UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

A mathematical framework for quantum neural networks

Autor: Núria Urgell Ollé

Director: Dr. Bruno Juliá Díaz

Realitzat a: Departament Quantum physics and astrophysics

Barcelona, January 23, 2023

Abstract

This thesis focuses on dissipative quantum neural networks, a subfield of quantum machine learning. Classical neural networks and the fundamental concepts of quantum mechanics, crucial for understanding quantum machine learning, are introduced from a mathematical perspective. We exhibit the parallelism between the training algorithms of classical neural networks and dissipative quantum neural networks and establish a mathematical framework to describe classical and quantum neural networks.

Resum

Aquesta tesi se centra en les xarxes neuronals quàntiques dissipatives, un subcamp de l'aprenentatge automàtic quàntic. Les xarxes neuronals clàssiques i els conceptes clau de la mecànica quàntica, crucials per entendre les xarxes neuronals quàntiques, són introduïts des d'una perspectiva matemàtica. Exhibim també el paral·lelisme entre els algorismes d'entrenament de les xarxes neuronals clàssiques i quàntiques i establim un marc matemàtic per descriure-les.

2020 Mathematics Subject Classification. 68Q12, 81P16, 68T07, 81P16, 81P40, 81P45, 81P68.

Acknowledgements

I want to thank my supervisor Prof. Dr. Bruno Juliá, for his guidance and encouragement throughout this process. His weekly feedback and supervision have greatly improved the quality of this work. I would also like to thank Dr. Kerstin Beer, the leading developer of dissipative quantum neural networks, for her availability to answer any questions.

Contents

0.1 Introduction

Have you ever wondered how a quantum computer operates? Analogously to classical bits in computers, quantum computers make use of the so-called *qubits*. A qubit is a two-level system represented by $|0\rangle$ and $|1\rangle$. These qubits give quantum computers interesting and powerful features. Pure qubits are a *coherent superposition* of the basis states. Meaning that we can describe the pure qubit as

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

This, alongside other properties of qubits, such as the so-called *entanglement*, allows for a parallelization of the calculations performed by the quantum computer [1, 2]. When two qubits are entangled, a single operation can be performed on those two qubits simultaneously, which doubles the computational power of the system. Such parallelization of the calculations is not possible for classical computers.

There are various models for implementing quantum computers, such as the adiabatic model, the gate model, and measurement-based quantum computing [3]. The most widely used approach, and the one focused on in this thesis, is the *gate model* of quantum computers. The gate model is based on quantum gates, which are analogous to logic gates. Quantum gates are described as $2^n \times 2^n$ unitary matrices acting on n qubits [4]. In this model, algorithms are performed by arranging a sequence of quantum gates in a particular order.

Quantum computing has rapidly evolved. The early work in quantum computing during the 1980s was purely theoretical. The first quantum algorithms were developed at the time. During the 1990s, scientists developed the first experimental quantum gates, and in the early 2000s, the first small-scale quantum computers were built. Recently, there has been an increased interest in the field [5]. Companies like IBM are building some of the most advanced quantum computers to date. IBM has currently built the largest quantum processor, composed of 433 functional qubits [6]. Despite the progress made, quantum computing is still in its early stages, and there are limitations on the number of gates and qubits that can be used. However, the potential for quantum computing to expand the limits of what can be calculated and revolutionize computation has led to continued research and development in the field.

On the other hand, *machine learning* is a field that is present in our lives in many ways and has significantly impacted the use of data. Machine learning algorithms enable the implementation of tasks such as decision-making, pattern recognition, and prediction [7]. *Neural networks*, which are a subfield of machine learning, are the focus of this thesis. They can be thought of as a network of neurons, each applying a parametrized function to its input. The parameters are then optimized so that the network's final output approximates the desired output. To build an effective neural network, the algorithm must "learn" by being run numerous times with different *training data* [8, 9].

Quantum machine learning is a recent field of study that aims to combine quantum science and machine learning. There are several approaches to achieving this goal, including using classical machine learning algorithms to process quantum data, quantum-enhanced machine learning, and quantum algorithms that resemble classical machine learning algorithms [10, 11, 12]. In this thesis, we concentrate on quantum neural networks, a combination of quantum computing and artificial neural networks. The building blocks of quantum neural networks are the quantum perceptrons composed of a set of qubits. It is important to note that there is no unified approach to building quantum neural networks. For example, some models are based on the use of quantum systems to represent the probability distribution of a classical neural network [13]. In this work, we focus on the mathematical formulation of the recently developed *dissipative quantum neural networks* introduced by K. Beer in 2020 [14]. These networks have an architecture similar to that of classical neural networks, but with the equivalent quantum neurons and algorithm implementations [15]. Quantum machine learning is a rapidly growing field of interest being explored by leading companies in the field of quantum computing.

Mathematics plays a significant role in this field of study, but there is a lack of literature that provides a unified mathematical approach. To my knowledge, no reviews cover the mathematical basis of classical and quantum machine learning, and most reviews are oriented from a physics perspective. This thesis aims to provide a mathematical framework for both classical neural networks and dissipative quantum neural networks. It presents a mathematical-oriented approach to the field, which is easily accessible to those with a background in mathematics.

Let us summarize the structure followed in this thesis. **Chapter 1** provides a mathematical approach to classical neural networks, including their structure and different types of artificial neurons, as well as an introduction to the training problem and algorithms. **Chapter 2** introduces the foundations of quantum computing and the mathematical definitions necessary for understanding Chapter 3. We review key concepts such as the definition of quantum states, the gate model and the universality theorems, and the concept of fidelity of quantum states, among others. **Chapter 3** focuses on the review of dissipative quantum neural networks, including the introduction of quantum neurons and the network's structure, the training problem and algorithm, and the implementation of these structures on current quantum computers. Finally, **appendix A** reviews the study's results on the impact of the learning rate on the accuracy of neural networks for handwritten digit recognition.

Chapter 1

Neural networks and deep learning

Machine learning aims to use data to get computers to develop specific tasks. Neural networks are a subset of machine learning and were first introduced in the mid-20^{th} century as a model for simulating the behavior of biological neurons computationally [16]. Machine learning is a fast-growing field with numerous applications. This growth is linked to the increase of data collected nowadays, needed to increase the efficiency of such structures, and the growing interest in the field.

Neural networks can be thought of as abstract mathematical concepts. This chapter illustrates this based on the definitions provided in [7, 8, 9], and highlights the structural similarities between classical and dissipative quantum neural networks presented in Chapter 3. First, we introduce artificial neurons and give a definition for neural networks. We then present the training algorithm and review the training problem, the feed-forward algorithm, backpropagation, and the network's updating procedure.

1.1 Artificial neurons

The so-called artificial neurons are the building blocks of neural networks. Such structures transform a set of inputs into a given output.

Definition 1.1. An *artificial neuron* is a quadruple $(\mathbf{x}, \mathbf{w}, \varphi, y)$, with $\mathbf{x}^T = (x_0, x_1, \ldots, x_n) \in \mathbb{R}^{n+1}$, $x_0 = -1$, and with input values $x_1, \ldots, x_n$, $\mathbf{w}^T = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ is the so-called weights vector with bias $w_0 = b$, φ is the activation function, and the output function $y : \mathbb{R} \to \mathbb{R}$ is defined as $y = \varphi\left(x^T w\right)$.

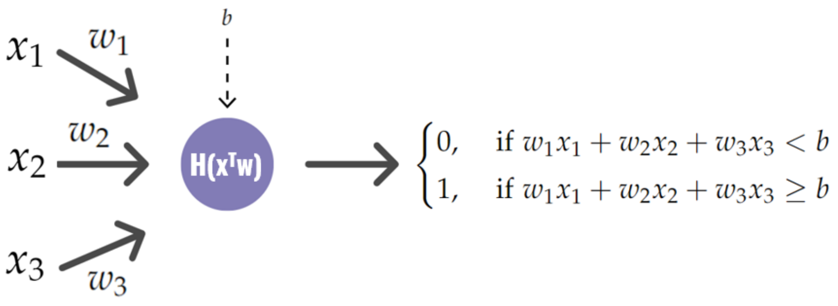

x_1
w_1
b
x_2
w_2
H(x^T w)
x_3
w_3
$\begin{cases} 0, & \text{if } w_1x_1 + w_2x_2 + w_3x_3 < b \\ 1, & \text{if } w_1x_1 + w_2x_2 + w_3x_3 \geq b \end{cases}$

Figure 1.1: Schematic representation of a perceptron with three inputs x_1, x_2, x_3; bias b, and weights w_1, w_2, w_3.

Remark 1.2. Note that the output function is $y = \varphi\left(x^T w\right) = \varphi\left(\sum_{i=0}^{n} w_i x_i\right) = \varphi(-b + x_1 w_1 + \ldots + x_n w_n)$.

1.1.1 Perceptrons

A perceptron neuron is an artificial neuron with the following properties

- $x_i \in \{0, 1\}$, $i \in \{1, \ldots, n\}$.
- $\varphi(x) = H(x)$, where $H(x)$ is the Heaviside function.

Recall that

$$H(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}.$$

Now, the output function of a perceptron neuron

$$y = \varphi\left(x^T w\right) = \begin{cases} 0, & \text{if } \sum_{i=1}^{n} w_i x_i < b \\ 1, & \text{if } \sum_{i=1}^{n} w_i x_i \geq b \end{cases}.$$

A comprehensive approach to visualize the action of a perceptron is to represent it as shown in figure 1.1. This approach will be useful in section 1.2 when we represent networks of neurons.

Perceptrons can be interpreted as devices that perform decision-making by weighing up the evidence. Suppose we want to implement a decision between choice 0 and choice 1. Then, the weights can be thought of as how much each factor x_i contributes to making a decision 0 or 1 given a threshold b.

Example 1.3. Suppose we want to decide whether or not to join a bachelor in mathematics at UB. Zero represents choosing not to join, and one represents joining the bachelor. Then, we weigh up different factors that influence our decision-making. We might be influenced by a set of factors: the closeness of UB to our house, whether we like the coursework, and the alumni reviews. For instance, we will have 0 if UB is far from our house and 1 if it is close. On the other hand, if we like the coursework, we input a 1, and if we dislike it, we input a 0. Lastly, if the reviews of alumni are reasonable, we input a 1, and if they are bad we input a 0. Then, it is time to decide how important each factor is. For example, the proximity is not very important to us, the alumni reviews are somehow important, and the coursework is critical. Therefore, we choose $w_1 = 1$, $w_2 = 4$, $w_3 = 6$. Finally, we choose a bias b=7. Now the perceptron implements the model outputting 1 or 0 depending on the different factors and their importance. By changing the weights and biases, we can get different decision-making devices.

1.1.2 The sigmoid neuron

A perceptron's output is a discontinuous binary function. If we interpret perceptrons as decision-making devices, an improved model should give an output $y = \varphi\left(x^T w\right) \in (0,1)$, such that outputs close to 0 describe a small likelihood of occurrence and outputs close to 1 describe a greater likelihood to occur. We define an artificial neuron following such properties: the sigmoid neuron.

A sigmoid neuron is an artificial neuron with an input vector $x^T = (x_0, \dots, x_n)$, weights and bias $w^T = (b, w_1, \dots, w_n)$, and with the logistic function,

$$\sigma(x) = \frac{1}{1+e^{-x}},$$

as the activation function. The output function for a sigmoid neuron is given by:

$$y = \sigma(x^T w) = \frac{1}{1+\exp\left(-\sum_{j=1}^{n} w_j x_j + b\right)}.$$

1.2 The structure of neural networks

In this section, we consider sets of connected artificial neurons with the same activation function, the so-called neural networks represented in figure 1.3. Let us first introduce the notation we will use. We refer to each group of perceptrons applied on the same input values as layers. As shown in figure 1.3, there are three types of layers: the input layer, which is the first layer of the network containing the initial inputs; the output layer, which is the last layer of a network, and the hidden layers located between the input and output layers. We use w_{jk}^{l} to refer

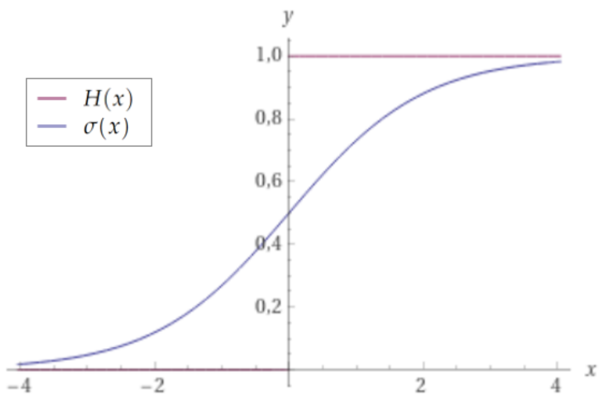


Figure 1.2: Plot of the Heaviside function, $H(x)$, and sigmoid function, $\sigma(x)$.

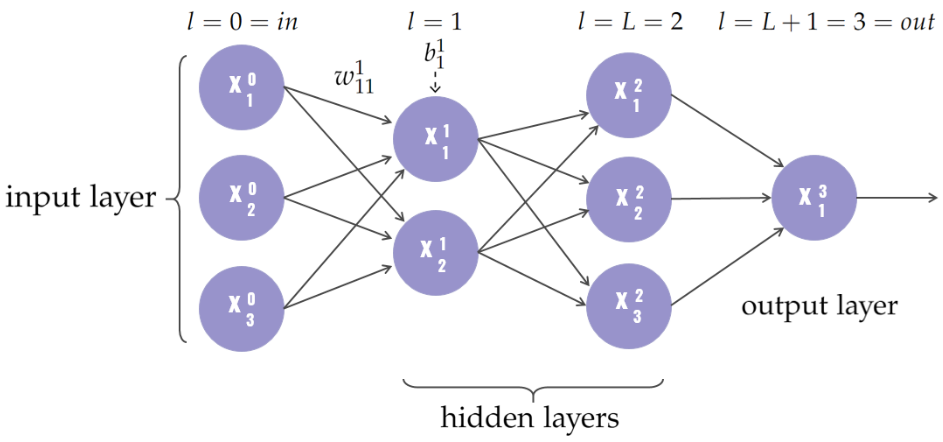


Figure 1.3: Schematic representation of a 4-layer neural network.

to the weight coupling the k^{th} neuron in the $(l-1)^{th}$ layer with the j^{th} neuron in the l^{th} layer. Additionally, the bias of the neuron located at position j in layer l is represented by the symbol b^l_j, and the output function of the j^{th} neuron in the l^{th} layer is represented as x^l_j.

Definition 1.4. A *fully connected feed forward neural network* is given by its architecture $a = (m, \varphi)$, where $m \in \mathbb{N}^{L+2}$ for $L \in \mathbb{N}$, and φ is the activation function. L is the number of hidden layers in the network, and m_l for $l \in \{0, \dots, L+1\}$ is the number of neurons in the l^{th} layer.

We will use the term neural network to refer to fully connected feed-forward neural networks.

1.3 The training algorithm

1.3.1 The training problem

Neural networks improve their behavior via the learning process, where the network is trained to solve a specific problem. Let us give a set of definitions to understand this process further.

Definition 1.5. Let $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ be measurable spaces, and $\mathcal{M}(\mathcal{X}, \mathcal{Y})$ the set of measurable functions from $\mathcal{X}$ to $\mathcal{Y}$. $\mathcal{X}$ is denoted as the input space, $\mathcal{Y}$ as the output space, and $\mathcal{Z}$ as the data space. Given $\mathcal{C} : \mathcal{M}(\mathcal{X}, \mathcal{Y}) \times \mathcal{Z} \to \mathbb{R}$ the so-called cost function, and the training data $(z^j)_{j=1}^n \in \mathcal{Z}$. The *learning process* implies finding the network's output function $f \in \mathcal{M}(\mathcal{X}, \mathcal{Y})$ which minimizes the cost function $\mathcal{C}(f, z)$.

In this work, we focus on supervised learning and assume $\mathcal{X}$ and $\mathcal{Y}$ are Euclidean. For simplicity, we take $\mathcal{X} \subset \mathbb{R}^d$ with $d \in \mathbb{N}$, and consider $\mathcal{Y} \subset \mathbb{R}^m$ with $m \in \mathbb{N}$.

Definition 1.6. A *training data set* is $s = ((x_j^0, y^j))_{j=1}^{m_0} \in \mathcal{X} \times \mathcal{Y} = \mathcal{Z}$, where $x_j^0 \in \mathcal{X}$ are the inputs of the network with their associated labels $y^j \in \mathcal{Y}$. The training data set is used to train the network, i.e., it is used to minimize the cost function.

Example 1.7. A well-known problem solved through the use of neural networks is handwritten digit recognition. In such a problem, the training data consists of vectorized images of handwritten digits and associated labels indicating the written number in the image taken from the MNIST data set. This problem is presented and studied in Appendix A.

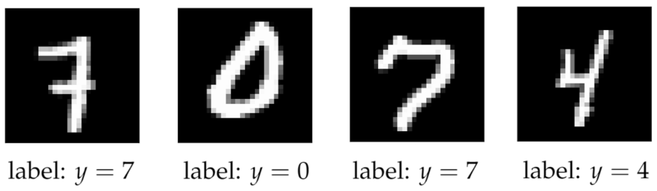


Figure 1.4: MNIST images of handwritten digits.

Aside from training data, there are two more types of data: validation and test data. Validation and test data sets consist of data pairs different from each other and different from the training data pairs. Those data sets are used to compute the accuracy of the network. Given a labeling problem, such as the problem of example 1.7, we define the accuracy of the network given a set of data pairs as the percentage of correctly labeled inputs. The resulting accuracy for the training data is used to assess the network's behavior and tune the network's parameters. On the other hand, the accuracy of test data is used after training to confirm that the algorithm is trained effectively.

Definition 1.8. A *prediction task* entails finding $f : \mathcal{X} \to \mathcal{Y}$ s.t., for unseen data $(x_j^0, y^j) \in \mathcal{X} \times \mathcal{Y}$, $f(x_j^0) \approx y^j$.

Remark 1.9. f is the output function of the network and thus can be expressed as a function of the weights and biases, as shown in section 1.3.2. The aim is to modify the weights and biases to minimize the cost function.

A widely used cost function in machine learning is the mean square error (MSE) function, also known as the quadratic cost function. It is defined as

$$\mathcal{C}(w,b) = \frac{1}{2m_0}\sum_{j=1}^{m_0}(f(x_j^0) - y^j)^2,$$

where w denotes the collection of all weights, and b all the biases.

1.3.2 Feed-forward

The first step in training a neural network is to compute the network's output given a certain set of inputs. Here, we derive the expression for the output of an arbitrary neural network. First, note that output of the j^{th} neuron of the l^{th} layer is

$$x_j^l = \varphi(\sum_{k=1}^{m_{l-1}} w_{jk}^l x_k^{l-1} - b_j^l).$$

Now, we can rewrite the expression above in matrix form as

$$X^l = \varphi(W^{l^T} X^{l-1} - B^l),$$

where

$$X^l = \left(x_1^l, \dots, x_{m_l}^l\right)^T, \quad W^l = \left(w_{jk}^l\right)_{j,k}, \quad B^l = \left(b_1^l, \dots, b_{m_l}^l\right)^T.$$

Remark 1.10. Assume we have a neural network with a linear activation function, $a = (m, \varphi = nx + p)$. Now, the output of the network

$$X^{\text{out}} = \varphi(W^{l^T} X^{l-1} - B^l) = \left(n \cdot w_{jk}^l\right)_{j,k}^T \left(x_1^{l-1}, \dots, x_{m_{l-1}}^{l-1}\right)^T - \left(n \cdot b_1^l - p, \dots, n \cdot b_{m_l}^l - p\right)^T =$$
$$= (W')^{l^T} X^{l-1} - (B')^l.$$

Thus given a linear activation function, the output is equivalent to a single neuron.

1.3.3 Back-propagation

The second step of the training process is to implement the backpropagation algorithm. Let us first define the signal

$$s_j^l = \sum_{k=1}^{m_{l-1}} w_{jk}^l x_k^{l-1} - b_j^l.$$

We aim to apply gradient descent to minimize the cost function. To do so, we need to compute $\nabla\mathcal{C}$. As the cost function can be expressed as a function of the variables w_{ij}^l and b_j^i, we can express $\nabla\mathcal{C} = (\nabla_w\mathcal{C}, \nabla_b\mathcal{C})$. Now, using the chain rule

$$\frac{\partial\mathcal{C}}{\partial w_{ij}} = \frac{\partial\mathcal{C}}{\partial s_j^l}\frac{\partial s_j^l}{\partial w_{ij}},$$

where

$$\frac{\partial s_j^l}{\partial w_{ij}} = x_i^{l-1}.$$

We now denote the sensitivity error with respect to s_j^l as

$$\delta_j^l = \frac{\partial\mathcal{C}}{\partial s_j^l}. \tag{1.1}$$

Thus,

$$\frac{\partial\mathcal{C}}{\partial w_{ij}} = \delta_j^l x_i^{l-1}.$$

Equivalently, given

$$\frac{\partial\mathcal{C}}{\partial b_j^l} = \frac{\partial\mathcal{C}}{\partial s_j^l}\frac{\partial s_j^l}{\partial b_j^l},$$

we compute

$$\frac{\partial s_j^l}{\partial b_j^l} = -1.$$

Thus,

$$\frac{\partial\mathcal{C}}{\partial b_j^l} = -\delta_j^l.$$

Thus, the gradient of the cost function

$$\nabla\mathcal{C} = \delta_j^l(x_i^{l-1}, -1).$$

Next, we need to find the value of δ_j^l. To do so, we use the backpropagation algorithm. To compute an arbitrary δ_j^l, we start computing δ_j^{L+1} and move backward through the network. As expected, the expression for δ_j^l depends on the cost function. In this section, we assume the cost function is the quadratic cost function

$$\mathcal{C}(w,b) = \frac{1}{2m_0}\sum_{j=1}^{m_0}(f(x_j^0) - y^j)^2.$$

Now, the expression above can be rewritten as

$$\mathcal{C}(w,b) = \frac{1}{2m_{L+1}}\sum_{j=1}^{m_{L+1}}(x_j^{L+1} - y^j)^2.$$

Using that $x_j^l = \varphi(s_j^l)$ for any l, j and the chain rule, equation 1.1 becomes

$$\delta_j^{L+1} = \frac{\partial \mathcal{C}}{\partial s_j^{L+1}} = \frac{1}{m_{L+1}}(x_j^{L+1} - y^j)\varphi'(s_j^{L+1}),$$

for $l = L+1$. Next, we express δ_i^{l-1} as a function of δ_j^l. First, applying the chain rule and using that s_j^l is a function of s_k^{l-1}, we get

$$\delta_k^{l-1} = \frac{\partial \mathcal{C}}{\partial s_k^{l-1}} = \sum_{j=1}^{m_l}\frac{\partial \mathcal{C}}{\partial s_j^l}\frac{\partial s_l^j}{\partial s_k^{l-1}} = \sum_{j=1}^{m_l}\delta_j^l\frac{\partial s_l^j}{\partial s_k^{l-1}}. \tag{1.2}$$

Now,

$$\frac{\partial s_l^j}{\partial s_k^{l-1}} = \frac{\partial(\sum_{k=1}^{m_{l-1}} w_{jk}^l\varphi(s_k^{l-1}) - b_j^l)}{\partial s_k^{l-1}} = w_{ij}^l\varphi'(s_k^{l-1}).$$

Lastly, substituting into 1.2

$$\delta_k^{l-1} = \varphi'(s_k^{l-1})\sum_{j=1}^{m_l}\delta_j^l w_{ij}^l.$$

This expression is known as the backpropagation formula.

1.3.4 Updating the network, gradient descent

Recall that we defined the cost function to fulfill $\mathcal{C}(w,b) \approx 0$, when $f(x_j^0) \approx y^j$ for all j. Thus, we aim to implement the learning process and find an algorithm that finds weights and biases so that $\mathcal{C}(w,b) \approx 0$. To do so, we use the gradient descent method. Let us first define $\Delta\mathcal{C}$ given two arbitrary directions x_1 and x_2

$$\Delta\mathcal{C} \approx \frac{\partial \mathcal{C}}{\partial x_1}\Delta x_1 + \frac{\partial \mathcal{C}}{\partial x_2}\Delta x_2.$$

Now recall

$$\nabla \mathcal{C} \equiv \left(\frac{\partial \mathcal{C}}{\partial x_1}, \frac{\partial \mathcal{C}}{\partial x_2} \right)^T .$$

Using $\Delta x \equiv (\Delta x_1, \Delta x_2)^T$ we get

$$\Delta \mathcal{C} \approx \nabla \mathcal{C} \cdot \Delta x.$$

Let us now define

$$\Delta x = -\eta \nabla \mathcal{C},$$

where $\eta \gtrapprox 0$. Therefore

$$\Delta C \approx -\eta \nabla \mathcal{C} \cdot \nabla \mathcal{C} = -\eta \|\nabla \mathcal{C}\|^2.$$

Since

$$\|\nabla \mathcal{C}\|^2 \geq 0 \Rightarrow \Delta \mathcal{C} \leq 0.$$

This result shows that $\mathcal{C}$ will never increase when modifying x as $\Delta x = -\eta \nabla \mathcal{C}$. This yields the "update rule" of gradient descent

$$x \to x' = x - \eta \nabla \mathcal{C}.$$

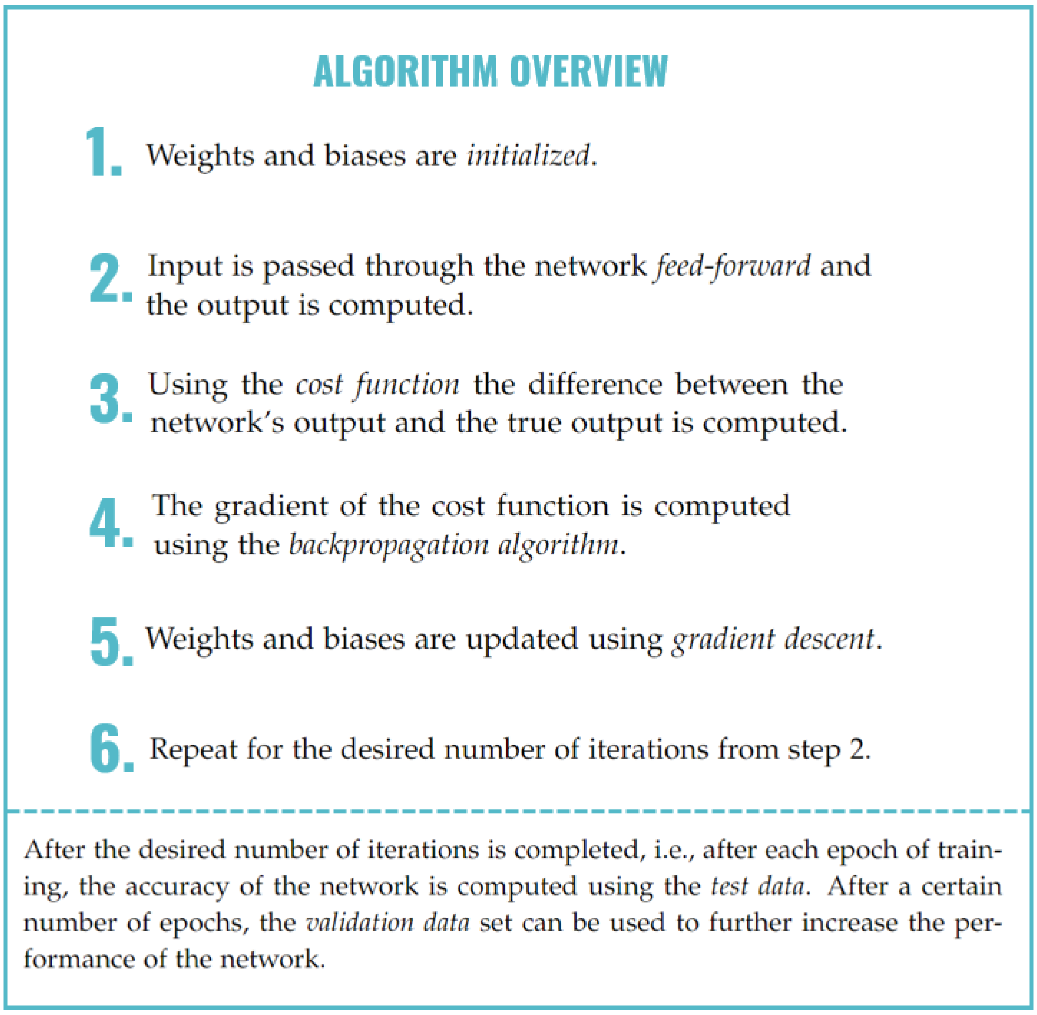
Now, let us consider a neural network. Rewriting the expression above for given weights and biases, we get

$$\begin{aligned} w^l_{ij}(n+1) &= w^l_{ij}(n) - \eta \delta^l_j(n) x^{l-1}_i(n) \\ b^l_j(n+1) &= b^l_j(n) + \eta \delta^l_j(n), \end{aligned}$$

where $x^l_i(n)$ and $\delta^l_j(n)$ are functions of the weights, $w^l_{ij}(n)$, and biases, $b^l_j(n)$, obtained at the n^{th} step and derived in sections 1.3.2 and 1.3.3.

1.3.5 Algorithm overview

We have presented the different steps of the training process. The training process is an iterative process that allows for "learning" to happen. We give a schematic containing the training steps to allow the reader to understand the training process. Before training an activation function, the learning rate and cost function need to be chosen. Afterward, these parameters can be changed to optimize the accuracy of the network.



ALGORITHM OVERVIEW

1. Weights and biases are *initialized*.
2. Input is passed through the network *feed-forward* and the output is computed.
3. Using the *cost function* the difference between the network's output and the true output is computed.
4. The gradient of the cost function is computed using the *backpropagation algorithm*.
5. Weights and biases are updated using *gradient descent*.
6. Repeat for the desired number of iterations from step 2.

After the desired number of iterations is completed, i.e., after each epoch of training, the accuracy of the network is computed using the *test data*. After a certain number of epochs, the *validation data* set can be used to further increase the performance of the network.

Chapter 2

The foundations of quantum computing

Classical computers have limited computational power. For example, modeling a many-body system, such as a molecule, can rapidly become computationally impossible for classical computers. Quantum computation has the potential to tackle solving complex systems problems. Although the implementation of quantum computers is in its early stages and has to overcome multiple challenges, it is a reality.

Understanding quantum mechanics is essential for understanding quantum computers. Thus it is also crucial for understanding quantum neural networks. Quantum mechanics has strong connections to many branches of mathematics, particularly functional analysis, and Hilbert spaces.

This chapter aims to introduce relevant quantum mechanics concepts from a mathematics perspective. These are the basic concepts of quantum computing that are critical for the understanding of chapter 3. First, we introduce Hilbert spaces, the quantum bit, and its characteristics and properties. Then, we present the gate model in quantum computing and the universality theorems. Lastly, we describe the current limitations of today's quantum computers and give two crucial definitions for understanding the implementation of dissipative quantum neural networks. In this chapter, the superscript * represents the complex conjugate of a given number.

2.1 The definition of a Hilbert Space

Hilbert spaces are the building blocks of quantum mechanics, as they appropriately describe the mathematical foundations of the theory. We illustrate the basic definitions of a Hilbert space described in [17].

Definition 2.1. Given a linear complex space X. We define the *inner product*

$$(\cdot,\cdot) : X \times X \to \mathbb{C}$$

such that for all $x, y, z \in X$

i $(x, \alpha y + \beta z) = \alpha(x, y) + \beta(x, z);\ \alpha, \beta \in \mathbb{C}$.

ii $(y, x) = (x, y)^*$.

iii $(x, x) = 0 \Leftrightarrow x = 0$.

iv $(x, x) \geq 0$.

Note that it can be inferred that $(\cdot,\cdot)$ is antilinear in the first argument, i.e.,

$$(\alpha x + \beta y, z) = \alpha^*(x, z) + \beta^*(y, z).$$

Definition 2.2. A *Hilbert space* is a complete inner product space.

2.2 Dirac notation

In this section, we will delve into the details of the Dirac notation, the prevalent method for expressing quantum states in quantum mechanics.

Given $\psi = (\alpha_0, \dots, \alpha_n) \in \mathbb{C}^{n+1}$, we refer to the column vector

$$|\psi\rangle = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_n \end{pmatrix}$$

as a ket $|\psi\rangle$, and we represent its adjoint with the so-called bra $\langle\psi|$

$$\langle\psi| = (\alpha_0^*, \dots, \alpha_n^*)\,.$$

Let us now define the following inner product referred to as a bra-ket. Let $\varphi = (\beta_0, \dots, \beta_n) \in \mathbb{C}^{n+1}$, now

$$\langle\psi \mid \varphi\rangle = (\alpha_0^*, \dots, \alpha_n^*) \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_n \end{pmatrix} = \alpha_0^*\beta_0 + \cdots + \alpha_n^*\beta_n.$$

2.3 Quantum bits

A classical computer operates with strings of zeros and ones. Every position in these strings is called a bit and can contain either a 0 or a 1. Quantum computers operate with the so-called qubits, which are quantum states that are allowed to be in a superposition of the two orthonormal vectors

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

These two vectors form the canonical basis of $\mathbb{C}^2$ and are referred to as the computational basis. Now, qubits in a so-called pure state can be expressed as

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

where $\alpha, \beta \in \mathbb{C}$ and fulfill $|\alpha|^2 + |\beta|^2 = 1$. Let us now build the tools to deal with multi-qubit systems. In this section, definitions are extracted from [18] and [19].

Definition 2.3. The *tensor product* of two vector spaces H_1 and H_2 is a vector space, denoted as $H_1 \otimes H_2$. It is constructed by taking the bases of each vector space $\{|\psi_i\rangle\}$ with $i \in \{1, \dots, n\}$, and $|\psi_j\rangle$ with $j \in \{1, \dots, m\}$, and combining them to form a new basis: $\{|\psi_i\rangle \otimes |\varphi_j\rangle\}$. The tensor product is a bilinear map denoted by $\otimes : V \times W \to V \otimes W$

Definition 2.4. Let M_1, M_2 be two matrix spaces. We define the *Kronecker product* as $\otimes : M_1 \times M_2 \longrightarrow M_1 \otimes M_2$ of $A \in M_1$ and $B \in M_2$ is defined as

$$A \otimes B = \begin{bmatrix} A_{11}B & \dots & A_{1L}B \\ \vdots & \ddots & \vdots \\ A_{K1}B & \dots & A_{KL}B \end{bmatrix},$$

where A_{kl} denotes the (k,l)-th entry of A.

Remark 2.5. Note, that for two qubits $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \in \mathbb{C}^2$ and $|\varphi\rangle = \gamma |0\rangle + \delta |1\rangle \in \mathbb{C}^2$ the Kronecker product

$$|\psi\rangle \otimes |\varphi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \otimes \begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} \alpha \cdot \gamma \\ \alpha \cdot \delta \\ \beta \cdot \gamma \\ \beta \cdot \delta \end{pmatrix} \in \mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^4.$$

From now on, we adopt the commonly used notation to refer to the Kronecker product of two quantum states $|\psi\rangle \otimes |\varphi\rangle$ as $|\psi\varphi\rangle$.

Example 2.6. Using this notation, we can express a two-qubit state generally as

$$|\psi\rangle = \beta_{00}|00\rangle + \beta_{01}|01\rangle + \beta_{10}|10\rangle + \beta_{11}|11\rangle\,,$$

where $|\beta_{00}|^2 + |\beta_{10}|^2 + |\beta_{01}|^2 + |\beta_{11}|^2 = 1$.

2.4 The Bloch sphere

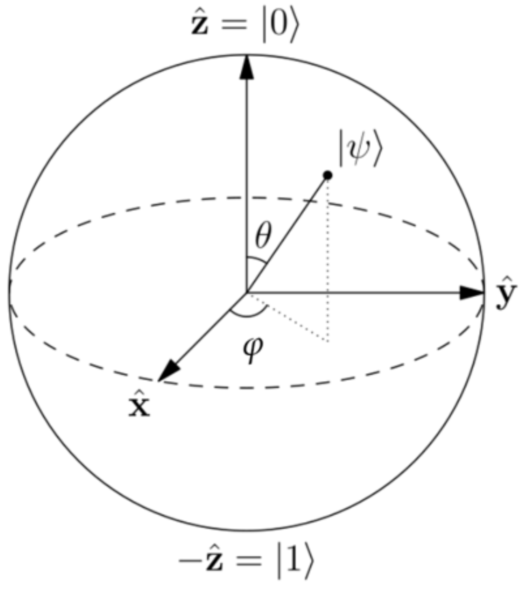


Figure 2.1: Bloch representation of a quantum state $|\psi\rangle$

Qubits can be visualized as points on the so-called Bloch sphere, named after F. Bloch, the Swiss physicist who first proposed this representation. Let us justify the Bloch representation [22]. We first consider the pure qubit state $|\psi\rangle$, normalized to 1 with $|\alpha|^2 + |\beta|^2 = 1$

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

where $\alpha, \beta \in \mathbb{C}$. The expression above can be rewritten as

$$|\psi\rangle \equiv \cos(\frac{\theta}{2})|0\rangle + \sin(\frac{\theta}{2})e^{i\varphi}|1\rangle\,,$$

where we take $\theta \in [0, \pi]$ and $\varphi \in [0, 2\pi)$ such that there is no angle repetition, refer to figure 2.1. Lastly, we use spherical coordinates to represent the quantum state on the surface of a unit sphere

$$
\begin{aligned}
x &= \sin\theta\cos\varphi, \\
y &= \sin\theta\sin\varphi, \\
z &= \cos\theta.
\end{aligned}
$$

2.5 Pure and mixed quantum states

Previously, we have studied quantum systems described by state vectors and referred to them as states. We distinguish those pure states from a general type of quantum state: the mixed state, using the definitions given in [1, 20, 21].

Definition 2.7. A *mixed state* is a set of quantum states

$$|\psi\rangle = \{(p_i, |\psi_i\rangle)\}_{i=1}^{n},$$

where $p_i \in [0,1]$ for $i \in \{1, \dots n\}$ form a probability distribution such that $\sum_i^n p_i = 1$ and each $|\psi_i\rangle$ is an associated pure state.

Definition 2.8. Given two vectors u and v of dimension n and m respectively. We define the *outer product of u and v* as

$$u \otimes v = \begin{bmatrix} u_1v_1 & u_1v_2 & \dots & u_1v_n \\ u_2v_1 & u_2v_2 & \dots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \dots & u_mv_n \end{bmatrix}.$$

Example 2.9. Given a quantum state $|\psi\rangle$ we express the outer product of the state with itself as $|\psi\rangle\langle\psi|$

$$|\psi\rangle\langle\psi| = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix} \begin{bmatrix} \alpha_0^* & \alpha_1^* & \dots & \alpha_N^* \end{bmatrix} = \begin{bmatrix} |\alpha_0|^2 & \alpha_0\alpha_1^* & \dots & \alpha_0\alpha_N^* \\ \alpha_1\alpha_0^* & |\alpha_1|^2 & \dots & \alpha_1\alpha_N^* \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_N\alpha_0^* & \alpha_N\alpha_1^* & \dots & |\alpha_N|^2 \end{bmatrix}.$$

Definition 2.10. The *density operator* of a mixed quantum state $|\psi\rangle$ is denoted as

$$\rho = \sum_{i=1}^{n} p_i |\psi_i\rangle\langle\psi_i|.$$

Remark 2.11. Given the pure state $|\psi\rangle$, the density operator can be expressed as

$$\rho = |\psi\rangle\langle\psi|.$$

Remark 2.12. For a pure state, the density operator fulfills

$$\rho^2 = \rho.$$

2.5.1 Reduced density matrix

Equivalently to example 2.5, we can build composite mixed-state systems. Given a state $\rho_A \in H_A$ and another state $\rho_B \in H_B$, we define the density matrix of the two states as

$$\rho_{AB} = \rho_A \otimes \rho_B \in H_A \otimes H_B.$$

Given a linear operator $\rho : H \to H$ in a Hilbert space H, the space composed by these operators is $\mathcal{L}(H)$. Now, recall that the trace of the operator ρ, $\mathrm{Tr} : \mathcal{L}(H) \to \mathbb{C}$, is defined as the sum of the diagonal elements of ρ. When using a basis $|\psi_i\rangle \in H$ we can then express the trace of ρ as $\mathrm{Tr}(\rho) = \sum_{i=1}^{dim} \langle\psi_i| \rho |\psi_i\rangle$, where *dim* is the dimension of H.

Definition 2.13. Given the density matrix of a compound system $\rho_{AB} \in H = H_A \otimes H_B$, we define the *partial trace* over B as a map $\mathrm{Tr}_B : \mathcal{L}(H) \to \mathcal{L}(H_A)$ defined as

$$\mathrm{Tr}_B(\rho_{AB}) = \sum_{j=1}^{d_B} \left(\mathbb{I}_A \otimes \langle b_j|\right) \rho_{AB} \left(\mathbb{I}_A \otimes |b_j\rangle\right),$$

where

$$|b_j\rangle = \begin{bmatrix} b_{j1} & b_{j2} & \cdots & b_{jd_B} \end{bmatrix}^t,$$

is a basis of the Hilbert space H_B, $d_B = \dim H_b$, and $\mathbb{I}_B$ is the identity operator in H_B.

2.6 Entangled states

Quantum entanglement is one of the distinctive features of quantum mechanics compared to classical physics. An entangled state is a quantum state which cannot be factored into a product of different quantum states; this concept is illustrated in definition 2.14 [2].

Definition 2.14. A quantum state $|\psi\rangle \in \mathbb{C}^4$ is *entangled* if given two quantum states $|\phi_1\rangle = \alpha|0\rangle + \beta|1\rangle \in \mathbb{C}^2$ and $|\phi_2\rangle = \gamma|0\rangle + \delta|1\rangle \in \mathbb{C}^2$ the system

$$|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \otimes \begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} \alpha\cdot\gamma \\ \alpha\cdot\delta \\ \beta\cdot\gamma \\ \beta\cdot\delta \end{pmatrix} \in \mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^4$$

has no solution.

A remarkable property of entangled states is their response to measurement. When a multiple qubit entangled state is measured, we force the collapse onto a new state of all qubits. We now introduce an example to illustrate this property better.

Example 2.15. Alice and Bob are building an experiment and studying the entangled quantum state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A |0\rangle_B + |1\rangle_A |1\rangle_B)$. Prior to Alice applying the gate σ_A^z, Bob can measure $\sigma_B^z = \pm 1$, but after Alice performs the measurement, the state in B collapses, and Bob can only obtain a single result.

Remark 2.16. Note that there are no restrictions to the separation at which two qubits are entangled, i.e., two qubits stay entangled given any distance between the two.

2.7 The gate model in quantum computing

2.7.1 Quantum gates

Quantum gates and qubits are the building blocks of gate-based quantum computations. Quantum gates are unitary operators acting on qubits and can be represented by unitary matrices of dimension $2^n \times 2^n$, where n is the number of qubits the gate acts on.

Definition 2.17. A $n \times n$ dimensional matrix U, is *unitary* if it satisfies

$$U^\dagger U = UU^\dagger = UU^{-1} = I,$$

where $U^\dagger$ denotes the hermitian conjugate.

Let us now introduce some helpful properties of unitary transformations; see [4] for further details.

Proposition 2.18. Given two unitary matrices U and V, the composition UV is unitary.

Proof. Let U and V be unitary matrices, then

$$(UV)^\dagger = V^\dagger U^\dagger = V^{-1}U^{-1} = (UV)^{-1}.$$

□

Proposition 2.19. Let U be a unitary matrix, then U preserves inner products,

$$(U\vec{a}, U\vec{b}) = (\vec{a}, \vec{b}).$$

Proof. Let U be a unitary matrix, then

$$(U\vec{a}, U\vec{b}) = (U\left|a\right\rangle)^{\dagger}(U\left\langle b\right|) = \left\langle a\right| U^{\dagger}U\left|b\right\rangle = \langle a|b\rangle = (\vec{a}, \vec{b}).$$

□

Proposition 2.20. Let U be a unitary matrix, then all of its eigenvalues satisfy $|\lambda| = 1$, i.e., all eigenvalues are of the form $e^{i\theta}$.

Proof. Let U be a unitary matrix. The eigenvalue equation for U is then

$$Uv = \lambda v.$$

Taking the conjugate transpose of the expression above,

$$v^*U^* = \lambda^* v^*.$$

Note that we used $*$ to refer to the conjugate transpose. By multiplying both relations, we get

$$v^*U^*Uv = \lambda^* v^* \lambda v \Rightarrow v^* I v \quad = (\lambda^*\lambda)\, v^* v \Rightarrow \|v\|^2 = |\lambda|^2 \|v\|^2 \Rightarrow |\lambda| \quad = 1.$$

□

This definition of the quantum gate is coherent for pure states, yet it rests unclear how to apply unitaries to mixed-states ρ. Given a unitary matrix U, we say we apply a gate on a mixed state ρ performing

$$U\rho U^{\dagger}.$$

In quantum computing, we have infinite single-qubit gates since we can map one qubit to any point on the Bloch sphere, refer to section 2.4. Let us now indicate several examples obtained from [23]. The Pauli X, Y, and Z gates are fundamental types of such single-qubit gates. We represent such gates with the standard basis $\{\left|0\right\rangle, \left|1\right\rangle\}$ as

$$X = \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

which represent a π radians rotation around the x, y, or z axis respectively.

Remark 2.21. The X-gate is also referred to as a NOT-gate. To understand it, let us study the state $|0\rangle$ when acted upon by the X-gate

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle .$$

Note that this gate acts analogously to the logical NOT-gate, which turns 0 bits into 1 bits and vice versa.

Let us also present the Hadamard gate

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.1}$$

which creates a superposition of $|0\rangle$ and $|1\rangle$. The Hadamard gate can also be interpreted as a $\frac{\pi}{2}$ radians rotation in y, followed by a π radians rotation in x. When applied to $|0\rangle$ and $|1\rangle$, the Hadamard gate gives

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

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

There also exist gates that act on multiple qubits. An essential type of these multiple qubit gates is the controlled gate.

Definition 2.22. Given a unitary matrix U of dimension n the (d+n)-dimensional *controlled-U gate*, $c - U$, is the matrix of the form

$$\begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix},$$

where I is the d-dimensional identity matrix.

Remark 2.23. Note that the two-qubit controlled-U gate applies the U gate to the second qubit when the first qubit is in state $|1\rangle$

$$(c - U)|00\rangle = |00\rangle, (c - U)|11\rangle = (I \otimes U)|11\rangle,$$

$$(c - U)|01\rangle = |01\rangle, (c - U)|10\rangle = (I \otimes U)|10\rangle.$$

An important example of controlled gates is the C-NOT gate, which expressed in the basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ is

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

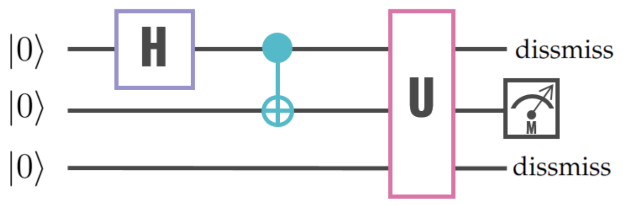


Figure 2.2: Schematic representation of a quantum circuit consisting of a Hadamard gate, followed by a C-NOT gate, and an arbitrary U three-qubit gate. Only the center qubit is measured, and the other two qubits are dismissed.

Quantum circuits

Equivalent to the classical logic gate implementation, we can build quantum circuits consisting of applied gates and measurements on qubits to perform complex algorithms. Let us first introduce the schematic representation of quantum circuits. In figure 2.2 we represent a quantum circuit consisting of 3 qubits. We represent time in the x-axis and depict qubits evolving in time as horizontal lines with their initial state at the left-most side. Gates and operations acting on the qubits are represented as different shapes on top of the qubits.

Let us now introduce two fundamental theorems which play a crucial role in the proper implementation of quantum circuits.

Solovay-Kitaev theorem

In classical circuits, a universal set of gates are those gates that can implement any Boolean function. Finding gates that possess such properties allow for cheaper and easier circuit implementations. In quantum circuits, there exists the equivalent concept to universal gates. We refer to [24] to derive the relevant definitions.

Definition 2.24. A gate set, G, is *universal* if the unitary transformations performed with such set are dense in $U(2^n)$, up to a given phase, i.e., for all $M \in U(2^n)$ and $\epsilon > 0$, exists a unitary $\bar{M}$ that fulfills $\sup(||M - \bar{M}||) \leq \epsilon$.

The Solovay-Kitaev theorem ensures that a universal set of gates exists for single-qubit gates.

Theorem 2.25. (Solovay-Kitaev theorem) Given a universal set of single-qubit gates, $G \in U(2)$, with $g^{-1} \in G$ for any $g \in G$, and such that the group generated by its elements $< G > \in SU(2)$. Then, there exists c s.t. for any $U \in SU(2)$ and a given ϵ; there is a sequence S of gates in G with length $O(\log^c \frac{1}{\epsilon})$ s.t. $||S - U|| \leq \epsilon$.

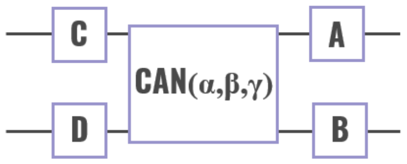


Figure 2.3: Universal single-qubit gate implementation.

There exist different sets of universal quantum gates. Let us introduce a universal set of two-qubit gates, which will be of use in section 3.4 derived from [25].

Theorem 2.26. Any two-qubit gate can be implemented using a two-qubit canonical gate, and twelve single-qubit gates can be implemented as shown in figure 2.3. The canonical gate is parametrized as a function of $\pi/2 \geq \alpha \geq \beta \geq |\delta|$, and where

$$\mathrm{CAN}(\alpha,\beta,\delta) = \exp\left(-i\left(\alpha\sigma_X^{\otimes 2} + \beta\sigma_Y^{\otimes 2} + \delta\sigma_Z^{\otimes 2}\right)\right) = RXX\left(\alpha\pi\right) RYY\left(\beta\pi\right) RZZ\left(\delta\pi\right).$$

Remark 2.27. A single qubit operation can be decomposed as $U = Z^{p_1}Y^{p_2}Y^{p_3}$, where $p_1, p_2, p_3 \in \mathbb{R}$, and

$$Y^p \approx R_Y(\pi p) = e^{-i\frac{\pi}{2}pY},$$
$$Z^p \approx R_Z(\pi p) = e^{-i\frac{\pi}{2}pZ}$$

are parametrized Z and Y unitaries up to a phase. Such reparametrization can be expressed as a single unitary

$$u\left(p_1, p_2, p_3\right) = \begin{pmatrix} \cos\left(p_1/2\right) & -e^{ip_3}\sin\left(p_1/2\right) \\ e^{ip_2}\sin\left(p_1/2\right) & e^{i(p_3+p_2)}\cos\left(p_1/2\right) \end{pmatrix}.$$

No-cloning theorem

For classical systems cloning data or copying data is feasible. Classical computers constantly copy bits of information, but what about quantum computers? The no-cloning theorem exhibits that no quantum operation can perfectly and deterministically duplicate a pure state. A single unitary operator can only be used to clone quantum states that are mutually orthogonal. We introduce the no-cloning theorem and two proofs extracted from [26].

Theorem 2.28. (No-cloning theorem) Let $|\phi\rangle$ and $|\psi\rangle$ be two quantum states, and U_{cl} a unitary which copies the quantum states:

$$\begin{aligned} |\phi\rangle \otimes |0\rangle &\xrightarrow{U_{cl}} |\phi\rangle \otimes |\phi\rangle, \\ |\psi\rangle \otimes |0\rangle &\xrightarrow{U_{cl}} |\psi\rangle \otimes |\psi\rangle, \end{aligned}$$

then $\langle\phi|\psi\rangle$ is either 0 or 1.

Remark 2.29. Note that if $\langle\phi|\psi\rangle = 0 \Leftrightarrow |\phi\rangle$ and $|\psi\rangle$ are orthogonal, and if $\langle\phi|\psi\rangle = 1 \Leftrightarrow |\phi\rangle = |\psi\rangle$.

We show two existing proofs of the no-cloning theorem. The first proof was independently derived by D. Dieks in 1982 and Zurek and Wooters the same year.

Proof. Suppose there exists a unitary operator U_{cl} which can clone the state $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$. First, since ϕ is a pure state we apply U_{cl}

$$|\phi\rangle \otimes |0\rangle \xrightarrow{U_{cl}} |\phi\rangle \otimes |\phi\rangle = \alpha^2|00\rangle + \beta^2|11\rangle + \alpha\beta|01\rangle + \beta\alpha|10\rangle .$$

On the other hand applying U_{cl} on each term:

$$(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle \xrightarrow{U_{cl}} \alpha|00\rangle + \beta|11\rangle .$$

Since both expressions differ, the unitary U_{cl} does not exist. □

H. Yuen presented the second proof in 1986.

Proof. Since the inner product is preserved under unitary operations

$$\begin{aligned} \langle\phi|\psi\rangle = ((\langle\phi|\otimes\langle 0|)(|\psi\rangle\otimes|0\rangle)) = ((\langle\phi|\otimes\langle 0|)U_{\mathrm{cl}}^{\dagger}U_{\mathrm{cl}}(|\psi\rangle\otimes|0\rangle)) = \\ = ((\langle\phi|\otimes\langle\phi|)(|\psi\rangle\otimes|\psi\rangle)) = \langle\phi|\psi\rangle^2 \Rightarrow \langle\phi|\psi\rangle \text{ is either 0 or 1.} \end{aligned}$$

□

2.7.2 Noisy intermediate-scale quantum devices

Currently, quantum computers are in the noisy intermediate-scale quantum era (NISQ), which refers to the fabrication limitations and noise sensitivity of state-of-the-art quantum computers. Quantum computers today are shallow and only have a limited number of qubits and applicable gates. Such qubits are limited in number and have high noise; therefore, they perform imperfect operations in the so-called coherence time [6]. As a reference, the most powerful quantum processor today is the Osprey processor, announced in 2022 by IBM and composed of 433 functional qubits. The number of qubits in quantum processors is fast-growing; according to IBM's development roadmap, their quantum processors are expected to have over 4,000 qubits by 2024. Common complications that current quantum computers encounter are the loss of coherence of the wave function as a function of time, non-precise gate implementation, and read-out errors. Quantum computing in the NISQ does not allow for most theoretical work to be implemented, and currently, methods such as variational approaches are being used.

2.8 Fidelity of a pair of quantum states

The fidelity of a pair of quantum states is a measure of their degree of similarity. This section gives the appropriate definitions extracted from [27]. Comparing two quantum states is an essential step in quantum machine learning because it enables us to evaluate how closely the output quantum state produced by our network matches the desired output quantum state, as discussed in Section 3.3.1.

Definition 2.30. Let ρ_1 and ρ_2 be pure states in a finite Hilbert space. We define the *fidelity between two pure states*, $F(\rho_1, \rho_2)$, as the transition probability:

$$F(\rho_1, \rho_2) = F\left(|\psi_1\rangle\langle\psi_1|, |\psi_2\rangle\langle\psi_2|\right) = |\langle\psi_1 \mid \psi_2\rangle|^2 .$$

Remark 2.31. Note that $F(\rho_1, \rho_2) \in [0, 1]$, and $F(\rho_1, \rho_2) = 1 \Leftrightarrow \rho_1 = \rho_2$ and $F(\rho_1, \rho_2) = 0 \Leftrightarrow$ the two states are orthogonal (i.e., perfectly distinguishable).

Now, let $\rho_1 = |\psi_1\rangle\langle\psi_1|$ be a pure state, and ρ_2 a non-pure state. We can generalize the definition of fidelity as follows:

$$F\left(|\psi_1\rangle\langle\psi_1|, \rho_2\right) = \langle\psi_1 |\rho_2| \psi_1\rangle .$$

Naturally, it is not trivial to define the fidelity of two non-pure quantum states which provide adequate physical meaning. In 1994, R. Jozsa introduced an axiomatic definition of the fidelity of a pair of quantum states.

Definition 2.32. The *fidelity* of a pair of quantum states ρ and σ is a function $F(\rho, \sigma)$ that follows

i $F(\rho, \sigma) \in [0, 1]$.

ii $F(\rho, \sigma) = 1 \iff \rho = \sigma$.

iii $F(\rho, \sigma) = F(\sigma, \rho)$.

iv $F\left(U\rho U^{\dagger}, U\sigma U^{\dagger}\right) = F(\rho,\sigma)$ for any unitary matrix U.

v $F(\rho,\sigma) = \mathrm{Tr}(\rho\sigma)$ if ρ or σ are pure states.

and gave a definition of fidelity that fulfills such properties

$$F(\rho,\sigma) = \left(\mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)^2,$$

where $\sqrt{\rho}$ is uniquely defined in theorem 2.33 [28]. This definition of the fidelity of two quantum states is one of the most commonly used.

Theorem 2.33. Given a positive semidefinite square matrix A of dimension n, there exists a unique positive semidefinite $n \times n$ matrix B, which fulfills $B^2 = A$.

Remark 2.34. Using the spectral theorem, we can write A as

$$A = U \,\mathrm{diag}\left[\lambda_1, \dots, \lambda_n\right] U^*,$$

where U is an orthogonal matrix and $\lambda_i \geq 0 \; \forall i \in \{1, \dots, n\}$. Thus, we can write B as

$$B = U \,\mathrm{diag}\left[\sqrt{\lambda_1}, \dots, \sqrt{\lambda_n}\right] U^*.$$

2.9 Completely positive maps

We give a set of definitions and results that will be of use in section 3.3, where we will use completely positive maps to describe quantum circuits. The theorems presented in this section provide essential properties to prove the propositions of chapter 3. All results are extracted from [29] and [30].

Definition 2.35. Let A and B be C^*-algebras, then a linear map $\mathcal{E} : A \to B$ is a *positive map* if it maps $\mathcal{E}(A_+) \to B_+$, where A_+, B_+ are the positive parts of A and B respectively.

Definition 2.36. Given the C^*-algebras A and B, and the map

$$\phi : id \otimes \mathcal{E} : \mathcal{C}^{n\times n} \otimes A \to \mathcal{C}^{k\times k} \otimes B.$$

We say $\mathcal{E}$ is a *completely positive map* if $id \otimes \mathcal{E}$ is a positive map for every k.

Definition 2.37. Let H be a Hilbert space, and B(H) be the bounded operators on H, $\rho : H \to H$. The map $\mathcal{E}$ is *trace preserving* if

$$\mathrm{Tr}(\mathcal{E}(\rho)) = \mathrm{Tr}(\rho) \quad \forall \rho \in B(H).$$

Theorem 2.38. *(Stinespring dilation theorem)*

All completely positive and trace-preserving maps $\mathcal{E}$ on $H = \mathbb{C}^n$ can be expressed as

$$\mathcal{E}(\rho) = \mathrm{Tr}_2 \left(U(\rho \otimes \eta) U^\dagger \right),$$

where U is a unitary on the space $\mathbb{C}^n \otimes \mathbb{C}^N$ with $n \leq N$, $\lambda \in \mathbb{C}^N$ is a quantum state, and Tr_2 represents the partial trace acting on the second factor of the tensor product.

Remark 2.39. According to the Stinespring dilation theorem, any quantum circuit that can be described using tensor products, unitary transformations, and partial trace operations can be analogously described using completely positive trace-preserving maps.

Theorem 2.40. *(Kraus' theorem)* Given two Hilbert spaces H and K, a linear map $\mathcal{E} : H \to K$ is completely positive and trace-preserving if

$$\mathcal{E}(\rho) = \sum_{j=1}^{r} K_j \rho K_j^\dagger,$$

where

$$\sum_{j=1}^{r} K_j^\dagger K_j = \mathbb{1}.$$

Chapter 3

Quantum machine learning

Quantum machine learning (QML) involves the combination of machine learning techniques and quantum computing and aims to use quantum science to tackle data-driven problems. Several approaches have been proposed to achieve this goal, including using classical machine learning algorithms to process quantum data, quantum-enhanced machine learning, and quantum algorithms that resemble classical machine learning algorithms [10, 11, 12, 13].

This chapter focuses on the so-called quantum neural networks first presented in 1995 by S. Kak [31]. In S. Kak's paper, quantum neural networks are presented theoretically. Today, there are multiple proposals for QNNs, and their implementation is now a reality, although it is affected by the current limitations of quantum computers. In this thesis, we review the so-called dissipative quantum neural networks (DQNN) proposed by K. Beer in 2020 [14] and focus on the theory introduced in K. Beer's Ph.D. thesis published in 2022 [15]. We focus on K. Beer's work as it has a solid mathematical foundation and allows for a description analogous to classical neural networks.

First, we present the quantum perceptron and introduce dissipative neural networks. Afterward, we introduce the training algorithm. We review the training problem, the feed-forward algorithm, backpropagation, and the network's updating procedure. Lastly, we describe the network's implementation on NISQ devices.

3.1 The quantum perceptron

The quantum perceptron is the equivalent of the classical perceptron neuron in DQNN. This perceptron is defined as a parametrized unitary matrix, U, of dimension $2^{m+n} \times 2^{m+n}$, where m is the number of input qubits and n the number of output qubits.

Recall the structure of a classical perceptron described in section 1.1.1. To build the quantum equivalent, we build an analogous system, shown in figure 3.1. To implement it, we initialize m input qubits in the desired state ρ^{in} and n qubits, which will act as output qubits in the state $|0\dots0\rangle_n \langle 0\dots0|$. Then, an arbitrary perceptron is applied, and the output is an n-qubit output state denoted as ρ^{out}.

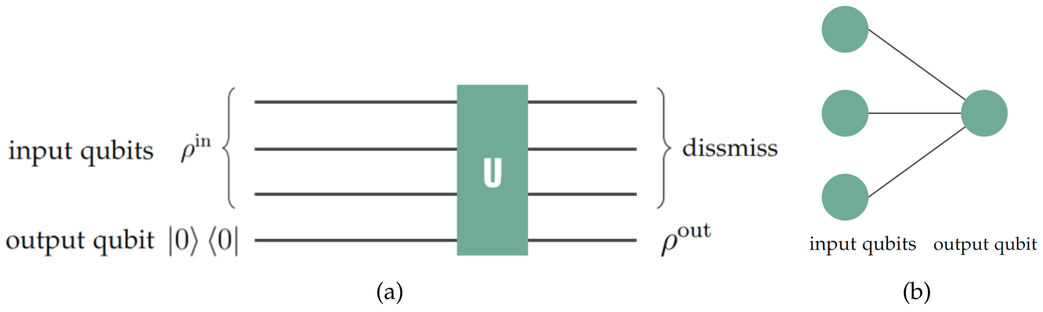


Figure 3.1: (a) Circuit implementation of a perceptron. (b) Schematic representation of quantum perceptrons analogous to the representation of classical perceptrons. The circles on the left are the input qubits, the circles on the right are the output qubits, and the connecting lines represent the unitary acting on the qubits.

3.2 Dissipative quantum neural networks

Dissipative quantum neural networks are the quantum equivalent of neural networks we will focus on in this work. Such structures can be represented and implemented as shown in figure 3.2 and 3.3. Perceptrons are composed of arbitrary unitary matrices; thus, two perceptrons do not usually commute. Therefore, we number the perceptrons acting between layers. Here, U_j^l is the j^{th} perceptron acting between layer l and $l-1$. As can be seen in figure 3.3 U_1^1 is the perceptron that first acts on the qubits, followed by the perceptron U_2^1, and so forth. When representing such structures schematically, we represent the perceptron that acts first on top, followed by the perceptrons acting afterward underneath it. Note that the structure chosen for implementing DQNNs follows from the no-cloning theorem described in section 2.7.1. If a gate could clone quantum states, we could reduce the number of qubits in the network.

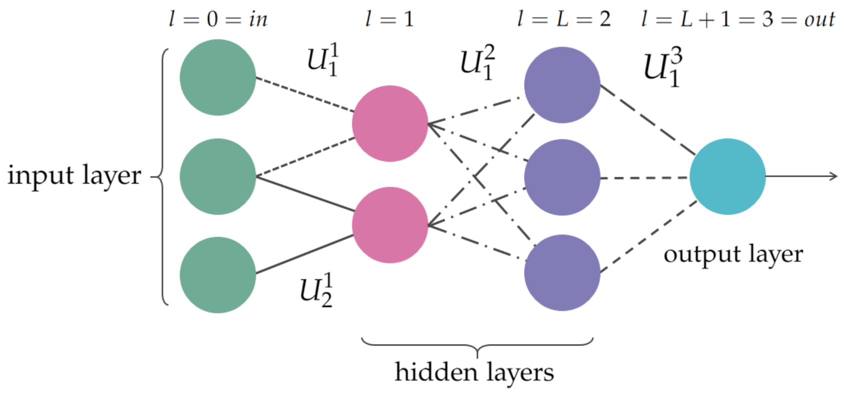


Figure 3.2: Network representation for a DQNN consisting of 2 hidden layers. Note that we use L to refer to the last hidden layer. The notation for input, hidden, and output layers are built equivalently to the classical neural network analog. The identical color circles are qubits in the same layer, and the same style lines represent the same applied perceptron.

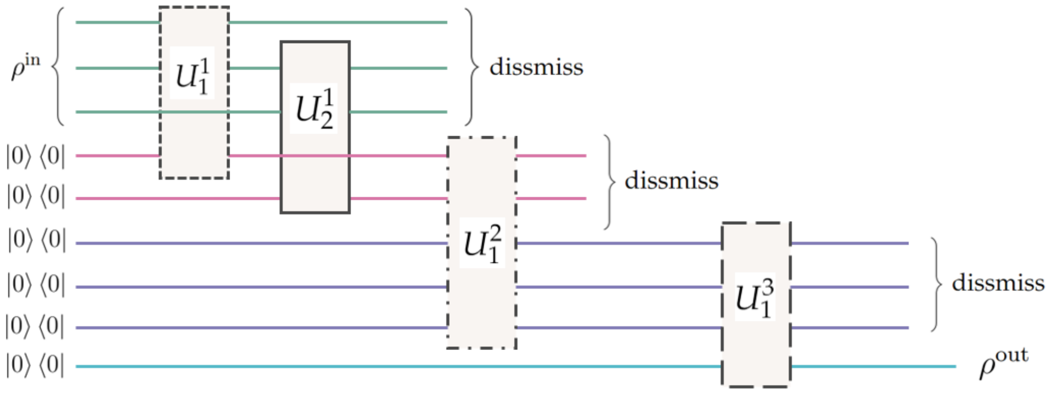


Figure 3.3: Implementation of the DQNN in figure 3.2.

3.3 The training algorithm

3.3.1 The training problem

Let us now define the training problem. In classical neural networks, we had a training data set. For example, when doing image recognition, we had images of handwritten numbers and labels with the number that each image represented. We aimed to train the network to output the correct number when given a specific image. For DQNNs, we have an equivalent situation. We have data sets of input quantum states and the corresponding output state. Our aim is that the DQNN outputs the desired quantum state, i.e., the network "learns" to apply a specific set of unitary matrices, depending on the input quantum state.

Let us define our data pairs $\{|\phi_x^{\text{in}}\rangle, |\phi_x^{\text{des}}\rangle\}$, where $\rho_x^{\text{in}} \equiv |\phi_x^{\text{in}}\rangle\langle\phi_x^{\text{in}}|$ is the input state and $|\phi_x^{\text{des}}\rangle$ is the desired output. To compare the output of our network ρ_x^{out} to $|\phi_x^{\text{des}}\rangle$ we use the definition of fidelity of two quantum states defined in section 2.8. Equivalently to classical neural networks, we divide our N training data pairs into two sets: S training data pairs $\{|\phi_x^{\text{in}}\rangle, |\phi_x^{\text{tr}}\rangle\}$, and N-S validation data sets $\{|\phi_x^{\text{in}}\rangle, |\phi_x^{\text{val}}\rangle\}$.

3.3.2 Feed-forward

The first step of the algorithm consists of applying the perceptrons as shown in figure 3.3. Let us derive the expression for the general output quantum state ρ^{out}.

Recall the expression for a unitary transformation given an initial state

$$U\rho^{\text{in}}U^\dagger.$$

Given the general initial state of our system and the first applied perceptron, U_1^1 the expression above becomes

$$U_1^1 \otimes \mathbb{1}_1^1(\rho^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|)U_1^{1\dagger} \otimes \mathbb{1}_1^1,$$

where $\mathbb{1}_1^1$ represents the identity matrix acting on the qubits that are not affected by U_1^1, and $|0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|$ is the zero product state considering all qubits in the hidden and output layers. Henceforward, we write $U_j^l \otimes \mathbb{1}_j^l$ as U_j^l for simplicity. Subsequently, for the first layer of perceptrons

$$U_{m_1}^1 \dots U_1^1(\rho^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|)U_1^{1\dagger} \dots U_{m_1}^{1}{}^\dagger,$$

where m_l is the number of perceptrons that act as the intermediary between layer l and layer $(l-1)$. We express the sequence of applied perceptrons as a single unitary matrix $U = (U_1^1 \dots U_{m_1}^1)\dots(U_1^l \dots U_{m_l}^l)\dots(U_1^{\text{out}}\dots U_{m_{\text{out}}}^{\text{out}}) = U^1\dots U^l \dots U^{out}$ where m_l is the number of perceptrons acting between the l^{th} layer and the $(l-1)^{\text{th}}$, and $U^l = U_1^l \dots U_{m_l}^l$. After applying all perceptrons

$$U(\rho^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|)U^\dagger.$$

We can now express the output quantum state ρ^{out}

$$\rho^{\text{out}} = \text{Tr}_{\text{in,hid}}(U(\rho^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|)U^\dagger), \tag{3.1}$$

where $\text{Tr}_{\text{in,hid}}$ denotes the partial trace on the system of qubits in the input and hidden layers. In section 2.9, we showed that any quantum circuit that can be described using unitary transformations, partial traces, and tensor products is equivalent to the completely positive map description. Thus, we can rewrite the expression above as

$$\rho^{\text{out}} = \mathcal{E}(\rho^{\text{in}}),$$

where $\mathcal{E}$ is a completely positive map. Equivalently to equation 3.1, we can express the output quantum state

$$\rho^{\text{out}} = \mathcal{E}^{L}(\mathcal{E}^{L-1}(\ldots\mathcal{E}^{2}(\mathcal{E}^{1}(\rho_{\text{in}}))\ldots)), \tag{3.2}$$

given

$$\mathcal{E}^{l}(X^{l-1}) = \text{Tr}_{l-1}(U^{l}(X^{l-1} \otimes |0\ldots0\rangle_{l}\langle 0\ldots0|\, U^{l\dagger}), \tag{3.3}$$

where Tr_{l-1} is the partial trace on the $(l-1)^{th}$ layer, and $|0\ldots0\rangle_{l}\langle 0\ldots0|$ is the zero product state of the qubits in layer l.

Example 3.1. For a better understanding of equations 3.2 and 3.3, let us compute the output quantum state of the neural network in figure 3.4. First, let us write the initial quantum state of layers 0 and 1

$$\rho^{\text{in}} \otimes |00\rangle\langle 00|\,.$$

We now apply the perceptrons acting on such a state

$$U_2^1 U_1^1(\rho^{\text{in}} \otimes |00\rangle\langle 00|)U_1^{1\dagger}U_2^{1\dagger}.$$

Note that we are omitting the identity matrices for clearer notation. We now trace out the qubits on the input layer and are left with the output of the first layer X^0

$$X^0 = \text{Tr}_0(U_2^1 U_1^1(\rho^{\text{in}} \otimes |00\rangle\langle 00|)U_1^{1\dagger}U_2^{1\dagger}).$$

We consider the quantum state of the output layer and the output state X^0

$$\text{Tr}_0(U_2^1 U_1^1(\rho^{\text{in}} \otimes |00\rangle\langle 00|)U_1^{1\dagger}U_2^{1\dagger}) \otimes |000\rangle\langle 000|\,.$$

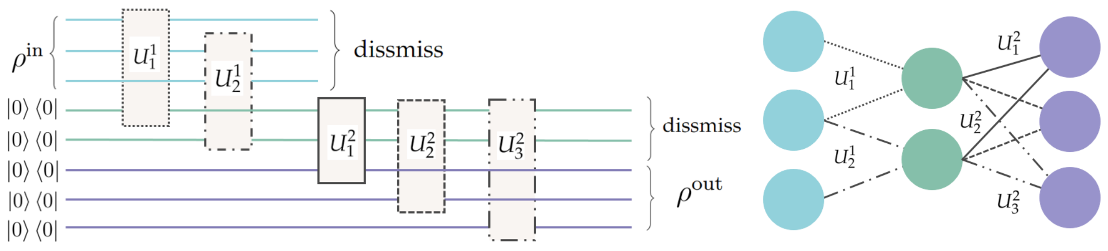


Figure 3.4: (a) Implementation of the quantum neural network. (b) Schematic representation of the quantum neural network.

We apply the perceptrons acting between layers 1 and 2 and, lastly trace out the hidden layer

$$\rho^{\text{out}} = \text{Tr}_1(U_3^2U_2^2U_1^2(tr_0(U_2^1U_1^1(\rho^{\text{in}} \otimes |00\rangle\langle 00|)U_1^{1\dagger}U_2^{1\dagger}) \otimes |000\rangle\langle 000|)U_1^{2\dagger}U_2^{2\dagger}U_3^{2\dagger}).$$

3.3.3 Back-propagation

Similarly to the classical backpropagation algorithm described in section 1.3.3, we need to derive an expression for backpropagating a quantum state through the network. We get a general expression for the backpropagation of a quantum state computing the adjoint of $\mathcal{E}$. We carry out this derivation for educational purposes to further understand how the network behaves. In [15], it is shown how the expression of the adjoint given below simplifies the simulation of DQNNs.

Proposition 3.2. The adjoint for the completely positive map

$$\mathcal{E}^l(X^{l-1}) = \text{Tr}_{l-1}(U^l(X^{l-1} \otimes |0\dots0\rangle_l\langle 0\dots0|)U^{l\dagger})$$

is

$$\mathcal{F}_t^l(X^l) = \text{Tr}_l((\mathbb{1}_{l-1} \otimes |0\dots0\rangle_l\langle 0\dots0|)U^{l\dagger}(t)(\mathbb{1}_{l-1} \otimes X^l)U^l(t)),$$

where $\mathbb{1}_{l-1}$ is the identity matrix acting on the $(l-1)^{th}$ layer, and $|0\dots0\rangle_l\langle 0\dots0|$ is the zero product state of all qubits in layer l.

Proof. Given $|\alpha\rangle, |\beta\rangle$ vectors acting on the l^{th} layer and $|\delta\rangle, |\gamma\rangle$ vectors acting on the $(l-1)^{th}$ layer we take

$$\langle\alpha| \mathcal{E}_t^l(|\delta\rangle\langle\gamma|)|\beta\rangle = \langle\alpha| \text{Tr}_{l-1}(U^l(|\delta\rangle\langle\gamma| \otimes |0\dots0\rangle_l\langle 0\dots0| U^{l\dagger})|\beta\rangle.$$

Recall that given an operator $\rho : H \to H$ and an orthonormal basis $|v_j\rangle_j \in H$ the $\text{Tr}(\rho) = \sum_j v_j^T \rho v_j$. Thus,

$$\begin{aligned}\langle\alpha| \mathcal{E}_t^l(|\delta\rangle\langle\gamma|)|\beta\rangle &= \sum_j \langle\alpha, v_j| U^l(|\delta\rangle\langle\gamma| \otimes |0\dots0\rangle_l\langle 0\dots0| U^{l\dagger}|\beta, v_j\rangle = \\ &= \sum_j \langle\alpha, v_j| U^l(|\delta, 0\dots0\rangle\langle\gamma, 0\dots0| U^{l\dagger}|\beta, v_j\rangle. \quad (3.4)\end{aligned}$$

Now, using theorem 2.40, we rewrite $\mathcal{E}$ as

$$\mathcal{E}_t^l\left(X^{l-1}\right) = \sum_j K_j^l(t)X^{l-1}K_j^{l\dagger}(t).$$

Here K_j^l is the Kraus operator acting between the layers $(l-1)$ and l. We now compute the adjoint of $\mathcal{E}_t^l\left(X^{l-1}\right)$, which leads to

$$\mathcal{F}_t^l\left(X^l\right) = \sum_j K_j^{l\dagger}(t) X^l K_j^l(t).$$

From equation 3.4, follows that $\langle\alpha| K_j^l |\delta\rangle = \langle\alpha, v_j| U^l(t) |\delta, 0\ldots 0\rangle$. Therefore, we get

$$\langle\delta| \mathcal{F}_t^l(|\alpha\rangle\langle\beta|) |\gamma\rangle = \sum_j \langle\delta| K_j^{l\dagger}(|\alpha\rangle\langle\beta|) K_j^l |\gamma\rangle =$$
$$= \sum_j \langle\delta, 0\ldots 0| U^l(t)^\dagger(|\alpha, v_j\rangle\langle\beta, v_j|) U^l(t) |\gamma, 0\ldots 0\rangle .$$

Now, note that $\sum_j |v_j\rangle\langle v_j| = \mathbb{1}_{l-1}$, where $\mathbb{1}_{l-1}$ is the identity matrix acting on the $(l-1)^{th}$ layer. Thus,

$$\langle\delta| \mathcal{F}_t^l(|\alpha\rangle\langle\beta|) |\gamma\rangle = \langle\delta, 0\ldots 0| U^l(t)^\dagger(\mathbb{1}_{l-1} \otimes |\alpha\rangle\langle\beta|) U^l(t) |\gamma, 0\ldots 0\rangle .$$

Note that we can write

$$\langle b|_B O |b\rangle_B = \mathrm{Tr}_B(\mathbb{1}_A \otimes |b\rangle_B \langle b|_B O),$$

given any $|b\rangle_B \in H_B$, and any operator O. Since

$$\mathrm{Tr}_B(\mathbb{1}_A \otimes |b\rangle_B \langle b|_B O) = \sum_v \langle v|_B \mathbb{1}_A \otimes |b\rangle_B \langle b|_B O |v\rangle_B ,$$

where $|v\rangle_B$ are the basis vectors of H_B, and thus

$$\mathrm{Tr}_B(\mathbb{1}_A \otimes |b\rangle_B \langle b|_B O) = \sum_v \langle b|_B |v\rangle_B \otimes \mathbb{1}_A O \langle v|_B |b\rangle_B = \langle b|_B O |b\rangle_B ,$$

where we used the basis representation of a vector, $\sum_v \langle v|_B |b\rangle_B = |b\rangle_B$. Thus, for a general operator X^l

$$\mathcal{F}_t^l(X^l) = \mathrm{Tr}_l((\mathbb{1}_{l-1} \otimes |0\ldots 0\rangle_l \langle 0\ldots 0|) U^{l\dagger}(t)(\mathbb{1}_{l-1} \otimes X^l) U^l(t)).$$

$\square$

3.3.4 Updating the network

Analogously to section 1.3.1, we define a cost function to compare the output quantum state of the network ρ_x^{out} to $|\phi_x^{\text{des}}\rangle$. Naturally, we make use of fidelity defined in section 2.8 to define such a function. We define the cost function as the fidelity of ρ_x^{out} and $|\phi_x^{\text{des}}\rangle$ averaged over the set of S data pairs studied. For the training data, the cost function becomes

$$\mathcal{C}_{\text{tr}} = \frac{1}{S}\sum_{x=1}^{S} F\left(|\phi_x^{\text{tr}}\rangle\langle\phi_x^{\text{tr}}|, \rho_x^{\text{out}}\right) = \frac{1}{S}\sum_{x=1}^{S}\langle\phi_x^{\text{tr}}|\rho_x^{\text{out}}|\phi_x^{\text{tr}}\rangle. \tag{3.5}$$

and for the $N-S$ validation data pairs the cost function

$$\mathcal{C}_{\text{val}} = \frac{1}{N-S}\sum_{x=S+1}^{N}\left\langle\phi_x^{\text{val}}|\rho_x^{\text{out}}|\phi_x^{\text{val}}\right\rangle. \tag{3.6}$$

The goal of the training process is to modify the parameters of the network so that the output quantum state approaches the desired state $|\phi_x^{\text{des}}\rangle$. We aim to maximize the cost function, as the fidelity reaches its maximum value when the two quantum states are equal. In this section, we derive that the network can be updated as

$$U_j^l(t+\epsilon) = e^{i\epsilon K_j^l(t)}U_j^l(t),$$

given a certain matrix K_j^l, whose structure is discussed in proposition 3.5. Let us now introduce a result that will be useful in the proof of proposition 3.5.

Proposition 3.3. The set of tensor products of all Pauli operators $\sigma \equiv \{\sigma_1 = \mathbb{I}, \sigma_x, \sigma_y, \sigma_z\}$ form an orthogonal basis for the vector space of $2^n \times 2^n$ complex matrices.

Recall the definition of Pauli matrices in section 2.7.1. This proposition 3.3 can be illustrated using the following result.

Corollary 3.4. Given a matrix A of dimension $2^n \times 2^n$ we can express the decomposition into Pauli matrices as

$$A = \sum_{\alpha_1,\ldots,\alpha_n=1,x,y,z} A_{\alpha_1,\ldots,\alpha_n}(\sigma_{\alpha_1}\otimes\ldots\otimes\sigma_{\alpha_n}),$$

where $A_{\alpha_1,\ldots,\alpha_n} = \frac{1}{2^n}\operatorname{Tr}((\sigma_{\alpha_1}\otimes\ldots\otimes\sigma_{\alpha_n})\cdot A)$ and $\sum_{i,j} = \sum_i\sum_j$.

Proposition 3.5. The update matrix is

$$K_j^l(t) = \frac{\eta 2^{m_{l-1}} i}{S}\sum_x \operatorname{Tr}_{\text{rest}}\left(M_j^l(x,t)\right),$$

where η is the so-called learning rate, S is the number of training data pairs; the trace acts on the qubits that remain unchanged by the operator U_j^l and

$$\begin{aligned} M_j^l(x,t) = &\Big[U_j^l(t)\dots U_1^1(t) \quad \left(\rho_x^{in} \otimes |0\dots 0\rangle\langle 0\dots 0|\right) U_1^{1\dagger}(t)\dots U_j^l(t), \\ &U_{j+1}^{l}{}^\dagger(t)\dots U_{m_{L+1}}^{L+1}(t)\left(\mathbb{1}_{\text{in, hid}} \otimes \left|\phi_x^{SV}\right\rangle\left\langle\phi_x^{SV}\right|\right) U_{m_{L+1}}^{L+1}(t)\dots U_{j+1}^l(t)\Big]. \end{aligned}$$

Here, the unitary U_j^l affects the $(l-1)^{\text{th}}$ and l^{th} layers, $\mathbb{1}_{\text{in, hid}}$ is the identity matrix acting on the output and all hidden layers, and layer L is the last hidden layer as shown in figure 3.2.

Proof. Suppose we can update the unitaries of the network in the following manner

$$U_j^l(t+\epsilon) = e^{i\epsilon K_j^l(t)}U_j^l(t), \tag{3.7}$$

with a given matrix K_j^l and $\epsilon \in \mathbb{R}$. Recall that the cost function equals 1 when the two compared states are equal. Thus, we aim to maximize the cost function $\mathcal{C}_{\text{tr}}$ at each step. Let us then find the derivative $\frac{d\mathcal{C}_{\text{tr}}}{dt}$

$$\frac{d\mathcal{C}_{\text{tr}}(t)}{dt} = \lim_{\epsilon\to 0}\frac{\mathcal{C}_{\text{tr}}(t+\epsilon)-\mathcal{C}_{\text{tr}}(t)}{\epsilon} = \lim_{\epsilon\to 0}\frac{\frac{1}{S}\sum_{x=1}^S \langle\phi_x^{\text{tr}}|\,\rho_x^{\text{out}}(t+\epsilon)-\rho_x^{\text{out}}(t)\,|\phi_x^{\text{tr}}\rangle}{\epsilon}. \tag{3.8}$$

The expression for $\rho^{\text{out}}(t)$ is known. Therefore, we only need to compute $\rho^{\text{out}}(t+\epsilon)$. Using expression 3.7 and the output expression of equation 3.2, we get

$$\begin{aligned} \rho_x^{\text{out}}(t+\epsilon) = &\text{Tr}_{\text{in,hid}}\Big(e^{i\epsilon K_{m_{L+1}}^{L+1}(t)}U_{m_{L+1}}^{L+1}(t)\dots e^{i\epsilon K_1^1(t)}U_1^1(t)\left(\rho_x^{\text{in}}\otimes|0\dots 0\rangle_{\text{hid,out}}\langle 0\dots 0|\right) \\ &U_1^{1\dagger}(t)e^{-i\epsilon K_1^1(t)}\dots U_{m_{L+1}}^{L+1}(t)e^{-i\epsilon K_{m_{L+1}}^{L+1}(t)}\Big), \end{aligned}$$

where $|0\dots 0\rangle_{\text{hid, out}}\langle 0\dots 0|$ is the zero product state of all qubits in the hidden and output layers. Note that we omit the identity matrices for clearer notation equivalently to example 3.1. Note that we can rewrite the expression above as a Taylor series using

$$f(h+t) = f(t) + \frac{df}{dh}(t)h + \mathcal{O}(h^2).$$

Thus,

$$\rho_x^{\text{out}}(\epsilon+t) = \rho_x^{\text{out}}(t) + \frac{d\rho_x^{\text{out}}}{d\epsilon}(t)\epsilon + \mathcal{O}(\epsilon^2). \tag{3.9}$$

Let us then compute $\frac{d\rho_x^{\text{out}}}{d\epsilon}$ separately. Using the chain rule

$$\begin{aligned} \frac{d\rho_x^{\text{out}}}{d\epsilon}(t) = &\text{Tr}_{\text{in, hid}}(i(K_{m_{L+1}}^{L+1}e^{i\epsilon K_{m_{L+1}}^{L+1}}A_{m_{L+1}}^{L+1} + \dots + K_1^1 e^{i\epsilon K_1^1}A_1^1 \\ &- K_{m_{L+1}}^{L+1}e^{-i\epsilon K_{m_{L+1}}^{L+1}}B_{m_{L+1}}^{L+1} + \dots + K_1^1 e^{-i\epsilon K_1^1}B_1^1)), \end{aligned}$$

where

$$A_j^l = \frac{1}{e^{i\epsilon K_j^l}} \left(e^{i\epsilon K_{m_{L+1}}^{L+1}(t)} U_{m_{L+1}}^{L+1}(t) \dots e^{i\epsilon K_1^1(t)} U_1^1(t) \left(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}} \langle 0\dots0| \right) \right.$$
$$\left. U_1^{1\dagger}(t) e^{-i\epsilon K_1^1(t)} \dots U_{m_{L+1}}^{L+1}(t) e^{-i\epsilon K_{m_{L+1}}^{L+1}(t)} \right),$$

and

$$B_j^l = \frac{1}{e^{-i\epsilon K_j^l}} \left(e^{i\epsilon K_{m_{L+1}}^{L+1}(t)} U_{m_{L+1}}^{L+1}(t) \dots e^{i\epsilon K_1^1(t)} U_1^1(t) \left(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}} \langle 0\dots0| \right) \right.$$
$$\left. U_1^{1\dagger}(t) e^{-i\epsilon K_1^1(t)} \dots U_{m_{L+1}}^{L+1}(t) e^{-i\epsilon K_{m_{L+1}}^{L+1}(t)} \right).$$

We can now substitute these results in equation 3.9

$$\rho_x^{\text{out}}(t+\epsilon) = \rho_x^{\text{out}}(t) + i\epsilon \,\text{Tr}_{\text{in , hid}} \left(\left[K_{m_{L+1}}^{L+1}(t), U_{m_{L+1}}^{L+1}(t) \dots U_1^1(t) \right. \right.$$
$$\left. \left(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}} \langle 0\dots0| \right) U_1^{1\dagger}(t) \dots U_{m_{L+1}}^{L+1}(t) \right] + \dots$$
$$+ U_{m_{L+1}}^{L+1}(t) \dots U_2^1(t) \left[K_1^1(t), U_1^1(t) \left(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}} \langle 0\dots0| \right) \right.$$
$$\left. \left. U_1^{1\dagger}(t) \right] U_2^{1\dagger}(t) \dots U_{m_{L+1}}^{L+1}(t) \right) + \mathcal{O}\left(\epsilon^2\right),$$

where $[A, B] = AB - BA$. Now, substituting this result into equation 3.8 and using that given any $|b\rangle_B \in H_B$, and any operator O

$$\langle b|_B O |b\rangle_B = \text{Tr}_B(\mathbb{1}_A \otimes |b\rangle_B \langle b|_B O),$$

result derived in the proof of proposition 3.2. We get

$$\frac{d\mathcal{C}_{\text{tr}}(t)}{dt} = \frac{1}{S} \sum_{x=1}^{S} \text{Tr} \left((\mathbb{1}_{\text{ in ,hid}} \otimes |\phi_x^{\text{tr}}\rangle\langle\phi_x^{\text{tr}}|) \left(\left[iK_{m_{L+1}}^{L+1}(t), U_{m_{L+1}}^{L+1}(t) \dots U_1^1(t) \right. \right. \right.$$
$$\left. \left(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}} \langle 0\dots0| \right) U_1^{1^\dagger}(t) \dots U_{m_{L+1}}^{L+1\,\dagger}(t) \right] + \dots$$
$$+ U_{m_{L+1}}^{L+1}(t) \dots U_2^1(t) \left[iK_1^1(t), U_1^1(t) \left(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}} \langle 0\dots0| \right) U_1^{1\dagger}(t) \right]$$
$$\left. \left. U_2^{1\dagger}(t) \dots U_{m_{L+1}}^{L+1}(t) \right) \right),$$

where we also used that $\text{Tr}_B(\text{Tr}_A(O)) = \text{Tr}(O)$ if $H_A \times H_B = H$ given any operator O, and $\lim_{\epsilon\to 0} \frac{\epsilon}{\epsilon} = 1$. Note that $\text{Tr}(A+B) = \text{Tr}(A) + \text{Tr}(B)$ and $\text{Tr}(AB) = \text{Tr}(BA)$ for all squared matrices of the same dimension A, B. Thus, we can rearrange the expression above as

$$\begin{aligned}\frac{d\mathcal{C}_{\text{tr}}(t)}{dt} = \frac{i}{S}\sum_{x=1}^{S} \text{Tr}\Big(&\Big[U^{L+1}_{m_{L+1}}(t)\dots\Big(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|\Big)\dots U^{L+1\,\dagger}_{m_{L+1}}(t)\,,\\ &\mathbb{1}_{\text{in,hid}} \otimes \Big|\phi_x^{\text{SV}}\Big\rangle\Big\langle\phi_x^{\text{SV}}\Big|\Big]\, iK^{L}_{m_{L+1}}(t) + \dots\\ &+\Big[U_1^1(t)\Big(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|\Big)U_1^{1^\dagger}(t),\\ &U_2^{1\dagger}(t)\dots U^{L+1\,\dagger}_{m_{L+1}}(t)\Big(\mathbb{1}_{\text{in,hid}} \otimes \Big|\phi_x^{\text{SV}}\Big\rangle\Big\langle\phi_x^{\text{SV}}\Big|\Big)U^{L+1}_{m_{L+1}}(t)\dots U_2^1(t)\Big]\, iK_1^1(t)\Big)\,.\end{aligned}$$

Now, we define

$$\begin{aligned}M^{L+1}_{m_{L+1}}(t) \equiv&\Big[U^{L+1}_{m_{L+1}}(t)\dots\Big(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|\Big)\dots U^{L+1\,\dagger}_{m_{L+1}}(t)\\ &\mathbb{1}_{\text{in,hid}} \otimes \Big|\phi_x^{\text{SV}}\Big\rangle\Big\langle\phi_x^{\text{SV}}\Big|\Big]\,,\\ M_1^1(t) \equiv&\Big[U_1^1(t)\Big(\rho_x^{\text{in}} \otimes |0\dots0\rangle_{\text{hid,out}}\langle 0\dots0|\Big)U_1^{1\dagger}(t)\\ &U_2^{1\dagger}(t)\dots U^{L+1\,\dagger}_{m_{L+1}}(t)\Big(\mathbb{1}_{\text{in,hid}} \otimes \Big|\phi_x^{\text{SV}}\Big\rangle\Big\langle\phi_x^{\text{SV}}\Big|\Big)U^{L+1}_{m_{L+1}}(t)\dots U_2^1(t)\Big]\,,\end{aligned}$$

and are left with

$$\frac{d\mathcal{C}_{\text{tr}}(t)}{dt} = \frac{i}{S}\sum_{x=1}^{S}\text{Tr}\Big(M^{L+1}_{m_{L+1}}(t)K^{L+1}_{m_{L+1}}(t) + \dots + M_1^1(t)K_1^1(t)\Big)\,.$$

We have not yet imposed the maximization of $\frac{d\mathcal{C}_{\text{tr}}(t)}{dt}$. Prior to said maximization, we rewrite the matrix $K_j^l(t)$ as a composition of the Pauli matrices $\sigma \equiv \{\mathbb{1}, \sigma_x, \sigma_y, \sigma_z\}$. Now, any single qubit matrix is a 2×2 matrix that can be rewritten using Pauli matrices, as shown in proposition 3.3. Note that K_j^l is a composition of single qubit matrices acting on all qubits in the $(l-1)^{\text{th}}$ layer and a single qubit in layer l. Thus, it can be rewritten as shown in corollary 3.4

$$K_j^l(t) = \sum_{q_1,\dots,q_{m_{l-1}},k=1,x,y,z}(K_j^l)_{q_1,\dots,q_{m_{l-1}},k}(t)(\sigma_{q_1}\otimes\dots\otimes\sigma_{q_{m_{l-1}}}\otimes\sigma_k), \tag{3.10}$$

where the index q_i is the i^{th} qubit the $(l-1)^{\text{th}}$ layer, k is the single qubit K_j^l acts on in layer l. Let us now impose the maximization of the derivative of the cost function. Note that if we were to maximize the cost function directly, it would yield an extremum of $\pm\infty$. Thus, we use the lagrange multiplier $\lambda \in \mathbb{R}$

$$\max_{(K_j^l)_{q_i,\dots,k}}\left(\frac{d\mathcal{C}_{\text{Tr}}(t)}{dt} - \lambda\sum_{q_1,\dots,q_{m_{l-1}},k=1,x,y,z}(K_j^l)_{q_i,\dots,k}(t)^2\right)$$

to find the maximum. First, we substitute the derivative of the cost function

$$\frac{i}{S}\sum_{x=1}^{S}\mathrm{Tr}\left(M^{L+1}_{m_{L+1}}(t)K^{L+1}_{m_{L+1}}(t)+\ldots+M^{1}_{1}(t)K^{1}_{1}(t)\right)-\lambda\sum_{q_i,\ldots,k=1,x,y,z}\left(K^{l}_{j}\right)_{q_i,\ldots,k}(t)^2,$$

which can be rewritten as

$$\frac{i}{S}\sum_{x=1}^{S}\mathrm{Tr}_{q_i,\ldots,k}\left(\mathrm{Tr}_{\mathrm{rest}}\left(M^{L+1}_{m_{L+1}}(t)K^{L+1}_{m_{L+1}}(t)+\ldots+M^{1}_{1}(t)K^{1}_{1}(t)\right)\right)-\lambda\sum_{q_i,\ldots,k=1,x,y,z}\left(K^{l}_{j}\right)_{q_i,\ldots,k}(t)^2,$$

where $\mathrm{Tr}_{\mathrm{rest}}$ now represents the trace on the qubits which K^l_j doesn't act on. To maximize this expression, let us take the derivative with respect to $(K^l_j)_{q_i,\ldots,k}(t)$ and equal the result to zero. Note that K^l_j depends on $(K^l_j)_{q_i,\ldots,k}(t)$ as shown in 3.10, thus

$$\frac{i}{S}\sum_{x=1}^{S}\mathrm{Tr}_{q_i,\ldots,k}\left(\mathrm{Tr}_{\mathrm{rest}}\left(M^{l}_{j}(t)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k)\right)-2\lambda(K^l_j)_{q_i,\ldots,k}(t)=0.$$

Let us now solve for $(K^l_j)_{q_i,\ldots,k}(t)$

$$(K^l_j)_{q_i,\ldots,k}(t)=\frac{i}{2S\lambda}\sum_{x=1}^{S}\mathrm{Tr}_{q_i,\ldots,k}\left(\mathrm{Tr}_{\mathrm{rest}}\left(M^{l}_{j}(t)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k)\right).$$

Substituting this result in equation 3.10, we get

$$\begin{aligned}K^l_j(t)&=\sum_{q_i,\ldots,k=1,x,y,z}(K^l_j)_{q_1,\ldots,k}(t)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k)\\&=\sum_{q_i,\ldots,k=1,x,y,z}\frac{i}{2S\lambda}\sum_{x=1}^{S}\mathrm{Tr}_{q_i,\ldots,k}\left(\mathrm{Tr}_{\mathrm{rest}}\left(M^{l}_{j}(t)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k).\end{aligned}$$

We now define $\eta=\frac{1}{\lambda}$ and refer to it as the learning rate. Substituting it into the expression above

$$K^l_j(t)=\frac{i\eta}{2S}\sum_{x=1}^{S}\sum_{q_i,\ldots,k=1,x,y,z}\mathrm{Tr}_{q_i,\ldots,k}\left(\mathrm{Tr}_{\mathrm{rest}}\left(M^{l}_{j}(t)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k).$$

We can now rewrite $\mathrm{Tr}_{q_i,\ldots,k}\left(\mathrm{Tr}_{\mathrm{rest}}\left(M^{l}_{j}(t)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k)\right)$ as

$$\sum_{q_i,\ldots,k=1,x,y,z}(\sigma_{q_i}\otimes\ldots\otimes\sigma_k)^{\dagger}\left(\mathrm{Tr}_{\mathrm{rest}}\left(M^{l}_{j}(t)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k)\right)(\sigma_{q_i}\otimes\ldots\otimes\sigma_k).$$

Now using that $\left(\sigma_{q_i} \otimes \ldots \otimes \sigma_k\right)\left(\sigma_{q_i} \otimes \ldots \otimes \sigma_k\right) = (\sigma_{q_i}^2 \otimes \ldots \otimes \sigma_k^2) = (\mathbb{I} \otimes \ldots \otimes \mathbb{I}) = \mathbb{I}_{2^{m_{l-1}+1} \times 2^{m_{l-1}+1}}$, we can rewritte

$$K_j^l(t) = \frac{2^{m_{l-1}+1} i\eta}{2S} \sum_{x=1}^{S} \mathrm{Tr}_{\mathrm{rest}} \left(M_j^l(t)\right) \mathrm{Tr}_{q_i,\ldots,k=1,x,y,z}(\mathbb{I} \otimes \ldots \otimes \mathbb{I}).$$

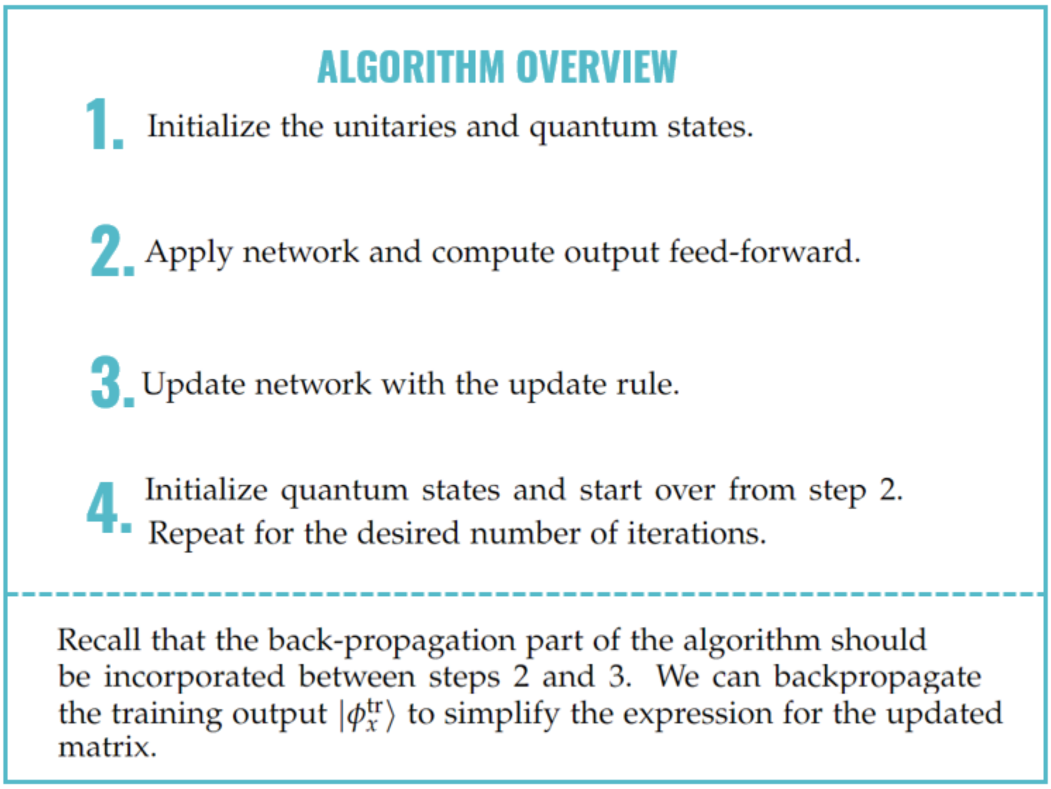
Note that we used $\mathrm{Tr}_{q_i,\ldots,k=1,x,y,z}(\mathbb{I} \otimes \ldots \otimes \mathbb{I}) = \sum_{q_i,\ldots,k=1,x,y,z} (\sigma_{q_i}, \ldots, \sigma_k)^\dagger (\mathbb{I} \otimes \ldots \otimes \mathbb{I})(\sigma_{q_i}, \ldots, \sigma_k)$. Now, $\mathrm{Tr}_{q_i,\ldots,k=1,x,y,z}(\mathbb{I} \otimes \ldots \otimes \mathbb{I}) = \mathrm{Tr}(\mathbb{I} \otimes \ldots \otimes \mathbb{I})$, and $\mathrm{Tr}(A \otimes B) = \mathrm{Tr}(A)\,\mathrm{Tr}(B)$. Lastly, since the trace over the identity on each qubit is two, this yields

$$K_j^l(t) = \frac{2^{m_{l-1}+1} i\eta}{2S} \sum_{x=1}^{S} \mathrm{Tr}_{\mathrm{rest}} \left(M_j^l(t)\right).$$

Thus, the matrix K_j^l exists and can be computed as stated above. This result also proves that the network can be updated following equation 3.7. □

3.3.5 The algorithm, an overview

We have discussed the different steps of the training process. Thus, we can now give a schematic overview of the steps of the training algorithm. This summary will give the reader an overview and a better understanding of the training procedure.



ALGORITHM OVERVIEW

1. Initialize the unitaries and quantum states.

2. Apply network and compute output feed-forward.

3. Update network with the update rule.

4. Initialize quantum states and start over from step 2. Repeat for the desired number of iterations.

Recall that the back-propagation part of the algorithm should be incorporated between steps 2 and 3. We can backpropagate the training output $|\phi_x^{\mathrm{tr}}\rangle$ to simplify the expression for the updated matrix.

3.4 NISQ enforcement

As mentioned in section 2.7.2, the implementation of theoretical models for quantum computers in NISQ devices is not straightforward, and some adjustments must be made to implement such models. First, as shown in [32], we know that any gate can be implemented with several two-qubit unitaries. This result is compelling since we deal with gates acting on a large number of qubits which are challenging to implement and uncommon in most quantum computing libraries. Now, we represent such two-qubit unitaries using the resulting universal set of gates of theorem 2.26 and remark 2.27. These results lead to the implementation of perceptrons, as shown in figure 3.5.

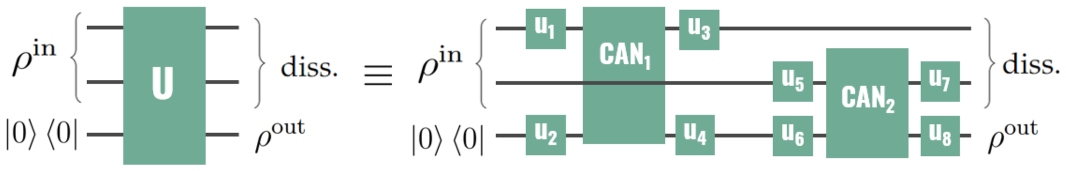


Figure 3.5: NISQ implementation of the unitary U using two-qubit gates and a universal representation shown in remark 2.27.

Let us now describe the implementation of the training algorithm on NISQ devices. We derive an update rule for current quantum devices using gradient descent on the cost function. Let us first rewrite the cost function as

$$\mathcal{C}_{\text{tr}}(w_t) = \frac{1}{S}\sum_{x=1}^{S}\langle\phi_x^{\text{tr}}|\rho_x^{\text{out}}(w_t)|\phi_x^{\text{tr}}\rangle\,,$$

where $w_t = (w_1(t), \ldots, w_n(t))^T$ and n is equal to the number of parameters that describe the quantum circuit. The value of n is computed using that u-gates can be parametrized with $3\sum_{l=1}^{L+1} m_{l-1} + 3m_{L+1}$ parameters, and the CAN-gates with $3\sum_{l=1}^{L+1} m_{l-1}m_l$ parameters. Which leads to

$$n = 3\sum_{l=1}^{L+1} m_{l-1}\left(1+m_l\right) + 3m_{L+1}.$$

Now, using gradient descent to maximize the cost function leads to the new update rule $w_{t+1} = w_t + dw_t$, with $dw_t = \eta\boldsymbol{\nabla}\mathcal{C}_{tr}(w_t)$. Given the gradient computed as

$$\nabla_k\mathcal{C}_{\text{tr}}\left(\omega_t\right) = \frac{\mathcal{C}_{\text{tr}}\left(\omega_t+\epsilon e_k\right)-\mathcal{C}_{\text{tr}}\left(\omega_t-\epsilon e_k\right)}{2\epsilon} + \mathcal{O}\left(\epsilon^2\right),$$

where $e_k^j = \delta_{k,j}$, for $k, j \in \{1, \ldots, n\}$, η is the learning rate, and $\epsilon > 0$ is the step size. An important observation done in [15] is that given this expression for the gradient, we will have to compute the cost function, and thus we will have to apply the entire circuit for $\omega_t + \epsilon e_k$ and $\omega_t - \epsilon e_k$ given every value of k.

Now that we described the mathematical tools used, let us roughly describe the implementation of the training algorithm. First all parameters η,ϵ, and w_0 are initialized. Then, the qubits corresponding to the initial and output states are initialized, i.e., several qubits are initialized as ρ_{in} and the rest as the product state $|0,\ldots,0\rangle\langle 0,\ldots,0|$. Lastly, we initialize the training data output states $|\phi_x^{\text{tr}}\rangle$ and the validation data quantum states $|\phi_x^{\text{val}}\rangle$. Once the initialization process is completed, the training starts. The process is analogous to the classical training described in section 1.3. First initialize, then apply the gates, lastly compare ρ_{out} to the training outputs $|\phi_x^{\text{tr}}\rangle$. Nevertheless, how to compare two states in NISQ devices rests

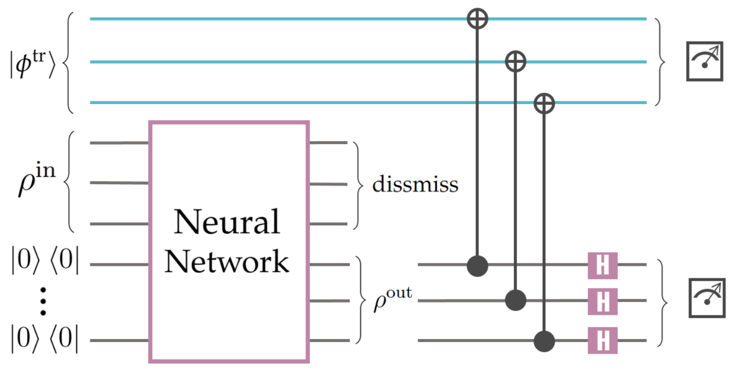


Figure 3.6: Implementation of the neural network followed by a SWAP-test to compute the fidelity between the training data and output quantum state.

unclear. To implement the computation of fidelity, we use an algorithm known as the SWAP-test described in [33]. Such SWAP-test algorithm involves applying a number of given gates as shown in figure 3.6.

NISQ IMPLEMENTATION OVERVIEW

1. Initialize all parameters and quantum states.

2. Compute the network and respective SWAP-test for all parameters $w_t + \epsilon e_k$, and $w_t - \epsilon e_k$ given any k.

3. Compute gradient descent classically.

4. Initialize quantum states and update parameters as $w_{t+1} = w_t + dw_t$.

5. Start over from step 2.

After one epoch is complete, we apply the network and swap test using the validation data instead of the training data. This process allows tracking of the accuracy of the network.

Summary and Conclusions

In this thesis, we have reviewed all aspects of the recently proposed dissipative quantum neural networks from a mathematical perspective. In an effort to produce a comprehensive study, we have first reviewed the basics of classical artificial neural networks and quantum mechanics. In this way, we have structured the discussion emphasizing the parallelism between the classical and quantum neural networks.

We formalized the structure of classical neural networks mathematically and gained a thorough understanding of the underlying principles of these structures. Additionally, the research on the impact of the learning rate on the accuracy of the neural network further enhanced our understanding of the potential of machine learning. We believe that presenting the theoretical insights could lead to facilitating the use of mathematical tools such as optimization and statistical methods to improve the performance of the networks.

We have also introduced the mathematical tools and physical principles needed for understanding the theoretical framework of DQNNs and highlighted the parallelism between the training algorithms of classical neural networks and DQNNs. Additionally, we added examples and extended the proofs of the reviewed work on the pioneering studies of the field. Overall, we aimed to provide a comprehensive overview of the field of quantum neural networks, with a focus on the dissipative quantum neural networks model.

More research is needed in both quantum machine learning and quantum computing. Quantum computers are currently limited in the tasks they can perform and are highly sensitive to their environment. While they have the potential to solve many problems faster than classical computers, many challenges must be overcome before this can be achieved. Quantum machine learning is also affected by such limitations, including the limited availability and scalability of quantum hardware and noise sensitivity.

Some methods discussed in this work have not yet been extensively tested. To continue this research from an applied perspective, one approach could be to test quantum neural networks for simple pattern recognition using quantum encoding.

Appendix A

Impact of the learning rate in the accuracy of neural networks for handwritten digit recognition

Before introducing the structure of the neural network used, let us introduce a few definitions crucial for understanding the network. To perform handwritten digit recognition, we use the MNIST dataset. A set of 60,000 labeled images of handwritten digits as shown in figure A.1. The handwritten numbers range from 0 to 9.

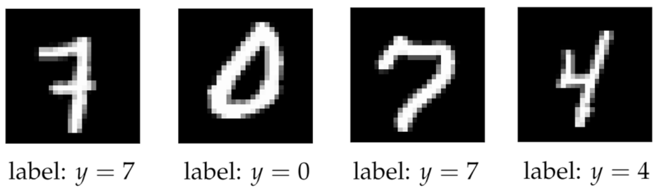


Figure A.1: MNIST images of handwritten digits.

We use the code in [7], which we modify to analyze the impact of changes in the different variables, initialization, and training methods. In this chapter, we show the impact of the learning rate on the accuracy of neural networks.

The code operates as follows; first, we vectorize the MNIST images. MNIST images consist of $28 \times 28 = 784$ pixels. Thus, we represent each image as a vector of dimension 784, whose inputs ranging from 0 to 1 represent how dark a pixel is. 0.0 represents a pure white pixel, and 1.0 a black pixel. Consequently, the input layer has 784 neurons, each input being an entry of the vectorized image. The output layer is comprised of 10 neurons. When the first output neuron indicates a value of approximately 1, the network indicates that the digit is recognized as a 0; if the second output neuron indicates ≈ 1, then the network identifies the handwritten digit as a 1, and so forth.

After vectorizing the images, we randomly partition 50,000 of the 60,000 images into the so-called mini-batches of a specific size. Backpropagation and gradient descent are then applied to each mini-batch. After a full pass over the training set is completed, i.e., after one epoch, we check the accuracy of the neural network. The accuracy is the number of correctly classified images out of 10,000 images different from the training data. For a better determination of the accuracy of the network, the set of 10,000 test images consists of digits written by a different set of people.

Training set size	50,000
Test set size	10,000
Mini-batch size	10
Epochs	30
Hidden neurons	30
Hidden layers	1
Cost function	Quadratic cost function
Weight and bias initialization	N(0,1)
Activation function	Sigmoid

Table A.1: Parameters of the neural network used for the study. N(0,1) stands for a gaussian distribution with mean 0 and standard deviation 1.

Next, we give the parameters used in this analysis in table A.1 and represent the evolution of the accuracy with time for different learning rates in figure A.3. Recall that the learning rate is the parameter η in the gradient descent update rule.

$$\begin{aligned} w_{ij}^{l}(n+1) &= w_{ij}^{l}(n) - \eta\delta_{j}^{l}(n)x_{i}^{l-1}(n) \\ b_{j}^{l}(n+1) &= b_{j}^{l}(n) + \eta\delta_{j}^{l}(n). \end{aligned} \tag{A.1}$$

Note that for a small learning rate, $\eta = 0.1$, the learning process is slower, as expected. On the other hand, for a learning rate that is too large, $\eta = 50$, learning never starts and the accuracy of the network is associated with purely probabilistic behavior. These results can be interpreted as follows. The update rule aims to minimize the cost function and returns the updated parameters in the direction of the steepest descent of the cost function. These descend steps are scaled by a certain amount (the learning rate). The learning rate can be interpreted as the length of each minimization step. Assume our goal is to move from point A to point B, the minimum, in the plot of figure A.2. Intuitively, when the step taken is too large, we risk going past the minimum, and if the learning rate is too low, many iterations are needed to reach the minimum. This behavior is observed in figure A.3.

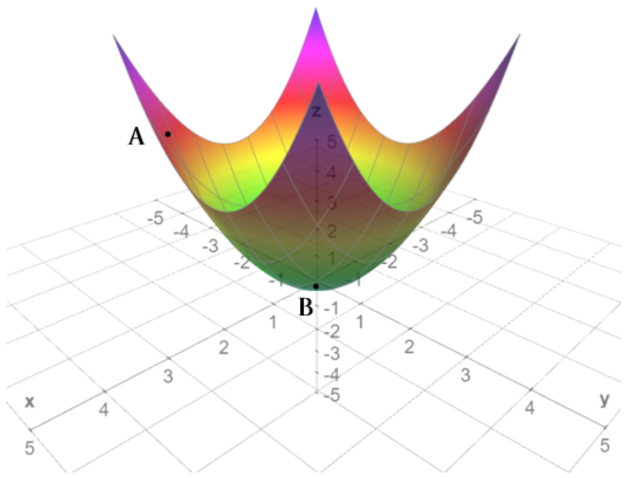


Figure A.2: Plot of a convex function.

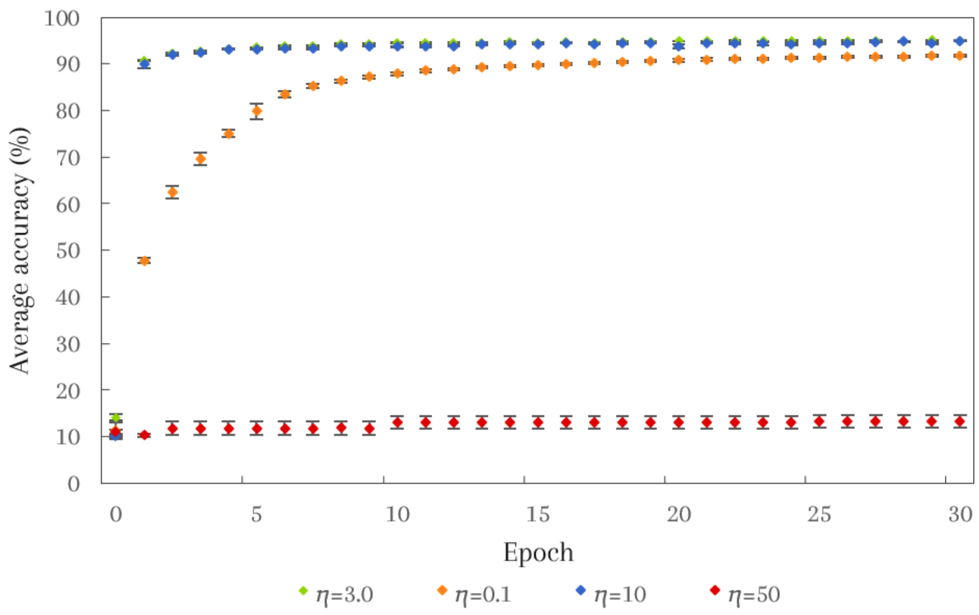


Figure A.3: Accuracy over time given different learning rate values.

Bibliography

[1] S. Aaronson, *Introduction to Quantum Information Science*, Lecture Notes, University of Texas, Austin (2018).

[2] J. Preskill, *Quantum Information*, Lecture Notes, California Institute of Technology, Pasadena (2015).

[3] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press (2010).

[4] G. Allen, *Linear Algebra for Applications*, Lecture Notes, Texas A&M University, Texas (2003).

[5] A. Laghari et al., *A Review on Quantum Computing Trends Future Perspectives*, EAI Endorsed Transactions on Cloud Systems 7(22) (2022).

[6] K. Bharti et al., *Noisy intermediate-scale quantum algorithms*, Rev. Mod. Phys. 94, 015004 (2022).

[7] M. Nielsen, *Neural Networks and Deep Learning* (2019).

[8] J. Berner et al., *The Modern Mathematics of Deep Learning*, arXiv:2105.04026 (2021).

[9] O. Calin, *Deep Learning Architectures: A Mathematical Approach*, Springer International Publishing (2021).

[10] P. Wittek, *Quantum Machine Learning: What Quantum Computing Means to Data Mining*, Elsevier (2014).

[11] D. García, J. Cruz-Benito, and F. García-Peñalvo, *Systematic Literature Review: Quantum Machine Learning and its applications*, arXiv:2201.04093 (2022).

[12] N. Mishra et al., *Quantum Machine Learning: A Review and Current Status*, Springer International Publishing (2020).

[13] A. Dawid et al., *Modern applications of machine learning in quantum sciences*, arXiv:2204.04198 (2022).

[14] K. Beer et al., *Training deep quantum neural networks*, Nature Communications volume 11, 808 (2020).

[15] K. Beer, *Quantum neural networks*, PhD Thesis, Gottfried Wilhelm Leibniz Universitat, Hannover (2022).

[16] V. Bansal, *The Evolution of Deep Learning*, Towards Data Science (2020).

[17] J. K. Hunter and B. Nachtergaele, *Applied Analysis*, World Scientific (2007).

[18] R. D, Schafer, *An Introduction to Nonassociative Algebras*, Dover (1996).

[19] G. Ballesteros, *Quantum algorithms for function optimization*, Bachelor Thesis, Universitat de Barcelona, Barcelona (2021).

[20] R. La Rose, *QuIc Seminar 6*, Lecture Notes, Michigan State University, Michigan (2018).

[21] F. Keller, *Algebraic Properties of Matrices; Transpose; Inner and Outer Product* (PDF), inf.ed.ac.uk. (2020).

[22] N. Yanofsky and M. Mannucci, *Quantum computing for computer scientists*, Cambridge Press (2018).

[23] A. Abbas et al., *Learn Quantum Computation Using Qiskit*, qiskit.org/textbook (2020).

[24] C. Dawson and M. Nielsen, *The Solovay-Kitaev algorithm*, Quantum Information and Computation 0, 000-000 (2005).

[25] J. Zhang et al., *Optimal quantum circuit synthesis from Controlled-U gates*, Phys. Rev. A 69, 042309 (2004).

[26] M. Crommie, *Unitary Evolution, No Cloning Theorem, Superdense Coding*, Lecture Notes, UC Berkeley, Berkeley (2003).

[27] Y. Liang et al., *Quantum fidelity measures for mixed states*, Rep. Prog. Phys. 82, 076001 (2019).

[28] S. Foucart, *Lecture 7: Positive (Semi)Definite Matrices*, Lecture Notes, Drexel University, Philadelphia (2013).

[29] Y. Poon, *Completely Positive Maps in Quantum Information*, Lecture Notes, Iowa State University, Ames (2009).

[30] J. Eisert, *Quantum Information Theory*, Lecture Notes, Freie Universitat Berlin, Berlin (2011).

[31] S. Kak, *Advances in Imaging and Electron Physics*, 94, 259 (1995).

[32] X. Gu, et al., *Fast Multiqubit Gates through Simultaneous Two-Qubit Gates*, PRX Quantum 2, 040348 (2021).

[33] W. Liu et al., *Multi-state Swap Test Algorithm*, arXiv:2205.07171 (2022).