Performance Evaluation of Free Fermionic Quantum Recurrent Neural Networks

Candidate number: **1079430**

University of Oxford

A dissertation submitted for the degree of

Master of Science
Mathematics and Foundations of Computer Science

Trinity 2024

Acknowledgements

I would like to express my gratitude to my three supervisors, Tuomas Laakkonen, Gabriel Matos, Aleks Kissinger, and Professor Bob Coecke, without whom none of this work would have been possible. Their guidance, continuous support, insightful discussions, and patience have filled me with ideas and directions as I discovered the exciting field of research at the intersection of Quantum and AI.

Thank you also to my friends, particularly at the Computer Science Department, the Mathematical Institute, and Lady Margaret Hall, for all the time we spent together and with whom I had so many lively debates that energised me both within and outside my work.

I am also very grateful to my parents and sister back home for their support over all the years that led to this dissertation.

Finally, I would also like to thank you, Carolina, for your continuous support and for enduring my midnight oils with a smile and kindness.

Abstract

The field of Quantum Machine Learning (QML) explores the intersection between quantum circuits and algorithms and how they can be applied to the field of Machine Learning to provide meaningful computational speedups. Unfortunately, due to the current lack of error correction and the so-called phenomenon of *barren plateaus*, it is difficult to scale QML models.

In this research, we introduce classically simulable quantum circuits, namely those mappable to free fermions, to Quantum Machine Learning models. Quantum Recurrent Neural Networks (QRNN), chosen as the QML model, are applied to Quantum Natural Language Processing (QNLP) tasks. Where universal quantum QML models are constrained to around 30 qubits, in contrast, free fermionic circuits, composed of matchgates, could offer an approach to scaling QML models up to 100 qubits and potentially extending to 1,000 qubits.

This research models QRNNs following two approaches: fully quantum models and free fermionic models composed of matchgates. Where the first type of algorithm is constrained to be executed on quantum computers, the second type is classically simulable. Our investigation simulates the behaviour of each type of models, establishing a framework to assess their respective performance in terms of accuracy, the primary criterion to assess QNLP models. We present evidence suggesting that free fermionic models can reach the same accuracy as fully quantum models. Furthermore, we study the entropy of the models to observe if the theoretical over-parametrisation threshold, occurring with other QML models, also arises with free fermions.

Last, the property of being classically simulable with a satisfactory level of accuracy and under polynomial execution time paves the way for QRNNs -and, by extension, QML models- to be both practical and scalable without the need for resource-intensive quantum hardware. This research seeks to determine whether these advantages can be achieved with classically simulable models, thus potentially reducing the need for quantum computers in the future if classical computers are all that is required.

Keywords— *classical simulation, free fermion, parametrised quantum circuit, quantum machine learning, quantum natural language processing, quantum simulation, NISQ computers.*

Contents

List of Abbreviations

DisCoCat . . . Categorical Compositional Distributional Framework

DisCoCirq . . Compositional Distributional Circuits Framework

GPU . . . . . Graphical Unit Processing

ML . . . . . Machine Learning

NISQ . . . . . Noise Intermediate-Scale Quantum

NLP . . . . . Natural Language Processing

PQC . . . . . Parameterised Quantum Circuit

QC . . . . . Quantum Computing

QML . . . . . Quantum Machine Learning

QNLP . . . . . Quantum Natural Language Processing

QRNN . . . . Quantum Recurrent Neural Network

RNN . . . . . Recurrent Neural Network

VQA . . . . . Variational Quantum Algorithm

1 Introduction

Contents

The development of high-performing computational resources and the proliferation of data have ushered in a period of fast advancements in Artificial Intelligence (AI). Core to AI, Machine Learning (ML) enables systems to learn from data, similar to how a human would, without explicit instructions. One focal point of technological innovation in AI is Natural Language Processing (NLP) technologies, which have experienced a rapid growth in the number of applications with the introduction of statistical methods. As a result, Recurrent Neural Networks (RNNs), the most fundamental networks for sequential supervised learning [1], have emerged as powerful tools for modelling sequential data while capturing the temporal dependencies intrinsically embedded in the various domains on which these tools are applied.

Once a separate field of research, quantum methodologies are now applied to Artificial Intelligence research. The focus is on solving ML tasks more efficiently with Quantum Machine Learning (QML) than with a classical computer. Some of the criteria for evaluating QML's performance are the accuracy of the models, the execution speed of the algorithms, and the cost of delivering the result.

As accuracy is the critical metric that fundamentally determines a model's performance and is straightforward and easily interpretable, we used it as the baseline of our work to

compare two different quantum approaches applied to four Machine Learning domains. Despite this, other aspects are important too, but we leave this for future work.

By performing the comparison on the quantum equivalent of Recurrent Neural Networks, Quantum Recurrent Neural Networks (QRNNs), this research seeks to advance the state-of-the-art in research involving current quantum computers known as Near-term Noisy Intermediate-Scale Quantum (NISQ) computers [2]. More specifically, this dissertation aims to expose the possibilities and limits of quantum algorithms built upon two different paradigms of quantum logic gates in the quantum circuit model of computation. One old, composed of universal-parametrised circuits, and one new, using matchgate-parametrised circuits. This research also expands on the possibilities of this new approach.

In practice, QRNNs are made of quantum circuits, which act on qubits, the smallest unit of quantum information. Qubits can be implemented in a multitude of ways. There exist superconducting [3], silicon [4], photon [5] and trapped ions types of qubits [6]. When discussing qubits, it is important to distinguish between two types: physical qubits and logical qubits. Quantum circuits are made up of noisy physical qubits. The current technologies are fragile and preclude the reliable encoding and retaining of information for long enough periods to be useful. This is where *Fault-Tolerant* [7] quantum computing comes in, where physical qubits are encoded into *logical qubits* using *surface codes* [8]. A logical qubit specifies how a single qubit should behave in a quantum algorithm according to quantum logic operations. These operations are built out of quantum logic gates changing the states of logical qubits. A group of quantum logic gates will then form a quantum circuit.

Currently, universal circuits used in QRNNs stay within the boundaries of Quantum Computing in which all information processing is carried out using only the principles of quantum mechanics. We will refer to these circuits as fully quantum circuits, representing the old paradigm. This research aims to determine whether a new implementation of QRNN models would improve accuracy compared to fully quantum QRNN models. We explored a new paradigm of QRNN models based on matchgate circuits. Matchgates are a class of two-qubit gates holding strict properties allowing the simulation of free fermionic systems in polynomial time. This means that matchgate circuits are classically simulable, which enables the use of classical resources for computations. This makes free fermionic QRNN models potentially cost-effective compared to fully quantum circuits, as running an algorithm on a quantum computer is very expensive.

Our research has the potential to radically change the way Quantum Computing (QC) is used. Finding the limit at which classical computing resources can or cannot be used for

a given level of accuracy of QRNN models would mean that there are whole classes of QML and QNLP tasks that could dispense with the implementation of fully quantum circuits.

If, in terms of accuracy, it is possible to achieve an advantage with free fermionic QRNNs, then it would be sensible to verify whether, in practice, there is truly an execution time speedup running fully quantum QRNNs on an actual quantum computer. If the speedup is insignificant, this would imply that there is no practical need for quantum computers for QRNNs with today's already powerful supercomputers. It would also signal that fully quantum models require further development to provide a quantum advantage in QML. If the accuracy of the two models is comparable, free fermionic models could be used to initialise fully quantum models running on NISQ computers, thus preserving costs and resources. This is known as *warm starting* [9]: a process where a quantum algorithm is initialised with an estimated or partially computed solution from a classical execution. The advantage of this solution is that computing resources are preserved. If the quantum states are initialised close to the optimal states found classically, the probability of quantum algorithms converging to an optimal solution is enhanced.

The research methodology to test the accuracy consists of applying the two approaches of QRNNs to sequential data epitomised by Natural Language Processing tasks, namely sentiment analysis, text generation, and document similarity.

- Sentiment analysis requires the correct prediction of a sentence's sentiment.
- Text generation requires contextual understanding and coherence by the machine.
- Document similarity involves grouping elements of texts based on their similarity, which, when grouped into topics, helps organise large text corpora.

Image classification is also performed and can be solved with methods similar to NLP tasks.

Testing these use cases according to the two approaches, free fermions versus fully quantum, we demonstrate the power and limitations of each, with granularity.

As the results in chapter 7 will show, free fermionic circuits can be as accurate as fully quantum circuits. We also observe that, in practice, free fermionic models differ in changes in convergence speed, following under or over-parametrisation requirements, compared to many other QML models that only change past a sharp threshold.

1.1 Main Contributions

The novel contributions of this research are:

1. The introduction of the free fermionic model in QML, which we implemented ex nihilo to build Quantum Recurrent Neural Networks solving four various Quantum Natural Language Processing tasks. As well as the design of the right structures of QRNNs to fit the particular needs of each task. (Chapter 5)

2. The introduction of a visual representation of the correspondence matchgate-to-graph to obtain the number of weighted perfect matching used to measure the matchgate circuits. (Section 3.2.4)

3. The implementation and build of free fermionic models to leverage its classical simulability, keeping it efficient on classical computers. (Section 3.3)

4. The design and setting of the experimentation framework with the free fermionic and fully quantum circuit structures, including selecting their hyperparameters. (Chapter 6)

5. The training of large amounts of models, and empirically testing how fully quantum and free fermionic models perform in the context of Quantum Machine Learning. We also observed if there is a quantum advantage or if classically simulable circuits perform the same or better (performance both in accuracy and time to train). (Section 7.1 and 7.2)

6. The empirical exploration of the relationship between convergence and the theoretical over-parametrisation threshold by defining a novel entropy measure of the cost function of a model. (Section 7.3)

1.2 Structure of this Dissertation

We begin this dissertation by defining in chapter 2 the necessary background information about QC and QML. This is necessary to understand the QRNN model. In chapter 3, free fermions are detailed, exploring how they are connected to matchgates and how the latter are classically simulable. Chapter 4 focuses on QRNN models, comprising an overview of classical RNNs for the understanding of their difference with their quantum versions.

The following chapters deal with the experimentation of the concepts introduced. Chapter 5 details the QML task used to test the different circuits. Detailed descriptions are given of how the data is used for the tasks, how it is processed, and how to adapt a QRNN to solve the tasks. Chapter 7 summarises the results from the experimentation. Finally, The conclusion in chapter 8 explains the significance of the results obtained by the series of experimentations, offering possible explanations and exploring future works.

2
Background Knowledge

Contents

In 1982, Richard Feynman pointed out that quantum mechanical systems are notoriously difficult to simulate on classical computers. He then speculated that this problem could be explored by building a computer based on quantum mechanics [10]. In 1985, physicist David Deutsch formalised a theory which described such a device, which he called *universal quantum computer*, otherwise known as the quantum Turing machine [11]. He then proposed, in 1992, alongside physicist Richard Jozsa, one of the first quantum algorithms designed to run on his quantum computer. Though it had limited practical use, it showed the first instance where a quantum algorithm provided an exponential speedup over classical algorithms [12]. However, it was not until 1996 that physicist Seth Lloyd proved that quantum mechanical systems could be efficiently simulated on quantum computers [13].

2.1 An overview of Quantum Computing

Quantum computing has the potential to unlock solutions to previously intractable problems and to revolutionise a wide range of fields, from cryptography to drug discovery, and, in particular, machine learning. The quantum speed-up comes from the exploitation of certain properties of quantum mechanics, which allows a quantum computer to process information differently than its classical counterpart. The two main quantum mechanics properties at play when performing quantum computing experiments are superposition and entanglement.

Superposition

Classical computers use bits as units of information. Bits can either be in the state 0 or 1. Quantum computers, on the other hand, use *qubits*, or quantum bits, as their unit of information. The particularity of qubits is that they can exist in superposition. When a qubit exists in superposition, it has a certain probability p of being in state 0, and thus a probability $1 - p$ of being in state 1. It is not until the state of the qubit is measured that the qubit collapses to a classical bit with a certain state. Quantum gates are used to manipulate qubits into their desired states.

It can be unclear what it means to measure a qubit in superposition. To explain, let us take an example of an experiment with a single qubit. Let us suppose that this qubit is in "perfect" superposition, meaning that it has probability $p = 50\%$ of being in state 0 and state 1 (this can be achieved with a Hadamard gate, which will be introduced later). This then means that around 50% of the times this qubit is measured, the state 0 is achieved and similarly for the state 1.

Entanglement

Entanglement is a phenomenon that occurs when two or more qubits are manipulated in such a way that none of the entangled qubits can be described independently of the others. This concept is particularly difficult to understand when such entangled qubits are connected over large distances, as it would indicate that information can travel faster than light. The reason for this is out of the scope of this research, though it is interesting to note that Einstein described this phenomenon as a "spooky action at a distance" [14].

Limitations of classical computing

However, the need for Quantum Computing not only comes from the fact that we wish to solve problems more efficiently, but also from the fact that it is starting to become increasingly hard to do better in classical computing. Practically, classical computers are composed of transistors that act as a switch to turn passing current on or off. Following Moore's law, the number of transistors in an integrated circuit seems to double about every two years. This is due to the fact that transistors are manufactured to be increasingly small. They have now arrived close to their potential, reaching an atomic level [15]. But, at this size, an interesting phenomenon happens, that of *electron tunneling*. Electron tunneling occurs when an electron passes through an insulating layer, or a classically insurmountable barrier. Then, it becomes hard to control the current when the gate randomly allows electrons to pass, or it becomes hard to distinguish leakage from useful current. Therefore, a new form of computation is required to circumvent this issue, the quantum computer, which uses this scale to its advantage.

2.2 Theory of Quantum Computing

The foundational unit of Quantum Computing is the *qubit*, or quantum bit. Just like a classical bit, which can be in the state 0 or 1, a qubit can have the state $|0\rangle$ and $|1\rangle$, which are called the Z-basis, or *computational* basis:

$$|0\rangle := \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle := \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The particularity of a qubit is that it can be in other states than just $|0\rangle$ or $|1\rangle$. A qubit can then be in a linear combination of states, which puts the qubit in *superposition*. Essentially, the state $|\psi\rangle$ of the qubit is represented using a unit vector in the complex vector space $\mathbb{C}^2$ [16]. The quantum state, under the Z-basis can be written as $\psi = \alpha\,|0\rangle + \beta\,|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. This means that, when a qubit is measured under superposition, the probability of getting the state $|0\rangle$ is $|\alpha|^2$ and the probability of getting the state $|1\rangle$ is $|\beta|^2$. Another way of writing the state of a qubit is using amplitude and phase parameters. The state $|\psi\rangle$ becomes:

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

where $\theta \in [0, \pi]$ represents the amplitude and $\phi \in [0, 2\pi[$, the phase of a qubit.

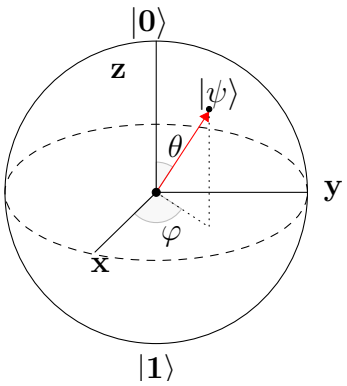


Figure 2.1: The Bloch Sphere

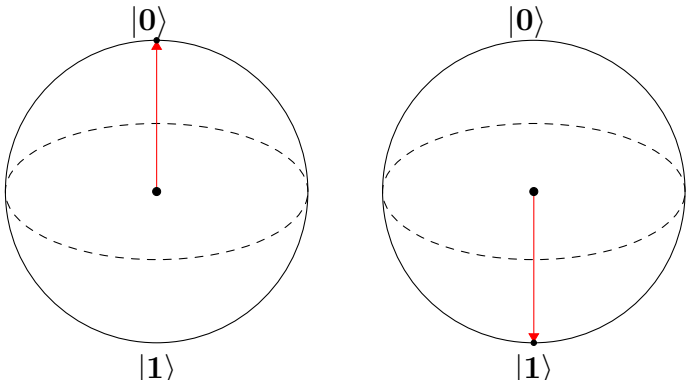


Figure 2.2: State $|0\rangle$ and $|1\rangle$

A popular method of representing a qubit is via the *Bloch Sphere*, as shown in figure 2.1, with $|0\rangle$ on the north pole and $|1\rangle$ on the south pole (figure 2.2). This representation fits the format of a qubit state which uses amplitude and phase, with θ and ϕ, respectively, on the Bloch sphere. The state of a qubit will then lie on the surface of the Bloch sphere.

However, a single qubit is not very powerful by itself. Quantum Computing's full power comes into effect when multiple qubits interact together to form a complex system. The interactions between the multiple qubits can be described using the *tensor product* operation $\otimes$, which also corresponds to the *Kronecker product*. In a system with n qubits, the state $|\psi\rangle$ would belong to $(\mathbb{C}^2)^{\otimes n} = \mathbb{C}^{2^n}$. Then, for example, a two-qubit system in $\mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^4$ will have four computational basis states:

$$|00\rangle = |0\rangle \otimes |0\rangle = (1,0,0,0)^T \qquad |10\rangle = |1\rangle \otimes |0\rangle = (0,0,1,0)^T$$
$$|01\rangle = |0\rangle \otimes |1\rangle = (0,1,0,0)^T \qquad |11\rangle = |1\rangle \otimes |1\rangle = (0,0,0,1)^T$$

And the state $|\psi\rangle$ which describes the two-qubit system will be:

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

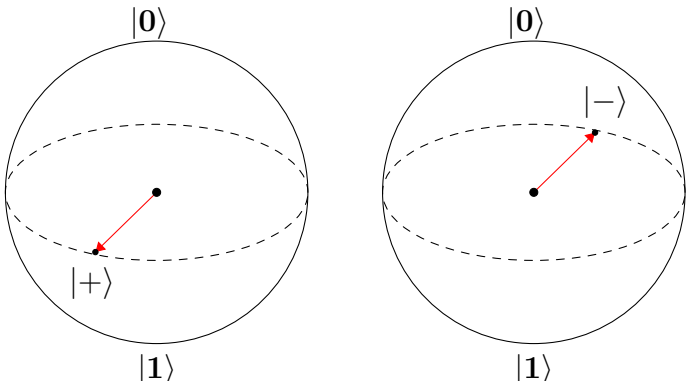


Figure 2.3: State $|+\rangle$ and $|-\rangle$

with $|\alpha_{00}|^2 + |\alpha_{10}|^2 + |\alpha_{01}|^2 + |\alpha_{11}|^2 = 1$.

2.2.1 Quantum Gates and Circuits

Quantum states on their own, once initialised, are not meaningful. However, they can be transformed and manipulated using quantum gates, which can be applied to one or multiple qubits, known as multi-qubit gates.

All quantum gates on quantum states $|\psi\rangle \in \mathbb{C}^{2^n}$ are actually unitary operations, or unitary matrices $U \in \mathbb{C}^{2^n \times 2^n}$. An operation U is unitary if $UU^\dagger = U^\dagger U = I$. Applying a quantum gate U will give the resulting state $|\psi'\rangle = U|\psi\rangle$.

Let us take, for example, the arguably most common gate, the Hadamard gate H which is represented by the following matrix:

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

which maps the computational Z-basis to another orthogonal basis called the X-basis with the two following states:

$$|+\rangle = H|0\rangle = \tfrac{|0\rangle+|1\rangle}{\sqrt{2}} \text{ and } |-\rangle = H|1\rangle = \tfrac{|0\rangle-|1\rangle}{\sqrt{2}}$$

The position of the X-basis on the block sphere is shown in figure 2.3.

An ensemble of quantum gates, or unitary operations, along with initialised qubit states (usually $|0\rangle^{\otimes n}$ for n qubits), will form a quantum circuit. The states and gates are connected via wires, which can not only represent physical wires but also the passage of time or particles in more abstract representations. Circuits are meant to be read from left-to-right. A simple example is shown in figure 2.4, which shows a 3-qubit circuit along with two arbitrary

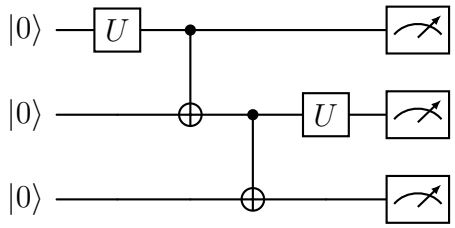


Figure 2.4: Example of a Quantum Circuit

single-qubit U gates and two $CNOT$ two-qubit gates. The example circuit also ends with measurement gates which will collapse the state of each measured qubits into classical bits.

Essentially, using the Bloch sphere representation, a quantum gate will move the state of a qubit, rotating it around that sphere, depending on the specific gate used. A quantum circuit will then rotate an ensemble of qubits to solve a specific task. Some more common quantum gates are the *Pauli operations*:

- The X gate which performs a 180° rotation along the X axis
- The Z gate which performs a 180° rotation along the Z axis
- The Y gate which performs a 180° rotation along the Y axis

with the corresponding unitary matrices:

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Z := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad Y := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Another important gate is the $CNOT$ gate (Controlled NOT gate), or the controlled Pauli-Z gate, which is a two-qubit gate. It involves a *controlled* qubit, represented by a $\bullet$ on the wire corresponding to the desired qubit, and a target qubit, represented by a $\oplus$. The two are then connected by an additional line, as shown in the example circuit in figure 2.4. Essentially, an X gate is applied to the target qubit, if and only if the controlled qubit is $|1\rangle$. The corresponding unitary matrix is then:

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

The $CNOT$ gate is essential, along with the Hadamard gate, to entangle two qubits where the $CNOT$ gate has been applied.

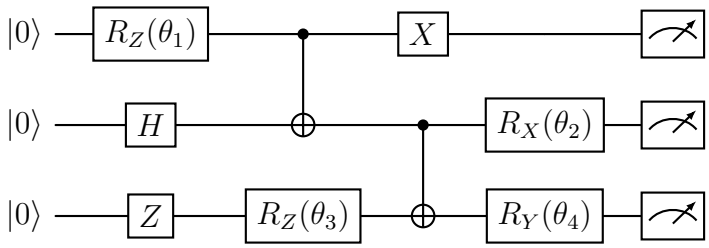


Figure 2.5: Example of a PQC

2.2.2 Parametrised Quantum Circuits

In some cases, it is required to have rotations by angles other than 180°. Three rotation gates, $R_X(\theta)$, $R_Z(\theta)$ and $R_Y(\theta)$, exist which will apply rotations along the X, Z or Y axis with a specific specified angle θ.

The corresponding unitary matrices are:

$$R_X(\theta) := \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$
$$R_Z(\theta) := \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$$
$$R_Y(\theta) := \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

If $\theta = 180°$, then the same Pauli operations for X, Z and Y defined above are achieved.

Having rotations be dependent on parametrised angles allows for much more customisability in the quantum circuit. If a circuit is built using parametrised rotational gates, it then becomes a *Parametrised Quantum Circuit* (PQC) [17], such as the one shown in figure 2.5.

The circuit shown in figure 2.5 is dependent on the parameters $\overrightarrow{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4)$. Since these parameters are not fixed, they can be changed to result in different measurements at the end of the circuit. When abstracting all of the parametrised gates into one unitary multi-qubit gate $U(\overrightarrow{\theta})$, where $\overrightarrow{\theta}$ is the vector describing all the parameters in the system, the state becomes $\left|\psi(\overrightarrow{\theta})\right\rangle = U(\overrightarrow{\theta})\left|\psi_0\right\rangle$, where $\left|\psi_0\right\rangle$ is the initial state.

However, as remarked earlier, the number of qubits in quantum systems is currently relatively small due to the noisy nature of NISQ devices; thus, PQC circuits are required to be shallow.

2.3 Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence that aims to mimic human behaviour through machines. Machines learn to solve complex tasks similarly to humans. Due to its great predictive capability, Machine Learning has its place in a wide range of applications, such as Natural Language Processing or computer vision.

All machine learning models start with the dataset on which they will be trained. The dataset is composed of data points (such as images or numerical values) and a way of describing those data points (like classes or more numerical values to indicate linear progression). The bigger the dataset, the better. The model will then parse the data, training itself to find patterns. Usually, a section of the dataset that has not been fed to the model for training will be passed after training, to test the model's accuracy. Afterwards, performance is tested on data that the model has never seen before. Changing hyperparameters of the model, either manually or automatically, could help the model reach a higher performance, as well as hinder it.

A Machine Learning system can serve different purposes: it can be *descriptive*, explaining past events based on the data; it can be *predictive*, forecasting future events using the data; or it can be *prescriptive*, providing recommendations for actions based on the data [18].

2.3.1 Neural Networks

There exist multiple Machine Learning methods to solve different tasks, such as Linear Regression, Decision Trees, K-nearest Neighbour, Random Forest, Kernel Methods and Deep Neural Networks, just to name a few.

However, with different methods come trade-offs in interpretability and performance. Deep Neural Networks usually have the best performance but with the downside of being the least interpretable [19], acting as a black box.

Neural Networks, as shown in figure 2.6, are a webbed structure composed of three parts, the input layer, the hidden layers and the output layer, with varying nodes (or neurons) in each section. All the nodes in a layer are connected to all the nodes on either side's layer. Additionally, a weight is attached to each connection between the nodes in the network that determines how much information passes through each node and will change numerically as the network trains for a task. The aim of a neural network is to make decisions in a manner similar to the human brain, by using processes that mimic the way biological neurons work together.

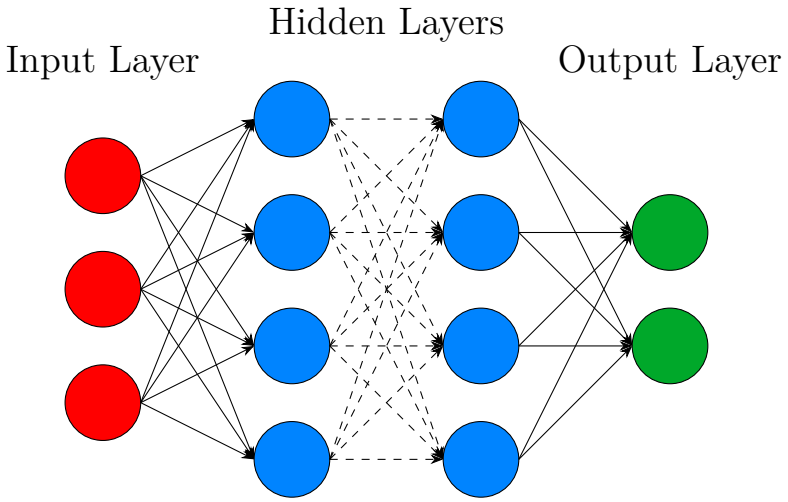


Figure 2.6: Example Neural Network

The input layer of the network is where the input data is passed through. The hidden layers are where the core of the learning and computation occurs. When a neuron receives an input from a previous layer, it is passed through an activation function that will determine whether the neuron is important to the process or not. Adding more hidden layers can increase the depth and complexity of the network. However, it is important to note that simply adding more hidden layers will not necessarily increase performance and could lead to overfitting, which occurs when the model fits too closely to the training dataset and performs poorly on unseen datasets. The output layer is where the final prediction is produced. The number of output nodes will vary depending on the specific task; for example, a classification task could have an output node for each class [20].

2.3.2 Natural Language Processing

As mentioned previously, one substantial application of Machine Learning is Natural Language Processing (NLP). It is a field at the intersection of both computer science and linguistics that aims to computationally understand a language either spoken or, most commonly, written. With an understanding of a language, a computer is then able to extract meaning from a corpus of texts or speeches [21]. Common tasks in NLP include speech recognition, text classification and natural language generation. Text classification, specifically, tries to organise sentences or texts into specific categories. For example, in sentiment analysis the task is to to predict whether a text has positive or negative sentiment.

Typically, textual NLP involves the following stages:

1. A pre-processing stage, which cleans the corpus of text from any unwanted words that do not exist in the given language. Depending on the task, we remove small words that would not add any meaning to the task at hand. For example, in *sentiment analysis*, the words "the" or "I" do not add any sentiment to the overall phrase; on the other hand, words like "best" or "bad" are essential to the task. We also removed numerals and punctuation most of the time, but they are crucial for tasks like *text generation*.
2. Occasionally, a dictionary stage which includes all individual elements from the corpus along with a unique index. Depending on the specific task, these elements could be whole words or specific characters.
3. An embedding stage, which transforms clean text into a form that can be processed by computers. Two elementary methods are:

   - Bag-of-Words: creates a zero vector of length the size of the dictionary. Then, for every word in a sentence, increments the element of the vector with the given index from the dictionary.
   - One-hot-vector: converts every word in the sentence with a zero vector of the size of the dictionary and only one 1 at the index given by the dictionary.

   It is important to note that not all embedding methods are equivalent and should be chosen carefully depending on the task at hand. Bag-of-words is useful for tasks like *text clustering*, where documents are grouped based on their content since it captures the frequency of words. However, it tends to be worse for *sentiment analysis* since it does not capture the structure of sentences, so the sentencse "I do not not like apples and oranges" and "I do not like apples and not oranges" would be equivalent but their meaning is not. Models that use one-hot-vectors would learn that two consecutive "not"s would be positive.
4. A machine learning stage which applies ML models to train for the task by optimising parameters to achieve the greatest accuracy.

As it turns out, neural networks are able to reach high performances on NLP tasks, and are a necessary step to apply Quantum Computing resources to Machine Learning tasks

2.4 Quantum Machine Learning

Quantum Machine Learning (QML) aims to achieve quantum advantage over classical Machine Learning, but like any quantum computation, it is currently constrained by NISQ computers which are very susceptible to noise. Furthermore, some classes of QML models have been shown to have higher expressibility compared to their classical counterparts, with possible quantum advantage [22]. However, this does not directly translate into improved results [23]. While there exists a wide range of theoretical QML models to choose from depending on the required tasks [24], in this dissertation, we will focus on Quantum Recurrent Neural Networks, which will be explored further in chapter 4.

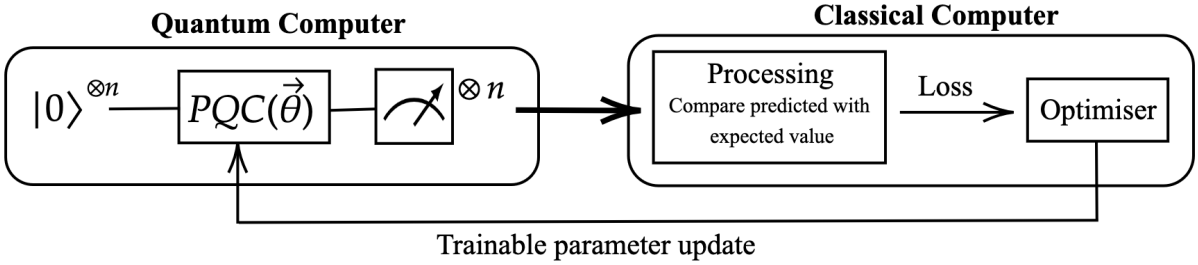


Figure 2.7: Variational Quantum Algorithm framework [25]

VQA Framework

This research centres on solving QML tasks by using *Variational Quantum Algorithms* (VQA), which do not solely use quantum computers. Rather, they consist in a hybrid approach on both a quantum device and a classical device, as seen in figure 2.7. The aim of VQAs is to train the parameters $\overrightarrow{\theta}$ of its PQC until optimal parameters $\overrightarrow{\theta}^*$ are found.

Unless a pre-trained model is used, in the 0^{th} step, usually the VQA will start with random angle parameters $\overrightarrow{\theta}_0$ for a selected PQC. The choice of the PQC is important as different structures will result in different performances. A Quantum Computer is then initialised, usually with $|0\rangle^{\otimes n}$ for an n qubits quantum computer. Following this, the selected PQC is applied on the initialised qubits with the angle parameters $\overrightarrow{\theta}_0$. The qubits are measured after the PQC and the measurements are then passed through to the classical device which will compute the loss, by quantifying the difference between the predicted measurements and the actual target measurements. The loss is passed through to an optimiser that will compute new parameters $\overrightarrow{\theta}_1$.

The same cycle then starts with the new parameters $\overrightarrow{\theta}_1$. New parameters $\overrightarrow{\theta}_i \xrightarrow{\text{Optimiser}} \overrightarrow{\theta}_{i+1}$ are computed until the optimal parameters $\overrightarrow{\theta}^*$ are found.

QML Optimiser

There exist multiple optimisers [26–28] which can be used for VQAs, each with their respective performance. Since VQAs heavily rely on classical optimisation with its optimiser, there is an important need for a performant optimiser. In this research, a *Gradient Descent* [29] based optimiser is used, more precisely the *Adaptive Moment Estimation* (Adam) [30] optimiser, which is an extension of *Stochastic Gradient Descent* [31] method.

But what about multiple neural network layers?

In classical Machine Learning, neural networks can add more abstraction and complexity by adding more hidden layers. In QML with VQAs, since PQCs are used as the neural network substitute, the same principle applies. If a higher number of layers is required, then more PQCs can be added, with their own specific set of parameters. Only the Quantum Computer side of the VQA is affected. Figure 2.8 shows an example of a VQA with k layers where k PQCs have been stacked together with their own respective parameters $\overrightarrow{\theta}^i$ for $i \in [\![1, k]\!]$. Since adding more layers increases the number of parameters to be trained, this increases the computational load on the optimiser. Therefore increasing the number of layers can have a drastic impact on the training time of the QML model.

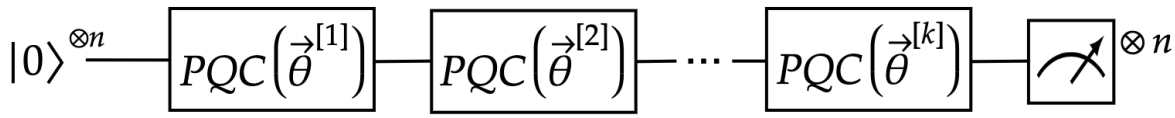


Figure 2.8: Layered PQCs

2.4.1 Quantum Natural Language Processing

Quantum Natural Language Processing (QNLP) aims to carry out the same tasks as classical NLP by harnessing the principles of quantum computing. The need for QNLP stems from the need of solving NLP tasks with novel methods, as classical NLP often suffers from poor performance and accuracy as the complexity of the problem increases.

It is important to note that QNLP is still in its early stages, but practical applications are currently being explored with methods such as Quantum Recurrent Neural Networks (QRNN) [32]. It is the main focus of this research and will be discussed in chapter 4. As well as Compositional Distributional Circuits [33] (DisCoCirc) model, a framework that transforms sentences into quantum circuits using category theory.

3

Classically Simulable Free Fermionic Circuits

Contents

3.1 What does classically simulable mean?

A quantum system is classically simulable if it can be simulated on a classical computer. Classical simulation has two meanings: a weak and a strong sense [34]. In the weak sense, weak simulations sample an output of a quantum circuit's measurements with a correct probability distribution. The weak classical simulator does not need to replicate the quantum states exactly, but rather, the outputs need to be indistinguishable from what a quantum circuit would produce. In the strong sense, strong simulations precisely calculate the probabilities and measurements of a quantum circuit. This requires a strong classical simulator to understand the quantum state's evolution and output probabilities. This distinction is essential as it highlights the different levels of complexity of quantum algorithms.

However, given enough time, any quantum algorithm can be simulated on a classical computer. Classically simulable has to include a further notion, that of **efficient simulation**. A weak or strong simulation is considered efficient if the classical computation that simulates runs in classical polynomial resources $poly(N)$, with N the number of operations. Strong quantum circuit simulations are usually considered inefficient, but multiple efficient methods have been developed given that the simulation process is sufficiently restricted [35].

This dissertation uses a class of quantum circuits that simulate free fermions. These free fermionic circuits are based on 2-qubit matchgate circuits that are strongly classically simulable. Matchgates are also efficiently classically simulable, which makes them ideal for use on classical devices. Furthermore, free fermionic circuits are the only known quantum system classically simulable while maintaining continuous angles in the matchgates, as opposed to, for example, the Clifford circuit set, which we know is classically simulable from the Gottesman-Knill theorem [36]. The Clifford circuit set comprises the Hadamard gate, a phase shift gate and the $CNOT$ gate. As can be seen, the set is discrete and leaves no room for continuous optimisation. This makes free fermionic circuits suitable in a Machine Learning setting that aims to optimise the circuit's continuous angles. This is further established by the fact that in most cases, if a parametrised circuit is trainable, it is efficient to simulate classically [37].

For the rest of this dissertation, when we refer to classical simulation, we imply efficient simulation.

3.2 Free Fermions

In this section, we will present why free fermionic circuits are efficiently classically simulable. First, we will introduce the concept of fermions, then expand to free fermions and then the classically simulable matchgates.

3.2.1 What are Fermions

In quantum physics, describing physical phenomena is usually done on a lattice consisting of a collection of $\mathcal{L}$ positions arranged in space [38]. Given a particle, described by the vector space $\mathcal{H}$ and a natural set of observables placed on the lattice, the full system of particles, or Hamiltonian H, can be described by taking the tensor product of the particles' vector space $H = \otimes_{p\in\mathcal{L}}\mathcal{H}_p$. We can omit the $\otimes$ symbol when it is not ambiguous.

Before delving into free fermions, it is necessary to understand what a single spinless fermion is and work up. In particle physics, a fermion, coined after Enrico Fermi by Paul

Dirac [39], can be an elementary particle, such as an electron, or a composite particle, such as protons. Spinless fermions are represented using the vector space with two states $\mathcal{H} = \text{span}\{|0\rangle, |1\rangle\}$, where $|0\rangle$ represents the absence of a fermion and $|1\rangle$ represents its presence.

Given a lattice of N points that can hold fermions, a set of *creation operators* $\{a_j^\dagger\}_{j=1}^N$ which act as:

$$|q_1, \cdots, q_N\rangle = (a_1^\dagger)^{q_1} \cdots (a_N^\dagger)^{q_N} |0\rangle$$

This means that, for example, on a lattice with 5 positions, $|10101\rangle = a_1^\dagger a_3^\dagger a_5^\dagger |0\rangle$ a fermion is created in positions 1, 3 and 5 of the lattice.

Respectively, *annihilation operators* are defined by taking the Hermitian conjugate of the creation operators. Essentially:

- Creation operators $a_j^\dagger$ adds a fermion to the j^{th} lattice site.
- Annihilation operators a_j removes fermions from the j^{th} lattice site.

Additionally, these two operators must satisfy canonical anticommutation relations:

$$\begin{aligned}
\{a_i, a_j\} &:= a_i a_j + a_j a_i = 0 \\
\{a_i^\dagger, a_j^\dagger\} &= a_i^\dagger a_j^\dagger + a_j^\dagger a_i^\dagger = 0 \\
\{a_i^\dagger, a_j\} &= a_i^\dagger a_j + a_j a_i^\dagger = \delta_{ij}
\end{aligned}$$

where δ_{ij} is the *Kronecker delta*, which is defined as:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

Fermions in this model are known as complex fermions. These are fermions that have distinct particles and antiparticles, like electrons and positrons. On the other hand, there exists another fermionic model called Majorana fermions, named after Ettore Majorana, where the particles are their own antiparticles. This means that a particle is indistinguishable from its antiparticle. Majorana used this model to describe electrically neutral particles [40]. The corresponding operators are described using the creation and annihilation operators from Dirac fermions:

$$\begin{aligned}
\gamma_{2j-1} &:= a_j^\dagger + a_j \\
\gamma_{2j} &:= i(a_j^\dagger - a_j)
\end{aligned} \tag{3.1}$$

Essentially, a fermionic qubit in position j is described by two Majorana fermions γ_{2j-1} and γ_{2j-1}. Majorana fermions must satisfy the following relations

$$\{\gamma_i, \gamma_j\} := 2\delta_{ij} \tag{3.2}$$

This indicates that the square of a Majorana operator, γ_i^2, is the identity, since $\{\gamma_i, \gamma_i\} = \gamma_i\gamma_i + \gamma_i\gamma_i = 2\gamma_i\gamma_i = 2\gamma_i^2 = 2\delta_{ii} = 2I$. This indeed shows that Majorana fermions are their own antiparticles. Using Majorana fermions will be useful for free fermions as it offers a simplified representation.

Introducing spin

The fermions introduced previously were spinless. However, studying the spin of particles is essential when explaining the structure of elements and their properties, like magnetic properties.

In fact, fermions are considered to have spin-$\frac{1}{2}$ [41]. However, the link between the fermions and spin-$\frac{1}{2}$ is not direct. Spin-$\frac{1}{2}$ particles are spanned by the orthonormal basis $\mathcal{H} = \text{span}\{|\uparrow\rangle, |\downarrow\rangle\}$ which indicates in which directions particles are spinning. Pauli operators are the observables that will describe along which axis the particle is spinning:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Now the question is how to get from a system of spin-$\frac{1}{2}$ particles to a system of fermions. This is done via the *Jordan-Wigner transformation*, proposed by Pascual Jordan and Eugene Wigner [42], which introduces a correspondence between the two. The transformation introduces the assignments:

$$\gamma_{2j-1} = \left(\prod_{k=1}^{j-1} Z_k\right) X_j \qquad \gamma_{2j} = \left(\prod_{k=1}^{j-1} Z_k\right) Y_j \tag{3.3}$$

Inverting the transformation, we obtain:

$$X_j = (-i)^{j-1} \left(\prod_{k=1}^{2(j-1)} \gamma_k\right) \gamma_{2j-1} \tag{3.4}$$

$$Z_j = -i\gamma_{2j-1}\gamma_{2j} \tag{3.5}$$

$$Y_j = (-i)^{j-1} \left(\prod_{k=1}^{2(j-1)} \gamma_k\right) \gamma_{2j} \tag{3.6}$$

Now let us show that equations 3.3 obey the anticommutation relations from equation 3.2.

First, with $\{\gamma_{2j-1}, \gamma_{2j}\}$, let us compute $\gamma_{2j-1}\gamma_{2j}$:

$$\begin{aligned}
\gamma_{2j-1}\gamma_{2j} &= \left(\prod_{k=1}^{j-1} Z_k\right) X_j \left(\prod_{k=1}^{j-1} Z_k\right) Y_j \\
&= X_j \left(\prod_{k=1}^{j-1} Z_k^2\right) Y_j \\
&= X_j Y_j \\
&= iZ_j
\end{aligned}$$

Similarly, since $XY = -YX$, we have that $\gamma_{2j-1}\gamma_{2j} = -iZ_j$, Therefore:

$$\begin{aligned}
\{\gamma_{2j-1}, \gamma_{2j}\} &= \gamma_{2j-1}\gamma_{2j} + \gamma_{2j}\gamma_{2j-1} \\
&= iZ_j - iZ_j \\
&= 0
\end{aligned}$$

So $\{\gamma_{2j-1}, \gamma_{2j}\} = 0 = \{\gamma_{2j}, \gamma_{2j-1}\}$.

Now, we need to verify that $\{\gamma_{2j-1}, \gamma_{2j-1}\} = \{\gamma_{2j}, \gamma_{2j}\} = 2I$.

Let us start with $\{\gamma_{2j-1}, \gamma_{2j-1}\}$:

$$\begin{aligned}
\{\gamma_{2j-1}, \gamma_{2j-1}\} &= 2\gamma_j^2 \\
&= 2\left(\left(\prod_{k=1}^{j-1} Z_k\right) X_j\right)^2 \\
&= 2\left(\prod_{k=1}^{j-1} Z_k^2\right) X_j^2 \\
&= 2II = 2I
\end{aligned}$$

Therefore $\{\gamma_{2j-1}, \gamma_{2j-1}\} = 2I$, similarly for $\{\gamma_{2j}, \gamma_{2j}\} = 2I$.

Thus, equations 3.3 obey the anticommutation relations.

3.2.2 What are Free Fermions

In the class of fermions, there exists a subset of fermions that do not interact with each other; these fermions are called *free fermions*. Physically, these can be considered as weakly interacting electrons.

Free Fermionic systems can be written in terms of a quadratic fermionic Hamiltonian. Using Majorana fermions, the Hamiltonian takes the following form:

$$H = -i \sum_{jk} h_{jk} \gamma_j \gamma_k$$

Where h_{jk} is a real antisymmetric $2N \times 2N$ matrix and N is the number of qubits in our system. Dealing with a matrix h of dimension $2N \times 2N$ means that our state space is greatly reduced from $2^N \times 2^N$ in a regular fully quantum circuit.

3.2.3 Matchgates

In 2002, Leslie G. Valiant proposed a class of quantum circuits based on matchgates [43]. This result was then related to free fermionic quantum computation by Knill [44] and DiVencenzo and Terhal [45].

Matchgates in the context of free fermionic Quantum Computing are 2-qubit gates $G(A, B)$ of the form:

$$G(A,B) = \begin{bmatrix} a_{11} & 0 & 0 & a_{12} \\ 0 & b_{11} & b_{12} & 0 \\ 0 & b_{21} & b_{22} & 0 \\ a_{21} & 0 & 0 & a_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ and } B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

where the constraints $\det(A) = \det(B)$ and $A, B \in U(2)$ have to be met for A and B. Thus the action of the matchgate $G(A, B)$ results in A acting on the even parity subspace $span(|00\rangle, |11\rangle)$ and B acting on the odd parity subspace $span(|10\rangle, |01\rangle)$. A matrix $M \in U(2)$ means that $MM^\dagger = M^\dagger M = I_2$ where $M^\dagger$ represents the Hermitian conjugate of M.

Using theorem 1 of Richard Jozsa and Akimasa Miyake's paper on *Matchgates and classical simulation of quantum circuits* [46]:

Theorem 1. *Consider any uniform quantum circuit family comprising of only $G(A, B)$ gates such that:*

- *The $G(A, B)$ gates act only on nearest neighbour (n.n.) lines.*
- *The input state is any product state.*
- *The output is a final measurement in the computational basis on any single line.*

Then the output may be classically efficiently simulated.

We can observe a first glimpse of classical simulation. For their part, Jozsa and Miyake characterise the classical simulation property as a possibility, not a certainty. In the following sections, we will see that the output of a circuit comprised of only matchgates can be obtained by classical simulation.

Linking free fermions to matchgates

In order to link free fermions to matchgates, we can use theorem 5 of Richard Jozsa and Akimasa Miyake's paper on *Matchgates and classical simulation of quantum circuits* [46].

Theorem. *Let $H = -i\sum_{jk} h_{jk}\gamma_j\gamma_k$ be any quadratic Hamiltonian with corresponding Gaussian gate $V = e^{iH}$ on n qubits. Then V as an operator on n qubits is expressible as a circuit of $O(n^3)$ nearest-neighbour $G(A, B)$ gates.*

This guarantees that free fermionic systems can be matched to a system of nearest-neighbour matchgates.

3.2.4 Matchgates are classically simulable

Next, we will present why matchgates are classically simulable. But first, we need to introduce the problem of finding perfect matchings.

Perfect Matchings

Given a graph $G = (V, E)$, a matching is a subset of E such that no two edges share a vertex. A perfect matching of a graph G is a subset $M \subset E$ such that all the vertices of V are used [47].

Given the graph in figure 3.1, a perfect matching can be found by selecting the subset $M = \{(v_1, v_3), (v_2, v_4), (v_5, v_6)\}$. With these matchings, all the vertices in V have been selected only once.

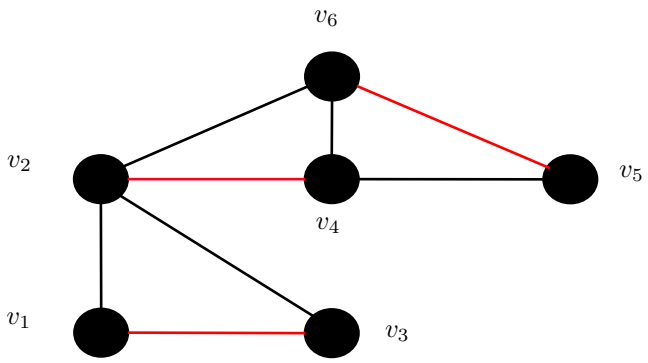


Figure 3.1: Example graph 1: Perfect Matching graph

However, with the graph in figure 3.2, no perfect matching can be found. For v_5 to be used, the edge (v_4, v_5) has to be selected. This leaves the triangle of $\{(v_1, v_2), (v_2, v_3), (v_1, v_3)\}$ and no perfect matching can be found for a triangle since one vertex will always overlap

when selecting two vertices. One simple way of checking if a graph contains a perfect matching is by counting the number of vertices. If a graph has an odd number of vertices, then no perfect matching can be found. With perfect matching comes weighted perfect

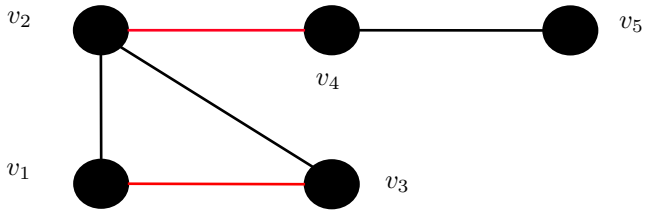


Figure 3.2: Example graph 2: Non-Perfect Matching graph

matching, where weights are attached to the edges of a graph. The weight of a perfect matching is the product of all the weights of the edges in the matching.

To connect matchgates to perfect matching, the problem of weighted perfect matching has to be extended to another problem of counting weighted perfect matching in a graph.

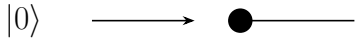
Counting the number of perfect weighted matchings means summing the product of the weights for each perfect matching. As such, for a weighted graph G with weights w, the number of weighted perfect matching is:

$$perfM(G) = \sum_{\text{(perfect matching } M)} \prod_{(ij)\in M} w_{ij} \tag{3.7}$$

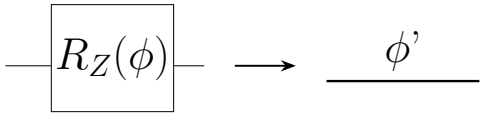
Connecting matchgates to perfect matchings

In order to connect matchgates to the problem of perfect matching, a matchgate circuit needs to be associated to a graph. The graph can then make use of the FKT algorithm, presented next, to find the number of perfect matching. For this section, we will use the matchgate detailed in full in chapter 6.1 and composed of an R_{XX} and R_Z gate. The corresponding association is as follows:

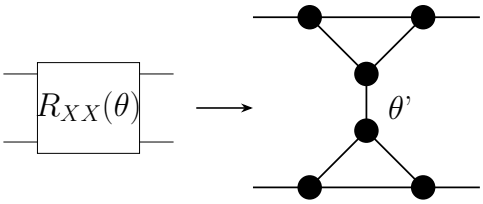
For a qubit that is initialised to $|0\rangle$, this turns into a vertex with an edge:



For the Z rotation gate $R_Z(\phi)$, this is represented as an edge with a weight ϕ' corresponding to the rotational angle ϕ:



As for the $R_{XX}(\theta)$ gate, for $\theta \neq 0$ and $\theta \neq \pi$, this corresponds to the following graph, with two pairs of vertices for the two qubits, connected by another pair of vertices adjoined by an edge of weight θ' corresponding to the angle θ.



The final state ψ after having applied the graph with the complete matchgate with gates $R_{XX}(\theta)$, $R_Z(\phi_1)$ and $R_Z(\phi_2)$ results in the graph shown in figure 3.3. Any edge that is not assigned a weight is given a weight of 1.

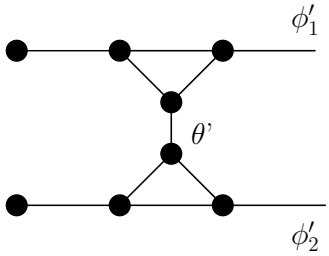


Figure 3.3: Planar graph of matchgate

The measurement of a circuit after the matchgate can be obtained by calculating the expectation value of Z_j, since R_Z gate was applied last, where j is the index of the qubit to be measured. The expected value of the measurements in the state ψ at qubit j can be obtained with the following equation:

$$\langle Z_j \rangle = \langle \psi | \, Z_j \, | \psi \rangle \tag{3.8}$$

Transforming equation 3.8 into a corresponding graph is shown below. To obtain the expectation at a qubit position j, means joining the graph of figure 3.3 with its Hermitian conjugate, i.e. the equivalent to mirroring the graph and taking the complex conjugate of the weights. Then we assign a weight of $\alpha = -1$ only on the j^{th} wire.

The resulting graph is shown in figure 3.4.

This graph is planar since no edges intersect. Notice that the weight ϕ_2' disappeared when combining the graph of $|\psi\rangle$ with its Hermitian conjugate $\langle\psi|$. This would imply that measuring the first and second qubit would result in the same expectation. This is because the measurement probabilities are symmetric in a 2-qubit circuit with a singular matchgate, with the same expectation is measured for both qubits. To obtain a circuit

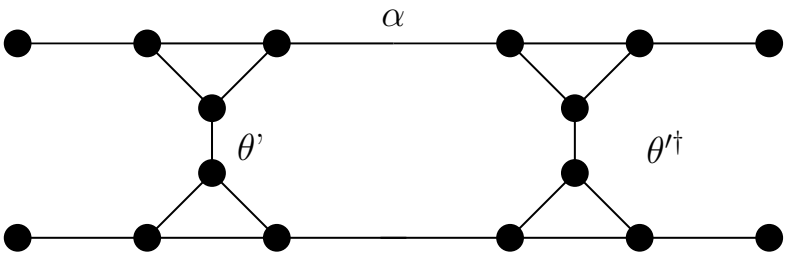


Figure 3.4: Planar graph for expectation

that results in non-symmetric probabilities, it is necessary to add at least one more qubit and one overlapping matchgate with the extra qubit. For a circuit containing matchgates with a number of qubits bigger than two, the resulting planar graph will require the necessary assemblage of the graph in figure 3.3.

To compute the expectation measurements, we can use Valiant's **Measurement Theorem** from his paper on *Quantum computers that can be simulated classically in polynomial time* [48]. The theorem states that the expectation value of a matchgate equals the number of weighted perfect matchings, over a constant C, of the corresponding planar graph.

Therefore:

$$\langle Z_j \rangle = \frac{1}{C} \sum_{(\text{perfect matching } M)} \prod_{(ij)\in M} w_{ij} \tag{3.9}$$

where C is a constant determined by the number of gates in the circuit.

What about classically simulable?

The last step in showing that free fermions are classically simulable involves linking the problem of finding the number of perfect matches to classical simulations.

For this, we can use the *Fisher–Kasteleyn–Temperley (FKT) algorithm* which is used to find the number of perfect matching of a planar graph in polynomial time. The algorithm was found independently in 1961 by Pieter Kasteleyn [49], and Neville Temperley and Michael Fisher [50] who found the number of domino tilings in an $m \times n$ rectangle. This research was motivated by dimers, which are pairs of connected atoms or pairs of adjacent sites occupying a lattice. The dimer model then aims to determine how dimers can be arranged on a lattice. The link with dimers and dominos becomes apparent as dimers are just dominos with each side of a domino being a different atom or point connected by the domino structure. This is also equivalent to the number of perfect matching in

an $m \times n$ lattice since a domino is equivalent to a matching. In 1963, Kasteleyn then generalised this result for all planar graphs [51].

The steps for the FKT algorithm of a graph $G = (V, E, W)$, with W a weight function that assigns a weight to an edge, assuming $|V|$ is even, are as followed:

1. Compute the planar embedding of G, ensuring edges only intersect at their endpoints.
2. Compute the spanning tree $S = (V_S, E_S)$ of G.
3. Compute a dual graph $D = (V_D, E_D)$ of G, such that at first $|E_D| = 0$, such that each face of G represents a vertex in V_D.
4. Create an edge by connecting two vertices of D if the faces of G, that are represented by the two vertices in V_D, are not touching at an edge in the spanning tree S. The edges of D should not cross the spanning tree S.
5. Assign orientations to the edges of G:
   - If an edge of G does not cross an edge of D, assign an arbitrary orientation.
   - Otherwise orient the edge so that the it contributes to an odd number of clockwise-oriented edges around each face of G.
6. Compute the skew-symmetric adjacency matrix A of G, such that $A_{i,j} = -A_{j,i}$ for every $i, j \in V_G$:
   - Create a 0-filled $n \times n$ matrix A where n is the number of vertices in G.
   - For every pair of vertices $i, j \in V_G$:
     - If the edge e_{ij} has an orientation from i to j, then set $A_{i,j} = W(e_{ij})$ weight of edge e_{ij}.
     - If the edge e_{ij} has an orientation from j to i, then set $A_{i,j} = -W(e_{ij})$.
     - Otherwise keep $A_{i,j} = 0$.
7. Compute the Pfaffian $pf(A) = \sqrt{\det(A)}$
8. The number of perfect matching of the graph G is the absolute value of Pfaffian $|pf(A)|$.

This algorithm is indeed executable in polynomial time since finding a spanning tree, assigning orientations to the edges of the graph, creating the skew-symmetric adjacency matrix, and computing the determinants can all be performed in polynomial time.

We have now shown that free fermionic circuits are classically simulable, by connecting free fermions to matchgates, translating matchagtes to the problem of finding the number of perfect matching, and found an algorithm that finds the number of perfect matching in polynomial time.

3.3 Methodology to implement Free Fermionic Systems in practice

A free ferminoic system with N qubits can be initialised from initial states $|0\cdots0\rangle$ with the $2N \times 2N$ covariance matrix Γ_0 where the matrix $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ is placed along the diagonal N times with the rest of the matrix set to 0, as shown:

$$\Gamma_0 = \left[\begin{array}{cc|ccc|cc} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & \ddots & & 0 & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots & \vdots \\ 0 & 0 & 0 & & \ddots & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 0 \end{array}\right]$$

This result comes from the formula of the covariance matrix with a state $|\psi\rangle$

$$\Gamma_{jk} = -\frac{i}{2} \langle\psi| [\gamma_j, \gamma_k] |\psi\rangle \tag{3.10}$$

The diagonal of Γ_0 is trivially 0 since Majorana fermions are their own antiparticles. However, for the first off-diagonals, $\Gamma_{2j-1,2j}$ and $\Gamma_{2j,2j-1}$ still need to be computed; the rest of the matrix set to 0 since the Majorana fermions are in different fermionic qubits. We can start by computing $[\gamma_{2j-1}\gamma_{2j}]$ using equations of the canonical anticommutation relations:

$$\begin{aligned} [\gamma_{2j-1}, \gamma_{2j}] &= \gamma_{2j-1}\gamma_{2j} - \gamma_{2j}\gamma_{2j-1} = \gamma_{2j-1}\gamma_{2j} + \gamma_{2j-1}\gamma_{2j} \\ &= 2\gamma_{2j-1}\gamma_{2j} \end{aligned}$$

So the covariance matrix:

$$\begin{aligned} \Gamma_{2j-1,2j} &= \frac{-i}{2} \langle\psi| [\gamma_{2j-1}, \gamma_{2j}] |\psi\rangle \\ &= \frac{-i}{2} \langle\psi| 2\gamma_{2j-1}\gamma_{2j} |\psi\rangle \\ &= -i \langle\psi| \gamma_{2j-1}\gamma_{2j} |\psi\rangle \\ &= \langle\psi| Z_j |\psi\rangle \end{aligned}$$

When initialising each qubit with $|0\rangle$, the covariance becomes:

$$\begin{aligned} \Gamma_{2j-1,2j} &= \langle 0| Z_j |0\rangle \\ &= \langle 0| \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} |0\rangle \\ &= 1 \end{aligned}$$

Therefore, the off-diagonal $\Gamma_{2j-1,2j} = 1$ for $j \in [|1, N|]$ and the other off-diagonal $\Gamma_{2j,2j-1} = -1$ since $[\gamma_{2j-1}\gamma_{2j}] = -[\gamma_{2j}\gamma_{2j-1}]$.

The evolution of the fermionic system under the action of h at time t [38] is given by:

$$\Gamma = e^{th}\Gamma_0 e^{-th} \tag{3.11}$$

In this context, h represents the $2N \times 2N$ antisymmetric matrix from the free fermionic quadratic Hamiltonian H where N is the number of free fermionic qubits. Finding the final evolution Γ includes computing all the evolutions with the individual matrix h for each gate that composes a matchgate.

To lay out the implementation, we will use the matchgate seen in chapter 6.1, with the reduced structure of figure 6.2. Using the inverse Jordan-Wigner transformation from equation 3.4 and 3.5 giving the equivalence of Pauli operators to Majorana fermions, we can express each of the gates in terms of Majorana operators.

The R_{XX} gates applied to subsequent qubits j and $j+1$ can be obtained by taking the product of the Pauli operators X_j and X_{j+1}, which results in:

$$\begin{aligned} X_j X_{j+1} &= (-i)^{j-1} \left(\prod_{k=1}^{2(j-1)} \gamma_k \right) \gamma_{2j-1} (-i)^j \left(\prod_{k=1}^{2j} \gamma_k \right) \gamma_{2j+1} \\ &= -i \left(\prod_{k=1}^{2(j-1)} \gamma_k^2 \right) \gamma_{2j-1} (\gamma_{2j-1}\gamma_{2j}) \gamma_{2j+1} \\ &= -i\gamma_{2j}\gamma_{2j+1} \end{aligned}$$

As for the R_Z gate applied to qubit j, this is directly given by the operator of equation 3.5:

$$Z_j = -i\gamma_{2j-1}\gamma_{2j}$$

In order to obtain the antisymmetric matrix h_{XX} corresponding to the application of an R_{XX} with angles $\theta_1^{XX}, \ldots, \theta_{N-1}^{XX}$, it is necessary to apply:

$$H_{XX} = \sum_{j=1}^{N-1} \theta_j^{XX} X_j X_{j+1} = -i \sum_{j=1}^{N-1} \theta_j^{XX} \gamma_{2j}\gamma_{2j+1}$$

Giving us the matrix:

$$h_{XX} = \left[\begin{array}{c|cc|cc|cc|c} 0 & 0 & 0 & \cdots & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & \theta_1^{XX} & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & -\theta_1^{XX} & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ \hline \vdots & 0 & 0 & \ddots & & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & 0 & 0 & \vdots \\ \hline 0 & \vdots & \vdots & \cdots & 0 & 0 & \theta_{n-1}^{XX} & 0 \\ 0 & \vdots & \vdots & \cdots & 0 & -\theta_{n-1}^{XX} & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & \cdots & 0 & 0 & 0 \end{array}\right]$$

Similarly, to obtain the antisymmetric matrix h_Z corresponding to the application of an R_Z with angles $\theta_1^Z, \ldots, \theta_N^Z$, it is necessary to apply:

$$H_Z = \sum_{j=1}^{N} \theta_j^Z Z_j = -i \sum_{j=1}^{N} \theta_j^Z \gamma_{2j-1}\gamma_{2j}$$

Giving us the matrix:

$$h_Z = \left[\begin{array}{cc|ccc|cc} 0 & \theta_1^Z & 0 & \cdots & 0 & 0 & 0 \\ -\theta_1^Z & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & \ddots & & 0 & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots & \vdots \\ 0 & 0 & 0 & & \ddots & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & 0 & \theta_N^Z \\ 0 & 0 & 0 & \cdots & 0 & -\theta_N^Z & 0 \end{array}\right]$$

Now, regarding the evolution, with h_{XX} and h_Z, we notice that we are required to exponentiate a matrix. Fortunatly, both h_{XX} and h_Z are block diagonal. For a block diagonal matrix of the form

$$M = \begin{bmatrix} A_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & A_N \end{bmatrix} \rightarrow e^M = \begin{bmatrix} e^{A_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & e^{A_N} \end{bmatrix}$$

For a matrix of the form $A = \begin{bmatrix} 0 & \theta \\ -\theta & 0 \end{bmatrix}$, taking the exponential of A, results in the following matrix:

$$e^A = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

This results in the following matrices:

$$e^{h_{XX}} = \left[\begin{array}{c|cc|cc|cc|c} 1 & 0 & 0 & \cdots & \cdots & 0 & 0 & 0 \\ \hline 0 & \cos(\theta_1^{XX}) & \sin(\theta_1^{XX}) & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & -\sin(\theta_1^{XX}) & \cos(\theta_1^{XX}) & 0 & \cdots & \cdots & \cdots & 0 \\ \hline \vdots & 0 & 0 & \ddots & & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & 0 & 0 & \vdots \\ \hline 0 & \vdots & \vdots & \cdots & 0 & \cos(\theta_{N-1}^{XX}) & \sin(\theta_{N-1}^{XX}) & 0 \\ 0 & \vdots & \vdots & \cdots & 0 & -\sin(\theta_{N-1}^{XX}) & \cos(\theta_{N-1}^{XX}) & 0 \\ \hline 0 & 0 & 0 & \cdots & \cdots & 0 & 0 & 1 \end{array}\right]$$

$$e^{h_Z} = \left[\begin{array}{cc|ccc|cc} \cos(\theta_1^Z) & \sin(\theta_1^Z) & 0 & \cdots & 0 & 0 & 0 \\ -\sin(\theta_1^Z) & \cos(\theta_1^Z) & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & \ddots & & 0 & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots & \vdots \\ 0 & 0 & 0 & & \ddots & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & \cos(\theta_N^Z) & \sin(\theta_N^Z) \\ 0 & 0 & 0 & \cdots & 0 & -\sin(\theta_N^Z) & \cos(\theta_N^Z) \end{array}\right]$$

with $e^{-h_{XX}} = (e^{h_{XX}})^T$ transpose of $e^{h_{XX}}$, similarly for e^{-h_Z}.

With these exponentials, the covariance matrix Γ_0 can be evolved with equation 3.11. Here $t = 1$ since we are only interested in one time step, i.e. one application of a matchgate. After applying the layer of R_{XX} gates, the resulting covariance matrix is:

$$\Gamma_1 = e^{h_{XX}} \Gamma_0 (e^{h_{XX}})^T \tag{3.12}$$

Then applying the layer of R_Z gates results in the final covariance matrix:

$$\Gamma = e^{h_Z} \Gamma_1 (e^{h_Z})^T \tag{3.13}$$

The final covariance matrix represents the free fermionic qubits' states after applying one layer of the matchgates with sets of angles through the N qubits. If more layers of matchgates are required, then we must apply the same process of applying the evolution of equation 3.12 followed by equation 3.13 iteratively with coraviance matrix Γ_i with different sets of angles at each iteration.

With the final layer of matchgates applied, a final output of the free fermionic circuit can be obtained by performing a Z measurement of the circuit, since the last gate applied is the R_Z gate. With free fermionic circuits the Z measurements of the final covariance matrix is obtained by selecting the elements $\{\Gamma_{2j-1,2j}\}_{j \in [|1,N|]}$.

We selected these elements are selected because using the expectation equation 3.8 on qubit j:

$$\begin{aligned}\langle Z_j \rangle = \langle \psi | \, Z_j \, | \psi \rangle &= -i \, \langle \psi | \, \gamma_{2j-1} \gamma_{2j} \, | \psi \rangle \\ &= \frac{-i}{2} \, \langle \psi | \, [\gamma_{2j-1}, \gamma_{2j}] \, | \psi \rangle \\ &= \Gamma_{2j-1,2j}\end{aligned}$$

is equivalent to selecting the matrix element $\Gamma_{2j-1,2j}$. Therefore, selecting the elements $\{\Gamma_{2j-1,2j}\}_{j \in [|1,N|]}$ gives the expected measurement for the whole circuit.

4

The Quantum Recurrent Neural Network QML Model

Contents

Nowadays, the ability to deal with and understand sequential data has become an essential capability as this type of data can be found everywhere in the systems supporting the activities of all economic agents. This is especially relevant for Natural Language Processing, which primarily deals with understanding language, whether spoken or more commonly written, and extracting meaning. This is because both text and audio are sequential. Fundamentally, a sentence is composed of words arranged in a sequence. What separates a sentence from a random assortment of words is that the words in a sentence have dependencies with each other. To understand a sentence, it is key to understand the dependencies between the words.

Machine Learning models that specifically deal with sequential data are known as *sequence models*. Sequence models do not solely deal with NLP tasks, and could also be applied to time-series data. Time-series data refer to data that is recorded over consistent intervals of time, for example, prices of shares in the stock market or varying temperatures throughout a time period. The main idea behind sequence models is that the data processed is not independently and identically distributed (i.i.d) but has dependencies, as expressed

earlier in this document. In practice, sequence models work by processing the data given as input in a sequential manner, making sure that the order and context of the data are taken into account. Machine Learning models that specifically deal with sequential data are known as *sequence models*. Sequence models do not solely deal with NLP tasks, and could also be applied to time-series data. Time-series data refer to data that is recorded over consistent intervals of time, for example, prices of shares in the stock market or varying temperatures throughout a time period. The main idea behind sequence models is that the data processed is not independently and identically distributed (i.i.d) but, on the contrary, has dependencies. In practice, sequence models work by processing the data given as input sequentially, making sure that the order and context of the data are taken into account.

Many complex sequence models exist in Machine Learning. Recurrent Neural Networks (RNNs) maintain hidden states but face difficulties in capturing long-term dependencies and coming with vanishing or exploding gradients, and too small to train or too large to cater for stability. Though, as they are foundational to the field of Machine Learning, we have selected this type of model as the central vehicle for our research. Alongside RNNs, there are other models: the Long Short-Term Memory (LSTM), a specialised type of RNNs, tackling processing issues with an architecture controlling the flow of information over long periods through the capture of dependencies, and the more recent Transformers, enabling the model to weigh the importance of the different parts of the input sequence when processing specific positions hence dynamically adjustable through the attention mechanism.

As with many classical Machine Learning models limited by classical processing power, problems arise when the data increases. In other words, there is a difficulty in capturing long-term dependencies. Furthermore, RNN are faced with additional problems of *vanishing* and *exploding* gradients [52], where gradients are either too small to train or too large, creating an unstable network.

With the advancements of quantum devices and quantum machine learning methods, Quantum Recurrent Neural Networks were developed to take advantage of quantum entanglement to enhance learning efficiency, stability and potentially solve the problems associated with RNNs. Multiple QRNNs have already been developed, with some aiming to construct canonical fully quantum QRNNs without relying on quantum-classical hybrid networks to ease the interface problem [53]. In this research, we will concentrate on the quantum-classical hybrid approach of QRNNs. The performance of some QRNNs has been verified concretely on varying classical sequential data with some testing better than their classical counterpart. One particular QRNN [1] stands out by having beaten RNNs with sequential tasks such as stock price prediction and Meteorological indicators.

In this chapter, we will explore how standard Quantum Recurrent Neural Networks are constructed. Firstly, it is important to start with the classical analog in order to understand how a neural network becomes "recurrent". Then we will explore how an RNN can be quantisised to be run on a quantum computer, expanding on the PQC heavy structure of Quantum Neural Network to create an equivalent "recurrent" relation.

4.1 Classical Recurrent Neural Networks

Recurrent Neural Networks expand on the structure of Neural Networks presented in chapter 2 with the common input, hidden and output layers. However, the difference lies in the hidden layers. What distinguishes an RNN from a regular NN is that it possesses some sort of "memory", a hidden state that is carried throughout the layers of the RNN. The hidden state will remember information about a sequence, as shown with the rolled diagram of figure 4.1 with the loop in the middle hidden state that carries through every time an input x_i is passed. There is a hidden layer for each element of the input sequence.

In a normal NN, the inputs and outputs are independent of each other. However, the output of an RNN depends on the previous layer. Let us suppose that a sequence of N elements is passed as input to the RNN shown in the unrolled version of an RNN in figure 4.1. The first element, in node x_1, is processed by the first hidden layer h_1. A first output y_1 can be obtained from the hidden layer, but the aim of sequence models is to get a result after having seen all the elements. Therefore, computation from the first hidden layer is stored in the hidden state, ready to be passed through to the second hidden layer. The second element from x_2 is passed through to h_2. But this time, the hidden layer h_2 also takes the output computation from h_1. Again, the output can be taken in y_2, or the process can be repeated until all the element of the sequence have been processed and a final output y_N is given. Usually, only the output y_N is considered as it is a result from the processing of all the elements of the sequence.

So far, RNNs with only one layer have been described, where hidden layers were arranged in 1-Dimension. But the term hidden layer can be ambiguous when dealing with deep RNNs, which are RNNs with a number of layers greater than one. There is indeed a hidden layer for each element of a sequence of N elements. But if an RNN with k total layers were to be constructed, then that row of N layers would be stacked k times on top of each other, forming a 2-Dimensional grid of layers, as shown in figure 4.2.

The process of multi-layer RNNs is very similar to single-layered RNNs. Let us suppose an RNN with k layer was to again process a sequence of N elements. The first word of

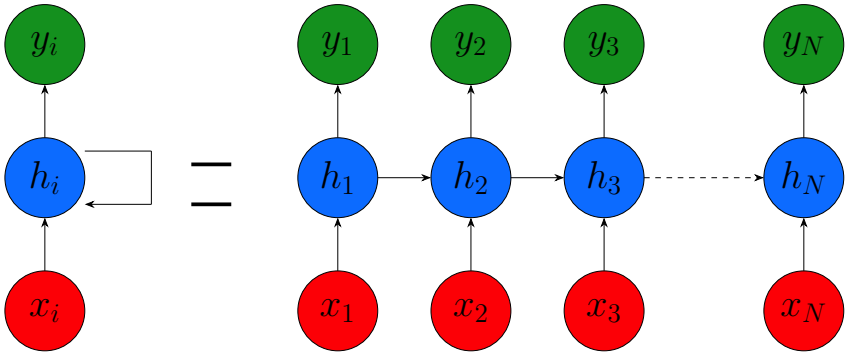


Figure 4.1: Rolled RNN versus Unrolled RNN

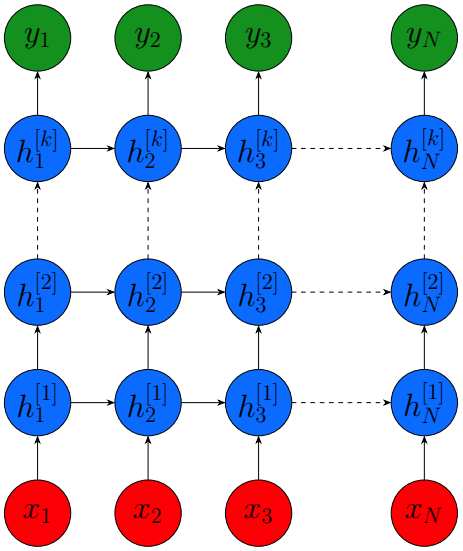


Figure 4.2: Multi-layer RNN

the sequence from x_1 is passed through to the layer $h_1^{[1]}$. The output of $h_1^{[1]}$ propagates through to the second row $h_1^{[2]}$ and the process repeats until the final row with the layer $h_1^{[k]}$ and an output y_1 is given. The second element is passed through to the second column of layer $h_2^{[1]}$ as well as an output from the previous column $h_1^{[1]}$. The output of $h_2^{[1]}$ is fed through to the second row layer $h_2^{[2]}$ and the process repeats until a second output y_2 is obtained. The process repeats for all the columns of the grid layer until a final output y_N. Here information is passed in a diagonal fashion. Essentially, the layer $h_i^{[j]}$ will require an output from both $h_{i-1}^{[j]}$ and $h_i^{[j-1]}$.

The structure of the RNN will vary depending on the task at hand. As such, the number of input or outputs can vary in the RNN. It can be one of 4 types:

- One-to-One: where the RNN has 1 input and 1 output. This is a simple Neural Network.

- One-to-Many: where the RNN has 1 input and many outputs. This is particularly useful in image captioning where given an image, a caption composed of multiple words is generated.
- Many-to-One: where the RNN has many inputs and 1 output. This structure is used in sentence classification, where a single class is predicted after having processed a sentence of multiple words.
- Many-to-Many: where the RNN has many inputs and many outputs. This structure is mostly used in text translation where a word is associated with another word from another language, and the context matters.

Another important feature of RNNs is backpropagation through time. At each time step, a backpropagation algorithm will traverse backwards from the output, collecting the derivatives of the error with respect to the gradients of the model. The parameters are then optimised using gradient descent.

Using RNNs for sequential tasks has the advantage of allowing to process inputs of any length. Additionally, the size of the model does not increase with the input size. Unfortunately, since there are computations in the hidden layers for each individual element of an input sequence, this usually results in slow executions. With this in mind, we can now explore how RNNs can be architected to run on quantum devices.

4.2 Quantum Recurrent Neural Networks

Quantum Recurrent Neural Networks [54] used in this research are based on the VQA framework described in chapter 2, with its quantum-classical hybrid approach. Though a QRNN uses both a quantum and classical device, only the quantum device part changes in order to transform an NN into a QRNN. The aim here is to include the "recurrent" property on quantum devices. This is achieved by increasing the number of PQCs in the quantum device.

Before running QRNN algorithms on quantum devices, the structure of PQCs to be used in the QRNN has to be predetermined. This will define the number of rotational angles that will be used as inputs for each PQC.

Given an input sequence with N elements, a QRNN functions by attaching a separate PQC for each element x_i of that sequence as shown in figure 4.3. Like a normal QNN, a circuit with n qubits is initialised with $|0\rangle$. Then the first element x_1 of the sequence is converted into a list of rotational angles $\overrightarrow{\theta}_{x_1}$. A first PQC is added to the initialised qubits and given $\overrightarrow{\theta}_{x_1}$ as parameters, changing the state of the qubits. Then the second element x_2

is also converted into a list of angles $\overrightarrow{\theta}_{x_2}$ and given as parameters to another PQC, which is also applied to the circuit. The process repeats until N PQCs have been applied sequentially to the circuit. A final measurement of the circuit can be taken. The measurements are then processed by the classical device, which optimises the assignment of angles.

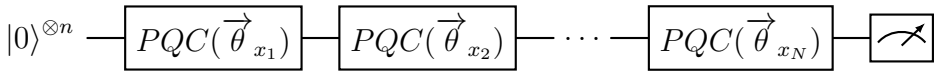


Figure 4.3: QRNN circuit

The recurrence relation is apparent since the final measurement is dependent on the application of each individual PQC, which will vary the quantum states. Classically, a hidden state acts as memory, which is carried and read throughout the network. With QRNNs, the quantum state serves as the network's memory, carrying information forward from one time step to the next.

Up to this point, only single-layered QRNNs were described. But like classically, the quantum network can be adapted to include the notion of layers to become deep QRNNs. This is done by creating k PQCs for one element in the sequence, and attaching them sequentially. Each group of k PQCs for each element is also attached sequentially as shown in figure 4.4. The quantum circuit shown in the figure represents one long circuit (and not rows of circuits), but this representation aids in picturing the repetition of PQCs for each sequence element. The end of the first row is labelled $\rightarrow_1$ and is connected to the start of the second row, also labelled $\rightarrow_1$. This is repeated $N-1$ times for all N elements in the sequence.

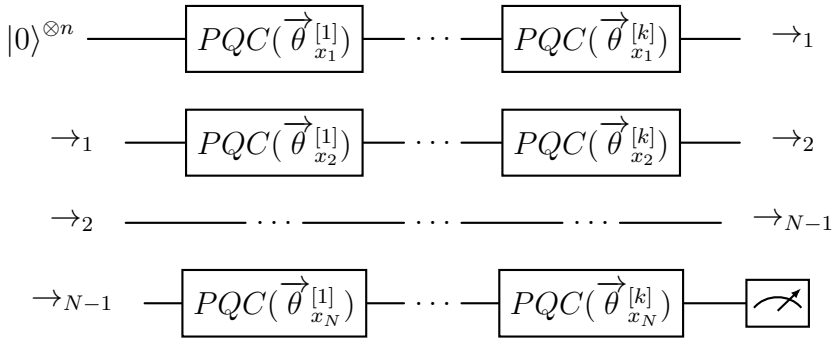


Figure 4.4: Layered QRNN

It is important to note that a new set of angles has to be generated for each layer for the same element in the sequence. This means that for layer i and element x_j of the sequence, the set $\overrightarrow{\theta}^{[i]}_{x_j}$ is generated for the PQC. But the next layer $i+1$ will generate a new $\overrightarrow{\theta}^{[i+1]}_{x_j}$ instead of just repeating $\overrightarrow{\theta}^{[i]}_{x_j}$.

With layered QRNNs, it is also possible to achieve the different One-to-One, One-to-Many, Many-to-One, and Many-to-Many structures by varying the number of angles generated for a single element and also when the measurements are taken. The circuit given above is a Many-to-One structure. If many outputs are desired, then measurements after each PQCs in the last layer can be taken. If only one input is required but many outputs are desired, then one element of a sequence should generate enough angles to initialise multiple PQCs.

Having explored how QRNNs are constructed, the experiments can now begin.

5 Solving Tasks using QML

Contents

This chapter describes the four tasks chosen to test QRNNs and how the QRNNs were constructed to solve these tasks. It also presents each dataset and how the data is processed. Multiple QNLP tasks were chosen to test QRNNs in different situations. While all tasks rely on QRNNs, different arrangements or structures of QRNNs are needed to fit the specific situation.

5.1 Sentiment Analysis (SA)

Sentiment Analysis is a subset of Text Classification. This task predicts the sentiment of a sentence or corpus of text. For example, if a QML model was faced with the sentence ”The man loves cats”, the model would have to predict a positive sentiment. On the other hand, with a sentence such as “The man hates dogs” a negative sentiment should be expected. The

QML models not only need to learn the sentiment of particular words, but also the sentiment of particular sentence structures to deal with sentences, for example, involving double negatives. This is why (as with most, if not all) a large dataset is needed to have the most variety. To test the performance, accuracy is measured by tracking the number of correctly predicted sentences against the total number of sentences. Since this task is a binary classification, a test accuracy above 50% suggests that the QML model is actually learning.

Dataset

To test how QRNNs would perform against different sentence structures or varying syntax, three different datasets for Sentiment Analysis were used. The three datasets are as follows:

- **Rotten Tomatoes Dataset** [1] contains the movie reviews from the Rotten Tomatoes website. It was pre-divided into positive sentences which were given the label 1 and negative sentences with the label 0. The dataset is composed of 5,326 positive and 5,326 negative sentences. The sentences are on average of length 21 words, with minimum 1 and maximum 59 words. This dataset is also interesting because it has a mix of formal and informal reasonably short sentences, providing a lot of variety.
- **IMDB Dataset** [2] also contains movie reviews but this time from the IMDB website. The labels are the same as Rotten Tomatoes. The dataset is composed of 24,999 positive and 24,998 negative sentences. The sentences are on average of length 231 words, with minimum 4 and maximum 2,470 words. This dataset is cleaner than Rotten Tomatoes comprising of more reasonably formal sentences and clear sentence structures. The sentences are also much longer.
- **Twitter Dataset** [3] contains tweets from Twitter. The labels are the same as the previous datasets. The dataset is composed of 8,581 positive and 7,780 negative sentences. This dataset is used to push the limits of the QRNN as it is composed of short sentences with a particularly informal tone. The dataset was originally composed of a third neutral sentiment, but we manually removed it to focus on the two main sentiments.

[1] Rotten Tomatoes Dataset
[2] IMDB Dataset
[3] Twitter Dataset

Processing the Data

In order for the dataset to be fed into the model, it needs to be cleaned so that only essential elements are kept. This involves converting all the characters to lowercase, removing all forms of punctuation, removing URLs, and removing any words that are less than 2 characters.

Once cleaned, it can be processed into computationally digestible inputs. First, a dictionary of individual words, along with unique indices, has to be created. Memory-wise, it is not advisable to directly convert all the dataset sentences into one-hot-vectors. This is because converting each sentence into lists of one-hot-vectors with large datasets, like IMDB, can be very costly memory-wise since each vector is the size of the index dictionary, and there is one vector per word of the sentence. Instead, it is more efficient to convert each sentence into a list with the indices of each word of the sentence. Then, each sentence can be converted into one-hot-vectors sequentially instead of loading everything in memory. As observed previously, there can be large sentence size differences between sentences in a dataset. To counter that and ensure that sentences of varying lengths can be efficiently processed uniformly, *padding* can be introduced. It is a process that ensures that sequences of varying lengths will be efficiently handled by the model while preserving information ordering within such sequences. Given a big enough padding size, if a sentence has more words than the padding, then it is cut off at that point. However, if a sentence is smaller, then arbitrary tokens (usually of value 0) are added until its size matches the padding. The added benefit of padding is that for datasets with particularly large sentences, like IMDB, it considerably reduces the computational overhead. The padding size for the datasets are the following:

- Rotten Tomatoes: 40
- IMDB: 100
- Twitter: 33

After the padding, the tokenised sentences are divided into training and testing dataset with a 0.9:0.1 ratio.

5.1.1 Solving Sentiment Analysis with a QRNN model

Constructing a QRNN to deal with Sentiment Analysis requires a Many-to-One structure since a sentence is constructed of a multitude of words, but in the end, only one sentiment is predicted.

Let us construct a QRNN with n qubits and k layers. The main question is obtaining angles from the one-hot-vectors that represent words in a sentence. Let us suppose that the size of the dictionary of individual words (i.e. the size of a one-hot-vector) is D, and a chosen PQC needs A number of angles. The angles can be computed by defining a linear map $L_1 : \mathbb{R}^D \to \mathbb{R}^{Ak}$ with random parameters at first.

When passing an input sentence, represented by a list of one-hot-vectors, each word (one-hot-vectors) x_i is converted into $A \times k$ angles which are all the angles necessary to be used by $PQC_{x_i}^{[1]}, \ldots, PQC_{x_i}^{[k]}$. Then all the PQCs are assembled for $i \in [|1, N|]$ in a sequence of N elements.

A measurement of the n qubits is taken at the end of the circuit and is applied to another linear map $L_2 : \mathbb{R}^n \to \mathbb{R}^2$ with random parameters. The resulting 2-element vector after the application of L_2 represents the probability of the sentence having the sentiment labelled 0 and the sentiment labelled 1. This probability distribution is passed through a cross-entropy function to compute the loss. The loss is then passed through an optimiser which will tune the parameters of the QRNN model, i.e. the parameters of L_1 and L_2.

The whole process, as shown in figure 5.1, repeats for all the elements in the dataset.

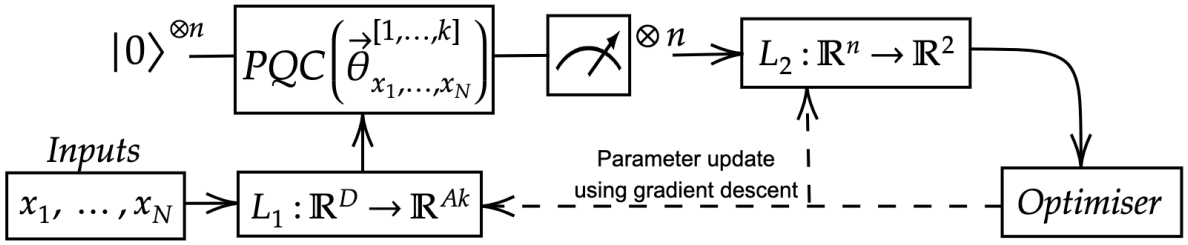


Figure 5.1: QRNN for Sentiment Analysis

The QRNN's performance is obtained by recording the accuracy with the test dataset. The predicted class is given by finding the index of the maximum value from the output of L_2. The predicted value is then compared with the expected value.

5.2 Document Similarity (DS)

Document Similarity is also a subset of Text Classification, but it goes a step further than Sentiment Analysis by comparing two texts together and judging if they are of equal meaning. It is not sufficient for two sentences to be about the same topic, they need to express the same meaning. For example, a QML model should be able to predict that the two following sentences are similar: "How I can speak English fluently?" and

"How can I learn to speak English fluently?". However, the two sentences "Is chocolate milk good for you?" and "Why is unrefrigerated milk good for you?" are not similar and do not convey the same meaning. The example sentences were taken from the dataset described next. Since this task is also a binary classification, a test accuracy above 50% suggests that the QML model is learning.

Dataset

The dataset used for this task is the **Quora Question Pair Dataset** [4] composed of question pairs from the Quora website. The label is given 0 if the pair is not similar and 1 otherwise. The dataset is composed of 31,347 non-similar sentences and 18,653 similar sentences. The sentences are on average of length 11 words, with minimum 2 and maximum 237 words. The original dataset is composed of 100,000 question pairs, but due to hardware constraints, the dataset was cut to the first 50,000 question pairs. The sentences in the dataset are composed of both formal and informal sentences. To test the performance, accuracy is measured by tracking the number of correctly predicted similar sentences against the total number of pairs.

Processing the Data

The dataset is processed in the same manner as the Sentiment Analysis task for all the pairs in the dataset, by creating a dictionary of individual words and indices. Here, the pairs of sentences are padded to 15 words. The dataset is then divided into training and testing datasets similarly to the Sentiment Analysis task.

5.2.1 Solving document similarity with a QRNN model

Building a QRNN with n qubits and k layers to predict the similarity between two documents is comparable to sentiment analysis, except this time, two sentences, $(x_1^1, \ldots, x_N^1)$ and $(x_1^2, \ldots, x_M^2)$, have to be handled. Two different circuits will be initialised for each sentence. Then both sentences are respectively converted into two lists of angles using a linear map $L_1 : \mathbb{R}^D \to \mathbb{R}^{Ak}$, with D the size of the dictionary and A the number of angles per PQC. The first list of angles will be used as inputs for the PQCs applied to the first circuit, and similarly for the second list of angles and the second circuit.

[4] Quora Question Pair Dataset

The two circuits are measured and the cosine similarity is taken between the two measurements. A second linear map $L_2 : \mathbb{R} \to \mathbb{R}^2$ will convert the cosine similarity into a vector of size 2 with the probabilities that the sentences are predicted to be similar or not.

Like previously, loss is measured and used to adapt the parameters of L_1 and L_2, by the optimiser.

The diagram process of the QRNN for document similarity is shown in figure 5.2.

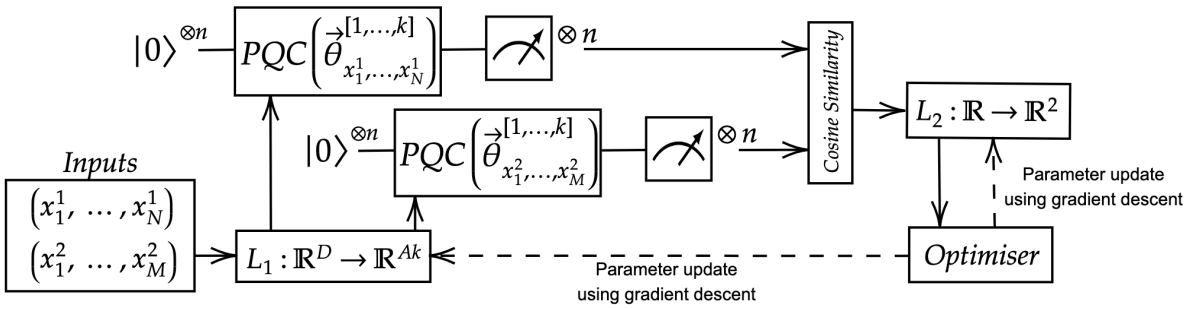


Figure 5.2: QRNN for Document Similarity

This QRNN's performance is also measured with the accuracy of the test dataset in the exact same manner as with QRNNs for Sentiment Analysis.

5.3 Text Generation (TG)

A Text Generation model, otherwise known as a language model, aims to generate new text given a corpus of texts. In language modelling, the aim is not to predict a new word or character but rather to predict a probability distribution over a vocabulary of possible words or characters. Text generation does not belong to classification tasks like the previous models discussed because multiple words or characters can be a valid continuation for an input sequence, with each having different probabilities. This is why simple accuracy is not enough to measure performance, as there is no single correct answer. It is necessary to take into account the varying probabilities. Instead, the performance of a model can be measured using *perplexity*, which assesses linguistic competence, with a lower score indicating superior generating capabilities, or more confident predictions. To obtain the perplexity score, it is required to obtain the total loss from the test dataset by summing the cross-entropy between the predicted values and target values of the test dataset. Then the perplexity is obtained by taking the exponential of the average loss. As opposed to accuracy, perplexity measures the uncertainty of the model's predictions

Dataset

The dataset used for training is the **TinyShakespeare** [5] which contains 40,000 lines from a variety of Shakespeare's plays. The dataset contains not only the texts but also keeps the structure of the play, with its dialogue and its speakers, as such:

CORIOLANUS:
I do owe them still
My life and services.

MENENIUS:
It then remains
That you do speak to the people.

Processing the Data

Data processing for Text Generation can happen in two ways, either at the word level or at the character level, which respectively generates text word by word or character by character. In this research, the latter was chosen. While they both have their advantages, character-level text generation does not have vocabulary limitations, can capture fine-grained linguistic structures such as spelling or typos, and has simplified tokenisation since the total amount of characters will be much lower than the total number of individual words in the corpus.

Tokenisation for this task is similar to Sentiment Analysis, with a dictionary of characters-to-indices and a conversion of indices to characters. Additionally, a window-size has to be defined in order to create the model input data with its corresponding target value. The text is parsed, and for every character, a set containing the next window-size amount of characters is created, and the character at window-size + 1 is selected as the target. For example, with the sentence: "Generation" and a window-size of 4, the training set will be:

Data	Target
[G,e,n,e]	r
[e,n,e,r]	a
[n,e,r,a]	t
...	...

until all the characters are parsed. For clarity in this example, the characters are selected instead of the indices.

The testing dataset is chosen to contain the last 1,024 data points, with the training dataset as the rest.

[5] TinyShakespeare Dataset

5.3.1 Solving text generation with a QRNN model

Building a QRNN with n qubits and k layers is also similar to how a QRNN is built for Sentiment Analysis, the difference is with the second linear layer L_2. Instead of converting the measurements from the circuits into a vector of size 2, the linear map should be $L_2 : \mathbb{R}^n \to \mathbb{R}^D$ where D is the size of the character dictionary. L_2 is built as such because for text generation, all the characters from the dictionary should possibly be predicted.

Again, the loss is measured and used to adapt the parameters of L_1 and L_2, by the optimiser.

The diagram process of the QRNN for Text Generation is shown in figure 5.3.

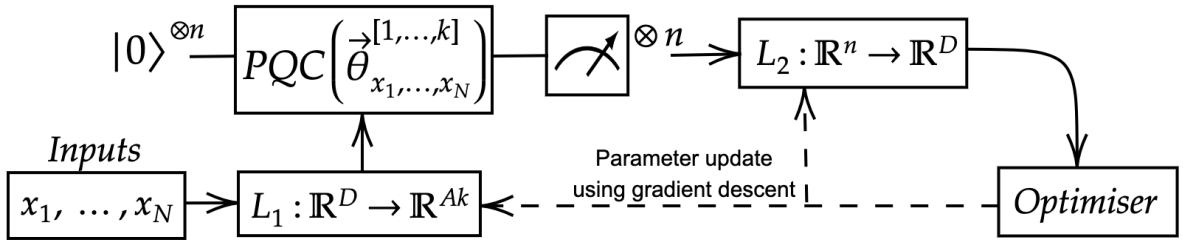


Figure 5.3: QRNN for Text Generation

This QRNN also uses the test dataset to evaluate the model's performance. And, as mentioned previously, perplexity is used as the performance metric.

5.4 Image Classification (IC)

The aim for this task is, given one image, to predict what digit is represented on an image. For example, in figure 5.5, a Machine Learning model would have to predict the digit 5. Otherwise, it would be incorrect. To test the performance, accuracy is measured by tracking the number of correctly predicted images against the total number of images. Since this task is a multi-class Classification task with 10 classes, a test accuracy above 10% suggests that the QML model is learning.

Dataset

The dataset used for this task is the standard **MNIST** dataset [6] (figure 5.4), comprised of an extensive collection of around 70,000 handwritten black and white digits ranging from 0 to 9. Specifically, each image comprises 28×28 pixels, with each cell containing a value in $[0, 1]$ (as shown in figure 5.5).

[6] MNIST dataset

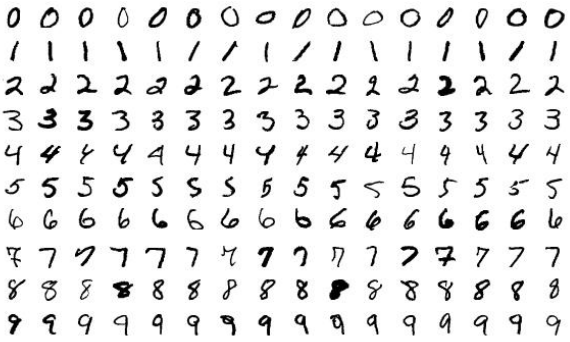


Figure 5.4: MNIST dataset
Source: `https://en.wikipedia.org/wiki/File:MnistExamplesModified.png`

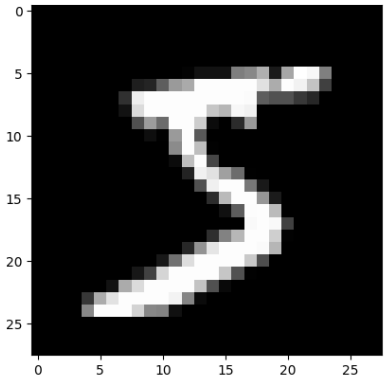


Figure 5.5: Example of MNIST datapoint

Processing the Data

The MNIST dataset is already encoded into an array of numerical values representing a number. Notwithstanding this, in order to considerably simplify the training and model processing time, each matrix can be transformed by joining each row of the matrix next to each other as such:

$$\text{Before } \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix} \text{ After}$$

Therefore, each 28×28 image is transformed into a vector of size $784 = 28 \times 28$.

The dataset is then divided into training and testing datasets, similar to the Sentiment Analysis task.

5.4.1 Solving Image Classification with a QNN model

Image Classification does not need a QRNN model to be solved, but rather a simple QNN based on the QRNN structure presented for the previous tasks. QRNNs are not needed since the image is processed fully, rather than line by line.

For a QNN with n qubits and k layers, constructed with PQCs each requiring A angles, a linear map $L_1 : \mathbb{R}^{784} \to \mathbb{R}^{Ak}$ is defined. The linear map L_1 will then convert a whole image into angles. A circuit is initialised, and PQCs with angles from L_1 are attached sequentially. The qubits are measured, and the measurements are processed by a second linear map $L_2 : \mathbb{R}^n \to \mathbb{R}^{10}$. The output of the second linear map is $\mathbb{R}^{10}$ since there are 10 digits from $0, \ldots, 9$, which the image could represent.

Again, loss is measured and used to adapt the parameters of L_1 and L_2, by the optimiser.

The diagram process of the QRNN for Image Classification is shown in figure 5.6.

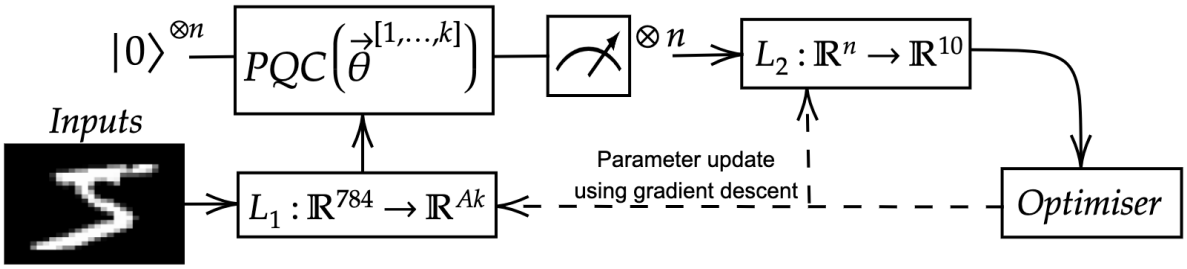


Figure 5.6: QRNN for Image Classification

This QNN's performance is obtained by measuring the accuracy of the test dataset similarly to Sentiment Analysis.

6
Setting up the Experimentation

Contents

As presented in the introduction, in chapter 1, this research aims to compare the performance of QRNNs built with either fully quantum or free fermionic circuits, as well as explore the possibilities of free fermionic circuits in the context of the four QML tasks that we introduced in chapter 5. This chapter will present how the experimentation was carried out for the purpose of this research. First, the structure of the PQCs used will be laid out, and then the experiment will be parametrised.

A fully classical machine learning baseline of the tasks has also been included using classical RNNs. This is not to make a direct numerical comparison between the two quantum methods but rather to observe how far QML is compared to already state-of-the-art classical ML methods. The experiment is then divided into four parts, one for each of the QML tasks, with each part subdivided into three different models: fully classical, fully quantum, and free fermions.

Without access to a quantum computer, it is important to note that all the experiments were performed classically. Therefore, all of the fully quantum models were also simulated

classically. However, the classical simulation is different from free fermionic classical simulation. This fully quantum simulation, powered by the *TorchQuantum*[55][1] library aims to simulate the exact functioning of quantum computers and hence is as though we had used a theoretical noise-free quantum computer. Since the simulated qubits are not affected by noise and do not decohere, we refer to logical qubits when discussing the number of qubits in a circuit. There is no need to pad logical qubits with physical qubits since we do not need to construct surface codes to protect information. On the other hand, the free fermionic simulation is set up in the exact same manner it would have been if the use of free fermionic circuits had been required in a practical setting; therefore, it does not need a quantum computer to scale. This means that only accuracy can be taken into account, and not processing or execution speed, as classical resources limit the experiments. Even though this research stays within the realm of simulations and we do not experiments on actual quantum computers, accuracy is still relevant as the simulations properly depict how these devices function but not how fast. Furthermore, training fully quantum models would have been difficult to perform on an actual quantum computer as backpropagation on a quantum computer is currently hard.

In this chapter, when mentioning expressiveness, we refer to the size of the vector space that a given model can cover.

6.1 Free Fermions approach - Choosing a Matchgate

First, let us define the free fermionic PQCs we will employ in our models. The fully quantum equivalent will then be constructed, keeping in mind a similar structure to ensure a proper comparison.

In order to simulate free fermionic systems, as seen in chapter 3, it is required to pick a set of gates which, put together carefully, will form a 2-qubit gate which should satisfy the matchgate property. Here, we explicitly illustrate that the ansatz we pick does satisfy this property. In this research, we concentrate on the following gates:

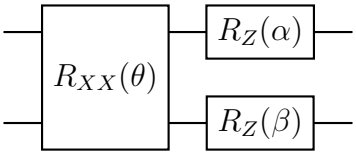
$$R_{XX}(\theta) = \exp\left(-i\frac{\theta}{2}X \otimes X\right) = \begin{bmatrix} \cos(\frac{\theta}{2}) & 0 & 0 & -i\sin(\frac{\theta}{2}) \\ 0 & \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) & 0 \\ 0 & -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) & 0 \\ -i\sin(\frac{\theta}{2}) & 0 & 0 & \cos(\frac{\theta}{2}) \end{bmatrix}$$

and

$$R_Z(\alpha) = \exp\left(-i\frac{\alpha}{2}Z\right) = \begin{bmatrix} e^{-i\frac{\alpha}{2}} & 0 \\ 0 & e^{i\frac{\alpha}{2}} \end{bmatrix}$$

[1] `https://github.com/mit-han-lab/torchquantum`

Forming a matchgate using these two gates requires one R_{XX} and two R_Z gates arranged in the following manner:



We now need to ensure that the proposed matchgate fits the matchgate property:

Proof. In order to prove that the proposed matchgate, composed of $R_{XX}(\theta)$ followed by two R_Z gates, $R_Z(\alpha)$ and $R_Z(\beta)$, does indeed form a matchgate, it is required that we find the 4×4 matrix form of the 2-qubit gate, and make sure that it has the same form as 4×4 matrix matchgates. Then, with the two generating A and B matrices of the proposed matchgate, we need to verify that $\det(A) = \det(B)$ and $A, B \in U(2)$.

The detailed computational proof is in appendix A. This is also implied by the properties of the Jordan-Wigner transformation in chapter 3.

With this, the proposed set of gates as arranged does indeed form a matchgate and can therefore be used for classical simulations. □

With an arrangement of gates that satisfy the matchgate property, the PQC for a single layer can be created. In this research, the free fermionic PQC is constructed by layering and overlapping the matchgate structure across all qubits, as shown in figure 6.1 for 4 qubits. For n qubits, a total of $n - 1$ matchgates will be applied to the circuit. Since the PQC is only composed of matchgates, the overall circuit becomes classically simulable.

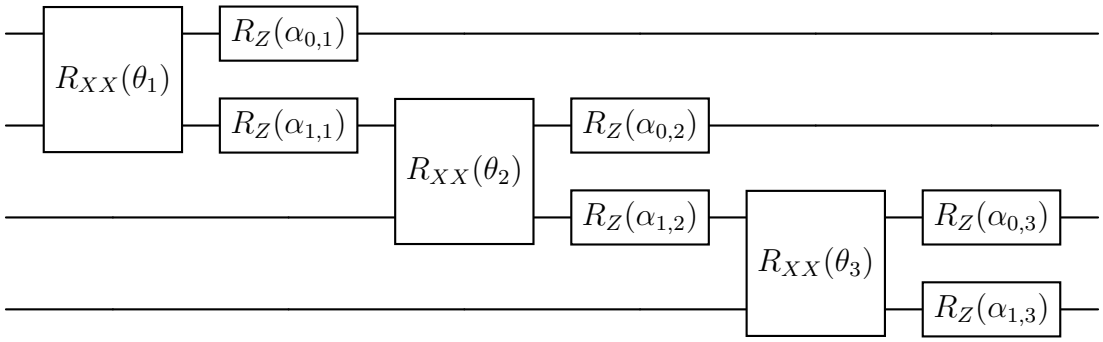


Figure 6.1: Free Fermionic PQC for 4 qubits

The figure in 6.1 represents one singular PQC with 9 individual angles. With multiple layers, that same similar PQC structure will be repeated k times for k layers, each containing 9 individual variable angles.

This means that for a circuit with n qubits and k layer, the number of angles that need to be generated is:

$$\#\text{ angles} = k \times 3(n-1)$$

However, the PQC presented previously needs to generate a lot of angles and its implementation is inefficient, due to the fact that some R_Z gates need to be applied in between two R_{XX} gates. As such, the PQC can be simplified by concentrating on a specific subset of the previous PQC. The new simplified PQC sets all the angles, $\alpha_{1,1}, \ldots, \alpha_{1,n-1}$ for a n qubit PQC, to 0, which all convert the rotational gates $R_Z(\alpha_{1,1}), \ldots, R_Z(\alpha_{1,n-1})$ to the identity wire. Since the new PQC is a subset of the PQC composed of $n-1$ matchgates, then it is also classically simulable. The old PQC implementation of figure 6.1 becomes the new simplified PQC implementation in figure 6.2.

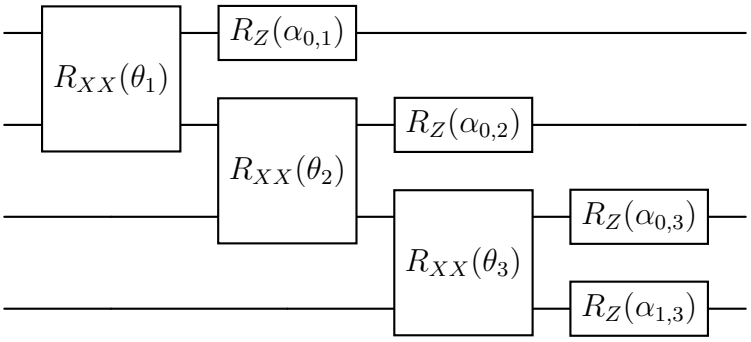


Figure 6.2: Reduced Free Fermionic PQC for 4 qubits

This new implementation is equivalent to the free fermionic one described in section 3.3.

This means that for a circuit with n qubits and k layer, the number of angles that need to be generated for the new PQC is:

$$\#\text{ angles} = k \times (2 \times n - 1) \tag{6.1}$$

The new fermionic PQC is less expressive as it contains fewer parameters which directly affects the circuit's ability to explore and represent different quantum states. However, expressiveness can be recovered by adding more layers, improving the precision at which quantum states are reached.

6.2 Fully Quantum approach - Choosing a PQC

The fully quantum circuits are constructed with a structure similar to free fermionic PQCs to properly compare their performance by ordering them to be as close as possible to each other. As such, the equivalent to the matchgate for fully quantum circuits uses the same R_{XX} followed by a new gate

$$U_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda}\sin(\frac{\theta}{2}) \\ e^{i\lambda}\sin(\frac{\theta}{2}) & e^{i(\phi+\lambda)}\cos(\frac{\theta}{2}) \end{bmatrix}$$

The quantum gate U_3 is a versatile and powerful single-qubit gate, as it can be used to represent any single-qubit operation. By itself, it nearly creates a *universal gate set*, capable of representing any multi-qubit operation. The only missing gate would be the $CNOT$ gate which is needed to entangle qubit, the only missing requirement to satisfy a universal gate set. The gate U_3 is parametrised by three angles θ, ϕ and λ which determine a specific rotation on a qubit. The angle θ represents a rotation around the X-axis, the angle ϕ represents a rotation around the Z-axis, and the angle λ represents a rotation around the Y-axis. In fact, we can retrieve known gates by setting specific values to the three parameters. For example, the Hadamard H gate can be achieved by assigning the following values to the parameters:

$$\theta = \frac{\pi}{2}, \quad \phi = 0, \quad \lambda = \pi$$

With the two gates R_{XX} and U_3, we construct a PQC for fully quantum circuits based on the previously proposed matchgate. Here we replace the R_Z gates by U_3, as shown in figure 6.3.

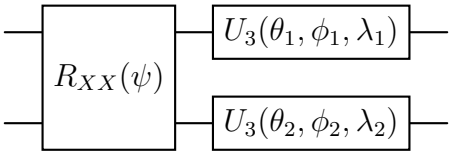


Figure 6.3: Fully Quantum matchgate equivalent

For a circuit with $n > 2$ qubits, the full PQC is assembled similarly to the free fermionic PQC by again, layering and overlapping the two-qubit gate of figure 6.3 along all the qubits as shown in figure 6.4 for 4 qubits and 1 layer. For clarity, we denote the three parameters of U_3 in position i, j of the PQC as $\overrightarrow{\Gamma}_{i,j} = (\theta_{i,j}, \phi_{i,j}, \lambda_{i,j})$. For k layers, the circuit is extended by staking k PQCs in a sequence.

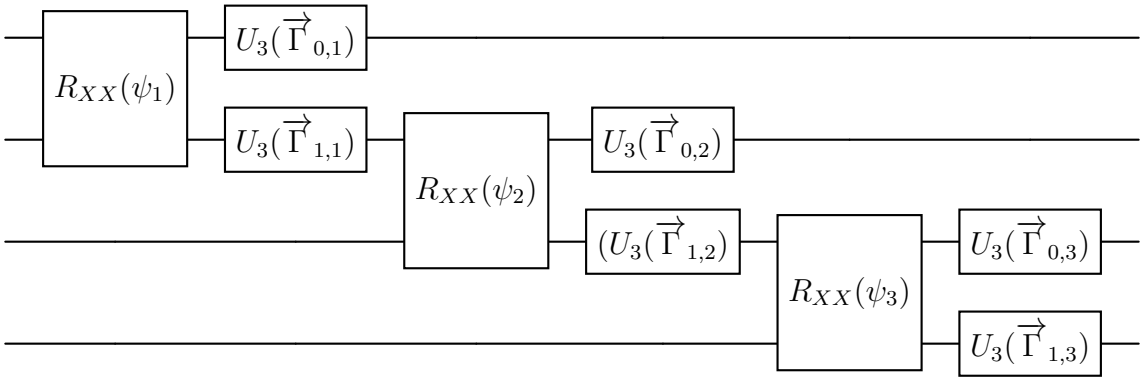


Figure 6.4: Fully Quantum PQC for 4 qubits

Again, for simplicity of implementation, the PQC can be reduced by setting all of the angles $\overrightarrow{\Gamma}_{1,i} = (0, 0, 0)$ for $i \in [|1, n-1|]$, reducing all of the U_3 gates surrounded by two R_{XX} to the identity wire, as shown in figure 6.5.

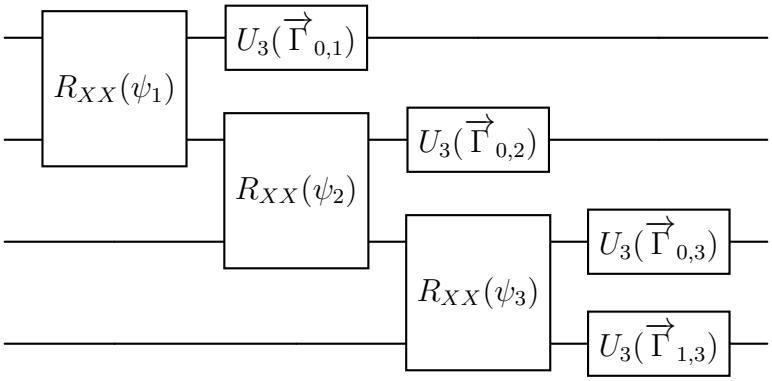


Figure 6.5: Reduced Fully Quantum PQC for 4 qubits

For a fully quantum circuit with n qubits and k layer, the number of angles that need to be generated is:

$$\#\text{ angles} = k \times (4 \times n - 1) \tag{6.2}$$

Just like free fermionic PQCs, the expressivity of fully quantum PQC is reduced with fewer parameters and can be augmented by increasing the number of layers.

6.3 Parametrising our Experiments

As stated previously, this research aims to compare the accuracy of QRNNs built with classically simulable free fermionic circuits with non-classically simulable fully quantum

circuits. In order to achieve this comparison, the PQCs of figure 6.2 and 6.5 will be used. Additionally, as stated previously, classical RNNs will be used in place of the PQCs to capture a classical baseline. The experimentation then consists of solving the four tasks of chapter 5. The two PQCs and the RNN are applied for each task. Multiple instances of the experiments are carried out, each by varying the numbers of qubits and layers in the PQCs, as well as the hidden size for RNNs.

However, as seen in chapter 3, the states of free fermions are reduced from $2^n \times 2^n$ to $2n \times 2n$, with n the number of qubits, due to the $2N \times 2N$ matrix from Majorana fermions that operates on a $2N$-dimensional vector space. This means that the expressiveness of free fermionic circuits is also reduced given a similar number of qubits. In order to achieve a similar level of expressiveness, additional qubits can be added to the free fermionic case. To achieve this correspondence, assuming the free fermionic system has N qubits and fully quantum systems have M qubits, it is necessary to have that $2N = 2^M$. Therefore, a free fermionic system needs $N = 2^{M-1}$ total qubits to match the fully quantum circuit's expressiveness.

As for classically built models, the number of qubits can be loosely correlated to the hidden layer size of the RNN. Directly correlating classical and quantum Machine Learning is difficult since information is processed very differently in both cases, but it is possible to imply an equivalence as a way to understand the relative capacity of either network. For a classical RNN, a hidden layer size of H means that the network maintains H dimensions in its hidden state vectors. Similarly, a quantum system composed of N qubits will represent a quantum state space of 2^N dimensions. Thus, for a QRNN of N qubits, a classical RNN would need a hidden layer size of 2^N nodes. Of course, increasing the number of qubits will exponentially increase the hidden layer size, drastically expanding the complexity of the classical system.

6.3.1 Hyperparameter selection

As established, this experiments consisted of training QRNNs with varying parameters to find the highest possible accuracy for each method of building the PQC depending on the task. Apart from the number of qubits and layers, the learning rate is also an important factor. Since we are dealing with a gradient descent algorithm to find the optimal set of angles for the PQCs, the learning will dictate if and how fast the optimal point is reached, by governing the pace at which an algorithm updates parameter values. A lower learning rate means a slower pace, and a higher rate a faster pace.

Through experimentation, it was observed that increasing the number of qubits did not require any change in the learning rate, even though the number of qubits has a direct correlation with the total number of angles to train. However, increasing the number of layers required decreasing the learning rate to keep the accuracy from decreasing substantially.

Furthermore, as some tasks require the processing of large amounts of data, in the interest of capturing results and saving computational resources, some QRNNs were constrained to only 1 layer in order, particularly with QRNNs built out of fully quantum gates. The following sub-sections will explore what hyperparameters were utilised to train the various models.

Fully Quantum QRNNs

Firstly, fully quantum QRNNs are parametrised so that free fermionic QRNNs can follow the same structure. As such, this part of the experiment aims to gauge the scalability and performance of the fully quantum QRNN by varying the number of qubits and layers. Specifically, the experiment iterates the configuration of the PQC from 3 to 10 qubits, which is a suitable range for this foundational observation. Starting with a lower bound of 3 qubits provides a minimum quantum state space, allowing the observation of the minimum capacity of such a system. The upper bound of 10 qubits strikes a good balance between computational resource requirements while allowing the QRNN to explore a deep quantum representation. Increasing the number of layers from 1 to 5 also allows the quantum network to improve in expressiveness while being computationally feasible.

The gradient descent algorithm's learning rate was found to be best at 0.005 for most tasks, except document similarity, which required a lower rate of 0.001 to perform best. The number of epochs was also chosen depending on the task and hyperparameters. This choice was made to preserve computational resources while balancing cases where models stopped learning after a particular epoch. Usually, models would run through 10 epochs, until resources became too demanding, which required the number of epochs to be dropped to 5.

The complete parametrisation for the experimentation containing fully quantum QRNNs can be found in figure 6.6.

Free Fermionic QRNN

The parametrisation of the free fermionic QRNNs flowed from the parametrisation of fully quantum QRNNs. As seen previously, a free fermionic system would need 2^{M-1} qubits to match the expressiveness of M fully quantum qubits. Thus, with the list of

the number of qubits parameters presented for the fully qubit system, the free fermionic equivalent raised the number of qubits respectively to the 2^{M-1} formula. Therefore, the free fermionic experiments required the parametrisation of the QRNNs from 4 qubits to 512 qubits, in power of 2 increments. Additionally, 3 qubits QRNNs are also tested to observe the effect on a minimum number of qubits. Unfortunately, computational resources halted the experiments on QRNNs possessing more than 64 qubits, corresponding to 7 fully quantum qubits.

Reflecting expressivity for free fermionic QRNNs is different in terms of the number of layers. For a free fermionic system of N qubits, it is possible to compress a matchgate circuit that has depth $> \frac{N}{2}$ into a circuit that has depth $\frac{N}{2}$ [56]. Additionally, in order to prepare every free fermionic Gaussian states on N qubits, it is necessary for a QRNN to have a layer depth of again $\frac{N}{2}$ [38]. This suggests that, in order to achieve maximum expressability for a QRNN made of matchgates, it is necessary for the circuit to have a layer depth of minimum $\frac{N}{2}$. This could in turn suggest that any QRNN model with the number of layer $< \frac{N}{2}$ is under-parametrised, i.e. the model is smaller than necessary to fit the data. A number of layer $> \frac{N}{2}$ would lead to over-parametrisation, i.e. the model is bigger than necessary to fit the data. This was included in the experimentation. Therefore, when possible given limited computational resources, for every number of qubits, the experiment included a case where the number of layers is greater than half the number of qubits present in the QRNN. A separate further study to study the effect of passing the $\frac{N}{2}$ threshold. This was done specifically with the image classification task as it required the least computational processing. For this separate study, the number of qubits increased from 8 to 16, with each iteration iterating through 1 to the number of qubits in question.

An additional experiment was also performed to observe the effect of increasing the number of qubits on maximum accuracy. As such, the fully quantum and free fermionic models were run with 1 layer and from 3 qubits to 12 qubits.

The learning rate stays mostly constant. However, with the Image Classification task, the learning rate needs to be decreased depending on the number of layers in order to stop the accuracy from dropping drastically. The total number of epochs to train the model is also mostly constant, apart from two sentiment analysis task that required the number of epoch to drop from 10 to 5 for a certain number of qubits and layers.

The complete parametrisation for the experiments containing free fermionic QRNNs can be found in figure 6.7.

Classical RNN

For the classical RNNs, this part of the experimentation translates directly from the conversion of qubits to hidden layer size presented previously. As such, the hidden layer size varies from 8 to 1024 in increments of powers of 2. The number of layers can be increased similarly to the fully quantum experimentation. Though in most cases, with RNN being already developed, most of the tasks will not need more than 1 layer to achieve maximum accuracy.

6.3.2 Effect of Batch Size

In chapter 5, batching was purposefully left out of the data processing sections for each task. The reason is that batching, as a hyperparameter, directly impacts both the computational efficiency of the training and the accuracy of a model, in a classical machine learning context [57]. Quantum Machine Learning is no different. Batching represents the number of samples that passes at once through a network. Large batches usually lead to faster training times but can come at the cost of lower accuracy and overfitting, while the converse also happens.

Batching can also affect the rate at which a model converges. Smaller batch sizes often lead to more frequent updates, thus making the model more sensitive to random variations or noise from the input data. Sensitivity can be optimal in some cases but also introduces instability. On the other hand, a bigger batch size can average out these fluctuations and produce a smoother convergence.

Given the impact of batch sizes, depending on which structure of QRNNs is used, it is important to maintain a consistent batch size across the same experiments between fully quantum, free fermioninc and classical approaches. Varying batch sizes can introduce differences in which data is interpreted, but it is paramount to have data interpreted similarly regardless of the structure. By keeping the batch size uniform, the impact of variables is minimised, guaranteeing a fair comparison between the structures since any changes would come from the model and not from training inconsistencies.

Batch size can differ from task to task since data should not be interpreted the same way, whether we are dealing with images or text. But it has to be similar structure to structure.

Therefore the batch size for each tasks are as such:

Task	Batch size
Sentiment analysis	256
Document similarity	200
Text generation	128
Image classification	128

Hyperparametrisation for the Experimentation

	Hyperparameters			
Task	Number of Qubits	Number of Layer	Learning Rate	Total Epochs
Sentiment Analysis (Rotten Tomatoes)	{3,4,5,6,7,8,9,10}	{1,2,3}	0.005	10 for layer = 1 5 otherwise
Sentiment Analysis (IMDB)	{3,4,5,6,7,8,9,10}	{1}	0.005	5
Sentiment Analysis (Twitter)	{3,4,5,6,7,8,9,10} {11,12}	{1,2,3} {1}	0.005	10 for layer = 1 5 otherwise
Document Similarity	{3,4,5,6,7,8,9,10}	{1}	0.001	10
Text Generation	{3,4,5,6}	{1,2,3}	0.001	10
Image Classification	{3,4,5,6,7,8,9,10} {11,12}	{1,2,3,4,5} {1}	0.005	10 for qubit <7 or layer <3 5 otherwise

Figure 6.6: Hyperparameters for Fully Quantum QRNNs

	Hyperparameters			
Task	Number of Qubits	Number of Layer	Learning Rate	Total Epochs
Sentiment Analysis (Rotten Tomatoes)	{3,4,8,16,32}	{1,2,5,10,25}	0.005	10 for qubit <32 or layer <5 5 otherwise
Sentiment Analysis (IMDB)	{3,4,8,16,32}	{1}	0.005	5
Sentiment Analysis (Twitter)	{3,4,8,16,32} {5,6,7,9,10,11,12}	{1,2,5,10,25} {1}	0.005	10 for qubit <32 or layer <5 5 otherwise
Document Similarity	{3,4,8,16,32,64}	{1}	0.001	10
Text Generation	{3,4,8,16}	{1,2,3,4,8}	0.001	10
Image Classification	{3,4,8,16,32,64} {5,6,7,9,10,11,12}	{1,2,5,10,25} {1}	0.005 for layer <6 0.0005 for 6 $\leq$ layer <10 0.0001 for layer $\geq$ 10	10

Figure 6.7: Hyperparameters for Free Fermionic QRNNs

7 Results of the Experimentation

Contents

Having run the experiments presented in chapter 6, we can analyse the results and observe how free fermionic circuits perform in a machine learning capacity and how they compare to fully quantum circuits. Note that all these results were obtained on a portable computer with the following specifications:

- CPU: 2,2 GHz 6-Core Intel Core i7
- GPU: Radeon Pro 555X 4 GB Intel UHD Graphics 630 1536 MB (important as the Python Machine Learning library **PyTorch** used the GPU for processing after being specified to do so)
- Memory: 16 GB 2400 MHz DDR4

7.1 Maximum Accuracy

First, let us examine the maximum accuracy obtained for the four tasks.

Starting with Sentiment Analysis, with the results shown in the bar plot figure 7.1: All three methods perform well, lying in the 74% to 81% range. The two quantum QRNNs match the accuracy of the classical RNN, even surpassing it by a small margin for the *Rotten*

Tomatoes and *IMDB* datasets. The accuracy of free fermionic QRNNs goes back and forth with fully quantum QRNNs, outperforming in both *Rotten Tomatoes* and *Twitter* datasets.

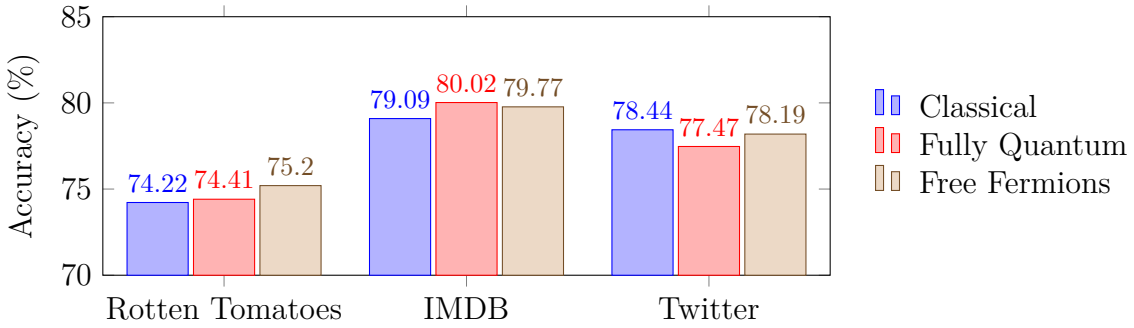


Figure 7.1: Sentiment Analysis Results

Moving to Document Similarity and Image Classification tasks, with the results in bar plot figure 7.2, we get a first glimpse of tasks where classical methods beat quantum methods by a more considerable margin. Document Similarity does not perform as well as the previous QNLP task, with a maximum accuracy of 71% classically and 68% with free fermions, compared to a minimum of 74% with Sentiment Analysis. Here, the free fermionic QRNN performed better than fully quantum, though marginally. The same situation occurs when dealing with Image Classification, except all three methods perform very well, with accuracies greater than 90%. Once again, the classical model performs best, followed by free fermionic ones, and finally the fully quantum QRNN model.

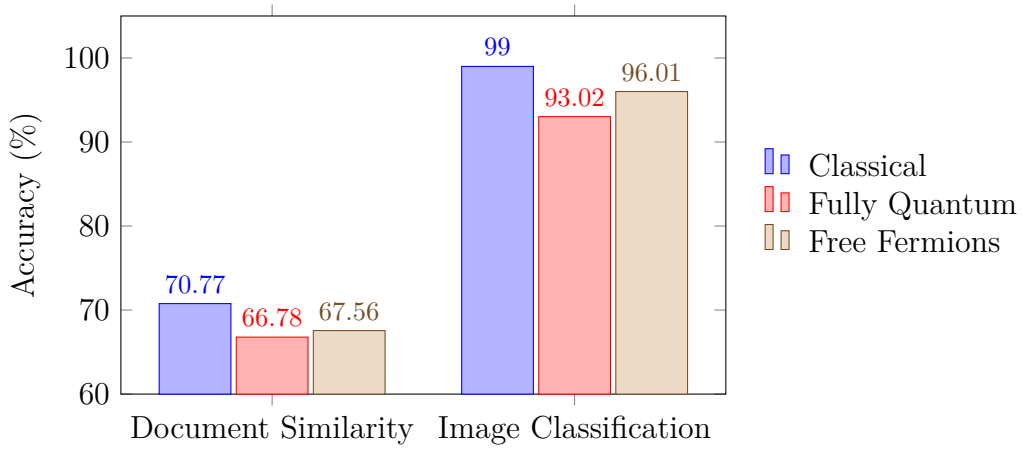


Figure 7.2: Document Similarity and Image Classification Results

The Text Generation task, with the results shown in bar plot figure 7.3, shows the highest gap between classical and quantum methods with a perplexity difference of around 10 between classical and free fermions. On the other hand, the maximum accuracy of

two QRNNs is closely related, with free fermionic circuits taking a slight perplexity lead. With Text Generation, apart from the perplexity, the performance of the models can be estimated by generating text with the same prompt for each model. The starting character prompt was chosen randomly in the text:

```
orter.

ANTONIO:
We two, my lord
```

Classically, the generated text, as seen in figure 7.4, resembles an English text. It contains proper English words, separates the characters properly, and even captures the Shakespearian structure of dialogue between characters in a play by including the next line character "\n" when needed and starting each dialogue with the name of the person speaking. Unfortunately, Text Generation using either fully quantum and free fermionic QRNNs, as shown in figure 7.5 and 7.6 respectively, did not lead to adequate results, with the two methods equally underperforming. However, though the results do not seem positive per se, they are promising. Indeed, even though the quantum-generated texts are incoherent with no English words properly generated, there is a resemblance of sentence structure with seemingly correct spacings between "words" and correct sentence lengths. The dialogue format of plays does also appear.

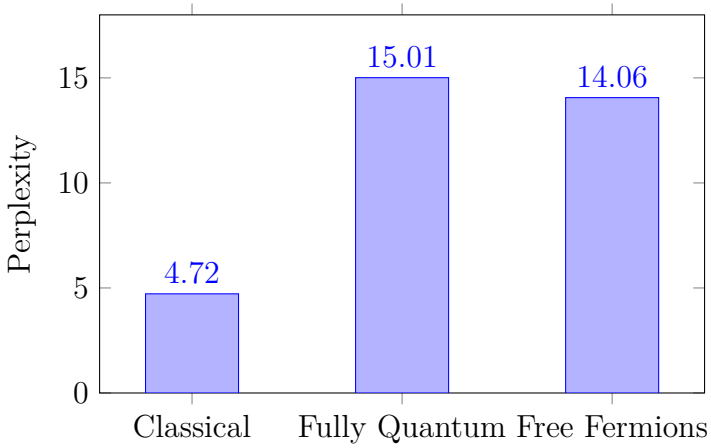


Figure 7.3: Text Generation Results

The hyperparameters that resulted in the maximum accuracy in each category and task are presented in figure 7.7.

When considering the hyperparameters, it is interesting to note that some QRNNs achieved the highest accuracy with the lowest possible number of qubits tested for some Sentiment Analysis tasks. However, the rest of the tasks required more complex QRNN

```
orter.

ANTONIO:
We two, my lords to hear's adoursed sure.
Upon groundskinmencius.

PMARGARET:
The right, tell you claterab part owl a wilt care. Yet come use themed.

CORIZARET:
That art is make to us.
What the lances, the bears, '
```

Figure 7.4: Classical Text Generation Prompt

```
orter.

ANTONIO:
We two, my lordri—otn bdnae o slo yte:
Whhel tbo tseid  ritrcsetek alnrel' ski
 ienth srii sth' bttbise ulste bitre
```

Figure 7.5: Fully Quantum Text Generation Prompt

```
orter.

ANTONIO:
We two, my lordirsn,'la the si gwe nthepsor ohrd morau biy
edr thoerH wiurel wy imm ato ve—elt, os theve.
KI O Bo'hn, otr.
Binse':

GAcAT LSI:?: ssuu rpeca thhi puso,: D
IACThAm hae, photr mesit tet wdr
Whee dhhral
```

Figure 7.6: Free Fermionic Text Generation Prompt

structures. Also, as seen in figure 7.8, which shows the maximum accuracy evolution for models with 1 layer for one sentiment analysis and image classification, increasing the number of qubits did not necessarily increase accuracy. Indeed, for both fully quantum and free fermionic models, the maximum accuracy for sentiment analysis peaked at 3 qubits and plateaued around the 77% mark as the number of qubits increased. Meanwhile, the maximum accuracy increased for the image classification task. It is important to remember that these results are for QRNNs with 1 layers. So even though free fermionic models have a lower maximum accuracy than fully quantum models in Image Classification, when increasing the number of layers and qubits, free fermionic models reach a higher maximum accuracy compared to fully quantum models.

Furthermore, even though adding more layers increases the QRNN's expressibility, as we can see, this does not necessarily translate into higher accuracy. For example, with

		Document Similarity					
Structure	**Param**	*Rotten*	*IMDB*	*Twitter*	**DS**	**TG**	**IC**
Classical	# layer	1	2	1	1	1	1
	Hidden size	1024	512	128	16	512	256
	Learning rate	0.001	0.001	0.005	0.001	0.001	0.001
Fully Quantum	# layer	2	1	2	1	3	2
	# qubits	4	3	10	4	6	10
	Learning rate	0.005	0.005	0.0005	0.0005	0.001	0.005
Free Fermions	# layer	10	1	1	1	8	14
	# qubits	16	3	3	64	16	15
	Learning rate	0.0005	0.005	0.005	0.0005	0.001	0.001

Figure 7.7: Hyperparameters for Maximum Accuracy Results
For the sake of simplicity, the names of the tasks in the headers were abbreviated as the following: Document Similarity (DS), Text Generation (TG) and Image Classification (IC).

Document Similarity, the free fermionic circuit with the highest accuracy contains 64 qubits but only one layer. This would indicate that only low expressibility is needed for these tasks. Indeed, adding more layers to the models (supposedly adding more expressibility) only resulted in overfitting, significantly decreasing accuracy. This effect was more prominent in the free fermionic image classification tasks with up to 70% less accuracy when adding more layers to the model with the number of qubits that gave the highest accuracy.

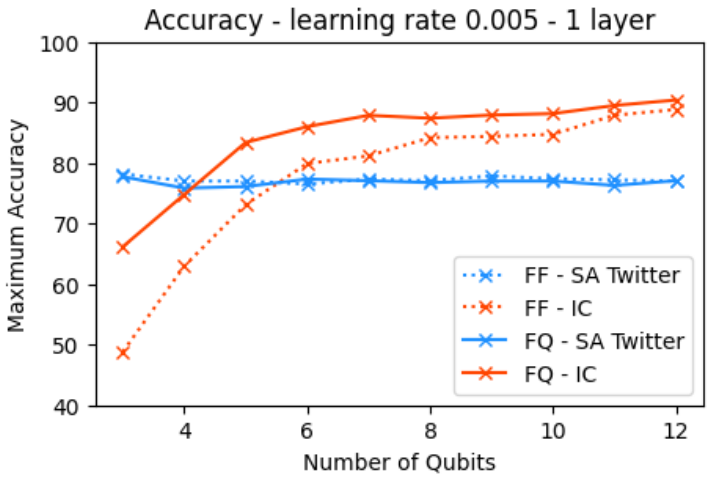


Figure 7.8: Accuracy Evolution of 1 Layer Models
For the sake of simplicity, Free Fermions and Fully Quantum has been abbreviated to FF and FQ respectively, with the tasks Sentiment Analysis and Image Classification also given SA and IC as abbreviations.

Overall, for most of the tasks (except one), free fermionic QRNNs, at their maximum,

performed better than their fully quantum equivalent. Yet, the difference is very marginal, with at most 3% increase. Nevertheless, this could mean that free fermionic circuits could be viable replacements for fully quantum circuits in terms of accuracy alone.

7.2 Time of learning

In this section, let us look at the time to train the quantum models for one epoch and observe how the time evolves as the number of qubits increases. As we can see in the hyperparameters table of figures 6.6 and 6.7, due to the high level of computational processing that some configurations required, some results, mainly free fermionic QRNNs with high number of qubits (mostly above 64 qubits), were not obtained. Still, the total learning time can be estimated without running a full experiment. Only one batch from the training dataset can be processed to compute the approximation. The approximate total time is the time to process one batch times the number of batches (length of training dataset / batch size) in the training dataset. Since each data point is padded [1] to the same length before the batching, each batch size has the same amount of data to process. Assuming that each batch is processed the same way, the approximation should give a good estimate of the total time to train for one epoch.

With this approach, we can fill in the absent results. The approximation was performed until all tasks (except Text Generation, which was too computationally demanding) had free fermionic timings up to 256 qubits and one layer. Indeed, only one layer was used to keep the computational overhead to a minimum. To approximate 512 qubits, the batch size has to be decreased, as keeping the same size led to memory leaks. Therefore, the 512 qubits parameter was not included, to keep the results consistent.

The training times for the fully quantum and free fermionic QRNNs are shown in figure 7.9. The free fermionic x-axis is scaled by $\log_2$ and incremented by 1 to match the Hilbert space of the two circuits. As observed, the time to train fully quantum and free fermionic QRNNs are closely related between 3 and 5 fully quantum qubits, after which free fermionic training time drastically increases. After 16 free fermionic qubits, the training time increases by around 5 with every increase of qubits, stretching into a factor of 10 in some tasks and towards 256 qubits. A probable reason for the dramatic training time increase of the free fermionic models is the limitations of the computer on which they were run. Indeed, with a 256 qubits model, this would involve creating a 512×512 matrix and multiplying it several times with matrices of the same dimensions. With the QNLP

[1] Refers to the padding mentioned in section 5.1

tasks, this constraint is even more apparent as those operations need to be performed for every word in a sentence and for every sentence in a text. As it can be seen in the train time graphs, the free fermions timings explode earlier than with Image Classification. This will load the CPU and the RAM, drastically slowing the training time.

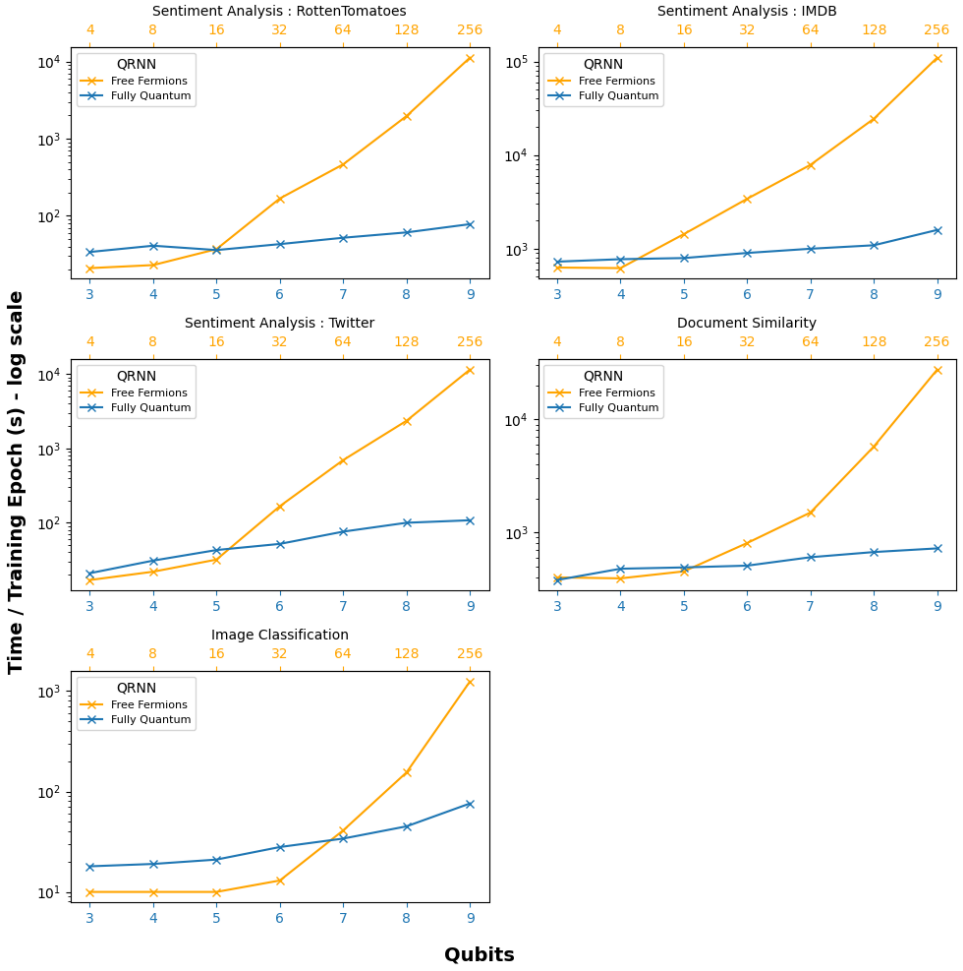


Figure 7.9: Time per Epoch to Train for each Tasks

However, this is performed using only a single layer, meaning minimum expressibility. As mentioned in section 6.3.1, the theory states that free fermionic circuits need a minimum of $\frac{N}{2}$ layers. In contrast, a fully quantum circuit will require at least as many parameters in its circuit as the number of dimensions in the Hilbert space. Therefore, if there are approximately N parameters per layer of the circuit, it needs around 2^{N-1} layers for minimum expressibility, which is exponential. A free fermionic circuit requires an exponential amount of qubits to match fully quantum circuits, but a fully quantum circuit needs an exponential number of layers. Considering the number of angles per model

shown in equations 6.1 and 6.2, it would be much slower to train fully quantum models with the same number of free fermionic qubits than just training free fermionic models. This difference in requirement explains the advantage of free fermionic circuits over fully quantum circuits, especially when scaling up QRNN models.

7.3 Over or Under-Parametrisation

Finally, let us look at the effect of increasing layers in free fermionic QRNNs and whether there is over or under-parametrisation at play, with which we can compare practice to theory. As stated in chapter 6, our results were found using a grid search that parsed through numbers of qubits and, for each qubit, iterated through all the layers from 1 to the number of qubits. This was performed on the Image Classification task, the easiest task to process. The aim here is to observe how the convergence speed of a model relates to the number of layers in a circuit.

Figures 7.10 and 7.11 respectively show the models' accuracy and loss across multiple epochs with varying numbers of qubits and layers. For clarity, the loss graph was shifted by subtracting every value by the minimum and then scaled by dividing every value by the difference between the maximum and minimum. In both figures, the models that ran with a number of layers less or equal to half the number of qubits were plotted in blue, called the blue models. The models with a greater number of layers were plotted in red, called the red models. This specific division with the number of layers greater or lower than $\frac{N}{2}$ is due to the theoretical line of a model being over or under-parametrised [58], respectively, as also mentioned in section 6.3.1. Theoretically, there should be a sharp increase in the convergence speed once a circuit has passed the $\frac{N}{2}$ mark, with models stagnating towards sub-optimal local minima in the under-parametrised case. This is the case in variational algorithms such as Quantum Alternating Operator Ansatz (QAOA) [59] and Hamiltonian Variational Ansatz (HVA) [58], where the models converge rapidly towards global minima past the threshold.

At first glance, the red models perform slightly better than the blue ones. We can see from the accuracy plots that the red models, on average, have a higher accuracy than the blue plots, very slightly. Conversely, on the loss plot, the red models are mostly lower than the blue models, suggesting that they also converge faster, only very slightly. However, there is not enough difference to confirm the theory that as the models increase in layers, they become more expressible and would likely resolve in a higher performance.

An interesting phenomenon occurs with the red models, which is more apparent with the models running on 11 qubits or more, and that does not seem to happen with the blue

models. On multiple layers, the model will converge towards an accuracy, mainly around the 85% mark, and stay within that boundary for a couple of epochs before increasing again and settling for an accuracy towards the high 90%.

However, a practical way to study the speed of convergence of a model is by measuring the entropy of the loss curve for each model. This is because entropy is related to how concentrated the loss of a model is at the start. Therefore, a lower entropy means a higher concentration of loss at the start, meaning the loss curve is steeper, and thus, the convergence of the model is higher. The first step is to normalise the loss curve by shifting the loss down by the minimum value then applying the following formula for entropy:

$$\text{Entropy} = -\sum_i \frac{f(x_i)}{\sum_k f(x_k)} \log\left(\frac{f(x_i)}{\sum_k f(x_k)} + \epsilon\right)$$

The constant ϵ here is 10^{-8} and is used to ensure we do not compute $\log_2(0)$. The resulting plot is shown in figure 7.12, where the entropy of each model is plotted against the number of layers over the number of qubits in the model. The learning rate here was chosen to portray this phenomenon better. A green dotted line is added on $x = 0.5$ to indicate the theoretical threshold. The section on the left of the line indicates under-parametrising, and the right indicates over-parametrising. A lower entropy occurs when the loss curve is steeper, thus indicating a faster entropy. With a linear regression on each plot, we notice a downward slope, indicating that the models converge faster as the number of layers increases. However, looking globally at the data, we notice no sharp decrease in entropy once the models move past the threshold. Instead, there is a more gradual decline, with the entropy decreasing even before the threshold, with some models decreasing close to linearly, for example, in the cases of 13, 15 and 16 qubits models.

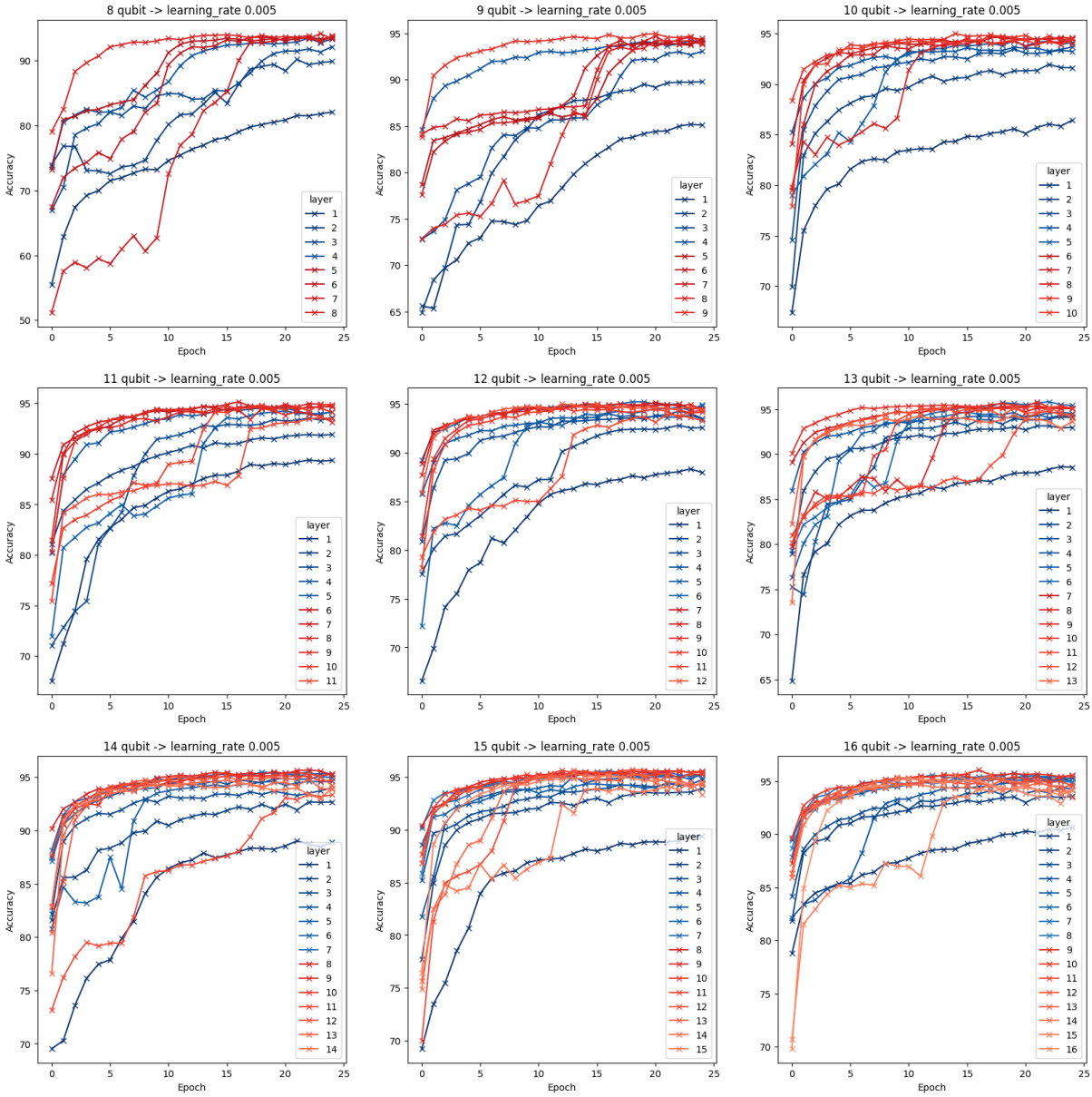


Figure 7.10: Accuracy of Grid Search

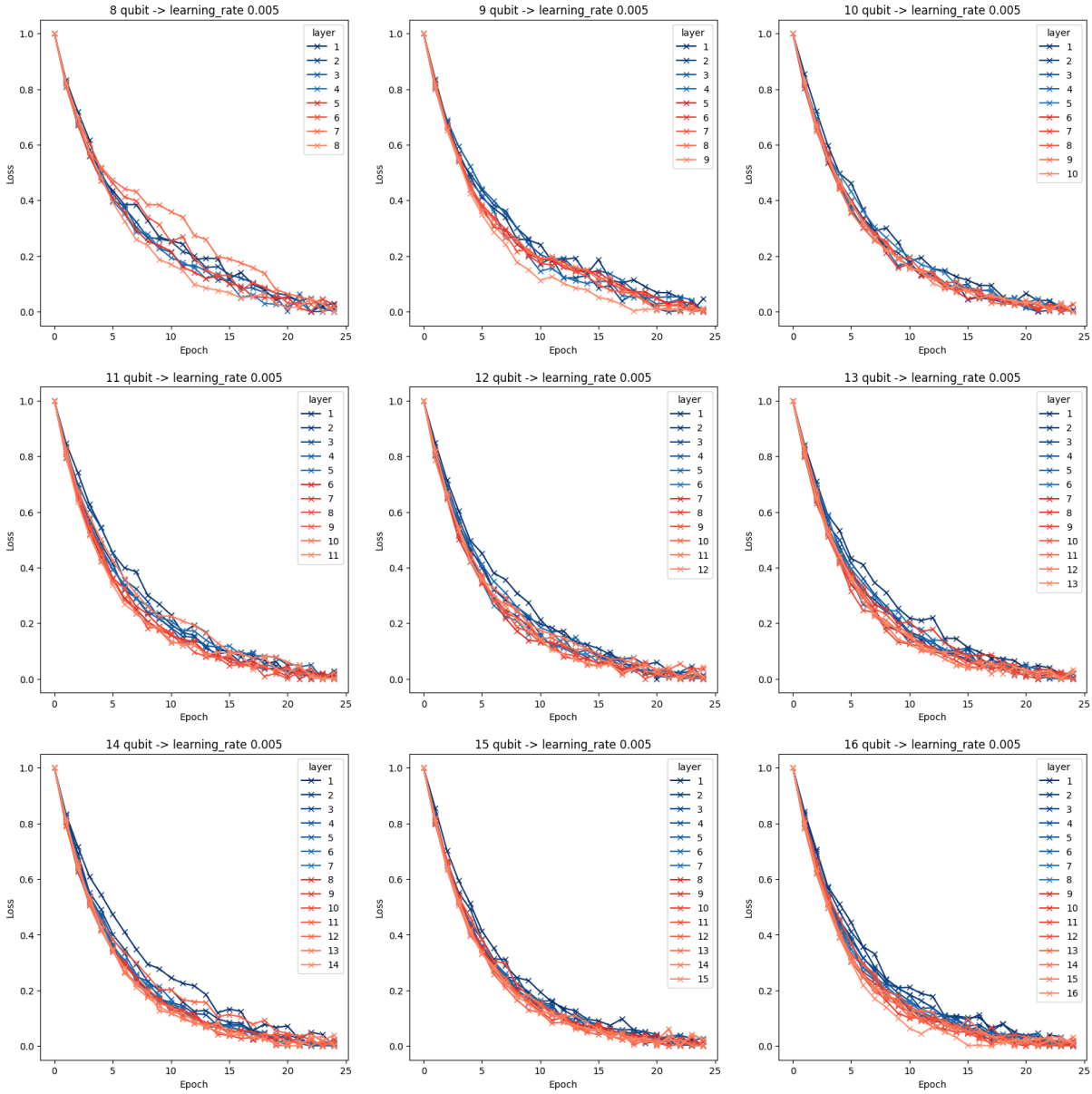


Figure 7.11: Loss of Grid Search

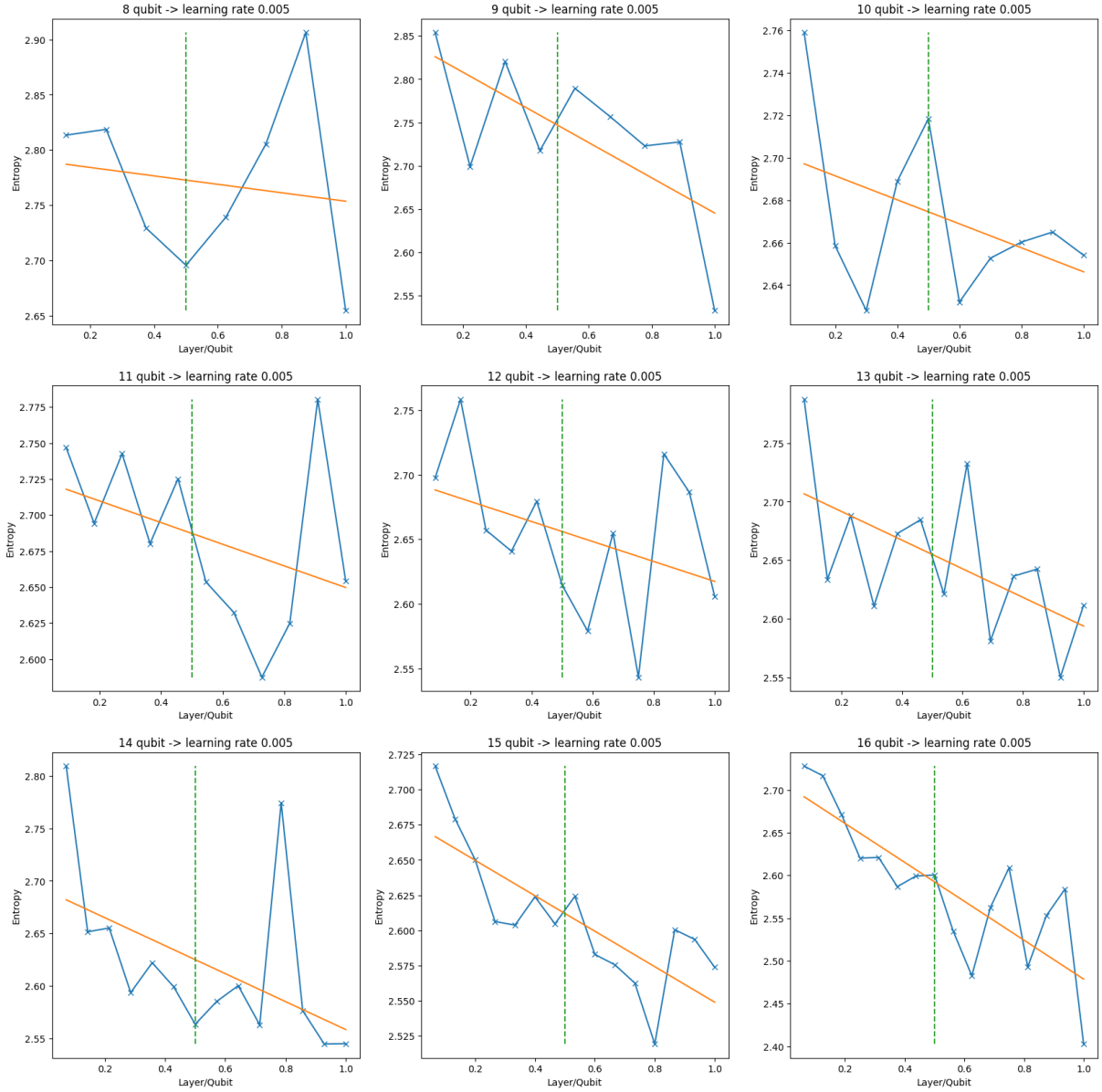


Figure 7.12: Entropy from Grid Search

8 Conclusion

Contents

8.1 Conclusion and Discussion of Results

In this research, we have investigated the use of free fermionic circuits to solve Quantum Natural Language Processing tasks. We have proposed a set of quantum gates that satisfies the matchgate property, making it classically simulable. We then evaluated the circuits' performance on the following tasks: Sentiment Analysis, Document Similarity, Text Generation as well as Image Classification. The circuits were used as part of a Quantum Recurrent Neural Network, the structure of which was carefully adapted to fit each QNLP task.

Our study of such systems shows that when taking the maximum reached accuracy (c.f. section 7.1) free fermionic QRNNs perform slightly better or the same as fully quantum models in tasks like Document Similarity and Text Generation, and even better than classical models like with Sentiment Analysis. This finding is significant because free fermionic models are computationally less demanding than fully quantum models. This research suggests that free fermionic circuits could be a more practical and resource-efficient

alternative to fully quantum circuits in some sequential data-heavy applications, reducing the computational overhead while maintaining competitive accuracy.

Given the current immense array of machine learning tasks, there could exist a task that requires the fully quantum model that we did not explore here. It might also be the case that the PQC architecture is not suited to fully quantum models. And in these cases, a classically simulable free-fermion equivalent suffices.

Our research also provides valuable insights into the optimisation of quantum circuits, whether free fermionic or fully quantum, by demonstrating that increasing the number of qubits and layers does not always translate into better performance (as shown in figure 7.8). It highlights that careful tuning of layers and qubits can optimise performance without exponentially increasing the computational demand (as shown in section 7.1 with the table of best hyperparameters).

Considering the time to train the models, if we desire the Hilbert spaces of free fermionic and fully quantum circuits to match, then, free fermionic circuits models will take much longer to train (c.f. figure 7.9). But if we are willing to accept an approximation of the Hilbert space and match the number of free fermionic qubits to fully quantum qubits, essentially elongating the free fermionic curve of figure 7.9 on the right, free fermionic models are faster. And we know that a few parameters can still result in high accuracies from section 7.1.

The analysis of different free fermionic model configurations reveals that increasing the number of layers relative to the number of qubits improves accuracy only slightly. It does not substantially enhance performance, suggesting marginal gains in the model's capacity to be expressive. Moreover, the entropy analysis indicates that increasing layers per qubit decreases entropy, implying that the convergence rate increases with the number of layers. Theoretically, there is a hard threshold for the minimum layer requirement for maximum expressibility, that of $\frac{N}{2}$. In practice, as can be seen in the graph of chapter 7, the entropy of the models decreases even before reaching $\frac{N}{2}$. Instead of having a hard threshold, it is instead a gradual decrease. This hard threshold is a result that holds for some QML models, like QAOA and HVA, as stated in section 7.3, but it is not a general rule for all QML models. Nevertheless, since free fermionic models are similar, we would have expected to see the same hard threshold phenomenon, which is not the case. Nonetheless, it could be the case that there is a sharp threshold, but the only way to observe would be by initialising a model with a very large number of qubits.

8.2 Future Work

The most apparent course of action for future work would be to leverage better computational resources to make this research even more rigorous. With higher processing power, we could take an average over several initial conditions to see if we can get a cleaner exploration into the range of accuracy, training, and parametrisation of free fermionic models. Averaging the results over multiple experiments will enable the formation of error bars on the accuracy range attainable with equal parameters and random seeding instead of using a seed. Additionally, the model could be initialised with the number of qubits more times than those equal to powers of 2, which will give a cleaner progression on the training of the models, helping us to identify the exact point when free fermionic models train longer than fully quantum models. As such, by expanding the quantum models to thousands of qubits, we would be able to test for any increased accuracy or to verify if the sharp threshold theory activates past a certain point instead of the gradual convergence speed increase that we observed.

Furthermore, multiple QML models, other than QRNNs, could be implemented using free fermionic circuits. It would be beneficial to use free fermionic circuits in models such as DisCoCat [60], or the more advanced framework, DisCoCirc [61], again comparing it with fully quantum methods. We would observe if other models could bring out the full quantum advantage or if once more, classically simulable methods are just as good or even slightly better. As such, the warm-starting method (mentioned in chapter 1), in combination with the free fermionic models, could be used to bring out a fully quantum advantage.

As we know, free fermionic circuits are classically simulable. Still, other circuits can be classically simulated, like those made up of only the Clifford set (briefly presented in chapter 3). As mentioned previously, the Clifford set is discrete, which leaves no room for continuous optimisation. Therefore, using Clifford gates as part of a Machine Learning objective would involve developing a theoretical framework to optimise circuits using only the Clifford set. This could then be applied to the QRNN models used for this research or the QDisCoCat models. A comparison could then be made with more classically simulable fragments to verify if classically simulable quantum fragments could bring an advantage compared to fully quantum circuits; potentially further confirming that pure, fully quantum methods need further development to achieve quantum advantage. But, it would also be interesting to observe if some classical fragments are more performant than others. If classically simulable models match the accuracy of fully quantum models, fully quantum ties can be abandoned for Quantum Machine Learning. Instead, quantum-inspired structures can be built with classically simulable models, taking advantage of their performance.

They can be integrated into more complex ML setups without worrying about building an actual expensive quantum computer with the error correction code to go with it.

Appendices

A

Matchgate Calculations Appendix

Detailed proof of chosen matchgate

Using the Kronecker product, we can obtain the matrix form of the two parallel R_Z gates:

$$R_Z(\alpha) \bigotimes R_Z(\beta) = \begin{bmatrix} e^{i\frac{(-\alpha-\beta)}{2}} & 0 & 0 & 0 \\ 0 & e^{i\frac{(-\alpha+\beta)}{2}} & 0 & 0 \\ 0 & 0 & e^{i\frac{(\alpha-\beta)}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{(\alpha+\beta)}{2}} \end{bmatrix}$$

Applying the two parallel R_Z gates to R_{XX}, we then get:

$$(R_Z(\alpha) \bigotimes R_Z(\beta)) \cdot R_{XX}(\theta) =$$
$$\begin{bmatrix} e^{i\frac{(-\alpha-\beta)}{2}}\cos(\frac{\theta}{2}) & 0 & 0 & -ie^{i\frac{(-\alpha-\beta)}{2}}\sin(\frac{\theta}{2}) \\ 0 & e^{i\frac{(-\alpha+\beta)}{2}}\cos(\frac{\theta}{2}) & -ie^{i\frac{(-\alpha+\beta)}{2}}\sin(\frac{\theta}{2}) & 0 \\ 0 & -ie^{i\frac{(\alpha-\beta)}{2}}\sin(\frac{\theta}{2}) & e^{i\frac{(\alpha-\beta)}{2}}\cos(\frac{\theta}{2}) & 0 \\ -ie^{i\frac{(\alpha+\beta)}{2}}\sin(\frac{\theta}{2}) & 0 & 0 & e^{i\frac{(\alpha+\beta)}{2}}\cos(\frac{\theta}{2}) \end{bmatrix}$$

This makes the generating matrices:

$$A = \begin{bmatrix} e^{i\frac{(-\alpha-\beta)}{2}}\cos(\frac{\theta}{2}) & -ie^{i\frac{(-\alpha-\beta)}{2}}\sin(\frac{\theta}{2}) \\ -ie^{i\frac{(\alpha+\beta)}{2}}\sin(\frac{\theta}{2}) & e^{i\frac{(\alpha+\beta)}{2}}\cos(\frac{\theta}{2}) \end{bmatrix} \text{ and } B = \begin{bmatrix} e^{i\frac{(-\alpha+\beta)}{2}}\cos(\frac{\theta}{2}) & -ie^{i\frac{(-\alpha+\beta)}{2}}\sin(\frac{\theta}{2}) \\ -ie^{i\frac{(\alpha-\beta)}{2}}\sin(\frac{\theta}{2}) & e^{i\frac{(\alpha-\beta)}{2}}\cos(\frac{\theta}{2}) \end{bmatrix}$$

The first step of checking that this composition of gates creates a matchgates is

checking that the determinant of the generating matrices are equal:

$$\begin{aligned}\det(A) &= e^{i\frac{(-\alpha-\beta)}{2}} e^{i\frac{(\alpha+\beta)}{2}} \cos^2\left(\frac{\theta}{2}\right) - i^2 e^{i\frac{(-\alpha-\beta)}{2}} e^{i\frac{(\alpha+\beta)}{2}} \sin^2\left(\frac{\theta}{2}\right) \\ &= \cos^2\left(\frac{\theta}{2}\right) + \sin^2\left(\frac{\theta}{2}\right) = 1\end{aligned}$$

and

$$\begin{aligned}\det(B) &= e^{i\frac{(-\alpha+\beta)}{2}} e^{i\frac{(\alpha-\beta)}{2}} \cos^2\left(\frac{\theta}{2}\right) - i^2 e^{i\frac{(-\alpha+\beta)}{2}} e^{i\frac{(\alpha-\beta)}{2}} \sin^2\left(\frac{\theta}{2}\right) \\ &= \cos^2\left(\frac{\theta}{2}\right) + \sin^2\left(\frac{\theta}{2}\right) = 1\end{aligned}$$

So indeed, $\det(A) = \det(B)$.

The next step involves checking that $A, B \in U(2)$.

Detailing the complex conjugate of A and B, we get that:

$$A^\dagger = \begin{bmatrix} e^{i\frac{(\alpha+\beta)}{2}} \cos(\frac{\theta}{2}) & ie^{i\frac{(-\alpha-\beta)}{2}} \sin(\frac{\theta}{2}) \\ ie^{i\frac{(\alpha+\beta)}{2}} \sin(\frac{\theta}{2}) & e^{i\frac{(-\alpha-\beta)}{2}} \cos(\frac{\theta}{2}) \end{bmatrix}$$

and

$$B^\dagger = \begin{bmatrix} e^{i\frac{(\alpha-\beta)}{2}} \cos(\frac{\theta}{2}) & ie^{i\frac{(-\alpha+\beta)}{2}} \sin(\frac{\theta}{2}) \\ ie^{i\frac{(\alpha-\beta)}{2}} \sin(\frac{\theta}{2}) & e^{i\frac{(-\alpha+\beta)}{2}} \cos(\frac{\theta}{2}) \end{bmatrix}$$

Therefore

$$AA^\dagger = \begin{bmatrix} \cos^2(\frac{\theta}{2}) + \sin^2(\frac{\theta}{2}) & 0 \\ 0 & \cos^2(\frac{\theta}{2}) + \sin^2(\frac{\theta}{2}) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2$$

And similarly for $BB^\dagger = I_2$. The same result is achieved with $A^\dagger A = B^\dagger B = I_2$.

Therefore, $A, B \in U(2)$.

List of Figures

References

[1] Yanan Li et al. *Quantum Recurrent Neural Networks for Sequential Learning*. 2023. arXiv: 2302.03244 [quant-ph]. URL: https://arxiv.org/abs/2302.03244.

[2] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. URL: http://dx.doi.org/10.22331/q-2018-08-06-79.

[3] P Krantz et al. "A quantum engineer's guide to superconducting qubits". en. In: *Appl. Phys. Rev.* 6.2 (June 2019), p. 021318.

[4] T. D. Ladd et al. "All-Silicon Quantum Computer". In: *Phys. Rev. Lett.* 89 (1 2002), p. 017901. URL: https://link.aps.org/doi/10.1103/PhysRevLett.89.017901.

[5] G J Milburn. "Photons as qubits". In: *Phys. Scr.* T137 (Dec. 2009), p. 014003.

[6] Francesco Bernardini, Abhijit Chakraborty, and Carlos Ordóñez. *Quantum computing with trapped ions: a beginner's guide*. 2023. arXiv: 2303.16358.

[7] P.W. Shor. "Fault-tolerant quantum computation". In: *Proceedings of 37th Conference on Foundations of Computer Science*. 1996, pp. 56–65.

[8] Andrew N. Cleland. "An introduction to the surface code". In: *SciPost Phys. Lect. Notes* (2022), p. 49. URL: https://scipost.org/10.21468/SciPostPhysLectNotes.49.

[9] Ricard Puig et al. *Variational quantum simulation: a case study for understanding warm starts*. 2024. arXiv: 2404.10044 [quant-ph]. URL: https://arxiv.org/abs/2404.10044.

[10] Richard P Feynman. "Simulating physics with computers". en. In: *Int. J. Theor. Phys.* 21.6-7 (June 1982), pp. 467–488.

[11] D Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". en. In: *Proc. R. Soc. Lond.* 400.1818 (July 1985), pp. 97–117.

[12] D Deutsch and R Jozsa. "Rapid solution of problems by quantum computation". en. In: *Proc., Math. Phys. Sci.* 439.1907 (Dec. 1992), pp. 553–558.

[13] S Lloyd. "Universal quantum simulators". en. In: *Science* 273.5278 (Aug. 1996), pp. 1073–1078.

[14] M A Bredig. "The born-einstein letters". en. In: *Science* 180.4091 (June 1973), p. 1118.

[15] Saswato R Das. *A single-atom transistor*. en. https://spectrum.ieee.org/a-singleatom-transistor. Accessed: 2024-6-19. Feb. 2012.

[16] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[17] Charles London et al. “Peptide binding classification on quantum computers”. In: *Quantum Machine Intelligence* 6.2 (June 2024). URL: http://dx.doi.org/10.1007/s42484-024-00154-3.

[18] Thomas W. Malone et al. “Artificial Intelligence and the Future of Work”. In: *MIT Work of the Future* (2020). URL: https://workofthefuture-taskforce.mit.edu/wp-content/uploads/2020/12/2020-Research-Brief-Malone-Rus-Laubacher2.pdf.

[19] Solveig Badillo et al. “An Introduction to Machine Learning”. In: *Clinical Pharmacology and Therapeutics* 107 (2020), pp. 871–885. URL: https://api.semanticscholar.org/CorpusID:212406273.

[20] Kevin Gurney. *An Introduction to Neural Networks*. USA: Taylor & Francis, Inc., 1997.

[21] K R Chowdhary. “Natural Language Processing”. In: *Fundamentals of Artificial Intelligence*. New Delhi: Springer India, 2020, pp. 603–649.

[22] Zhan Yu et al. *Provable Advantage of Parameterized Quantum Circuit in Function Approximation*. 2023. arXiv: 2310.07528 [quant-ph]. URL: https://arxiv.org/abs/2310.07528.

[23] Joseph Bowles, Shahnawaz Ahmed, and Maria Schuld. *Better than classical? The subtle art of benchmarking quantum machine learning models*. 2024. arXiv: 2403.07059 [quant-ph]. URL: https://arxiv.org/abs/2403.07059.

[24] David Peral-García, Juan Cruz-Benito, and Francisco José García-Peñalvo. “Systematic literature review: Quantum machine learning and its applications”. In: *Computer Science Review* 51 (2024), p. 100619. URL: https://www.sciencedirect.com/science/article/pii/S1574013724000030.

[25] Chen Zhao and Xiao-Shan Gao. “Analyzing the barren plateau phenomenon in training quantum neural networks with the ZX-calculus”. In: *Quantum* 5 (June 2021), p. 466. URL: http://dx.doi.org/10.22331/q-2021-06-04-466.

[26] J.C. Spall. “Multivariate stochastic approximation using a simultaneous perturbation gradient approximation”. In: *IEEE Transactions on Automatic Control* 37.3 (1992), pp. 332–341.

[27] Xavier Bonet-Monroig et al. “Performance comparison of optimization methods on variational quantum algorithms”. In: *Phys. Rev. A* 107 (3 2023), p. 032407. URL: https://link.aps.org/doi/10.1103/PhysRevA.107.032407.

[28] Ken M. Nakanishi, Keisuke Fujii, and Synge Todo. “Sequential minimal optimization for quantum-classical hybrid algorithms”. In: *Physical Review Research* 2.4 (Oct. 2020). URL: http://dx.doi.org/10.1103/PhysRevResearch.2.043158.

[29] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG]. URL: https://arxiv.org/abs/1609.04747.

[30] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[31] Austin Williams et al. “Stochastic gradient descent for optimization for nuclear systems”. en. In: *Sci. Rep.* 13.1 (May 2023), p. 8474.

[32] Yanan Li et al. *Quantum Recurrent Neural Networks for Sequential Learning*. 2023. arXiv: 2302.03244.

[33] Vincent Wang-Mascianica, Jonathon Liu, and Bob Coecke. *Distilling Text into Circuits*. 2023. arXiv: 2301.10595 [cs.CL]. URL: https://arxiv.org/abs/2301.10595.

[34] M. Van den Nest. *Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond*. 2009. arXiv: 0811.0898 [quant-ph]. URL: https://arxiv.org/abs/0811.0898.

[35] T H Johnson et al. "Solving search problems by strongly simulating quantum circuits". en. In: *Sci. Rep.* 3.1 (2013), p. 1235.

[36] Daniel Gottesman. *The Heisenberg Representation of Quantum Computers*. 1998. arXiv: quant-ph/9807006 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/9807006.

[37] M. Cerezo et al. *Does provable absence of barren plateaus imply classical simulability? Or, why we need to rethink variational quantum computing*. 2024. arXiv: 2312.09121 [quant-ph]. URL: https://arxiv.org/abs/2312.09121.

[38] Gabriel da Fonseca Matos. *Variational quantum algorithms and the complexity of many-body systems*. 2023. URL: https://etheses.whiterose.ac.uk/33650/.

[39] Graham Farmelo. *The strangest man*. London, England: Basic Books, Aug. 2009.

[40] Steven R. Elliott and Marcel Franz. "Colloquium: Majorana fermions in nuclear, particle, and solid-state physics". In: *Reviews of Modern Physics* 87.1 (Feb. 2015), pp. 137–163. URL: http://dx.doi.org/10.1103/RevModPhys.87.137.

[41] Robert Martin Eisberg and Robert Resnick. *Quantum physics of atoms, molecules, solids, nuclei and particles; 2nd ed.* New York, NY: Wiley, 1985. URL: https://cds.cern.ch/record/105889.

[42] P Jordan and E Wigner. "Über das Paulische Äquivalenzverbot". de. In: *Eur. Phys. J. A* 47.9-10 (Sept. 1928), pp. 631–651.

[43] Leslie G. Valiant. "Quantum Circuits That Can Be Simulated Classically in Polynomial Time". In: *SIAM Journal on Computing* 31.4 (2002), pp. 1229–1254. URL: https://doi.org/10.1137/S0097539700377025.

[44] E. Knill. *Fermionic Linear Optics and Matchgates*. 2001. arXiv: quant-ph/0108033 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0108033.

[45] Barbara M. Terhal and David P. DiVincenzo. *Adaptive Quantum Computation, Constant Depth Quantum Circuits and Arthur-Merlin Games*. 2004. arXiv: quant-ph/0205133 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0205133.

[46] Richard Jozsa and Akimasa Miyake. "Matchgates and classical simulation of quantum circuits". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 464.2100 (July 2008), pp. 3089–3106. URL: http://dx.doi.org/10.1098/rspa.2008.0189.

[47] Archontia C. Giannopoulou, Meike Hatzel, and Sebastian Wiederrecht. *Braces of Perfect Matching Width 2*. 2024. arXiv: 1902.06307 [math.CO]. URL: https://arxiv.org/abs/1902.06307.

[48] Leslie G. Valiant. "Quantum computers that can be simulated classically in polynomial time". In: *Symposium on the Theory of Computing*. 2001. URL: https://api.semanticscholar.org/CorpusID:17054776.

[49] P.W. Kasteleyn. “The statistics of dimers on a lattice: I. The number of dimer arrangements on a quadratic lattice”. In: *Physica* 27.12 (1961), pp. 1209–1225. URL: https://www.sciencedirect.com/science/article/pii/0031891461900635.

[50] H. N. V. Temperley and Michael E. Fisher. “Dimer problem in statistical mechanics-an exact result”. In: *The Philosophical Magazine: A Journal of Theoretical Experimental and Applied Physics* 6.68 (1961), pp. 1061–1063. URL: https://doi.org/10.1080/14786436108243366.

[51] P W Kasteleyn. “Dimer statistics and phase transitions”. en. In: *J. Math. Phys.* 4.2 (Feb. 1963), pp. 287–293.

[52] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[53] Johannes Bausch. *Recurrent Quantum Neural Networks*. 2020. arXiv: 2006.14619 [cs.LG]. URL: https://arxiv.org/abs/2006.14619.

[54] Wenduan Xu et al. *Quantum Recurrent Architectures for Text Classification*. Submitted.

[55] Hanrui Wang et al. “Quantumnas: Noise-adaptive search for robust quantum circuits”. In: *The 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA-28)*. 2022.

[56] Efekan Kökcü et al. “Algebraic compression of quantum circuits for Hamiltonian evolution”. In: *Phys. Rev. A* 105 (3 2022), p. 032420. URL: https://link.aps.org/doi/10.1103/PhysRevA.105.032420.

[57] Noor Baha Aldin and Shaima Safa Aldin Baha Aldin. “Accuracy Comparison of Different Batch Size for a Supervised Machine Learning Task with Image Classification”. In: *2022 9th International Conference on Electrical and Electronics Engineering (ICEEE)*. 2022, pp. 316–319.

[58] Xuchen You, Shouvanik Chakrabarti, and Xiaodi Wu. *A Convergence Theory for Over-parameterized Variational Quantum Eigensolvers*. 2022. arXiv: 2205.12481 [quant-ph]. URL: https://arxiv.org/abs/2205.12481.

[59] Bobak Toussi Kiani, Seth Lloyd, and Reevu Maity. *Learning Unitaries by Gradient Descent*. 2020. arXiv: 2001.11897 [quant-ph]. URL: https://arxiv.org/abs/2001.11897.

[60] Robin Lorenz et al. “QNLP in practice: Running compositional models of meaning on a quantum computer”. In: *J. Artif. Intell. Res.* 76 (Apr. 2023), pp. 1305–1342.

[61] Tuomas Laakkonen, Konstantinos Meichanetzidis, and Bob Coecke. “Quantum Algorithms for Compositional Text Processing”. In: *Electronic Proceedings in Theoretical Computer Science* 406 (Aug. 2024), pp. 162–196. URL: http://dx.doi.org/10.4204/EPTCS.406.8.