provide the circuit and a description of the effect that it has on the input data.
Firstly, we will describe the multiple aggregator quantum algorithms (MAQA) as it was defined by Macaluso (2021) since it is the theoretical foundation for the quantum single layer perceptron implemented in the experiments. The MAQA framework introduces an exponential scaling in the number of the aggregated function with respect to a classical machine learning approach. Furthermore, it opens the possibility to implement many quantum machine learning models not yet present in the literature.
Second we discuss the theoretical foundations of a quantum single layer perceptron and how it can be utilised as a quantum variational algorithm.
In the third section, we describe the main focus of the thesis: two implementation for qSLP.
Lastly, we present three baseline models taken from the literature, in particular we will introduce two quantum neural network classifiers and a quantum support vector classifier, all proposed by qiskit as basic quantum machine learning models.

3.1 MAQA

The multiple aggregator quantum algorithm leverages three properties of quantum computing (entanglement, superposition and interference) to aggregate in a quantum state the sum of many different transformation of the input. It holds the theoretical ability to reproduce all machine learning models that can be represented by the function 2.2, providing advantages over classical models. Here we propose a short description of the algorithm, for a more in-depth description refer to Macaluso (2021) The algorithm starts with d qubits in the control register and n qubits in the data register and it is divided into four main parts: state preparation, multiple trajectories in superposition, transformation via interference, and measurement.

3.1.1 State preparation in MAQA

We already discussed state preparation in section 2.2. While the specific routine for state preparation will be discussed alongside each model in the following sections, we will now refer to two different state preparations: S_x and S_β. The first one is the state preparation circuit for the n-qubit data register and is applied to encode the inputs in a quantum state. The second is the circuit for the d-qubit control register and it is utilized to encode a set of parameters $\{\beta_i\}_{i=0,\dots,2^d-1}$.

$$|\phi_0\rangle = (S_\beta \otimes S_x)\,|0\rangle_{\text{control}} \otimes |0\rangle_{\text{data}} = \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_i\,|k\rangle \otimes |x\rangle \tag{3.1}$$

3.1.2 Multiple trajectories in superposition

The second step aims to generate 2^d different transformation of the inputs by entangling each possible state to a qubit in the control register. This unitary transforma-

tion on the inputs depending on a set of parameters Θ and a function $G(\theta_1, \theta_2, ..., \theta_{2^d})$[1] The implementation of G can be accomplished in d steps with two entanglement operation $g(x; \theta_{i,1})$ and $g(x; \theta_{i,2})$ between the i^{th} control qubit and $|x\rangle$. This is typically done with a controlled operation that targets $|x\rangle$ and uses as control the states of the current control qubit. To summarize, the second step of the multiple aggregation quantum algorithm produces:

$$|\phi_1\rangle = G(\theta_1, \theta_2, ..., \theta_{2^d}) |\phi_0\rangle \tag{3.2}$$

$$= \frac{1}{\sqrt{E}} \sum_{k=0}^{2^d-1} \beta_k |k\rangle G(\theta_k) |x\rangle \tag{3.3}$$

$$= \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_k |k\rangle |g(x, \theta_k)\rangle \tag{3.4}$$

This formulation can be obtained if for the generic control qubit c_i we perform two steps:

- A controlled unitary $C^{(1)}G(\theta_{i,1})$ is executed to entangle $|x\rangle$ with the state $|1\rangle$ of the control qubit c_i:

$$\begin{aligned} |\phi_{i,1}\rangle &= \left(C^{(1)}G(\theta_{i,1})\right) |c_i\rangle \otimes |x\rangle \\ &= \left(C^{(1)}G(\theta_{i,1})\right) (a_i |0\rangle + b_i |1\rangle) \otimes |x\rangle \\ &= (a_i |0\rangle |x\rangle + b_i |1\rangle G(\theta_{i,1}) |x\rangle)) \end{aligned} \tag{3.5}$$

- Then a controlled unitary $C^{(0)}G(\theta_{i,2})$ is executed to entangle $|x\rangle$ with the state $|0\rangle$ of the control qubit c_i

$$\begin{aligned} |\phi_i\rangle &= \left(C^{(0)}G(\theta_{i,2})\right) |\phi_{i,1}\rangle \\ &= \left(C^{(0)}G(\theta_{i,1})\right) (a_i |0\rangle |x\rangle + b_i |1\rangle G(\theta_{i,1}) |x\rangle)) \\ &= (a_i |0\rangle G(\theta_{i,2}) |x\rangle + b_i |1\rangle G(\theta_{i,1}) |x\rangle)) \end{aligned} \tag{3.6}$$

[1]Note that the generic assumed in the general formulation will assume a specific implementation in section 3.3

This process is repeated d times and produces 2^d different transformations, potentially leading to an exponential speed-up with respect to classical methods.

3.1.3 Transformation via interference

The third step in the process is to apply a generic quantum gate F to the data register.

$$\begin{aligned}
|\phi_f\rangle &= \left(\mathbb{1}^{\otimes d} \otimes F\right) |\phi_d\rangle \\
&= \left(\mathbb{1}^{\otimes d} \otimes F\right) \left[\frac{1}{\sqrt{E}} \sum_{k=0}^{2^d-1} \beta\, |k\rangle\, |g(x;\Theta_k)\rangle\right] \\
&= \frac{1}{\sqrt{E}} \sum_{k=0}^{2^d-1} \beta\, |k\rangle\, |f^*(x;\Theta_k)\rangle \\
&= \frac{1}{\sqrt{E}} \sum_{k=0}^{2^d-1} \beta\, |k\rangle\, |f_k^*\rangle
\end{aligned} \tag{3.7}$$

Here we make the assumption that $F(G(x;\theta_k)$ on the quantum state $|x\rangle$ is equivalent to the target function f_k^*. In this third step of the algorithm, we apply a function F on the circuit, but its application is propagated to all the 2^d superpositions. This could be a crucial advantage when we need to repeat many times the application of the same function as we do for the activation function of a neural network layer with lots of neurons.

3.1.4 Measurement

The fourth and last step is the measurement. We apply a measurement operator like the one described in section 1.5 and subsection 1.1.3 to the data register, obtaining the weighted average of all 2^d functions in a quantum version of equation 2.2.

The application of the measurement can be represented as follows:

$$\begin{aligned}
\langle M \rangle &= \langle \phi_f | \mathbb{1}^{\otimes d} \otimes M | \phi_f \rangle \\
&= \sum_{k=0}^{2^d-1} \beta' \langle k|k \rangle \otimes \langle f_k^* | M | f_k^* \rangle \\
&= \sum_{k=0}^{2^d-1} \beta' \langle f_k^* | M | f_k^* \rangle \\
&= \sum_{k=0}^{2^d-1} \beta' \langle M_k \rangle \\
&= \sum_{k=0}^{2^d-1} \beta' f_k = f_{agg}
\end{aligned} \tag{3.8}$$

where $f_k = \langle f_k^* | M | f_k^* \rangle$ and $\beta'_k = |\beta_k|^2$ with β_k following the normalization condition $\sum_k |\beta_k|^2 = 1$.

Following this procedure we are able to extract the result by measuring only the data register, furthermore we are able to change the learning algorithm by specifying different $S_\beta, S_x, \{G(\theta_{i,1}), G(\theta_{i,2}\}_{i=0,\dots,d-1}$ and F.

3.2 Variational algorithm for qSLP

This section describes how we use the MAQA framework to extend the quantum single layer perceptron Macaluso et al. (2020b) as variational algorithm.

A quantum variational algorithm can be divided in three parts, the classical update rule, the quantum circuit measurement and a parametric quantum circuit. In this section we dissect the quantum parametric circuit in three distinct sub-parts that depend on the specific implementation and that are specific for a quantum single layer perceptron: state preparation, linear gate operators and activation function.

3.2.1 State preparation

The state preparation algorithm utilized is a data amplitude encoding method that associates quantum amplitudes with a real vector of observations at the cost of introducing a normalization constraint. The normalized vector $x \in \mathbb{R}^{2^n}$ can be described as:

$$|x\rangle = \sum_{k=1}^{2^n} x_k |k\rangle \leftrightarrow x = \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix} \tag{3.9}$$

This allows using the index register to indicate the k^th feature. Furthermore, we only need n qubits to encode a vector of 2^n elements meaning that if an algorithm is polynomial in n it will be poly-logarithmic when computed in a quantum environment.

3.2.2 Linear gate operators

A parametric variational circuit $U(\theta)$ is composed by a series of parametric gates each one with a set of parameters $\{\theta_l\}_{l=1,\ldots,K}$. Formally $U(\theta)$ is the product of L matrices $\prod_{l=1}^{L} U_l$ each representing a single or multi qubits gate. The gates need to have learnable parameters, and to achieve that in the single-qubit gate we utilize a G defined by the unitary 2×2 matrix (Barenco et al., 1995):

$$G(\alpha, \beta, \gamma) = \begin{pmatrix} e^{i\beta} cos(\alpha/2) & e^{i\gamma} sin(\alpha/2) \\ -e^{-i\gamma} sin(\alpha/2) & e^{-i\beta} cos(\alpha/2) \end{pmatrix} \tag{3.10}$$

Thus, we can write each U_l in terms of G_i, a single qubit-gates acting on the i^{th} qubit:

$$U_p = \mathbb{1}_1 \otimes \ldots \otimes G_i \otimes \ldots \otimes \mathbb{1}_n \tag{3.11}$$

where n is the total number of qubits in the quantum system. Furthermore this representation allows us to compute the gradient analytically (Schuld et al., 2018).

3.2.3 Activation functions

Building an activation function is, to date, one of the biggest obstacles to the construction of a theoretical and complete quantum neural network. The restriction imposed by postulates 2.1 forces us to use linear transformations, meaning that it might be impossible to embed a non-linear activation in a qSLP.
One of the most famous approaches comes from Cao et al. (2017) that proposed a repeat-until-success approach to achieve non-linearity. This technique has the non-trivial limitation of requiring the input to be in the range $[0, \pi/2]$, which is a severe constraint for real-world scenarios. Another possible method consists in the application of quantum splines (QSpline) (Macaluso et al., 2020a), that approximate non-linear functions via quantum algorithms. Although the QSpline is a fitting method to compute the value of non-linear functions, it uses the HHL as subroutine which is a full coherent protocol with high computational requirements.
An implementation of a non-linear activation function is beyond the scope of this thesis, and for this reason, in the practical implementation, we set a single unitary matrix ($\mathbb{1}$ gate) as a placeholder. However, the algorithm still provides a parametric circuit able to train a qSLP for a given activation function Σ. The architecture can natively incorporate any implementation for Σ and it allows for the training of its parameters, like for instance, the one described by Hu (2018).

3.3 qSLP

This section provides the first part of the research contribution of this thesis by describing the development of a quantum single layer perceptron from the theory in section 3.1 and 3.2.
The advantages that come from the possibility of developing a quantum single hidden

layer perceptron derive from the universal approximation theorem and the exponential scaling in the number of neurons provided by a quantum implementation. From these properties, we know that, by using a qSLP, it is theoretically possible to approximate any continuous function on a closed and bounded subset of $\mathbb{R}$.
In subsection 2.1.1 we described the single hidden layer neural network as an aggregation of functions (Equation 2.3), now we will explain in detail how the two different proposed implementations work by describing their defining elements.

3.3.1 Single data qubit qSLP

We refer to the single data qubit qSLP as one of the two implementations proposed in this thesis. As the name suggests, the single data qubits qSLP only relies on a single qubit to store the data.
As we discussed in the previous sections the parametric quantum circuit of a quantum single layer perceptron is composed of four parts: state preparation, linear parametric gate operators or ansatz, activation function, and measurement operation. After the measurement, a classical update rule is applied and the parameters are updated.

State preparation. The state preparation phase differs between the control qubits and the data qubits. The control qubits start from the state $|0\rangle$ and a parametric $R_y(\beta_i)$ gate is applied. On the other hand, the encoding of the data qubit is slightly more sophisticated and relies on the procedure described by Shende et al. (2004) that performs the operation of bringing a quantum state to the state $|0\dots0\rangle$ and then reverses the circuit. For a single qubit this translates into two parametric rotation $R_y(x_1), R_z(x_2)$ where $\{x_1, x_2\}$ are two feature representing the input data x. For simplicity in equation 3.12 we consider the case with only one control qubit, but this

procedure can easily be extended to for any positive integer number d.

$$\begin{aligned} |\Phi_1\rangle &= \big(R_y(\beta) \otimes S_x\big)\, |\Phi_0\rangle = \big(R_y(\beta) \otimes S_x\big)\, |0\rangle\, |0\rangle \\ &= (\beta_1\, |0\rangle + \beta_2\, |1\rangle) \otimes |x\rangle = \beta_1\, |0\rangle\, |x\rangle + \beta_2\, |1\rangle\, |x\rangle), \end{aligned} \tag{3.12}$$

Where $S_x = R_y(x_1) \cdot R_z(x_2)$, $|\beta_1|^2 + |\beta_2|^2 = 1$ and $\beta_1, \beta_2 \in \mathbb{R}$

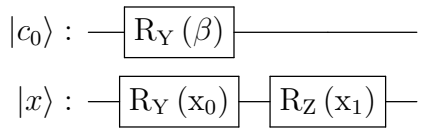


Figure 3.1: Quantum circuit for state preparation in a single data qubit qSLP with $d = 1$.

Ansatz. In the ansatz we exploit the procedure described in section 3.1 to generate two different linear operations in superposition, each entangled with a control qubit. We now show how a circuit with $d = 1$ is built:

- First a controlled U_3 gate is applied to the data qubit based on the state $|1\rangle$ of c_1. A generic cU_3 gate works as in equation 3.13 (IBMQiskit, 2022).

$$cU_3(\zeta, \phi, \lambda)\ q_0, q_1 = I \otimes |0\rangle\langle 0| + U3(\zeta, \phi, \lambda) \otimes |1\rangle\langle 1| \tag{3.13}$$

  For our purpose we will utilize $\theta_{i,j} = \{\zeta, \phi, \lambda\}_j$ for $j = 1, 2$ indicating the cU_3 gate. The transformation is described in the following equation:

$$\begin{aligned} |\phi_{i,1}\rangle &= \big[C^{(1)} \otimes U3(\theta_{i,1})\big]\, |\phi_{i-1}\rangle \\ &= \big[C^{(1)} \otimes U3(\theta_{i,1})\big]\, (\alpha_i\, |0\rangle + \beta_i\, |1\rangle) \otimes |x\rangle \\ &= (\alpha_i\, |0\rangle\, |x\rangle + \beta_i\, |1\rangle\, U3(\theta_{i,1})\, |x\rangle) \end{aligned} \tag{3.14}$$

- Then we apply a Pauli X gate on the i^{th} control qubit:

$$\begin{aligned} |\psi_{i,2}\rangle &= [X \otimes \mathbb{1}] \, |\phi_{i,1}\rangle \\ &= (\alpha_i \, |1\rangle \, |x\rangle + \beta_i \, |0\rangle \, U3(\theta_{i,1}) \, |x\rangle) \end{aligned} \tag{3.15}$$

- Finally we apply the second cU_3 gate:

$$\begin{aligned} |\phi_i\rangle &= \left[C^{(1)} \otimes U3(\theta_{i,2})\right] |\phi_{i,2}\rangle \\ &= (\alpha_i \, |1\rangle \, U3(\theta_{i,2}) \, |x\rangle + \beta_i \, |0\rangle \, U3(\theta_{i,1}) \, |x\rangle) \end{aligned} \tag{3.16}$$

When the circuit presents more than one control qubit ($d > 1$) we repeat this procedure for all control qubits, ending up with the state:

$$\begin{aligned} |\phi_d\rangle &= \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_k \, |k\rangle \, G(\theta_k) \, |x\rangle \\ &= \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_k \, |k\rangle \, |g(x;\theta_k)\rangle \end{aligned} \tag{3.17}$$

Where $G(\theta_k)$ is the product of d different $cU_3(\theta_{i,j})$ for $i = 1, \ldots, d$ and $j = 1, 2$

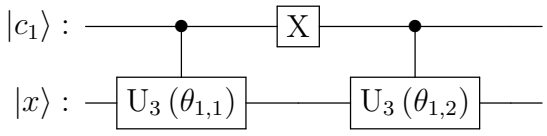


Figure 3.2: Example circuit for the ansatz of a single data qubit qSLP with $d = 1$

Activation function. Since an efficient implementation of a non-linear activation function doesn't exist yet (as we discussed in subsection 3.2.3), we use an unitary matrix $\mathbb{1}$ as a placeholder.

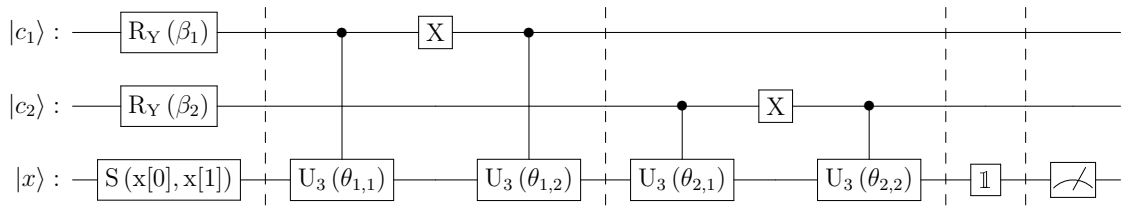


Figure 3.3: Quantum circuit of single data qubit qSLP with $d = 2$

Measurement. To obtain the needed result we don't have to measure the whole register $|\phi\rangle$. We can just perform expectation measurement with the Pauli-Z operator acting on the quantum state $|x\rangle$:

$$\langle M \rangle = \langle \Phi_0 | U^\dagger(\beta, \theta)(\mathbb{1} \otimes \sigma_z \otimes \mathbb{1}) U(\beta, \theta) | \Phi_0 \rangle = \pi(x; \beta, \theta), \tag{3.18}$$

Where $U(\beta, \theta)$ represents the qSLP circuit. To estimate $\pi(\cdot)$, we have to run the circuit multiple times. It is useful to see that the results can be obtained by a single measurement of the data qubit, leaving the control qubits untouched. An example of a more general, complete circuit with $d = 2$ is shown in figure 3.3.

Classical post-process One of the advantages of this model is the possibility to build a quantum circuit and to insert it in a standard function provided by the qiskit library (IBMQiskit, 2022). The function handles the classical part of the variational algorithm, but we have to provide an interpreter and a loss function.
Even if we only need the value of the data qubit as we discussed before, in practice, the function measures the whole quantum register $|\phi\rangle$ and transforms the bits obtained in an integer. The interpreter is a function $f(m)$ that takes an integer m representing the result of the measurement $\pi(\phi; \beta, \theta)$ and computes the true result of the quantum computation. Since we only need to measure the data qubit, our interpreter is a simple bit-wise *and* operation that returns the value of $|x\rangle$ based on its index on the

quantum register:

$$f(x, \beta, \theta) = \begin{cases} 1 & \text{if } \pi(\phi; \beta, \theta) \wedge 2^d \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

where 2^d indicates the index of $|x\rangle$ on the register $|\phi\rangle$ because it comes after d control qubits. We can interpret Eq. (3.19) directly as a probability distribution for a binary classification problem, whose final output is one of two possible classes.
The loss function we utilize is the *Sum of Squared Error* (SSE) loss:

$$SSE = Loss(\Theta; D) = \sum_{i=1}^{N} \left[y_i - f\left(x_i; \Theta\right) \right]^2 . \tag{3.20}$$

Lastly, since we utilize a state preparation function that doesn't allow the use of a gradient base optimizer we utilize the Constrained Optimization By Linear Approximation (COBYLA) optimizer (Powell, 2007, 1994, 1998).

3.3.2 Padded qSLP

The second implementation we propose is called padded qSLP. It utilizes a two-qubit data register where the input data are padded, encoded through a different state preparation technique, and processed by a different ansatz. As we did in the previous section we present now the implementation's details:

State preparation. The input data is normalized and preprocessed through a function $f_p(x)$ that given the classical data in input, it returns five rotation angles $\hat{x}_i$ for $i = 0, \ldots, 4$ used as parameter for multiple rotation gates. The circuit is then built by applying a sequence of cX gates, $R_y(x_i)$ and X gates as we can see in figure 3.4. For a more in-depth explanation of this state preparation technique see Motten et al.. Mottonen et al. (2004).

At the same time, the control qubits are initialized through a parametric rotation $R_y(\beta_i)$.

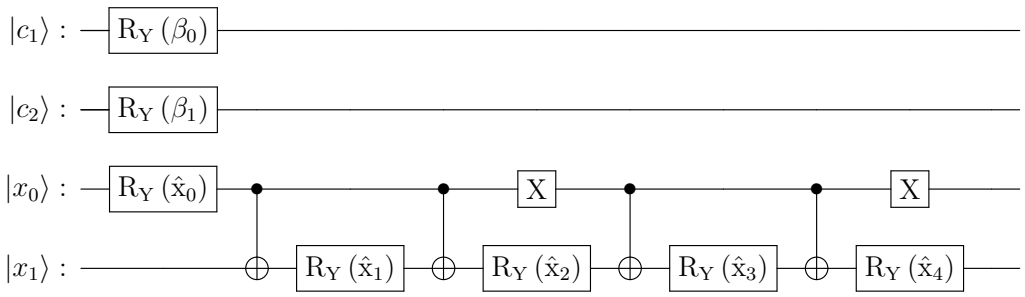


Figure 3.4: Example of a state preparation circuit in a padded qSLP with $d = 2$

For simplicity, from now on, we will refer to the state preparation of the data as the procedure S_x. The mathematical formalization of a circuit where $d = 1$ is the following:

$$\begin{aligned} |\Phi_1\rangle &= \big(R_y(\beta) \otimes S_x\big)\,|\Phi_0\rangle = \big(R_y(\beta) \otimes S_x\big)\,|0\rangle\,|0\rangle \\ &= (\beta_1\,|0\rangle + \beta_2\,|1\rangle) \otimes |x\rangle = \beta_1\,|0\rangle\,|x\rangle + \beta_2\,|1\rangle\,|x\rangle), \end{aligned} \tag{3.21}$$

Ansatz As in single data qubit qSLP we generate multiple linear operations at the same time by exploiting superposition and entanglement. For each control qubit c_i with $i = 1, ..., d$ we repeat the same five steps considering four different set of parameters $\theta_{i,j} = \{\zeta, \phi, \lambda\}_{i,j}$ for $j = 1, ..., 4$ indicating the different cU_3 gates (Equation 3.13).

- Apply $cU_3(\theta_{i,2})$ and $cU_3(\theta_{i,2})$ gates to the two data qubits $|x_0\rangle$, entangling them with the excited state of c_i,

- Apply a Toffoli gate (CCNOT) targeting $|x_1\rangle$, based on the excited state of $|c_i\rangle$ and $|x_0\rangle$.
- Apply a X gate to $|c_i\rangle$:
- Repeat the first step and apply a $cU_3(\theta_{i,3})$ and $cU_3(\theta_{i,4})$ respectively on $|x_0\rangle$ and $|x_1\rangle$, entangling them with the excited state of $|c_i\rangle$.
- Lastly Apply another CCNOT gate targeting $|x_1\rangle$ and controlling the excited state of $|c_i\rangle$ and $|x_0\rangle$

After we repeat this d times we end up with the following state:

$$\begin{aligned} |\phi_d\rangle &= \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_k |k\rangle G(\Theta_k) |x\rangle \\ &= \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_k |k\rangle |g(x;\Theta_k)\rangle \end{aligned} \tag{3.22}$$

Where $G(\Theta_{k,j})$ is the product of d different cU_3 gates and their respective parameter sets Θ_k for $k = 1, \ldots, d$ applied on the quantum register $|x\rangle = |x_0\ x_1\rangle$. A more intuitively view can be seen in figure 3.5 where an example of ansatz circuit for padded qSLP with $d = 1$ is presented.

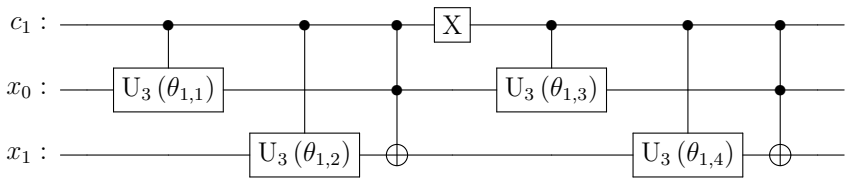


Figure 3.5: Example circuit of the ansatz of a padded qSLP with $d = 1$

Activation function. Since an efficient non linear activation function doesn't exists to this day (as we discussed in subsection 3.2.3), we use an unitary matrix $\mathbb{1}$ as a placeholder and we apply it to both $|x_0\rangle$ and $|x_1\rangle$.

Measurement The measurement can be expressed as the expected value of the Pauli-Z operator acting on the quantum state $|x_1\rangle$:

$$\langle M \rangle = \langle \Phi_d | U^\dagger(\beta, \theta)(\mathbb{1} \otimes \sigma_z \otimes \mathbb{1}) U(\beta, \theta) | \Phi_d \rangle = \pi(x_1; \beta, \theta), \tag{3.23}$$

Where $U(\beta, \theta)$ represents the qSLP circuit. To estimate $\pi(\cdot)$, we have to run the circuit multiple times. It is useful to see that the results can be obtained by a single measurement of one of the data qubits, leaving the other qubits untouched.

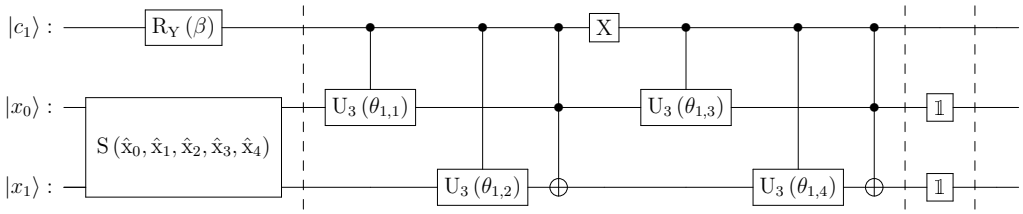


Figure 3.6: Quantum circuit for padded sQLP with d = 1

Classical update. One of the advantages of this model is the possibility to build a quantum circuit and to insert it in a standard function provided by the qiskit library. The function handles the classical part of the variational algorithm but we have to provide an interpreter and a loss function.

Even if we only need the value of one qubit in the data register, as we discussed before, in practice, the function measures the whole quantum register and transforms the bits obtained in an integer. The interpreter is a function $f(m)$ that takes in input an integer m representing the result of the measurement $\pi(\phi; \beta, \theta)$ and computes the

true result of the quantum computation. Since we only need to measure one of the data qubits, our interpreter is a simple bit-wise *and* operation that returns the value of $|x_1\rangle$ based on its index on the quantum register:

$$f(x, \beta, \theta) = \begin{cases} 1 & \text{if } \pi(\phi; \beta, \theta) \wedge 2^{d+1} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.24}$$

where 2^{d+1} indicates the index of $|x_1\rangle$ on the register $|\phi\rangle$ because it comes after d control qubits and $|x_0\rangle$. We will perform binary classification, for this reason, the possible results are in $\{1, 0\}$. As loss function we utilize the *Sum of Squared Error* (SSE) loss introduced in subsection 3.3.1. Lastly, we utilize Constrained Optimization By Linear Approximation (COBYLA) (Powell, 2007, 1994, 1998) as optimizer.

3.4 Baseline models.

This section presents the implementation of two classes of quantum classifiers (quantum neural network classifier QNNC and quantum support vector classifier QSVC) developed from an initial proposal by the qiskit library.

First, we present two QNNC models. The proposed quantum neural network classifiers are variational algorithms, and, as such, they are constituted by a quantum parametric circuit, a measurement operation, and a classical update rule.

The two implementations presented in the following sections have different quantum circuits, but the same measurement and post-processing are applied. For this reason, we will first describe the state preparation and ansatz of each model and later present how the results are extracted and how the parameters are updated.

Finally, we discuss the implementation of the QSVC, focusing on its kernel method. All the models in this section utilize a 2-qubit register to encode the data vector.

3.4.1 QNNC v1

QNNC v1 is the first baseline model based on the qnnc structure proposed by qiskit. The difference between the two architecture is small but it can highly influence the final performance. We included both models in the study since they seem to perform better in different scenario.

State preparation. The state preparation is done employing a *ZZFeature-map*. A *ZZFeature-map* is a state preparation circuit that was first introduced as a quantum kernel method. It belongs to the family of Pauli expansion circuits, and it makes heavy use of the U_1 gate, a unitary gate that can be represented by the matrix:

$$U_1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix} \tag{3.25}$$

The procedure to build the *ZZFeature-map* is the following:

- Apply a Hadamard gate to each qubit, leaving it in a superposition.
- Apply $U_1(2x_0)$ gate to $|q_0\rangle$ and $U_1(2x_1)$ to $|q_1\rangle$.
- Apply a cX gate with $|q_1\rangle$ as target and $|q_0\rangle$ as control.
- Apply a $U_1(\gamma)$ on $|q_1\rangle$ with $\gamma = 2 * (\pi - x_0) * (\pi - x_1)$
- Lastly, apply a cX gate as in the second step.

Where $\{x_0, x_1\}$ are two feature that represent the input x. In the end, the register undergoes the transformation described in section 2.2.
The circuit can be seen in figure 3.7.

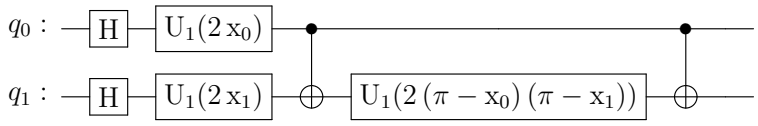


Figure 3.7: ZZFeature-map: state preparation circuit for QNNC v1

Ansatz For the ansatz, we utilize the *real amplitude* circuit, a heuristic trial wave function typically used as an ansatz in chemistry applications or, like this case, classification circuits in machine learning. it consists in the application of $R_y(\theta_j)$ rotations, cX entanglements gates and $R_y(\theta_j)$ again to each qubit. The name, real amplitudes, comes from the fact that the prepared state will only contain real amplitude numbers where their complex part is always 0.

The circuit of a QNNC v1, represented in figure 3.8, uses four parameters θ_i with $i = 1, ..., 4$

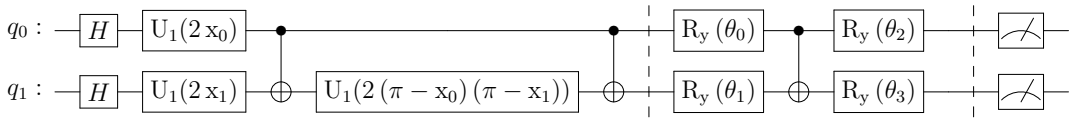


Figure 3.8: QNNC v1 complete circuit.

3.4.2 QNNC v2

QNNC v2 is the second baseline model based on the original proposal by qiskit. It has a more simple state preparation circuit but a more complex ansatz when compared to QNNC v1, effectively showing how a small change in the quantum circuit can change the performance.

State preparation The state preparation for QNNC v2 is done with the *ZFeature-map* circuit. Similar to the aforementioned *ZZFeature-map*, the *ZFeature-map* is a state preparation technique that belongs to the Pauli expansion circuits group. It was firstly described by Havlicek et al. (2018) as a kernel method and, in the case of a two-qubit register, it consists in the application of an H gate, followed by a U_1 gate on both the qubits.

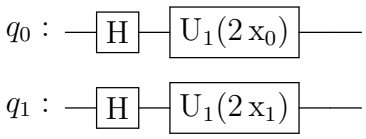


Figure 3.9: ZFeature-map: State preparation circuit for QNNC v2.

Ansatz The ansatz utilized is *Real amplitude* as in QNNC v1. The difference between the two models is in the number of repetitions. In QNNC v2 we repeat twice the pair $\{R_y(\theta_i),\, cX\}$ before the final rotation, increasing the number of parameter θ_i

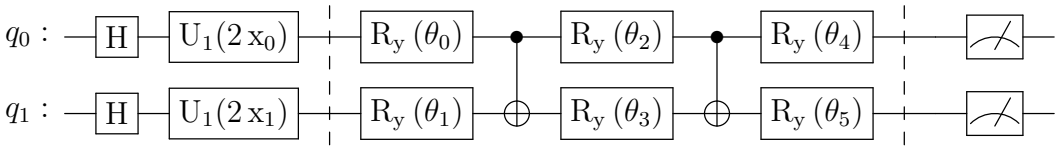


Figure 3.10: QNNC v2 complete circuit

3.4.3 Measurement and classical process in QNNC

This subsection explains the measurement operations and the classical part of the QNNC models. Since they have the same basic structure and the same number of qubits, the training strategy explained here is applied for both.

Measurement Since the input data is encoded in the two qubits of the register and the ansatz involves both, all of them are measured at the end of the computation.

Classical computation The final result will be a two-digit binary number interpreted as an integer m. We reverse the bit to integer transformations and apply a function $f(q_0, q_1)$ that, given the two bits representing the measurement results, computes the predicted class through the parity function:

$$f(q_0, q_1) = q_0 \oplus q_1 \tag{3.26}$$

The parity function computes the number of bits with value 1 and returns 0 if the number is even (even parity) or returns 1 if the number is odd (odd parity).
The loss is computed as the sum of squared error (SSE) (see subsection3.3.1), and the whole process is optimized with the COBYLA optimizer.

3.4.4 QSVC

The quantum support vector classifier (QSVC) utilized in this thesis implements the theoretical aspects discussed in 2.5. We chose to use a *ZZFeature-map* as kernel method. As the name suggests, this state preparation technique allows us to map the input features into a quantum feature space that is inherently bigger. Even if the name is the same, this circuit is twice deeper than the one presented in subsection 3.4.1. This design choice comes from the fact that to obtain an advantage over classical computation, the feature mapping in a quantum state should be hard to simulate

classically, and for this reason, we decided to utilize a more complex circuit. In figure 3.11 we can see the feature mapping circuit.

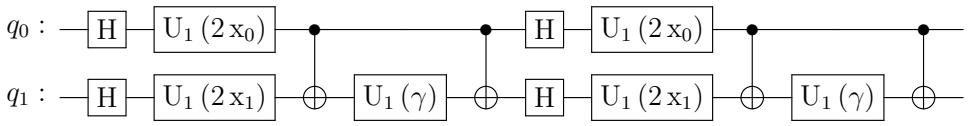


Figure 3.11: ZZFeature map with two repetitions. Here $\gamma = 2(\pi - x_0)(\pi - x_1)$

3.5 Discussion

As discussed in section 3.3 we were able to build a quantum single layer perceptron through the use of the multiple aggregator quantum algorithm frameworks. Since the architecture follows the theoretical approach proposed by Macaluso (2021), it can natively generate an exponential number of neurons while scaling only linearly in the number of steps. The state preparation for both implementations of qSLP uses an amplitude encoding strategy that grants us exponential advantages in terms of space complexity while encoding the data register. This also implies a poly-logarithmic advantage in the number of parameters when comparing it with its classical counterpart.

One thing that seems different between qSLP and its classical implementation is the normalization constraint (subsection: 1.1.4) that is applied to the data and the weights in a quantum scenario. However, in traditional neural networks, rescaling inputs and limiting weights' magnitude are common strategies adopted, called respectively batch normalization and weight decay. Thus the constraints introduced by quantum computations are the automatic implementation of ad-hoc procedures specific for neural networks.

From a computational viewpoint, the training of a classical neural network has a computational cost in terms of operations $\mathcal{O}(NpHL)$, that scales linearly with respect to the number of observations N, the features p , the hidden neurons H and the training epochs L. Even if modern GPU and parallel computing grant a speed up in the process, when the number of neuron H greatly increases, as required for SLP to be a universal approximator, it becomes impossible to train an algorithm in a reasonable amount of time. In comparison, our quantum single hidden layer perceptron has a lower cost in term of complexity: $\mathcal{O}(NndL)$, where N and L represent observation and training epochs, and $n = \log(p)$ and $d = \log(H)$.

Importantly, the cost of state preparation and measurement must be taken into account when discussing the overall complexity of a qSLP: here the measurement operation is only applied to a restricted number of qubits (one in our case since we perform binary classification), reducing its cost to a multiplication constant that can be ignored. However, state preparation can greatly increase the complexity of a circuit, and when qSLP deals with big datasets, it needs an efficient encoding strategy, since even the ones adopted in this thesis scale linearly with respect to N and p.

Chapter 4

Experiments

The second part of the research contribution of this thesis is exposed in this chapter. We will present here the testing of the qSLP models defined in the previous chapter and the comparison of our model with the baseline in terms of accuracy on real-world benchmark datasets.

We implemented the models through the Qiskit library, a software framework for optimization and quantum computation. This library can be used for both quantum and hybrid computation, allowing the use of quantum objects like qubits and quantum gates, together with classical elements like variables and functions. One of the advantages of the models we implemented is the full compatibility with the standard library function that handles the training of supervised machine learning algorithms (like we mentioned in section 3.3).

We selected two multi-class datasets and divided them into five smaller datasets for binary classification, aiming to find the best parameters $\{\beta^\star, \theta^\star\}$ for each model. After the training, we compared the results with the baseline models proposed by the library and trained by us.

4.1 Dataset description

The simulation of a quantum system is a challenging task even for a small-sized model, for this reason, we utilized small samples of data from the dataset divided into train set (80%) and test set (20%). Furthermore, since the complexity of the simulation scales exponentially with the number of qubits utilized, the model utilizes a maximum of two qubits to encode the data, and the datasets' features are reduced by applying a principal component analysis (PCA) (Hotelling, 1936; Pearson, 1901). PCA is a linear transformation that shifts the data into a new coordinate system such that the new variables contain the greatest variance in the first coordinate, the second greatest in the second coordinate, and so on. The performance of PCA is measured by the explained variance of the new features; the total variance of the new variables is equal to the variance of the old ones and the performance is computed as the ratio of the variance of the principal components utilized, over the total variance.
The input data we use consists of the first two components computed by the PCA and the expected variance of each model will be reported in the following subsections. The data used in the experiment come from the benchmark datasets MNIST (Deng, 2012) and Iris (Anderson, 1936; Fisher, 1936).

4.1.1 MNIST dataset

The Modified National Institute of Standards and Technology database (MNIST) is a large dataset containing images of handwritten digits that is commonly used as a standard benchmark for machine learning techniques and image processing algorithms. The dataset contains a train set of 60000 records and a test set of 10000 records (Kussul and Baidyk, 2004). Each record consists of black and white 28×28 images classified in one of ten classes representing the digit in the image. Each image can be seen as a vector of binary 784 features (0 if the pixel is withe and 1 if it's

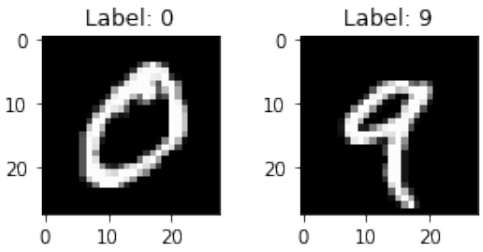
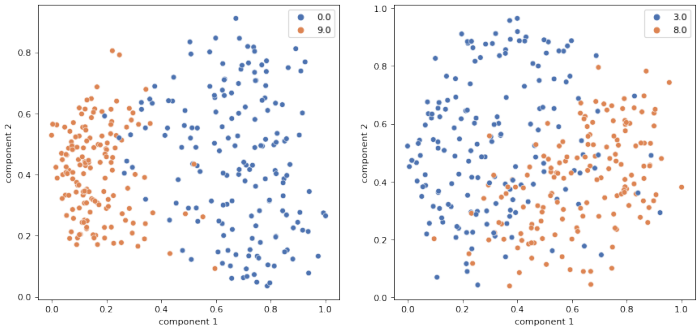


Figure 4.1: Image representation of the digits 0 and 9 in the MNIST dataset

black).

Since the current implementation of the models works for binary classification problems we consider only a subset of two different classes: classes $\{0, 9\}$ in one experiment and classes $\{3, 8\}$ in another (from now on we will refer to this two partial datasets as MNIST09 and MNIST38). Furthermore, since each model utilizes inputs with only two features, we compute more significant features through the use of PCA that changes the feature space, resulting in a new set of features with changed variance. From this new set of features, we take the two most representative components achieving a 31% explained variance in MNIST09 and 20% in MNIST38 (Figure 4.2).



(a) MNIST09 (b) MNIST38

Figure 4.2: Scatter plot of the first two principal component of MNIST09 4.2a and MNIST38 4.2b. The x axis represents the first component while the y the second. The first two components explain 31% of variance in MNIST09 and 20% in MNIST38.

For our training, we choose a set of 200 elements divided into train and test set with an 8 by 2 ratio.

4.1.2 Iris

The Iris flower dataset is a collection of data used to quantify the morphological variation of Iris flowers of three related species. The data consists of 50 examples for each species (Iris *setosa*, Iris *virginica* and Iris *versicolor*). For each record, the dataset contains four features representing the length and the width in centimeters of the flower's petals and sepals. The dataset is a standard benchmark for the test of classification and clustering algorithms.

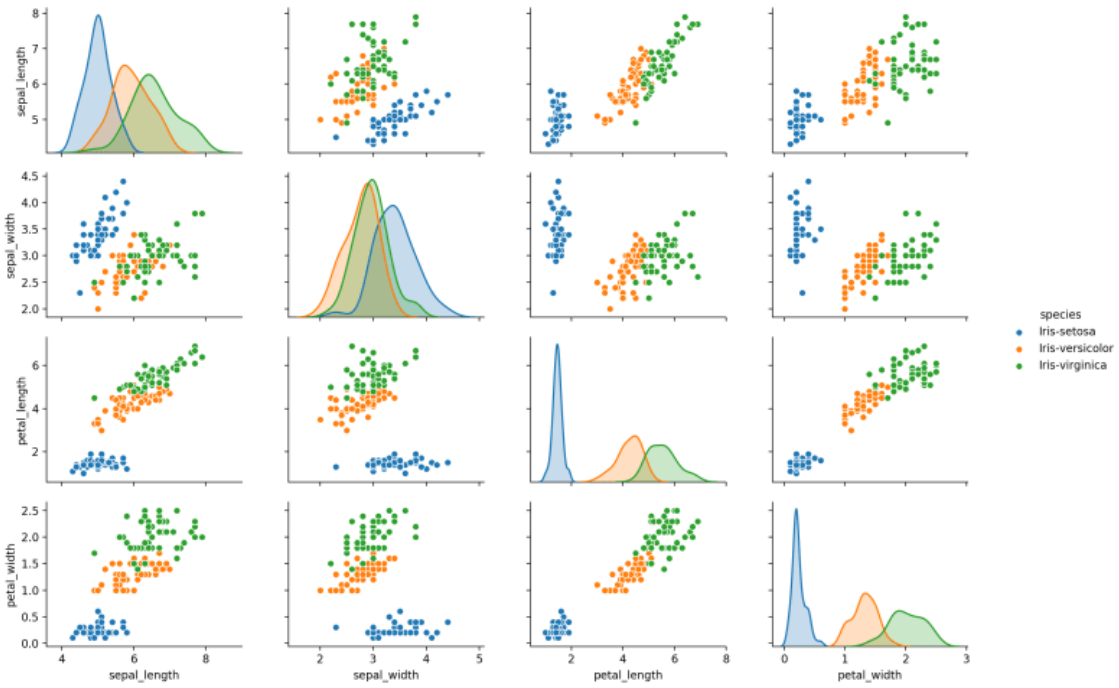
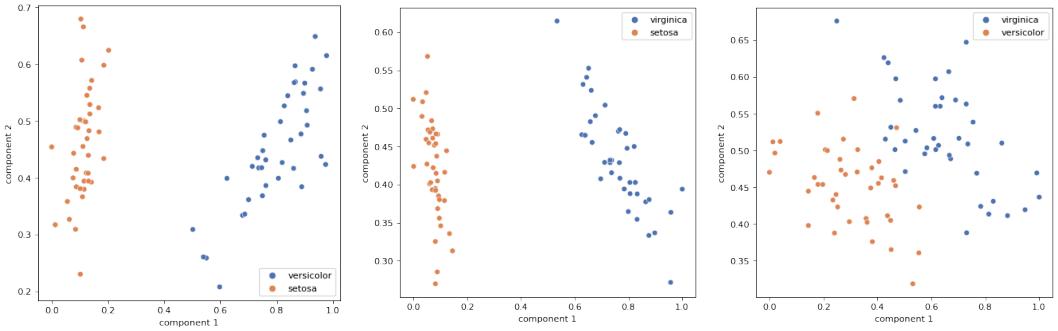


Figure 4.3: Original scatterplot matrix of the Iris dataset

Since the proposed models only supports binary classification we create three different datasets considering two classes at the time: *virginica-versicolor* (ViVe



(a) SeVe (b) SeVi (c) ViVe

Figure 4.4: Scatter plot of the first two component of SeVe 4.4a, SeVi 4.4b, and ViVe 4.4c. The x axis represents the first component while the y the second. They explain respectively the 98.0% (SeVe), 98.4% (SeVi), and 92.3% (ViVe) of the total variance.

dataset), *setosa-virginica* (SeVi dataset), and *setosa-versicolor* (SeVe dataset). Furthermore since the models have two-feature input we have to change the four features using PCA to another feature space. Each transformed dataset has different explained variance: 98.0% (SeVe), 98.4% (SeVi), and 92.3% (ViVe).
Lastly, It can be useful for future considerations to note that the *setosa* class, in the original feature space, is linearly separable from the other two and that this behavior is still present after the PCA (see figure 4.4a and 4.4b)

4.2 Models training

In our experiments, we considered nine different models based on the four architecture described in chapter 3. We choose to train one quantum support vector classifier (section 3.4.4), two quantum neural network classifiers (section 3.4), and six quantum single layer perceptron (section 3.3) of which three are single data qubits qSLP and three are padded qSLP. The different qSLP models are defined by the type

of qSLP architecture and the number of control qubits. In table 4.1 we show the specifics of each model reporting the number of qubits used for the data, the number of control qubits, and the depth of the circuit. With the term depth of a circuit, we indicate the longest path in a quantum circuit, also called critical path, that depends on the optimization level and the architecture of the processor it runs on. Often the gates utilized are not native to the processors the circuit runs on, and they need to be transpiled in a series of equivalent operations supported by the device. The depth of the circuits, both the original and transpiled in the simulator, are shown in the table.

Model	**n**	**d**	**Total qubits**	**depth**	**Transpiled depth**
QSVC	2	-	2	10	10
QNNC v1	2	-	2	13	13
QNNC v2	2	-	2	9	9
pad qSLP 1	2	1	3	52	37
pad qSLP 2	2	2	4	93	64
pad qSLP 3	2	3	5	134	91
sdq qSLP 1	1	1	2	13	12
sdq qSLP 2	1	2	3	23	22
sdq qSLP 3	1	3	4	33	31

Table 4.1: Summary of the specifics for the models we trained. Here d is the number of control qubits, n is the number of data qubits, *depth* and *Transpiled depth* are the length of the longest path of the circuit expressed in the number of gates respectively, for the circuit as we designed and for the circuit executed on the backend.

The training is done on a QASM simulator, a backend that emulates the execution of a quantum computer on a real device, including noise models that affect the

computation. Specifically, we utilized the *Aer Simulator*, a simulator belonging to the qiskit library that mimics an actual device sampling 1024 execution of the circuit to compute the final result.

When training the model we aim to find the parameter of the quantum circuit (β, θ). In the absence of the activation function Σ the final quantum state of the register $|\phi\rangle$ is represented as:

$$|\psi\rangle = \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} \beta_k |k\rangle |g(x, \theta_k)\rangle \tag{4.1}$$

Which is a linear transformation of the input data and defines a linear classifier. Notice that $Pr(y_i = 1|x_i)$ for a given observation x_i corresponds to the square of the linear transformation of hidden neurons with coefficients β_j.

The results of training on a simulated quantum computer can be seen in table 4.2.

	MNIST09	**MNIST38**	**SeVe**	**SeVi**	**ViVe**
QSVC	**.97**	.82	**1.0**	**1.0**	**.96**
QNNC v1	.86	.55	.93	**1.0**	.93
QNNC v2	.85	.80	.96	**1.0**	.89
pad qSLP 1	.89	**.86**	**1.0**	**1.0**	.83
pad qSLP 2	.91	.85	**1.0**	**1.0**	.83
pad qSLP 3	.92	.84	**1.0**	**1.0**	.83
sdq qSLP 1	.86	.79	**1.0**	**1.0**	.8
sdq qSLP 2	.87	.85	**1.0**	**1.0**	.8
sdq qSLP 3	.88	.84	**1.0**	**1.0**	.79

Table 4.2: Accuracy on the training set of each model for each dataset. The best results for each dataset are highlighted. The training is performed on a simulated quantum processor.

4.3 Results

Each model has been trained on all five datasets to better compare the performances of different architectures when applied in different scenarios. The trained models are then tested on an unseen set of records in both simulated and real quantum processors.

4.3.1 Tests on simulated hardware

The performances on the test set (reported in table 4.3) highlight the potential of the proposed models since they achieve high test accuracy in almost all datasets (94% on MNIST09, 82% on MNIST38 100% on SeVe and SeVi, and only 65% on ViVe). The accuracy is, in most cases, on par with the best performing baseline (QSVC) except for the ViVe dataset. One great achievement is the ability of the qSLP of reaching more than 80% of accuracy on the MNIST38 dataset, outperforming all baselines on the dataset with the smaller explained variance. In all experiments the test accuracy is in line with the training accuracy, hence the models are not affected by overfitting.

	MNIST09	MNIST38	SeVe	SeVi	ViVe
QSVC	**.94**	.8	**1.0**	**1.0**	**1.0**
QNNC v1	.86	.46	.85	**1.0**	**1.0**
QNNC v2	.83	.8	.95	.95	.8
pad qSLP 1	.93	**.82**	**1.0**	**1.0**	.65
pad qSLP 2	.89	.81	**1.0**	**1.0**	.65
pad qSLP 3	**.94**	.81	**1.0**	**1.0**	.6
sdq qSLP 1	.81	.81	**1.0**	**1.0**	.65
sdq qSLP 2	.83	.76	**1.0**	**1.0**	.65
sdq qSLP 3	.83	.78	**1.0**	**1.0**	.6

Table 4.3: Accuracy on the test set for each model for each dataset, performed with a simulated quantum processor. The best results for each dataset are highlighted.

4.3.2 Quantum processors

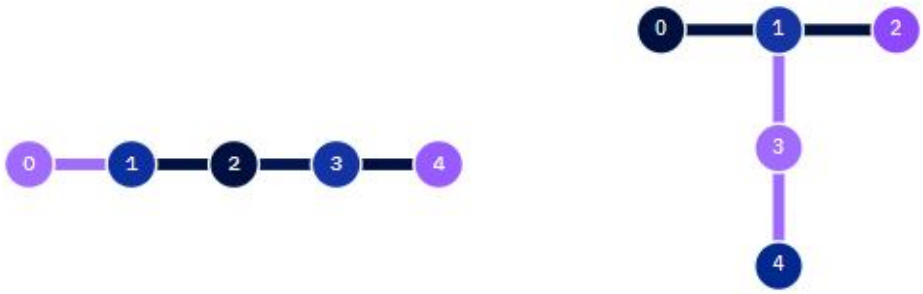
To better understand the power of our models we also included a series of tests performed on real quantum devices. Qiskit allows for the use of 24 different quantum processors with registers of different sizes and different specifics, named after the city they are placed in.

Processors belong to certain families depending on the size and the scale of the circuit possible on the chip. This is primarily determined by the number of qubits and the connectivity graphs, dividing them in four groups: *Eagle*, *Hummingbird*, *Falcon*, and *Canary*. The *Eagle* family has one processor with 127 qubits and more scalable packaging technologies than the other families. The *Hummingbird* family has a hexagonal qubit layout and up to 65 qubits depending on the specific processor. The *Falcon* are platforms for medium-scale circuits, valuable for demonstrating performance and

scalability improvements. Lastly, the processors in the *Canary* family are constituted by an optimized 2D lattice layout with up to 16 qubits.

Each processor has also different specifications regarding the quantum volume, a measurement of the performance of gate-base quantum computer regardless of the underlying technology, and circuit layer operation per flops (CLOPS) (Wack et al., 2021) that measures the speed of the device. Such values can vary a lot based on the processor's family and versions giving us an idea of their computation's quality and speed.

One key difference in the processor is the different graph connections of the qubits, if qubits are directly connected, they can interact easily, and the computation might be subjected to fewer errors. In figure 4.5 we show the two possible structures of a *Falcon* processors as examples of different structures. The specification for all the devices can be found in the official documentation at IBMQuantum (2021)



(a) (b)

Figure 4.5: The figures show the graph view of the processors at *ibm_bogota* 4.5a and *ibm_lima* 4.5b. Both are part of the *Falcon* family of IBM quantum processors

Source: IBMQuantum (2021)

4.3.3 Tests on real devices

Lastly, we tested our models on the real devices that IBM lends to the general public. Many of the devices mentioned in the previous section have restricted access and can be utilized only by members of the IBM quantum network. For this reason, we only used processors from the *Falcon* family with five qubits and with a QV up to 32, since they can be accessed by everyone. Each test chooses the device to use depending on their availability from the list presented in table 4.4.

As we can see in Table 4.5, the accuracy of every model on the test set decreases on

Device	**Qubits**	**QV**	**CLOPS**
ibmq_manila	5	32	2.8K
ibmq_bogota	5	32	2.3K
ibmq_quito	5	16	2.5K
ibmq_belem	5	16	2.5K
ibmq_lima	5	8	2.7K

Table 4.4: List and specifics of the real devices on which the models could be tested. The choice was made at test time considering the availability of each device

average with respect to the performance on a simulated environment. This is to be expected since real devices are subjected to external noise and are way less reliable than simulations. Unfortunately QVSC tests on MINST09 and MNIST38 couldn't be performed on a real devicedue to computational limitation.

An interesting phenomenon is that for the ViVe dataset we see a downward trend in the accuracy when increasing the controls qubit in the padded qSLP, and at the same time, on average the padded qSLP performs much worst than single data qubit qSLP or the other benchmark models in all dataset. Consulting table 4.1 we can observe an inverse correlation between the depth of the circuit and the accuracy, and we see that

single data qubits qSLP has a depth number roughly three time smaller than padded qSLP when transpiled in the simulator[1]. This trend highlights how much the effect of noise in the system increases when the number of gates in the computation increases, since the accuracy tends to deteriorate more when more qubits, and consequently more gates, are present in the tested circuit.

	MNIST09	**MNIST38**	**SeVe**	**SeVi**	**ViVe**
QSVC	-	-	1.0	1.0	.95
QNNC v1	.81	.51	.95	1.0	.95
QNNC v2	.85	.83	1.0	1.0	.89
pad qSLP1	.68	.51	.60	.50	.60
pad qSLP2	.63	.69	.65	.70	.55
pad qSLP3	.51	.51	.50	.60	.5
sdq qSLP1	.83	.79	1.0	1.0	.55
sdq qSLP2	.84	.78	1.0	1.0	.70
sdq qSLP3	.76	.80	1.0	1.0	.65

Table 4.5: Accuracy of the models tested on a real device. We can observe a negative correlation between the complexity of the tested circuit and the accuracy. This is likely generated by the noise affecting the quantum processor that has a higher impact when a high number of gates is involved.

From these experiments, we can also conclude that with the current technology, the single data qubit sQLP is a more reliable choice when running on a real device, because, even if it has slightly worse performance on the simulation compared with the padded version, in a real processor the smaller depth produces less noise derived

[1]Even if the depth depends on the chosen quantum processor, we can safely assume the depth of padded qSLP to be much greater of the one of sdq qSLP

errors, preserving the simulated accuracy.

4.4 Future works

In this thesis, we showed how our qSLP, in both versions of the model, has the capability of becoming one of the standards for quantum machine learning, since it can reproduce and even surpass in terms of accuracy the benchmarks proposed in the literature. However, this ability is highly dependent on two factors: the implementation of an efficient routine that emulates non-linear activation functions and the future computational power of quantum hardware.

The true strength of this model is in the intrinsic ability of the implementation to produce the effect that n neurons have in a classical model with just $\log(n)$ steps. This potentially could allow us to exploit the universal approximation theorem, but we still don't have enough quantum computational power to test it. Since the work of this thesis was limited to a test on five qubits hardware a natural follow up would be to test the models on better processors increasing the number of control qubits d. Concerning the other current limitation, as we already discussed in subsection 3.2.3, we still do not have an efficient way to emulate non-linear functions and we are limited by the linearity constraint of quantum mechanics. One possible direction for future studies could try to take advantage of the modular architecture of qSLP and change it by switching the proposed state preparation circuit in favor of one that performs feature mapping since it has been proven by Goto et al. (2020) that machine learning models induced from the quantum-enhanced feature space are universal approximators of continuous functions.

As technology progresses the number and the computational power of quantum processors increases, and we could experiment with different, bigger, and more complex datasets. An increase in both the number of observations, that today imposes a limit

on the model's training, and in the number of features used to make the prediction, could allow us to increase our performance since they could provide more comprehensive training and more feature to guide the classification. Another direct follow-up in this direction could be experimenting on different classes of problems, including, but not limited to, multi-class classification and regression, that are natively supported by the MAQA framework. As mentioned before, our testing was limited by the computational power of the quantum processors available to us *Falcon* family of processors, but more complex experiments could be done by someone with access to more powerful machines from other families (*Hummingbird*, *Eagle*)

One last subject that was not discussed in this thesis is the possibility to change the gates of qSLP's ansatz from a cU_3 to other parametrized gate to alter the function $g(x, \cdot)$. This operation is supported by the proposed implementation and can be easily tested.

To summarize, our model is very promising but the current technological and theoretical state of the art doesn't allow us to properly conduct a complete evaluation of its potential. On top of that our model focuses only on binary classification, which is one of many possible applications of the MAQA framework, and proposed specific, even if modular, implementations in both the state preparation and ansatz circuits. Here we produced the first-ever implementation of a quantum variational algorithm for a single layer perceptron and we demonstrated the potentiality of this system. Still, many questions remain open, and with them, many opportunities for further improvement.

Chapter 5

Conclusions

Quantum machine learning (qML) is a very promising branch of quantum computation, where rich possibilities are offered by just a small amount of qubits. However, the computational advantages over classical machine learning have yet to be proven, and qML techniques are limited by the state preparation phase and by the application of an arbitrary number of gates.

In this thesis, we gave an overview of the MAQA framework and we showed how it can be utilized to implement a quantum single layer perceptron (qSLP). The key idea behind the proposal is to take advantage of properties of quantum computation like entanglement, superposition, and interference, to create a more efficient version of the classical algorithm. We propose an implementation of qSLP as an aggregator of multiple functions $g(x, \cdot)$ where each function propagates the input on a different quantum trajectory, all of which are in superposition with each other. The speedup provided by our model derives from the ability to create an exponentially large number of neurons in a linear number of steps and from the possibility of propagating a function to each quantum state by applying it only once. This allows building a model with an incredible descriptive power capable of being a universal approximator

if equipped with a proper activation function.

We presented the implementation for a general model with a variable number of control qubits, that allows great flexibility when dealing with the small size of modern quantum hardware. Indeed, our qSLP can be created with a number of qubits as low as two. Furthermore, the resulting quantum circuit is modular and it gives the possibility to change the ansatz's parametric gates, the activation function and the state preparation circuit, making it able to exploit more advanced techniques as soon as they are found.

We provided two implementations of the algorithm that differ in the size of the data register and the formulation of the ansatz and the state preparation. Both encode the classical input with an amplitude encoding technique, but the first one utilizes only one qubit to store the data while the second uses a two qubits register. The second step, the ansatz, entangles the control register with the data register through the use of controlled not and controlled parametric rotations, to achieve the wanted superposition of multiple functions. An activation function F is finally applied on the data qubit and through quantum interference, it is propagated to all trajectories in superposition. Here F only contributes to the overall time complexity in an additive manner while the same operation in a classical setting would affect complexity in a multiplicative way.

Following the original design of MAQA, the qSLP is treated as a quantum variational algorithm, meaning that the results are measured from the circuit, and then the parameters are updated classically. In our implementation the measurement, that is usually a very expensive operation, reveals one of the advantages of the model, since, to obtain the results we only have to perform it on a small subset of the qubits, greatly reducing the complexity.

The model we presented has been implemented utilizing the qiskit library for quantum computing. It is a flexible and highly customizable model. Different state prepara-

tion circuits besides the one presented can be plugged in allowing for more exhaustive future research. It can change the parametric gates in the ansatz and it is also compatible with the standard functions since it was designed to be consistent with the qiskit library.

Alongside our model, we presented and implement other architectures already existing in the literature: two quantum neural network classifiers and one quantum support vector classifier. These architecture are then utilized as the baseline for our tests.

Lastly, one of this thesis contributions is the testing of our model on the real-world benchmark datasets MNIST and Iris, to perform binary classification. The experiments demonstrated the ability of the model to correctly classify linearly separable observations while highlighting the need for a non-linear activation function to improve performance on discerning overlapping records. Furthermore, when the performance on a test set is compared to the baseline, our architecture stands at the same level as today's standards and outperforms them in some situations. It is also worth mentioning that, as expected, the model performance in the simulation increases when more control qubits are used (hence more nodes in the hidden layer), since, theoretically, thanks to the universal approximation theorem, a single layer perceptron can approximate any continuous bounded function, given enough nodes.

The main challenge for the near future is to tackle the problem of designing routines to emulate non-linear functions in a quantum scenario, or to find alternatives to surpass what is today the biggest obstacle for a complete quantum neural network as a quantum variational algorithm. Furthermore future works might include the test with more complex or more populated dataset, changes in the structure of the state preparation function, and changes in the parametric gates of the ansatz.

To conclude, quantum machine learning, much like quantum computing, is still at its beginning stage. We are moving our first step towards new technologies and we still understand very little of how they can bring us advantages. Nonetheless, many

researches suggest that quantum computing could change the way we think about problems like machine learning, giving us benefits we would have never expected.

Bibliography

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

Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., Sleator, T., Smolin, J., and Weinfurter, H. (1995). Elementary gates for quantum computation. *Physical Review A*.

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

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

Benioff, P. (1980). The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22(5):563–591.

Bennett, C. H. (1973). Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532.

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. (2016). Quantum machine learning. *Nature*.

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

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

Einstein, A. (1987). *The Collected Papers of Albert Einstein, Volume 1 (English)*. Princeton University Press.

Farhi, E., Goldstone, J., Gutmann, S., and Neven, H. (2017). Quantum algorithms for fixed qubit architectures.

Farhi, E. and Neven, H. (2018). Classification with quantum neural networks on near term processors.

Feynman, R. P. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488.

Fisher, R. A. (1936). The use of multiple measurement in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

Giovannetti, V., Lloyd, S., and Maccone, L. (2007). Quantum random access memory. *Physical Review Letters*.

Goto, T., Tran, Q. H., and Nakajima, K. (2020). Universal approximation property of quantum machine learning models in quantum-enhanced feature spaces. *Physical Review Letters*.

Gupta, S. and Zia, R. K. P. (2002). Quantum neural networks.

Havlicek, V., Córcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., and Gambetta, J. M. (2018). Supervised learning with quantum enhanced feature spaces. *Nature*.

Heisenberg, W. (1925). Quantum-theoretical re-interpretation of kinematic and mechanical relations. *Z. Phys*, 33:879–893.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.

Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, 28(3/4):321.

Hu, W. (2018). Towards a real quantum neuron. *Natural Science*, 10(03):99–109.

IBM (2022). Ibm's quantum computing systems https://www.ibm.com/quantum-computing/systems/. Online.

IBMQiskit (2022). https://qiskit.org/. Online.

IBMQuantum (2021). https://quantum-computing.ibm.com/. Online.

J. Faber, G. G. (2002). Quantum models for artificial neural network.

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*.

Kussul, E. and Baidyk, T. (2004). Improved method of handwritten digit recognition tested on MNIST database. *Image and Vision Computing*, 22(12):971–981.

Macaluso, A. (2021). *A novel framework for quantum machine learning*. PhD thesis, Alma Mater Studiorum - Università di Bologna.

Macaluso, A., Clissa, L., Lodi, S., and Sartori, C. (2020a). 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. (2020b). A variational algorithm for quantum neural networks. In *Lecture Notes in Computer Science*, pages 591–604. Springer International Publishing.

Maria Schuld, F. P. (2018). *Supervised Learning with Quantum Computers*. Springer International Publishing.

McClean, J. R., Boixo, S., Smelyanskiy, V. N., Babbush, R., and Neven, H. (2018). Barren plateaus in quantum neural network training landscapes. *Nature Communications*.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Michael A. Nielsen, I. L. C. (2010). *Quantum Computation and Quantum Information*. Cambridge University Pr.

Mitarai, K., Negoro, M., Kitagawa, M., and Fujii, K. (2018). Quantum circuit learning.

Moll, N., Barkoutsos, P., Bishop, L. S., Chow, J. M., Cross, A., Egger, D. J., Filipp, S., Fuhrer, A., Gambetta, J. M., Ganzhorn, M., Kandala, A., Mezzacapo, A., Müller, P., Riess, W., Salis, G., Smolin, J., Tavernelli, I., and Temme, K. (2017). Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*.

Mottonen, M., Vartiainen, J. J., Bergholm, V., and Salomaa, M. M. (2004). Transformation of quantum states using uniformly controlled rotations. *Quantum Information and Computation*.

Park, J. L. (1970). The concept of transition in quantum mechanics. *Foundations of Physics*, 1(1):23–33.

Pearson, K. (1901). LIII. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

Poincaré, H., Lamotte, M., and Hurmuzescu, D. (1892). *Théorie mathématique de la lumière*. Cours de la Faculté des Sciences de Paris. Cours de physique mathématique. Georges Carré.

Powell, M. (2007). A view of algorithms for optimization without derivatives. *Mathematics TODAY*, 43.

Powell, M. J. D. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis*, pages 51–67. Springer Netherlands.

Powell, M. J. D. (1998). Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336.

Rebentrost, P., Mohseni, M., and Lloyd, S. (2013). Quantum support vector machine for big data classification. *Physical Review Letters*.

Ristè, D., da Silva, M. P., Ryan, C. A., Cross, A. W., Smolin, J. A., Gambetta, J. M., Chow, J. M., and Johnson, B. R. (2015). Demonstration of quantum advantage in machine learning. *npj Quantum Information*.

Schrödinger, E. (1926). Quantisierung als eigenwertproblem. *Annalen der Physik*, 384(4):361–376.

Schuld, M., Bocharov, A., Svore, K., and Wiebe, N. (2018). Circuit-centric quantum classifiers. *Physical Review A*.

Schuld, M. and Killoran, N. (2018). Quantum machine learning in feature hilbert spaces. *Physical Review Letters*.

Schuld, M., Sinayskiy, I., and Petruccione, F. (2014a). The quest for a quantum neural network. *Quantum Information Processing*.

Schuld, M., Sinayskiy, I., and Petruccione, F. (2014b). Simulating a perceptron on a quantum computer. *Physics Letters A*.

Schützhold, R. (2002). Pattern recognition on a quantum computer. *Physical Review A*.

Shende, V. V., Bullock, S. S., and Markov, I. L. (2004). Synthesis of quantum logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509.

Stephen Boyd, L. V. (2004). *Convex Optimization*. Cambridge University Press.

Tacchino, F., Macchiavello, C., Gerace, D., and Bajoni, D. (2018). An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*.

Temkin, M. (2021). Investors bet on the technologically unproven field of quantum computing.

Toth, G., Lent, C. S., Tougaw, P., 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.

Wack, A., Paik, H., Javadi-Abhari, A., Jurcevic, P., Faro, I., Gambetta, J. M., and Johnson, B. R. (2021). Quality, speed, and scale: three key attributes to measure the performance of near-term quantum computers.

Wecker, D., Hastings, M. B., and Troyer, M. (2015). Towards practical quantum variational algorithms. *Physical Review A*.