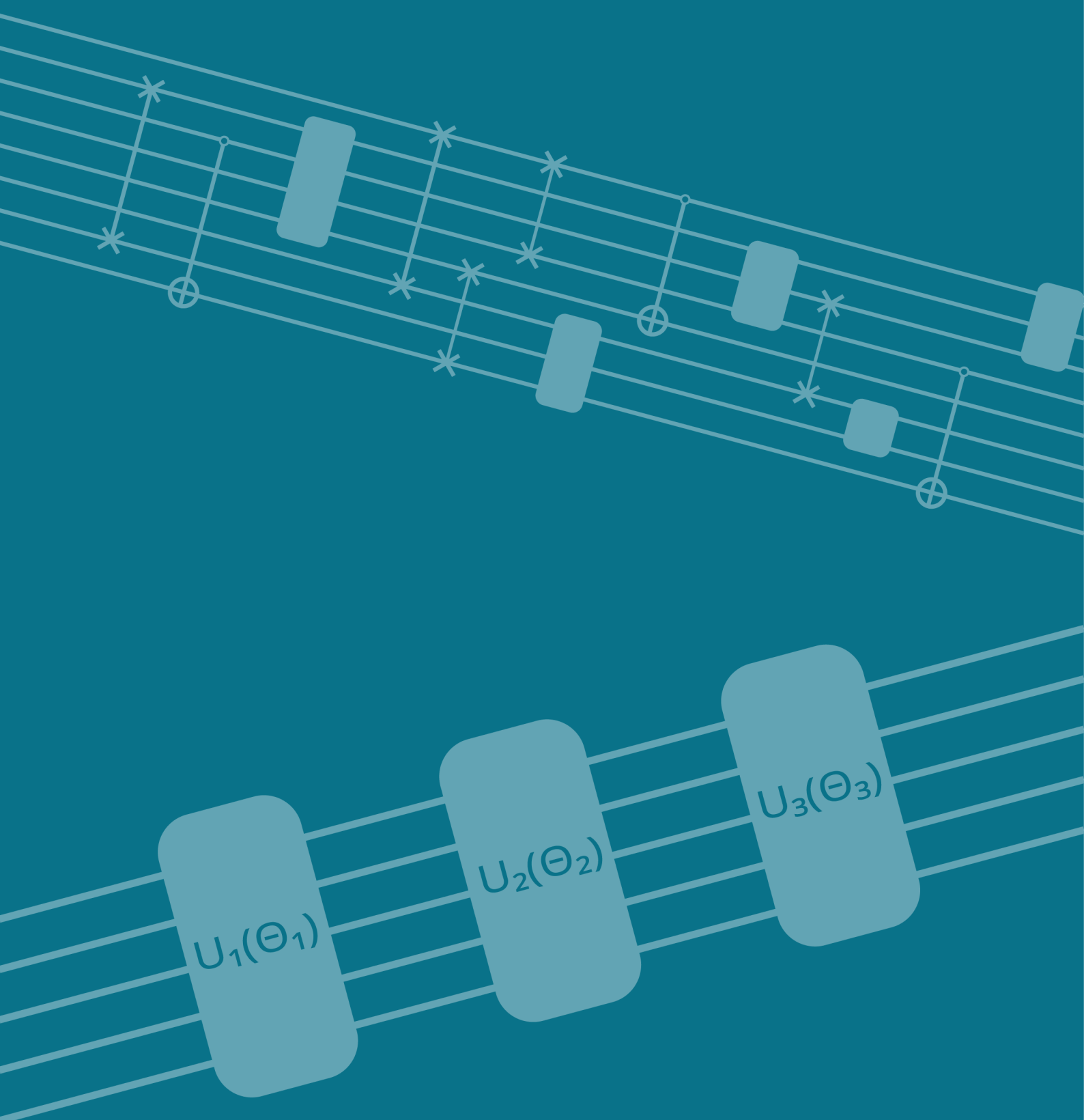


# Learning based hardware-centric quantum circuit generation

by Merel Schalkers





# Learning based hardware-centric quantum circuit generation

by

Merel Schalkers

to obtain the degree of Master of Science  
at the Delft University of Technology,  
to be defended publicly on Wednesday May 19, 2021 at 13:00.

Student number: 4932110  
Project duration: September 1, 2020 – May 12, 2021  
Thesis committee: Prof. dr. ir. K. Aardal, TU Delft, responsible professor  
Dr. M. Möller, TU Delft, supervisor  
Dr. D. de Laat, TU Delft, supervisor

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.



# Abstract

Large fault-tolerant universal gate quantum computers will provide a major speed-up to a variety of common computational problems. While such computers are years away, we currently have noisy intermediate-scale quantum (NISQ) computers at our disposal. In this project we present two quantum machine learning approaches that can be used to find quantum circuits suitable for specific NISQ devices.

We present one gradient-based and one non-gradient based machine learning approach to optimize the created quantum circuits, to best mimic the behaviour of a given function up to measurement. We make sure that the created quantum circuits obey the restrictions of the chosen hardware, therefore the approaches can be used to find circuits perfectly suited for specific NISQ devices. This enables the user to make the best use of quantum technology in the near future.

In doing this we created our own quantum simulator which can be used to simulate small quantum circuits that obey hardware restrictions. We also present the method used to implement this simulator. Finally we present the results of applying both machine learning approaches to different problem types and compare their performance.



# Preface

This thesis is written as a final project to finish the Master of Applied Mathematics at the TU Delft. I truly enjoyed my time working on this project and I hope the reader finds it as enjoyable as I did.

I would like to express my gratitude towards Matthias Möller for his continued trust in me and the many hours he spent to guide me on this project. I would also like to thank David de Laat for his valuable advice and feedback, which greatly helped improve and shape this project. Furthermore I want to thank Karen Aardal for being part of the thesis committee.

I want to thank Thomas Hazenoot for creating the cover design and Nando Leijenhorst, Clara Frick and Michael Goddijn for proofreading this thesis.

Furthermore I want to express my love and appreciation towards my family and friends who have always supported my endeavours and who play a large part in all of my accomplishments.

*Merel Schalkers  
Delft, May 2021*





# Contents

<b>Glossary</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Aim of the project . . . . .	1
1.2 Influential properties of quantum hardware . . . . .	2
1.2.1 Noise . . . . .	2
1.2.2 Native gates . . . . .	2
1.2.3 Hardware connectivity . . . . .	3
1.3 Possible approaches and earlier research . . . . .	3
1.3.1 Ansatz-centric ansatz . . . . .	4
1.3.2 Circuit-centric ansatz . . . . .	4
1.3.3 Hardware-centric ansatz . . . . .	4
<b>2 Quantum hardware-centric circuit</b>	<b>7</b>
2.1 Quantum hardware . . . . .	7
2.1.1 NISQ devices and quantum volume . . . . .	7
2.1.2 IBM quantum experience . . . . .	8
2.1.3 Rigetti's Aspen quantum computers . . . . .	9
2.2 Valid quantum ansatz . . . . .	9
2.3 Problem description . . . . .	10
2.3.1 Correctness of formulation . . . . .	11
2.3.2 Problem size . . . . .	12
2.4 Cost of the ansatz . . . . .	12
2.4.1 Cost estimation function . . . . .	12
2.4.2 Cost estimation and noise . . . . .	14
<b>3 Kronx quantum circuit simulator</b>	<b>15</b>
3.1 Algorithm 993 . . . . .	15
3.2 Non-adjacent two-qubit extension . . . . .	19
<b>4 Machine learning for quantum circuit simulation</b>	<b>23</b>
4.1 Ansatz creation and selection . . . . .	23
4.2 Parameter optimization . . . . .	23
4.3 Gradient-free machine learning - particle swarm optimization . . . . .	24
4.3.1 PSO . . . . .	24
4.3.2 PSO for hardware-centric quantum circuit generation . . . . .	25
4.4 Gradient-based machine learning . . . . .	25
4.4.1 Forward pass . . . . .	25
4.4.2 Backpropagation . . . . .	26
4.4.3 Partial derivatives of quantum nodes . . . . .	27
4.4.4 Gradient-based backpropagation for quantum circuits . . . . .	31
<b>5 Results</b>	<b>33</b>
5.1 Simple function . . . . .	33
5.1.1 PSO - basis states . . . . .	34
5.1.2 Gradient descent - basis states . . . . .	35
5.1.3 PSO - complex states . . . . .	35
5.1.4 Gradient descent - real states . . . . .	36
5.1.5 Simple problem with $n > 2$ . . . . .	36
5.2 CNOT gate . . . . .	37
5.2.1 PSO - basis states . . . . .	37

5.2.2	Gradient descent - basis states . . . . .	37
5.2.3	PSO - complex states . . . . .	38
5.2.4	Gradient descent - real states . . . . .	38
5.3	The search problem . . . . .	38
5.3.1	$N = 4$ and $t = 1$ . . . . .	41
5.3.2	$N = 8$ and $t = 2$ . . . . .	43
5.4	Comparison of performance of PSO and gradient-based machine learning. . . . .	44
5.5	Overview of results . . . . .	44
<b>6</b>	<b>Discussion</b> . . . . .	<b>47</b>
6.1	Recommendations for further research . . . . .	48
<b>A</b>	<b>Quantum computing</b> . . . . .	<b>51</b>
A.1	Qubits . . . . .	51
A.1.1	Single qubit . . . . .	51
A.1.2	Multiple qubit states . . . . .	52
A.1.3	Superposition . . . . .	53
A.1.4	Qubit order and indentation . . . . .	53
A.1.5	Measurement of a qubit . . . . .	53
A.1.6	Entanglement. . . . .	54
A.1.7	Bra-ket notation. . . . .	54
A.2	Quantum Gates. . . . .	54
A.2.1	Quantum operations as matrix multiplications . . . . .	55
A.2.2	Universal gate set . . . . .	55
A.2.3	Phase of a quantum gate . . . . .	56
<b>B</b>	<b>The quantum circuit generator</b> . . . . .	<b>57</b>
B.1	Library. . . . .	57
B.1.1	connectivity.py . . . . .	57
B.1.2	native_gates.py. . . . .	57
B.1.3	ibm.py . . . . .	57
B.1.4	rigetti.py. . . . .	57
B.1.5	backends.py . . . . .	57
B.1.6	distances.py . . . . .	58
B.2	Engine. . . . .	58
B.2.1	circuit_simulator.py . . . . .	58
B.2.2	kronx.py. . . . .	58
B.2.3	operations.py . . . . .	58
B.2.4	order_creator.py . . . . .	58
B.2.5	thetas_creator.py . . . . .	58
B.2.6	circuit_creator.py . . . . .	58
B.3	ML. . . . .	58
B.3.1	gradient.py . . . . .	58
B.3.2	PSO.py . . . . .	59
B.3.3	qcg.py . . . . .	59
B.3.4	runner.py . . . . .	59
<b>C</b>	<b>Quantum gates</b> . . . . .	<b>61</b>
C.1	Single-qubit gates . . . . .	61
C.2	Two-qubit gates. . . . .	63
C.3	Three-qubit gates. . . . .	65
C.4	Quantum circuit symbols . . . . .	66
<b>D</b>	<b>Properties of considered quantum computers and their native gates</b> . . . . .	<b>67</b>
D.1	Gradients of implemented native gates . . . . .	67
D.1.1	Native gate: $R_Z(\theta)$ . . . . .	67
D.1.2	Native gate: $R_{ZX}(\theta)$ . . . . .	68
D.1.3	Extension of partial derivatives to circuits with more qubits . . . . .	68

D.2 Qubit connectivity of considered quantum computers . . . . . 70

# Glossary

**ansatz** In this thesis the term ansatz refers to an initial guess of placement of the quantum gates in the circuit. Therefore an ansatz describes a quantum circuit of which the angles of the gates still need to be determined.

**basis state** One of the  $2^n$  quantum states that together form a basis to span the  $2^n$  dimensional Hilbert space. Unless specified otherwise, a basis state is a member of the computational basis.<sup>1</sup>

**computational basis** The standard choice of basis to span the  $2^n$  dimensional Hilbert space, equivalent to the standard basis vectors. The  $i$ -th element of the computational basis is written  $|i\rangle$ .

**circuit depth** The depth of a circuit is the amount of gates applied to each qubit in a circuit.

**circuit layer** A single layer of a quantum circuit, which consists of applying exactly one quantum gate to each qubit.

**epoch** The number of epochs refers to the amount of time a machine learning algorithm works through the set of training data.

**hardware connectivity** Refers to which qubits are connected on a chosen quantum computer. A two-qubit gate can be directly applied on two qubits only when they are connected on the hardware.

**native gate** Quantum gate that can be directly performed on the hardware considered.

**NISQ device** Noisy intermediate scale quantum devices are relatively small quantum computers that are heavily influenced by noise.

**PSO** Particle swarm optimization (PSO) is a computational optimization method.

**quantum algorithm** A sequence of quantum gates that perform a desired operation or can be used to solve a class of problems.

**quantum circuit** A sequence of quantum gates to be performed on the qubits.

**quantum gate** A basic operation that can be performed on a (small) number of qubits.

**valid ansatz** A valid ansatz is an ansatz which obeys the restrictions of the chosen quantum hardware.

---

<sup>1</sup>The variable  $n$  represents the number of qubits.



# Introduction

This chapter introduces the reader to the optimization problem we consider in this project. We will first give an overview of the current state of quantum technology and how the limitations of modern quantum computers make it infeasible to run textbook quantum algorithms. This leads us to the problem considered in this project, namely that of finding quantum circuits which can be feasibly run on a near-term quantum device to mimic the behaviour of a chosen function. In the final section of this chapter, we give an overview of several approaches that could be used to find such quantum circuits and we indicate which approach we have taken.

Throughout this thesis we assume the reader has a basic knowledge of quantum computing. If this is not the case, we recommend first reading Appendix A.

## 1.1. Aim of the project

In this project, we present a computational approach that enables the user to find quantum circuits that mimic the action of a given quantum operation on specified near-term quantum hardware. Quantum computers are the quantum equivalent of classical computers which behave differently from classical computers. Quantum computers are known to provide speed-ups for several computational problems, which are predicted to have a large impact on society [Ver+] [Sho94] [Gro96].

Known quantum algorithms require quantum computers of thousands to millions of qubits to run reliably on real world problems. Therefore, running such algorithms in the near term is unlikely due to the instability and limited qubit numbers of today's quantum technology. Current quantum technologies are nearing the age of NISQ computers. NISQ stands for Noisy Intermediate-Scale Quantum and such computers have around 50-100 noisy qubits to their disposal [Pre18]. The fact that the qubits are noisy implies that they have a limited coherence time and gate fidelity, on top of that there is a trade-off between coherence time, gate fidelity and the amount of qubits [MN01].

The aforementioned limitations of modern quantum computers imply that we are not currently able to run quantum algorithms of potential interest. In this project, we present two quantum machine learning approaches that can be used to find quantum circuits which are specifically designed for the hardware of a chosen quantum computer. The quantum circuits found take into account the limitations of the chosen hardware by restricting the potential ansatzes to instances which obey the hardware limitations. Here the term ansatz refers to the layout of the quantum circuit in terms of which gate is applied to which qubit in each layer. By allowing any potential ansatz that obeys the restrictions of the hardware, we ensure that we do not add unnecessary and potentially unforeseen depth to the circuit. This, in turn, ensures that we make optimal use of the limited amount of qubits and circuit depth available on near term quantum computers.

In this project, we first give a description of the different hardware restrictions modern quantum computers have and the choices we made regarding how to represent these restrictions in the project. We then take these restrictions to give a description of the created problem we want to optimize over. The aim of our approach is to find a quantum circuit that is able to mimic the behaviour of a chosen function. We optimize over the distance between the probability vectors of finding a certain basis state

upon measurement for the result of running our created circuit and the desired operation.

After having presented this problem, we present the two different machine learning methods used to find a solution with minimal cost. Note that the problem of finding a suitable quantum circuit to mimic the behaviour of an arbitrary unitary operation is known to be generally NP-complete [BKM18], and a circuit to approximate a chosen operation for  $n$  qubits might require an exponential number of gates [NC16] [SBM06].

In this project, we present both a gradient-based and a non-gradient based machine learning approach to find an optimal circuit to mimic the desired function. We also present our own implementation of a small quantum simulator used in this project. Finally, we present the results of our implementation of the presented quantum circuit generating approach. We used our implementation to find quantum circuits to mimic the action of a given operation for which quantum algorithms are known, though a quantum algorithm need not be known for our approach to be applicable.

We implemented the approach described making use of the PyTorch, PennyLane and PySwarms packages [Pas+19] [Ber+18] [Mir18]. We present the results for the gradient-based and the non-gradient based machine learning approaches separately, to compare the performance of both approaches for the given problems.

## 1.2. Influential properties of quantum hardware

There are several hardware characteristics that influence the performance of a quantum computer [Mol+18]. In the following section, we will briefly discuss the three main factors that need to be taken into account when translating theoretical quantum computing to its real world application.

### 1.2.1. Noise

Noise is an important factor in modern-day quantum computers. There are two types of noise which influence the performance of a quantum computer.

First of all, the application of a physical quantum gate to one or multiple qubits will never be precisely equal to its theoretical counterpart. In this case noise reflects that we have imperfect control over the qubits, and so when we aim to apply a certain operation to the qubits, the operation that gets implemented in reality is slightly different [Pre18]. Not all native quantum gates are physically equally noisy when applied on specific quantum hardware. Therefore, applying many of a certain type of quantum gate on a certain hardware will have a greater negative affect on the fidelity of the outcome than another type of quantum gate would. Specifically two-qubit gates are known to be more noisy than single qubit operations [Pre18] [Mol+18].

The second type of noise that has an effect on the performance of modern day quantum computers, is the noise associated with the interaction a quantum system has with its environment [NC16]. It is unavoidable that the qubits of a quantum computer interact with the environment in some capacity, this interaction negatively impacts the stability of the system which will ultimately affect the final measurement outcomes. The term coherence time refers to the time a specific quantum device can keep a stable state, after this interactions with the environment have influenced the system to the extent that a measurement outcome is meaningless.

### 1.2.2. Native gates

The gates used in quantum algorithms and the gates natively applied to qubits on physical quantum computers are not always the same. The quantum gates natively implemented on a physical quantum computer usually consist of two single-qubit gates that represent rotations around one of the axes of the Bloch sphere and an entanglement two-qubit gate; see Appendix A.1.1 and A.2.2. Precisely which quantum gates are physically implemented depends on the technical features of the quantum computer. In theoretical quantum computing, however, any unitary operation can be used as a quantum gate. The gates commonly used in theoretical quantum computing are often handy in their matrix expression and logical features.

Due to this mismatch in theoretical and physical quantum gates, known quantum algorithms need to be decomposed into physical quantum gates prior to running the algorithm on a quantum computer. This usually leads to quantum algorithms having more depth when ran on a physical quantum computer, then can be seen from the algorithm on paper. As described in Section 2.1.1 modern quantum computers perform best when the depth of the circuit is as small as possible. Since this decomposition adds depth

to the circuit, this has a negative impact on the feasibility of implementing the designed algorithm.

### 1.2.3. Hardware connectivity

In a theoretical setting all qubits in a quantum computer are connected, enabling the user to perform a two- or three-qubit gate on any combination of qubits. Physical quantum computers, however, do not have a physical connection between all  $n$  qubits in the system.

When a quantum circuit requires a two-qubit gate to be performed between two qubits which are not physically connected, this gate is decomposed on the physical quantum computer into multiple two-qubit gates. As a first step, two-qubit gates need to be performed that virtually swap the order of the qubits. These qubits are swapped such that the two qubits, on which we wish to perform the two-qubit gate, are virtually moved to adjacent positions. Then the two-qubit gate can be performed between the two qubits, subsequently the qubits need to be replaced in the correct order.

In doing so, depth and noisy operations are added to the circuit. This has an overall negative effect on the feasibility of running the circuit and the reliability of the result.

In Figure 1.1 we present an example of the connectivity of a current quantum computer. In Figure 1.2 we show how applying a CNOT gate between qubits q0 and q4, on this quantum computer, would have to be decomposed on the hardware.<sup>1</sup>

Figure 1.1: Layout of IBM Lima quantum computer [IBM], the lines the physical qubits are connected.

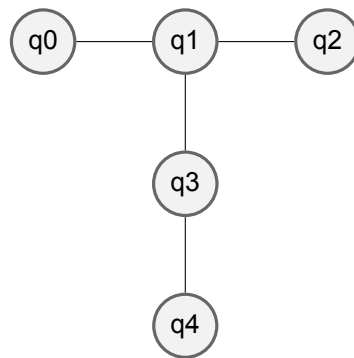
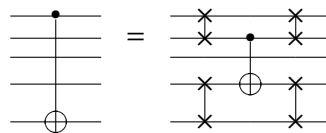


Figure 1.2: Decomposition to apply a CNOT gate between q0 and q4 on the IBM Lima quantum computer.



## 1.3. Possible approaches and earlier research

The aim of the project is to find quantum circuits tailored to the properties of specific NISQ devices, in order to make optimal use of current quantum technologies. There are several possible approaches to be considered. The approaches can be divided into three main categories. The first category is the ansatz-centric approach, in which a predetermined circuit structure is used regardless of problem and hardware at hand.

The second category is the so-called circuit-centric approach. The circuit-centric approach starts from a known textbook quantum algorithm to solve a specific problem. This known textbook algorithm is subsequently altered and compiled in such a way that the chosen quantum hardware is optimally suited to run the resulting quantum circuit.

The third approach is the hardware-centric approach, in which we start from the requirements that the chosen quantum hardware puts on the possible circuits. Subsequently a quantum circuit is built that optimally suits the hardware. This circuit is then possibly altered to be able to mimic the behaviour of

<sup>1</sup>See Appendix C.4 for an overview of symbols used when representing a quantum circuit by a figure.



the desired function.

In the sections below we will further elaborate on the possible approaches and relevant research that has been conducted in those areas.

### 1.3.1. Ansatz-centric ansatz

This approach is most common when a quantum layer is used in a larger machine learning scheme to exploit the probabilistic nature of its measurement outcomes. In this approach, a predetermined structure of parameterized and non-parameterized gates is used, the parameters of the parameterized gates are subsequently optimized to mimic the desired behaviour [LW18]. Usually the chosen structure of the quantum gates consists of layers, where entanglement layers are alternated with rotation layers. The rotation layer generally consists of three separate rotation gates to allow arbitrary rotation for each qubit; see Appendix A.1.1. A complete entanglement layer usually consists of a CNOT gate between all possible qubits. Since such layers add a lot of extra depth to the system and CNOT gates of current quantum computers are known to be rather noisy [Dav+19], some extra steps need to be taken to make these layers more sparse. One method that could be used is to sparsen the structure depending on the estimated relation between qubit states, estimated by the desired distributions and inputs [CC68] [LW18]. Note that this method gives no indication on the direction of the CNOT gates in the found gate structure.

### 1.3.2. Circuit-centric ansatz

The second approach for finding quantum circuits to mimic the behaviour of a known quantum circuit, is to start from the known quantum circuit. This circuit is then decomposed into native quantum gates and subsequently some optimization scheme is used to reduce the depth of the created circuit. In doing this, we allow the circuit to be altered in a way that does not add up to identity. In this case, the overall outcome of the quantum state will be slightly altered in return for a decrease in depth when running the circuit on the quantum hardware.

A downside of this approach is that it can only be used to simulate functions for which a quantum circuit is known. Another downside is that with this approach the outcome depth of the found quantum circuit cannot be directly controlled and is more heavily reliant on the input depth of the known quantum circuit. A benefit of this approach is that it can always be implemented in such a way that a quantum circuit mimicking the behaviour of a target algorithm is found up to a chosen error margin.

### 1.3.3. Hardware-centric ansatz

Another possible approach for finding quantum circuits to estimate a function, is to start from the hardware restrictions and build the quantum circuit from scratch. The idea is to take the hardware restrictions and from that create an ansatz consisting only of gates and operations that can natively be applied to the chosen hardware. This ensures that no unforeseen depth or qubits are added when the circuit is run on the hardware. Because of this it is possible to always choose to find a circuit which can be reliably run on the chosen quantum computer. Another benefit of this approach, is that it can be used to try to mimic functions for which no quantum algorithms are known. A downside of this approach is that it can not be guaranteed that a quantum circuit of the chosen depth which can mimic the chosen function even exists [NC16], let alone that it will be found. There are still open question regarding the complexity of quantum circuit compilation for certain subproblems, but the problem is generally NP-hard [BKM18] [Ale21].

The main challenge of this approach is to find a suitable circuit ansatz. One way to approach this is to start from a known depth  $D$  and build up the ansatz from layers of entanglement followed by layers of rotation; this approach is taken in [Ben+19a] [Ben+19b]. A downside would be that to properly implement the entanglement and rotation layers would require quite a lot of depth per layer. Specifically, the implementation of the entanglement layers could add a lot of depth to the circuit when applied to quantum hardware for which the qubits are sparsely connected, as many extra swap operations would be required to even perform a two-qubit operation between most of the qubits. Since complete entanglement layers on a sparsely connected quantum computer is not realistic, this method is best used on completely connected qubit architecture structures [Ben+19a]. Another implementation of this approach, could be to simply restrict the entanglement layers to act on connected qubits [Zhu+19]. An approach as described in [LW18] [Ben+19b] making use of a Chow-Liu graph [CC68] to reduce entanglement gates could also be considered. All three approaches still lead to potentially quite some

unnecessary two-qubit gates adding noise to the system. On top of that, each rotation layer needs to contain three gates per qubit. Even though single qubit operations are decidedly cheaper, this still adds unnecessary depth to the circuit.

One could also consider building up a well performing quantum circuit by repeatedly adding relatively thin, but structured, layers to the quantum circuit [Dav+19]. A downside of this approach is that it is not guaranteed that adding a layer which at that point results in a minimal cost as compared to other options, is also the layer that leads to the best possible circuit as it grows. In fact, many known quantum algorithms work by first spreading out the amplitudes of the quantum states over the space and then exploiting certain structures to obtain the right result. As the complexity of the problem increases, it is increasingly difficult to gauge the goodness of a very thin layer.

Another approach would be to keep the chosen ansatz as flexible as possible. This could be done by randomly creating ansatzes that respect the hardware restrictions [Cin+20] [Kha+19]. Subsequently, a promising ansatz can be filtered out by running a shallow optimization scheme on the created ansatzes and picking out the one that performs best. The best performing ansatz can then be further optimized. Notice that the amount of possible ansatzes adhering to the hardware restrictions quickly becomes intractable as the amount of qubits and circuit depth grows. Furthermore gradient-based machine learning methods are known to run into trouble as the size of quantum circuits grow [McC+18]. Nonetheless, this approach is good when restricted to smaller problem sizes. When expanding to problems requiring a larger circuit size, this method can serve as a basis that could be extended by adding a discrete optimization layer [Cin+20] [Kha+19] or gradient descent with momentum (GDM) to combat the potential scaling problems mentioned before [Ben+19a].

In this project, we use the approach in which we allow for most flexibility in circuit design and control over circuit depth, by creating random ansatzes suited for hardware restrictions and optimizing the associated parameters to mimic a chosen function. We find a good ansatz by creating multiple random ansatzes and first running a shallow and cheaper optimization scheme, to identify the best ansatz and subsequently optimize that further. This serves as a proof of concept for the parameter optimization approach and cost function chosen in combination with a hardware-centric ansatz approach. The approach described can be extended upon as the problem size grows.

Our approach differs from [Kha+19] [Cin+20] as they restrict their learning to find circuits which create exactly the same quantum state as the original circuit. We only require to find a circuit that creates quantum states that have the same probability amplitudes as the desired states, this leads to more flexibility and therefore potentially thinner quantum circuits. Since upon measurement in the chosen basis we can only find one basis state, our approach does not lose any valuable information for most use cases.

It is also important to point out that many of the approaches mentioned before treat the software native gates of a quantum computer as a building block to learn [Kha+19] [Dav+19] [LW18] [Cin+20]. These gates are actually build up using several different native gates implemented as pulses on the hardware [Gok+20] [Ale+20]. In our approach we optimize using the matrix representation of the pulses in an effort to find the thinnest possible circuit.



# 2

## Quantum hardware-centric circuit

In this section, we further introduce and specify the problem of finding a suitable hardware-centric quantum circuit. We do this by first defining the precise hardware restrictions. Subsequently the mathematical formulation of the problem and its restrictions are given. Finally, we present the cost function used to evaluate and compare cost of found quantum circuits.

### 2.1. Quantum hardware

This section presents further information on current quantum hardware and its restrictions. We first elaborate on NISQ computers and their qualities. After, we discuss the specific properties of the quantum computers that are available for public use at the time of writing.

#### 2.1.1. NISQ devices and quantum volume

Quantum technology is entering the stage of the NISQ devices. These are quantum computers with about 50 to 100 qubits that can execute limited depth quantum circuits [Cin+20] [Pre18]. Since NISQ devices are limited in the amount of qubits they have at their disposal and the maximum circuit depth, IBM designed a metric known as quantum volume which expresses the power of a quantum computer by these factors [Mol+18].

Quantum volume  $V_Q$  expresses the maximum size of a circuit that can be successfully run on a quantum computer. The metric directly depends on the amount of qubits  $n$  available in the computer and the associated maximum depth of the circuit  $d(n)$ . The maximum depth of the circuit is decided by the gate fidelity and coherence time. Quantum algorithms need a certain amount of qubits and circuit depth in order to be implemented. Furthermore, there is a trade-off between the two, where adding extra qubits can be used to reduce the depth of a circuit [MN01]. These properties make quantum volume a natural metric to compare the power of NISQ devices [Mol+18]. Quantum volume can be expressed by the following equation:

$$V_Q = \min[n, d(n)]^2. \quad (2.1)$$

The metric considers the square of the smallest value of  $d(n)$  and  $n$  to determine the quantum volume  $V_Q$ , since this forces high scoring quantum computers to have both good circuit depth and a good maximum amount of qubits. Another expression for the quantum volume is

$$\tilde{V}_Q = \max_{\tilde{n} < n} \left( \min[\tilde{n}, \frac{1}{\tilde{n}\epsilon_{\text{eff}}(\tilde{n})}]^2 \right). \quad (2.2)$$

The value  $\tilde{V}_Q$  expresses that it can sometimes be beneficial for the quantum volume to only run on a subset of the qubits available on the computer. In the equation above  $\epsilon_{\text{eff}}$  is the effective error rate. This effective error rate expresses the error when applying a two-qubit gate on any two qubits in the hardware, taking into account hardware connectivity, native gates and possible parallelism. This means that if all qubits are connected with the two-qubit gate natively implemented, we have  $\epsilon = \epsilon_{\text{eff}}$ , otherwise  $\epsilon < \epsilon_{\text{eff}}$ , where  $\epsilon$  of course expresses the gate infidelity when applying a quantum gate to two connected

qubits. If a quantum computer is a linear chain of qubits, we have  $\epsilon_{\text{eff}} \propto \tilde{n}\epsilon$ . This implies that the metric takes into account that on the hardware we do not have all-to-all connectivity and we have a limited native gate set.

Since our approach restricts the quantum gates applied, to gates that can be natively applied to the hardware, we have that  $\epsilon_{\text{eff}} = \epsilon$  for all our found circuits. This implies that when we only consider the valid ansatzes, the effective quantum volume  $\tilde{V}_Q$  could be higher.

### 2.1.2. IBM quantum experience

In this section we will give a small overview of the quantum computers IBM currently has available and their near-term goals. We do this to give the some more insight in the state of current quantum computers.

At the time of writing this thesis, IBM shows 20 different quantum computers available on their website [IBMa]. The largest available quantum computer in terms of number of qubits is the *ibmq\_manhattan* with 65 qubits and a quantum volume of 32. The quantum computer with the largest available quantum volume is the *ibmq\_montreal*, which has a quantum volume of 128. This means that we can at least reliably run a circuit with up to 11 qubits of depth 11.

IBM has released a quantum roadmap in which they lay-out their predictions for the size and processor types of the quantum computers they will release in the coming years [IBMc]. In the quantum roadmap IBM predicts to have a functioning quantum computer with 127 qubits become available in 2021. By 2022 they claim to have a 433 qubit counting quantum processor functioning. Furthermore, they predict to have a 1.121 qubit processor available by 2023. From 2023 onwards, they plan to scale up to a million qubits, but no precise roadmap is given for this process.

The largest quantum computer IBM currently offers for free use is the *ibmq\_16\_melbourne*, which consists of 15 qubits and has a quantum volume of 8. The IBM computer available for free use with the largest quantum volume is the *ibmq\_santiago*, which has a quantum volume of 32 with 5 qubits.

The IBM qubits have two natively implemented single-qubit gates and one native two-qubit gate [Mur+19]. On the hardware these gates are implemented as pulses [Ale+20]. The natively implemented single-qubit gates are rotations around the x- and z-axis; see Appendix A.1.1

$$R_X\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix}, \quad (2.3)$$

$$R_Z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}. \quad (2.4)$$

Note that the rotations around the x- and z-axis are generated by the Pauli-spin matrices  $\sigma_z$  and  $\sigma_x$ . We have  $R_Z(\theta) = e^{-i\frac{\theta}{2}\sigma_z}$  and  $R_X\left(\frac{\pi}{2}\right) = e^{-i\frac{\pi}{4}\sigma_x}$ . Where the Pauli-spin matrices have the following definitions

$$\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad (2.5)$$

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \quad (2.6)$$

The IBM two-qubit gates are given rise to by the cross-resonance (CR) pulse [Mur+19], which implements a ZX interaction [IBMb] C. The  $R_{ZX}(\theta)$  gate is generated by the  $X \otimes Z$  matrix,  $R_{ZX}(\theta) =$

$$e^{-i\frac{\theta}{2}X\otimes Z}$$

$$R_{ZX}(\theta) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & 0 & -i\sin\left(\frac{\theta}{2}\right) & 0 \\ 0 & \cos\left(\frac{\theta}{2}\right) & 0 & i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & 0 & \cos\left(\frac{\theta}{2}\right) & 0 \\ 0 & i\sin\left(\frac{\theta}{2}\right) & 0 & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}. \quad (2.7)$$

Similarly the  $R_{XZ}(\theta)$  gate can be applied which is generated by the  $Z \otimes X$  matrix

$$R_{XZ}(\theta) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) & 0 & 0 \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) & 0 & 0 \\ 0 & 0 & \cos\left(\frac{\theta}{2}\right) & i\sin\left(\frac{\theta}{2}\right) \\ 0 & 0 & i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}. \quad (2.8)$$

Note that the native gates that we consider for this project are the gates that are implemented by the pulses used to control the qubits on the hardware. IBM has other software native gates visible, but those are ultimately decomposed in terms of pulses described by our chosen matrices [Gok+20] [Mur+19]. Each IBM quantum computer also has its own qubit connectivity, for an overview of the connectivity of the IBM quantum computers considered for this project see Appendix D.

### 2.1.3. Rigetti's Aspen quantum computers

Rigetti is another company currently providing sizeable quantum computers, we will also give a small overview of the computers they have available.

The Rigetti Aspen 8 is available for use via Amazon Web Services and it provides a 32 qubit quantum computer. Other than IBM, the Rigetti computers have four natively implemented quantum gates

$$R_X\left(\pm\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & \pm i \\ \pm i & 1 \end{bmatrix}, \quad (2.9)$$

$$R_Z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}, \quad (2.10)$$

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad (2.11)$$

$$R_{XY}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\left(\frac{\theta}{2}\right) & i\sin\left(\frac{\theta}{2}\right) & 0 \\ 0 & i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.12)$$

Rigetti added the fourth gate,  $R_{XY}(\theta)$ , in 2019 to limit the circuit depth after gate decomposition [Abr+20]. Before that they only had the  $R_X\left(\pm\frac{\pi}{2}\right)$ ,  $R_Z(\theta)$  and  $CZ$  gates natively implemented [Mur+19]. In Appendix D we have listed the connectivity structure of the Rigetti computers considered in this project.

## 2.2. Valid quantum ansatz

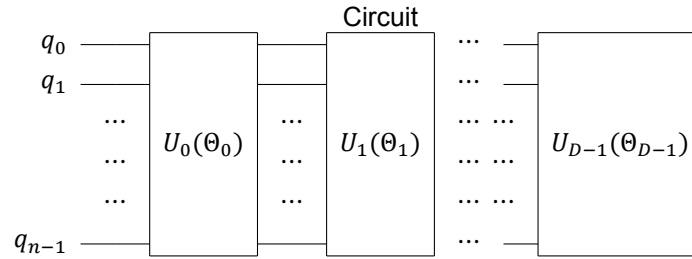
In order to make optimal use of quantum computers, our approach finds quantum circuits which take the properties of the chosen backend into consideration.

Concretely, this means that all the gates used in the found quantum circuit will be gates that are native to the chosen backend. Because of this, the gates will no longer need to be decomposed when running the circuit. This means that the depth of the circuit found, will be the actual depth of the circuit ran on the computer, thereby avoiding unforeseen circuit depth.

Another important aspect is that our approach takes into account which qubits are actually connected in the backend. The found quantum circuit will only apply two-qubit gates between qubits which are physically connected on the hardware.

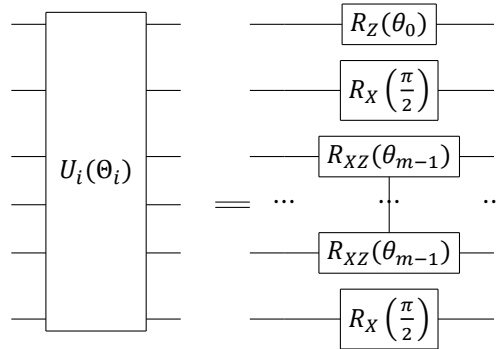
A valid ansatz represents the lay-out of a quantum circuit, such that the hardware connectivity and native gates are taken into account. Our approach allows to create a valid ansatz  $U$  for any of the implemented quantum computers with specified circuit depth  $D$  and number of qubits used  $n$ . In Figure 2.1 we give a drawing of a quantum circuit which consists of a valid ansatz  $U$  with some parameter  $\Theta$  as input for the parameterized gates. In this project we will sometimes refer to a circuit generated by a valid ansatz as a valid ansatz.

Figure 2.1: Quantum circuit representing the found valid ansatz of which the unitary operations are built up using only natively implemented quantum gates.



A valid ansatz  $U$  of  $D$  layers for  $n$  qubits can be used to express a quantum circuit  $U(\Theta)$  consisting of  $D$  layers  $U_i(\theta_i)$  successively applied to the  $n$  qubits. These unitary operations are built up of natively implemented quantum gates for the chosen hardware. Hence, the dimension of the parameter  $\theta_i$  depends on the amount of parameterized gates  $m$  used in the layer  $U_i$ :  $\theta_i \in [0, 2\pi]^m \subset \mathbb{R}^m$ , where  $m \leq n$  since no more than  $n$  qubits have a parametrized gate applied to it in a layer. Figure 2.2 gives the circuit drawing of a possible decomposition of a layer  $U_i(\theta_i)$  of a valid ansatz.

Figure 2.2: An example of a valid  $U_i$  for an IBM quantum computer, where all the gates used are natively implemented on the specified hardware.



### 2.3. Problem description

In the previous section we have defined what constitutes a valid ansatz for a given quantum computer. In this section we will formulate the optimization problem of finding a quantum circuit to mimic a certain operation, where such a quantum circuit is a valid ansatz with the optimal gate parameters  $\Theta$  to mimic the chosen operation. In our approach, the depth of the circuit  $D$  and number of qubits  $n$  are determined beforehand and so the quantum volume is to not exceed its maximum.

The problem of finding the best ansatz to simulate a quantum circuit with a given depth is a mixed integer problem with nonlinear objective. We need to optimize over which of the quantum gates native to the chosen computer is used in each position, as well as over the angles of the parameterized gates. Let  $n$  be the number of qubits of the quantum computer and  $\mathcal{N} := \{0, \dots, n-1\}$ . Let  $D$  represent the depth of the quantum circuit and  $\mathcal{D} := \{0, \dots, D-1\}$ . Let  $\mathcal{G}$  be the set of gate operations native to the computer hardware and  $\mathcal{G}' \subset \mathcal{G}$  the set of all native two-qubit gates. Let  $\mathcal{P}$  be the set of all directed qubit pairs connected on the chosen hardware, we have  $|\mathcal{P}| \leq n(n-1)$ .

Define the variables  $x_{ijk}$ , where  $i \in \mathcal{G}$ ,  $j \in \mathcal{N}$ ,  $k \in \mathcal{D}$  and  $x_{ijk} \in \{0, 1\}$ . We define  $x_{ijk} = 1$  if qubit  $j$  has gate type  $i$  applied to it in layer  $k$ ,  $x_{ijk} = 0$  otherwise.

Another variable will be  $y_{ilk}$ , where  $i \in \mathcal{G}'$ ,  $l \in \mathcal{P}$ ,  $k \in \mathcal{D}$ . Where  $y_{ilk} = 1$  if for  $l = \{p, q\}$  qubit  $p$  and qubit  $q$  share a the two-qubit gate  $i$  in layer  $k$ , and  $y_{ilk} = 0$  otherwise. The problem of finding a valid ansatz of minimal cost with given depth, which mimics a known quantum circuit, can be expressed by the following restrictions

$$\text{Minimize } c(x, y, \theta) \quad (2.13)$$

subject to

$$\sum_{i \in \mathcal{G}} x_{ijk} \leq 1 \quad \forall j \in \mathcal{N}, \forall k \in \mathcal{D} \quad (2.14)$$

$$\sum_{l: j \in l, l \in \mathcal{P}} y_{ilk} = x_{ijk} \quad \forall i \in \mathcal{G}', \forall j \in \mathcal{N}, \forall k \in \mathcal{D} \quad (2.15)$$

$$x_{ija}, y_{ilk} \in \{0, 1\} \quad \theta \in [0, 2\pi]^n \quad \forall i \in \mathcal{G}, \forall j \in \mathcal{N}, \forall k \in \mathcal{N}, \forall d \in \mathcal{D}. \quad (2.16)$$

It is now left to define the cost function (2.13). Finding a good cost function in this case is non-trivial and the choice of cost function is further discussed in Section 2.4. We will first argue the correctness of the formulation.

### 2.3.1. Correctness of formulation

In order to show that this formulation is correct, we show that any point in the space spanned by this formulation is a valid ansatz of a quantum circuit. Secondly, we show that any valid ansatz of a quantum circuit obeys the following restrictions, and therefore is in the space spanned by the formulation. First, we will show that any point which satisfies Equations (2.14) to (2.16) must represent a valid quantum circuit.

By Equation (2.14) we have that each qubit has at most one native quantum gate assigned to it in each layer.

By Equation (2.15) we have that in each layer, if a qubit  $j$  has a two-qubit gate  $i \in \mathcal{G}'$  assigned to it, then there exists exactly one ordered pair  $l$  s.t.  $l = (j, m)$  or  $l = (m, j)$  for which  $y_{ilk} = 1$ . This means that  $j$  shares the two-qubit  $i$  gate with precisely one other qubit  $m \in l$  in the layer  $k$ . Then, by Equation (2.15) this other qubit must have  $x_{imk} = 1$ , since  $y_{ilk} = 1$ . And so, the other qubit must have the same two-qubit gate applied to it in the same layer. Then, also by Equation (2.15), this qubit has exactly one  $l' \in \mathcal{P}$  s.t.  $m \in l'$  and  $y_{il'k} = 1$ . Since  $y_{ilk} = 1$  and  $m \in l$  we must have  $y_{il'k} = y_{ilk}$ . And so, we have that if a two-qubit gate is applied to a qubit in a layer, it has exactly one qubit with which it shares this two-qubit gate in the layer, this pair is ordered and its qubits are connected on the hardware. In Equation (2.16) we make sure that  $x_{ija}$  and  $y_{jka}$  take binary values, where the angles of the parametrized gates  $\theta_i$  can take any value between  $[0, 2\pi]$ .

To summarize we make sure that each qubit is assigned at most one gate operation in each layer. We also make sure that if a qubit is assigned a two-qubit gate operation, there is precisely one qubit with which it forms an ordered pair which also has this two-qubit gate operation assigned to it in the layer and the pair of qubits is connected on the hardware. Since the qubits form an ordered pair, it is also clear which qubit takes which position in the two-qubit operation. Therefore any combination  $x$  and  $y$  which satisfies the above requirements also describes a valid quantum circuit obeying the hardware of a chosen backend.

Next we will show that any valid quantum circuit satisfies the restrictions described.

Take  $x$  and  $y$ , which describe a valid quantum circuit, and show that they obey the restrictions (2.14) to (2.16). Since this is a valid quantum circuit, we have that each qubit has at most one gate applied



to it in each layer. Therefore, Equation (2.14) must be satisfied by  $x$ .

In a valid quantum circuit it must be so that if a qubit  $j$  has a two-qubit gate  $i$  applied to it, there exists another qubit  $m$  with which it shares this two-qubit gate in the layer, the two must be connected on the hardware and they both have a different position in the operation. Let  $j$  be the qubit in the first position of the two-qubit operation and  $m$  the qubit in the second position and let  $l = (j, m)$ , then clearly we have  $l \in P$ . Since for our valid ansatz we have that  $j$  and  $m$  share the two qubit gate  $i$  in the layer  $k$  with  $j$  in the first position and  $m$  in the second, we have  $x_{ijk} = 1$ ,  $x_{imk} = 1$ ,  $y_{ilk} = 1$  and  $y_{il'k} = 0 \forall l' \neq l$  and so Equation (2.15) is satisfied.

In a valid quantum system it must be that either a gate is applied or not applied, and either two qubits are paired or are not paired. This is clearly a binary system and therefore Equation (2.16) must also be satisfied. From this we can conclude that any valid quantum circuit obeying the hardware of a chosen backend satisfies the requirements given in the problem description.

We have shown that any combination of vectors  $x$  and  $y$  that satisfies the Equations (2.14), (2.15), and (2.16) also describes a valid quantum circuit obeying the hardware of a chosen backend and vice versa. Therefore, we have shown that the given description of the problem is correct.

### 2.3.2. Problem size

The vector  $x$  consisting of the  $x_{ija}$ 's has length  $|G|nD$  which in our case can be capped by  $|G|nD \leq 4nD$ , as we will not have more than 4 native gates in the quantum computers we are considering. So the magnitude of this vector is of order  $\mathcal{O}(nD)$ . This simplification generally hold as quantum computers have very few native gates.  $y$  has length  $|G||P|D \leq 4n(n-1)D$ . This is of order  $\mathcal{O}(n^2D)$ . The described problem consists of  $\mathcal{O}(nD)$  restrictions.

## 2.4. Cost of the ansatz

It is not directly clear how to define the cost function  $c$  for this problem. There are several aspects of a potential cost function that would make it difficult to deal with.

First of all, due to the nature of the problem, the cost function will be nonlinear with respect to the input variables, therefore we can not use a standard linear programming approach to find  $x$ ,  $y$  and  $\theta$  to minimize it.

The other challenging aspect of any potential cost function, is that it would scale exponentially with the amount of qubits  $n$  in the system, due to the fact that we are working in a  $2^n$  dimensional Hilbert space. The naive choice of cost function would be to find the distance between the matrix expressing the quantum circuit found by our machine learning approach and the matrix expressing the aimed for unitary operation. This approach assumes that the matrix of the desired unitary operation is known and exists, which limits our use case. More importantly it requires the distance to be determined between two  $2^n \times 2^n$  sized matrices. This operation is too computationally costly to perform. As a first step we would need to find the matrix expression of the learned circuit, using approximately  $D \times n$  Kronecker products followed by  $D$  matrix multiplications; see Appendix A.2.1. Subsequently, we would need to calculate the distance between two  $2^n \times 2^n$  matrices. To avoid these difficulties, we use another cost function to evaluate the goodness of fit of the found quantum circuit compared to the aimed for operation.

### 2.4.1. Cost estimation function

Since evaluating the cost of the circuit created by comparing the distance to the aimed for unitary operation is too costly, we use a different method to estimate the cost. Let  $f : \mathbb{C}^{2^n} \rightarrow \mathbb{R}^{2^n}$  be the function for which we wish to find a quantum circuit to mimic the behaviour of this function on the chosen quantum hardware. The input to this function  $f$  is a prepared quantum state  $|\phi\rangle$ . The output of the function  $f(|\phi\rangle)$  is the probability vector representing the probabilities of finding each basis state after measurement in the computational basis. Let  $f_A(|\phi\rangle)$  be the function that describes the resulting probability vector of applying a known quantum circuit  $A$  to the input qubit state  $|\phi\rangle$  and subsequently measuring the result in the computational basis. Then, the vector  $f_A(|\phi\rangle) \in \mathbb{R}^{2^n}$  represents the resulting probability vector describing the probabilities of measuring each of the  $2^n$  basis states after applying the quantum circuit  $A$  to the state  $|\phi\rangle$

$$f_A(|\phi\rangle) = (|\langle i | A | \phi \rangle|^2)_{i=0, \dots, 2^n-1}. \quad (2.17)$$

In this project we will write  $f(|\phi\rangle)$  and  $f_A(|\phi\rangle)$  interchangeably when the applied quantum circuit is clear from context. Note that a quantum measurement in the computational basis always results in one specific basis state. Therefore, it is impossible to find this value  $f(|\phi\rangle)$  directly on a quantum computer. We can estimate the value of  $f(|\phi\rangle)$  by running the circuit multiple times and taking the average of the results. When using our quantum simulator, as described in Chapter 3, we can access these values directly by creating the resulting vector  $A|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$  and turning the amplitudes  $\alpha_i$  of the different basis states  $|i\rangle$  into the probability of measuring the state  $|i\rangle$  namely  $|\alpha_i|^2$ .

Let  $U(\theta)$  be the learned quantum circuit, where  $U$  represents the chosen ansatz represented by  $x$  and  $y$  in Equation (2.13), with depth  $D$  and  $n$  qubits and  $\theta$  represent the angles of the parameterized quantum gates in the circuit. Define  $f_{U(\theta)}$  to be the function which maps the quantum states created with  $U(\theta)$  to its associated expectation values

$$f_{U(\theta)}(|\phi\rangle) = (| \langle i | U(\theta) | \phi \rangle |^2)_{i=0, \dots, 2^n-1}. \quad (2.18)$$

In this project, we estimate the cost of the created circuit  $U(\theta)$  by evaluating the distance between  $f_{U(\theta)}$  and  $f$  for several input states  $|\phi\rangle$ , where  $f$  represents the function which takes the input quantum state to the desired probability vector. Let  $Y = f(B)$  with  $y \in Y$  such that  $y \in \mathbb{R}^{2^n}$ . Let  $B$  be a chosen batch of quantum states to evaluate the cost over. Then, we can define

$$c_{U(\theta), f}(B) = \frac{1}{|B|} \sum_{|\phi\rangle \in B} \|f(|\phi\rangle) - f_{U(\theta)}(|\phi\rangle)\|_2 \quad (2.19)$$

as our cost function. Similarly we could write

$$c_{U(\theta)}(Y, B) = \frac{1}{|B|} \sum_{i=0}^{|B|-1} \|y_i - f_{U(\theta)}(b_i)\|_2. \quad (2.20)$$

Above we have written the cost function as a function of the input quantum states  $B$  and the associated optimal output values. This makes sense when considering the meaning of the chosen cost function. As we will use this cost function in the context of optimizing the parameters  $\theta$  of a quantum circuit  $U(\theta)$ , we can also view this cost function as a function of  $\theta$  with  $U$  and  $B$  (and  $Y$ ) as fixed parameters

$$c_{U, B}(\theta) = \frac{1}{|B|} \sum_{|\phi\rangle \in B} \|f(|\phi\rangle) - f_{U(\theta)}(|\phi\rangle)\|_2. \quad (2.21)$$

It is important to note here that this cost function is defined over the resulting probability vectors of having evaluated the quantum circuit on certain input states, we do not measure the distance between the actual resulting quantum states.

This is a sensible choice since quantum computing does not allow you to recover the actual quantum states. Upon measurement of a qubit in a chosen basis it always collapses to a basis state with nonzero amplitude. Therefore the information that is not taken into account in our cost function, is information that is lost upon measurement. This makes our cost function suitable to use in cases where we want to learn a quantum circuit that performs a certain operation and is subsequently measured. This cost function is not suitable for purposes in which it is necessary to distinguish the actual quantum output state of having run the circuit.

A second important aspect of our cost function, is that its outcome value is not directly informative. The distance we use is not designed to express the likeliness of two probability vectors. Other functions, such as the Kullback-Leibler divergence, would result in a more informative cost function, but are more computationally costly [NC16].

Notice that the result of the cost function depends on the batch chosen to learn over. Generally a larger batch will result in a more reliable cost function. However, each probability vector resulting from running a circuit on an input quantum state, is exponential in size with respect to the amount of qubits. This implies a trade-off between computational speed and reliability of the cost function.

Another important aspect of the cost function is that it is defined over the distance between the resulting measurement and the aimed for measurement, which means that non-unitary and non-linear operations have taken place on the quantum states before the cost is determined. This means that we cannot

choose a basis which spans the space and conclude that if our ansatz performs well for these basis states, it will perform well on all linear combinations of these states. To see this, consider the example of the Hadamard gate. We have that

$$H |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad (2.22)$$

and

$$H |1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \quad (2.23)$$

Then, by linearity of the quantum operation we know that

$$H \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |0\rangle. \quad (2.24)$$

Define the unitary operation  $H'$  to do the following on the two basis states

$$H' |0\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{i}{\sqrt{2}} |1\rangle \quad (2.25)$$

and

$$H' |1\rangle = \frac{-i}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle. \quad (2.26)$$

It can easily be checked that  $H'$  is a unitary operation and therefore a valid quantum circuit. Now consider

$$H' \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \left(\frac{1}{2} - \frac{-i}{2}\right) |0\rangle + \left(\frac{i}{2} + \frac{1}{2}\right) |1\rangle. \quad (2.27)$$

Now, clearly  $C_{H',H}(|0\rangle) = C_{H',H}(|1\rangle) = 0 \neq C_{H',H}\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)$ . It is important to take this into account when determining the batch to let the quantum circuit learn over.

### 2.4.2. Cost estimation and noise

Note that when we use the chosen physical quantum computer to run our machine learning approach, we directly take noise into account with our cost estimation function. As noise influences the measurement outcomes, the estimated expectation values we find will actually have noise incorporated in them, as they are the exact expectation values of running the found circuit on the chosen quantum computer. When using our approach with a quantum simulator, noise is not taken into account.

Both our machine learning approaches are designed in such a way that they can be used on a physical quantum computer. Scaling suggests that using a physical quantum computer becomes unavoidable as the Quantum Volume of NISQ devices increases. In this way our optimization approach naturally takes noise into consideration without explicitly implementing this in the chosen cost function.

# 3

## Kronx quantum circuit simulator

To learn over the performance of quantum circuits, we need to be able to simulate the effect a quantum circuit has on the state of a quantum system. For this purpose, we wrote our own quantum simulator that takes quantum states and operations expressed as PyTorch tensors as input, and outputs the resulting quantum state.

The foundation of our Kronx quantum circuit simulator is the so-called Algorithm 993 [Fac19], which enables the user to efficiently find vector states resulting from a matrix operation that was built up of Kronecker products. The Algorithm 993 [Fac19] as presented is designed for programming languages which store vectors in a column-major form. As PyTorch stores matrices in row-major format we altered the Algorithm 993 for this case. Furthermore we wrote our own extension of the algorithm to enable two-qubit operations to be performed on two qubits which are non-adjacent in number, but connected on the hardware; see Section 3.2.

### 3.1. Algorithm 993

The Algorithm 993 [Fac19] is designed to efficiently perform matrix-vector multiplication, where the matrix is the Kronecker product of smaller matrices, without assembling the full matrix explicitly. Algorithm 993 avoids calculating the Kronecker products and reshapes the vector to apply the original matrix operations one-by-one.

As an example let  $A = A_1 \otimes A_2 \otimes A_3$ , with  $A_1 \in \mathbb{K}^{2 \times 2}$ ,  $A_2 \in \mathbb{K}^{4 \times 4}$  and  $A_3 \in \mathbb{K}^{2 \times 2}$ . The worked-out Kronecker product yields  $A \in \mathbb{K}^{16 \times 16}$ , which requires the storage of  $16^2 = 256$  matrix entries and the application to a target vector of size 16. Algorithm 993 only requires  $2^2 + 4^2 + 2^2 = 24$  matrix entries, which is less than 10% of the fully worked-out matrix. In summary, Algorithm 993 is particularly helpful to reduce the memory consumption.

This algorithm is useful in our setting as the application of a quantum circuit can be expressed as a matrix vector multiplication. Let  $U$  be the quantum circuit we wish to apply. The circuit  $U$  has depth  $D$ , so  $U = U_0 U_1 \dots U_{D-1}$ , where the  $U_i$  are the layers of the circuit. Each layer  $U_i$  of  $U$  is the Kronecker product of the matrix expressions of the quantum gates applied to each qubit in the layer  $U_i = u_i^0 \otimes \dots \otimes u_i^{n-1}$ . For each of these layers  $U_i$  we make use of the Algorithm 993 to create the result of applying  $U_i$  to a quantum state, without explicitly calculating  $U_i$  by performing the Kronecker products. By doing this we make use of the Algorithm 993 [Fac19] to avoid the costly Kronecker products and memory storage associated with storing the  $2^n \times 2^n$  matrix  $U_i$ .

Notice that this algorithm is particularly well suited for our use case, since we only implement quantum circuits consisting of native gates. This is due to the fact that native gates are at most two-qubit gates, and so we potentially save a lot of computational steps and memory, in the form of calculating and saving the Kronecker products.

In the following section, we will show how this algorithm enables the simulation of a single-qubit gate on the least significant qubit. We will then show how the index order of the resulting state vector is such that a virtual qubit shift has appeared, which enables us to directly perform the simulation of a quantum

gate on what was originally the second-to-last qubit. We use this method to virtually apply a quantum gate to all the qubits in the system, after which the virtual shifts have brought all the qubits back to the original position. In the following section we argue how to extend this to be able to apply single- and two-qubit quantum gates on all possible qubit combinations.

Consider an  $n$  qubit state  $|\phi\rangle$ , we wish to perform a layer of a quantum circuit  $U_i = u_0 \otimes u_1 \otimes \dots \otimes u_m$  on this qubit state such that we end up in the state

$$|\psi\rangle = U_i |\phi\rangle. \quad (3.1)$$

Applying the quantum circuit to the state  $|\phi\rangle$  is the same as first creating a state

$$|\psi'\rangle = I \otimes \dots \otimes I \otimes u_m |\phi\rangle, \quad (3.2)$$

followed by

$$|\psi''\rangle = I \otimes \dots \otimes I \otimes u_{m-1} \otimes I |\psi'\rangle, \quad (3.3)$$

etcetera. We will now show how to use the Algorithm 993 [Fac19] to create the state  $|\psi\rangle$  by first creating a state  $|\xi'\rangle$  that is equal to  $|\psi'\rangle$  up to a shift in qubit order, which can then be used to create  $|\xi\rangle$  which is equal to  $|\psi''\rangle$  up to a shift in qubit order until we end up in the state  $|\psi\rangle$ .

First notice that we can rewrite the state

$$|\phi\rangle = \sum_{i=0}^{2^n} \alpha_i |i\rangle = \sum_{i=0}^{2^{n-1}} |i\rangle \otimes (\alpha_{2i} |0\rangle + \alpha_{2i+1} |1\rangle), \quad (3.4)$$

which essentially isolates the least significant qubit from the rest of the system. Define  $\beta_i$  to be such that

$$u_m \begin{bmatrix} \alpha_{2i} \\ \alpha_{2i+1} \end{bmatrix} = \begin{bmatrix} \beta_{2i} \\ \beta_{2i+1} \end{bmatrix}. \quad (3.5)$$

We can combine Equations (3.3), (3.4) and (3.5) to find an expression for  $|\psi'\rangle$ , which is

$$|\psi'\rangle = \sum_{i=0}^{2^{n-1}} |i\rangle \otimes u_m (\alpha_{2i} |0\rangle + \alpha_{2i+1} |1\rangle) = \sum_{i=0}^{2^{n-1}} |i\rangle \otimes (\beta_{2i} |0\rangle + \beta_{2i+1} |1\rangle). \quad (3.6)$$

Now that we have found an expression for  $|\psi'\rangle$ , we will show how we can create this quantum state as a PyTorch tensor up to a qubit shift by using the [Fac19] algorithm altered for row-major form vectors. Using the equality expressed in Equation (3.6) we can simulate this single qubit operation on the last qubit, by virtually reshaping the  $2^n$  state vector to a  $2 \times 2^{n-1}$  size matrix. First we isolate the last qubit as in Equation (3.4) by reshaping the  $2^n$  state tensor to a  $2^{n-1} \times 2$  and subsequently taking the transpose `phi = transpose(phi.reshape(2**(n-1), 2))`, these operations transform the vector representation of  $|\phi\rangle$  as follows

$$|\phi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^{n-1}-1} \end{bmatrix} \Rightarrow \begin{bmatrix} \alpha_0 & \alpha_2 & \dots & \alpha_{2^{n-2}} \\ \alpha_1 & \alpha_3 & \dots & \alpha_{2^n} \end{bmatrix}. \quad (3.7)$$

To see that doing this essentially isolates the last qubit for each possible basis state of the  $n - 1$  preceding qubits as in Equation (3.4) consider the following representation of Equation (3.7) in terms

of the quantum basis states associated with each index

$$\begin{bmatrix} |0 \dots 0\rangle |0\rangle \\ |0 \dots 0\rangle |1\rangle \\ |0 \dots 1\rangle |0\rangle \\ |0 \dots 1\rangle |1\rangle \\ \vdots \\ |1 \dots 1\rangle |0\rangle \\ |1 \dots 1\rangle |1\rangle \end{bmatrix} \rightarrow \begin{bmatrix} |0 \dots 0\rangle |0\rangle & |0 \dots 1\rangle |0\rangle & \dots & |1 \dots 1\rangle |0\rangle \\ |0 \dots 0\rangle |1\rangle & |0 \dots 1\rangle |1\rangle & \dots & |1 \dots 1\rangle |1\rangle \end{bmatrix}. \quad (3.8)$$

From the above expression it can be seen that the basis states represented in that column are such that only the last qubit state flips while all the other qubit states remain constant in each column. This means we are essentially isolating the last qubit and in each column, combining it with another basis state for the rest of the qubits, which is precisely what is expressed in Equation (3.4).

Performing matrix multiplication between

$$u_m = \begin{bmatrix} u_{m,11} & u_{m,12} \\ u_{m,21} & u_{m,22} \end{bmatrix} \quad (3.9)$$

and the result of applying operation (3.7) to the quantum state  $|\phi\rangle$  can now be performed by `phi=matmul(u_m, phi)`, which gives

$$\begin{bmatrix} u_{m,11} & u_{m,12} \\ u_{m,21} & u_{m,22} \end{bmatrix} \begin{bmatrix} \alpha_0 & \alpha_2 & \dots & \alpha_{2^n-2} \\ \alpha_1 & \alpha_3 & \dots & \alpha_{2^n-1} \end{bmatrix} = \begin{bmatrix} \beta_0 & \beta_2 & \dots & \beta_{2^n-2} \\ \beta_1 & \beta_3 & \dots & \beta_{2^n-1} \end{bmatrix}. \quad (3.10)$$

Above the  $\beta_i$ 's are defined precisely to be the resulting matrix entries from applying the matrix multiplication. The matrix is subsequently reshaped to vector form by `xi=phi.reshape(2**n)`, which results in the vector

$$\begin{bmatrix} \beta_0 \\ \beta_2 \\ \dots \\ \beta_{2^n-2} \\ \beta_1 \\ \beta_3 \\ \dots \\ \beta_{2^n-1} \end{bmatrix}. \quad (3.11)$$

The reason we get the entries of the vector in this order is that we are using a row-major format, note that this also means that this reshape operation does not cost any memory operations. Since Equation (3.11) expresses a quantum state we can call it  $|\xi'\rangle$ .

We now claim that the state  $|\xi'\rangle$  expresses a quantum state equal to  $|\phi'\rangle$  up to a difference in qubit order. In fact, we claim that  $|\xi'\rangle$  expresses a state precisely equal to the state  $|\phi'\rangle$  but with the last qubit of  $|\phi'\rangle$  being the first qubit of  $|\xi'\rangle$  in which all other qubits have shifted one index up.

We use this claim to explain how to simulate an entire layer of a quantum circuit. Since we can use the method described above to simulate a quantum operation acting on the last qubit and we then get the resulting quantum state with the last qubit virtually shifted to the first position and all the other qubits virtually shifted one position up, we can then repeat the same operations to perform a quantum gate on the formerly second to last (now virtually last) qubit.

We simply repeat this process until we have performed a quantum gate operation on all qubits and

now each qubit has virtually shuffled back to its original position. Leaving us with the correct vector expression of having applied exactly one quantum gate on each qubit. As a result we have simulated one layer  $U_i$  of a quantum circuit.

We will now show that the claim that  $|\xi\rangle$  expresses a state equal to  $|\phi'\rangle$  up to a shift in qubit order holds. We do this by introducing a function  $f$ , which expresses how the indexation of a quantum state vector of an  $n$  qubit system should change if we virtually shuffle the last qubit to the first position, while all the other qubits move one position up. By showing that the function is correct we have proven our claim since in our case  $|\xi'\rangle = f(|\phi'\rangle)$ .

Define the function

$$f(\alpha_i |i\rangle) := \begin{cases} \alpha_i | \frac{i}{2} \rangle & \text{if } i \text{ is even} \\ \alpha_i | 2^{n-1} + \frac{i-1}{2} \rangle & \text{if } i \text{ is odd} \end{cases}, \quad (3.12)$$

below we will show that this function expresses how the indexation of a quantum state vector of an  $n$  qubit system changes, when virtually shuffling the last qubit to the first position and moving all other qubits one index up.

*Proof.* Assume  $i$  is even. In this case the state of the least significant qubit, the last qubit, must be 0 as it adds 1 to  $i$  when it is in the state 1 and it is the only index that can add an odd number.

This means that only the first  $n - 1$  qubits can be nonzero for these states. These  $n - 1$  qubits virtually move one position up, and so they each become one step less significant. When each of the nonzero qubits become one step less significant they add half as much weight to the index by being in state 1. This is of course because in a binary system each index brings two times more weight than the one that is one step less significant.

And so in the case that the last qubit is in the state 0, we have  $i$  is even and we can simply find the new position of this basis state in our vector in position  $\frac{i}{2}$  since all the nonzero qubits virtually become one place less significant and thus add half as much weight as before.

For  $i$  is odd, we know that the last qubit has to be in the state 1. Since in our new ordering this qubit becomes the most significant qubit, this qubit adds weight  $2^{n-1}$  to the index number. Furthermore we need to add the factor  $\frac{i-1}{2}$ . This is because all the other qubits that were in the state 1 still add weight to the index we are in, albeit half as much as before. Before dividing the former index  $i$  by 2 we must first, however, subtract the weight 1 which came from the last qubit being in the state 1. □

Now we have shown how we can simulate applying a layer of a quantum circuit to a given quantum state. The method described above can be trivially extended to apply two-qubit operations on two adjacent qubits. To avoid only being able to apply two-qubit operations in one direction, we have implemented the matrix expression for both the original and the swapped version each two-qubit operation in the project. As an example consider the CNOT operation. We have implemented both

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (3.13)$$

and

$$CNOT_{\text{flipped}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \quad (3.14)$$

Things become a bit more complicated when we want to apply a two-qubit operation on two-qubits which are not adjacent in number, but are nevertheless connected on the hardware. The next section explain how we handle this case.

### 3.2. Non-adjacent two-qubit extension

In order to be able to use this algorithm to simulate general quantum circuits, we need to allow for the case where a two-qubit gate is applied to two qubits which are non-adjacent in the ordering of the qubits, but connected on the hardware. The Algorithm 993 is written for simulating the operation performed by a matrix built up from smaller matrices with a Kronecker product. However, a non-adjacent two-qubit gate cannot be directly expressed as a matrix. One option would be to first compose the matrix representing the non-adjacent two-qubit gate and all the quantum gates applied to in-between qubits. This would be rather costly as it would require at most  $n - 1$  Kronecker products, creating a  $2^n \times 2^n$  sized matrix (for the case that the two-qubit gate is performed on the first and last qubit of the system).

Instead we implemented an approach where we virtually alter the order of the qubits to make the non-adjacent qubits virtually adjacent, allowing for a direct two-qubit gate to be applied to them. There are two options, either we swap the targeted control qubit with the qubit adjacent to the controlled qubit, or we insert the targeted control qubit next to the controlled qubit.<sup>1</sup>

The first approach would be to virtually swap the targeted control qubit with a qubit adjacent to the controlled qubit. We can then perform the two-qubit operation normally. Now the two qubits are swapped. In order for the Algorithm 993 to work properly, we need to finish the layer of quantum gates. To make sure that the right gate is applied to the right qubit, we need to swap the gates associated with the swapped qubits in the gate list accordingly. Note that swapping the gates in the gates list is much simpler than swapping the logical qubits, as there is no superposition we need to keep track of. We do need to make sure that the qubit we swap away, to allow for the operation to occur, does not have a two-qubit gate performed on it in the layer as well, as we then need to alter its connectivity marker.<sup>2</sup> After having completed the qubit operations of the layer, we can then reset the qubit order and start the next layer. Another option would be to first continue simulating the other layers of the quantum circuit and only reset the qubits in the right order before returning the quantum state vector. This has as a downside that it requires us to keep track of how the qubits have shifted throughout the layers, to use this to change the gate order and connectivity markers of later two-qubit gates accordingly. This would lead to a lot of extra computational steps, making it an unattractive option.

Figures 3.1 and 3.2 give a visual representation of this swapping method.

Figure 3.1: Swapping q0 and q3 to perform two-qubit gate on q4 and q0.

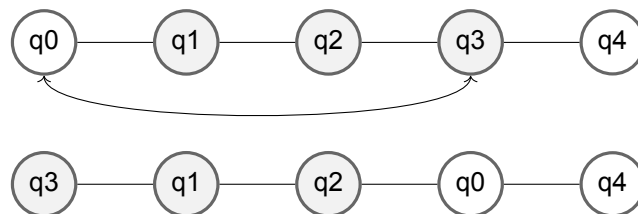
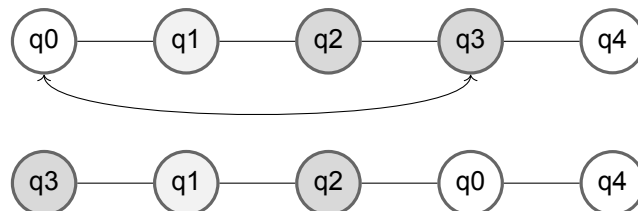


Figure 3.2: Swapping q0 and q3 to perform two-qubit gate on q4 and q0, while q3 was itself part of a two-qubit operation. It can be seen that this changes the distance between q3 and its control qubit q2.



The second approach is to virtually insert the targeted control qubit next to the controlled qubit. This

<sup>1</sup>Here the term control qubit refers to the qubit that is in the left position, i.e. the most significant qubit of the two. The term controlled qubit refers to the qubit that is in the right position, i.e. the least significant qubit of the two. These names are due to the fact that in a CNOT operation the left qubit is the control qubit and the right qubit is the controlled qubit.

<sup>2</sup>The connectivity marker is part of our implementation of the Kronx quantum simulator, it keeps track of the distance between two qubits on which a two-qubit gate needs to be performed.



approach has the benefit that it is not necessary to change the order of the gates in the gate list. A downside to this is that we are required to reset the qubit order after each layer. We are also required to employ extra steps in the case that a layer has multiple separated two-qubit gates of which the qubits cross. This is to account for the case where we insert a qubit to be the control qubit to the two-qubit operation with the last qubit, when this qubit itself was originally positioned between another pair of qubits that have a two-qubit gate performed on them in that layer. Since then we change the qubit distance between these two qubits, which we need to keep track of for when we bring them into position to perform the two-qubit operation between those two. In Figure 3.3 we show the swapping method and in Figure 3.4 an example of the case where multiple two-qubit gates were applied to non-adjacent qubits in the layer.

Figure 3.3: Inserting q0 to perform two-qubit gate on q4 and q0.

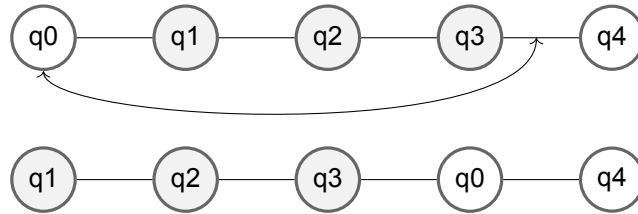
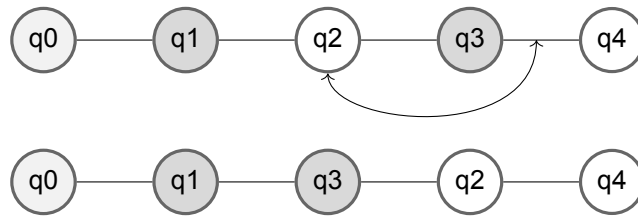


Figure 3.4: Inserting q2 to perform two-qubit gate on q2 and q4, while we also require a two-qubit gate to be performed on q3 and q1.



In this thesis, the approach where we virtually swap the targeted control qubit with a qubit adjacent to the controlled qubit is implemented.

This paragraph will give a detailed description of the implementation of the two qubit swapping method. When the Kronx function encounters a two-qubit gate of which the control qubit is non-adjacent, it enters a special case. We first read out the distance between the targeted control qubit and the controlled qubit. We use this to determine the right operation to swap the qubit currently adjacent to the controlled qubit, with the targeted control bit. The virtual qubit swap itself is done by turning the qubit state into vector shape and creating another vector which expresses the new indexation given the qubit swap. The vector expressing the indexation after the qubit swap is created using the replacer function  $f_{\text{rep}}$ . The replacer function works by exploiting the binary nature of quantum computing. It first creates a vector  $v_{\text{rep}}$  of length  $n$  with each index  $i$  having value  $2^{n-1-i}$ , with the values at the indices associated to the qubits  $j$  and  $k$  swapped

$$v_{\text{rep}}(j, k)_i = \begin{cases} 2^i, & \text{for } i \notin \{j, k\} \\ 2^j, & \text{for } i = k \\ 2^k, & \text{for } i = j. \end{cases} \quad (3.15)$$

Subsequently the vector  $v_{\text{rep}}$  is multiplied with the matrix  $M_{\text{bin}}$  of size  $2^n \times n$ , in which each row contains a binary representation of its index. Let  $M_{\text{bin}, l}$  be the  $l^{\text{th}}$  row of the matrix  $M_{\text{bin}}$  and  $b_i^l$  the state of the  $i^{\text{th}}$  bit in the binary representation of the number  $l$ , then we have

$$M_{\text{bin}, l} = [b_0^l \quad \dots \quad b_n^l]. \quad (3.16)$$

The function  $f_{\text{rep}}(j, k)$  then returns the vector expressing the indexation, after virtually swapping the

qubits  $j$  and  $k$

$$f_{\text{rep}}(j, k)_l = M_{\text{bin}, l} \cdot v_{\text{rep}}(j, k). \quad (3.17)$$

Where  $f_{\text{rep}}(j, k)_l$  represents the  $l^{\text{th}}$  index of the vector returned by the function  $f_{\text{rep}}(j, k)$  which virtually swaps the  $j^{\text{th}}$  and  $k^{\text{th}}$  qubits.

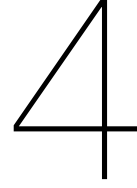
We have now created the vector which expresses how the order of the indexation of a vector representing a quantum state changes, when virtually swapping two qubits. We use this vector to reorder the indices of the quantum state. Before continuing the regular process of the Algorithm 993, we need to make sure that we still perform the right operation on the qubit which was originally positioned next to the controlled qubit in a later stage of the process.

We do this by first reading out which type of operation we are working with. If this was a single-qubit gate we simply put it in the position in the list of matrix operations associated with the targeted control qubit we swapped it with.

A bit more subtlety is required when this gate was a two-qubit gate. As replacing this qubit will then influence its position with respect to its control qubit (or controlled qubit). We need to make sure to track this accordingly. If the moved qubit overtakes its control qubit (or controlled qubit), we ensure the two-qubit gate remains expressed in the list at the first qubit encountered. We also ensure that the control qubit remains the control qubit of the operation and the controlled qubit remains the controlled qubit.

After each layer we reset the qubits to their original position by using the same replacer function described above.





# Machine learning for quantum circuit simulation

In this chapter we present the method employed to find and optimize suitable quantum circuits to mimic a chosen function.

## 4.1. Ansatz creation and selection

The aim of the project is to find values for the parameters  $x$ ,  $y$  and  $\theta$  for a given quantum computer, which minimize the cost function as described in Section 2.4.1. We do this by creating vectors  $x$  and  $y$  which meet the restrictions of Equations (2.13) to (2.16) and therefore describe a valid ansatz  $U$  for the chosen quantum computer. Subsequently, we run a machine learning technique to find optimal values of  $\theta$  for the given ansatz. We create multiple valid ansatzes  $U$  at random and run the machine learning technique to find the optimal value of the parameter  $\theta$  for each ansatz. We then compare the costs of the created circuits  $U(\theta)$  to identify which circuit performs best for the given function.

## 4.2. Parameter optimization

Once we have chosen a valid ansatz to optimize over, we use a machine learning approach to optimize the parameters  $\theta$  of the chosen circuit in order to minimize the cost estimation function

$$\min_{\theta \in [0, 2\pi]^m} c_{U,B}(\theta). \quad (4.1)$$

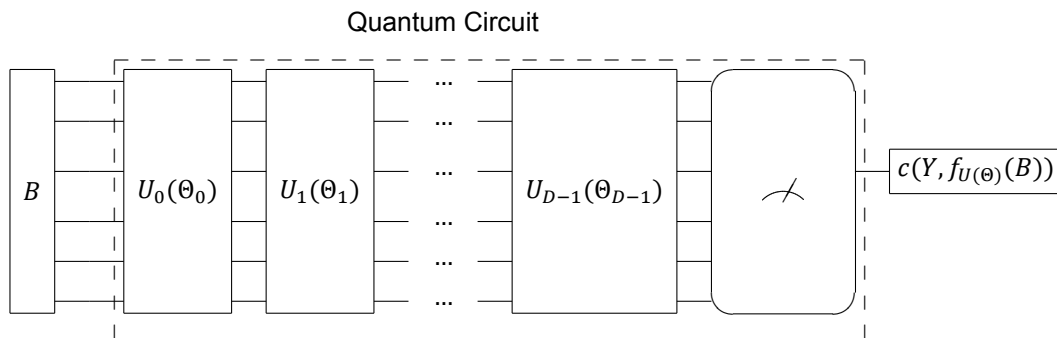
The cost estimation function is defined in Equation (2.21). In the above expression  $m$  is the number of parameterized gates in the chosen ansatz  $U$ .

The ansatz  $U$  consists of  $D$  layers  $U_i$  with  $i \in \mathcal{D} = \{0, \dots, d-1\}$ . When using a machine learning approach to optimize the angles of the gates, we can view the angles as the weights of the machine learning structure. In the learning process, the goal is to find the weights which minimize the cost function. In our case this means that we find the angles of the quantum gates which create a circuit  $U(\theta)$  as close as possible to the desired circuit.

Our training data consists of the sets  $B$  and  $Y$ . Where  $B$  is a set of  $|B|$  vectors  $b = |\phi\rangle$  with  $b \in \mathbb{C}^{2^n}$  and  $b$  unitary. The vectors  $y \in Y$  represent the probability vectors  $f(B) = Y$  associated with the circuit we wish to simulate, where  $y \in \mathbb{R}^m$ ; see Section 2.4.1.

In each iteration of the machine learning process we use the created quantum circuit  $U(\theta)$  to find the value  $f_{U(\theta)}(b)$  for each  $b \in B$ . As described in Section 2.4.1 we can find the values of  $f_{U(\theta)}(b)$  directly when using our own Kronx quantum simulator and we have to estimate it when running on an quantum computer. Using the values for  $f_{U(\theta)}(b)$  and  $y$  we can find a value for the cost function and use our machine learning approach to find gate parameters to minimize the cost function.

Figure 4.1: Machine learning approach to optimize the quantum gate parameters  $\Theta$  to let the circuit  $U(\Theta)$  best mimic a desired operation.



Notice that in the chosen approach the ansatz  $U$  describes the structure for the machine learning network, where the weights are the angles of the parameterized gates of the quantum ansatz; see Figure 4.1. Now we can use a machine learning approach to minimize the cost function. The resulting weights that minimize the cost are precisely the angles  $\Theta$  for which the resulting quantum circuit best mimics the desired function.

We have implemented two different machine learning approaches to minimize this cost function, a gradient-based and a non-gradient based approach.

### 4.3. Gradient-free machine learning - particle swarm optimization

In this project, we make use of Particle Swarm Optimization (PSO) as our gradient-free machine learning technique. PSO is used to optimize the weights  $\Theta$  of the chosen ansatz in order to minimize the cost function. PSO is an evolutionary approach that aims to find the point of minimum cost in a space.

#### 4.3.1. PSO

PSO works by first creating a swarm  $S$  of particles distributed over the space  $R^m$ , where  $m$  represents the amount of parameterized gates in the chosen ansatz. Each particle  $j$  has an initial velocity  $v_j^0 \in \mathbb{R}^m$ , let  $v_j^i$  represent the velocity of the particle  $j$  in the  $i$ -th iteration. In the first iteration, we start by evaluating the cost associated to the position each particle of the swarm takes in the space. After having evaluated the cost of the different particle positions, the velocities of the particles are updated to

$$v_j^{i+1} = \omega v_j^i + c_1 (p_j - \Theta_j^i) + c_2 (\Theta_{opt} - \Theta_j^i). \quad (4.2)$$

In the above equation  $\omega$  is the inertia constant,  $c_1$  is the memory coefficient and  $c_2$  the social coefficient. The inertia coefficient  $\omega$  represents the unwillingness of the particle to change direction. The memory coefficient  $c_1$  determines how heavily past success influences the direction the particle is travelling in. The social coefficient  $c_2$  represents how heavily the success of the best found position by any particle in the swarm influences the search direction of each particle. The parameter  $p_j$  represents the best position in space the particle  $j$  has had. The parameter  $\Theta_{opt}$  represents the point in space with the lowest cost function found by any particle in the swarm so far. The parameter  $\Theta_j^i$  represents the position in space of the  $j$ -th particle in the  $i$ -th iteration.

After having found the updated expression of the velocities of the particles we use this to find the next position in space of each particle  $j$

$$\Theta_j^{i+1} = \Theta_j^i + v_j^{i+1}. \quad (4.3)$$

The above equation does not have an expression for time, as we simply move the particles one timestep. We subsequently evaluate the cost of the new positions of the particles  $\Theta_j^{i+1}$  and again update the velocities accordingly.

This process is repeated a predetermined number of times or until a point in space has been found with a cost below a certain threshold. The goal is for the particles of the PSO algorithm to start clustering around a point in space which represents a (local) minimum.

### 4.3.2. PSO for hardware-centric quantum circuit generation

We have implemented the particle swarm optimization approach using PySwarms [Mir18] in combination with our Kronx quantum simulator as described in Chapter 3 and the cost function as described in Section 2.4.1.

## 4.4. Gradient-based machine learning

The second machine learning approach we use for optimizing the parameters of the quantum gates is a gradient-based machine learning approach. In a gradient-based machine learning we compute the gradients of the cost function with respect to the input parameters, in our case, the angles of the parameterized gates, to optimize the parameters and ultimately minimize the loss function. The angles of the parameterized gates are updated in each step as follows

$$\Theta^{i+1} = \Theta^i - l \nabla c(\Theta^i). \quad (4.4)$$

Where  $l \in \mathbb{R}_+$  is the learning rate and  $\nabla c(\Theta^i)_j$  represents the partial derivatives of the cost function with respect to the  $\theta_j$  of  $\Theta$  evaluated at  $\Theta^i$ . The learning rate is a hyperparameter, the value of which is established beforehand.

The idea of the gradient-descent method is that the parameters  $\Theta$  are altered in each iteration in a direction that further minimizes the cost function, so that we eventually find a minimum. Since in our case there are more input than output parameters, we use backpropagation to calculate the gradients of the cost function with respect to the angles of the parameterized gates. In the next subsections we will first discuss the forward pass of the machine learning approach and then discuss how backpropagation is used to find the gradients.

### 4.4.1. Forward pass

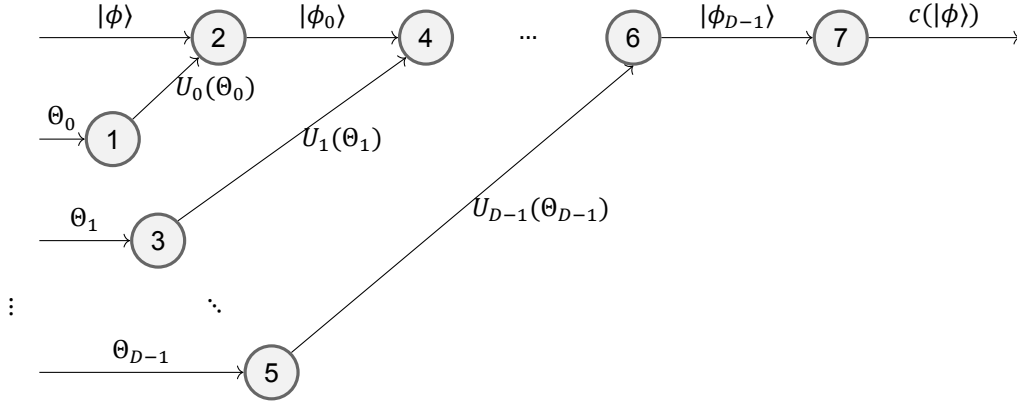
In the forward pass we go through the quantum circuit, as depicted in Figure 4.1, for several quantum input states  $b$  to find  $f_{U(\Theta)}(b)$  and eventually  $c_{U(\Theta)}(b)$ . During the forward pass, as the value of  $c_{U(\Theta)}(b)$  is calculated, a graph is created which gives structure to the process of calculating the derivatives during the backpropagation step.

The graph is built up when variables of which we require to know the gradients, in our case the angles of the parameterized gates, have some operation applied to them. Of these operations the partial derivatives with respect to the variables requiring the gradients, are calculated and stored in the nodes of the created graph. The variables created by applying operations to variables which require derivatives, will also require derivatives.

Similarly, applying an operation to these parameters results in the expansion of the graph with the associated derivative values of that operation stored in each node. This graph is built up further until we call to start the backpropagation process. During the backpropagation step, this graph is used to calculate the partial derivatives of the output variables with respect to the chosen input variables.

As an example, consider the following figure representing the graph that could be created during the forward pass, when running our gradient machine learning approach using a naive quantum simulator.

Figure 4.2: Graph created during the forward pass, where the arrows reflect the direction of information during the forward pass and the nodes store the values of the derivatives for the backward direction.



Here it can be seen that the input to the graph consists of the quantum states  $|\phi\rangle$  (where as before  $b = |\phi\rangle$ ) and the input angles  $\theta_i$  to the layers  $U_i$  of the quantum circuit. Then in node 1, a function is applied that transforms  $\theta_0$  to  $U_0(\theta_0)$  and, as we require to know the gradient of the gate parameters, the Jacobian of  $U_0(\theta_0)$  with respect to the vector  $\theta_0$  is stored in the first node. The same is done at all the other nodes in the graph, until we have completed the forward pass and found an expression for  $c(|\phi\rangle)$ .

#### 4.4.2. Backpropagation

In order to understand how backpropagation works, we will first look into calculating partial derivatives of a system of functions using the chain rule.

Let  $c$  be the output variable of which we want to calculate the partial derivative with respect to the input variable  $\theta_i$ , using the chain rule we get

$$\frac{\partial c}{\partial \theta_i} = \sum_j \frac{\partial c}{\partial u_j} \frac{\partial u_j}{\partial \theta_i}. \quad (4.5)$$

We stored the values of the partial derivatives of each operation in the nodes of the graph depicted in Figure 4.2, created during the forward pass. Therefore, we can now just go through the graph starting at  $\theta_i$  and find the value of  $\frac{\partial w}{\partial \theta_i}$ , by multiplying with the intermediate values  $\frac{\partial u_j}{\partial \theta_i}$  and  $\frac{\partial w}{\partial u_j}$  stored in the nodes. Since we are working with  $m$  input variables  $\theta_i$ , we would have to go through the graph  $m$  times to calculate the partial derivatives with respect to the cost function  $c$  using (4.5).

Since we have more input than output parameters, we make use of backpropagation instead. As the name suggests backpropagation evaluates the derivatives by going through the derivatives graph created in the forward pass, starting from the output variable working to the input variables.

Let  $c$  be the output variable of which we wish to calculate the gradient, let  $w_i$  be some intermediate variable with respect to which we wish to calculate the partial derivative. Now write the chain rule slightly different

$$\frac{\partial c}{\partial w_i} = \sum_j \frac{\partial u_j}{\partial w_i} \frac{\partial c}{\partial u_j}. \quad (4.6)$$

We evaluate this starting from the chosen output variable  $c$  and work our way down the chain. We start at the end of the graph by calculating the trivial  $\frac{\partial c}{\partial c} = 1$ . Then we go backwards through the created graph and first multiply this with the values  $\frac{\partial c}{\partial u_j}$  saved in the nodes adjacent to  $c$  and then work our way back through the graph multiplying by the partial derivatives saved in the nodes till we find  $\frac{\partial c}{\partial w_i}$ .

In our machine learning approach, we want to find the partial derivatives of  $c$  with respect to the  $m$  input parameters  $\theta_i$ . Using the graph created in the forward pass, we can calculate this most efficiently by working backwards through the graphs to the leaves of the graph to find the value of  $\frac{\partial c}{\partial \theta_i}$  when reaching

the leaf associated with  $\theta_i$ .

Since we only have one output variable  $c$ , this means that we can find the required gradient by working our way backwards through the graph only once. Note that if we had more output than input parameters, it would be most efficient to calculate the gradients by creating the same graph as described, but going through it in the forward direction.

In our machine learning approach, calculating the partial derivatives to store in the nodes during the forward pass is not trivial since the operations described are applying quantum gates and it is not directly clear how to calculate the derivative of a quantum gate. One approach could be to write out the quantum operations as matrix multiplications and find the partial derivatives of these matrix multiplications with respect to the gate parameters that determine the matrix.

Notice that that would again imply us needing to work with  $2^n \times 2^n$  dimensional matrices in complex space, which is precisely something we wish to avoid. Furthermore, this would also mean that we could never use this machine learning approach in combination with a physical quantum computer, as we would always need to work from the classical simulation to be able to find the derivatives.

To avoid only being able to use our gradient-based machine learning approach in combination with a classical simulator we use a quantum machine learning package [Ber+18] which uses a method by Schuld [Sch+19] that enables us to calculate the partial derivatives of the angles of a quantum gate directly on a quantum computer.

### 4.4.3. Partial derivatives of quantum nodes

In order to use backwards propagation in combination with a quantum computing layer, we need to be able to find the partial derivatives of quantum operations. To do this, we use a method from [Sch+19] to calculate the partial derivatives of the expectation values of quantum states after an operation.

The expectation value of the observable  $\hat{Z}$  upon measurement of the state  $|\psi\rangle$  can be expressed as

$$\langle \hat{Z} \rangle = \langle \psi | \hat{Z} | \psi \rangle. \quad (4.7)$$

It follows that the expectation value of a measurement with  $\hat{Z}$  after having our quantum circuit  $U(\theta)$  will be

$$\langle \hat{Z} \rangle_{U(\theta)} = \langle \psi | U(\theta)^\dagger \hat{Z} U(\theta) | \psi \rangle. \quad (4.8)$$

In the above equation  $U(\theta)^\dagger$  expresses the Hermitian conjugate of  $U(\theta)$ .

We write  $|\phi_{U(\theta)}\rangle = U(\theta) |\psi\rangle$ , which gives

$$\langle \hat{Z} \rangle_{U(\theta)} = \langle \phi_{U(\theta)} | \hat{Z} | \phi_{U(\theta)} \rangle. \quad (4.9)$$

Using the method from Schuld [Sch+19], we can calculate the partial derivatives of  $\frac{\partial}{\partial \theta_i} \langle \hat{Z} \rangle$ . To use this method, we first need to link our cost function to expectation values of quantum observables. As before, our cost function  $c$  can be expressed as a function of  $\theta$  as follows

$$c(\theta) = \frac{1}{|B|} \sum_{|\phi\rangle \in B} \|f_{U(\theta)}(|\phi\rangle) - f(|\phi\rangle)\|_2. \quad (4.10)$$

Where  $f(|\phi\rangle)$  expresses the desired probability vector for the input state  $|\phi\rangle$  and  $f_{U(\theta)}(|\phi\rangle)$  describes the resulting probability vector of applying  $U(\theta)$  to  $|\phi\rangle$  as before

$$f_{U(\theta)}(|\phi\rangle) = (|\langle i | U(\theta) | \phi \rangle|^2)_{i=0, \dots, 2^n-1}. \quad (4.11)$$

If we choose to perform the measurement with the observable  $\hat{Z}_i = |i\rangle \langle i|$ , we get the following expectation value

$$\langle \hat{Z}_i \rangle_{U(\theta)} = \langle \phi | U(\theta)^\dagger |i\rangle \langle i| U(\theta) | \phi \rangle = |\langle i | U(\theta) | \phi \rangle|^2. \quad (4.12)$$

This shows that we can use the expectation values of the observables  $\hat{Z}_i$  to find the values  $f_{U(\theta)}(|\phi\rangle)_i$  to use in our cost function.

Now that we have shown how the expectation values of the observables  $\hat{Z}_i$  relate to our cost function, we can use the [Sch+19] to show how to calculate the partial derivatives of the expectation values.



These partial derivatives can then be used to calculate the partial derivatives of our cost function with respect to the angles of the parameterized quantum gates.

Define  $f_{U,|\phi\rangle} : \mathbb{R}^m \mapsto \mathbb{R}^{2^n}$  the function which maps from the parameterized gate angles to the resulting probability vector of measuring the basis states

$$f_{U,|\phi\rangle}(\theta) = (|\langle i | U(\theta) | \phi \rangle|^2)_{i=0, \dots, 2^n-1}. \quad (4.13)$$

Notice that this function is essentially equal to (2.18), only now we consider it a function of  $\theta$ . For notational simplicity let  $f(\theta) := f_{U,|\phi\rangle}(\theta)$  for the remainder of this section and let  $\theta$  be an element in  $\Theta$  and let  $\hat{Z}$  be the observable, we then get

$$\partial_\theta f(\theta) = \langle \phi | U(\theta)^\dagger \hat{Z} \partial_\theta U(\theta) | \phi \rangle + \langle \phi | \partial_\theta U(\theta)^\dagger \hat{Z} U(\theta) | \phi \rangle. \quad (4.14)$$

The ansatz of our quantum circuit consists of  $D$  layers

$$U(\theta) = U_0(\theta_0) U_1(\theta_1) \dots U_{D-1}(\theta_{D-1}). \quad (4.15)$$

Since each gate in the ansatz has at most one parameter and each parameter has at most one gate associated to it, there exists a unique  $i$  such that  $\partial_\theta U_i(\theta_i) \neq 0$ , since each parameter  $\theta$  is an element of one unique  $\theta_i$  and does not effect a quantum gate in any other layer.

We can decompose this layer  $U_i(\theta_i) = V'G(\theta)W'$  where  $V'$  and  $W'$  only depend on  $\theta'$ , which are elements of  $\Theta_i$  such that  $\theta' \neq \theta$  and  $G(\theta) = I^{\otimes n-t} \otimes g(\theta) \otimes I^{\otimes t-1}$ . Here we assume that  $g(\theta)$  is a single-qubit gate, but the same argument holds for multiple qubit gates. Then we can write

$$U(\theta) = U_0(\theta_0) \dots V'G(\theta)W' \dots U_D(\theta) = VG(\theta)W \quad (4.16)$$

which leads to

$$\partial_\theta f(\theta) = \langle \phi' | G(\theta)^\dagger \hat{Q} \partial_\theta G(\theta) | \phi' \rangle + \text{h.c.} \quad (4.17)$$

In the above equation +h.c. stands for plus hermitian conjugate and we have  $|\phi'\rangle = W |\phi\rangle$  and  $\hat{Q} = V^\dagger \hat{Z} V$ . Note that for any two operators  $A$  and  $B$  we have

$$2 \langle \psi | A^\dagger \hat{Q} B | \psi \rangle + 2 \text{ h.c.} = \langle \psi | (A+B)^\dagger \hat{Q} (A+B) | \psi \rangle - \langle \psi | (A-B)^\dagger \hat{Q} (A-B) | \psi \rangle. \quad (4.18)$$

This implies that if  $G(\theta) \pm \partial_\theta G(\theta)$  are themselves unitary operations, we can evaluate Equation (4.17) directly by running the circuits  $V(G(\theta) \pm \partial_\theta G(\theta))W$ .

Case 1:  $G(\theta) \pm \partial_\theta G(\theta)$  a unitary operation with a known decomposition

We will now show a class of quantum gates for which  $G(\theta) \pm \partial_\theta G(\theta)$  are known unitary operations [Sch+19]. Consider a quantum gate  $G(\theta)$  generated by a Hermitian operator  $G$  such that  $G(\theta) = e^{-i\theta G}$  then  $\partial_\theta G(\theta) = -iG e^{-i\theta G}$ . Let  $G(\theta) |\phi'\rangle = |\phi''\rangle$  and rewrite Equation (4.17).

$$\partial_\theta f(\theta) = \langle \phi'' | \hat{Q} (-i)G | \phi'' \rangle + \text{h.c.} \quad (4.19)$$

Making use of (4.18) we get

$$\frac{2}{r} \partial_\theta f = \langle \phi'' | (I - ir^{-1}G) \hat{Q} (I - ir^{-1}G) | \phi'' \rangle - \langle \phi'' | (I + ir^{-1}G) \hat{Q} (I + ir^{-1}G) | \phi'' \rangle. \quad (4.20)$$

From this it follows that if you can implement  $\frac{1}{\sqrt{2}}(I - ir^{-1}G)$  as a quantum gate, you can evaluate Equation (4.17) directly. In Equation (4.20) we multiply both sides with  $r^{-1}$ , as there is a class of quantum gates  $G(\theta)$  for which we know how to implement  $\frac{1}{\sqrt{2}}(I - ir^{-1}G)$  as a quantum gate.

**Theorem 1** ([Sch+19]). *If the Hermitian generator  $G$  of the quantum gate  $G(\theta) = e^{-i\theta G}$  has at most two unique eigenvalues  $\pm r$  the following holds:*

$$G\left(\frac{\pi}{4r}\right) = \frac{1}{\sqrt{2}}(I - ir^{-1}G). \quad (4.21)$$

*Proof.* Since  $G$  has spectrum  $\{\pm r\}$  and is unitary we have  $G^2 = r^2 I$ . This leads to the following Taylor series of  $G(\theta)$ .

$$\begin{aligned}
G(\theta) &= e^{-i\theta G} \\
&= \sum_{k=0}^{\infty} \frac{(-i\theta)^k G^k}{k!} \\
&= I \sum_{k=0}^{\infty} \frac{(-1)^k (r\theta)^{2k}}{(2k)!} + (-ir^{-1}G) \sum_{k=0}^{\infty} \frac{(-1)^k (r\theta)^{2k+1}}{(2k+1)!} \\
&= I \cos(r\theta) - ir^{-1}G \sin(r\theta)
\end{aligned} \tag{4.22}$$

Hence  $G\left(\frac{\pi}{4r}\right) = \frac{1}{\sqrt{2}}(I - ir^{-1}G)$  □

It follows that for quantum gates generated by a Hermitian operator with at most two unique eigenvalues  $\pm r$ , we can evaluate partial derivatives directly by finding the expectation values for the circuit with a gate  $G\left(\pm\frac{\pi}{4r}\right)$  added after  $G(\theta)$ . This is the same as finding the expectation values of the circuit with  $G(\theta)$  replaced by  $G\left(\theta \pm \frac{\pi}{4r}\right)$ . And so we can find the partial derivative of the function  $f(\theta)$  with respect to the parameter  $\theta$  using the following equation

$$\partial_{\theta} f(\theta) = \frac{r}{2} \left( \langle \phi' | G\left(\theta + \frac{\pi}{4r}\right) \hat{Q} G\left(\theta + \frac{\pi}{4r}\right) | \phi' \rangle - \langle \phi' | G\left(\theta - \frac{\pi}{4r}\right) \hat{Q} G\left(\theta - \frac{\pi}{4r}\right) | \phi' \rangle \right). \tag{4.23}$$

Rotation gates which are generated by the Pauli-spin matrices to implement a rotation around a specific axis fall into the class of quantum gates for which we can use the method described above to calculate the partial derivatives, as they are generated by matrices with only two unique eigenvalues  $\pm 1$  [Sch+19].

In Appendix D.1 we show that the parameterized native gates considered in this project all fall in the class of quantum gates for which  $G(\theta) \pm \partial_{\theta} G(\theta)$  can be evaluated using the method described above.

Case 2:  $G + \partial_{\theta} G$  not a unitary operation with a known decomposition

Not all two-qubit gates natively implemented on a quantum computer belong to the class described, where  $G \pm \partial_{\theta} G$  are unitary operations we can implement on a quantum computer. For these gates, we can use another method from [Sch+19] to estimate the gradients of the quantum nodes.

Note that we can write  $\partial_{\theta} G(\theta)$  as a complex square matrix. We can subsequently decompose this matrix  $\partial_{\theta} G(\theta)$  as a linear combination of unitary matrices

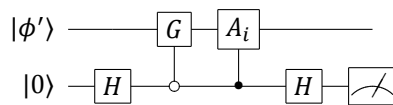
$$\partial_{\theta} G(\theta) = \sum_{i=1}^k \alpha_i A_i, \tag{4.24}$$

where  $\alpha_i \in \mathbb{R}$  and  $A_i$  unitary complex matrices [Sch+19]. Combining this decomposition with equation (4.17) gives

$$\partial_{\theta} f = \sum_{i=1}^k \alpha_i \left( \langle \phi' | G(\theta)^{\dagger} \hat{Q} A_i | \phi' \rangle + \text{h.c.} \right). \tag{4.25}$$

And so we can compute the partial derivatives if we can find the matrices  $A_i$  and their amplitudes  $\alpha_i$ , as well as an estimation for  $\langle \phi' | G(\theta)^{\dagger} \hat{Q} A_i | \phi' \rangle + \text{h.c.}$ . We claim that this latter value can be estimated by running the circuit presented in Figure 4.3 multiple times [Sch+19].

Figure 4.3: Circuit that can be run to estimate  $\langle \phi' | G(\theta)^{\dagger} \hat{Q} A_i | \phi' \rangle + \text{h.c.}$  [Sch+19].



Here the wire with  $|\phi'\rangle$  represent  $n$  qubits in the state  $|\phi'\rangle = W|\phi\rangle$  and the extra qubit in the state  $|0\rangle$  is a so-called ancilla qubit. A ancilla qubit is a qubit which gets added in a quantum algorithm to be able to perform some computational step. In order to calculate the gradients using this method we need  $n + 1$  qubits.

In the circuit presented in Figure 4.3, we first apply a Hadamard gate to the ancilla qubit, which starts in the state  $|0\rangle$ , resulting the  $n + 1$ -qubit system to be in the state

$$\frac{1}{\sqrt{2}} |\phi'\rangle (|0\rangle + |1\rangle). \quad (4.26)$$

Then, we apply the controlled- $G$  and controlled- $A_i$  gates. The controlled- $G$  gate is controlled in such a way that  $G$  is only applied to the quantum states  $|\phi'\rangle$  if the ancilla bit is in the state  $|0\rangle$ . The controlled- $A_i$  gate is such that the gate  $A_i$  is only applied to the qubits of the first wire where the ancilla bit is in the state  $|1\rangle$ . This results in the system being in the state

$$\frac{1}{\sqrt{2}} (G(\theta) |\phi'\rangle |0\rangle + A_i |\phi'\rangle |1\rangle). \quad (4.27)$$

Then a second Hadamard gate is applied to the ancilla bit, resulting in the state

$$\frac{1}{2} ((G(\theta) + A_i) |\phi'\rangle |0\rangle + (G(\theta) - A_i) |\phi'\rangle |1\rangle). \quad (4.28)$$

Subsequently the ancilla qubit is measured. The measurement of the ancilla qubit finds  $|0\rangle$  with probability  $p_0^i = \frac{1}{4} (\langle \phi' | (G + A_i)^\dagger (G + A_i) | \phi' \rangle)$ . In this case, the first  $n$  qubits are now in the state

$$|\phi'_0\rangle = \frac{1}{2\sqrt{p_0^i}} (G + A_i) |\phi'\rangle. \quad (4.29)$$

Similarly, with probability  $p_1^i = \frac{1}{4} (\langle \phi' | (G - A_i)^\dagger (G - A_i) | \phi' \rangle)$  we find the ancilla qubit to be in the state  $|1\rangle$ , in which case the first  $n$  qubits are in the state

$$|\phi'_1\rangle = \frac{1}{2\sqrt{p_1^i}} (G - A_i) |\phi'\rangle. \quad (4.30)$$

We run the described circuit multiple times to find the expectation values  $E_j^i$ , where

$$E_0^i = \frac{1}{4p_0^i} \langle \phi' | (G + A_i)^\dagger \hat{Q} (G + A_i) | \phi' \rangle \quad (4.31)$$

and

$$E_1^i = \frac{1}{4p_1^i} \langle \phi' | (G - A_i)^\dagger \hat{Q} (G - A_i) | \phi' \rangle. \quad (4.32)$$

When we combine these results with Equation (4.18) we get

$$\langle \phi' | G(\theta)^\dagger \hat{Q} A_i | \phi' \rangle + \text{h.c.} = 2 (p_0^i E_0^i - p_1^i E_1^i) \quad (4.33)$$

from which it follows that

$$\partial_\delta f = \sum_{i=1}^k \alpha_i 2 (p_0^i E_0^i - p_1^i E_1^i) \quad (4.34)$$

using Equation (4.25). We can estimate the values of  $(p_0^i E_0^i - p_1^i E_1^i)$  by estimating the values  $p_j^i$  and  $E_j^i$  for each  $A_i$ , by running the described circuit multiple times.

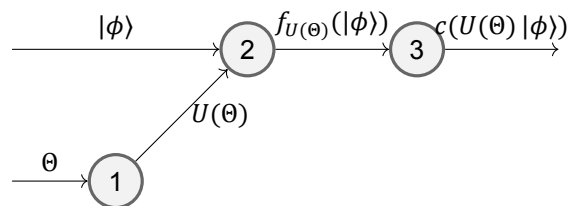
Finding the gates  $A_i$  and their amplitudes  $\alpha_i$  is non-trivial, but needs to be done only once for all implemented quantum gates.

#### 4.4.4. Gradient-based backpropagation for quantum circuits

In Section 4.4.1 we gave the graph created during the forward pass, when using backpropagation to optimize the parameters of a quantum circuit; see Figure 4.2 .

Using the methods from [Sch+19], presented in the sections above, we can rewrite this graph where we treat the entire quantum circuit as one operation and we can calculate these partial derivatives, with respect to all gate parameters, directly. In this way, we can use backpropagation when (part of) the machine learning structure is performed on a quantum computer, without having to be able to find the partial derivatives of  $U_i(\theta_i)$  on a classical computer. This method also saves time in the backpropagation step, as we now have less nodes to go through.

Figure 4.4: Graph created during the forward pass when using PennyLane [Ber+18] to evaluate the gradients of a circuit using an actual quantum computer.



Note that in our case, where the quantum circuit is one of the few computational steps of the algorithm, backpropagation does not save as much computational time as it would in a larger machine learning structure, when there are more layers in the graph for calculating the derivatives.

We still save computational time by using backpropagation, as the last node where we calculate the cost function actually consists of several computational steps, all with their own partial derivatives. Furthermore, this method allows us to use a quantum circuit as a layer in a larger machine learning structure which is optimized using backpropagation.

For the implementation of the classical part of this backpropagation algorithm we make use of PyTorch [Pas+19]. PyTorch is a classical computing package which can be used to calculate the gradients of the output parameters with respect to the chosen input parameters by creating a derivative graph and performing backpropagation as described. Since we are using a quantum layer we make use of PennyLane [Ber+18]. PennyLane is a quantum machine learning plugin that can be used in combination with PyTorch and has the recipes implemented to calculate the derivatives of quantum operations as described above. PennyLane has a function that allows the user to create random quantum states as input quantum states, these are subsequently prepared on a quantum computer using a method from Möttönen and Vartiainen [MV05]. When using this function from the PennyLane package the function required real input vectors, and so we only use gradient-based machine learning to learn over quantum states with real amplitudes.



# 5

## Results

In the previous section, we have described two different machine learning approaches that can be used for finding quantum circuits to mimic the behaviour of a given function. In this chapter, we will give the results of using both approaches on different problems. First, we will show the result of running our quantum machine learning algorithms to mimic simple known quantum operations. Secondly, we use our approach to find a circuit that could be used for the search problem. All the results presented in this section were created for the IBM Ruschlikon computer; see Figure D.3.

### 5.1. Simple function

The first test for our machine learning approach is to learn a simple function

$$f_{\text{simple}} : \mathbb{C}^{2^n} \mapsto \mathbb{R}^{2^n}. \quad (5.1)$$

Where we define  $f_{\text{simple}}$  to be the function that returns the associated probabilities of each quantum basis state after applying the quantum circuit  $A$ . The quantum circuit  $A$  consists of the following quantum gates, one for each of the  $n$  qubits

$$A = \begin{cases} X \otimes H \otimes X \otimes H \otimes \cdots \otimes X \otimes H, & \text{for } n \text{ even} \\ X \otimes H \otimes X \otimes H \otimes \cdots \otimes X, & \text{for } n \text{ odd.} \end{cases} \quad (5.2)$$

In the above equation  $X$  is the NOT gate

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (5.3)$$

and  $H$  represents the Hadamard gate

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (5.4)$$

We can express  $f_{\text{simple}}$  as follows

$$f_{\text{simple}}(|\phi\rangle) = \langle \phi | A \hat{Z}_i A | \phi \rangle_{i=0, \dots, 2^n-1}. \quad (5.5)$$

The theoretical decomposition of this quantum circuit into rotation gates consists of the decompositions of the  $H$  gate and the  $X$  gate. They are as follows

$$H = e^{\frac{i\pi}{2}} R_Z\left(\frac{\pi}{2}\right) R_X\left(\frac{\pi}{2}\right) R_Z\left(\frac{\pi}{2}\right) \quad (5.6)$$

and

$$X = e^{\frac{i\pi}{2}} R_X(\pi). \quad (5.7)$$

Phase shifts  $e^{\frac{i\theta\pi}{2}}$  can be ignored as they have an immeasurable effect on the system; see Appendix A.2.3. The Hadamard gate has a theoretical decomposition of three rotation operations and the  $X$  gate can be performed using only one rotation around the x-axis. Note that since most quantum computers only have  $R_X\left(\frac{\pi}{2}\right)$  natively implemented, the  $X$  gate usually requires two rotations of  $\pi/2$  around the x-axis.

First, we run the quantum machine learning circuit for this problem giving only the computational basis states as input vectors, this is done for both the gradient-based and non-gradient based approach.

Then, we use our PSO machine learning approach to find a quantum circuit mimicking the behaviour of  $f_{\text{simple}}$  while learning over a predetermined number of random quantum states.

Finally, we use our gradient-based machine learning algorithm to learn over a predetermined number of random real unit states to find a quantum circuit that mimics the circuit  $A$ .

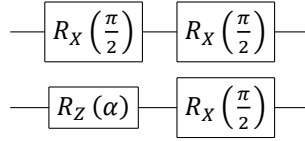
### 5.1.1. PSO - basis states

We start by running the PSO machine learning approach for finding a circuit that mimics the behaviour of  $A$  for the case  $n = 2$ . We let the algorithm create 50 random ansatzes and for each ansatz repeated the PSO procedure 70 times with a swarm of 450 particles. This was done several times and we found four ansatzes of depth  $D = 2$  that perfectly mimic the given circuit for the computational basis states. We will now take a more in-depth look at one of the results. The resulting ansatz looks as follows

$$U(\theta) = \left( R_X\left(\frac{\pi}{2}\right) \otimes R_X\left(\frac{\pi}{2}\right) \right) \left( R_X\left(\frac{\pi}{2}\right) \otimes R_Z(\alpha) \right), \quad (5.8)$$

where the angle  $\alpha$  is not defined as it does not have an impact on the cost function.

Figure 5.1: The found circuit  $U(\theta)$  for simulating the effect of Equation (5.2) for 2 qubits learning over the computational basis states.



In the found circuit the Hadamard gate is mimicked by

$$R_X\left(\frac{\pi}{2}\right)R_Z(\alpha) = \frac{1}{\sqrt{2}} \begin{bmatrix} e^{-i\frac{\alpha}{2}} & -ie^{i\frac{\alpha}{2}} \\ -ie^{-i\frac{\alpha}{2}} & e^{i\frac{\alpha}{2}} \end{bmatrix}. \quad (5.9)$$

It can be seen that this has the desired effect on the computational basis states for any  $\alpha$ , as for any value of  $\alpha$  the matrix described in Equation (5.9) simply spreads the amplitude of the basis states out to an equally divided superposition. This leads to a probability of  $\frac{1}{2}$  to measure either basis state, for both input basis vectors. Therefore, our circuit results in the same vector describing the probabilities of measuring each basis state as the desired operation. For other input states such as

$$|\phi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad (5.10)$$

however, this matrix does not work for all values of  $\alpha$ . For instance

$$R_X\left(\frac{\pi}{2}\right)R_Z(0)|\phi\rangle = \frac{1}{2} \begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1-i \\ 1-i \end{bmatrix} \quad (5.11)$$

which leads to an equal probability of measuring either basis state, whereas

$$H|\phi\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (5.12)$$

leads to finding  $|0\rangle$  upon measurement with certainty.

In the found decomposition the  $X$  gate is simulated by applying  $R_X\left(\frac{\pi}{2}\right)$  twice

$$R_X\left(\frac{\pi}{2}\right)R_X\left(\frac{\pi}{2}\right) = e^{\frac{-i\pi}{2}} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \quad (5.13)$$

Note that this is precisely equal to the theoretical decomposition of the  $X$  gate.

### 5.1.2. Gradient descent - basis states

The gradient descent method gives rise to the same results as PSO when learning over the computational basis states. Here we run 70 epochs of our gradient descent method and all the other parameters remain as before.

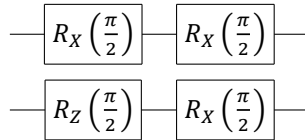
### 5.1.3. PSO - complex states

Finally, we find a quantum circuit mimicking the behaviour of  $A$  for all possible input states. We run the PSO machine learning approach with a batch of 16 random unitary complex input vectors to learn over. We performed 50 iterations of the PSO scheme on each ansatz and we again used a swarm of 450 particles and 50 ansatzes. The circuit resulting from running the quantum circuit generator with the described input is as follows

$$U(\theta) = \left(R_X\left(\frac{\pi}{2}\right) \otimes R_X\left(\frac{\pi}{2}\right)\right) \left(R_X\left(\frac{\pi}{2}\right) \otimes R_Z\left(\frac{\pi}{2}\right)\right). \quad (5.14)$$

The found circuit results in the same probability vectors for all possible input states as the aimed-for circuit  $A$ .

Figure 5.2: The found circuit  $U(\theta)$  for simulating the effect of Equation (5.2) for 2 qubits learning over random quantum states.



The found circuit has less depth than a theoretical decomposition of the operation in the native gates. This is due to the decomposition of the Hadamard gate, and the fact that in our found circuit we apply a different operation that results in the same probability distribution of the output states. In our found circuit the Hadamard gate is replaced by  $R_X\left(\frac{\pi}{2}\right)$  followed by  $R_Z\left(\frac{\pi}{2}\right)$ , the matrix expression of these operations combined is

$$R_X\left(\frac{\pi}{2}\right)R_Z\left(\frac{\pi}{2}\right) = \frac{1}{2} \begin{bmatrix} 1-i & 1-i \\ -1-i & 1+i \end{bmatrix}. \quad (5.15)$$

It can be seen that this results in the right probabilities of measuring each basis state by writing

$$R_X\left(\frac{\pi}{2}\right)R_Z\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}} \begin{bmatrix} e^{\frac{-i\pi}{4}} & e^{\frac{-i\pi}{4}} \\ -e^{\frac{i\pi}{4}} & e^{\frac{i\pi}{4}} \end{bmatrix}. \quad (5.16)$$

Then for any input qubit state

$$|\phi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix}, \quad (5.17)$$

we have

$$R_X\left(\frac{\pi}{2}\right)R_Z\left(\frac{\pi}{2}\right)|\phi\rangle = \begin{bmatrix} e^{\frac{-i\pi}{4}}(\alpha_0 + \alpha_1) \\ -e^{\frac{i\pi}{4}}(\alpha_0 - \alpha_1) \end{bmatrix}. \quad (5.18)$$



The probability of finding  $|0\rangle$  upon measurement is  $|\alpha_0 + \alpha_1|^2$  and the probability of finding  $|1\rangle$  is  $|\alpha_0 - \alpha_1|^2$ , which is exactly the same as after applying the Hadamard gate.

In the found decomposition the  $X$  gate is again simulated by applying  $R_X\left(\frac{\pi}{2}\right)$  twice, which gives

$$R_X\left(\frac{\pi}{2}\right)R_X\left(\frac{\pi}{2}\right) = e^{-\frac{i\pi}{2}} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \quad (5.19)$$

#### 5.1.4. Gradient descent - real states

We also used the gradient descent method to find a quantum circuit that perfectly mimics the desired circuit  $A$ . We used the gradient descent method by learning over 16 random real unit vectors and the associated desired outcome vectors. As before, we kept the parameters the same as in the PSO case and we used 70 epochs of the machine learning scheme.

Using gradient descent we found the exact same result as in Section 5.1.3.

#### 5.1.5. Simple problem with $n > 2$

Having found the right ansatz for  $n = 2$  we generalize this for the case  $n$  and first use this as the ansatz for our quantum machine learning approach. This gives the following ansatz:

$$U(\theta) = \begin{cases} \left( R_X\left(\frac{\pi}{2}\right) \otimes \dots \otimes R_X\left(\frac{\pi}{2}\right) \right) \left( R_X\left(\frac{\pi}{2}\right) \otimes R_Z(\theta_0) \otimes \dots \otimes R_X\left(\frac{\pi}{2}\right) \otimes R_Z(\theta_{n/2}) \right), & \text{for } n \text{ even} \\ \left( R_X\left(\frac{\pi}{2}\right) \otimes \dots \otimes R_X\left(\frac{\pi}{2}\right) \right) \left( R_X\left(\frac{\pi}{2}\right) \otimes R_Z(\theta_0) \otimes \dots \otimes R_X\left(\frac{\pi}{2}\right) \right), & \text{for } n \text{ odd.} \end{cases} \quad (5.20)$$

We do this to test our algorithm's ability to learn the right parameters when we are working in a larger Hilbert space with more parameters to optimize. We also want to know how large the batch size needs to be when learning over random input states.

Using this result we also use our machine learning approach to find a circuit to mimic the function  $f_{\text{simple}}$  for  $n > 2$  without giving the right ansatz as input.

##### Simple problem with $n = 5$

We run our machine learning approaches to optimize the parameters  $\theta$  of  $U(\theta)$  as described above in Equation (5.20) to mimic the behaviour of  $A$  in the case of 5 qubits.

Using our particles swarm approach, we create a swarm of 450 particles and run 70 iterations of the scheme. We use this approach for different batch sizes ranging from 32 to two random unitary complex input states. For each batch size, PSO was able to perfectly optimize the angles to  $\frac{\pi}{2}$  for each attempt. Note here that the space we are working in has dimension 32 so the approach works with a batch much smaller than would minimally be required to span the space.

Furthermore, we noticed that for such small batch sizes wrong ansatzes still perform bad, meaning we can run our optimization scheme with a very small batch size saving a considerable amount of time. We tried this and created 1000 random ansatzes and used our PSO optimization scheme as described above learning over a batch of size two. Our machine learning algorithm correctly identified the right ansatz (as described before) and found the correct angles.

We did the same with the gradient descent approach which was consistently able to find the right angles. However, as the batch size decreased the gradient descent method became less reliable. Our gradient-based machine learning approach performed consistently well up to a batch size as small as five.

##### Simple problem with $n = 10$

Doing the same with  $n = 10$  PSO is able to correctly learn the angles using a batch size as small as 2. Our gradient-descent method was not consistently effective. The algorithm seemed unable to find a negative gradient to optimize in that direction. This is a known problem of gradient-based machine learning for quantum computers as the circuit size grows [McC+18].

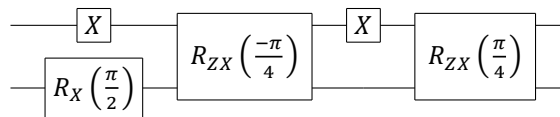
## 5.2. CNOT gate

As a next step we test our algorithms ability to learn a circuit that mimics the behaviour of the CNOT gate

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (5.21)$$

We do this for the IBM native gate set. The decomposition of the CNOT gate as used by IBM is given in Figure 5.3.<sup>1</sup>

Figure 5.3: Circuit to implement CNOT on IBM computer [Gok+20].

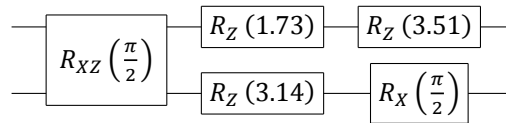


This decomposition uses six layers, as the X gate needs to be implemented using two  $R_Z\left(\frac{\pi}{2}\right)$  gates.

### 5.2.1. PSO - basis states

Using the four computational basis states as input to train over we found multiple circuits with depth  $D = 3$  that perfectly mimic the desired operation. See Figure 5.4 for an example of this circuit.

Figure 5.4: Circuit to mimic CNOT over computational basis.

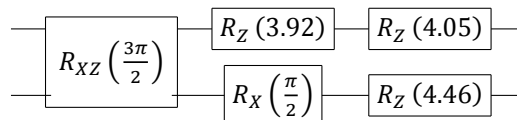


To find this circuit we tried 50 different ansatzes used 50 repetitions of the PSO scheme for each ansatz.

### 5.2.2. Gradient descent - basis states

Using gradient descent to optimize the angles by learning over the basis states as input states we found the circuit presented in Figure 5.5. This circuit mimics the CNOT gate over the input states and from the gradient descent method we can see that only the input parameter for the  $R_{XZ}$  gate has a measurable effect.

Figure 5.5: Circuit to mimic CNOT over computational basis.



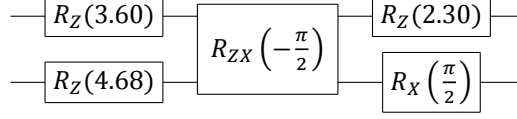
To find this circuit we tried 100 different ansatzes used 50 epochs in the gradient-descent method for each ansatz.

<sup>1</sup>Here we used that the CR pulse gives rise to the  $R_{ZX}$  gate.

### 5.2.3. PSO - complex states

Using 4 random input states to learn over, the thinnest circuit that we found has depth  $D = 3$  and perfectly mimics the CNOT gate on all possible input states. The circuit found is represented in Figure 5.6.

Figure 5.6: Circuit to mimic CNOT learned over complex input states.

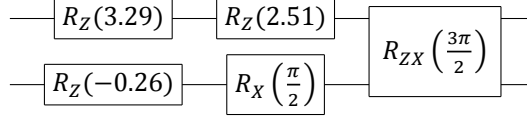


To find this circuit we tried 50 different ansatzes used 50 repetitions of the PSO scheme for each ansatz.

### 5.2.4. Gradient descent - real states

Using 4 random real input states, we were able to a circuit that perfectly mimics the CNOT operation for the input. The circuit is depicted in Figure 5.7.

Figure 5.7: Circuit to mimic CNOT learned over real input states



To find this circuit, we tried 100 different ansatzes used 50 epochs of the gradient descent scheme for each ansatz.

## 5.3. The search problem

The search problem is one of the problems for which an efficient quantum algorithm is known. The search problem is as follows, given a binary vector  $x$  of length  $N$ , return a nonzero index of the vector. In this problem we assume that the hamming weight  $t$  of the binary vector is known. Furthermore, we assume to have access to an oracle function  $O_x$ . The oracle function negates the amplitudes of the basis states corresponding to a nonzero index in  $x$

$$O_x |\phi\rangle = \sum_{i=0}^N (-1)^{x_i} |i\rangle. \quad (5.22)$$

Grover's search algorithm allows the user to solve the search problem using  $\mathcal{O}(\sqrt{N})$  queries to the oracle function and using  $\mathcal{O}(\sqrt{N} \log N)$  other gates [Gro96] [Wol19]. Grover's search algorithm starts with an  $n = \log_2(N)$  qubit system in the state  $|0\rangle$ . Subsequently a Hadamard gate is applied to all qubits creating a superposition over all the basis states  $|U_0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$ .

After the state  $|U_0\rangle$  is prepared, Grover's search algorithm runs the so-called Grover's iterate  $\mathcal{G}$   $\tilde{k}$  times. A subsequent measurement of the qubits returns a state  $|i\rangle$  for which  $x_i = 1$  with probability  $P_{N,t}$

$$P_{N,t} = \begin{cases} 1, & \text{for } k \in \mathbb{Z} \\ 1 - \sin^2(2(k - \tilde{k})\theta), & \text{for } k \notin \mathbb{Z}. \end{cases} \quad (5.23)$$

Where in the above expression  $k = \frac{\pi}{4\theta} - \frac{1}{2}$  and  $\tilde{k}$  is the integer closest to  $k$ . In the sections below we will give an expression for the angle  $\theta$  and the Grover iterates followed by a proof of the given probability of success  $P_{N,t}$ .

Let  $|B\rangle = \frac{1}{\sqrt{N-t}} \sum_{i:x_i=0} |i\rangle$  be the state that is a linear combination of all the basis states representing

indices of the vector  $x$  for which  $x_i = 0$ . Similarly  $|G\rangle = \frac{1}{\sqrt{t}} \sum_{i:x_i=1} |i\rangle$  is a linear combination of all the good basis states that represent indices for which  $x_i = 1$ .

Let  $|U_0\rangle$  be the quantum state that is the input state for the first Grover iterate. We define  $\theta$  to be the angle between the input state and the superposition of bad states  $|B\rangle$

$$\theta = \arccos \operatorname{Re}(\langle B | U_0 \rangle). \quad (5.24)$$

A Grover iterate  $\mathcal{G}$  consists of the following combination of actions. Let  $|U_l\rangle$  be the quantum state that is the input to the  $l^{\text{th}}$  Grover iterate. First we reflect the state  $|U_l\rangle$  along the bad states  $|B\rangle$  by applying the oracle function. Then we reflect the resulting state  $O_x |U_l\rangle$  along the original position of the input state  $|U_0\rangle$ , this is called the amplitude amplification step. This combination of actions results in the state  $|U_{l+1}\rangle = \mathcal{G} |U_l\rangle$ . In the Figures 5.8, 5.9 and 5.10 we depict the effect these actions have on the qubit state.

Figure 5.8: Visual representation of the states  $|B\rangle$ ,  $|G\rangle$  and  $|U_0\rangle$ .

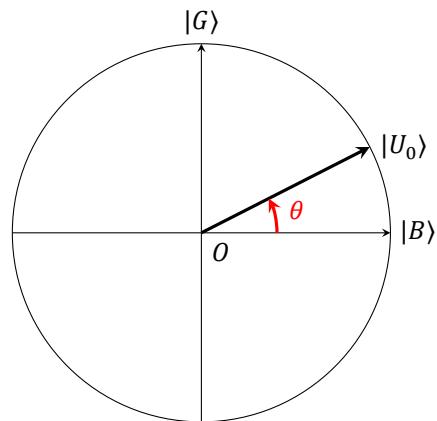


Figure 5.9: Visual representation of the states  $|G\rangle$ ,  $|B\rangle$  and  $O_x |U_0\rangle$ .

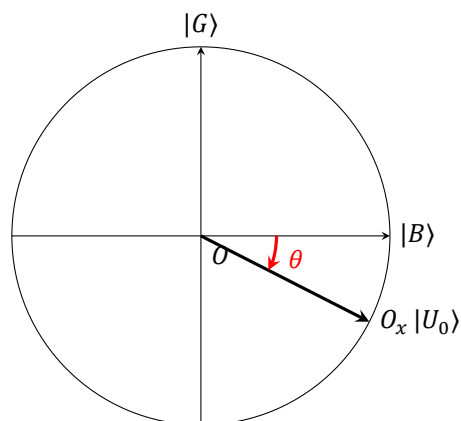
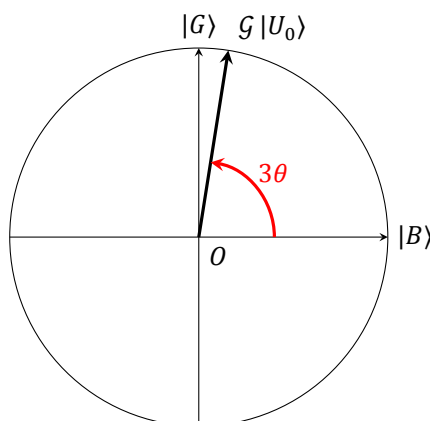


Figure 5.10: Visual representation of the states  $|G\rangle$ ,  $|B\rangle$  and  $\mathcal{G}|U_0\rangle$ .

We will now present a proof of the probability of success  $P_{N,t}$ , as given in Equation (5.23), of Grover's search for the cases  $k \in \mathbb{Z}_+$  and  $k \notin \mathbb{Z}_+$  [Wol19].

For  $k \in \mathbb{Z}_+$  apply the Grover iterate  $k = \frac{\pi}{4\theta} - \frac{1}{2}$  times starting from the state  $|U_0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$ . Note that  $|U_0\rangle = \sin(\theta) |G\rangle + \cos(\theta) |B\rangle$ . We apply the Grover iterate  $\mathcal{G}$  to the state  $|U_0\rangle$   $k$  times, each time adding  $2\theta$  to the angle

$$\mathcal{G}^k |U_0\rangle = \sin((2k+1)\theta) |G\rangle + \cos((2k+1)\theta) |B\rangle. \quad (5.25)$$

Since  $k = \frac{\pi}{4\theta} - \frac{1}{2}$ , we end up in the state  $\mathcal{G}^k |U_0\rangle = \sin\left(\frac{\pi}{2}\right) |G\rangle + \cos\left(\frac{\pi}{2}\right) |B\rangle = |G\rangle$ . Which means that  $\mathcal{G}^k |U_0\rangle$  is a linear combination of basis states representing an index of  $x$  for which  $x_i = 1$ . Upon measurement we find a state  $|i\rangle$  for which  $x_i = 1$  with certainty, and so we solve the search problem with success probability 1.

For  $k \notin \mathbb{Z}_+$  we repeat the Grover iterate  $\tilde{k}$  times. Where we choose  $\tilde{k}$  to be the integer closest to  $k$ . We then apply the Grover iterate  $\tilde{k}$  times.

$$\mathcal{G}^{\tilde{k}} |U_0\rangle = \sin((2\tilde{k}+1)\theta) |G\rangle + \cos((2\tilde{k}+1)\theta) |B\rangle \quad (5.26)$$

And so the probability of finding a state  $|i\rangle$  representing a solution to the Grover's search problem upon measurement after the  $\tilde{k}$  iterations is  $P_{N,t} = \sin^2((2\tilde{k}+1)\theta) = \sin^2\left(\frac{\pi}{2} + 2(\tilde{k}-k)\theta\right) = 1 - \sin^2(2(\tilde{k}-k)\theta)$ . This proves that Equation (5.23) holds.

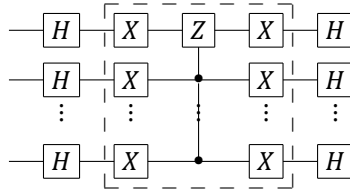
We can use the above to derive how many Grover iterations should be necessary for each type of the search problem and what the subsequent probability of success is. The angle between the states  $|B\rangle$  and the input state  $|U_0\rangle$  only depend on the problem size  $N$  and the hamming weight  $t$ .

$$\theta = \arccos \sqrt{\frac{N-t}{N}} \quad (5.27)$$

This can be used to directly find an expression for  $k$  or  $\tilde{k}$  and the success probability  $P_{N,t}$  for each instance of the search problem.

We will use our hardware-centric machine learning approach to find a circuit to simulate the second step of the Grover iterate, the amplitude amplification step. As in Grover's search algorithm we assume access to a quantum oracle, which implicitly holds the information of which indices  $i$  are such that  $x_i = 1$  in our generated circuit. In Grover's search algorithm the amplitude amplification step consists of a reflection around the states  $|U_0\rangle$ , that were the input state to the first iteration of the Grover iterate. Remember that we have  $|U_0\rangle = H^{\otimes n} |0\rangle$ , which means  $|0\rangle = H^{\otimes n} |U_0\rangle$  since the Hadamard gate is its own inverse. This means it is easiest to find this operation by finding a reflection around the state  $|0\rangle$  say  $R_0$ . A reflection around  $|U_0\rangle$  will then be expressed by  $H^{\otimes n} R_0 H^{\otimes n}$ . A reflection around the state  $|0\rangle$  is equal to negating the amplitudes of all states  $|i\rangle$  such that  $i \neq 0$ . It is easy to see that the circuit of Figure 5.11 does that job.

Figure 5.11: Amplitude amplification step for  $n$  qubits.



Notice that the part of this circuit in the dashed box expresses a reflection around the state  $|0\rangle$ . This can be seen as first the  $X^{\otimes n}$  operation ensures that the states  $|0\rangle$  and  $|2^n - 1\rangle$  change positions (it also switches the position of the other basis states). Subsequently the  $C^{\otimes(n-1)}Z$  (i.e. the Z gate on the first qubit controlled by all other qubits) ensures that the amplitude of the state  $|2^n - 1\rangle$  gets negated and all other amplitudes remain unchanged. Finally the X gates ensure that the states  $|0\rangle$  and  $|2^n - 1\rangle$  (and all other basis states) change back to their original positions. This ensures that a negative amplitude is added only to the state  $|0\rangle$ , while all other states remain unchanged. This is equal to adding a negative amplitude to all states but the state  $|0\rangle$ , up to a phase shift.

Since we perform Hadamard gates on all qubits before and after the part of the circuit in the dashed box, the amplitude amplification step as presented performs a rotation around the states  $|u_0\rangle = H|0\rangle$ . Notice that the implementation of this operation on a physical quantum computer potentially adds a lot of depth, as the  $C^{\otimes(n-1)}Z$  gate is a  $n$ -qubit gate that needs to be decomposed in terms of native gates. Therefore this amplitude amplification step is a good candidate for our quantum circuit generator to see if we can find a circuit to simulate this operation. We run the quantum circuit generator for each combination of  $N$  and  $t$  separately. We are looking for a quantum circuit for simulating the amplitude amplification step of the search problem for several vector length and hamming weight combinations. Since we are only interested in training a circuit to learn the amplitude amplification step, the input training data fed into our machine learning algorithm consists of the vectors  $x \in R^{2^n}$  which represent the vectors  $|U_0\rangle$  for different options of answer vectors  $x$  of length  $N$  and hamming weight  $t$ . We only consider  $N$  such that there exists an  $n$  with  $2^n = N$ . We train the circuit using the input data

$$|\phi_x\rangle = O_x H^{\otimes n} |0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{2^n-1} (-1)^{x_i} |i\rangle, \tag{5.28}$$

where  $x$  is a vector of length  $n$  with hamming weight  $t$ . The associated output training data  $Y$  consists of the associated probability vectors  $y_x$

$$(y_x)_i = \frac{x_i}{t}. \tag{5.29}$$

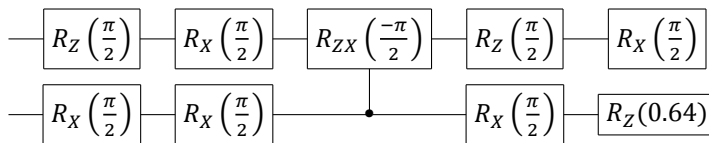
### 5.3.1. $N = 4$ and $t = 1$

The case  $N = 4$  and  $t = 1$  has that  $t = \frac{N}{4}$ , therefore Grover's algorithm can solve this problem with certainty, using only one query to the oracle.

Using our machine learning algorithm we were able to find a circuit solving this problem with success probability 1, requiring only one query to the oracle and depth  $D = 5$ . The found circuit is represented in Figure 5.12.

We found this circuit by running the approach with PSO to optimize the parameters. We learned over  $N = 1000$  ansatzes, using a swarm of size  $|S| = 450$ , a batch of size  $|B| = 4$  and 70 repetitions.

Figure 5.12: Resulting circuit mimicking the amplitude amplification step of Grover's search problem with  $N = 4$  and  $t = 1$ .



The above circuit can be worked out to create the following matrix

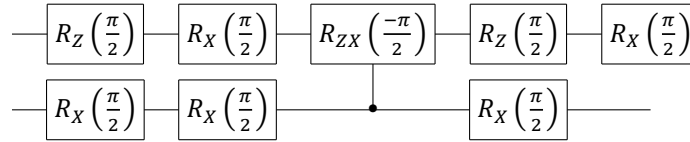
$$\begin{bmatrix} 0.4747 - 0.1569i & -0.4747 + 0.1569i & -0.4747 + 0.1569i & -0.4747 + 0.1569i \\ -0.4747 - 0.1569i & 0.4747 + 0.1569i & -0.4747 - 0.1569i & -0.4747 - 0.1569i \\ 0.1569 + 0.4747i & 0.1569 + 0.4747i & -0.1569 - 0.4747i & 0.1569 + 0.4747i \\ -0.1569 + 0.4747i & -0.1569 + 0.4747i & -0.1569 + 0.4747i & 0.1569 - 0.4747i \end{bmatrix}. \quad (5.30)$$

From this matrix expression it can easily be seen that our circuit gives the right result for the described input vectors.

Note that the final angle of the  $R_Z$  gate is inconsequential as it can only alter the relative phase of the basis states in which the last qubit is in state 1 instead of 0. As we perform the measurement directly after this operation it does not affect the outcomes.

If we replace the angle of the final  $R_Z$  operation to 0 then  $R_Z(0) = I$ . Using this and the above logic that the angle  $\theta$  of the final  $R_Z(\theta)$  gate is inconsequential we know that we can remove the final  $R_Z$  operation without changing the measurement outcome of the circuits. The alternative circuit resulting from this change is represented in Figure 5.13.

Figure 5.13: Alternative circuit for mimicking the amplitude amplification step of Grover's search with  $N = 4$  and  $t = 1$ .



This circuit can be worked out to the following matrix

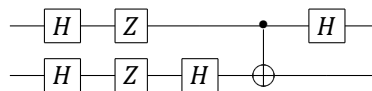
$$\begin{bmatrix} 0.5000 & -0.5000 & -0.5000 & -0.5000 \\ -0.5000 & 0.5000 & -0.5000 & -0.5000 \\ 0.5000i & 0.5000i & -0.5000i & 0.5000i \\ 0.5000i & 0.5000i & 0.5000i & -0.5000i \end{bmatrix}. \quad (5.31)$$

It can easily be seen that an operation with this matrix on the described input vectors gives the right solution. Notice as well that this is not what the theoretical matrix for the amplitude amplification step looks like. The theoretical matrix for this operation is

$$\begin{bmatrix} -0.5000 & 0.5000 & 0.5000 & 0.5000 \\ 0.5000 & -0.5000 & 0.5000 & 0.5000 \\ 0.5000 & 0.5000 & -0.5000 & 0.5000 \\ 0.5000 & 0.5000 & 0.5000 & -0.5000 \end{bmatrix}. \quad (5.32)$$

This matrix can be created by the circuit shown in Figure 5.14.

Figure 5.14: Theoretical decomposition of the amplitude amplification step of Grover's search algorithm for  $n = 2$



This decomposition of the amplitude amplification step is different from the one we gave in Section 5.11, for  $n = 2$  this decomposition is also known, and it is thinner. The theoretical decomposition of the

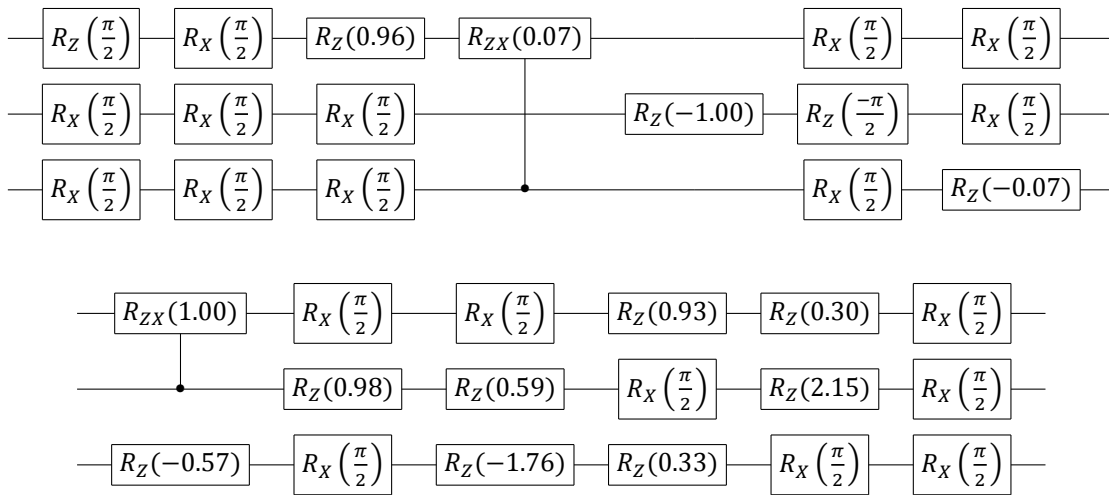
CNOT gate takes 6 layers to be decomposed in IBM native gates [Gok+20]; see Figure 5.3. Therefore, the circuit to perform the amplitude amplification step consists of 11 layers. This means that the circuit we found has a significantly lower quantum volume, but still performs the amplitude amplification step perfectly well.

Using our gradient-based machine learning approach we were also able to find a circuit that also perfectly mimics the amplitude amplification step for  $N = 4$  and  $t = 1$ .

### 5.3.2. $N = 8$ and $t = 2$

We are in the case  $t = \frac{N}{4}$  and so we know that Grover can solve this problem with probability 1. Using our PSO machine learning approach we found the circuit depicted in Figure 5.15 with depth  $D = 12$ .

Figure 5.15: Found amplitude amplification step for 3 qubits.

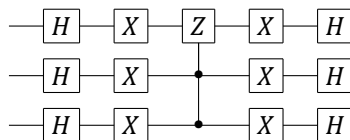


In 24 out of the 28 possible input states this circuit yields a success probability of approximately .80, where both states  $|i\rangle$  such that  $x_i = 1$  have a probability of approximately .40 to be measured. In the other four possible cases the probability of finding a state  $|i\rangle$  for which  $x_i = 1$  is approximately .45. In these four cases the probability of finding the state  $|i\rangle$  for which  $x_i = 1$ , is approximately .23 for both states which have  $x_i = 1$ .

Therefore the probability of solving the search problem using the amplitude amplification step found by our circuit is approximately .65.

The theoretical amplitude amplification step leads to the user solving the search problem with probability 1 using the circuit depicted in Figure 5.16.

Figure 5.16: Amplitude amplification step for  $n$  qubits.



The theoretical decomposition of the controlled-controlled-Z operation consists of 12 layers using at least 6 CNOT operations. The CNOT operations need to be decomposed, using 6 layers of native gates for the IBM computer [Gok+20]. Therefore the theoretical decomposition of this operation would require about 48 layers. From which we can conclude that even though our quantum circuit leads to a lower probability of finding the right state upon measurement, it does require a NISQ device with a



much smaller quantum volume.

## 5.4. Comparison of performance of PSO and gradient-based machine learning

This section will give a comparison of the performance of the PSO and gradient-based machine learning algorithms.

In the examples described above, we used both PSO and the gradient-based approach to learn over the same problems. The first difference we noticed is that for the simple example where we trained to find a circuit that mimics  $f_{\text{simple}}$ , the gradient-based approach required a larger batch size to learn over in order to consistently optimize the gradients. This difference became particularly clear as the number of qubits of the system grew. When we reached  $f_{\text{simple}}$  with 10 qubits the PSO approach was still able to optimize the parameters, where the gradient-based approach seemed unable to find negative gradients.

The other difference we noticed is that when learning a circuit to mimic the CNOT gate and the amplitude amplification step of Grover's search algorithm, the gradient-based approach needed more ansatzes to learn over, in order to find a well performing circuit. For some ansatzes for which we knew well performing angles were possible (and found by the PSO approach), the gradient-based approach was unable to optimize the parameters.

## 5.5. Overview of results

In this section we presented the results of running our machine learning approaches on several different problems.

We started by defining a small problem  $f_{\text{simple}}$ , which is designed in such a way that mimicking the function requires little circuit depth, but the amount of qubits used in the problem can grow arbitrarily. For  $n = 2$  we were able to find a circuit that perfectly mimics the chosen function, using both our PSO and gradient-based machine learning approach.

Using both PSO and gradient descent, we were able to find and perfectly optimize the right ansatz up to  $n = 5$ . We were able to learn the right angles and identify the right circuit using a much smaller batch to learn over than would be required to span the space.

When learning to mimic  $f_{\text{simple}}$  for  $n = 10$  qubits, our PSO approach was perfectly able to optimize the angles correctly and distinguish the right ansatz in a batch of ansatzes using only two quantum states to learn over (which is much smaller than the 1024 states minimally required to span the Hilbert space we are working in). For  $n = 10$  our gradient-based approach was no longer able to consistently optimize the gate parameters. This is in line with what is to be expected of gradient-based machine learning for quantum circuits, as the circuit size grows [McC+18] [Kha+19].

Another important aspect of our results when learning to find a circuit to mimic  $f_{\text{simple}}$ , is that we were able to find a circuit which is thinner than a theoretical decomposition would require and has the desired effect on all possible input states. This is due to choice of cost function, which only discriminates between the probability vector describing the probabilities of each basis state being found upon measurement.

Another interesting result we saw when learning quantum circuits for mimicking  $f_{\text{simple}}$ , is that if you define the batch to be the computational basis you run the risk of finding a circuit that performs perfectly well on the computational basis states, but potentially very bad on states that are in a superposition. This stands out as we do find the right result when learning over a much smaller amount of random quantum states.

Next, we used our machine learning approaches to mimic the behaviour of applying a single CNOT operation. In this case both the gradient-based and PSO approach performed well. We were also able to show that using our approach, in combination with a cost function that only discriminates over the outcome probabilities, we can find thinner circuits than would be theoretically required.

Finally, we used our machine learning approaches to find quantum circuits that can be used to mimic the behaviour of the amplitude amplification step of Grover's search algorithm.

For the problem with  $N = 4$  and the hamming weight  $t = 1$ , we can find a circuit that performs the am-

plitude amplification step perfectly for all possible input states. The circuit we found has a significantly lower quantum volume than the decomposition of the theoretical amplitude amplification step. We also used our machine learning approach to find a circuit to simulate the amplitude amplification step in the case  $N = 8$  and  $t = 2$ . Here we are again in the case where  $t = \frac{N}{4}$  and so we know that the amplitude amplification step as given by Grover's search algorithm leads to the right result with probability 1. We were able to find a quantum circuit with much lower depth (and therefore smaller quantum volume) but a probability of approximately  $\frac{2}{3}$  to find the right result.



# 6

## Discussion

In this thesis, we presented an approach that can be used to find circuits designed to make optimal use of NISQ devices, by creating a hardware-centric quantum circuit.

We identified the aspects of modern quantum computers that have a large impact on its performance and used those to restrict the allowed ansatzes of the quantum circuits.

In particular, we require that any quantum circuit we create only uses quantum gates which are native to the chosen device.

We also restrict two-qubit gates to be applied only between qubits which are connected on the hardware. By implementing this requirement, we are in control of the true depth of the created quantum circuit, as there is no extra depth hidden in the decomposition from used quantum gates to native quantum gates.

A third aspect of NISQ devices, that has a large impact on its performance, is noise. Our approach makes sure that we can always choose our circuit to be thin enough, such that it can be run without noise having a large effect on the final measurement outcome. We do this by allowing the user to choose the depth of the circuit and the number of qubits used. As long as these choices are such that the maximum quantum volume of the chosen computer is not surpassed, we know the circuit can be reliably run.

In our machine learning approach, we learn over an input batch of quantum states and the associated desired probabilities of measuring each basis state after performing the desired function. We then calculate the mean squared error between the probability vectors resulting from applying our created quantum circuit and the desired probability vectors. This cost function can be evaluated relatively efficiently, but is still exponential in terms of the number of qubits. This exponential size of the cost function is unavoidable, as long as we use a classical computer to calculate the costs of the quantum circuit.

Since we learn over an input batch of quantum states and the desired probabilities, our optimization method automatically takes noise into account when we run our approach using an actual quantum computer.

In this project we subsequently presented our own implementation of a quantum simulator, to be able to evaluate the cost associated with our created circuits. In designing this quantum simulator we exploited that the quantum circuits, represented by the valid ansatzes considered, always consist of applying single- and two-qubit gates to the system. This means that the unitary applied to the quantum state vector is the Kronecker product of many smaller matrices. We then used the Algorithm 993 [Fac19] to efficiently simulate applying this unitary to the quantum state vector. Thereby, we avoid the extra memory and computational costs associated to calculating this Kronecker product.

We also added extra cases and structure to allow for two-qubit operations to be performed between qubits which are adjacent on the hardware but not in qubit order.

After having defined the aspects of NISQ devices to be taken into account when creating a valid ansatz, as well as a cost function and a simulator to be able to evaluate the cost function, we use machine learning to optimize the parameters of the quantum gates in the ansatz. We have presented two different machine learning methods that can be used to optimize the angles of the parameterized quantum gates. The first method was a gradient-free machine learning approach, where we made use of particle swarm

optimization. The second machine learning method we used was gradient-based, where we made use of backpropagation in combination with the method from [Sch+19] to calculate the gradients of quantum gates.

Both machine learning approaches presented can be used in combination with a quantum computer or a quantum simulator. The gradient-based approach, however, is designed specifically to enable the gradients of a quantum operation to be evaluated directly on a quantum computer. Nonetheless this approach can still be used in combination with a quantum simulator.

We then used both machine learning approaches to learn over several example problems and compared their results. The parameter optimization approaches performed equally well for smaller problem sizes. Specifically for finding a circuit to mimic the function  $f_{\text{simple}}$ , which required a circuit of small depth, the approaches both performed well when learning the circuit for a small number of qubits. As the amount of qubits got larger, the PSO approach performed more consistently.

As the depth of the circuits used to mimic certain functions got larger, the PSO approach also started performing better, relative to the gradient-based approach. This can be explained by the fact that finding non-zero gradients in a quantum circuit grows less likely as the size of the circuit grows [McC+18]. We have shown that our approaches can find quantum circuits which create the desired probability vectors for all possible input states, while learning over a batch size much smaller than would be required to span the space. This is an interesting result, as any quantum state vector and its associated probability vector have an exponential amount of elements with respect to the number of qubits, so by significantly reducing the amount of quantum vectors to learn over we save memory usage and computational steps.

We have also shown that by defining the cost function over the probabilities of measuring each basis state upon measurement, we can find smaller quantum circuits than would be required to create the correct output quantum state. This can save valuable circuit depth and no information measurable in the chosen basis is lost.

## 6.1. Recommendations for further research

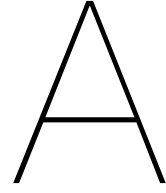
In this project we presented a machine learning based optimization technique to optimize the parameters of the gates in a quantum circuit, in order to mimic a chosen cost function. The method we presented is designed to be used on smaller circuit sizes.

The next step of this research should be to extend these methods to enable them to be used for generating quantum circuits of larger quantum volume.

# Bibliography

- [CC68] C.K.Chow and C.N.Liu. “Approximating Discrete Probability Distributions with Dependence Trees”. In: *IEEE transactions of information theory* (1968). URL: <https://ieeexplore-ieee-org.tudelft.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=1054142>.
- [Sho94] Peter W. Shor. “Algorithms for Quantum Computation: Discrete Logarithms and Factoring”. In: (1994). URL: <https://doi.org/10.1109/SFCS.1994.365700>.
- [Gro96] Lov K. Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of 28<sup>th</sup> ACM STOC* (1996), pp. 212–219. URL: <https://arxiv.org/pdf/quant-ph/9605043.pdf>.
- [MN01] Christopher Moore and Martin Nilsson. “Parallel Quantum Computation and Quantum Codes”. In: *SIAM Journal on Computing*, vol. 31, no. 3 (2001). URL: <https://arxiv.org/pdf/quant-ph/9808027.pdf>.
- [MV05] Mikka Möttönen and Juha Vartiainen. “Decompositions of general quantum gates”. In: (2005). URL: <https://arxiv.org/pdf/quant-ph/0504100.pdf>.
- [SBM06] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. “Synthesis of Quantum-Logic Circuits”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2006). URL: <https://ieeexplore-ieee-org.tudelft.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=1629135>.
- [NC16] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information. 10th Anniversary Edition*. 4th ed. Cambridge University Press, 2016. ISBN: 9781107002173.
- [Ber+18] Ville Bergholm et al. “PennyLane: Automatic differentiation of hybrid quantum-classical computations”. In: (2018). URL: <https://arxiv.org/abs/1811.04968>.
- [BKM18] Adi Botea, Akihiro Kishimoto, and Radu Marinescu. “On the Complexity of Quantum Circuit Compilation”. In: *The Eleventh International Symposium on Combinatorial Search* (2018). URL: <https://www.aaai.org/ocs/index.php/SOCS/SOCS18/paper/viewFile/17959/17110>.
- [LW18] Jin-Guo Liu and Lei Wang. “Differentiable Learning of Quantum Circuit Born Machine”. In: (2018).
- [McC+18] J.R. McClean et al. “Barren plateaus in quantum neural network training landscapes”. In: *Nature Communications* (2018). DOI: DOI : 10 . 1038 / s41467 - 018 - 0709 - 4. URL: <https://www-nature-com.tudelft.idm.oclc.org/articles/s41467-018-07090-4>.
- [Mir18] Lester James V. Miranda. “PySwarms: a research toolkit for Particle Swarm Optimization in Python”. In: *The Journal of Open Sourced Software* (2018). DOI: <https://doi.org/10.21105/joss.00433>.
- [Mol+18] Nikolaj Moll et al. “Quantum optimization using variational algorithms on near-term quantum devices”. In: *Quantum Science and Technology, Vol.3, Numb. 3* (2018). DOI: <https://arxiv.org/abs/1710.01022>.
- [Pre18] John Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum Journal* (2018). URL: <https://arxiv.org/pdf/1801.00862.pdf>.
- [Ben+19a] Marcello Benedetti et al. “A generative modelling approach for benchmark and training shallow quantum circuits”. In: *npj Quantum Information*(2019)5:45 (2019). URL: <https://doi.org/10.1038/s41534-019-0157-8>.
- [Ben+19b] Marcello Benedetti et al. “Parameterized quantum circuits as machine learning models”. In: (2019). URL: <https://arxiv.org/abs/1906.07682>.

- [Dav+19] Marc Grau Davis et al. "Heuristics for Quantum Compiling with a Continuous Gate Set". In: (2019). URL: <https://arxiv.org/pdf/1912.02727.pdf>.
- [Fac19] Paul L. Fackler. "Algorithm 993: Efficient Computation with Kronecker Products". In: *ACM Transactions on Mathematical Software* (2019). URL: <https://dl-acm-org.tudelft.idm.oclc.org/doi/abs/10.1145/3291041>.
- [Kha+19] Sumeet Khatri et al. "Quantum-assisted quantum compiling". In: *Quantum* (2019). DOI: <https://doi.org/10.22331/q-2019-05-13-140>.
- [Mur+19] Prakash Murali et al. "Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights". In: *ISCA '19: Proceedings of the 46th International Symposium on Computer Architecture* (2019). URL: <https://arxiv.org/abs/1905.11349>.
- [Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *journal* (2019). URL: <https://arxiv.org/pdf/1912.01703.pdf>.
- [Sch+19] Maria Schuld et al. "Evaluating analytic gradients on quantum hardware". In: *Physical Review*, vol.99 Iss 3 (2019). DOI: <https://doi-org.tudelft.idm.oclc.org/10.1103/PhysRevA.99.032331>.
- [Wol19] Ronald de Wolf. *Quantum Computing: Lecture Notes. Lecture notes for the MasterMath course Quantum Computing 2019*. 2019.
- [Zhu+19] Zhu et al. "Training of Quantum Circuits on a Hybrid Quantum Computer". In: *Science Advances* (2019). URL: <https://arxiv.org/abs/1812.08862>.
- [Abr+20] Deanna Abrams et al. "Implementation of XY entangling gates with a single calibrated pulse". In: *nature electronics* (2020). DOI: <https://doi.org/10.1038/s41928-020-00498-1>.
- [Ale+20] Thomas Alexander et al. "Qiskit pulse: programming quantum computers through the cloud with pulses". In: *Quantum Science and Technology* (2020). DOI: <https://doi.org/10.1088/2058-9565/aba404>.
- [Cin+20] Lukasz Cincio et al. "Machine Learning of Noise-Resilient Quantum Circuits". In: *American Physical Society March Meeting* (2020). URL: <https://arxiv.org/abs/2007.01210>.
- [Gok+20] Pranav Gokhale et al. "Optimized Quantum Compilation for Near-Term Algorithms with OpenPulse". In: (2020). URL: <https://arxiv.org/abs/2004.11205>.
- [HGB20] Maxwell Philip Henderson, Jarred Gallina, and Michael Brett. "Methods for Accelerating Geospatial Data Processing using Quantum Computers". In: (2020). URL: [https://www.researchgate.net/publication/340500470\\_Methods\\_for\\_Accelerating\\_Geospatial\\_Data\\_Processing\\_Using\\_Quantum\\_Computers](https://www.researchgate.net/publication/340500470_Methods_for_Accelerating_Geospatial_Data_Processing_Using_Quantum_Computers).
- [Nan20] Giacomo Nannicini. "An introduction to Quantum Computing without the Physics". In: *SIAM review* (2020). DOI: DOI : 10.1137/18M1170650. URL: <https://epubs-siam-org.tudelft.idm.oclc.org/doi/pdf/10.1137/18M1170650>.
- [Ale21] Robert Wille Alexandru Paler Alwin Zulehner. "NISQ circuit compilation is the travelling salesman problem on a torus". In: (2021). URL: <https://arxiv.org/pdf/1806.07241.pdf>.
- [IBMa] IBM. *List of IBM quantum computers and structures*. URL: <https://quantum-computing.ibm.com/>. Accessed: 10.04.2021.
- [IBMb] IBM. *Page shows the IBM RZX-gate and notes it comes from the implementation of the cross resonance gate*. URL: <https://qiskit.org/documentation/stubs/qiskit.circuit.library.RZXGate.html>. (accessed: 08.04.2021).
- [IBMc] IBM. *Webpage with the IBM quantum roadmap*. URL: <https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/>. Accessed: 10.12.2020.
- [Ver+] Pieter Vermaas et al. *TU Delft Quantum Vision Team Report on Quantum Computing*. URL: <https://www.tudelft.nl/over-tu-delft/strategie/vision-teams>. (Not yet published, expected in August 2021).



# Quantum computing

## A.1. Qubits

Quantum computers are built using qubits. Qubits are the quantum equivalent to classical bits and as such they have specific quantum properties. In this chapter we will introduce the properties of qubits necessary to understand the basic ideas of quantum computing.

### A.1.1. Single qubit

A qubit is the basic unit of quantum information. A single qubit is a two-state quantum system and can be expressed as a 2 dimensional vector in complex Hilbert space. In quantum mechanics we express the state of a physical system using bra-ket notation. For a single qubit  $q_0$  we can write the state as

$$|q_0\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix}, \quad (\text{A.1})$$

with  $\alpha_0, \alpha_1 \in \mathbb{C}$ . Since any quantum mechanical object lives in Hilbert space and has a normalized inner product we must have that  $|\alpha_0|^2 + |\alpha_1|^2 = 1$ .

Therefore a qubit state can be expressed as a linear combination of the two basis states  $|0\rangle$  and  $|1\rangle$ . The basis states  $|0\rangle$  and  $|1\rangle$  form the computational basis states of the space  $\mathbb{C}^2$ . We can also write  $|q_0\rangle$  as

$$|q_0\rangle = \alpha_0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \alpha_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \alpha_0 |0\rangle + \alpha_1 |1\rangle, \quad (\text{A.2})$$

with  $\alpha_i \in \mathbb{C}$  and  $\sum_{i=0}^1 |\alpha_i|^2 = 1$  as before. Since  $|\alpha_0|^2 + |\alpha_1|^2 = 1$  we can rewrite Equation (A.2) as follows:

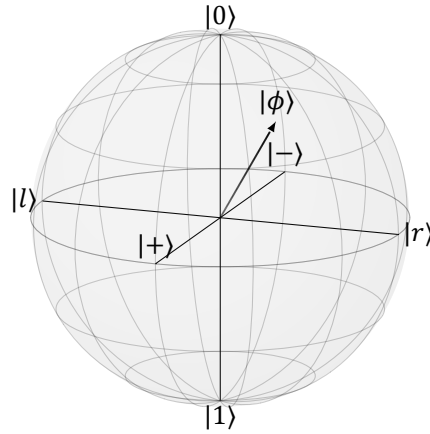
$$|q_0\rangle = e^{i\gamma} \left( \cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) |1\rangle \right). \quad (\text{A.3})$$

The factor  $e^{i\gamma}$  does not effect an observation of the qubit; see Appendix A.2.3. Therefore we can write

$$|q_0\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) |1\rangle. \quad (\text{A.4})$$

The equation above can be used to understand the Bloch sphere representation of a single qubit system. This representation is useful for understanding the single qubit rotation gates  $R_X(\theta)$ ,  $R_Y(\theta)$  and  $R_Z(\theta)$ , which are often used as native gates on physical quantum computers; see Section 1.2.2. The Bloch sphere is given in Figure A.1.



Figure A.1: Bloch sphere representation of a single qubit state  $|\phi\rangle$ .

The states on the x-axis are

$$|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \quad (\text{A.5})$$

$$|-\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle). \quad (\text{A.6})$$

The states on the y-axis are

$$|r\rangle = \frac{1}{\sqrt{2}} (|0\rangle + i|1\rangle) \quad (\text{A.7})$$

$$|l\rangle = \frac{1}{\sqrt{2}} (|0\rangle - i|1\rangle). \quad (\text{A.8})$$

The states on the z-axis are the computational basis state, from this it can be seen why the computational basis is sometimes referred to as the z-basis.

From the Bloch sphere representation it can also be seen that any single-qubit quantum gate can be performed by performing a total of three rotations around the axes [NC16].

### A.1.2. Multiple qubit states

Now consider a two-qubit system and call  $|\phi\rangle$  the state of our quantum system. Then the state  $|\phi\rangle$  can be written as

$$|\phi\rangle = \alpha_0 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \alpha_1 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle \quad (\text{A.9})$$

with  $\alpha_i \in \mathbb{C}$  and  $\sum_{i=0}^3 |\alpha_i|^2 = 1$ . The basis states can also be given in terms of decimal instead of binary numbers, this gives

$$|\phi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + \alpha_2 |2\rangle + \alpha_3 |3\rangle, \quad (\text{A.10})$$

where the  $\alpha_i$  are as before.

The benefit of using the binary representation of the basis states, is that you can immediately read out which qubit is in the state  $|0\rangle$  and which is in the state  $|1\rangle$ . However, as the amount of qubits and therefore the dimensions of the Hilbert space increase, it is more practical to use decimal numbers to indicate the basis states.

Combining two separate qubits  $|q_0\rangle$  and  $|q_1\rangle$  to form a single qubit system  $|\phi\rangle$ , can be seen as taking the tensor product

$$|\phi\rangle = |q_0\rangle \otimes |q_1\rangle. \quad (\text{A.11})$$

Similarly combining two quantum states consisting of multiple qubits can be seen as taking the tensor product between the two. As such an  $n$  qubit quantum system  $|\phi\rangle$  is an element of  $\mathbb{C}^{2^n}$  dimensional complex Hilbert space.

A quantum state  $|\phi\rangle$  consisting of  $n$  qubits can be written as

$$|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \quad (\text{A.12})$$

with  $\alpha_i \in \mathbb{C}$  and  $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$ . Note that as before we express our quantum state as a sum of computational basis states  $|i\rangle$ , the computational basis is equal to the standard basis.

### A.1.3. Superposition

As seen in Equation (A.12), a system of qubits can be expressed as a linear combination of its basis states.

As long as there is more than one basis state with non-zero amplitude we say that the quantum system is in a superposition of these basis states.

### A.1.4. Qubit order and indentation

In order to avoid confusion it is important we choose a convention of which qubit to give which index when working with multiple qubit systems. When working with a system of  $n$  qubits we use the following indexation

$$|q_0\rangle \otimes |q_1\rangle \otimes \cdots \otimes |q_{n-1}\rangle. \quad (\text{A.13})$$

From left to right the first qubit we encounter is the  $0^{\text{th}}$  qubit and the last qubit is the  $n - 1^{\text{th}}$  qubit. This convention of qubit ordering is known as the little-endian convention [Nan20].

### A.1.5. Measurement of a qubit

As described in the last section, a qubit can be in a superposition of several basis states. We cannot, however, measure this superposition. Upon measurement we find one of the basis states that the qubit was in a superposition of.

To make this more precise, consider the measurement of the multiple qubit system expressed in Equation (A.12). Since upon measurement we can only find a single basis state, we must first define the basis we are working in. The most common choice of basis is the computational basis.

If we measure this qubit in the computational basis we will find the state  $|i\rangle$  with probability  $|\alpha_i|^2$ , we cannot, however, recover the entire state of system  $|\phi\rangle$ .

In fact, after measuring of the state  $|\phi\rangle$  and finding a certain basis state  $|i\rangle$  the original state of  $|\phi\rangle$  is lost. Upon measurement of a qubit system  $|\phi\rangle$  the system collapses completely to the basis state found upon measurement and now

$$|\phi\rangle = \frac{\alpha_i}{\|\alpha_i\|_2} |i\rangle. \quad (\text{A.14})$$

This implies that measuring the qubits may actually change the state they are in and information might be lost. This also means that we cannot measure the same quantum state multiple times in order to find an estimation for the values  $|\alpha_i|^2$ .

### A.1.6. Entanglement

Consider three separate qubits  $q_0, q_1, q_2$ , we can combine them to form one quantum state  $|\phi\rangle$ . This can be written as taking the tensor product

$$\begin{aligned} |\phi\rangle &= |q_0\rangle \otimes |q_1\rangle \otimes |q_2\rangle = \begin{bmatrix} \alpha_0 \\ \beta_0 \end{bmatrix} \otimes \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} \otimes \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} \\ &= \gamma_0 |000\rangle + \gamma_1 |001\rangle + \gamma_2 |010\rangle + \gamma_3 |011\rangle \\ &\quad + \gamma_4 |100\rangle + \gamma_5 |101\rangle + \gamma_6 |110\rangle + \gamma_7 |111\rangle. \end{aligned}$$

with  $\alpha_i, \beta_i \in \mathbb{C}$  and  $\sum_{i=0}^2 |\gamma_i|^2 = 1$ . Here the state  $|\phi\rangle$  is the Kronecker product of the states associated with the three separate qubits. For notational simplicity we can write  $|q_0\rangle |q_1\rangle |q_2\rangle = |q_0\rangle \otimes |q_1\rangle \otimes |q_2\rangle$ . However, we cannot always write a quantum state  $|\phi\rangle$  consisting of more than one qubit, as the Kronecker product of the separate qubit states. This leads us to the concept of entanglement. Consider the following two-qubit state

$$|\phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \quad (\text{A.15})$$

This quantum state  $|\phi\rangle$  consists of two qubits, but cannot be written as the Kronecker product of two quantum states<sup>1</sup>. When this is the case we say that the qubits are entangled.

To see what the implications of entanglement are, consider measuring the second qubit. You will find state  $|0\rangle$  or  $|1\rangle$  with equal probability. Assume we measure the second qubit and find state  $|0\rangle$ . As described in Section A.1.5 this means that the second qubit is now in the state  $|0\rangle$  and so if we look at the two-qubit system as a whole, we know that only the basis states in which the second qubit has state  $|0\rangle$  can still exist. In particular, after having performed this measurement the two qubit system  $|\phi\rangle$  will be

$$|\phi\rangle = |00\rangle. \quad (\text{A.16})$$

Now if we were to measure the first qubit we will find it in the state  $|0\rangle$  with probability 1. We can see that upon measuring of the second qubit, we have also altered the state of the first qubit as they were entangled.

### A.1.7. Bra-ket notation

In quantum computing bra-ket notation is used to denote vectors in Hilbert space. A ket  $|\phi\rangle$  is used to represent the state of a qubit system. A bra  $\langle\psi|$  represents the Hermitian conjugate of  $|\psi\rangle$ . The expression  $\langle\psi|\phi\rangle$  represents an inner product between the two states.

## A.2. Quantum Gates

In order for us to make use of qubits we need to be able to perform operations on them. Quantum mechanics only allows for linear operations to be performed [Wol19] and so we can consider an operation performed on a qubit state  $|\phi\rangle$  to be a matrix-vector multiplication. Let

$$|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \quad (\text{A.17})$$

and  $U$  be the quantum operation we wish to perform, we then get

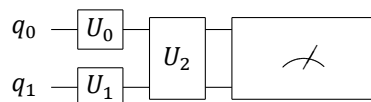
$$|\phi'\rangle = U|\phi\rangle = U \sum_{i=0}^{2^n-1} \alpha_i |i\rangle = \sum_{i=0}^{2^n-1} \alpha_i U|i\rangle. \quad (\text{A.18})$$

Since a system of  $n$  qubits can be expressed as a vector in  $2^n$  dimensional complex Hilbert space, the matrix of the quantum gate acting on  $n$  qubits can be expressed as an  $2^n \times 2^n$  dimensional complex

<sup>1</sup>The proof is simple. Assume  $|\phi\rangle = (\alpha_0 |0\rangle + \alpha_1 |1\rangle) \otimes (\beta_0 |0\rangle + \beta_1 |1\rangle)$ , then we must have that  $\alpha_0 \neq 0$  and  $\beta_1 \neq 0$  as  $\alpha_0 \beta_0 = \frac{1}{\sqrt{2}}$  and  $\alpha_1 \beta_1 = \frac{1}{\sqrt{2}}$ . Therefore  $\alpha_0 \beta_1 \neq 0$ , this is a contradiction as then  $|\phi\rangle$  must have a nonzero amplitude for the state  $|01\rangle$ .

matrix. Since the unitarity of a quantum state needs to be preserved we have that the matrix representing the quantum operation  $U$  must be a unitary matrix.<sup>2</sup> An operation  $U$  performed on one or multiple qubits is often referred to as a quantum gate. When multiple quantum gates are subsequently applied to a system of qubits we call this a quantum circuit.

Figure A.2: A quantum circuit consisting of the gates  $U_0$ ,  $U_1$  and  $U_2$  followed by a measurement.



The figure above shows a quantum circuit where we first apply the quantum gate  $U_0$  to the first qubit  $q_0$  and simultaneously apply  $U_1$  to  $q_1$ . We subsequently apply the two-qubit gate  $U_2$  to  $q_0$  and  $q_1$ . The final operation portrayed in the circuit is a measurement of the quantum state resulting from applying the described operations to  $q_0$  and  $q_1$ .

Let  $|\phi\rangle$  be the quantum state  $q_0$  and  $q_1$  comprise at the start of the circuit. Then we can also write the state resulting from applying the quantum gates, as described in Figure A.2, by  $U_2(U_0 \otimes U_1)|\phi\rangle$ . It is common to represent a quantum circuit in the form of such a figure, as it gives a good overview of which operations are applied to which qubits in what order. See Appendix C.4 for an overview of commonly used symbols when representing quantum circuits by such figures.

### A.2.1. Quantum operations as matrix multiplications

As stated before, performing an operation on a quantum state can be seen as a matrix-vector multiplication. As an example consider performing a Hadamard gate  $H$  as defined in Appendix C on a single qubit state

$$|\phi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix}. \quad (\text{A.19})$$

This can be expressed as

$$H|\phi\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} \alpha_0 + \alpha_1 \\ \alpha_0 - \alpha_1 \end{bmatrix}. \quad (\text{A.20})$$

We can generalise this idea of writing quantum operations as matrix vector multiplications to a system with multiple qubits.

When applying a quantum operation  $U$  on a state  $|\phi\rangle$  consisting of multiple qubits, usually we must first create the correct matrix expression for the quantum operation  $U$ . This is because most quantum gates are single- or two-qubit gates, so the quantum operation  $U$  is built up from single- and two-qubit operations.

Say we have an  $n$  qubit state  $|\phi\rangle$  and we wish to apply a single-qubit gate  $u_i$  to each qubit  $i$ . This can be seen as performing the following operation  $u_0 \otimes u_1 \otimes \dots \otimes u_{n-1} |\phi\rangle$ . To write this as a matrix-vector multiplication, we must first create the matrix  $U = u_0 \otimes u_1 \otimes \dots \otimes u_{n-1}$  by performing  $n - 1$  Kronecker products.

Notice that it is rather expensive to simulate such operations classically, as we need to perform  $n - 1$  Kronecker products to create a  $2^n \times 2^n$  size matrix.

### A.2.2. Universal gate set

The universal gate set is the set of quantum operations required to be able to perform any arbitrary quantum gate, up to an arbitrary accuracy. Since all operations performed on a quantum computer

<sup>2</sup>Since all operations performed on a quantum state are unitary we have that quantum computing is invertible. This follows directly from the definition of unitary matrices. A matrix is unitary if  $U^\dagger U = U U^\dagger = I$  or equivalently  $U^* U = U U^* = I$ , therefore any quantum operation  $U$  has an inverse which is also a unitary operation  $U^\dagger$ . This does not hold for classical computing.

must be unitary, a set of gates is universal for quantum computation if any unitary operation can be approximated to arbitrary accuracy using only gates from the given set [NC16].

It can be shown that any unitary operation acting on a  $d$  dimensional Hilbert space can be decomposed as the product of two-level unitary matrices. A two-level unitary matrix is a matrix that acts non-trivially on two or less vector components.

It can also be shown that by considering the single qubit operations and the CNOT-gate we can build up all two-level unitary matrices. And so we can conclude that the set of single qubit operations and the CNOT-gate are universal. This universal gate set is infinitely large and can be expressed as  $\{R_X(\alpha), R_Y(\beta), R_Z(\gamma), \text{CNOT}\}$  where  $\alpha, \beta, \gamma \in [0, 2\pi]$ . Notice that the three rotation matrices  $R_X$ ,  $R_Y$  and  $R_Z$  can be used to decompose any arbitrary single qubit gate, the matrix expressions of the described quantum gates can be found in Appendix C.

Another well known and often used universal gate set consists of the Hadamard gate, the phase gate, CNOT and the  $\pi/8$  gates for  $n \in \{x, y, z\}$ . This gate set is known as the standard set and is a finite universal gate set.

There are many different possible universal gate sets [NC16]. Any quantum computer that can natively implement all the gates in a universal gate set can be used to perform any quantum operation. Note, however, that for most unitary operations we do not have a good recipe to decompose it in terms of the gates of a universal gate set [BKM18] [NC16]. In fact, you can prove that for any finite universal gate set there will always be some universal operations that require an exponential number of quantum gates to be approximated up to precision  $\epsilon > 0$  [NC16]. Generally speaking the problem of quantum compilation is NP-complete [BKM18].

### A.2.3. Phase of a quantum gate

Any  $n$  qubit quantum state can be expressed as

$$|\phi\rangle = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle. \quad (\text{A.21})$$

Consider a phase  $e^{\frac{i\theta\pi}{2}}$ , then define the  $\beta_j$  such that  $\alpha_j = e^{\frac{i\theta\pi}{2}} \beta_j$ , this gives

$$|\phi\rangle = e^{\frac{i\theta\pi}{2}} \sum_{j=0}^{2^n-1} \beta_j |j\rangle. \quad (\text{A.22})$$

We can always find such  $\beta_j$ , where  $\sum |\beta_j|^2 = 1$  as  $|e^{\frac{i\theta\pi}{2}}| = 1$ . Notice that

$$|\phi'\rangle = \sum_{j=0}^{2^n-1} \beta_j |j\rangle \quad (\text{A.23})$$

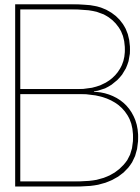
is also a valid quantum state. Furthermore notice that the two states  $|\phi\rangle$  and  $|\phi'\rangle$  are indistinguishable when a measurement is applied to them, as  $|\alpha_j| = |e^{\frac{i\theta\pi}{2}} \beta_j| = |\beta_j|$ . Applying the same unitary operation on both states  $|\phi\rangle$  and  $|\phi'\rangle$ , also leads to indistinguishable quantum states

$$U|\phi\rangle = \sum_{j=0}^{2^n-1} U\alpha_j |j\rangle = \sum_{j=0}^{2^n-1} \alpha'_j |j\rangle \quad (\text{A.24})$$

$$U|\phi'\rangle = \sum_{j=0}^{2^n-1} U\beta_j |j\rangle = \sum_{j=0}^{2^n-1} \beta'_j |j\rangle. \quad (\text{A.25})$$

The states  $U|\phi\rangle$  and  $U|\phi'\rangle$  are indistinguishable as we have  $U|\phi\rangle = e^{i\frac{\theta\pi}{2}} U|\phi'\rangle$  so again both quantum states give the same probabilities of each basis state being measured.

From this it can be seen that two quantum gates which are the same up to a phase shift can be seen as the same operation. Similarly, two quantum states which are the same up to a phase shift can be seen as the same state.



# The quantum circuit generator

This section is written to give the reader some insight in the code created to implement the approaches described in this project.<sup>1</sup>

The quantum generator takes as input the operation we wish to mimic together with other parameters specifying the circuit properties and which type of machine learning we want to use, and outputs the optimized circuit for the given depth and computer.

The quantum circuit generator consists of three layers which build on each other, the Library B.1, the Engine B.2 and the ML layer B.3.

## B.1. Library

The first layer of the program is the Library layer. In the library we have several files which translate technical properties of certain computers to PyTorch objects, which we are able to use in our optimization and learning steps. The library also contains the list of distances specified for our purpose.

### B.1.1. connectivity.py

The connectivity file contains information on the connectivity of several Rigetti and IBM quantum computers. In this file the connectivity of the computers is expressed as a function of the amount of qubits required. The user can specify how many qubits they are interested in using and the function returns a PyTorch object representing which qubits are connected on the hardware.

### B.1.2. native\_gates.py

This file contains a list of the gates natively implemented in the quantum computers considered. The gates are expressed in terms of their matrix representation, for use with the Kronx simulator, and in terms of their PennyLane representation.

### B.1.3. ibm.py

This file combines the connectivity and native gate properties for the IBM computers and ensures that, when considering IBM computers, we only use IBM native gates and hardware structures.

### B.1.4. rigetti.py

This file combines the connectivity and native gate properties for the Rigetti computers and ensures that for Rigetti computers we only use Rigetti native gates and hardware structures.

### B.1.5. backends.py

This file contains functions which can be used to get the native gate sets in PennyLane gates or the Kronx matrix operation and quantum computing properties for the chosen computer necessary to run the ML approach. The file also contains a function which returns which native gates are parameterized for

---

<sup>1</sup>If you are interested in obtaining the code used, please contact me at [M.A.Schalkers@tudelft.nl](mailto:M.A.Schalkers@tudelft.nl).

each quantum computer.

### **B.1.6. distances.py**

In the `distances` file we have a function named `distance`. This returns the average of the distance between the probability vector associated with measuring each basis state of the quantum states outputted by our learned circuit and the desired probability vector.

The user can choose between several distances. The standard distance to use corresponds to the mean squared error of the associated probability vectors of each measurement outcome, as described in Section 2.4.

## **B.2. Engine**

The engine layer contains the functions necessary to run the steps of the machine learning files. In this layer we prepare the data, the ansatzes and the angles to optimize over.

### **B.2.1. circuit\_simulator.py**

This file contains a function which returns a function that takes a quantum state as input and acts on it as a quantum computer would. The user can choose to use our Kronx quantum simulator to get the quantum state created by our circuit. The user can also choose to create the function using PennyLane, in which case the probability vector of measuring the basis states would be the output of the created function.

### **B.2.2. kronx.py**

This file contains the functions and operations necessary to apply the Kronx function as described in Chapter 3.

### **B.2.3. operations.py**

This file contains a list of functions that can be used to create the data to learn over.

### **B.2.4. order\_creator.py**

This file creates a function that randomly creates a valid ansatz given the quantum computer properties and desired qubit number and gate depth.

### **B.2.5. thetas\_creator.py**

This file consists of the function that creates a vector of length equal to the amount of parameterized gates. Simultaneously a tensor is created which keeps track of which parameter index is connected to which quantum gate. This file makes use of functions from the `backends.py` file to ensure we do not optimize over an unnecessary number of parameters.

### **B.2.6. circuit\_creator.py**

This file translates the created ansatz in the `order_creator.py` file to a circuit which can be directly used by PennyLane or our Kronx quantum simulator.

## **B.3. ML**

The final layer of our code consists of the machine learning section. The files in this folder make use of the Library and Engine layers to be able to run the quantum circuit generator. This layer contains the file `runner.py` which should be evoked by the user to use our approach on a chosen function. In this folder we also keep a `results` folder in which all the results of running the circuits are automatically stored if this option is turned on by the user.

### **B.3.1. gradient.py**

This file contains our implementation of the gradient-based machine learning approach as described in Section 4.4.

**B.3.2. PSO.py**

This file contains our implementation of the Particle Swarm Optimization approach for optimizing the parameters of a quantum circuit as described in Section 4.3.

**B.3.3. qcg.py**

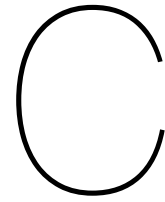
This file can be used to potentially combine the gradient-based and non-gradient based machine learning approach in several configurations. In this file the data to learn over is also created and the ansatzes are created.

**B.3.4. runner.py**

The runner file is the file directly run by the user. At the top of the file the user can specify the computer they want to use, as well as which function they want to learn a circuit for and the amount of qubits and circuit depth. Here the user can also specify if they want to save the result in the Results folder. The user can also choose if they want to use the PSO or gradient-based machine learning approach to optimize the gradients as well as how many different ansatzes they want to create. This is also the file where the user can specify the swarm size and number of iterations in case of PSO and the number of epochs in case of gradient-based learning.







# Quantum gates

In this appendix we provide lists of commonly used quantum gates. Note that since every unitary matrix can be a quantum gate, this list cannot contain all quantum gates. It does, however, contain all the quantum gates which are commonly used and named.

In this last section we give an overview of the commonly used symbols in the visual representation of a quantum circuit.

## C.1. Single-qubit gates

Symbol	Name	Matrix	Description	Comments
$I$	Identity	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	This gate does not change the state of the qubit	One of the four Pauli matrices
$X$	Pauli-X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	This gate is the quantum version of the classical not gate	One of the four Pauli matrices
$Y$	Pauli-Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$Y = iXZ$	One of the four Pauli matrices
$Z$	Pauli-Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	This gate shifts the phase of the $ 1\rangle$ state	One of the four Pauli matrices
$H$	Hadamard	$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$	Spreads the amplitude of the one or zero state over both	This operation is the equivalent of a two by two QFT
$R_X(\theta)$	Rotate X	$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i \sin\left(\frac{\theta}{2}\right) \\ -i \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$	This gate rotates a qubit around the X-axis. See Bloch sphere; Figure A.1	

$R_Y(\theta)$	Rotate Y	$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$	This gate rotates a qubit around the X-axis. See Bloch sphere; Figure A.1	
$R_Z(\theta)$	Rotate Z	$\begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$	This gate rotates a qubit around the X-axis. See Bloch sphere; Figure A.1	
$S$	S gate	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	This gate changes the phase of the $ 1\rangle$ state	$S = RZ(\pi)$
$T$	T gate	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$	This gate changes the phase of the $ 1\rangle$ state	$T = \pi/8$ and $T = \sqrt{S} = R_Z\left(\frac{\pi}{2}\right)$
$\pi/8$	$\pi/8$ gate	$e^{i\frac{\pi}{8}} \begin{bmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{bmatrix}$	This gate changes the phase of the $ 1\rangle$ state	$\pi/8$ and $\pi/8 = \sqrt{S} = R_Z\left(\frac{\pi}{2}\right)$
$R(\phi)$	Phase shift gate	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$	This gate changes the phase of the $ 1\rangle$ state	A generalisation of the $S$ gate
$R_k(\phi)$	$R_K$ gate	$\begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix}$	This gate changes the phase of the $ 1\rangle$ state	A quantised phase shift gate

## C.2. Two-qubit gates

Symbol	Name	Matrix	Description	Comments
$CX$	Controlled X	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	Controlled version of the single qubit X gate	This gate is often referred to as Controlled NOT (or CNOT)
$CY$	Controlled Y	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{bmatrix}$	Controlled version of the single qubit Y gate	
$CZ$	Controlled Z	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$	Controlled version of the single qubit Z gate	
$CR(\theta)$	Controlled phase shift	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{bmatrix}$	Controlled version of the single qubit phase shift gate	
$CR_k(\theta)$	Controlled k phase shift	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix}$	Controlled version of the single qubit $R_k$ gate	
$SWAP$	Swap gate	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	This gate can be used to change the order of two qubits	
$\sqrt{SWAP}$	Square root of swap	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{i+1}{2} & \frac{i-1}{2} & 0 \\ 0 & \frac{i-1}{2} & \frac{i+1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$		$\sqrt{SWAP}\sqrt{SWAP} = SWAP$
$iSWAP$	Imaginary SWAP gate	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$		$iSWAP$ is a physical implementation gate

$\sqrt{iSWAP}$	Square root of iSWAP	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} & 0 \\ 0 & \frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$		
$MS$	Mølmer-Sørensen gate	$\begin{bmatrix} 1 & 0 & 0 & i \\ 0 & 1 & i & 0 \\ 0 & i & 1 & 0 \\ i & 0 & 0 & 1 \end{bmatrix}$		MS is a physical implementation gate
$R_{ZX}$	ZX gate gate	$\begin{bmatrix} \cos(\frac{\theta}{2}) & 0 & -i \sin(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) & 0 & i \sin(\frac{\theta}{2}) \\ -i \sin(\frac{\theta}{2}) & 0 & \cos(\frac{\theta}{2}) & 0 \\ 0 & i \sin(\frac{\theta}{2}) & 0 & \cos(\frac{\theta}{2}) \end{bmatrix}$		The ZX-gate is a physical implementation gate given rise to by the Cross Resonance gate [IBMb]

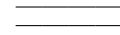
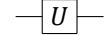
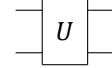



### C.3. Three-qubit gates

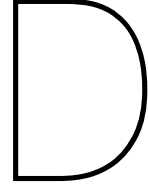
Symbol	Name	Matrix	Description	Comments
<i>CCNOT</i>	Toffoli or Controlled Controlled not	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$	This gate is the controlled controlled not gate	
<i>F</i>	Fredkin	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$	This gate is the controlled controlled swap gate	

## C.4. Quantum circuit symbols

As mentioned in Appendix A.2.1, quantum circuits are often given in the form of a figure. In this section we will give a table providing an overview of symbols used in such figures.

Table C.3: An overview of used symbols when representing a quantum circuit as a figure.

Operation	Circuit symbol	Further explanation
Identity		Each wire represents a qubit. When only the wire is drawn no operation is performed on the qubit.
U		The quantum gate $U$ is performed on the qubit.
Multiple-qubit gate		A multiple qubit gate $U$ is performed on the qubits whose wires cross the gate.
CNOT		CNOT is performed, here the first qubit is the control qubit the second qubit is the target qubit.
SWAP		This symbol represents a swap operation being performed between the two qubits.
Measurement		A measurement is performed on the qubit.



# Properties of considered quantum computers and their native gates

In this appendix we give some of the properties of the quantum computers and their native gates considered for this project. The first section shows how to calculate the partial derivatives of the native gates by using the theorem from [Sch+19]. The second section shows the lay-out and qubit connectivity of some of the quantum computers considered for this project.

## D.1. Gradients of implemented native gates

In Section 4.4 we gave a derivation of how the gradients of quantum gates can be determined. In this section we will go over the native gates considered for this project. Both parameterized native gates considered belong to the class for which we can evaluate the derivative by implementing  $G \pm \delta G$  as a quantum gate.

Below we show that the native gates considered belong to the class for which Theorem 4.4.3.1 holds and then we give the gate which we can use to implement  $G \pm \delta G$  and subsequently calculate the partial derivatives. We first show that this holds in the case we are working with a quantum circuit consisting of as many qubits as the native gates apply to (i.e. a single-qubit circuit in the case of  $R_Z(\theta)$  and a two-qubit circuit in the case of  $R_{ZX}(\theta)$ ). In this case we have that the layer  $G$  is precisely equal to the quantum gate. After having shown this case, we show that this generalises to the case that we are working with multiple qubits and we have that  $G$  consists of the native gate applied to the target qubit, with identity operations applied to the other qubits.

### D.1.1. Native gate: $R_Z(\theta)$

The rotation around the z-axis is generated by the Hermitian matrix

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad (\text{D.1})$$

we have

$$R_Z(\theta) = e^{-i\theta \frac{1}{2}Z}. \quad (\text{D.2})$$

Since  $\frac{1}{2}Z$  is also a Hermitian matrix and it has eigenvalues  $\pm \frac{1}{2}$ , it follows that this gate is in the class of quantum gates for which Theorem 4.4.3.1 holds, and so we know

$$\partial_\theta f(\theta) = \langle \phi' | R_Z\left(\theta + \frac{\pi}{4}\right) \hat{Q} R_Z\left(\theta + \frac{\pi}{2}\right) | \phi' \rangle - \langle \phi' | R_Z\left(\theta - \frac{\pi}{2}\right) \hat{Q} R_Z\left(\theta - \frac{\pi}{4}\right) | \phi' \rangle \quad (\text{D.3})$$

holds. This means we can evaluate the partial derivatives with respect to  $\theta$  by simply evaluating the circuit again, with  $R_Z(\theta)$  replaced by  $R_Z\left(\theta + \frac{\pi}{2}\right)$  and then again with  $R_Z(\theta)$  replaced by  $R_Z\left(\theta - \frac{\pi}{2}\right)$ .



### D.1.2. Native gate: $R_{ZX}(\theta)$

We know that

$$R_{ZX} = e^{\frac{-i\theta}{2}(X \otimes Z)} \quad (\text{D.4})$$

holds. Where we have that

$$X \otimes Z = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, \quad (\text{D.5})$$

so  $X \otimes Z$  has eigenvalues  $\pm 1$  and is a Hermitian matrix. Therefore the  $R_{ZX}$  gate also belongs to the class for which Theorem 4.4.3.1 holds and we can evaluate the partial derivatives by running the circuit again with  $R_{ZX}(\theta)$  replaced by  $R_{ZX}\left(\theta + \frac{\pi}{2}\right)$  and again with  $R_{ZX}(\theta)$  replaced by  $R_{ZX}\left(\theta - \frac{\pi}{2}\right)$  and subsequently calculating the expectation values. Finally we get

$$\partial_{\theta} f(\theta) = \left\langle \phi' \left| R_{ZX}\left(\theta + \frac{\pi}{4}\right) \hat{Q} R_{ZX}\left(\theta + \frac{\pi}{2}\right) \right| \phi' \right\rangle - \left\langle \phi' \left| R_{ZX}\left(\theta - \frac{\pi}{2}\right) \hat{Q} R_{ZX}\left(\theta - \frac{\pi}{2}\right) \right| \phi' \right\rangle \quad (\text{D.6})$$

### D.1.3. Extension of partial derivatives to circuits with more qubits

In the sections above we have shown how we can calculate the partial derivatives of the native gates, in case of a circuit consisting of as many qubits as the gate natively applies to (i.e. a single-qubit circuit for  $R_Z(\theta)$  and a two-qubit circuit for  $R_{ZX}(\theta)$ ).

In order to do this we will show that if we have a native gate  $R_Z(\theta)$  or  $R_{ZX}(\theta)$  as part of a circuit with more qubits, then the layer  $G(\theta) = I^{\otimes m} \otimes R_Z(\theta) \otimes I^{\otimes l}$  or  $G(\theta) = I^{\otimes m} \otimes R_{ZX}(\theta) \otimes I^{\otimes l}$  can still be seen as a quantum gate generated by a hermitian matrix with two unique eigenvalues  $\pm \frac{1}{2}$ .

Case 1:  $R_Z(\theta)$

Consider  $I^{\otimes m} \otimes Z \otimes I^{\otimes l}$  this matrix clearly has the same eigenvalues as  $Z$  and the claim is that

$$I^{\otimes m} \otimes R_Z(\theta) \otimes I^{\otimes l} = e^{\frac{-i\theta}{2}(I^{\otimes m} \otimes Z \otimes I^{\otimes l})}. \quad (\text{D.7})$$

This claim can be proven by writing out the matrix exponential

$$e^{\frac{-i\theta}{2}(I^{\otimes m} \otimes Z \otimes I^{\otimes l})} = \sum_{k=0}^{\infty} \frac{1}{k!} \frac{(-i\theta)^k}{2^k} (I^{\otimes m} \otimes Z \otimes I^{\otimes l})^k. \quad (\text{D.8})$$

We can use that  $(I^{\otimes m} \otimes Z \otimes I^{\otimes l})^2 = I^{\otimes(m+l+1)}$ , to get

$$e^{\frac{-i\theta}{2}(I^{\otimes m} \otimes Z \otimes I^{\otimes l})} = \sum_{k=0}^{\infty} \frac{1}{(2k)!} \frac{(-1)^k \theta^{2k}}{2^{2k}} I^{\otimes(m+l+1)} + i \sum_{k=0}^{\infty} \frac{1}{(2k+1)!} \frac{(-1)^k \theta^{2k+1}}{2^{2k+1}} (I^{\otimes m} \otimes Z \otimes I^{\otimes l}). \quad (\text{D.9})$$

Using the Taylor series of the sine and cosine we get

$$e^{\frac{-i\theta}{2}(I^{\otimes m} \otimes Z \otimes I^{\otimes l})} = \cos\left(\frac{\theta}{2}\right) I^{\otimes(m+l+1)} + I^{\otimes m} \otimes i \sin\left(\frac{\theta}{2}\right) Z \otimes I^{\otimes l} = I^{\otimes m} \otimes R_Z(\theta) \otimes I^{\otimes l}. \quad (\text{D.10})$$

We can conclude that  $G(\theta) = I^{\otimes m} \otimes R_Z(\theta) \otimes I^{\otimes l}$  is generated by a Hermitian matrix with two eigenvalues  $\pm \frac{1}{2}$  and so we can calculate the partial derivatives as follows

$$\partial_{\theta} f(\theta) = \left\langle \phi' \left| G\left(\theta + \frac{\pi}{2}\right) \hat{Q} G\left(\theta + \frac{\pi}{2}\right) \right| \phi' \right\rangle - \left\langle \phi' \left| G\left(\theta - \frac{\pi}{2}\right) \hat{Q} G\left(\theta - \frac{\pi}{2}\right) \right| \phi' \right\rangle. \quad (\text{D.11})$$

Case 2:  $R_{ZX}(\theta)$

Consider  $I^{\otimes m} \otimes X \otimes Z \otimes I^{\otimes l}$  again this matrix has the same eigenvalues as  $X \otimes Z$  and we claim

$$I^{\otimes m} \otimes R_{ZX}(\theta) \otimes I^{\otimes l} = e^{-\frac{i\theta}{2}(I^{\otimes m} \otimes X \otimes Z \otimes I^{\otimes l})}. \quad (\text{D.12})$$

We use the same structure as in the section above and write this out

$$e^{-\frac{i\theta}{2}(I^{\otimes m} \otimes X \otimes Z \otimes I^{\otimes l})} = \sum_{k=0}^{\infty} \frac{1}{k!} \frac{(-i\theta)^k}{2^k} (I^{\otimes m} \otimes X \otimes Z \otimes I^{\otimes l})^k. \quad (\text{D.13})$$

Where as before  $(I^{\otimes m} \otimes X \otimes Z \otimes I^{\otimes l})^2 = I^{\otimes(m+l+2)}$ . This leads to

$$e^{-\frac{i\theta}{2}(I^{\otimes m} \otimes X \otimes Z \otimes I^{\otimes l})} = \sum_{k=0}^{\infty} \frac{1}{(2k)!} \frac{(-1)^k \theta^{2k}}{2^{2k}} I^{\otimes(m+l+2)} + i \sum_{k=0}^{\infty} \frac{1}{(2k+1)!} \frac{(-1)^k \theta^{2k+1}}{2^{2k+1}} (I^{\otimes m} \otimes X \otimes Z \otimes I^{\otimes l}). \quad (\text{D.14})$$

Using the Taylor expansion for the sine and cosine we get

$$e^{-\frac{i\theta}{2}(I^{\otimes m} \otimes X \otimes Z \otimes I^{\otimes l})} = \cos\left(\frac{\theta}{2}\right) I^{\otimes(m+l+2)} + I^{\otimes m} \otimes i \sin\left(\frac{\theta}{2}\right) (X \otimes Z) \otimes I^{\otimes l} = I^{\otimes m} R_{ZX}(\theta) I^{\otimes l}. \quad (\text{D.15})$$

And we can conclude that  $G(\theta) = I^{\otimes m} R_{ZX}(\theta) I^{\otimes l}$  is generated by a Hermitian matrix with two eigenvalues  $\pm \frac{1}{2}$  and so we can calculate the partial derivative of  $f(\theta)$  with respect to  $\theta$  can be evaluated by

$$\partial_{\theta} f(\theta) = \left\langle \phi' \left| G\left(\theta + \frac{\pi}{2}\right) \hat{Q} G\left(\theta + \frac{\pi}{2}\right) \right| \phi' \right\rangle - \left\langle \phi' \left| G\left(\theta - \frac{\pi}{2}\right) \hat{Q} G\left(\theta - \frac{\pi}{2}\right) \right| \phi' \right\rangle. \quad (\text{D.16})$$

## D.2. Qubit connectivity of considered quantum computers

In this section we present the layout of several quantum current quantum computers, of which the connectivity is coded in the implementation of our approach. This is done to give the reader some more insight in the connectivity and size of modern NISQ devices.

Figure D.1: Layout of the Rigetti Aspen 7 quantum computer [HGB20].

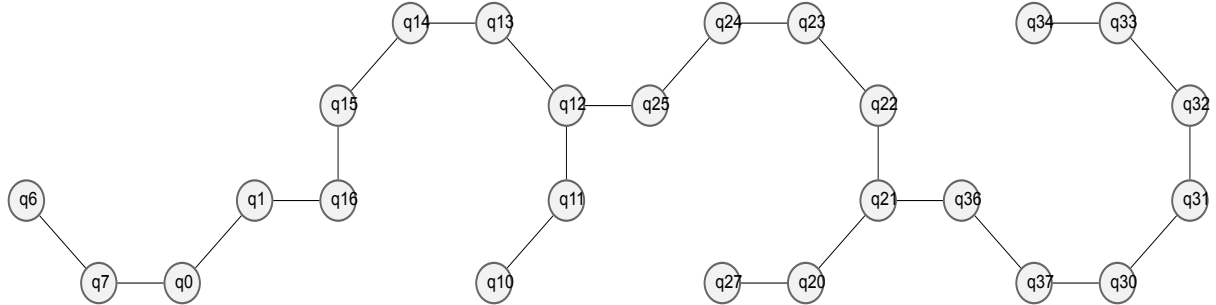


Figure D.2: Layout of the Rigetti Agave quantum computer [Mur+19].



Figure D.3: Layout of the IBM Ruschlikon quantum computer [IBMa].

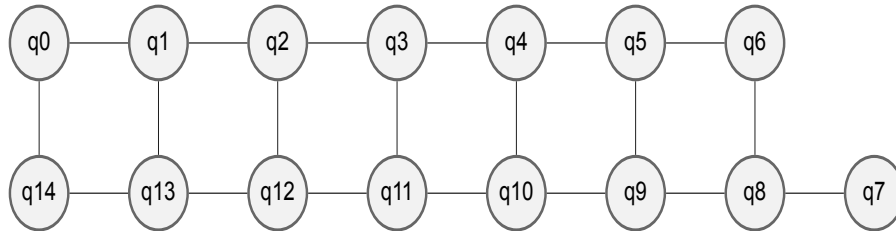


Figure D.4: Layout of IBM Lima quantum computer [IBMa], the lines the physical qubits are connected.

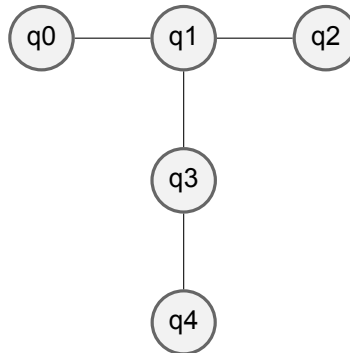


Figure D.5: Layout of IBM Yorktown quantum computer [IBMa].

