Quantum Perceptrons

Francisco Horta Ferreira da Silva

Thesis to obtain the Master of Science Degree in

Engineering Physics

Supervisor(s):  Prof. João Carlos Carvalho de Sá Seixas
                Prof. Yasser Rashid Revez Omar

Examination Committee
Chairperson: Prof. Jorge Manuel Rodrigues Crispim Romão
Supervisor: Prof. João Carlos Carvalho de Sá Seixas
Member of the Committee: Prof. Luís Henrique Martins Borges de Almeida

November 2018
“Que otros se jacten de las páginas que han escrito; a mí me enorgullecen las que he leído.”

Jorge Luis Borges
Acknowledgments

This work was supported in part by a New Talents in Quantum Technologies scholarship from the Calouste Gulbenkian Foundation.

I would like to thank my advisors Professor João Seixas and Professor Yasser Omar for their guidance. They were the ones who introduced me to scientific research, and any capability that I might now have of producing original work, I owe to them.

Special thanks are also in order to Dr. Mikel Sanz, for his unwavering patience for the barrage of questions that inevitably came to me whenever I was trying to grasp a new paper, as well as for all the time spent reading and rereading my first attempt at a paper of my own.

On a more personal level, I must thank my family. My father, for setting a standard of character and work ethic that I will always aim for and hope to one day reach. My mother, for her attentive ear and sage advice regarding whatever existential matter I confided in her. My sisters, for putting up with me.

Last but not least, the denizens of the P10 study room, as well as its regular visitors, that at times had to endure some hostility to be there with me. During the past 5 years they’ve gone from study group to friend group to support group. If it were not for their incessant motivation disguised as nagging this thesis would probably not be done on time and, at least as importantly, if it were not for their intolerance of my less-than-stellar jokes, the quality of my humour would have long ago dropped below acceptable levels.
Resumo

A recente adição de memristors quânticos às ferramentas disponíveis em circuitos quânticos abriu um novo mundo de possibilidades em computação neuromórfica quântica. Por outro lado, houve uma explosão de interesse em redes neurais quânticas (RNQ). Este trabalho tem como objetivo combinar estas áreas propondo uma abordagem completamente original às RNQs baseada em memristors quânticos. Nesse sentido, propomos modelos para percepções clássicas de uma e várias camadas baseados exclusivamente em memristors clássicos, preenchendo uma lacuna na literatura de memristors no contexto de computação neuromórfica. Desenvolvemos algoritmos de treino baseados no algoritmo de backpropagation. Efectuamos simulações de ambos os modelos e dos algoritmos. Estas mostram que têm um bom desempenho e estão de acordo com o teorema de Minsky-Papert, motivando a possibilidade de construir redes neurais físicas baseadas em memristors. Este trabalho resultou num artigo submetido para publicação. Passando para a quantização destes modelos, temos que memristores quânticos são sistemas quânticos abertos e canais quânticos são uma estrutura matemática extensivamente estudada que descreve o comportamento desses sistemas. Portanto, para estudar RNQs no contexto de sistemas quânticos abertos, propomos um modelo de uma RNQ baseada em canais quânticos, bem como um método de treino baseado em optimização em variedades de Stiefel. Mostramos que a rede é universal no sentido de portas lógicas clássicas, sendo capaz de implementar padrões não linearmente separáveis tais como o XOR. Isto implica que a RNQ é mais poderosa do que o seu equivalente clássico e que não se lhe aplica uma versão quântica do teorema de Minsky-Papert.

Palavras-chave: redes neurais quânticas, memristors, redes neurais, canais quânticos, sistemas quânticos abertos.
Abstract

The recent addition of quantum memristors to the quantum circuit toolbox has opened up a new world of possibilities in neuromorphic quantum computing. On the other hand, interest in quantum neural networks (QNN) has boomed. We aim to combine these areas by launching a completely original approach to QNNs based on quantum memristors. To this end, we propose models for classical single and multilayer perceptrons based exclusively on classical memristors, filling a gap in the literature of memristors for neuromorphic computation. We develop training algorithms based on the backpropagation algorithm. We run simulations of both the models and the algorithms. These show that both perform well and in accordance with Minsky-Papert’s theorem, motivating the possibility of building memristor-based hardware for physical neural networks. This work resulted in a paper that is currently submitted for publication. Moving towards quantizing these models, we have that quantum memristors are open quantum systems, and that quantum channels are a widely studied framework that describes the behaviour of such systems. Therefore, in order to study QNNs in the context of open quantum systems, we propose a model for QNNs based on quantum channels, as well as a training method relying on optimization over Stiefel manifolds. We show that the network is universal in the classical logic gate sense, being capable of implementing non-linearly separable patterns such as the XOR gate. This implies that the QNN is more powerful than its classical equivalent and that it is not subject to a quantum version of Minsky-Papert’s theorem.

Keywords: quantum neural networks (QNNs), memristors, neural networks (NNs), quantum channels, open quantum systems.
# Contents

Acknowledgments ........................................................................................................ v
Resumo ........................................................................................................................ vii
Abstract ......................................................................................................................... ix
List of Tables .................................................................................................................. xiii
List of Figures ............................................................................................................... xv
Acronyms ....................................................................................................................... xvii

1 Introduction 1
  1.1 Motivation ................................................................. 1
  1.2 Topic Overview ......................................................... 2
    1.2.1 Quantum Computing ........................................... 2
    1.2.2 Perceptrons ......................................................... 2
    1.2.3 Memristors ........................................................ 3
    1.2.4 Quantum Memristors .......................................... 4
    1.2.5 Quantum Channels ............................................. 5
  1.3 State of the art ........................................................... 6
    1.3.1 Memristors in learning systems ......................... 6
    1.3.2 Quantum neural networks ................................. 6
  1.4 Objectives ................................................................. 9

2 Perceptrons from Memristors 11
  2.1 Memristor-based Perceptrons ................................... 11
    2.1.2 Memristor-based Multilayer Perceptron ............ 13
  2.2 Simulation results ...................................................... 15
    2.2.1 Single-Layer Perceptron Simulation Results ...... 17
    2.2.2 Multilayer Perceptron Simulation Results ........ 17
    2.2.3 Receiver Operating Characteristic Curves ........ 18
  2.3 Concluding Remarks ................................................ 19

3 Quantum Neural Network 21
  3.1 Classical Network as a Stochastic Matrix ................. 21
List of Tables

3.1 Encoding of the inputs. ................................................. 22
3.2 Encoding of the output. .............................................. 22
3.3 Outputs of the network in terms of the network's weights. ........ 25
3.4 Correspondence between input bits $x_0, x_1$ and input diagonal density matrix $\rho$. . . 27
3.5 Correspondence between result of measuring $\rho_{out}$ and network output. .......... 28
3.6 Diagonal entries of $\rho'$ for each of the possible input pairs, which are encoded ac-
    cording to table 3.4. .................................................... 30
3.7 Correspondence between total state and states of system and environment. .... 31
3.8 Diagonal entries of $\rho'$ for each of the possible input pairs. .................. 32
3.9 The sixteen 2-input, 1-output classical logic gates. ......................... 32
1.1 In a single-layer perceptron (SLP) the inputs $x_i$ are multiplied by their respective weights $w_i$ and added, together with a bias $b$ to form the net input to the SLP, $v$. The output $y$ of the SLP is given by some activation function, $\phi(v)$. .......................... 3

1.2 In a multilayer perceptron (MLP), single-layer perceptrons (SLP) are arranged in layers and connected to each other, with the outputs of the SLPs in the output layer being the outputs of the MLP. Here, each SLP is represented by a disc. ................. 3

2.1 Evolution of the learning progress of our single-layer perceptron (SLP), quantified by its total error, given by Equation (2.19), for the OR, AND and XOR gates over 1000 epochs. The total error of our SLP for the OR and AND gates goes to 0 very quickly, indicating that our SLP successfully learns these gates. The same is not true for the XOR gate, which our SLP is incapable of learning, in accordance with Minsky-Papert’s theorem (23). ................................................................. 17

2.2 Evolution of the learning progress of our multilayer perceptron (MLP), quantified by its total error, given by Equation (2.19) for the OR, AND and XOR gates over 1000 epochs. As can be seen, the total error of our MLP for the these gates approaches 0, indicating that it successfully learns all three gates. ............................................ 18

2.3 ROC curves obtained with the SLP for the OR and XOR gates, and with the MLP for the XOR gate. The thresholds used were $t = 0.3$, 0.5 and 0.7. We can see that the SLP correctly classifies the inputs for the OR gate every time, but it does not perform better than random guessing for the XOR gate, as expected. On the other hand, the MLP correctly classifies the XOR gate inputs every time. ................................. 19

3.1 Single-layer perceptron with inputs $x_1$ and $x_2$, output $y$ and connection weights $w_1$ and $w_2$. ........................................................................................................ 21

3.2 Learning progress of the network on the OR gate, quantified by the cost function over 30 epochs and averaged over 1000 different randomly chosen starting points. The blue line is the median of the 1000 realizations, the top limit is the 95th percentile and the bottom limit is the 5th percentile. As we can see, the value of the cost function goes to zero, indicating that the network is capable of learning the OR gate. ............................................................................ 26
3.3 Learning progress of our QNN on the XOR gate using a steepest descent algorithm in a complex Stiefel manifold, quantified by the cost function over 20 iterations and averaged over 1000 different randomly chosen starting points. The blue line is the median of the 1000 realizations, the top limit is the 95th percentile and the bottom limit is the 5th percentile. As we can see, the value of the cost function goes to zero very quickly, indicating that the network is capable of learning the XOR gate.
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPTP</td>
<td>Completely Positive and Trace-Preserving.</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptron.</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network.</td>
</tr>
<tr>
<td>POVM</td>
<td>Positive-Operator Valued Measure.</td>
</tr>
<tr>
<td>QC</td>
<td>Quantum Computing.</td>
</tr>
<tr>
<td>QNN</td>
<td>Quantum Neural Network.</td>
</tr>
<tr>
<td>SLP</td>
<td>Single-layer Perceptron.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

The introduction of Shor’s algorithm in 1994 (1) showed that quantum computing is not just a field of theoretical interest, but can in fact have extraordinary advantages over traditional paradigms of computation. Since then, interest in quantum computing research has boomed and it is widely believed that the next years will bring about a quantum revolution.

On the other hand, machine learning has had tremendous impact in the past couple of decades, performing remarkably well on tasks as diverse as image recognition, medical diagnosis and natural language processing (2–4). In particular, NNs, a subset of machine learning, have found many applications in a wide range of disciplines.

Taking into account the massive impact that NNs have had, and the exciting possibilities brought by quantum computing, it is natural to wonder if it is possible to combine the two in a QNN, and if this can result in faster algorithms. The concept of QNNs has been around since the 1990s (5) and since then several proposals claiming that name have been put forward (6–9). These vary wildly in scope, but none of these earlier models can be said to be the definitive model of a QNN, either because they fail to incorporate neural computing mechanisms or because they violate quantum theory in some way. Recent proposals (10–14) seem to introduce more adequate QNN models, but it is safe to say that there is not yet a standard model that is widely accepted in the field. There has also been a growing interest in memristors, devices with history-dependent resistance which provide memory effects in form of a resistive hysteresis. Although the first theoretical prediction of memristive behaviour came in 1971 (15), it was not until 2008 that the existence of memristors was demonstrated, at HP Labs (16). This discovery reignited interest in memristor research, and their intrinsic memory features suggest that they might be suitable building blocks for neural computing. Recently, memristors have been added to the quantum circuit toolbox (17), constituting a building block for quantum neural computing, so one might wonder if they can be used in the context of QNNs.
1.2 Topic Overview

1.2.1 Quantum Computing

Quantum computing is a non-traditional paradigm of computation that uses quantum systems to process information. The unique properties of these systems, such as superposition and entanglement, allow for speedups over classical computing in some tasks [18].

The time evolution of a closed quantum system is deterministic and reversible in time. For time-independent Hamiltonians, we can represent the time evolution of the system by an unitary operator, which is known as a quantum gate. The notable exception to the unitarity of quantum computing is that of measurement - when a quantum system in a superposition is measured it collapses into one of the states of the superposition, with probability determined by the amplitude of said state. The outcome of a quantum measurement is then probabilistic and, consequently, so is the output of a quantum algorithm.

The advantage of quantum computing over classical computing lies in the fact that quantum bits, unlike classical bits, can be in a state of superposition of 0 and 1. In the standard, digital model of quantum computing, quantum algorithms aim to employ quantum gates in such a way that interference effects maximize the amplitude of the desired result and minimize the amplitude of all others. Along these lines, quantum algorithms that are believed to be more efficient than their classical counterparts have been proposed in tasks such as database searching, solving systems of linear equations and, most notably due to its impact on cryptography and the exponential speedup it is believed to provide, integer factorization (1, 19, 20).

1.2.2 Perceptrons

NNs are a computational model vaguely inspired by our understanding of how the brain processes information and, in particular, perceptrons are a class of NNs. The perceptron, introduced by Rosenblatt in 1958 [21], was one of the first models for supervised learning. In a perceptron, the inputs \( x_1 \ldots x_n \) are linearly combined with coefficients given by the weights \( w_1 \ldots w_n \), as well as with a bias \( b \) to form the input \( v \) to the neuron (see Fig. 1.1). \( v \) is then fed into a nonlinear function whose output is either 0 or 1. The goal of the perceptron is thus to find a set of weights \( \{ w_i \} \) and bias \( b \) that correctly assigns inputs \( \{ x_i \} \) to one of two predetermined binary classes. Geometrically, the perceptron tries to find a boundary that correctly splits the data into the two classes, with the weights changing the slope of the boundary and the bias its position. Looking at the task this way, it becomes obvious that the bias is fundamental to successful training in many problems, since there are data sets that require moving the boundary to be properly split. The correct parameters for the task of classifying the inputs are found by an iterative training process, such as the delta rule [22]. However, the perceptron is only capable of learning linearly separable patterns, as was shown in 1969 by Minsky and Papert [23]. These limitations triggered a search for more capable models, which eventually resulted in the proposal of the multilayer...
perceptron. These objects can be seen as several layers of perceptrons connected to each other by synapses (see Fig. 1.2). This structure ensures that the multilayer perceptron does not suffer from the same limitations as Rosenblatt’s perceptron. In fact, a multilayer perceptron with at least one hidden layer of neurons and with conveniently chosen activation functions can approximate any continuous function to an arbitrary accuracy [24].

![Image 1.1: In a single-layer perceptron (SLP) the inputs $x_i$ are multiplied by their respective weights $w_i$ and added, together with a bias $b$ to form the net input to the SLP, $v$. The output $y$ of the SLP is given by some activation function, $\phi(v)$.](image1.1)

![Image 1.2: In a multilayer perceptron (MLP), single-layer perceptrons (SLP) are arranged in layers and connected to each other, with the outputs of the SLPs in the output layer being the outputs of the MLP. Here, each SLP is represented by a disc.](image1.2)

There are various methods to train a NN such as a multilayer perceptron. One of the most widespread is the backpropagation algorithm, a generalization of the original delta rule [25]. Artificial NNs such as the multilayer perceptron have proven extremely useful in solving a wide variety of problems [26–28], but they have thus far mostly been implemented in digital computers. This means that we are not profiting from some of the advantages that these networks could have over traditional computing paradigms, such as very low energy consumption and massive parallelization [29]. Keeping these advantages is, of course, of utmost interest, and this could be done if a physical NN was used instead of a simulation on a digital computer. In order to construct such a network, a suitable building block must be found, with the memristor being a good candidate.

### 1.2.3 Memristors

The memristor was first introduced in 1971 as a two-terminal device that behaves as a resistor with memory [15]. The three known elementary circuit elements, namely the resistor, the capacitor and the inductor, can be defined by the relation they establish between two of the four fundamental circuit variables: the current $i$, the voltage $u$, the charge $q$ and the flux-linkage $\phi$. There are six possible combinations of these four variables, five of which lead to widely-known relations: three from the circuit elements mentioned above, and two given by $q(t) = \int_{-\infty}^{t} i(\tau) d\tau$ and $\phi(t) = \int_{-\infty}^{t} u(\tau) d\tau$. This means that only the relation between $\phi$ and $q$ remains to be defined: the memristor provides this missing relation. Despite having been predicted in 1971 using this argument, it was not until 2008 that the existence of memristors was demonstrated at HP Labs [16], which led to a new boom in memristor-related research [30]. In particular, there have
been proposals of how memristors could be used in Hebbian learning systems \([31–33]\), in the simulation of fluid-like integro-differential equations \([34]\), in the construction of digital quantum computers \([35]\) and of how they could be used to implement non-volatile memories \([36]\).

The pinched current-voltage hysteresis loop inherent to memristors endows them with intrinsic memory capabilities, leading to the belief that they might be used as a building block in neural computing architectures \([37–39]\).

### 1.2.4 Quantum Memristors

As was shown in \([17]\), a quantum memristor can be modelled as an open quantum system consisting of a Markovian tunable dissipative environment \([40]\), a weak-measurement protocol \([41]\), and a classical feedback controlling the coupling of the system to the dissipative environment.

The evolution of the circuit quantum state \(\rho\) is then given by:

\[
\dot{\rho} = \dot{\rho}_H + \dot{\rho}_{\text{meas}} + \dot{\rho}_{\text{damp}}^{(\mu)},
\]

where \(\dot{\rho}_H\) is the Hamiltonian term that describes the circuit evolution by itself, with no interaction with the environment, \(\dot{\rho}_{\text{meas}}\) is the term due to the presence of weak measurements and \(\dot{\rho}_{\text{damp}}^{(\mu)}\) contains the dissipative contribution, which depends on the state variable \(\mu\). This equation of state is the quantum analogue of Ohm’s law. If the interaction with the state variable \(\mu\) is done via a measurement scheme, the records of the voltage \(M_V(t)\) govern the state variable dynamics:

\[
\dot{\mu}(t) = f(\mu(t), M_V(t)).
\]

If we shunt an LC circuit with a quantum memristor, the description of the Hamiltonian part requires only one degree of freedom, the top node flux \(\phi\), and its conjugate momentum \(q\), which corresponds to the charge on the capacitor connected to the top node. The time evolution of the Hamiltonian part is then given by:

\[
\dot{\rho}_H = -\frac{i}{\hbar} [H(\phi, q), \rho(t)] dt.
\]

Measuring the voltage applied to the quantum memristor implies monitoring the node charge \(q\), since the charge on the capacitor is determined by the voltage and the capacitance \(C\), i.e., \(q = CV\). Therefore, as follows from the theory of continuous measurements, the state update and the measurement output have the following form, respectively \([41]\):

\[
\begin{align*}
\dot{\rho}_{\text{meas}} &= -\frac{\tau}{q_0^2} [q, [q, \rho(t)]] dt + \frac{2\tau}{q_0^2} \{q, \rho(t)\} - 2 \langle q \rangle \rho(t) dW, \\
M_V(t) &= \frac{1}{C} \left( \langle q(t) \rangle + \frac{q_0^2}{8\tau} \zeta(t) \right).
\end{align*}
\]
A numerical analysis of the evolution of this quantum memristor reveals that it in fact shows hysteretic behaviour. This implies that the system has memory, which hints at the possibility of using quantum memristors as a building block for quantum neural computing.

1.2.5 Quantum Channels

As we mentioned in 1.2.1, the operations on a closed quantum system must be unitary, with the exception of measurement. However, if we are studying an open quantum system, i.e., a quantum system that interacts with its environment, there are three effects that must be taken into account: the unitary time evolution of the system given by Schrödinger’s equation, the interactions with the environment and measurement. It turns out that there is a mathematical formalism that can be used to describe all three effects simultaneously: quantum channels. A quantum channel is a completely positive trace-preserving (CPTP) map acting on the space of density matrices. Suppose that the initial state of our quantum system $A$ is $\rho_A$ and that the initial state of its environment $E$ is $|0\rangle_E$. The initial state of the joint system $AE$ is thus $\rho_A \otimes |0\rangle_E$. Let’s say that the interaction between these two systems is given by a unitary operator $U_{AE}$ acting on both systems and that we only have access to $A$ after the interaction. We then obtain the state of the system $A$, $\sigma_A$, by taking the partial trace over the environment:

$$\sigma_A = \text{Tr}_E \left[U_{AE}(\rho_A |0\rangle_E)U_{AE}^\dagger \right]. \quad (1.6)$$

This is known as the physical representation of a quantum channel. It can be shown that this evolution is equivalent to that of a CPTP map given by the following Kraus operators (42):

$$B_i = \{(I_A \otimes |i\rangle_E) U_{AE} (I_A \otimes |0\rangle_E)\}_i. \quad (1.7)$$

This equivalence is formalized in the Choi-Kraus representation theorem.

**Theorem 1.2.1** (Choi-Kraus representation theorem). Let $|\mathcal{H}_A|$ denote the dimension of the Hilbert space $\mathcal{H}_A$ and let $\mathcal{L}(\mathcal{H}_A)$ denote the set of linear operators in $\mathcal{H}_A$. Let the map $N : \mathcal{L}(\mathcal{H}_A) \rightarrow \mathcal{L}(\mathcal{H}_B)$ be denoted by $N_{A\rightarrow B}$. $N_{A\rightarrow B}$ is linear and CPTP if and only if it has a Choi-Kraus decomposition of the following type:

$$N_{A\rightarrow B}(X_A) = \sum_{l=0}^{d-1} V_l X_A V_l^\dagger, \quad (1.8)$$

where $X_A \in \mathcal{L}(\mathcal{H}_A), V_l \in \mathcal{L}(\mathcal{H}_A, \mathcal{H}_B)$ for all $l \in \{0, ..., d - 1\}$,

$$\sum_{l=0}^{d-1} V_l^\dagger V_l = 1_A, \quad (1.9)$$

where $1_A$ is the identity matrix in $\mathcal{H}_A$ and $d$ is at most $|\mathcal{H}_A| \times |\mathcal{H}_B|$. A proof of this result can be found in (43). We thus have that any operation on a quantum
system, including unitary time-evolution, measurement and interaction with environment, can be represented by Kraus operators using equation 1.8. These operators are subject only to the constraint given by equation 1.9.

1.3 State of the art

1.3.1 Memristors in learning systems

There have been proposals of how memristors can be used in learning systems (31–33), but only regarding Hebbian learning and spike-timing-dependent plasticity (STDP). Hebbian learning is an unsupervised learning rule inspired by Hebbian theory, which proposes an explanation for synaptic plasticity. In simple terms, the theory states that the connection between two neurons strengthens if they fire together and weakens if they fire separately. The process of training a network through Hebbian learning consists of feeding it inputs and increasing the weights of the connections between neurons that have the same output and decreasing the weights between those who do not. STDP is an extension of Hebbian learning that takes causality into account. If two neurons fire at exactly the same time, one cannot have influenced the firing of the other, so in STDP the strength of a connection only increases if the pre-synaptic neuron fires immediately before the post-synaptic neuron. Furthermore, the papers using memristors in the context of STDP are more focused on replicating biological effects than obtaining algorithmical advantages.

This being said, as far as we know, there is as of yet no model for how a NN based exclusively on memristors can be used to perform supervised learning. Such a model is thus the first main result of this thesis and its presentation here is based on work submitted for publication by the author in IEEE Transactions on Neural Networks and Learning Systems. This work is also available online as an arXiv preprint (44).

1.3.2 Quantum neural networks

The main difficulty in introducing a QNN lies in the fact that it tries to combine two fundamentally different areas: quantum mechanics, which is unitary and therefore linear and NNs, which must contain nonlinearities in order to be universal (24). In an extensive review article (45), the authors introduced the following criteria that a meaningful QNN proposal should fulfill:

1. The initial state of the quantum system encodes any binary string of length $N$. The QNN produces a stable output configuration encoding the one state of $2^M$ possible output binary strings of length $M$ which is closest to the input by some distance measure.

2. The QNN reflects one or more basic neural computing mechanisms (attractor dynamics, synaptic connections, integrate and fire, training rules, structure of a NN).

3. The evolution is based on quantum effects, such as superposition, entanglement and interference, and it is fully consistent with quantum theory.
Most attempts made so far failed in conciliating the need for nonlinear activation functions with the linear nature of quantum theory, which means they did not fulfill either 2 or 3. However, models proposed mainly in the last two years seem to be of higher quality, in the sense that they do not violate the above criteria in any obvious way. We will now review the most promising of these models one by one.

In [12], the authors introduce a QNN that can represent labeled data, classical or quantum, and be trained by supervised learning, through a process similar to gradient descent. They derive an expression for the number of two qubit unitaries needed to represent any label function, which can be seen as a quantum representation result analogous to the classical representation theorem by Cybenko [24]. In general, the number of unitaries needed to represent the label function grows exponentially, so there is no advantage over the classical case. As the authors state, it may be that certain functions can be represented more efficiently, but that is still an open problem.

In [10], the authors propose a quantum neuron as follows: let the inputs $x_1, \ldots, x_n \in \{0,1\}$ be linearly combined to form an input $\theta = w_1x_1 + \ldots w_nx_n + b$. Use the state $|x\rangle = |x_1 \ldots x_n\rangle$ as a control state and apply $R_y(2w_1)$, a rotation of $w_1$ generated by the Pauli Y operator, onto an ancilla qubit conditioned on the $i$-th qubit, followed by $R_y(2b)$ on the same ancilla qubit. The second step is to perform a rotation by $R_y(2\sigma(\theta))$ where $\sigma$ is a sigmoid function. This rotation can be approximated by a repeat-until-success (RUS) circuit [46]. By recursively applying the RUS circuit $k$ times a rotation $R_y(2q^k(\theta))$ can be performed, where $q^k$ is the self-composition of $q$ $k$ times. This results in a threshold behaviour on $\theta$: if $\theta > \pi/4$ then we want the output qubit to be as close to $|1\rangle$ as possible. This threshold behaviour is key in neural computing and up until this point no QNN model had been successful in implementing it.

In [11], the authors propose a quantum perceptron intimately related to this quantum neuron. Here, a quantum perceptron is introduced as a two-level system that exists in a superposition of resting and active states, as a nonlinear reaction to a classical or quantum field, i.e.:

$$\hat{U}_j(x_j; f) |0_j\rangle = \sqrt{1 - f(x_j)} |0_j\rangle + \sqrt{f(x_j)} |1_j\rangle .$$

This transformation can be implemented as a SU(2) rotation parameterized by a general input field $x_j$:

$$\hat{U}_j(x_j; f) = \exp \left\{ i\tilde{f}(x_j) \hat{\sigma}_y \right\}$$

The unitary operator $\hat{U}_j(x_j; f)$ is fully defined by an activation function $f(x_j)$, the weights $w_{jk}$ and the thresholds $\theta_j$. The rotation angle $\tilde{f}(x_j) = \arcsin (f(x_j)^{1/2})$ depends nonlinearly on the input field $x_j$. In a feed-forward network setup, the perceptron gate depends on the field $x_j$ generated by neurons in earlier layers, in analogy with what happens in a classical NN. The authors use this fact and go on to prove that a network based on this perceptron is a universal function approximator.

Although these two proposals use different mechanisms to implement their quantum neu-
rons, namely RUS circuits in the first case and rotations in the second, the underlying idea is very similar. Both of them aim to find a way of embedding a classical perceptron in a quantum framework. This is made evident, for instance, in the proof of universality of the quantum perceptron given in (11). The proof is done by first determining the parameters of the classical network that approximates a given function and then encoding those parameters in a qubit circuit. Because of this, a function that can be approximated with an accuracy of $\epsilon$ classically with $M + N\epsilon$ neurons, needs $k \times M + N\epsilon$ quantum neurons for an approximation of the same quality. There is actually a decrease in efficiency when moving the network to the quantum realm, which is due to the fact that these models consist of blueprints of how to encode classical networks in quantum settings, not in actual quantum networks.

In (47), the authors propose how the quantum memristor model introduced in (17) can be used to quantize a simplified version of the Hodgkin-Huxley neuron with one ionic channel in a quantum circuit. The main result is that the area of the hysteresis curve of the quantum neuron is bigger for quantum inputs and, in particular, for entangled quantum inputs. This indicates that there is an advantage in terms of memory for the quantum neuron versus the classical neuron. The main drawback is the difficulty in scaling up the network. Using just one memristor, the network is limited to just one neuron with one ionic channel, i.e., one output, so being able to add more neurons and enrich them is of utmost importance. However, the equations that describe the behaviour of the quantum memristor used for the neurons are extremely complicated, which makes expanding the network unfeasible until these equations are in some way simplified.

In (13), the authors aim to train near-term quantum circuits to distinguish between non-orthogonal quantum states, i.e., to perform quantum state discrimination (48). This is achieved by iterative interactions between a classical device and a quantum processor to discover the parameters of an unknown non-unitary quantum circuit. Numerical simulations show that shallow quantum circuits can be trained to discriminate among various pure and quantum states. The authors establish an analogy between this circuit and a QNN: the unitary evolution of the states in the circuit can be considered as a layer of symmetric fully-connected neurons, and the multiple layers of POVMs provide the necessary nonlinearities. This approach consists of a quantum circuit learning approach for discrimination and classification of quantum data, and it is successful in this sense. However, it is not, nor does it claim to be, a a universal QNN, i.e., a quantum version of a NN that can be applied to all the tasks a classical NN can.

In (49), the authors propose an architecture for NNs unique to quantum optical systems, a quantum optical neural network (QONN). The authors argue that the main features of classical NNs can be directly translated to quantum optical systems. The architecture of the QONN consists of photonic quantum states as inputs, linear optical unitaries as linear transformations and nonlinear layers that act on single sites by adding a constant phase. This network is applied to several problems, such as quantum state generation, quantum gate implementation, hamiltonian simulation, quantum optical autoencoder and quantum reinforcement learning.
It constitutes a powerful simulation tool for the design of quantum optical systems as well as an experimental platform.

In [14], the authors propose how a QNN can be created by building a quantum circuit in a continuous-variable architecture, which is based on encoding quantum information in continuous degrees of freedom such as the amplitudes of the electromagnetic field (50). This circuit contains a layered structure of continuously parameterized gates which is universal for continuous-variable QC. The introduction of non-Gaussian gates provides both the nonlinearity and universality of the network. They also allow it to encode highly nonlinear transformations while remaining completely unitary. The authors go on how to show how to embed a classical network into their quantum formalism and propose quantum versions of various machine learning models. Finally, they apply their model to several different problems with good results, namely the fitting of 1D functions, the generation of quantum states to encode images, data classification and autoencoding. These last two problems were done using a hybrid quantum-classical network.

1.4 Objectives

Having surveyed the current state of the field of quantum machine learning and, in particular, of QNNs, we can see that despite the increase in quantity and quality of attempts in recent years, a standard model that is widely accepted in this field does not yet exist. The goal of this work is thus to take a completely original approach to this problem by proposing a QNN based on quantum memristors. However, as we saw in section 1.2.4, the equations describing the behaviour of the quantum memristors put forward in (17) are extremely complicated, and become even more so when several quantum memristors are allowed to interact. We then have that the task of putting several quantum memristors together to form a network looks daunting. The plan is thus to split it into smaller, more tractable parts.

First, we will propose and validate a model of a classical perceptron made exclusively with classical memristors and adapt a learning algorithm for its training. Taking into account the fact that quantum memristors are open quantum systems, we would like to learn more about QNNs in the context of open quantum systems before proceeding with the quantization of the classical model. Quantum channels constitute a formalism to describe open quantum systems that has been extensively studied in the past couple of decades, so the wealth of results that has been accumulated about them is immediately available to any construction based on this formalism. The second goal of this thesis is thus to propose a model for a QNN based on quantum channels, as well as a training algorithm for it. A detailed study of this model will be performed in order to understand what implications for the QNN come from known results about quantum channels and to determine if there are any mechanisms that may limit the QNN at a fundamental level, such as a quantum generalization of Misnky-Papert’s theorem. Hopefully, the knowledge the study of this model brings about QNNs in the context of open quantum systems will be of use in
the future when trying to use quantum memristors for neuromorphic quantum computation.
Chapter 2

Perceptrons from Memristors

This chapter is based on work submitted for publication by the author in IEEE Transactions on Neural Networks and Learning Systems: "Francisco Silva, Mikel Sanz, João Seixas, Enrique Solano, and Yasser Omar - Perceptrons from Memristors”. It is also available online as an arXiv preprint (44).

In general, a current-controlled memristor is a dynamical system whose evolution is described by the following pair of equations [15]:

\[
\begin{aligned}
V &= R(\vec{\gamma},I)I, \\
\dot{\vec{\gamma}} &= \vec{f}(\vec{\gamma}, I).
\end{aligned}
\]

The first one is Ohm’s law and relates the voltage output of the memristor \( V \) with the current input \( I \) through the memristance \( R(\vec{\gamma}, I) \), which is a scalar function depending both on \( I \) and on the set of the memristor’s internal variables \( \vec{\gamma} \). This dependence of the memristance on the internal variables induces the memristor’s output dependence on past inputs, i.e., this is the mechanism that endows the memristor with memory. The second equation describes the time-evolution of the memristor’s internal variables by relating their time derivative, \( \dot{\vec{\gamma}} \), to an \( n \)-dimensional vector function \( \vec{f}(\vec{\gamma}, I) \), depending on both previous values of the internal variables and the input of the memristor. The internal variables can, in practice, be any physical memristor characteristic over which we have some control. For instance, in [16], the internal variable specifies the distribution of dopants in the device.

2.1 Memristor-based Perceptrons

2.1.1 Memristor-based Single-Layer Perceptron

Our goal is to implement a perceptron and an adaptation of the delta rule to train it using only a memristor. To this end, we use the memristor’s internal variables to store the SLP’s weights. Equation (2.1b) allows us to control the evolution of the memristor’s internal variables and implement
a learning rule. If, for example, we want to implement a SLP with two inputs we need a memristor with three internal variables, two of them to store the weights of the connections between the inputs and the SLP and the other one to store the SLP’s bias weight.

Let us then consider a memristor with three internal state variables, from now on labeled by \( \vec{\gamma} = (\gamma_1, \gamma_2, \gamma_3) \) and in which \( \vec{f} = (f_1, f_2, f_3) \). It could be difficult to externally control multiple internal variables. However, a possible solution is to use several memristors with the chosen requirements and with an externally controlled internal variable each.

In order to understand the form of these functions, we must remember that we expect different behaviours from the perceptron depending on the stage of the algorithm. In the forward propagation stage, the weights must remain constant to obtain the output for a given input. In this phase the internal variables must not change. On the other hand, in the backpropagation stage, we want to update the perceptron’s weights by changing the internal variables. However, it may happen that the update is different for each of the weights, so we need to be able to change only one of the internal variables without affecting the others.

There are thus three different possible scenarios in the backpropagation stage: we want to update \( \gamma_1 \), while \( \gamma_2 \) and \( \gamma_3 \) should not change; we want to update \( \gamma_2 \), while \( \gamma_1 \) and \( \gamma_3 \) should not change, and we want to update \( \gamma_3 \), while \( \gamma_1 \) and \( \gamma_2 \) should not change. To conciliate this with the fact that a memristor takes only one input, we propose the use of threshold-based functions, as well as a bias current \( I_b \), for the evolution of the internal variables

\[
V(t) = g(I, \gamma_1, \gamma_2, \gamma_3), \quad (2.2)
\]

\[
\dot{\gamma}_i = (I - I_b) (\theta(I - I_{\gamma_i}) - \theta(I - (I_{\gamma_i} + a))) + (I + I_b) (\theta(-I - I_{\gamma_i}) - \theta(-I - (I_{\gamma_i} + a))), \quad (2.3)
\]

where \( g \) is an activation function, \( \theta \) is the Heaviside function function, \( I_{\gamma_i} \) is the threshold for the internal variable \( \gamma_i \) and \( a \) is a parameter that determines the dimension of the threshold, i.e., the range of current values for which the internal variables are updated. The first term of the update function can only be non-zero if the input current is positive, whereas the second term can only be non-zero if the input current is negative. If \( I_{\gamma_1}, I_{\gamma_2} \) and \( I_{\gamma_3} \) are sufficiently different from each other and from zero, we can reach the correct behaviour by choosing the memristor’s input appropriately. We thus have that the thresholds and the \( a \) parameter are hyperparameters that must be calibrated for each problem. In the aforementioned construction in which our memristor with three internal variables is constructed as an equivalent memristor, we can also use an external current or voltage control to keep the internal variable fixed. In fact, this is how it is usually addressed experimentally (39, 51–53). Therefore, we can assume that this construction is possible. It is important to note that, in an experimental implementation, this threshold system does not need to be based on the input currents’ intensities. It can, for instance, be based on the use of signals of different frequencies for each of the internal variables or in the codification of the signals meant for each of the internal variables in AC voltage signals. We are now ready to present a learning algorithm for our SLP based on the delta rule, which is described in Algorithm 1.
Algorithm 1 Delta rule for Single-layer Perceptron

Initialization
Set the bias current $I_b$ to 0.
Initialize the weights $w_1, w_2, w_b$.
Set the internal state variables $\gamma_1, \gamma_2, \gamma_3$ to $w_1, w_2$ and $w_b$, respectively.

for d in data do

Forward Pass
Compute the net input to the perceptron:
\[ I = w_1 x_1 + w_2 x_2. \] (2.4)
Compute the perceptron’s output:
\[ V = g(I, \gamma_1, \gamma_2, \gamma_3). \] (2.5)

Backward Pass
Compute the difference $\Delta$ between the target output and the actual output:
\[ \Delta = T - V. \] (2.6)
Compute the derivative of the activation function with respect to the net input, $g'$.

for i in internal variables do
if $\Delta \geq 0$ then
    Set the bias $I_b = I_\gamma$.
else
    Set the bias $I_b = -I_\gamma$.
    Update $\gamma_i$ by inputting $I = \Delta x_i g' + I_b$.
Update the weights by setting them to the updated values of the internal state variables.
Set the bias $I_b = 0$.

In case one wants to generalize this procedure to an arbitrary number of inputs $n$, this can be trivially achieved by using a memristor with $n + 1$ internal variables and adapting Algorithm 1 accordingly.

2.1.2 Memristor-based Multilayer Perceptron

In this model, memristors are used to emulate both the connections and the nodes of an MLP. In principle, the nodes could be emulated by non-linear resistors, but using memristors allows us to take advantage of their internal variable to implement a bias weight, which in some cases proves fundamental for a successful network training.

The equations describing the evolution of the memristor at each node in this model are the same as in the seminal HP Labs paper [16]. We have chosen the experimentally tested set
\[ V(t) = \left( R_{ON} \frac{\gamma(t)}{D} + R_{OFF} \left( 1 - \frac{\gamma(t)}{D} \right) \right) I(t), \] (2.7)
\[ \dot{\gamma} = \left( \mu V \frac{R_{ON}}{D} I(t) - I_\gamma \right) \theta \left( \mu V \frac{R_{ON}}{D} I(t) - I_\gamma \right). \] (2.8)

Here, $R_{ON}$ and $R_{OFF}$ are, respectively, the doped and undoped resistances of the memristor, $D$ and $\mu V$ are physical memristor parameters, namely the thickness of its semiconductor film and its average ion mobility, and $I_\gamma$ is a threshold current playing the same role as the $I_\gamma$ in the model.
for the memristor-based SLP introduced above. Equation (2.7) can be approximated by

\[ V(t) = R_{\text{OFF}} \left(1 - \frac{\gamma(t)}{D}\right) I(t), \]  

(2.9)
since we have that \( \frac{R_{\text{ON}}}{R_{\text{OFF}}} \approx \frac{1}{100} \), as seen in (16). If, for instance, we impose a constant current input \( I \) to the memristor for a time \( t \), the output is given by

\[ V(t) \propto -I^2 t. \]  

(2.10)

This can be achieved in practice by using a current integrator. It is then possible to implement non-linear activation functions starting from Equation (2.7), which is an important condition for the universality of NNs (54).

Looking now at synaptic memristors, their evolution is described by

\[ V(t) = \gamma(t) I(t), \]  

(2.11)

with the evolution of the internal variable being given by equation (2.8).

In synaptic memristors, the internal variable \( \gamma \) is used to store the weight of the respective connection, whereas in node memristors the internal variable is used to store the node’s bias weight.

As explained before, the node memristors are chosen to operate in a non-linear regime, which allows us to implement non-linear activation functions. On the other hand, we choose a linear regime for synaptic memristors, which allows us to emulate the multiplication of weights by signals.

It must be mentioned that Equation (2.8) is only valid for \( \gamma \in [0, D] \), due to reasons related to the physical memristor device, as detailed in (16). If we were to store the network weights in the internal variables using only a rescaling constant \( A \), i.e., \( w = A\gamma \), then the weights would all have the same sign. Although convergence of the standard backpropagation algorithm is still possible in this case (55), it is usually slower and more difficult, so it is convenient to redefine the variable (16) \( D \to D' \) so that the interval of the internal variable in which Equation (2.8) is valid becomes \([-D'/2, D'2]\). Using a rescaling constant \( B \), the network weights can then be in the interval \([-BD'/2, BD'2]\).

The new learning algorithm is an adaptation of the backpropagation algorithm, chosen due to its widespread use and robustness. In our case, the activation function of the neurons is the function that relates the output of a node memristor with its input, as seen in Equation (2.7). The
local gradients of the output layer and hidden layer neurons are respectively given by:

\[
\begin{align*}
\text{Output: } & \quad \delta_k = T_k \phi' \left( \sum_i V_{ik} \right), \quad (2.12) \\
\text{Hidden: } & \quad \delta_k = \phi' \left( \sum_i V_{ik} \right) \sum_l \delta_l w_{kl}. \quad (2.13)
\end{align*}
\]

In Equation (2.12), \( T_k \) denotes the target output for neuron \( k \) in the output layer. In Equations (2.12) and (2.13), \( \phi' \) is the derivative of the neuron’s activation function with respect to the input to the neuron \( \sum_i V_{ik} \). Finally, in Equation (2.13), the sum \( \sum_l \delta_l w_{kl} \) is taken over the gradients of all neurons \( l \) in the layer to the right of the neuron that are connected to it by weights \( w_{kl} \). The update to the bias weight of a node memristor is given by:

\[
\Delta w_k = \eta \delta_k, \quad (2.14)
\]

where \( \eta \) is the learning rate. The connection weight \( w_{ij} \) is updated using \( \Delta w_{ij} = \eta \delta_j V_i \), where \( \delta_j \) is the local gradient of the neuron to the right of the connection, and \( V_i \) is the output of the neuron to the left of the connection.

We count now with all necessary elements to adapt the backpropagation algorithm for our memristor-based MLP, as described in Algorithm 2.

### 2.2 Simulation results

In order to test the validity of our SLP and MLP, we tested their performance on three logical gates: OR, AND and XOR. The first two are simple problems which should be successfully learnt by SLP and MLP, whereas only the MLP should be able to learn the XOR gate, due to Minsky-Papert’s theorem.

The Glorot weight initialization scheme \([56]\) was used for all simulations, as it has been shown to bring faster convergence in some problems when compared to other initialization schemes. In this scheme the weights are initialized according to a uniform distribution with extremal values \(-1\) and \(1\), weighed by \( \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}} \right) \), where \( n_{\text{in}} \) and \( n_{\text{out}} \) are the number of neurons in the previous and following layers, respectively. The data sets used contain 100 randomly generated labeled elements, which were shuffled for each epoch, and the cost function is:

\[
E = \frac{1}{2} (T - O)^2, \quad (2.18)
\]

where \( T \) is the target output and \( O \) the actual output.
Algorithm 2 Backpropagation for Multilayer Perceptron

**Initialization**

- Set the bias current $I_b$ to 0.
- Initialize the weights $\{w_{ij}\}$ and $\{w_{bk}\}$.
- Set the internal variable $\gamma_{ij}$ of each connection memristor $ij$ to the respective connection weight $w_{ij}$.
- Set the internal variable $\gamma_k$ of each connection memristor $k$ to the respective bias weight $w_{bk}$.

for $d$ in data do

**Forward Pass**

for $l$ in layers do

- Compute the output of each connection memristor $ij$ in layer $l$:
  \[ V_{ij}(w_{ij}, I) = w_{ij}I. \] (2.15)

- Sum the outputs of the connection memristors connected to each node memristor $k$ in layer $l$:
  \[ I_{nk} = \sum I_{ik} \] (2.16)

- Compute the node memristor’s output:
  \[ V_k = R_{OFF} \left( 1 - \frac{n_k}{D} + \frac{R_{ON}}{R_{OFF}} \frac{\gamma_k}{D} \right) I_{nk}. \]

**Backward Pass**

for $k$ in output layer do

- Compute the difference $\Delta$ between the target output and the actual output of the node memristor:
  \[ \Delta_k = T_k - V_k. \] (2.17)

- Compute the local gradient of the node memristor using Equation (2.12).

for layer in hidden layers do

  for node in layer do

    Compute the local gradient of node memristor $l$ in layer using Equation (2.13).

    for connection in connections do

      Compute the weight update.
      Set the bias current: $I_b = I_{\gamma_{ij}}$.
      Update the connection memristor’s internal variable by inputting $I = \Delta w_{ij} + I_b$ to it.
      Update the connection’s weight by setting it to the updated value of the respective internal variable.

    for node in nodes do

      Compute the bias weight update according to Equation (2.14).
      Set the bias current: $I_b = I_{\gamma_k}$.
      Update the node memristor’s internal variable by inputting $I = \Delta w_k + I_b$.
      Update the bias weight by setting it to the updated value of the respective internal variable.
2.2.1 Single-Layer Perceptron Simulation Results

For the SLP, a learning rate of 0.1 was used for all tested gates, a value set by trial and error. The metric we used to evaluate the evolution of the network’s performance on a given problem was its total error over an epoch, which is given by Equation (2.19).

\[ E_{\text{total}} = \sum_j E_j = \frac{1}{2} \sum_j (T_j - O_j)^2, \]

where the sum is taken over all elements in the training set. In Fig. 2.1, the evolution of the total error over 1000 epochs, averaged over 100 different realizations of the starting weights, is plotted.

![Graph showing the evolution of the total error over 1000 epochs for OR, AND, and XOR gates.]

Figure 2.1: Evolution of the learning progress of our single-layer perceptron (SLP), quantified by its total error, given by Equation (2.19), for the OR, AND and XOR gates over 1000 epochs. The total error of our SLP for the OR and AND gates goes to 0 very quickly, indicating that our SLP successfully learns these gates. The same is not true for the XOR gate, which our SLP is incapable of learning, in accordance with Minsky-Papert’s theorem [23].

We observe that our SLP successfully learns the gates OR and AND, with the total error falling to 0 within 200 epochs, as expected from a SLP. However, the total error of our SLP for the XOR gate does not go to zero, which means that it is not able to learn this gate, in accordance with Minsky-Papert’s theorem.

2.2.2 Multilayer Perceptron Simulation Results

The structure of the network was chosen following [57]. There, a network with one hidden layer of two neurons is recommended for the case of two inputs and one output. As noted in [57], networks with only one hidden layer are capable of approximating any function, although in some problems, adding extra hidden layers improves the performance. However, the results obtained by employing only one hidden layer are satisfactory, thus there is no need for a more complex network structure. There is also the matter of how many neurons must be employed in the hidden layer. In this case, there is a trade-off between speed of training and accuracy. A network with more neurons in the hidden layer counts with more free parameters, so it will be able to output a more accurate fit, but at the cost of a longer time required to train the network.
A rule of thumb for choosing the number of neurons in the hidden layer is to start with an amount that is between the number of inputs and the number of outputs and adjust according to the results obtained. This leads to two neurons for the hidden layer and, similarly to what happened with the number of hidden layers, the results obtained using two neurons in the hidden layer are sufficiently accurate, so there was no need to try other structures. The learning rates used, which we have chosen through trial and error, are $0.1$ for the OR and AND gates, and $0.01$ for the XOR gate. In Fig. 2.2, the evolution of the total error over 1000 epochs, averaged over 100 different realizations of the starting weights, is plotted.

![Figure 2.2: Evolution of the learning progress of our multilayer perceptron (MLP), quantified by its total error, given by Equation (2.19) for the OR, AND and XOR gates over 1000 epochs. As can be seen, the total error of our MLP for these gates approaches 0, indicating that it successfully learns all three gates.](image)

As was the case for our SLP, our MLP successfully learns the OR and AND gates. In fact, it is able to learn them faster than our SLP, which is a consequence of the larger number of free parameters. Additionally, it is able to learn the XOR gate, indicating that it behaves as well as a regular MLP.

In summary, both memristor-based perceptrons behave as expected. Our SLP is able to learn the OR and AND gates, but not the XOR gate, so it is limited to solving linearly separable problems, just as any other single-layer NN. However, our MLP is not subject to such a limitation and it is able to learn all three gates.

### 2.2.3 Receiver Operating Characteristic Curves

As another measure of the perceptrons’ performance, we show in Fig. 2.3 the receiver operating characteristic (ROC) curves obtained with perceptrons trained for 500 epochs on data sets of size 100. The curves shown were obtained using a SLP trained for the OR gate, a SLP trained for the XOR gate and a MLP trained for the XOR gate, with thresholds of $t = 0.3$, $0.5$ and $0.7$ for each. Again, we see that the SLP is capable of learning the OR gate but not XOR, since it correctly classifies the inputs for OR every time, but its performance is equivalent to random guessing for XOR. We can also see that the MLP is capable of learning the XOR gate, since it
correctly classifies its inputs every time. The learning rates used in training were 0.1 for the SLP on both gates and 0.01 for the MLP on XOR gate, as explained in the previous subsection.

![ROC curves](image)

Figure 2.3: ROC curves obtained with the SLP for the OR and XOR gates, and with the MLP for the XOR gate. The thresholds used were $t = 0.3$, 0.5 and 0.7. We can see that the SLP correctly classifies the inputs for the OR gate every time, but it does not perform better than random guessing for the XOR gate, as expected. On the other hand, the MLP correctly classifies the XOR gate inputs every time.

### 2.3 Concluding Remarks

As we saw in section 2.2, the simulations ran show that the memristor-based perceptrons we proposed perform well and in accordance with Minsky-Papert’s theorem, which motivates the possibility of building memristor-based hardware for a physical NN. They can also show advantages in terms of energy conservation and open up possibilities for other learning systems to be adapted to a memristor-based paradigm, both in the classical and quantum learning realms.

In particular, the success of these models motivates the possibility of developing a QNN based on quantum memristors. However, as we discussed in chapter 1, this task must be split into several smaller tasks, so the next step is to propose a method of how to implement a QNN based on quantum channels, and that is therefore the focus of the following chapter.
Chapter 3

Quantum Neural Network

We will now see how we can take advantage of an equivalence between quantum channels and classical channels to implement a QNN. A classical channel can be represented by a stochastic matrix, so if we encode the weights of a NN in a stochastic matrix and use the equivalence we mentioned, we have reason to believe that we can use a quantum channel as a model for a QNN.

3.1 Classical Network as a Stochastic Matrix

A stochastic matrix is a matrix whose entries are non-negative real numbers that represent probabilities. In particular, a left-stochastic matrix $A$ is a matrix whose entries $\{A_{ij}\}$ fulfill the following constraints:

$$\sum_k A_{kj} = 1, \forall j,$$  \hspace{1cm} (3.1)

$$A_{ij} \geq 0, \forall i, j,$$  \hspace{1cm} (3.2)

i.e., its entries are non-negative and each of its columns sums to 1.

Figure 3.1: Single-layer perceptron with inputs $x_1$ and $x_2$, output $y$ and connection weights $w_1$ and $w_2$.  

21
The naive way to store the weights of a perceptron with two inputs like the one in figure 3.1 in a stochastic matrix would be the following:

\[
y = \begin{bmatrix} w_0 & w_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix},
\]

where \( y \) is the output of the perceptron and \( w_i \) is the weight of the synapse connecting input \( x_i \) to the neuron. However, the stochasticity constraint given by equation 3.2 implies that \( w_0 = w_1 = 1 \), which means that this network would be useless, so we need a different way to store the weights. A possible solution is to encode the two bits of the inputs, \( x_0 \) and \( x_1 \), into four bits, \( x_{00}, x_{01}, x_{10}, \) and \( x_{11} \), and the output bit \( y \) into two bits, \( y_0 \) and \( y_1 \), as shown in tables 3.1 and 3.2. The idea is that the value of \( x_{ij} \) is 1 if \( x_i \) is \( j \) and 0 otherwise. Similarly, the value of \( y_i \) is 1 if \( y \) is \( i \) and 0 otherwise. Using this encoding, we get a 2 × 4 matrix for the NN weights. The network has only two synapses, each of them connecting one of the input bits to the output, therefore there are 2 degrees of freedom, not 8 as one might expect. The relation between the outputs and the inputs is given by the weight matrix \( W \):

\[
\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} w_0 & 1 - w_0 & w_1 & 1 - w_1 \\ 1 - w_0 & w_0 & 1 - w_1 & w_1 \end{bmatrix} \begin{bmatrix} x_{00} \\ x_{01} \\ x_{10} \\ x_{11} \end{bmatrix},
\]

where \( w_0 \) is the weight of synapse connecting the first input to the output and \( w_1 \) is the weight of the synapse connecting the second input to the output. The stochasticity constraints for this weight matrix are:

\[
w_0 + (1 - w_0) = 1,
\]

\[
w_1 + (1 - w_1) = 1,
\]
which result in

\[ w_{0,1} \in [0, 1]. \] (3.7)

This is the most freedom we can have in choosing the parameters of a stochastic matrix.

### 3.1.1 Stochastic Learning

Now that we have established how the weights of the network should be stored in a stochastic matrix, we must find a way to train them, that is, we must find an algorithm to update the weights with the goal of minimizing some cost function \( J \) in such a way that the stochasticity constraints remain fulfilled. Typically, in the context of supervised learning, a gradient descent-based algorithm is used to solve problems like this. However, this method imposes no restriction on the values that the weights can take. This means that even if we start with a stochastic matrix, updating the weights with a gradient descent-based method will violate the stochasticity, both by making entries negative and by making it so that the columns no longer add to 1. We therefore need to go about this in a different way.

The multiplicative update rule is a rule on which a set of algorithms for optimizing cost functions with respect to matrices are based. If we are optimizing a cost function \( J \) with respect to a matrix \( W \), the update rule is given by (58):

\[
W'_{ij} = W_{ij} \frac{\nabla^-_{ij}}{\nabla^+_{ij}},
\] (3.8)

in which \( \nabla = \frac{\partial J}{\partial W} = \nabla^+ - \nabla^- \). \( \nabla^+ \) is the positive part of the gradient and \( \nabla^- \) the absolute value of its negative part. The cost function is non-increasing under this update rule and it is invariant if and only if \( W \) is at a stationary point of the cost function (58). This rule also guarantees that the entries of the matrix remain non-negative. It does not, however, guarantee that the columns sum to 1. The naive solution would be to normalize the columns at the end of each iteration of the learning process, but this has been shown to raise convergence issues (59), so we need a more sophisticated solution. One such solution is to use the reparameterization method (59).

Reparameterization is a method for learning with stochastic matrices that imposes stochasticity as a hard requirement, as we will now see. Again, let \( W \) be the stochastic matrix that stores the weights of our network:

\[
W = \begin{bmatrix}
w_0 & 1-w_0 & w_1 & 1-w_1 \\
1-w_0 & w_0 & 1-w_1 & w_1
\end{bmatrix}.
\] (3.9)

\( W \) is reparameterized as a non-stochastic matrix \( U \) in the following manner:

\[
W_{ij} = \frac{U_{ij}}{\sum_k U_{kj}}.
\] (3.10)
Taking into account the constraints on $W$ shown in equation 3.9, we can see that $U$ will have the following form:

$$U = \begin{bmatrix}
    u_{00} & 1 - u_{00} & u_{02} & 1 - u_{02} \\
    u_{10} & 1 - u_{10} & u_{12} & 1 - u_{12}
\end{bmatrix}. \quad (3.11)$$

The learning rule for $U$ is analogous to the multiplicative update rule for $W$ shown in equation 3.8:

$$U'_{ij} = U_{ij} \left( \frac{\nabla U}{\nabla U}_{ij} \right) = U_{ij} \left( \nabla U_{ij} + \sum_k \nabla k_j W_{kj} \right), \quad (3.12)$$

where $\nabla U = \frac{\partial J}{\partial U}$. Each use of this update rule is followed by a column normalization of $W$, i.e.:

$$W'_{ij} = \frac{U'_{ij}}{\sum_k U'_{kj}}. \quad (3.13)$$

This guarantees that the stochasticity of $W$ is maintained during all steps of the learning process.

All that is left is to define the cost function $J$. As can be seen in table 3.2, although we give two bits as inputs to the network, the information about the input can be encoded in just one. Therefore, we will make $J$ depend only on $y_0$ and thus apply the training process we described to just the first line of the weight matrix. The cost function used was the square of the error:

$$J(W) = (t_0 - y_0)^2, \quad (3.14)$$

where $t_0$ is the target output for a given set of inputs and $y_0$ is the actual network output, given by:

$$y_0 = \begin{cases} 
1 & \text{if } \sum_i w_{0i} x_i > l, \\
0 & \text{otherwise},
\end{cases} \quad (3.15)$$

for some threshold $l$. We also have that the even-numbered columns depend on the odd-numbered columns, so only the odd-numbered columns will be updated according to the rule given in equation 3.13. The others will be updated in such a way that the restrictions on the weight matrix shown in equation 3.9 are fulfilled. We are now ready to present a learning algorithm for our network based on the reparameterization algorithm, which is shown in algorithm 3.

### 3.1.2 Simulation Results

In order to test the validity of this network and learning algorithm, we tested its performance on two logical gates: OR and XOR. The first is a linearly separable problem which should be successfully learnt, whereas the network should not be capable of learning the XOR gate, due to Minsky-Papert’s theorem. First, we will check if the network is capable of implementing the OR gate at all, i.e., if there is a set of weights $w_0$ and $w_1$ and a threshold $l$ that result in the network
Algorithm 3 Reparameterization for Neural Network

**Initialization**
Initialize $U$ randomly subject to the constraints of equation 3.11.
Initialize $W$ using equation 3.10.

**for** $d$ in data **do**
Compute the output of the network according to equation 3.4.
Compute the cost function according to equation 3.14.

**for** $j$ in even columns **do**
Compute the derivative of $J$ with respect to $w_{0j}$.

**for** $j$ in even columns **do**
Update $u_{0j}$ according to equation 3.12.
Update $w_{0j}$ according to equation 3.13.

**for** $j$ in odd columns **do**
Set $u_{0j} = 1 - u_{0j-1}.$
Set $w_{0j} = 1 - w_{0j-1}.$

**for** $j$ in columns **do**
Set $w_{1j} = 1 - w_{0j}$.

giving the correct output for each pair of inputs. To do so, we will express the OR gate as a set of inequalities and see if it is possible to simultaneously satisfy all of them. Using the encodings given in tables 3.1 and 3.2, as well as the relation between output and input in the network shown in equation 3.4, we get the following table:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{00}$ $x_{01}$ $x_{10}$ $x_{11}$</td>
<td>$y_0$ $y_1$</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>$w_0 + w_1$ $2 - w_0 - w_1$</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>$1 + w_0 - w_1$ $1 - w_0 + w_1$</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>$1 - w_0 + w_1$ $1 + w_0 - w_1$</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>$2 - w_0 - w_1$ $w_0 + w_1$</td>
</tr>
</tbody>
</table>

Table 3.3: Outputs of the network in terms of the network’s weights.

We can thus conclude that, for the network to implement the OR gate, the following set of inequalities must be satisfied:

\[
\begin{align*}
  w_0 + w_1 &< 2l, \\
  w_1 - w_0 &> 2l - 1, \\
  w_0 - w_1 &> 2l - 1, \\
  w_0 + w_1 &< 2 - 2l, \\
  0 &\leq w_0 \leq 1, \\
  0 &\leq w_1 \leq 1.
\end{align*}
\] (3.16) (3.17) (3.18) (3.19) (3.20) (3.21)

Solving this system, it can be seen that are several sets of weights and threshold that satisfy all conditions. Now that we know that there are solutions, we want to check if the reparameterization algorithm converges to them. To do so, we coded an implementation of the algorithm. One of the values of threshold that allows for solutions to the set of inequalities above is $l = 0.25$, so in the simulation we set the target for $w_x$ at 0.25 if the target output was 1 and 0 otherwise.
measure of the network’s performance is the cost function, given by:

\[
E = (T - O)^2
\]  

(3.22)

where \( E \) is the error, \( T \) is the target and \( O \) is the network output. The evolution of the cost function of the network for the OR gate over 30 epochs and averaged over 1000 different randomly chosen starting points is shown in figure 3.2. We can see that it goes to 0, so the training process succeeds and the network is capable of implementing the OR gate.

Figure 3.2: Learning progress of the network on the OR gate, quantified by the cost function over 30 epochs and averaged over 1000 different randomly chosen starting points. The blue line is the median of the 1000 realizations, the top limit is the 95th percentile and the bottom limit is the 5th percentile. As we can see, the value of the cost function goes to zero, indicating that the network is capable of learning the OR gate.

Let’s now see if it is possible for the network to learn the XOR gate. Proceeding as we did for the OR gate, we can reduce the XOR gate to a set of inequalities, equations 3.23 to 3.28.

\[
w_0 + w_1 < 2l
\]  

(3.23)

\[
w_0 - w_1 > 2l - 1
\]  

(3.24)

\[
w_1 - w_0 > 2l - 1
\]  

(3.25)

\[
w_0 + w_1 < 2 - 2l
\]  

(3.26)

\[
0 \leq w_0 \leq 1
\]  

(3.27)

\[
0 \leq w_1 \leq 1
\]  

(3.28)

There is no set of weights and threshold that satisfies all of these inequalities simultaneously, so the network is incapable of implementing the XOR gate. This is what we expect from a NN with no hidden layers.
In summary, the network does not seem to lose capabilities by having its weights encoded into a stochastic matrix. It can learn linearly separable problems, but is incapable of learning non-linearly separable ones, just as a regular NN with no hidden layers.

3.2 Quantum Neural Network

3.2.1 Quantum-classical Equivalence

We will now use the correspondence between quantum and classical channels the authors show in (60) to lay the foundations for our QNN. Let $W = (a_{ij})$ be a left-stochastic matrix, i.e.:

$$
\sum_{k=1}^{\text{dim}} a_{kj} = 1 \forall j \in [1, \text{dim}],
$$

$$
a_{ij} \geq 0 \forall i, j.
$$

Let the action of the quantum channel $E_A$ be given by the Kraus operators:

$$
A_{i,j} = \sqrt{a_{i,j}} |i\rangle \langle j|.
$$

Let $\rho$ be a diagonal density operator with entries $\rho_{i,j} = \delta_{i,j}p_i \geq 0$. The fact that $\rho$ is a density operator implies that $p_i$ are real numbers with $0 \leq p_i \leq 1$ and $\sum_i p_i = 1$. For such an operator, $E_A$ has the property that $\rho' := E_A(\rho)$ is also diagonal with $\rho'_{i,j} = \delta_{i,j}p'_i$ and $\rho' = Wp$, i.e., the quantum channel $E_A$ reduces to the classical case when applied to diagonal density operators. In order to translate the 2-input, 1-output NN structure to the quantum realm, we must use a $4 \times 4$ input density matrix $\rho_{\text{in}}$ and a $2 \times 2$ output density matrix $\rho_{\text{out}}$. Refer to appendix A for an explanation of why a lower-dimensional matrix cannot be used. The input given to the quantum channel is thus the $4 \times 4$ diagonal matrix $\rho_{\text{in}}$:

$$
\rho_{\text{in}} = \begin{bmatrix}
\rho_{00} & 0 & 0 & 0 \\
0 & \rho_{11} & 0 & 0 \\
0 & 0 & \rho_{22} & 0 \\
0 & 0 & 0 & \rho_{33}
\end{bmatrix}.
$$

A simple encoding between the inputs bits $x_0, x_1$ and $\rho_{\text{in}}$ is the one shown in table 3.4. The

<table>
<thead>
<tr>
<th>Classical</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>$x_1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.4: Correspondence between input bits $x_0, x_1$ and input diagonal density matrix $\rho$. 

27
output of the network, \( \rho_{out} \), is given by:

\[
\rho_{out} = \mathcal{E}(\rho_{in}) = \sum_{i,j} A_{i,j} \rho_{in} A_{i,j}^\dagger = \begin{bmatrix} \rho'_{00} & 0 \\ 0 & \rho'_{11} \end{bmatrix},
\]

(3.33)

where:

\[
\begin{bmatrix} \rho'_{00} \\ \rho'_{11} \end{bmatrix} = \begin{bmatrix} \rho_{00}w_0 + \rho_{11}(1 - w_0) + \rho_{22}w_1 + \rho_{33}(1 - w_1) \\ \rho_{00}(1 - w_0) + \rho_{11}w_0 + \rho_{22}(1 - w_1) + \rho_{33}w_1 \end{bmatrix}.
\]

(3.34)

Finally, we have to establish an equivalence between \( \rho_{out} \) and the output of the network. A simple possibility is to equate the result of measuring \( \rho_{out} \) with the network’s output, as shown in table 3.5. In summary, using the equivalence between quantum and classical channels given in (60), using the encoding of the inputs into \( \rho_{in} \) given in table 3.4 and the correspondence between the result of measuring \( \rho_{out} \) and the network output in table 3.5, we obtain a quantum channel that implements a classical network.

### Training the Quantum Neural Network

Now that we have defined the structure of the QNN, we must establish how the training is done. To do so, we will introduce an adaption of the reparameterization algorithm, given in algorithm 3, to the QNN we just defined.

Keeping the definitions for the Kraus operators \( A_{i,j} \) and the classical stochastic matrix \( W \) given in the previous section, we have that optimizing \( A_{i,j} \) corresponds to optimizing \( W \) classically. The stochasticity constraints on \( W \), given in equations 3.29 and 3.30, result in the following constraints on the Kraus operators:

\[
\sqrt{a_{ij}} \geq 0, \forall i, j,
\]

(3.35)

\[
\sum_k A^2_{k,j} = 1, \forall j.
\]

(3.36)

Due to the stochasticity constraints, each Kraus operator depends on some of the others, namely the ones with non-zero entries on the same column as its non-zero entry. Therefore, the training cannot be done for each of the operators independently. We will instead optimize the matrix given by the sum of the square of the Kraus operators, \( S = \sum_{i,j} A^2_{i,j} \). Note that this matrix is the same as \( W \). In order to do the training, we define an auxiliary matrix \( V \) through equation 3.10. Using the update rule given in equation 3.12 to update \( S \) and doing a column normalization after each iteration of the training process, we arrive at the update rule for each entry of the Kraus operators:
\[ A'_{ij} = \sqrt{S'_{ij}} \]
\[ = \sqrt{\frac{V'_{ij}}{\sum_k V'_{kj}}} \tag{3.37} \]

This learning algorithm is a direct adaptation of the algorithm shown in algorithm 3, with just an extra step to update the Kraus operators, so the performance of this QNN should be identical to that of the classical network we saw in section 3.1.

### 3.2.2 Generalization

We will now remove the stochasticity restriction we had imposed on the Kraus operators to obtain the link between classical and quantum channels and we will consider a completely general quantum channel. This channel maps a density matrix in \( \mathbb{C}^4 \) to a density matrix in \( \mathbb{C}^2 \), so, as a result of theorem 1.2.1, it is defined by a maximum of 8 Kraus operators, \( A_0 \) through \( A_7 \):

\[
A_0 = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \end{bmatrix}, \quad A_7 = \begin{bmatrix} h_{00} & h_{01} & h_{02} & h_{03} \\ h_{10} & h_{11} & h_{12} & h_{13} \end{bmatrix}.
\tag{3.38}

A quantum channel is a CPTP map so, again in accordance with theorem 1.2.1, these operators are subject to the following constraint:

\[
\sum_k A_k^\dagger A_k = \mathbb{1}_4,
\tag{3.39}
\]

where \( \mathbb{1}_4 \) is the identity matrix of dimension 4. Besides this constraint, the operators will also be subject to conditions analogous to the ones in the classical network, which arise from the fact that the input is 4-dimensional, the output is 2-dimensional, but the network has only two synapses. Essentially, we want to translate the restrictions on the weight matrix made evident in equation 3.9 to the Kraus operators. To do this, it is useful to gain an intuition of the effect this quantum channel has on its input. The action of a quantum channel on an input density matrix \( \rho \) is given, in terms of its Kraus operators, by:

\[
\rho' = E_A(\rho) = \sum_k A_k \rho A_k^\dagger,
\tag{3.40}
\]

where \( \rho' \) is the output density matrix. We will establish an equivalence between the result of measuring \( \rho' \) and the network output just as we did for the ‘classical’ quantum case (see table 3.5). If we encode the inputs according to table 3.4, the values of each entry in the main diagonal of \( \rho' \) for each possible combination of inputs are given in table 3.6.

Comparing equation 3.4 with table 3.6, we can establish a correspondence between the weights of the classical stochastic and the Kraus operators of the generalized quantum channel. In doing so, we find conditions for the operators that are analogous to the conditions for the classical weights given in equations 3.5 and 3.6. The set of conditions for the Kraus operators emerging from this analysis are given in equations 3.41 to 3.44.
Table 3.6: Diagonal entries of $\rho'$ for each of the possible input pairs, which are encoded according to table 3.4.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>$\rho'_{00}$</th>
<th>$\rho'_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>$\sum_{i=0}^{n}</td>
<td>i_{00}</td>
<td>^2$</td>
</tr>
<tr>
<td>0 1</td>
<td>$\sum_{i=0}^{n}</td>
<td>i_{01}</td>
<td>^2$</td>
</tr>
<tr>
<td>1 0</td>
<td>$\sum_{i=0}^{n}</td>
<td>i_{02}</td>
<td>^2$</td>
</tr>
<tr>
<td>1 1</td>
<td>$\sum_{i=0}^{n}</td>
<td>i_{03}</td>
<td>^2$</td>
</tr>
</tbody>
</table>

The constraints are imposed on the set of all Kraus operators as a whole, which means that the equivalence is being established between the stochastic matrix and the set of the operators, not each operator individually. This is done because the action of the channel is represented by all of the operators, it cannot be reduced to just one of them, just as the action of the classical network is represented by the whole weight matrix.

Writing down explicitly the conditions that result from equation 3.39, we get:

\[ \sum_{i=0}^{n} |i_{00}|^2 = 1 - \sum_{i=0}^{n} |i_{10}|^2 \] (3.41) \[ \sum_{i=0}^{n} |i_{01}|^2 = 1 - \sum_{i=0}^{n} |i_{11}|^2 \] (3.42)

\[ \sum_{i=0}^{n} |i_{02}|^2 = 1 - \sum_{i=0}^{n} |i_{12}|^2 \] (3.43) \[ \sum_{i=0}^{n} |i_{03}|^2 = 1 - \sum_{i=0}^{n} |i_{13}|^2 \] (3.44)

We note that the conditions coming from the diagonal elements of 1.9 are the same as the ones in equations 3.41 to 3.44, so equations 3.45 to 3.54 contain all the restrictions to which the Kraus operators are subjected.
Unitary representation

We would also like to know the restrictions on the unitary representation of the quantum channel, i.e., the unitary operator that acts on the closed composite system consisting of the system of study and the environment. In this framework, the action of the quantum channel is given by:

\[
\rho' = \mathcal{E}(\rho) = \text{Tr}_{env} \left[ U (\rho_0 \otimes |j\rangle \langle j|) U^\dagger \right],
\]

where \( \rho_0 \) is the initial state of the system, \(|j\rangle \langle j|\) is the initial state of the environment, \(\rho'\) is the final state of the system and \(U\) is the unitary that acts on the composite system. Writing the trace over the environment explicitly, equation 3.55 becomes:

\[
\rho' = \sum_i \langle i | U | j \rangle \rho_0 \langle j | U^\dagger | i \rangle,
\]

where \(\{|i\rangle\}\) is an orthonormal basis of the environment. We want to write out \(\rho'\) in terms of the elements of \(U\), \(\rho\), the initial state of the composite system is given by:

\[
\rho = \rho_0 \otimes |j\rangle \langle j|.
\]

It is, in our case, a \(4 \times 4\) diagonal density matrix. The correspondence between the input bits \(x_0, x_1\) and the diagonal entries of \(\rho\) is the same as the one used in the Kraus representation (see table 3.4). We need to figure out which combinations of \(\rho_0\) and \(|j\rangle \langle j|\) result in the desired initial states \(\rho\). Writing down \(\rho_0\) as follows:

\[
\rho_0 = \begin{bmatrix}
\rho_{00} & \rho_{01} \\
\rho_{10} & \rho_{11}
\end{bmatrix},
\]

we obtain the desired combinations, as shown in table 3.7. In order to write down the output of the quantum channel, we write the elements of the unitary \(U\) explicitly:

\[
U = \begin{bmatrix}
u_0 & u_1 & u_2 & u_3 \\
u_4 & u_5 & u_6 & u_7 \\
u_8 & u_9 & u_{10} & u_{11} \\
u_{12} & u_{13} & u_{14} & u_{15}
\end{bmatrix},
\]

Table 3.7: Correspondence between total state and states of system and environment.
where $u_0$ through $u_{15}$ are complex numbers that are subject to restrictions coming from the unitarity of $U$. Taking all of this into account, the diagonal entries of $\rho'$ are given by:

\[
\begin{array}{cccc}
\text{Inputs} & \text{Outputs} \\
x_0 & x_0 & \rho'_{00} & \rho'_{11} \\
0 & 0 & |u_0|^2 + |u_4|^2 & |u_8|^2 + |u_{12}|^2 \\
0 & 1 & |u_2|^2 + |u_6|^2 & |u_{10}|^2 + |u_{14}|^2 \\
1 & 0 & |u_1|^2 + |u_5|^2 & |u_9|^2 + |u_{13}|^2 \\
1 & 1 & |u_3|^2 + |u_7|^2 & |u_{11}|^2 + |u_{15}|^2 \\
\end{array}
\]

Table 3.8: Diagonal entries of $\rho'$ for each of the possible input pairs.

### 3.2.3 Capabilities of the QNN

Now that we have some understanding of how the network behaves, the next step is to look into its capabilities and limitations. One natural question to ask is if it is universal in the classical sense, i.e., if it is capable of implementing every classical 2-input, 1-output logic gate. There are sixteen of them, as shown in table 3.9. We start by checking if there is a set of Kraus operators that implements each of these gates, starting with gate 7, because it corresponds to the XOR gate. If there is a set of Kraus operators that implements it, we have that there is no quantum version of Minsky-Papert’s theorem, which means that our QNN is, in a sense, more powerful than a classical NN with no hidden layers.

Using the correspondence between measurement result and output given in table 3.5 and the values of the diagonal entries of the output density matrix given in table 3.6, we get the following conditions for successful implementation of gate 7:

\[
\sum_{i=0}^{h} |i_{00}|^2 = 1, \quad (3.60) \quad \sum_{i=0}^{h} |i_{10}|^2 = 0, \quad (3.61) \quad \sum_{i=0}^{h} |i_{01}|^2 = 0, \quad (3.62) \quad \sum_{i=0}^{h} |i_{11}|^2 = 1, \quad (3.63)
\]
\[
\sum_{i=0}^{h} |i_{02}|^2 = 0, \quad (3.64) \quad \sum_{i=0}^{h} |i_{12}|^2 = 1, \quad (3.65) \quad \sum_{i=0}^{h} |i_{03}|^2 = 1, \quad (3.66) \quad \sum_{i=0}^{h} |i_{13}|^2 = 0. \quad (3.67)
\]
These imply:

\[ i_{10} = 0, \forall i \in [a, h], \]  
\[ i_{01} = 0, \forall i \in [a, h], \]  
\[ i_{02} = 0, \forall i \in [a, h], \]  
\[ i_{13} = 0, \forall i \in [a, h]. \]  

(3.68)  
(3.69)  
(3.70)  
(3.71)

So equations 3.49, 3.50, 3.53 and 3.54 are satisfied. Equations 3.51 and 3.52 become:

\[ \sum_{i=a}^{b} i_{03}i_{00}^* = 0, \]  
\[ \sum_{i=a}^{b} i_{12}i_{11}^* = 0. \]  

(3.72)  
(3.73)

One of many possible combinations of Kraus operators that satisfies all these conditions is then:

\[ a_{00} = 1, \quad i_{00} = 0 \forall i \neq a, \]  
\[ a_{11} = 1, \quad i_{11} = 0 \forall i \neq a, \]  
\[ b_{12} = 1, \quad i_{12} = 0 \forall i \neq b, \]  
\[ b_{03} = 1, \quad i_{03} = 0 \forall i \neq b. \]  

(3.74)  
(3.75)  
(3.76)  
(3.77)

These equations, together with equations 3.68 through 3.71, fully determine a set of Kraus operators that successfully implement gate 7.

As shown in appendix B, proceeding identically for the remaining fifteen gates, it is trivial to show that a combination of Kraus operators that satisfies each of them can be found. Therefore, we conclude that the QNN can implement any classical 2-input, 1-output logic gate, so it is universal in this sense. This shows that the network is not restricted to learning only linearly separable patterns, i.e. it is not subject to a quantum version of Minsky-Papert’s theorem, so it is more powerful than its classical equivalent.

Now that we have seen that there exist configurations of the QNN that successfully implement the aforementioned gates, we need to find a learning procedure that in fact converges to said configurations.

### 3.2.4 Training the QNN

Having generalized the QNN by lifting the stochasticity constraints, the reparameterization algorithm described in section 3.1.1 no longer applies. One possible solution, as the authors note in (61), is to recast the constraint on the Kraus operators, given in equation 1.9, as a Stiefel manifold.
and then optimize over that manifold, as we will now see.

Let $\mathcal{M}(n, k, \mathbb{C})$ be the set of $n \times k$ matrices with complex elements, i.e., elements in $\mathbb{C}$. The Stiefel manifold is defined as:

$$V_k(\mathbb{C}^n) = \{ S \in \mathcal{M}(n, k, \mathbb{C}) : S^\dagger S = 1_k \}. \quad (3.78)$$

Given a set of $nk$ $n \times k$ Kraus operators, $\{A_i\}$, we define the corresponding Stiefel matrix $S$ as follows:

$$S = \begin{bmatrix}
A_1 \\
A_2 \\
\vdots \\
A_{nk}
\end{bmatrix}. \quad (3.79)$$

The constraint on the Kraus operators given in equation 1.9 can be expressed in terms of the Stiefel matrix as:

$$S^\dagger S = 1_k. \quad (3.80)$$

This constraint defines the complex Stiefel manifold $S = V_k(\mathbb{C}^{n^2k})$. We thus have an isometry between Kraus operators and a Stiefel matrix (61).

The training of the network will then proceed as follows: first, we generate a random set of Kraus operators. Then, we use the equivalence given above to obtain the corresponding Stiefel matrix, which we train using some optimization procedure with an appropriate cost function. This optimization procedure must keep the matrix in the manifold. Finally, we use the equivalence again to obtain the final Kraus operators corresponding to the trained network.

Before going into an example, a quick explanation of the optimization procedure we used is called for. We tried several of the algorithms available in manopt, a Matlab toolbox for optimization on manifolds (62), and the one that consistently performed better in our problem was a first-order steepest descent algorithm that uses back-tracking line-search. The steepest descent algorithm is arguably the simplest approach to optimization on manifolds and, conceptually, it is a straightforward generalization of the traditional gradient descent methods in Euclidean space to Riemannian manifolds. A formal description of how this generalization is done is given in (63).

Moving on to line-search methods, their goal is to find a step size that adequately reduces the cost function. However, we do not want to devote substantial resources to finding the value the step size that provides the largest reduction, since those are resources that could instead be used to find a better search direction. The goal is thus to find a value that reasonably minimizes the cost function, rather than the actual minimizing value. The backtracking line-search starts with a large estimate of the step size and iteratively shrinks it until we find a value that provides a decrease in the cost function consistent with what we expect to achieve based on the local function gradient.

Now that we have given an overview of how the optimization methods we used work, let’s give an example of an application of our QNN. It is implemented as a quantum channel de-
scribed by 8 $2 \times 4$ Kraus operators, so, the corresponding Stiefel matrix $S$ is a $16 \times 4$ matrix:

$$S = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_7 \end{bmatrix}, \quad (3.81)$$

where $A_0$ through $A_7$ are the Kraus operators that describe the network. Taking into account the correspondence between network output and measurement result of the output density matrix $\rho'$ given in table 3.5, a sensible choice for the cost function would be the sum over the $\rho'$ entries that correspond to the output we do not want for each of the inputs. As we have discussed in previous sections, the XOR gate is of particular importance, so let’s start by seeing how the network performs when trying to learn it. The cost function for this gate is:

$$f(s) = |\rho'_{11}(0,0)|^2 + |\rho'_{00}(0,1)|^2 + |\rho'_{00}(1,0)|^2 + |\rho'_{11}(1,1)|^2, \quad (3.82)$$

where $s$ is a point on the manifold and the cost function dependence on it comes from the relation between the output density matrix $\rho'$ and the input density matrix $\rho$. $\rho'_{ij}(k,l)$ denotes entry $ij$ of $\rho'$ for inputs $(k,l)$. As an example of the learning progress of our network using this method, we show in figure 3.3 the evolution of the cost function given in equation 3.82 for the XOR gate during 20 iterations and averaged over 1000 randomly chosen realizations of the Kraus operators, i.e., 1000 randomly chosen starting points on the manifold. As can be seen, the cost function approaches 0 after about 15 to 20 iterations, so we can conclude that the network learns the XOR gate successfully.

![Figure 3.3: Learning progress of our QNN on the XOR gate using a steepest descent algorithm in a complex Stiefel manifold, quantified by the cost function over 20 iterations and averaged over 1000 different randomly chosen starting points. The blue line is the median of the 1000 realizations, the top limit is the 95th percentile and the bottom limit is the 5th percentile. As we can see, the value of the cost function goes to zero very quickly, indicating that the network is capable of learning the XOR gate.](image)
By redefining the cost function appropriately and performing the training process again, similar results can be obtained for any of the fifteen other gates, showing that the QNN is capable of learning these gates.

3.2.5 Scaling

We have seen some properties of the simplest unit of our QNN and how to perform its training. However, in order for the network to be applicable to practical problems, we need to increase its dimension, both by adding hidden layers and increasing the size of the input and output layers. We are thus interested in studying how increasing the size of the network affects the training process. In more concrete terms, we want to know how the dimension of the Stiefel manifold in which we perform the optimization grows with the number of neurons and layers in the network.

To this end, the following result on quantum channels is useful:

**Proposition.** Let $H$ be a separable Hilbert space and let $L_1, L_2$ be two quantum channels on $L_1(H)$ with Kraus decompositions:

\[
L_1(T) = \sum_i A_i T A_i^\dagger \quad \text{and} \quad L_2(T) = \sum_j B_j T B_j^\dagger.
\]

The composition $L_2 \circ L_1$ is a quantum channel on $L_1(H)$. It admits a Kraus decomposition given by:

\[
L_2 \circ L_1(T) = \sum_{i,j} B_j A_i T A_i^\dagger B_j^\dagger.
\]

**Proof.** $L_1$ and $L_2$ are quantum channels on $L_1(H)$ so, by definition, for $i = 1, 2$ there exist Hilbert spaces $K_i$, states $\omega_i$ on $K_i$ and unitary operators $U_i$ on $K_i$ such that:

\[
L_i = \text{Tr}_{K_i} (U_i (\rho \otimes \omega_i) U_i^\dagger),
\]

for all $T \in L_1(H)$.

Let us now consider the Hilbert space $K = K_1 \otimes K_2$ and the quantum state $\omega = \omega_1 \otimes \omega_2$. We consider the natural extensions $\hat{U}_i$ of $U_i$ to $H \otimes K$ by taking the tensor product of $U_i$ with the identity operator on the space $K_j$ for $i \neq j$. We thus have that $U = \hat{U}_2 \hat{U}_1$ is a unitary operator on $H \otimes K$. Using the properties of partial traces, we have:

\[
\text{Tr}_K(U(T \otimes \omega) U^\dagger) = \text{Tr}_{K_1 \otimes K_2} \left( \hat{U}_2 \hat{U}_1 (T \otimes \omega_1 \otimes \omega_2) \hat{U}_1^\dagger \hat{U}_2^\dagger \right)
\]

\[
= \text{Tr}_{K_2} \left( \text{Tr}_{K_1} \left( \hat{U}_2 (U_1 (T \otimes \omega_1) U_1^\dagger \otimes \omega_2) \hat{U}_1^\dagger \right) \right)
\]

\[
= \text{Tr}_{K_2} \left( U_2 (\text{Tr}_{K_1} (U_1 (T \otimes \omega_1) U_1^\dagger \otimes \omega_2) U_1^\dagger) \right)
\]

\[
= \text{Tr}_{K_2} (U_2 (L_1(T) \otimes \omega_2) U_2^\dagger)
\]

\[
= L_2 (L_1(T)).
\]

We have expressed $L_2 \otimes L_1$ as the partial trace $T \rightarrow \text{Tr}_K(U(T \otimes \omega) U^\dagger)$. This is, by definition,
a quantum channel, so the composition of two quantum channels is also a quantum channel. We now want to find the Kraus representation of this quantum channel.

The unitary dilations of $\mathcal{L}_1$ and $\mathcal{L}_2$ can be chosen in such a way that $\omega_1$ and $\omega_2$ are pure states $|\psi_1\rangle\langle\psi_1|$. Let $(e_i)_{i\in\mathbb{N}}$ and $(f_j)_{j\in\mathbb{N}}$ be orthonormal bases of $\mathbb{K}_1$ and $\mathbb{K}_2$, respectively. It can be shown [64] that the coefficients of a Kraus representation of $\mathcal{L}_1$ can be obtained as:

$$A_i = \kappa_1(e_i|U_1|\psi_1)_{\mathbb{K}_1},$$

and similarly for $\mathcal{L}_2$:

$$B_j = \kappa_2(e_j|U_1|\psi_1)_{\mathbb{K}_2}.$$  

Now if we compute the coefficients of $\mathcal{L}_2 \circ \mathcal{L}_1$ we get:

$$M_{ij} = \kappa_1 \otimes \kappa_2 (e_i \otimes f_j |U| \psi_1 \otimes \psi_2)_{\mathbb{K}_1 \otimes \mathbb{K}_2}$$

$$= \kappa_2(f_j |e_i| U_2 \hat{U}_1 |\psi_1 \rangle \langle \psi_2|)_{\mathbb{K}_2}$$

$$= \kappa_2(f_j |U_2 \kappa_1 (e_i|U_1|\psi_1)_{\mathbb{K}_1}|\psi_2)_{\mathbb{K}_2}$$

$$= \kappa_2(f_j |U_2|\psi_2)_{\mathbb{K}_2} \kappa_1 (e_i|U_1|\psi_1)_{\mathbb{K}_1}$$

$$= B_j A_i.$$  

This gives the expected Kraus representation.

Just as our single-layer QNN can be seen as a quantum channel, a QNN with several layers can be seen as a composition of quantum channels, so it follows from the above proposition that a QNN can always be reduced to a single quantum channel of higher dimension. Therefore, adding layers and neurons to the QNN corresponds to increasing the maximum number of Kraus operators that describe the quantum channel.

If the input to the network is an $n$-dimensional density matrix and the output is an $m$-dimensional density matrix then the Kraus operators of the quantum channel that implements the network have dimension $m \times n$. If there are $k$ layers in the network, each with $j_k$ neurons, then the maximum number of Kraus operators is $N_{\text{Kraus}} = j_1 j_2 \ldots j_k$. Therefore, the Stiefel matrix of a $k$-layer QNN with $n$-dimensional input, $m$-dimensional output and $j_i$ neurons in the $i$th layer is given by:

$$S = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{j_1 j_2 \ldots j_k} \end{bmatrix}.$$

$\{A_i\}$ is a set of $j_1 j_2 \ldots j_k$ Kraus operators of dimension $n \times m$. The Stiefel matrix $S$ thus has dimension $j_1 j_2 \ldots j_k n \times m \approx j_{\text{av}}^k n \times m$, $j_{\text{av}}$ being the average number of neurons in a layer. This means that the dimension of the matrix, and thus the dimension of the search space, grows exponentially.
with the number of layers in the QNN. This presents a problem that must be overcome before this QNN can be useful in practice, since the exponential scaling implies that the search space will be far too big for problems with more than a few tens of neurons.

### 3.2.6 Dimensionality Reduction

As we just discussed, we would like to find a way of avoiding the exponential scaling of the network. One possibility would be to approximate the quantum channel that implements our QNN by a lower-dimensional quantum channel. In (65), the authors show that any quantum channel mapping states on some input Hilbert space $\mathcal{H}_A$ to states on some output Hilbert space $\mathcal{H}_B$ can be compressed into one with order $d \log(d)$ Kraus operators, where $d = \max(|\mathcal{H}_A|, |\mathcal{H}_B|)$, which is much less than the usual $|\mathcal{H}_A| \times |\mathcal{H}_B|$. More formally, the main result of (65) is the following proposition:

**Proposition.** Fix $0 < \epsilon < 1$ and let $N : \mathcal{L}(\mathcal{H}_A) \to \mathcal{L}(\mathcal{H}_B)$ be a CPTP map with Kraus rank $|\mathcal{H}_E| \geq |\mathcal{H}_A|, |\mathcal{H}_B|$. Then, there exists a CP map $\hat{N} : \mathcal{L}(\mathcal{H}_A) \to \mathcal{L}(\mathcal{H}_B)$ with Kraus rank at most

$$C \max(|\mathcal{H}_A|, |\mathcal{H}_B|) \frac{1}{\epsilon^2} \log \left( \frac{|\mathcal{H}_E|}{\epsilon} \right),$$

where $C > 0$ is a universal constant and such that:

$$\forall \rho \in \mathcal{D}(\mathcal{H}_A), \quad -\epsilon \left( N(\rho) - \frac{1}{|\mathcal{H}_B|} \right) \leq \hat{N}(\rho) - N(\rho) \leq \epsilon \left( N(\rho) + \frac{1}{|\mathcal{H}_B|} \right).$$

In particular,

$$\forall \rho \in \mathcal{D}(\mathcal{H}_A), \quad \|\hat{N}(\rho) - N(\rho)\|_1 \leq 2\epsilon,$$

where $\hat{N}(\rho)$ is strictly trace preserving.

A proof of this result can be found in the article. Now that we know that an approximation $\hat{N}$ to the quantum channel that implements the network, $N$, exists, it is natural to ask how it is constructed. We write $N$ in the physical form:

$$N(\rho) = \text{Tr}_E(U \rho U^\dagger).$$

(3.100)

For any given unit vector $\phi$ in the environment $E$ define the CP map $N_\phi : \mathcal{L}(A) \to \mathcal{L}(B)$ by:

$$N_\phi(\rho) = |E| \text{Tr}_E \left( (1 \otimes \phi) U \rho U^\dagger (1 \otimes \phi) \right).$$

(3.101)

For $\phi_1, \ldots, \phi_n$ independent uniformly distributed unit vectors in $E$, set $N_\phi(n) = \frac{1}{n} \sum_{i=1}^n N_{\phi_i}$. Finally, we have that for $n \leq C \max(|\mathcal{H}_A|, |\mathcal{H}_B|) \log \left( \frac{|\mathcal{H}_E|}{\epsilon^2} \right)$:

$$\|N_\phi(n) - N(\rho)\|_1 \leq 2\epsilon.$$  

(3.102)
The implications of this result to our QNN are clear: the number of layers and hidden layer neurons in the network does not significantly affect its capabilities, since we can always find a quantum channel with small Kraus rank that approximately implements the QNN. Moreover, this Kraus rank depends only on the dimension of the input and output Hilbert spaces. This dependence is linear, but, since the dimension of the Hilbert spaces scales exponentially with the number of bits, the same is true for the Kraus rank of the approximate QNN. This means that for networks with a large number of inputs, say of the order of $100$, the training process will still be intractable, because this network will be implemented by about $2^{100}$ Kraus operators. A number of inputs of this order is not unusual, so our network cannot be used for a sizable amount of problems unless the training process is in some way simplified.

3.3 Concluding Remarks

We have introduced a QNN based on quantum channels by taking advantage of a connection between classical and quantum channels. We have performed an analysis of its capabilities and concluded that it is universal in the classical logic gate sense, and that it is not subject to a quantum version of Minsky-Papert’s theorem. It is thus, in a sense, more powerful than its classical equivalent. By using an isometry between Kraus operators and Stiefel matrices, we have introduced a training procedure for our QNN based on optimization on manifolds. Through the use of a result on the approximation of quantum channels by lower-dimensional ones, we have seen that enriching the network by adding hidden layers is feasible.

As we mentioned in the previous section, the main question that remains to be addressed is the matter of the exponential scaling of the search space with the size of the input and output layers. Possible paths towards solving this problem will be discussed in the final chapter.
Chapter 4

Conclusions

The goal of this thesis was to begin making headway with a completely original approach to quantum neural networks (QNN) based on quantum memristors. Due to its complexity, we split this task into smaller steps. First, develop models for memristor-based classical neural networks (NN). Then, develop a QNN based on quantum channels to gain knowledge about QNNs in the context of open quantum systems, and in particular, of quantum memristors. The next step would be to use the knowledge we gained with this QNN model to help quantize the classical memristor-based perceptrons we proposed. We will use this final chapter to summarize the main results of this thesis, clarify its place in the field of quantum machine learning and go over the work that lies ahead.

We introduced models for classical single and multilayer perceptrons based on classical memristors. We provided learning algorithms for both, based on the delta rule and on the back-propagation algorithm, respectively. Using a threshold-based system, our models are able to use the internal variables of memristors to store and update the perceptron’s weights. We also ran simulations of both models, which revealed that they behaved as expected and in accordance with Minsky-Papert’s theorem. Our memristor-based perceptrons have the same capabilities and are subject to the same limitations of regular perceptrons and thus show the feasibility and power of a memristor-based NN.

Moving on to the quantum realm, we found a way to go from classical NNs to QNNs based on quantum channels by taking advantage of an equivalence between classical and quantum channels. An analysis of the capabilities of the resulting quantum network revealed that it can implement any 2-input, 1-output classical logic gate, so it is universal in this sense. In particular, it can implement the XOR gate, which means that there is no quantum version of Minsky-Papert’s theorem, so the QNN is not subject to the same limitations as its classical equivalent. By using an isometry between Kraus operators and Stiefel matrices, we defined a training procedure for the QNN based on optimization over Stiefel manifolds. Taking advantage of a result on the approximation of quantum channels by completely positive maps with low Kraus rank, we showed that the complexity of the learning process does not increase with the number of layers or the
number of hidden layer neurons.

Now that we have recapitulated the main results we have obtained, we will move on to what is left to be done. As we have already mentioned, since the experimental demonstration of memristors in 2008 there has been a boom in memristor-related research. There have been proposals of how memristors could be used in Hebbian learning systems, in the simulation of fluid-like integro-differential equations, in the construction of digital quantum computers and of how they could be used to implement non-volatile memories, among others. Despite all this, our models are the first ones in which memristors are used as both the nodes and the synapses in the context of supervised learning, thus paving the way for other NN architectures and algorithms based exclusively on memristors.

Although the idea of QNNs has been around since the 1990s, the growing interest in quantum machine learning that has been going for a few years has led to an increase in both the amount and overall quality of QNN proposals lately. Despite this, it is safe to say that it is not yet clear what the standard model of a QNN is, or even if any of the models proposed so far can eventually take that place. The goal of this thesis was to launch a completely original approach to this problem in the form of a quantum memristor based QNN. To this end, we have introduced a mapping between classical and quantum networks, a fully quantum framework for a NN, we have shown that our model is more powerful than its classical equivalent, we have proposed a training algorithm for the network and we have shown that scaling it by adding hidden layers is feasible.

However, there is still work to be done; the main issue that remains to be solved is related to the scaling of the input and output layers. For several problems, there will be a large amount of features and a large amount of outputs, so our network must have large input and output layers in order to be applicable. However, as we saw in section 3.2.5, the dimension of the search space in which the training is done grows exponentially with the size of the input and output layers. This means that having more than a handful of neurons in these layers makes the training process too slow to be practical. The reason why this is a problem for our QNN but not for classical networks is that classically the training is done locally, meaning that the parameters of each neuron are updated independently of the others and therefore the time needed to train the network scales linearly with its size. On the other hand, our QNN is trained globally, with all parameters being dependent on one another, which leads to the exponential scaling. We would thus like to find a method that allows us to optimize locally, node by node, instead of optimizing the whole network at once. This would result in a reduction of the effective degrees of freedom, solving the problem of the exponential scaling, similar to what happens with the density matrix renormalization group in quantum many-body physics (66) and, more generally, with renormalization group methods (67). In conclusion, the issue of the exponential scaling of the search space must be addressed before the QNN can be applied to real-world problems.

After the issues with the scaling of the QNN are addressed, the question that inspired this work remains: how to use quantum memristors, and their implementation in different quantum
technologies, such as superconducting circuits (68) or quantum photonics (69), to implement a QNN. This is not an easy task due to the complexity of the behaviour of quantum memristors, as was explained in section 1.2.4, but we hope that the results we have obtained about QNNs in the context of open quantum systems can serve as a guide. The most obvious path towards the connection between quantum channels and quantum memristors would be to express the quantum memristor as a quantum channel and use the resulting channel to implement a network in the way we described in this work. However, the matter of the classical feedback controlling the strength of the coupling between the system and the environment, which is integral to endow the quantum memristor with memory, is one that, as far as we know, has not yet been studied in the context of quantum channels, so that is a challenge that must be overcome before this connection can be made.

Moving on to future work in a broader sense, outside of the scope of our model and of quantum memristors, there is also a lot to be done. In this work, we have looked at QNNs from a supervised learning perspective, but there are other machine learning paradigms, such as reinforcement learning and unsupervised learning that might be transported to the quantum realm. There has been work done in these fields in the past couple of years (70–73), but there are still plenty of promising avenues of research that remain to be explored.
Bibliography


Appendix A

Encoding Neural Network in Stochastic Matrix

We claimed in section 3.2 that in order to encode the weights of a 2-input 1-output neural network in a stochastic matrix, a $2 \times 4$ matrix must be used. We will now show that it is not possible to use a lower dimensional matrix. A left-stochastic matrix $A$ is a matrix whose entries $\{A_{ij}\}$ fulfill the following constraints:

\[
\sum_k A_{kj} = 1, \forall j, \tag{A.1}
\]

\[
A_{ij} \geq 0, \forall i, j. \tag{A.2}
\]

We want to encode the weights of a simple perceptron with two inputs in a stochastic matrix. The naive way to do it would be something like:

\[
y = \begin{bmatrix} w_0 & w_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, \tag{A.3}
\]

where $y$ is the output of the perceptron and $w_i$ is the weight of the synapse connecting input $i$ to the neuron. However, the stochasticity constraint given by equation A.2 implies that $w_0 = w_1 = 1$, which means that learning cannot happen, so we clearly need a different way to store the weights of the connections in a stochastic matrix. A natural solution would be to add a second output neuron, and two corresponding extra synapses, which results in:

\[
\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, \tag{A.4}
\]

where $y_0$ and $y_1$ are the outputs and $w_{ij}$ are the weights of the synapses connecting output $y_i$ to input $x_j$. In this encoding, we’d set the second line of the weight matrix to:
\[ w_{1i} = 1 - w_{0i}, \]  

which means that the weights in the first line can take any value between 0 and 1, which is as much freedom we’re going to get using a stochastic matrix. However, when making the transition to the quantum case, we have that the input to the QNN is a density matrix. A 2D density matrix has, of course, two eigenvalues, i.e., two possible measurement outcomes. However, we are including four different bit combinations, that can be mapped to four different states, so we need a higher-dimensional solution.

Having excluded these alternatives, the natural step to take next is to use 4 bits for the inputs and 2 for the outputs, as was done in the main text.
Appendix B

QNN capabilities

All 2-input, 1-output classical logic gates are shown in table 3.9. We’ve seen in section 3.2 that our QNN is capable of learning the XOR gate, gate 7 in this table. We will now see if it can also learn the remaining gates.

Gate 1

Using the correspondence between measurement result and output mentioned in table 3.5 and the values of the diagonal entries of the output density matrix given in table 3.6, we get the following conditions for successful implementation of gate 1:

\[ \sum_{i=\alpha}^{h} |i_{00}|^2 = 1, \quad (B.1) \]
\[ \sum_{i=\alpha}^{h} |i_{10}|^2 = 0, \quad (B.2) \]
\[ \sum_{i=\alpha}^{h} |i_{01}|^2 = 1, \quad (B.3) \]
\[ \sum_{i=\alpha}^{h} |i_{11}|^2 = 0, \quad (B.4) \]

\[ \sum_{i=\alpha}^{h} |i_{02}|^2 = 1, \quad (B.5) \]
\[ \sum_{i=\alpha}^{h} |i_{12}|^2 = 0, \quad (B.6) \]
\[ \sum_{i=\alpha}^{h} |i_{03}|^2 = 1, \quad (B.7) \]
\[ \sum_{i=\alpha}^{h} |i_{13}|^2 = 0. \quad (B.8) \]

These imply:

\[ i_{10} = 0, \quad \forall i \in [a, h], \quad (B.9) \]
\[ i_{11} = 0, \quad \forall i \in [a, h], \quad (B.10) \]
\[ i_{12} = 0, \quad \forall i \in [a, h], \quad (B.11) \]
\[ i_{13} = 0, \quad \forall i \in [a, h]. \quad (B.12) \]
Equations 3.49 through 3.54 become:

\[ \sum_{i=a}^{h} i_{01}i_{00}^* = 0, \quad (B.13) \]
\[ \sum_{i=a}^{h} i_{02}i_{00}^* = 0, \quad (B.14) \]
\[ \sum_{i=a}^{h} i_{03}i_{00}^* = 0, \quad (B.15) \]
\[ \sum_{i=a}^{h} i_{02}i_{01}^* = 0, \quad (B.16) \]
\[ \sum_{i=a}^{h} i_{03}i_{01}^* = 0, \quad (B.17) \]
\[ \sum_{i=a}^{h} i_{03}i_{02}^* = 0. \quad (B.18) \]

One of many possible combinations of Kraus operators that satisfies all these conditions is then:

\[ a_{00} = 1, \quad i_{00} = 0 \forall i \neq a, \quad (B.19) \]
\[ b_{01} = 1, \quad i_{01} = 0 \forall i \neq b, \quad (B.20) \]
\[ c_{02} = 1, \quad i_{02} = 0 \forall i \neq c, \quad (B.21) \]
\[ d_{03} = 1, \quad i_{03} = 0 \forall i \neq d. \quad (B.22) \]

These equations, together with equations B.9 through B.12, fully determine a set of Kraus operators that successfully implement gate 1.

**Gate 2**

Conditions for successful implementation of gate 2:

\[ \sum_{i=a}^{h} |i_{00}|^2 = 1, \quad (B.23) \]
\[ \sum_{i=a}^{h} |i_{10}|^2 = 0, \quad (B.24) \]
\[ \sum_{i=a}^{h} |i_{01}|^2 = 1, \quad (B.25) \]
\[ \sum_{i=a}^{h} |i_{11}|^2 = 0, \quad (B.26) \]
\[ \sum_{i=a}^{h} |i_{02}|^2 = 1, \quad (B.27) \]
\[ \sum_{i=a}^{h} |i_{12}|^2 = 0, \quad (B.28) \]
\[ \sum_{i=a}^{h} |i_{03}|^2 = 0, \quad (B.29) \]
\[ \sum_{i=a}^{h} |i_{13}|^2 = 1. \quad (B.30) \]

These imply:

\[ i_{10} = 0, \forall i \in [a, h], \quad (B.31) \]
\[ i_{11} = 0, \forall i \in [a, h], \quad (B.32) \]
\[ i_{12} = 0, \forall i \in [a, h], \quad (B.33) \]
\[ i_{03} = 0, \forall i \in [a, h]. \quad (B.34) \]
So equations 3.51, 3.53 and 3.54 are satisfied. Equations 3.49, 3.50 and 3.52 become:

\[
\sum_{i=a}^{h} i_{01}i_{00}^* = 0, \quad \text{(B.35)}
\]
\[
\sum_{i=a}^{h} i_{02}i_{00}^* = 0, \quad \text{(B.36)}
\]
\[
\sum_{i=a}^{h} i_{02}i_{01}^* = 0. \quad \text{(B.37)}
\]

One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[
a_{00} = 1, \quad i_{00} = 0 \quad \forall i \neq a, \quad \text{(B.38)}
\]
\[
b_{01} = 1, \quad i_{01} = 0 \quad \forall i \neq b, \quad \text{(B.39)}
\]
\[
c_{02} = 1, \quad i_{02} = 0 \quad \forall i \neq c, \quad \text{(B.40)}
\]
\[
a_{13} = 1, \quad i_{13} = 0 \quad \forall i \neq a. \quad \text{(B.41)}
\]

These equations, together with equations B.31 through B.34, fully determine a set of Kraus operators that successfully implement gate 2.

**Gate 3**

Conditions for successful implementation of gate 3:

\[
\sum_{i=a}^{h} |i_{00}|^2 = 1, \quad \text{(B.42)}
\]
\[
\sum_{i=a}^{h} |i_{10}|^2 = 0, \quad \text{(B.43)}
\]
\[
\sum_{i=a}^{h} |i_{01}|^2 = 1, \quad \text{(B.44)}
\]
\[
\sum_{i=a}^{h} |i_{11}|^2 = 0, \quad \text{(B.45)}
\]
\[
\sum_{i=a}^{h} |i_{02}|^2 = 0, \quad \text{(B.46)}
\]
\[
\sum_{i=a}^{h} |i_{12}|^2 = 1, \quad \text{(B.47)}
\]
\[
\sum_{i=a}^{h} |i_{03}|^2 = 1, \quad \text{(B.48)}
\]
\[
\sum_{i=a}^{h} |i_{13}|^2 = 0. \quad \text{(B.49)}
\]

These imply:

\[
i_{10} = 0, \quad \forall i \in [a, h], \quad \text{(B.50)}
\]
\[
i_{11} = 0, \quad \forall i \in [a, h], \quad \text{(B.51)}
\]
\[
i_{02} = 0, \quad \forall i \in [a, h], \quad \text{(B.52)}
\]
\[
i_{13} = 0, \quad \forall i \in [a, h]. \quad \text{(B.53)}
\]
So equations 3.50, 3.52 and 3.54 are satisfied. Equations 3.49, 3.51 and 3.53 become:

\[ \sum_{i=a}^{h} t_{0i}i_{00}^* = 0, \]  
(B.54)

\[ \sum_{i=a}^{h} t_{03i}i_{00}^* = 0, \]  
(B.55)

\[ \sum_{i=a}^{h} t_{03i}i_{01}^* = 0. \]  
(B.56)

One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[ a_{00} = 1, \quad i_{00} = 0 \quad \forall i \neq a, \]  
(B.57)

\[ b_{01} = 1, \quad i_{01} = 0 \quad \forall i \neq b, \]  
(B.58)

\[ c_{03} = 1, \quad i_{03} = 0 \quad \forall i \neq c, \]  
(B.59)

\[ a_{12} = 1, \quad i_{12} = 0 \quad \forall i \neq a. \]  
(B.60)

These equations, together with equations B.50 through B.53, fully determine a set of Kraus operators that successfully implement gate 3.

**Gate 4**

Conditions for successful implementation of gate 4:

\[ \sum_{i=a}^{h} |i_{00}|^2 = 1, \]  
(B.61)

\[ \sum_{i=a}^{h} |i_{10}|^2 = 0, \]  
(B.62)

\[ \sum_{i=a}^{h} |i_{01}|^2 = 1, \]  
(B.63)

\[ \sum_{i=a}^{h} |i_{11}|^2 = 0, \]  
(B.64)

\[ \sum_{i=a}^{h} |i_{02}|^2 = 0, \]  
(B.65)

\[ \sum_{i=a}^{h} |i_{12}|^2 = 1, \]  
(B.66)

\[ \sum_{i=a}^{h} |i_{03}|^2 = 0, \]  
(B.67)

\[ \sum_{i=a}^{h} |i_{13}|^2 = 1. \]  
(B.68)

These imply:

\[ i_{10} = 0, \quad \forall i \in [a, h], \]  
(B.69)

\[ i_{11} = 0, \quad \forall i \in [a, h], \]  
(B.70)

\[ i_{02} = 0, \quad \forall i \in [a, h], \]  
(B.71)

\[ i_{03} = 0, \quad \forall i \in [a, h]. \]  
(B.72)

So equations 3.50, 3.51, 3.52 and 3.53 are satisfied. Equations 3.49 and 3.54 become:

\[ \sum_{i=a}^{h} t_{0i}i_{00}^* = 0, \]  
(B.73)

\[ \sum_{i=a}^{h} t_{13i}i_{12}^* = 0. \]  
(B.74)
One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[
\begin{align*}
    a_{00} &= 1, \ i_{00} = 0 \ \forall i \neq a, \quad \text{(B.75)} \\
    b_{01} &= 1, \ i_{01} = 0 \ \forall i \neq b, \quad \text{(B.76)} \\
    b_{12} &= 1, \ i_{12} = 0 \ \forall i \neq b, \quad \text{(B.77)} \\
    a_{13} &= 1, \ i_{13} = 0 \ \forall i \neq a. \quad \text{(B.78)} 
\end{align*}
\]

These equations, together with equations B.69 through B.72, fully determine a set of Kraus operators that successfully implement gate 4.

**Gate 5**

Conditions for successful implementation of gate 5:

\[
\begin{align*}
    \sum_{i=a}^{h} |i_{00}|^2 &= 1, \quad \text{(B.79)} \\
    \sum_{i=a}^{h} |i_{01}|^2 &= 0, \quad \text{(B.80)} \\
    \sum_{i=a}^{h} |i_{02}|^2 &= 0, \quad \text{(B.81)} \\
    \sum_{i=a}^{h} |i_{11}|^2 &= 1, \quad \text{(B.82)} \\
    \sum_{i=a}^{h} |i_{02}|^2 &= 1, \quad \text{(B.83)} \\
    \sum_{i=a}^{h} |i_{12}|^2 &= 0, \quad \text{(B.84)} \\
    \sum_{i=a}^{h} |i_{03}|^2 &= 1, \quad \text{(B.85)} \\
    \sum_{i=a}^{h} |i_{13}|^2 &= 0. \quad \text{(B.86)} 
\end{align*}
\]

These imply:

\[
\begin{align*}
    i_{10} &= 0, \ \forall i \in [a, h], \quad \text{(B.87)} \\
    i_{01} &= 0, \ \forall i \in [a, h], \quad \text{(B.88)} \\
    i_{12} &= 0, \ \forall i \in [a, h], \quad \text{(B.89)} \\
    i_{13} &= 0, \ \forall i \in [a, h]. \quad \text{(B.90)} 
\end{align*}
\]

So equations 3.49, 3.52 and 3.53 are satisfied. Equations 3.50, 3.51 and 3.54 become:

\[
\begin{align*}
    \sum_{i=a}^{h} i_{02}i_{00}^* &= 0, \quad \text{(B.91)} \\
    \sum_{i=a}^{h} i_{03}i_{00}^* &= 0, \quad \text{(B.92)} \\
    \sum_{i=a}^{h} i_{03}i_{02}^* &= 0. \quad \text{(B.93)} 
\end{align*}
\]

One of many possible combination of Kraus operators that satisfies all these conditions is then:
\[ a_{00} = 1, \quad i_{00} = 0 \quad \forall i \neq a, \quad (B.94) \]
\[ b_{02} = 1, \quad i_{02} = 0 \quad \forall i \neq b, \quad (B.95) \]
\[ c_{03} = 1, \quad i_{03} = 0 \quad \forall i \neq c, \quad (B.96) \]
\[ a_{11} = 1, \quad i_{11} = 0 \quad \forall i \neq a. \quad (B.97) \]

These equations, together with equations B.87 through B.90, fully determine a set of Kraus operators that successfully implement gate 5.

**Gate 6**

Conditions for successful implementation of gate 6:

\[ \sum_{i=a}^h |i_{00}|^2 = 1, \quad (B.98) \]
\[ \sum_{i=a}^h |i_{10}|^2 = 0, \quad (B.99) \]
\[ \sum_{i=a}^h |i_{01}|^2 = 0, \quad (B.100) \]
\[ \sum_{i=a}^h |i_{11}|^2 = 1. \quad (B.101) \]
\[ \sum_{i=a}^h |i_{02}|^2 = 1, \quad (B.102) \]
\[ \sum_{i=a}^h |i_{12}|^2 = 0, \quad (B.103) \]
\[ \sum_{i=a}^h |i_{03}|^2 = 0, \quad (B.104) \]
\[ \sum_{i=a}^h |i_{13}|^2 = 1. \quad (B.105) \]

These imply:

\[ i_{10} = 0, \quad \forall i \in [a, h], \quad (B.106) \]
\[ i_{01} = 0, \quad \forall i \in [a, h], \quad (B.107) \]
\[ i_{12} = 0, \quad \forall i \in [a, h], \quad (B.108) \]
\[ i_{03} = 0, \quad \forall i \in [a, h]. \quad (B.109) \]

So equations 3.49, 3.51, 3.52 and 3.54 are satisfied. Equation 3.50 and 3.53 become:

\[ \sum_{i=a}^h i_{02}i_{00}^* = 0, \quad (B.110) \]
\[ \sum_{i=a}^h i_{13}i_{11}^* = 0. \quad (B.111) \]

One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[ a_{00} = 1, \quad i_{00} = 0 \quad \forall i \neq a, \quad (B.112) \]
\[ a_{11} = 1, \quad i_{11} = 0 \quad \forall i \neq a, \quad (B.113) \]
\[ b_{02} = 1, \quad i_{02} = 0 \quad \forall i \neq b. \quad (B.114) \]
\[ b_{13} = 1, \quad i_{11} = 0 \quad \forall i \neq b. \quad (B.115) \]

These equations, together with equations B.106 through B.109, fully determine a set of Kraus operators that successfully implement gate 6.
operators that successfully implement gate 6.

**Gate 8**

Conditions for successful implementation of gate 8:

\[
\sum_{i=a}^{h} |i_{00}|^2 = 1, \quad (B.116) \quad \sum_{i=a}^{h} |i_{10}|^2 = 0, \quad (B.117) \quad \sum_{i=a}^{h} |i_{01}|^2 = 0, \quad (B.118) \quad \sum_{i=a}^{h} |i_{11}|^2 = 1, \quad (B.119)
\]

\[
\sum_{i=a}^{h} |i_{02}|^2 = 0, \quad (B.120) \quad \sum_{i=a}^{h} |i_{12}|^2 = 1, \quad (B.121) \quad \sum_{i=a}^{h} |i_{03}|^2 = 0, \quad (B.122) \quad \sum_{i=a}^{h} |i_{13}|^2 = 1. \quad (B.123)
\]

These imply:

\[
i_{10} = 0, \quad \forall i \in [a, h], \quad (B.124)
\]

\[
i_{01} = 0, \quad \forall i \in [a, h], \quad (B.125)
\]

\[
i_{02} = 0, \quad \forall i \in [a, h], \quad (B.126)
\]

\[
i_{03} = 0, \quad \forall i \in [a, h]. \quad (B.127)
\]

So equations 3.49, 3.50 and 3.51 are satisfied. Equations 3.52, 3.53 and 3.54 become:

\[
\sum_{i=a}^{h} i_{12}^* i_{11} = 0, \quad (B.128)
\]

\[
\sum_{i=a}^{h} i_{13}^* i_{11} = 0, \quad (B.129)
\]

\[
\sum_{i=a}^{h} i_{13}^* i_{12} = 0. \quad (B.130)
\]

One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[
a_{00} = 1, \quad i_{00} = 0 \quad \forall i \neq a, \quad (B.131)
\]

\[
a_{11} = 1, \quad i_{11} = 0 \quad \forall i \neq a, \quad (B.132)
\]

\[
b_{12} = 1, \quad i_{12} = 0 \quad \forall i \neq b, \quad (B.133)
\]

\[
c_{13} = 1, \quad i_{13} = 0 \quad \forall i \neq c. \quad (B.134)
\]

These equations, together with equations B.124 through B.127, fully determine a set of Kraus operators that successfully implement gate 8.

**Gate 9**

Conditions for successful implementation of gate 9:

\[
\sum_{i=a}^{h} |i_{00}|^2 = 0, \quad (B.135) \quad \sum_{i=a}^{h} |i_{10}|^2 = 1, \quad (B.136) \quad \sum_{i=a}^{h} |i_{01}|^2 = 1, \quad (B.137) \quad \sum_{i=a}^{h} |i_{11}|^2 = 0. \quad (B.138)
\]

59
These imply:

\[ i_{00} = 0, \forall i \in [a, h], \quad (B.143) \]
\[ i_{11} = 0, \forall i \in [a, h], \quad (B.144) \]
\[ i_{12} = 0, \forall i \in [a, h], \quad (B.145) \]
\[ i_{13} = 0, \forall i \in [a, h]. \quad (B.146) \]

So equations 3.49, 3.50 and 3.51 are satisfied. Equations 3.52, 3.53 and 3.54 become:

\[ \sum_{i=a}^{h} |i_{02}|^2 = 0, \quad (B.147) \]
\[ \sum_{i=a}^{h} |i_{03}|^2 = 0, \quad (B.148) \]
\[ \sum_{i=a}^{h} |i_{03}|^2 = 0. \quad (B.149) \]

One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[ a_{10} = 1, i_{10} = 0 \forall i \neq a, \quad (B.150) \]
\[ a_{01} = 1, i_{01} = 0 \forall i \neq a, \quad (B.151) \]
\[ b_{02} = 1, i_{02} = 0 \forall i \neq b, \quad (B.152) \]
\[ c_{03} = 1, i_{13} = 0 \forall i \neq c. \quad (B.153) \]

These equations, together with equations B.143 through B.146, fully determine a set of Kraus operators that successfully implement gate 9.

**Gate 10**

Conditions for successful implementation of gate 10:

\[ \sum_{i=a}^{h} |i_{02}|^2 = 0, \quad (B.154) \]
\[ \sum_{i=a}^{h} |i_{10}|^2 = 1, \quad (B.155) \]
\[ \sum_{i=a}^{h} |i_{01}|^2 = 1, \quad (B.156) \]
\[ \sum_{i=a}^{h} |i_{11}|^2 = 0. \quad (B.157) \]
\[ \sum_{i=a}^{h} |i_{02}|^2 = 1, \quad (B.158) \]
\[ \sum_{i=a}^{h} |i_{12}|^2 = 0, \quad (B.159) \]
\[ \sum_{i=a}^{h} |i_{03}|^2 = 0, \quad (B.160) \]
\[ \sum_{i=a}^{h} |i_{13}|^2 = 1. \quad (B.161) \]

These imply:
\[ i_{00} = 0, \forall i \in [a, h], \quad \text{(B.162)} \]
\[ i_{11} = 0, \forall i \in [a, h], \quad \text{(B.163)} \]
\[ i_{12} = 0, \forall i \in [a, h], \quad \text{(B.164)} \]
\[ i_{03} = 0, \forall i \in [a, h]. \quad \text{(B.165)} \]

So equations 3.49, 3.50, 3.53 and 3.54 are satisfied. Equations 3.51 and 3.52 become:

\[
\sum_{i=a}^{h} i_{13}i_{10}^\ast = 0, \quad \text{(B.166)}
\]
\[
\sum_{i=a}^{h} i_{02}i_{01}^\ast = 0. \quad \text{(B.167)}
\]

One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[
a_{10} = 1, \quad i_{10} = 0 \forall i \neq a, \quad \text{(B.168)}
\]
\[
a_{01} = 1, \quad i_{01} = 0 \forall i \neq a, \quad \text{(B.169)}
\]
\[
b_{02} = 1, \quad i_{02} = 0 \forall i \neq b, \quad \text{(B.170)}
\]
\[
b_{13} = 1, \quad i_{13} = 0 \forall i \neq b. \quad \text{(B.171)}
\]

These equations, together with equations B.162 through B.165, fully determine a set of Kraus operators that successfully implement gate 10.

**Gate 11**

Conditions for successful implementation of gate 11:

\[
\sum_{i=a}^{h} |i_{00}|^2 = 0, \quad \text{(B.172)}
\]
\[
\sum_{i=a}^{h} |i_{10}|^2 = 1, \quad \text{(B.173)}
\]
\[
\sum_{i=a}^{h} |i_{01}|^2 = 1, \quad \text{(B.174)}
\]
\[
\sum_{i=a}^{h} |i_{11}|^2 = 0, \quad \text{(B.175)}
\]
\[
\sum_{i=a}^{h} |i_{02}|^2 = 0, \quad \text{(B.176)}
\]
\[
\sum_{i=a}^{h} |i_{12}|^2 = 1, \quad \text{(B.177)}
\]
\[
\sum_{i=a}^{h} |i_{03}|^2 = 1, \quad \text{(B.178)}
\]
\[
\sum_{i=a}^{h} |i_{13}|^2 = 0. \quad \text{(B.179)}
\]

These imply:

\[
i_{00} = 0, \forall i \in [a, h], \quad \text{(B.180)}
\]
\[
i_{11} = 0, \forall i \in [a, h], \quad \text{(B.181)}
\]
\[
i_{02} = 0, \forall i \in [a, h], \quad \text{(B.182)}
\]
\[
i_{13} = 0, \forall i \in [a, h]. \quad \text{(B.183)}
\]
So equations 3.49, 3.51, 3.52 and 3.54 are satisfied. Equations 3.50 and 3.53 become:

\[
\begin{align*}
\sum_{i=a}^{b} i_{12}i_{10}^* &= 0, & \text{(B.184)} \\
\sum_{i=a}^{b} i_{03}i_{01}^* &= 0. & \text{(B.185)}
\end{align*}
\]

One of many possible combinations of Kraus operators that satisfies all these conditions is then:

\[
\begin{align*}
&a_{10} = 1, \quad i_{10} = 0 \quad \forall i \neq a, & \text{(B.186)} \\
&a_{01} = 1, \quad i_{01} = 0 \quad \forall i \neq a, & \text{(B.187)} \\
&b_{12} = 1, \quad i_{12} = 0 \quad \forall i \neq b, & \text{(B.188)} \\
&b_{03} = 1, \quad i_{03} = 0 \quad \forall i \neq b. & \text{(B.189)}
\end{align*}
\]

These equations, together with equations B.180 through B.183, fully determine a set of Kraus operators that successfully implement gate 11.

**Gate 12**

Conditions for successful implementation of gate 12:

\[
\begin{align*}
\sum_{i=a}^{b} |i_{00}|^2 &= 0, & \text{(B.190)} \\
\sum_{i=a}^{b} |i_{10}|^2 &= 1, & \text{(B.191)} \\
\sum_{i=a}^{b} |i_{01}|^2 &= 1, & \text{(B.192)} \\
\sum_{i=a}^{b} |i_{11}|^2 &= 0, & \text{(B.193)} \\
\sum_{i=a}^{b} |i_{02}|^2 &= 0, & \text{(B.194)} \\
\sum_{i=a}^{b} |i_{12}|^2 &= 1, & \text{(B.195)} \\
\sum_{i=a}^{b} |i_{03}|^2 &= 0, & \text{(B.196)} \\
\sum_{i=a}^{b} |i_{13}|^2 &= 1. & \text{(B.197)}
\end{align*}
\]

These imply:

\[
\begin{align*}
&i_{00} = 0, \quad \forall i \in [a, h], & \text{(B.198)} \\
&i_{11} = 0, \quad \forall i \in [a, h], & \text{(B.199)} \\
&i_{02} = 0, \quad \forall i \in [a, h], & \text{(B.200)} \\
&i_{03} = 0, \quad \forall i \in [a, h]. & \text{(B.201)}
\end{align*}
\]

So equations 3.49, 3.52 and 3.53 are satisfied. Equations 3.50, 3.51 and 3.54 become:

\[
\begin{align*}
\sum_{i=a}^{b} i_{12}i_{10}^* &= 0, & \text{(B.202)} \\
\sum_{i=a}^{b} i_{13}i_{10}^* &= 0, & \text{(B.203)} \\
\sum_{i=a}^{b} i_{13}i_{12}^* &= 0. & \text{(B.204)}
\end{align*}
\]
One of many possible combination of Kraus operators that satisfies all these conditions is then:

$$a_{10} = 1, \quad i_{10} = 0 \quad \forall i \neq a, \quad \text{(B.205)}$$
$$a_{01} = 1, \quad i_{01} = 0 \quad \forall i \neq a, \quad \text{(B.206)}$$
$$b_{12} = 1, \quad i_{12} = 0 \quad \forall i \neq b, \quad \text{(B.207)}$$
$$c_{13} = 1, \quad i_{13} = 0 \quad \forall i \neq c. \quad \text{(B.208)}$$

These equations, together with equations B.198 through B.201, fully determine a set of Kraus operators that successfully implement gate 12.

**Gate 13**

Conditions for successful implementation of gate 13:

\[
\sum_{i=a}^{h} |i_{00}|^2 = 0, \quad \text{(B.209)}
\]
\[
\sum_{i=a}^{h} |i_{10}|^2 = 1, \quad \text{(B.210)}
\]
\[
\sum_{i=a}^{h} |i_{01}|^2 = 0, \quad \text{(B.211)}
\]
\[
\sum_{i=a}^{h} |i_{11}|^2 = 1, \quad \text{(B.212)}
\]
\[
\sum_{i=a}^{h} |i_{02}|^2 = 1, \quad \text{(B.213)}
\]
\[
\sum_{i=a}^{h} |i_{12}|^2 = 0, \quad \text{(B.214)}
\]
\[
\sum_{i=a}^{h} |i_{03}|^2 = 1, \quad \text{(B.215)}
\]
\[
\sum_{i=a}^{h} |i_{13}|^2 = 0. \quad \text{(B.216)}
\]

These imply:

\[
i_{00} = 0, \quad \forall i \in [a, h], \quad \text{(B.217)}
\]
\[
i_{01} = 0, \quad \forall i \in [a, h], \quad \text{(B.218)}
\]
\[
i_{12} = 0, \quad \forall i \in [a, h], \quad \text{(B.219)}
\]
\[
i_{13} = 0, \quad \forall i \in [a, h]. \quad \text{(B.220)}
\]

So equations 3.50, 3.51, 3.52 and 3.53 are satisfied. Equations 3.49 and 3.54 become:

\[
\sum_{i=a}^{h} i_{11}i_{10}^* = 0, 
\quad \text{(B.221)}
\]
\[
\sum_{i=a}^{h} i_{03}i_{02}^* = 0. 
\quad \text{(B.222)}
\]

One of many possible combinations of Kraus operators that satisfies all these conditions is then:

$$a_{10} = 1, \quad i_{10} = 0 \quad \forall i \neq a, \quad \text{(B.223)}$$
$$b_{11} = 1, \quad i_{11} = 0 \quad \forall i \neq b, \quad \text{(B.224)}$$
$$a_{02} = 1, \quad i_{02} = 0 \quad \forall i \neq a, \quad \text{(B.225)}$$
$$b_{03} = 1, \quad i_{03} = 0 \quad \forall i \neq b. \quad \text{(B.226)}$$
These equations, together with equations B.217 through B.220, fully determine a set of Kraus operators that successfully implement gate 13.

**Gate 14**

Conditions for successful implementation of gate 14:

\[
\sum_{i=a}^{h} |i_{00}|^2 = 0, \quad \sum_{i=a}^{h} |i_{10}|^2 = 1, \quad \sum_{i=a}^{h} |i_{01}|^2 = 0, \quad \sum_{i=a}^{h} |i_{11}|^2 = 1, \quad \sum_{i=a}^{h} |i_{02}|^2 = 0, \quad \sum_{i=a}^{h} |i_{12}|^2 = 0, \quad \sum_{i=a}^{h} |i_{03}|^2 = 0, \quad \sum_{i=a}^{h} |i_{13}|^2 = 1.
\]

These imply:

\[
\begin{align*}
i_{00} &= 0, \quad \forall i \in [a, h], \\
i_{01} &= 0, \quad \forall i \in [a, h], \\
i_{12} &= 0, \quad \forall i \in [a, h], \\
i_{03} &= 0, \quad \forall i \in [a, h].
\end{align*}
\]

So equations 3.50, 3.52 and 3.54 are satisfied. Equations 3.49, 3.51 and 3.53 become:

\[
\begin{align*}
\sum_{i=a}^{h} i_{11}i_{10}^* &= 0, \\
\sum_{i=a}^{h} i_{13}i_{10}^* &= 0, \\
\sum_{i=a}^{h} i_{13}i_{11}^* &= 0.
\end{align*}
\]

One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[
\begin{align*}
a_{10} &= 1, \quad i_{10} = 0 \quad \forall i \neq a, \\
b_{11} &= 1, \quad i_{11} = 0 \quad \forall i \neq b, \\
a_{02} &= 1, \quad i_{02} = 0 \quad \forall i \neq a, \\
c_{13} &= 1, \quad i_{13} = 0 \quad \forall i \neq c.
\end{align*}
\]

These equations, together with equations B.235 through B.238, fully determine a set of Kraus operators that successfully implement gate 14.

**Gate 15**

Conditions for successful implementation of gate 15:
\[ \sum_{i=a}^{h} |i_{00}|^2 = 0, \quad (B.246) \quad \sum_{i=a}^{h} |i_{10}|^2 = 1, \quad (B.247) \quad \sum_{i=a}^{h} |i_{01}|^2 = 0, \quad (B.248) \quad \sum_{i=a}^{h} |i_{11}|^2 = 1, \quad (B.249) \]
\[ \sum_{i=a}^{h} |i_{02}|^2 = 0, \quad (B.250) \quad \sum_{i=a}^{h} |i_{12}|^2 = 1, \quad (B.251) \quad \sum_{i=a}^{h} |i_{03}|^2 = 1, \quad (B.252) \quad \sum_{i=a}^{h} |i_{13}|^2 = 0. \quad (B.253) \]

These imply:

\[ i_{00} = 0, \forall i \in [a, h], \quad (B.254) \]
\[ i_{01} = 0, \forall i \in [a, h], \quad (B.255) \]
\[ i_{02} = 0, \forall i \in [a, h], \quad (B.256) \]
\[ i_{13} = 0, \forall i \in [a, h]. \quad (B.257) \]

So equations 3.51, 3.53 and 3.54 are satisfied. Equations 3.49, 3.50 and 3.52 become:

\[ \sum_{i=a}^{h} i_{11}^* i_{10} = 0, \quad (B.258) \]
\[ \sum_{i=a}^{h} i_{12}^* i_{10} = 0, \quad (B.259) \]
\[ \sum_{i=a}^{h} i_{12}^* i_{11} = 0. \quad (B.260) \]

One of many possible combinations of Kraus operators that satisfies all these conditions is then:

\[ a_{10} = 1, \quad i_{10} = 0 \forall i \neq a, \quad (B.261) \]
\[ b_{11} = 1, \quad i_{11} = 0 \forall i \neq b, \quad (B.262) \]
\[ c_{12} = 1, \quad i_{12} = 0 \forall i \neq c, \quad (B.263) \]
\[ a_{03} = 1, \quad i_{03} = 0 \forall i \neq a. \quad (B.264) \]

These equations, together with equations B.254 through B.257, fully determine a set of Kraus operators that successfully implement gate 15.

**Gate 16**

Conditions for successful implementation of gate 16:

\[ \sum_{i=a}^{h} |i_{00}|^2 = 0, \quad (B.265) \quad \sum_{i=a}^{h} |i_{10}|^2 = 1, \quad (B.266) \quad \sum_{i=a}^{h} |i_{01}|^2 = 0, \quad (B.267) \quad \sum_{i=a}^{h} |i_{11}|^2 = 1, \quad (B.268) \]
\[ \sum_{i=a}^{h} |i_{02}|^2 = 0, \quad (B.269) \quad \sum_{i=a}^{h} |i_{12}|^2 = 1, \quad (B.270) \quad \sum_{i=a}^{h} |i_{03}|^2 = 0, \quad (B.271) \quad \sum_{i=a}^{h} |i_{13}|^2 = 1. \quad (B.272) \]

These imply:
\[ i_{00} = 0, \forall i \in [a, h], \quad (B.273) \]
\[ i_{01} = 0, \forall i \in [a, h], \quad (B.274) \]
\[ i_{02} = 0, \forall i \in [a, h], \quad (B.275) \]
\[ i_{03} = 0, \forall i \in [a, h]. \quad (B.276) \]

Equations 3.49 through 3.54 become:

\[ \sum_{i=a}^{h} i_{11} i_{10}^* = 0, \quad (B.277) \]
\[ \sum_{i=a}^{h} i_{12} i_{10}^* = 0, \quad (B.278) \]
\[ \sum_{i=a}^{h} i_{13} i_{10}^* = 0, \quad (B.279) \]
\[ \sum_{i=a}^{h} i_{12} i_{11}^* = 0, \quad (B.280) \]
\[ \sum_{i=a}^{h} i_{13} i_{11}^* = 0, \quad (B.281) \]
\[ \sum_{i=a}^{h} i_{13} i_{12}^* = 0. \quad (B.282) \]

One of many possible combination of Kraus operators that satisfies all these conditions is then:

\[ a_{10} = 1, \quad i_{10} = 0 \forall i \neq a, \quad (B.283) \]
\[ b_{11} = 1, \quad i_{11} = 0 \forall i \neq b, \quad (B.284) \]
\[ c_{12} = 1, \quad i_{12} = 0 \forall i \neq c, \quad (B.285) \]
\[ d_{13} = 1, \quad i_{13} = 0 \forall i \neq d. \quad (B.286) \]

These equations, together with equations B.273 through B.276, fully determine a set of Kraus operators that successfully implement gate 16.