Generative Adversarial Learning with Quantum Neural Networks

Masterarbeit
von

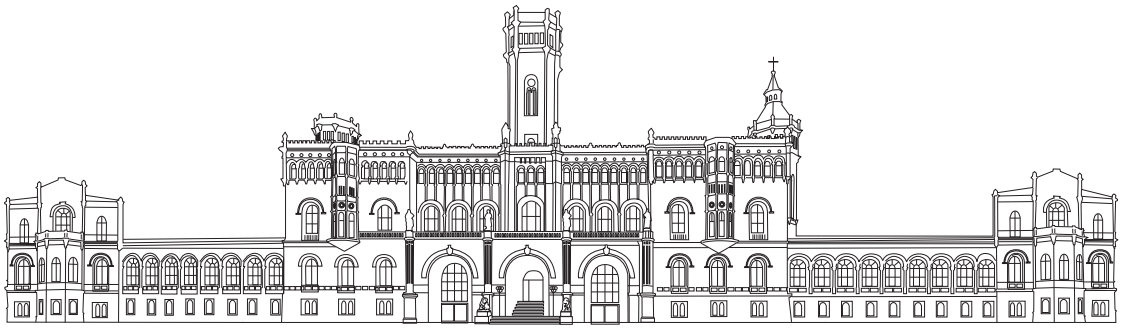
Gabriel Müller

angefertigt am
Institut für theoretische Physik

unter Anleitung von
Prof. Dr. Tobias J. Osborne
am 15.09.2021

Generative Adversarial Learning with Quantum Neural Networks

Masterarbeit



vorgelegt der Fakultät für Mathematik und Physik
der Leibniz Universität Hannover

Referent: Prof. Dr. Tobias J. Osborne
Koreferent: Prof. Dr. Reinhard F. Werner

15.09.2021

ABSTRACT

In the past few years, the quantum physicist's longstanding dream of quantum computers became more and more a reality. The upcoming quantum computers are expected to offer classically unattainable computational possibilities and a variety of novel applications. Among these are the simulation of many-body quantum systems, which requires exponentially growing computational resources with respect to the system size, and the realisation of quantum algorithms such as the prominent Shor's factoring algorithm. However, today's noisy intermediate-scale quantum computers come with high noise levels and a small number of qubits. These limitations still prevent fault-tolerant quantum computation and, thus, the execution of the just mentioned applications.

Quantum neural networks, the quantum analogue of classical neural networks, promise to harness the advantages of quantum computing already on today's quantum computers. These quantum neural networks are implemented as hybrid quantum-classical algorithms that execute a parameterised quantum circuit on a quantum computer while training its parameters via classical side computation. The training employs a loss function that is efficiently computed on a quantum computer and guides the parameter optimisation via the gradient descent method well known from classical machine learning. Due to the generally short parameterised quantum circuits, quantum neural networks can learn despite today's noise environment. This work proposes a specific parameterised quantum circuit, namely, the dissipative quantum neural network, which features completely positive layer-to-layer transition maps between different qubits of adjacent network layers. A comparison to the well-known quantum approximate optimisation algorithm in the supervised task of learning an unknown unitary transformation under different quantum gate noise levels and on an actual quantum computer shows a slight advantage of the dissipative quantum neural network in generalising the training data.

Finally, the classically successful method of generative adversarial learning is transferred to the quantum case resulting in quantum generative adversarial networks. The main concept is to alternately train two separate dissipative quantum neural networks, namely, a generator and a discriminator. The generator's task is to produce fake states similar to the training states while the discriminator's task is to distinguish the fake from the real states. The ultimate goal is to obtain a generator that produces quantum states with the same internal features as the training states. The training is guided by an objective function that describes a two-player minimax game. Besides a general training algorithm, this work proposes the fidelity objective function and the Hilbert-Schmidt objective function with different computational requirements and possibilities. A proof-of-concept for training quantum generative adversarial networks is given. Additional numerical investigations compare the training performance featuring both objective functions, various network architectures, and different types of training sets.

Contents

Chapter 1

Introduction

Quantum computing is, without doubt, one of the most exciting and rapidly evolving research areas of the 21st century. Although the power of classical computers has reliably grown exponentially over the past decades, famously known as Moore's law [1], this technology is still naturally prevented from solving certain problems such as simulating interestingly large quantum systems. This is due to the exponential scaling of the quantum systems' size in the number of objects. In 1982, Feynman proposed to use quantum computing to simulate many-body quantum systems by directly exploiting quantum mechanical objects as the computational resource [2]. The fundamental components of quantum computing are qubits which offer classically unattainable tools such as superposition and entanglement [3]. Based on these resources many more applications of quantum computing have been proposed with the hope to solve specific tasks more efficient than classical computers could. Prominent amongst those are Shor's factoring algorithm [4] and Grover's search algorithm [5].

The first generation of universal quantum computers has been built in the past few years. These quantum computers are commonly called *noisy intermediate-scale quantum* devices because of their relatively small number of qubits ranging from 50 to a few 100 and their high noise levels [6]. The research interest in this field is not only aroused by early claims of quantum supremacy [7] but also due to the public access to some of these devices [8]. However, under the circumstances of today's quantum computers, the realisation of fault-tolerant quantum computation [9] is impossible preventing the implementation of major generic quantum algorithms such as Shor's factoring and Grover's search algorithm.

The notion of machine learning has revolutionised the applications of classical computers in that it enables solving problems that no generic algorithm exists for. Consequently, machine learning [10] has long influenced everyone's daily life through applications in a variety of domains such as social networks [11], au-

tonomous driving [12], and genetics [13], to name a few. Especially relevant machine learning models are deep neural networks [14, 15] that are trained through experience and by large amounts of data. These neural networks offer a general framework that can easily be exploited in any learning setting.

The quantum version of classical neural networks, namely quantum neural networks, can be executed on today's quantum computers despite their challenging circumstances. The training of quantum neural networks employs hybrid quantum-classical algorithms [16–19] consisting of two parts: a parameterised quantum circuit [20–22] which is executed on a quantum computer and classical side computations for the training of the parameters. These quantum neural networks promise to resist the noisy environment of currently available quantum computers due to their short circuit length [23]. Many different definitions of quantum perceptrons as their fundamental building block have been proposed [24–27] and remain to be tested thoroughly for a wide range of tasks.

Quantum machine learning aims to exploit quantum neural networks for solving quantum tasks for which no generic and already executable quantum algorithm exists. These tasks include the initial idea of simulating quantum mechanical systems with the goals of finding the dynamical evolution [28, 29] or the corresponding eigenstates [30] for some system Hamiltonian. However, quantum machine learning is also characterised by the attempt to translate the promising ideas from its classical role model. One of the most successful classical techniques is generative adversarial learning [31, 32] which trains a neural network to generate new samples within a given data distribution. An analogous implementation of **generative adversarial learning with quantum neural networks** could prove itself extremely useful in finding new quantum states with some desired set of internal features.

Chapter 2 takes off by introducing the most important concepts of quantum computing. This starts with a revision of the relevant notions from quantum mechanics including the definition of the qubit along with its famous properties superposition and entanglement. Thereupon, it is explained how qubits can be exploited for computation by manipulating their states with quantum gates and how quantum algorithms can harness the quantum mechanical properties to get an advantage over classical computers. In the end, the physical realisation of quantum computers is discussed.

Chapter 3 continues the journey with a thorough review of classical neural networks. As a foundation, the general architecture of a neural network based on the perceptron is explained. This is followed by a description of the two fundamental tools for training such a network, namely, the loss function and the gradient descent method. As a final note, the major challenges and advanced strategies of training neural networks are presented.

Chapter 4 reaches the stopover and combines the previously discussed con-

cepts to quantum neural networks. First, the general notion of quantum neural networks and their training is introduced alongside a review of the most relevant architectures and applications. After that, two concrete proposals for quantum neural networks are presented which is followed by their implementation as quantum circuits. In conclusion, some numerical results, as well as executions on an actual quantum computer, are shown.

Chapter 5 arrives at the final destination with the implementation of generative adversarial learning with quantum neural networks. As a first step, the basic ideas of generative adversarial learning are laid out for the classical case. Thereafter, the algorithm for quantum generative adversarial networks is defined including two different objective functions for training. Finally, after performing a proof-of-concept, a variety of numerical results is discussed.

Chapter 2

Quantum Computing

Before thinking about quantum neural networks, the first ingredient is to understand the basic ideas and formalism of quantum computing. This chapter is therefore concerned with answering the following questions: *What distinguishes quantum computing from classical computing? How to work with and program quantum computers? Why should quantum computers be of any advantage compared to classical computers? Is it even possible to build a quantum computer?*

In Section 2.1, the famous concept of the qubit will be introduced as the building block of a quantum computer. Superposition and quantum entanglement as essential properties of the qubit are discussed thoroughly to promote a fundamental understanding of why quantum computing is intriguing. The Bloch sphere as a representation of a qubit's state is explained. Section 2.2 smoothly leads over to convey how qubits evolve in time and how they can be used for quantum computing. This is followed by a review of the most important tools of quantum computing, the so-called quantum gates, and how a few of them are already universal for quantum computing. Section 2.3 resolves the mystery behind how quantum computers can compute things that are unattainable classically. Furthermore, the most crucial quantum algorithms are reviewed. Section 2.4 shows the physical existence of qubits in Nature and examines the main conditions for building an actual quantum computer before finally, today's noisy intermediate-scale quantum computers are briefly discussed.

Many parts in this chapter are, if not explicitly cited differently, inspired and backed by the excellent book *"Quantum computation and quantum information"* written by *Michael A. Nielsen and Isaac L. Chuang* [3]. I, therefore, refrain from citing this publication repeatedly for the largest part of this chapter.

2.1 Quantum mechanics basics

The playground of quantum computation is given in the form of quantum bits, usually called qubits, as opposed to bits in classical computation. The classical bit and the qubit as the elementary components share the concept of having access to two different states, traditionally called 0 and 1 for a classical bit, and $|0\rangle$ and $|1\rangle$ for a qubit. The symbol $|\cdot\rangle$ reads as 'ket' and corresponds to the *Dirac notation* indicating a quantum mechanical state. A classical bit is always either in the state 0 or in the state 1. However, a qubit does not have to be in exactly one of these states but can occupy both states, $|0\rangle$ and $|1\rangle$, simultaneously. This is easily visualised through the image of a coin with two sides: *heads* and *tails*. While a 'classical' coin would always lie still on the ground with either *heads* or *tails* pointing up, a 'quantum' coin could additionally rotate such that it carries along with it both possibilities to be *heads* and *tails*. Eventually, the coin could be forced to the ground where finally either *heads* or *tails* would point up - a quantum measurement. Please note that the rotating 'quantum' coin's state is not just unknown to the observer, but the coin actually is in both states simultaneously.

The unique possibility of a qubit to be in two states $|0\rangle$ and $|1\rangle$ simultaneously is referred to as *superposition*. The state of a single qubit can mathematically be described as a normalised vector $|\psi\rangle$ in a two-dimensional complex Hilbert space $\mathcal{H}$ with two orthonormal basis vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.1}$$

Each possible state can now be written as a superposition of the basis vectors:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.2}$$

with complex factors $\alpha, \beta \in \mathbb{C}$ such that the normalisation of $|\psi\rangle$ is ensured through $|\alpha|^2 + |\beta|^2 = 1$. If a qubit in the state $|\psi\rangle$ is measured in the computational basis from Equation (2.1), it will be found in the state $|0\rangle$ with probability $|\alpha|^2$ and in the state $|1\rangle$ with probability $|\beta|^2$. One famous example for a quantum state in an equal superposition between the basis states is:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \tag{2.3}$$

which corresponds to an equal probability of 0.5 to measure either of the basis states, matching the previous example of a rotating coin. However, a qubit can not only be in an equal superposition but can occupy an infinite number of superpositions in the full continuum between the two computational basis states.

Another distinctive feature of qubits compared to classical bits is *quantum entanglement*, a phenomenon that can not be imitated classically. To observe this feature, a multi-qubit system is necessary which is mathematically given as the

tensor product of all single-qubit Hilbert spaces $\mathcal{H}$. For n qubits, this results in a 2^n-dimensional Hilbert space $\mathcal{H}^{\otimes n}$. For example, the two-qubit Hilbert space $\mathcal{H}^{\otimes 2}$ has $2^{n=2} = 4$ orthonormal basis states: $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. The notation $|xy\rangle = |x\rangle \otimes |y\rangle$ means that the first qubit is in state $|x\rangle$ and the second qubit is in state $|y\rangle$. In analogy to the single-qubit state, any two-qubit state can now be written as a superposition of these basis vectors:

$$|\psi\rangle = \alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |10\rangle + \alpha_4 |11\rangle , \tag{2.4}$$

again with complex factors α_1,α_2,α_3,$\alpha_4 \in \mathbb{C}$ such that $|\psi\rangle$ is normalised. If now, for example, $\alpha_1 \neq 0 \neq \alpha_4$, the two-qubit state from Equation (2.4) is an entangled quantum state - the state can not be written as a single tensor product of one-qubit states. Here, entanglement means that the two qubits can not be described separately, but their states depend on each other. A famous example for an entangled state is one of a total of four Bell states:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) . \tag{2.5}$$

The measurement of only one of the two qubits from $|\Phi^+\rangle$ determines the other qubit's state: if the first qubit's measurement yields $|0\rangle$, the second qubit's state is $|0\rangle$, too, and vice versa. Quantum entanglement in connection with superposition is why quantum mechanics might offer computational possibilities that are intractable classically.

For visualising the basic ideas of quantum computation it is often helpful to represent a qubit's state as a point on the so-called *Bloch sphere*. Any normalised one-qubit state from Equation (2.2) can be rewritten as

$$|\psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle \right) \tag{2.6}$$

with $\gamma, \theta, \phi \in \mathbb{R}$. Thinking about measuring this state, e.g. in the computational basis, it becomes clear that the complex phase factor $e^{i\phi}$ does not influence the measurement outcome. As this translates to all observations the factor $e^{i\phi}$ can be absorbed from Equation (2.6) and the state of a normalised qubit can be written as

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle \tag{2.7}$$

with only two parameters $\theta, \phi \in \mathbb{R}$. This allows a representation of the qubit on the surface of a Bloch sphere as shown in Figure 2.1. As an arbitrary choice, the north and south pole are defined as the computational basis states $|0\rangle$ and $|1\rangle$, respectively. A simple generalisation to represent also multi-qubit states with the concept of the Bloch sphere is not known. The possibility to represent the state

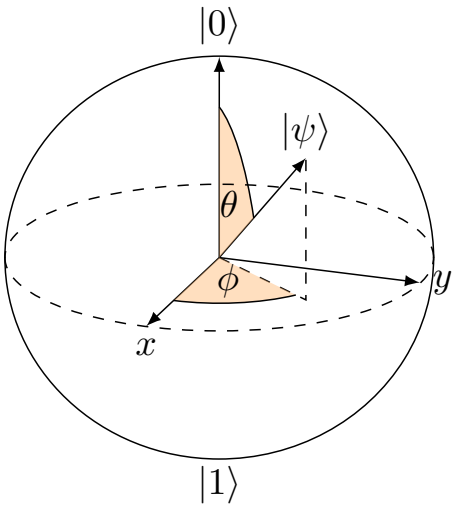


Figure 2.1: Definition of the Bloch sphere. The computational basis states $|0\rangle$ and $|1\rangle$ are defined at the north and south pole of the sphere, respectively. Except for a global phase factor, any normalised quantum state $|\psi\rangle$ can explicitly be represented by two rotation angles $\theta, \phi \in \mathbb{R}$ corresponding to Equation (2.7).

of multiple qubits by assigning each qubit to its own Bloch sphere is only partially helpful because entanglement between qubits can not easily be visualised.

What distinguishes quantum computing from classical computing?

The main difference between quantum computing and classical computing arises in the **quantum bit (qubit) compared to the classical bit**. While a classical bit either takes the value 0 or 1, a qubit can take up these values or any value in between. If a qubit takes a value in between, it is in a **superposition** of the values 0 and 1. An additional effect, called **quantum entanglement**, comes into play when multiple qubits interact with each other and enter into a correlation that is unattainable classically.

2.2 Quantum circuits

To utilise quantum mechanics for computation, i.e., quantum computing, it is necessary to understand how the computing resources, the qubits, evolve in time. Ideally, some set of n qubits is given as an isolated system which can then be described as previously discussed: as a state vector in the 2^n-dimensional Hilbert space $\mathcal{H}^n$. According to one of the postulates of quantum mechanics, any closed quantum system evolves in time through unitary transformations. Such a unitary transformation can act on single qubits or multiple qubits. It can always be written

as

$$|\psi'\rangle = U\,|\psi\rangle \tag{2.8}$$

with a unitary operator U. Such a unitary operator could act on each qubit separately like $U = U_1 \otimes U_2 \otimes \cdots \otimes_n$ where the unitary U_i acts only on the i^{th} qubit. It could also act on multiple qubits like $U = U_{12} \otimes \mathbb{1}_3 \otimes \cdots \otimes \mathbb{1}_n$ where the unitary U_{12} acts on the first two qubits with the possibility to entangle them and the identity $\mathbb{1}_i$ acts on the i^{th} qubit. The defining feature of a unitary transformation is that it preserves the inner product, i.e., the normalisation $\langle\psi|\psi\rangle = \langle\psi'|\psi'\rangle = 1$ holds throughout the evolution.

The operations in quantum computing are given in the form of *quantum circuits*: defining the initial qubit states, the unitary operations, and the measurements. Such a quantum circuit, which is read from left to right, is shown in Figure 2.2. Usually, at the beginning of computation, all qubits are in the computational basis state $|0\rangle$ without having any correlation at all. To each qubit, a line is assigned that contains all actions the qubit is exposed to. Throughout the computation, unitary operations act on the qubits. These are called *quantum gates* and are depicted as boxes and other symbols on the respective lines. At the end of the circuit, it is possible to measure each qubit's state in the computational basis, which is depicted by  . The result of a single qubit's measurement is binary. Either the state $|0\rangle$ or the state $|1\rangle$ is measured. These measurement results are saved to one classical bit per measurement, depicted as the double line below the qubits' lines. To get relevant results from such a quantum circuit, it is executed many times, and the results of the single measurements are averaged.

A few quantum gates will be presented here exemplary to promote an understanding of their effects on qubits and their role in quantum circuits. The presented gates are applied within the quantum circuit in Figure 2.2. Their effects are emphasised with the help of Bloch sphere representations of each qubit's state. All the quantum gates mentioned below are shown with their circuit symbol and their matrix representation in Table 2.1. The simplest quantum gates are single-qubit gates, these do only act on one qubit at a time, and their effects do not depend on other qubits. The most important of these is the Hadamard gate because of its valuable property to map a qubit to an equal superposition of $|0\rangle$ and $|1\rangle$ if it previously is in one of the computational basis states. In particular, it maps a qubit from $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and from $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Another single-qubit gate is the T gate, which implements a unitary that only influences the phase, i.e., the use of the T gate does not affect the measurement results in the computational basis. The last single-qubit gate listed here is the Pauli-X gate, also called the NOT-gate in the style of its classical equivalent. It performs a rotation around the x-axis of the Bloch sphere by π radians, i.e., it maps from $|0\rangle$ to $|1\rangle$ and from $|1\rangle$ to $|0\rangle$. There are also two-qubit gates, these act on two qubits simultaneously, and their effects on one of the qubits may depend on the other qubit's state. The most important

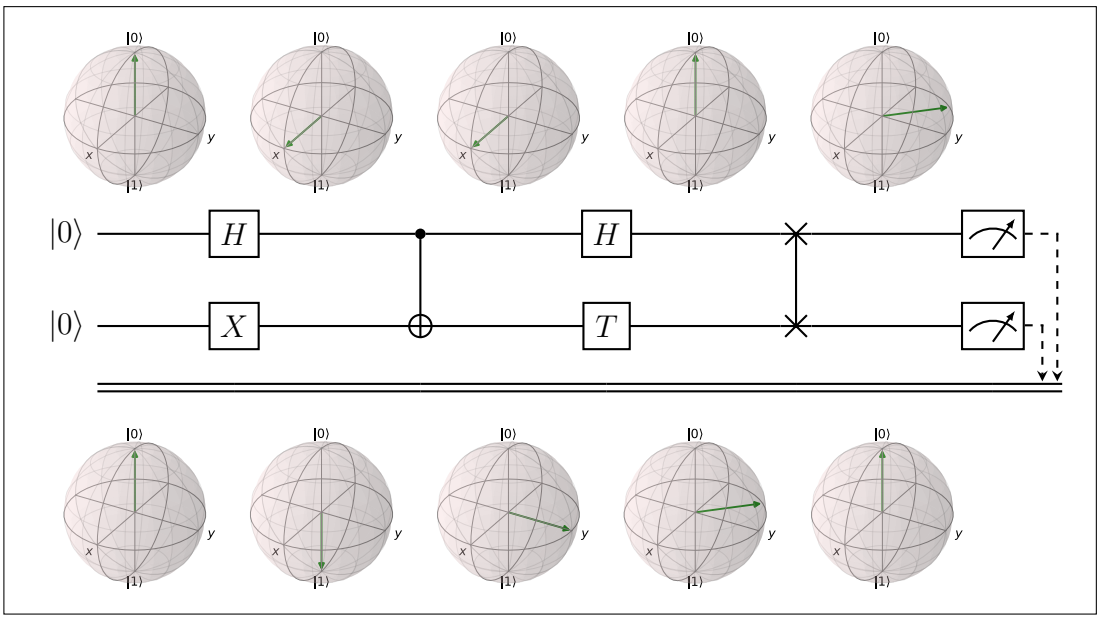
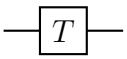
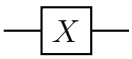
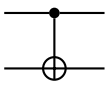
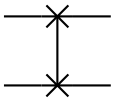


Figure 2.2: Example quantum circuit with Bloch sphere representations. The quantum circuit is depicted in the center with a single line for each qubit and one double line below for a classical memory out of bits. The current state of the upper (lower) qubit is represented by the upper (lower) Bloch spheres. Both qubits are initialised in the computational basis state $|0\rangle$. The quantum circuit has a depth of 4, the definition of the individual gates is given in Table 2.1. As an example, the effect of the first Hadamard gate on the upper qubit can be understood by comparing the first and second Bloch sphere representations: the Hadamard gate creates an equal superposition between $|0\rangle$ and $|1\rangle$ when acting on a qubit that previously is in the state $|0\rangle$. At the end of the circuit, both qubits are measured in the computational basis $\{|0\rangle, |1\rangle\}$ and the individual results are saved to the classical memory.

of these is the controlled NOT gate, usually called the CNOT gate. It acts on the two qubits asymmetrical. One of the qubits serves as the control qubit and remains unchanged while the other is acted on - the target qubit. The effect of the CNOT gate is that the NOT gate is only applied to the target qubit if the control qubit is in the $|1\rangle$ state, otherwise, nothing happens to the target qubit. If the control qubit is in some superposition of $|0\rangle$ and $|1\rangle$, the NOT gate can be thought of as only partially executed with the corresponding ratio of the superposition. Another two-qubit gate is the SWAP gate which can be built out of three successive CNOT gates with alternating control and target qubits. As the name suggests, the SWAP gate swaps the states of the two qubits and thus, acts symmetrically.

It is possible to compute any 2^n-dimensional unitary operation for an arbitrarily large number of n qubits using only a few of the already mentioned quantum gates: the Hadamard gate, the T gate, and the CNOT gate [33]. The proof of this statement consists of three steps [3]. First, it can be shown that any single-qubit unitary can be approximated to arbitrary accuracy with only the Hadamard gate and the T gate [33]. As a second step, it is possible to show that the available single-qubit unitaries can be combined with CNOT gates to implement any 2-level

Table 2.1: Overview of the most important quantum gates. The Hadamard, the T and the Pauli-X gates are single qubit gates. The CNOT and the SWAP gates are two qubit gates whereas the SWAP gate can be build with 3 successive alternating CNOT gates.

Name	**Circuit**	**Matrix**
Hadamard		$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$
T		$\begin{bmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{bmatrix}$
Pauli-X		$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$
CNOT		$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$
SWAP		$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$

unitary in a 2^n-dimensional Hilbert space [34], i.e., a 2^n-dimensional unitary that acts non-trivial on only two components of the n qubit state vector. The third and concluding step shows that any 2^n-dimensional unitary can be built as a product of these 2-level unitaries [35]. However, the implementation of high-dimensional unitaries in this manner may not be efficient. Therefore the use of specifically designed unitaries with better performance will be addressed in the following subchapter.

How to work with and program quantum computers?

According to the laws of quantum mechanics, the qubits as the building block of a quantum computer evolve in time through **unitary transformations**. Thus, to program a quantum computer means to perform specific unitary transformations on the qubits in the form of **quantum gates**. A complete quantum program is written as a **quantum circuit** which can be seen as a schedule of quantum gates and measurements to the qubits. Interestingly, the access to single-qubit gates and the so-called CNOT gate is sufficient to perform any unitary transformation on arbitrarily many qubits.

2.3 Quantum algorithms

"A quantum computer calculates exponentially many routes simultaneously" is an often heard sentence in physics pop culture - it carries truth but it can not be taken for granted. As discussed in Section 2.1, a quantum computer with n qubits builds an exponentially large 2^n-dimensional Hilbert space with 2^n basis vectors (shown for $n = 1$ in Equation (2.1)). In comparison, the state space of a classical computer only grows linearly in the number of bits n, i.e., the dimension of a classical state equals the number of bits n. The quantum feature of superposition allows the qubits to occupy all 2^n basis states simultaneously (shown for $n = 1$ in Equation (2.3)). The idea of *quantum parallelism* is to utilise this fact for calculating some function $f(x)$ for 2^n different inputs x simultaneously [36]. A simplified example to compute the one-bit function $f(x) : \{0,1\} \rightarrow \{0,1\}$ on $2^{n=1}$ different inputs $|0\rangle, |1\rangle$ in only one execution will be sketched in the following, the ideas however can be generalised to larger n [37]. Two qubits are used: one input qubit initialised as $x = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and one output qubit initialised as $y = |0\rangle$. The function $f(x)$ can be evaluated on both inputs simultaneously with the 4-dimensional unitary U_f acting on both qubits as

$$U_f \, |x,y\rangle = |x,y \oplus f(x)\rangle = \frac{|0,f(0)\rangle + |1,f(1)\rangle}{\sqrt{2}}, \tag{2.9}$$

where $\oplus$ indicates an addition modulo 2. However, if the qubits are measured now, only one of the results is preserved: either the input qubit is $|0\rangle$ and $f(0)$ can be obtained through a measurement of the output qubit but $f(1)$ is lost or vice versa. Although the parallel calculated results are available at some point, only one can be extracted by this procedure - this could be done by a classical computer as well. To benefit from quantum parallelism an extra step is needed to make the results available.

By applying *Deutsch's algorithm* [36, 38] as an additional step to quantum parallelism, properties can be obtained in a single execution that a classical computer

would need multiple executions for. The function of interest still is the one-bit function $f(x) : \{0,1\} \to \{0,1\}$ and by modification of the previously mentioned procedure it will be possible to determine the global property $f(0) \oplus f(1)$ with only one execution of U_f. The input qubit is again initialised as $x = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. The output qubit is initialised differently as $y = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. (This initialisation can be achieved through applying the Hadamard gate on the basis states $|0\rangle$ and $|1\rangle$, respectively.) By thinking through the two possible cases of $f(x) = 0$ and $f(x) = 1$, it becomes clear that the results after applying U_f on both qubits only differ by a global sign for a general input qubit $|x\rangle$:

$$U_f \left(|x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \right) = (-1)^{f(x)} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]. \tag{2.10}$$

After thinking through the two possible cases of $f(0) \oplus f(1) = 0$ and $f(0) \oplus f(1) = 1$, the following result is obtained:

$$U_f \left(\left[\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \right) = \pm \begin{cases} \left[\frac{|0\rangle+|1\rangle}{\sqrt{2}} \right] \left[\frac{|0\rangle-|1\rangle}{\sqrt{2}} \right] & \text{if } f(0) = f(1) \\ \left[\frac{|0\rangle-|1\rangle}{\sqrt{2}} \right] \left[\frac{|0\rangle-|1\rangle}{\sqrt{2}} \right] & \text{if } f(0) \neq f(1). \end{cases} \tag{2.11}$$

By applying an additional Hadamard gate to the input qubit, it is in the state $|0\rangle = f(0) \oplus f(1)$ if $f(0) = f(1)$ and in the state $|1\rangle = f(0) \oplus f(1)$ if $f(0) \neq f(1)$ such that the final state of the qubits can be written without separation in cases as

$$\pm |f(0) \oplus f(1)\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]. \tag{2.12}$$

Now the input qubit may be measured without the loss of information and the global property $f(0) \oplus f(1)$ can be obtained with only one execution of the unitary U_f which is not possible with a classical computer.

The possibility of using Deutsch's algorithm to harness the advantage of quantum parallelism [39, 40] calls for a search for other quantum algorithms that offer speedups for more practical problems. One group of quantum algorithms utilises the quantum version of the Fourier transformation [41], which can be calculated with an exponential speedup compared to the classical Fourier transformation. The most famous algorithm that uses the quantum Fourier transformation to obtain a quantum advantage is *Shor's factoring algorithm* [4]. The goal is to find the prime factors of any given number N. While the best known classical algorithm runs in sub-exponential time $\mathcal{O}(e^{1.9(\log N)^{1/3}(\log\log N)2/3})$ [42], Shor's algorithm offers a nearly exponential speed up and runs in polynomial time [43]. Another family of quantum algorithms is famously represented by *Grover's search algorithm* [5] which accomplishes the task of finding a specific

item in an unstructured list of length N. With approximately $\sqrt{N}$ required operations, it achieves a quadratic speed up compared to classical algorithms that would approximately need N operations. The third and last important group of quantum algorithms is *quantum simulation* [2, 44, 45]. In this task, classical computers suffer from exponentially increasing system sizes of quantum mechanical systems because classical memories only increase linearly with the number of bits and therefore run out of memory very quickly. By contrast, the simulation of quantum mechanical systems on a quantum computer benefits from an exponential memory size in the number of qubits. However, similar to Deutsch's algorithm, it is not sufficient to just simulate a quantum system via a quantum computer, but the relevant information must be accessible.

Why should quantum computers be of any advantage compared to classical computers?

One of the main advantages of quantum computing is the **exponential scaling of the Hilbert space with the number of qubits**. This, in combination with superposition, enables the simultaneous access of exponentially many different states for computation - **quantum parallelism**. Using clever algorithms, this can be used to extract quantities in a single execution with a quantum computer that a classical computer would need exponentially many executions for.

2.4 Quantum computers

The mathematical model of qubits and its possible applications in quantum computing sound very promising, however, that is only really helpful if there is a physical realisation of qubits. Such a physical realisation would be given by a naturally existing 2-level system, i.e., a physical object that accesses exactly two independent states, which could be interpreted as basis states $|0\rangle$ and $|1\rangle$, and can exist in a superposition of the former states. The properties sought can be observed in the so-called *Stern-Gerlach experiment* [46, 47]: hydrogen atoms are heated in an oven and exit into an inhomogeneous magnetic field after which their flight direction is observed. What happens is that the atoms equally split into two discrete angles, one half of the atoms is deflected upwards, and the other half is deflected downwards. This behaviour can not be explained by the atom's magnetic dipole moment caused by the electron's current around the atom. Instead, its origin lies in a quantity of the atom's electron, called spin, which will now be characterised through a series of observations. Suppose the magnetic field is oriented along the z-axis. Then the atoms split into two discrete points along the z-axis, the upper atoms have a *spin-up* and their state is called $|Z+\rangle \equiv |0\rangle$, the

lower atoms have a *spin-down* and their state is called $|Z-\rangle \equiv |1\rangle$. This fulfils the first condition for exactly two independent states as the spin is a discrete property with only two observable states. Suppose now, the upper atoms in the state $|0\rangle$ are exposed to another magnetic field, this time along the x-axis. The atoms split into two discrete points again: to the right ($|X+\rangle$) and the left ($|X-\rangle$). Now the atoms in the state $|X+\rangle$ are sent through a magnetic field along the z-axis again, and surprisingly, the atoms split into two discrete points just like after the first magnetic field: upwards and downwards. Both states, $|0\rangle$ and $|1\rangle$, still exist, although the first part of the experiment should have filtered out the latter. The relevant conclusion is that the state $|X+\rangle$ resembles a superposition of what is called spin-up and spin-down, i.e., $|X+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. This fulfils the second condition that the physical object can exist in a superposition of the two independent states. Thus, it can be concluded that a qubit is not only a mathematical model but also exists as a physical object.

With the existence of qubits in Nature ensured, one main obstacle to building a quantum computer stands above all [3]. On the one side, a quantum computer and its physical qubits should be well isolated from the outer world to prevent undesired interactions and any destruction of their quantum properties. On the other side, the qubits must be accessible for applying quantum gates to perform computation and measurement of their state. Consequently, there can not be a perfect quantum computer, but rather a good balance between the reduction of noise and the improvement of computational possibilities is worth striving for.

What follows is a brief review of four general conditions and steps for building a quantum computer capable of quantum computing [3, 48]. The first essential condition is that the quantum computer has to store quantum information precisely and robustly. A discrete physical system is needed whose different states can reliably be distinguished and optimally remain in any state as long as possible, especially in arbitrary superposition states. The second condition is that the quantum computer can perform a universal set of elementary quantum gates such that any unitary transformation can be performed. As discussed in Section 2.2, it would be sufficient if the Hadamard gate, the T gate, and the CNOT gate can be executed. In realising this, the concrete goals are to maximise the fidelity of any single or two-qubit gate, i.e., to increase their working precision and to minimise the time to perform one of the elementary gates. As these gates are always executed through some real-world instrument, it is also necessary to ensure that each qubit can be addressed individually. The third condition is the ability to repeatedly initialise all qubits into the same known state before each new execution of a quantum circuit. If this were not the case, the ability to perform quantum gates with high precision would be useless because the resulting state lacks any context. It is sufficient to prepare a simple state such $|0\ldots0\rangle$ as an initial state because any other state could be created with unitary transformations afterwards. The initial state is considered optimal if it is pure and can be provided with high

fidelity. The fourth and last condition is concerned with the measurement of the qubits' states. Here a measurement means coupling a qubit to a classical system such that eventually, the classical system indicates the qubit's state. It is crucial to ensure that such a measurement only occurs if desired, otherwise, it resembles a decoherence source as, for example, it can destroy superposition states. A relevant quantity to rate the measurement quality is the signal to noise ratio, as it includes the available signal strength and any possible measurement inefficiencies.

In the past few years, the first generation of quantum computers capable of quantum computing has been built: *noisy intermediate-scale quantum* (NISQ) devices [6]. The computational possibilities on these early quantum computers are mainly limited by two factors: their high noise levels and a small number of qubits. Two of the dominant noise sources are the imperfect control over the qubits via the quantum gates and the imperfect measurements. For superconducting quantum computers, for example, the error rate per two-qubit gate is larger than 0.1%, and the measurement error probability is about 1%. Due to the quantum gate noise, the arising limitation is that only approximately 1000 two-qubit gates can be applied before the qubits are overwhelmed with noise. The use of quantum error correction to tackle these limitations is impossible as this would require many additional qubits, which are not yet available. The number of qubits available in today's quantum computers is around 50 to a few 100 - rising steadily. Despite many limitations to NISQ devices, a system of 50 qubits is already so large that certain quantum circuits can not be simulated via the best classical supercomputers anymore. In 2019, [7] even claimed that they had realised quantum supremacy with their 53-qubit superconducting processor, i.e., their quantum computer solved a specific task many times faster than the best classical supercomputer could. What is more, many big tech companies start to invest in building quantum computers. Among them, IBM even offers the free use of some of their quantum computers since 2016 [8].

Is it even possible to build a quantum computer?

Yes, it is. The physical existence of the qubit as a **two-level system in Nature** has been shown by the Stern-Gerlach experiment. One of the main obstacles of building a quantum computer is that its **qubits have to be well-isolated** from the outer world **while being accessible** to the physical realisation of quantum gates. Despite that and many other challenges, the first publicly available generation of quantum computers has been built - called **noisy intermediate-scale quantum devices**.

Chapter 3

Classical Neural Networks

The second ingredient before thinking about quantum neural networks is, after quantum computing has been discussed, an understanding of the basic concepts of classical neural networks as their predecessor. For this purpose, this chapter aims to thoroughly answer the following questions: *What does a classical neural network look like, and what can it do? What is the training process for such a network? What are the challenges of training these networks?*

In Section 3.1, the perceptron as the fundamental component of a neural network is introduced. Then an explanation of the computationally more powerful sigmoid neuron and the composition of many of those into a neural network follows. After establishing that such a network can compute any continuous function to arbitrary precision, Section 3.2 discusses the training of a neural network according to supervised learning. Two main concepts are introduced: the loss function as a quantity to evaluate the training process and the gradient descent method to carry out the training. This is proceeded with discussing improvements to the training process before reviewing the most crucial pillar for the success of classical neural networks: the backpropagation algorithm. Finally, Section 3.3 reviews some common challenges and techniques when training these networks, namely, finding well-behaving hyper-parameters, getting a generalising neural network, and avoiding the vanishing gradient problem.

All the knowledge that is conveyed through this chapter, I initially gained from the extraordinary online book *"Neural Networks and Deep Learning"* created by *Michael A. Nielsen* [15]. In the reader's interest, I refrain from citing this publication during this chapter.

3.1 Architecture and activation functions

The fundamental component of an artificial *neural network* is the *perceptron* [49] which represents a function with several binary inputs and a single binary output. The perceptron as shown in Figure 3.1 is defined by a weight parameter $w_j \in \mathbb{R}$ for each binary input value $x_j \in \{0,1\}$ and a bias parameter $b \in \mathbb{R}$. The perceptron's binary output is given by:

$$\text{output} = \begin{cases} 0 & \text{if} \quad \sum_j w_j x_j + b \leq 0 \\ 1 & \text{if} \quad \sum_j w_j x_j + b > 0 \end{cases} \tag{3.1}$$

where $z' = \sum_j w_j x_j$ is called the *weighted sum*. Using this definition, the perceptron serves as a binary decision-maker considering some bias b towards one of the outcomes and multiple binary inputs x_j that influence the final decision differently through the corresponding weights w_j. Suppose you have to decide whether to do a PhD in Physics or not. Your bias will probably lean towards doing the PhD because you love physics (positive bias) or not doing the PhD because you love your life (negative bias). Additionally, there will be arguments supporting the decision to do the PhD (positive weights) if they apply (binary input 1), e.g., the fact that you like working autonomously. Analogously, there may exist arguments against doing a PhD (negative weights). By assigning weights w_j to the different arguments j and choosing whether each of them applies to you (x_j), the decision is given by Equation (3.1). By finding plausible values for the weights and the bias, the decision-making process can be improved. This process of improving the parameter values is understood as training a perceptron, or later an entire neural network.

There are alternatives to the binary perceptron which provide larger computational power and better training conditions. The perceptron's limitation to binary inputs and outputs can easily be removed by allowing non-binary inputs and taking the weighted input $z = z' + b$ as the output directly. This definition of a perceptron that is not a step function anymore introduces continuous gradients with respect to the weights and the bias, which can be used for training these parameters. This version of a perceptron has the form of the linear function

$$\text{output} = z = w_1 \cdot x_1 + \cdots + w_k \cdot x_k + b \tag{3.2}$$

and can as such be used in linear regression as a model between k explanatory variables x_j and one dependent scalar variable. To improve the perceptron's computational power beyond linear, an additional non-linear activation function such as the sigmoid function σ can be introduced. The sigmoid function takes the weighted input z' and returns the output of the so-called *sigmoid neuron*, also referred to as the neuron activation:

$$a = \sigma(z) = \frac{1}{1 + \exp(-z)} \in (0,1). \tag{3.3}$$

The desired feature of continuous differentiability holds for the sigmoid neuron, making it a useful non-linear basis component for a neural network. It is shown in Figure 3.1.

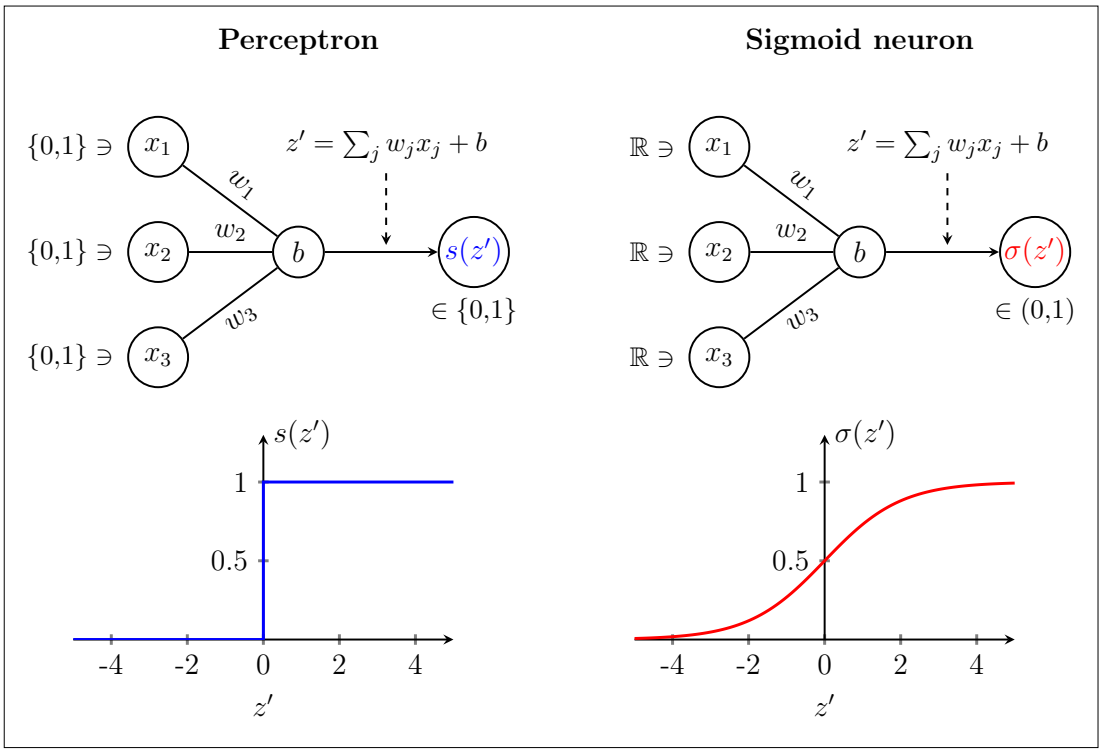


Figure 3.1: Perceptron and sigmoid neuron. While the perceptron takes only binary inputs $x_j \in \{0,1\}$, the sigmoid neuron allows any real inputs $x_j \in \mathbb{R}$. Both architectures calculate the weighted input z' with respect to the inputs x_j, the weights w_j and the bias b. As an activation function, the perceptron uses the step function from Equation (3.1) while the sigmoid neuron uses the sigmoid function according to Equation (3.3). Consequently, the perceptron's output is binary while the sigmoid neuron's output can be any real number between 0 and 1.

A neural network is a composition of many artificial neurons with the loose goal of imitating biological neural networks. The general architecture of the most basic neural network, the *multilayer perceptron* [50], is obtained by connecting m_l neurons in parallel as the l^{th} layer of the network and connecting all neurons of adjacent layers in series. The first and last layers are called the input and output layers, respectively, while the L layers in between are called the hidden layers. By connecting many neurons in this manner, a neural network can be used to represent a high-dimensional function $f_{\text{NN}} : \mathbb{R}^{m_0} \to \mathbb{R}^{m_{L+1}}$ where m_0, m_{L+1} are the numbers of neurons in the input layer $l = 0$ and output layer $l = L + 1$, respectively. The network is fed m_0 scalars as an input assigned to the neurons in the first layer. From here on, the neuron activations of every following layer depend on the neurons from the previous layer. A feed-forward mechanism gives the translation through the network from layer l to layer $l+1$ until the last layer, the output layer $l = L + 1$, is reached. Each of the m_{l+1} neurons in the layer $l + 1$ is connected

to all m_l neurons in the previous layer l by an individual set of weights and a bias. A popular choice for the artificial neuron is the sigmoid neuron, such that the neuron activations of subsequent layers are given by Equation (3.3). Overall, the neural network f_{NN} is parameterised via a set of weights $\vec{w}_l^j$ and a bias b_l^j for every neuron $j = 1,\dots,m_l$ in the layers $l = 1,\dots,L+1$ and its output is given by the neuron activations in the output layer. Such a feed-forward neural network with the sigmoid neuron is shown in Figure 3.2.

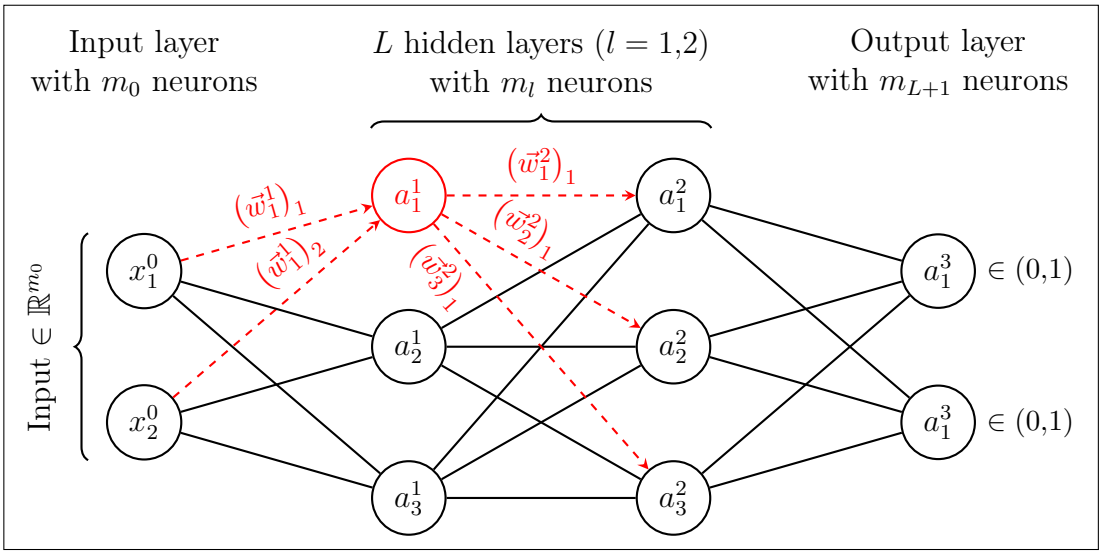


Figure 3.2: Feed-forward neural network with sigmoid neuron. The exemplary neural network features the input layer ($l = 0$), $L = 2$ hidden layers ($l = 1,2$), and the output layer $l = 3$. Each layer consists of m_l neurons, respectively. The sigmoid neuron according to Equation (3.3) is used which allows an input vector $\in \mathbb{R}^{m_0}$ and produces an output vector $\in [0,1]^{m_{L+1}}$. The feed-forward mechanism is shown by the example of the first neuron in the hidden layer $l = 1$: it gets the weighted input from the previous layer $l = 0$, calculates its sigmoid activation a_1^1 and forwards it to the subsequent layer $l = 2$.

A neural network with the sigmoid neuron can compute any continuous function when provided enough neurons in only one hidden layer [51], a visual proof [15] of which will be sketched here. More precisely, a neural network f_{NN} is said to be able to compute an arbitrary continuous function $g(x)$ if it satisfies

$$|f_{\text{NN}}(x) - g(x)| < \epsilon \tag{3.4}$$

for all inputs x and arbitrary small $\epsilon > 0$. Suppose there is a scalar function $g : \mathbb{R} \to \mathbb{R}$ that the neural network attempts to compute. One basic requirement of this network would be to have one input neuron and one output neuron to imitate the structure of g. The proof's general idea is to let the network act as a multi-step function such that it mirrors the target function g. A single step function can be approximated with a sigmoid neuron in the hidden layer by using large weight parameters. Its position can be adjusted via the bias parameter. Any arbitrary multi-step function can be designed through the

iterative process of adding neurons to the hidden layer in this manner. By increasing the number of neurons, the network's accuracy can be improved, and the condition from Equation (3.4) can be fulfilled for any $\epsilon > 0$. This concludes the sketched proof for f_{NN} to be able to compute any one-variable target function g. The idea of this sketched visual proof for one-variable functions can be extended to multi-variable functions $h : \mathbb{R}^N \to \mathbb{R}^M$. The interested reader is kindly referred to [15]. Please note that this sketched proof does not claim to be rigorous but instead tries to convey the idea of a neural network being able to compute any function by combining many multi-variable step functions.

What does a classical neural network look like, and what can it do?

A classical neural network is a **composition of many neurons sorted into layers**. A neuron is characterised by its activation. Except for the neurons in the first layer, each neuron's activation is given as a parameterised function of the neuron activations in the previous layer. These connecting pieces are called **perceptrons** and feature individual **weights and bias parameters**. The network works by using a given input as the input layer's neuron activations and giving its output as the output layer's neuron activations. In principle, a classical neural network can act as **any continuous function**.

3.2 Supervised training algorithm

A neural network can be trained according to different kinds of tasks: unsupervised, semi-supervised, or supervised learning, the latter of which will be explained in the following. Supervised learning means that the training data is labelled, i.e., it consists of N input and output data pairs $\{(\vec{x}_i, \vec{y}_i)\}_{i=1}^N$. In general, the input and output data is given as d_{in} and d_{out} -dimensional vectors, respectively. During training, the neural network is given access to a part of the labelled training data, and its task is to imitate the map $f_{\text{data}} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ that is implicitly defined by the training data pairs. In particular, the desired result is that the trained neural network f_{NN} maps each input vector $\vec{x}_i$ to the corresponding output vector $\vec{y}_i$, i.e., $f_{\text{NN}}(\vec{x}_i) = \vec{y}_i$ for all $i = 1,...,N$. Such a network must, in the first place, match the dimensions of the training data. This is realised on the one hand by choosing the number of neurons in the input layer such that the input data vector $\vec{x}_i$ can be stored in the input layer of the network. For real data vectors, as assumed here, that is achieved by using a neural network with d_{in} input neurons. On the other hand, the neural network analogously needs d_{out} neurons in the output layer to return its network output $f_{\text{NN}}(\vec{x}) \in \mathbb{R}^{d_{\text{out}}}$ as the respective neuron values.

To train a neural network in a supervised manner, a loss function is used to

evaluate the learning progress. The loss function is the only quantity that the network infers information from during the training process and should therefore be designed thoughtfully. It should be trustworthy in the sense that it converges to a global extreme point, say the minimum, if and only if the neural network works perfectly. Otherwise, it should move away from the minimum. Furthermore, the loss function should offer information that enables the neural network to improve towards the global minimum, e.g. in the form of a continuous gradient. These conditions are fulfilled by one of the widely adopted loss functions for supervised learning, i.e., the *mean squared error* between the desired output $\vec{y}_i$ and the network's output $f_{\text{NN}}(\vec{x}_i)$ for different samples $i = 1,...,N$:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{N} \|\vec{y}_i - f_{\text{NN}}(\vec{x}_i)\|^2. \tag{3.5}$$

As this loss function converges to 0 if the network's output $f_{\text{NN}}(\vec{x}_i)$ matches the desired output $\vec{y}_i$, the neural network should be trained such that it minimises Equation (3.5).

A neural network can be trained via the *gradient descent* method which iteratively updates the network parameters using the gradient of the loss function. The goal of training the neural network is to find weight and bias parameters such that the supervised learning task is accomplished, i.e., the loss function is minimised. Before training, the network parameters are initialised randomly as $\vec{w}_0$ and $\vec{b}_0$, resulting in a parameterised network function $f_{\text{NN}}(\vec{w}_0,\vec{b}_0)$. During training, the gradient of the loss function with respect to the weight and bias parameters is used to find the direction for each parameter that minimises the loss function. A local minimum in the parameter space can be found by repeatedly calculating the gradient and slightly changing the parameters to reduce the loss function. This procedure is called gradient descent with the optimised parameters for every new epoch given by:

$$\begin{aligned} \vec{w}_{\text{epoch}+1} &= \vec{w}_{\text{epoch}} - \eta \cdot \nabla_{\vec{w}} \mathcal{L} \\ \vec{b}_{\text{epoch}+1} &= \vec{b}_{\text{epoch}} - \eta \cdot \nabla_{\vec{b}} \mathcal{L} \end{aligned} \tag{3.6}$$

with the learning rate η as a hyper-parameter of the training and the loss function $\mathcal{L}$ that depends on the current network parameterisation $f_{\text{NN}}(\vec{w}_{\text{epoch}},\vec{b}_{\text{epoch}})$. The gradient descent algorithm is not guaranteed to converge to a global minimum or to converge at all, however, in practice it yields good results for supervised learning provided a reasonable choice of the hyper-parameter η [52, 53].

The neural network training can be accelerated by modifying the gradient descent method such that it does not use all training data simultaneously. The parameter update steps in the standard gradient descent method averages over the complete training data to compute the gradient. This means that the computing time for each training epoch scales linearly with the number of pairs N in the

training data, i.e., with $\mathcal{O}(N)$. The computing time can be decreased to $\mathcal{O}(1)$ by calculating the gradient for the parameter update with only one randomly chosen training pair. By this, the training becomes a stochastic process and is therefore called *stochastic gradient descent* [54]. However, this modification has the disadvantage that a single optimisation step highly depends on the drawn training pair and thus, the training process may be exposed to significant fluctuations. An interim solution inheriting the advantage of a shorter computation time per epoch and avoiding the disadvantage of high fluctuations is *minibatch stochastic gradient descent* [55]. Here, a minibatch with fixed size m is randomly drawn from the complete training data with $1 < m < n$. The gradient for the gradient descent step is now averaged over the minibatch. This modification of the standard gradient descent method accelerates the training banking on the fact that although the computing time is shorter by a factor $\frac{n}{m}$, the information in the gradient is similar compared to using the complete training data for a reasonable choice of m.

The loss function's gradients with respect to the weights and biases of the network can be computed efficiently by the *backpropagation* algorithm [56]. The basic idea of this algorithm is to calculate all neuron activations with one feedforward propagation through the network and to calculate all derivatives of the loss function with one successive backpropagation through the network. Each neuron's activation can be written in terms of the activations in the previous layer in the following matrix form:

$$a^l = \sigma(w^l a^{l-1} + b^l) \tag{3.7}$$

with the activation function σ applied elementwise, the current layer l, the weights w^l from layer l as a matrix, and the activations and biases a^l, b^l from layer l as a vector, respectively. By using Equation (3.7) all neuron's activations can be calculated successively from the first hidden layer $l = 1$ to the output layer $l = L$. It is after that when the backpropagation algorithm starts. First, the error in the output layer $l = L$ is calculated which measures the loss' dependence on the last layer's neuron activations and how fast the activation function is changing at the given weighted inputs:

$$\delta^L = \nabla_a \mathcal{L} \odot \sigma'(z^L) \tag{3.8}$$

with the Hadamard product $(\vec{x} \odot \vec{y})_j = x_j y_j$, the activation function's derivative σ', and the weighted inputs of all neurons from the layer L in vector form z^L. After calculating the error in the output layer, the error can be propagated backwards through the network via:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{3.9}$$

for the remaining layers $l = L - 1, \ldots, 1$. The backpropagated errors can now be used to calculate the desired derivatives with respect to all weights and biases in

the network:

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l \tag{3.10}$$

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l. \tag{3.11}$$

These derivatives, or better, the gradients $\nabla_{\vec{w}} \mathcal{L}$ and $\nabla_{\vec{b}} \mathcal{L}$ can be used for training the neural network via gradient-based optimisation methods like gradient descent.

Not all available data is used when training a neural network, but the data is split into a training set and a validation set. As mentioned above, the objective while training a neural network is to minimise some loss function $\mathcal{L}$, e.g. Equation (3.5). However, it is essential to clarify that the general goal of training a neural network is not to find the network that yields the smallest possible value for $\mathcal{L}$. Instead, the desired outcome is that the trained neural network works out the underlying correlation between the input data and the output data and generalises it to unknown data. The resulting neural network should be as general as possible, i.e., it should not only map input samples from the known training data to the correct output but also generalise to previously unseen data. Therefore the complete data is split into two: one large training set and one smaller validation set. After carrying out the training using the data from the training set, the network is tested on the unknown validation set. The training is considered successful if the loss is also small on the validation set, which indicates a network with good generalisation properties.

What is the training process for a classical neural network?

A classical neural network can be trained in a **supervised learning** setting where it has access to a **training set with input and output data pairs**. As a guide for training, a **loss function** can be defined, which becomes smaller if the network's output for specific input data is close to the expected output data. The network can be trained by modifying its weight and bias parameters to minimise the loss function. The network parameters can be optimised via **gradient descent** by moving in small steps along the gradient of the loss function towards the minimum. The mentioned gradient can be efficiently computed via the **backpropagation algorithm**.

3.3 Challenges and further techniques

One of the main challenges of training a neural network is to find hyper-parameters that yield a well-performing network for a given learning task. The main hyper-parameters are the learning rate η from Equation (3.6), the network architecture,

the minibatch size, and the number of training epochs. It is very rarely successful to choose an extensive neural network and just train it with some guesses for the batch size and the learning rate. Instead, a common strategy is to simplify the learning task [15] by either decreasing the problem complexity, i.e., choosing some self-contained subset of the training data or using a small neural network with no or only one hidden layer. For a fixed minibatch size and the simplified task, the first goal should be to find a learning rate η that achieves any non-trivial learning. By starting with a random guess for the learning rate, the training loss can be monitored and used to indicate whether to decrease or increase the learning rate. If the training loss oscillates or increases, the learning rate should be decreased, otherwise, the learning rate should be increased until the former happens. The learning rate should then be chosen as large as possible such that the loss still decreases reliably. Using the chosen learning rate, the optimal minibatch size can be found by comparing different minibatch sizes in terms of the accuracy reached in a fixed real computation time. For the chosen learning rate and minibatch size, the problem complexity and the number of hidden network layers can gradually be increased, and the above steps repeated. The number of training epochs should be chosen to yield a generalising neural network without overfitting the training data.

Often, training a neural network for a large number of training epochs or a small training set suffers from overfitting the training data, which leads to poor generalisation to unseen data [57]. This is a natural phenomenon of minimising the loss function from Equation (3.5) via minibatch stochastic gradient descent. In the early training process of updating the network parameters via Equation (3.6), the gradients are usually large and contain very general information towards a better network. During this phase, the network learns the fundamental features of the training data and therefore, these network improvements generalise to unseen data. Later in the training process, the gradients yield very specific information that does not target the fundamental features of the training data but individual features of specific training pairs. The resulting changes to the network do not generalise to unseen data anymore. This effect is reflected in the fact that at later training times, the loss on the training set continues to fall while the loss on the unseen validation set starts to increase. The natural answer to this problem is early stopping [58], i.e., the neural network training is stopped as soon as the loss on unseen data stops to decrease. For this purpose, the training set is again divided into two: a large training set to compute the loss gradients and a smaller *test set* to monitor the loss and to detect overfitting. Overall, the data is split into three sets with the training goals, ensuring and validating a generalising neural network: the training set, the test set and the validation set. Other relevant techniques are the occasional dropout [59] of random neurons and soft weight-sharing [60].

A common challenge when training neural networks with many hidden layers and the sigmoid activation is the *vanishing gradient problem* [61]. It describes the

effect that the gradients with respect to the weights and biases decrease exponentially while moving backwards through the network, i.e., the gradients become very small and effectively vanish for the early layers. This can be explained by the gradient calculation via backpropagation where the gradient of an early layer $l = l_{\text{early}}$ depends on the gradients of the subsequent layers $l = l_{\text{early}} + 1, \ldots, L$ and the connecting weights. Thus, according to Equation (3.9), for each layer from the final layer $l = L$ to the layer of interest $l = l_{early}$, the following reducing factor is accumulated:

$$w^l \sigma'(z^l) < \frac{1}{4} \tag{3.12}$$

with the weights usually initialised as $|w^l| < 1$ and the derivative of the sigmoid activation $\sigma'(z^l) \leq \frac{1}{4}$ everywhere. The vanishing gradients cause different learning speeds for each layer and can effectively prevent the early layers from learning completely. This prevents the neural network from harnessing its full computation power and thus, makes its use impracticable. To resolve the vanishing gradient problem, the error propagation factor from Equation (3.12) should be closer to 1. This can be achieved by using another activation function, e.g., the *rectified linear unit*:

$$\rho(z) = \max(0,z) \tag{3.13}$$

with a derivative $\rho'(z) = 1$ for all inputs $z > 0$. It is for this reason of avoiding the vanishing gradient problem that the rectified linear unit is one of the most popular activation functions when working with deep neural networks [62].

What are the challenges of training classical neural networks?

There are several practical challenges when training classical neural networks. First, there are **many free and a priori unknown parameters**, the best strategies to find these are based on the idea of **systematic trial-and-error**. Another challenge is the problem of **overfitting** which means that the networks perform well on the training data but poorly on unseen data with the same internal features. This can be dealt with by **early stopping** the training before the generality is lost. A technical problem is the **vanishing gradient problem** which prevents the training of deep networks. This can be addressed by using the **rectified linear unit** instead of the sigmoid neuron.

Chapter 4

Quantum Neural Networks

After building the foundation by discussing the basic principles of quantum computing and classical neural networks, it is possible to merge the two concepts into what is called *quantum neural networks*. This chapter will guide through the implementation of the latter by answering the following questions: *What does a quantum neural network look like, and how is it trained? What are specific definitions for quantum neural networks? What is the concrete implementation of these networks and the training algorithm as a quantum circuit?*

In Section 4.1, the general notions of quantum neural networks will be discussed. This includes classifying quantum neural networks in the broader area of variational quantum algorithms and a general explanation of how a loss function is used for training. In addition, there will be a short review of the most relevant architectures and challenges for variational quantum algorithms. Thereupon, Section 4.2 becomes more specific with a thorough definition of two concrete quantum neural networks, namely, the dissipative quantum neural network and the quantum approximate optimisation algorithm. These will be implemented in Section 4.3 as concrete quantum circuits with different realisations of the quantum gates and requirements in the number of qubits. Additionally, a supervised learning algorithm will be introduced that can be used with any parameterised quantum neural network, including a quantum circuit sequence for evaluating the network's loss function during training. Section 4.4 presents some results, namely, a comparison of the effect of quantum gate noise on both networks and their training on actual quantum computers.

Please note that the contents of this chapter have partly been published already in the collaborative work [63]. In particular, this concerns the Sections 4.3 and 4.4.

4.1 General notions of quantum neural networks

Loss function and training

A *quantum neural network* (QNN) as a tool for *quantum machine learning* can be understood as a quantum analogue to the classical neural network and its role in classical machine learning. During this work, QNNs are treated as tools that can be adapted to specific tasks rather than fixed constructs - which makes one of its strengths in quantum computing in general. QNNs belong to the family of *variational quantum algorithms* (VQAs) [23] whose shared general notions will therefore be reviewed here. The additional term of VQAs is mainly introduced here to enable a broad overview of the literature while stating which ideas are more or less related to this work's focus on QNNs.

VQAs employ *parameterised quantum circuits* (PQCs) [16, 20–22] as the parts that perform the actual quantum computing. Such a quantum circuit can be modified through continuous or discrete parameters $\vec{\theta}$ such that the operating part of the PQC is given by some variational unitary transformation $U(\vec{\theta})$. The training of such a PQC happens in a hybrid quantum-classical manner [16–19], the PQC is executed on a quantum computer, and the results are used to update its parameters $\vec{\theta}$ via classical side computation. Like this, no quantum resources are wasted for what can efficiently be done classically.

The interplay of quantum and classical computations lies at the heart of why VQAs are so well suited for today's quantum computers. These so-called NISQ devices are highly impaired through noise entering with each quantum gate [6]. This limits the number of quantum gates that can be executed in parallel within one quantum circuit - called quantum circuit depth - before the qubits are overwhelmed with noise. By executing only short quantum circuits, the noise can be reduced, which is made possible by outsourcing the training procedure to classical computation. By contrast, famous quantum algorithms such as Shor's factoring algorithm [4] or Grover's search algorithm [5] require much more gates than tolerable and are therefore not suitable to run on NISQ devices.

The training of a PQC is, similar to classical neural networks, usually done with the objective to find parameters $\vec{\theta}^*$ that minimise (or alternatively maximise) some loss function $\mathcal{L}(\vec{\theta})$:

$$\vec{\theta}^* = \min_{\vec{\theta}} \mathcal{L}(\vec{\theta}). \tag{4.1}$$

The loss function usually averages over different input (and possibly output) states of a training set, i.e., the PQC is evaluated multiple times for different training states. To find an optimal set of parameters $\vec{\theta}$, the gradient of the loss function $\nabla_{\vec{\theta}}\, \mathcal{L}(\vec{\theta})$ can be exploited via the gradient descent method, as discussed in Section 3.2 for classical neural networks, or related methods [17].

For the VQA to solve a given problem successfully, the loss function as the connecting part between quantum computation and problem definition has to fulfil

some conditions [23]. First and foremost, the loss function has to be meaningful to the problem, i.e., it has to incorporate the problem such that minimising the loss function yields better solutions and reaching the global minimum is only possible for a PQC that solves the problem. Another necessary condition is that the loss function can efficiently be computed on a quantum computer. The computation via the PQC on a quantum computer should be more efficient than on a classical computer. Otherwise, there can never be any quantum advantage. Finally, the loss function should offer a non-vanishing gradient almost everywhere such that the PQC is trainable, i.e., the gradient-based methods can find a global minimum with a reasonable number of optimisation steps.

The gradient of the loss function can either be calculated via the central-difference approximation or the parameter-shift rule. In both cases, the gradient with respect to the parameter $\vec{\theta}_k$ is calculated by evaluating the PQC twice with $\vec{\theta}_k$ shifted by an amount ϵ in either direction. For a general PQC with some unitary $U(\vec{\theta})$, it is always possible to compute the gradient via the central-difference approximation:

$$\nabla_k \mathcal{L}(\vec{\theta}) = \frac{\mathcal{L}(\vec{\theta} + \epsilon\vec{e}_k) - \mathcal{L}(\vec{\theta} - \epsilon\vec{e}_k)}{2\epsilon} + \mathcal{O}(\epsilon^2) \tag{4.2}$$

where $\vec{e}_k$ has components $e_k^j = \delta_k^j$ and $\epsilon > 0$. The parameter-shift rule can be applied to a PQC with the special form $U(\vec{\theta}) = V_m U_m(\theta_m) \dots V_1 U_1(\theta_1)$ with constant arbitrary circuits V_i and rotation-like gates $U_i(\theta_i)$. A rotation-like gate is generated by an hermitian operator H_j with the condition $H_j^2 = \mathbb{1}$ such that $U_j = e^{-\frac{i}{2} H_j \theta_j}$. The parameter-shift rule gives the exact formula for the loss function's gradient [64]:

$$\nabla_k \mathcal{L}(\vec{\theta}) = \frac{\mathcal{L}(\vec{\theta} + \epsilon\vec{e}_k) - \mathcal{L}(\vec{\theta} - \epsilon\vec{e}_k)}{2\sin(\epsilon)} \tag{4.3}$$

where again $\vec{e}_k$ has components $e_k^j = \delta_k^j$ and $\epsilon > 0$. Please note that for small ϵ, the approximation $\sin(\epsilon) = \epsilon$ can be made and thus, Equations (4.2) and (4.3) become equivalent.

Architectures

There are two fundamentally different approaches to designing the PQC for a VQA: the problem-inspired ansatz and the problem-agnostic ansatz. If helpful information about the problem is available, e.g., in the form of a Hamiltonian, this information may be used to determine the structure of the PQC. Consequently, these problem-inspired ansatzes look different for specific problems, which can lead to simpler training because the general structure of the problem is already incorporated. In contrast, problem-agnostic ansatzes can be used without specific

information about the problem, here the focus may be on the reduction of noise in the circuit, e.g., for the hardware-efficient ansatz [65], or the general computational possibilities. QNNs, as they are understood in this work, fall into the category of problem-agnostic ansatzes. Although different levels of complexity can be chosen for the QNN architecture, their general structure does not depend on the problem at hand.

A QNN is, in analogy to classical neural networks, characterised by its fundamental building block, the *quantum perceptron*. The quantum perceptron is usually repeatedly applied within one QNN, often in parallel on different qubits to form a layer, whereby multiple layers can be used subsequently. Therefore, the definition of the quantum perceptron and its embedding into the quantum circuit are crucial parts and a few proposals from the literature are briefly reviewed in the following.

In [25], the QNN is defined as a sequence of L variational layers $B_l(\vec{\theta}_l)$ and final parameterised single qubit gates $G_{\text{final}}(\vec{\theta}_{\text{final}})$ on their measured output qubits:

$$U(\vec{\theta}) = G_{\text{final}}(\vec{\theta}_{\text{final}})B_L(\vec{\theta}_L)\dots B_1(\vec{\theta}_1). \tag{4.4}$$

Each variational layer $B_l(\vec{\theta}_l)$ implements a parameterised single-qubit gate on each qubit. After that, a cyclic node, i.e., a controlled parameterised single-qubit gate on each qubit where the control proximity range r determines which qubit is used as the control qubit, respectively. This choice is motivated by the fact that single and two-qubit gates are universal for quantum computing, and the cyclic node strongly entangles the qubits.

A more general definition of a QNN is given by [24], who define the quantum perceptron as an arbitrary unitary and build the QNN via completely positive layer-to-layer transition maps in a dissipative manner. As opposed to the QNN discussed before, each layer features an independent set of qubits, and the quantum perceptron acts on all qubits of a previous layer and one qubit of the current layer. In this manner, a set of quantum perceptrons connects all qubits of subsequent layers, whereby the quantum perceptron itself can be any arbitrary unitary. This proposal of a QNN is shown to be universal for quantum computation and will therefore be presented later in Section 4.2 in more detail.

The work of [26] discusses the use of global and local quantum perceptrons for dissipative QNNs. Here, a global perceptron acts, similar to [24], on all qubits from the previous layer and one qubit from the current layer, while a local perceptron only acts one a subset of the qubits from the input layer and one qubit from the current layer. Again, the full QNN consists of L subsequent layers connecting two different sets of qubits. The layers are built by assigning a local or global perceptron to each qubit. The single quantum perceptron V_j^l corresponding to the k^{th} qubit in the l^{th} layer is defined as:

$$V_j^l(\vec{\theta}_j^l) = \prod_k R_k(\theta_k)W_k \tag{4.5}$$

with a constant unitary W_k, and a parameterised unitary $R_k(\theta_k) = e^{-\frac{i}{2}H_k\theta_k}$ where H_k is a hermitian matrix with $\mathrm{Tr}(H_k^2) \leq 2^n + 1$ with the number of qubits n from the previous layer that the perceptron is acting on. They found that the QNN with a local perceptron is more easily trained in some cases.

Another popular ansatz is that arising in the *quantum approximate optimisation algorithm* (QAOA), which features a sequence of alternating unitary operators [27, 66, 67] that can be interpreted as a quantum perceptron. Originally, the QAOA has been proposed as a problem-inspired ansatz for combinatorial optimisation problems, where one of the unitary operators is generated by the Hamiltonian describing the problem system. By now, it is also used as a problem-agnostic ansatz as it has been shown that it is universal for quantum computation [68]. It is for this reason that it will be presented later in Section 4.2 alongside the QNN from [24] in more detail.

Applications

The framework of VQAs, i.e., using the computational power of a PQC and training its parameters classically, is shown to be universal for quantum computing [69]. This motivates the application of VQAs to a wide range of problems, and in fact, these models have been successfully implemented on a variety of tasks - a few of these are reviewed in Appendix A.

Challenges

One of the main challenges of working with VQAs is the presence of *Barren plateaus* which describe the phenomenon of exponentially vanishing loss gradients in the number of qubits n [70]. This means that the loss landscape with respect to the parameters becomes increasingly flat for VQA models that exploit many qubits n. As the training of a PQC is usually carried out via a gradient-based method such as gradient descent, the training performance strongly relies on the properties of the loss gradient. If for some not optimal parameters $\vec{\theta}_i$ the gradient $\nabla_{\vec{\theta}}\mathcal{L}(\vec{\theta})\Big|_{\vec{\theta}=\vec{\theta}_i}$ is close to zero for all components, the optimisation of the parameters will stagnate as $\vec{\theta}_{i+1} \approx \vec{\theta}_i$ and the model can not improve towards the global minimum.

If Barren plateaus prevent a VQA from learning depends on the specific design of the PQC and the loss function. Particularly impaired are random circuits without any implicit structure, which form 2-designs [70]. Another relevant property based on [71] is whether a global or a local loss function is used. Here, a global loss function refers to a quantity based on a measurement of all output qubits, while a local loss function only depends on a small subset of the output qubits. Using a local loss function instead of a global loss function may help avoid the Barren plateau phenomenon. What is more, the use of local quantum perceptrons as described in the previous subsection may additionally reduce the effect of Barren plateaus on

dissipative QNN models [26]. Another way of addressing the problem of exponentially small gradients, especially at the beginning of the training, is not to initialise the parameters randomly but with the following strategy [72]: initialise a subset of the parameters randomly and choose the remaining parameters such that the circuit consists of blocks of shallow circuits that each evaluate to the identity.

Another main challenge in executing VQAs on NISQ devices is the impact of quantum gate noise when employing higher-depth quantum circuits. As already mentioned, the comparably short depth of quantum circuits used in VQAs makes them practical in the first place. However, as the application tasks become more complex, the PQCs must be extended to offer enough computational power. The higher the quantum circuit depth, the more noise is induced through the application of quantum gates [6] until eventually, the evaluation of the loss function and its gradient becomes too noisy to be used for the training [73, 74]. Additionally, if the depth of the circuit scales linearly with the dimension of the problem, this can lead to a noise-induced Barren plateau [75] which further impairs the training of VQAs on NISQ devices.

What does a quantum neural network look like, and how is it trained?

A quantum neural network is implemented as a **parameterised quantum circuit**. Such a quantum neural network, as treated in this work, is characterised by the definition of the **quantum perceptron** as its fundamental building block. A quantum neural network is usually **trained in a hybrid quantum-classical manner**: a loss function is evaluated by executing the parameterised quantum circuit on a quantum computer, and its parameters are optimised via **gradient descent** as a classical side computation. One of the main challenges is the **Barren plateau phenomenon** which describes exponentially vanishing gradients in the number of qubits.

4.2 Definition of two quantum neural networks

Dissipative quantum neural network

In [24], a *dissipative* QNN (DQNN) has been introduced, which features a completely positive map as its building block and implements each neuron as a separate qubit. For training the DQNN, they introduced a quantum version of the classical backpropagation algorithm, which can be simulated classically. Using this algorithm, their DQNN succeeded in learning unitary transformations in a supervised manner from a set of training states. In particular, their numerical results indicate a well-generalising QNN architecture and high robustness to noisy data. Addition-

ally, no disturbing effects of the Barren plateau phenomenon have been observed during their studies. Motivated by these promising results, their DQNN definition will be reviewed here and used as a starting point for implementing a DQNN for the execution on NISQ devices.

The DQNN architecture is composed of one input layer ($l = 0$) with m_0 neurons, $L \geq 0$ hidden layers ($l = 1, \ldots, L$) with m_l neurons, respectively, and one output layer ($l = L + 1$) with m_{L+1} neurons. The neurons or better, the qubits in the input layer $l = 0$ are initialised in a possibly mixed input state ρ_{in} that depends on the task at hand. The qubits from the remaining layers $l = 1, \ldots, L + 1$ are initialised in the vacuum product state $|0 \ldots 0\rangle_l$ independent of the task.

Two subsequent layers, $l-1$ and l, are fully connected through a set of quantum perceptrons each of which is defined to be a general unitary operator U_j^l acting on the j^{th} qubit of the l^{th} layer and all m_{l-1} qubits of the preceding $(l-1)^{\text{th}}$ layer. Hereafter, two subsequent layers are also called input and output layers. The full layer unitary U^l combines the quantum perceptrons U_j^l for each qubit j from the output layer l as $U^l = U_{m_l}^l \ldots U_1^l$ and consequently, acts on all qubits in the respective input and output layer. As the quantum perceptrons U_j^l are arbitrary unitaries, it is important to note that they do not, in general, commute. Thus, the application order from the first to the last neuron in the output layer must be maintained.

The building block of the DQNN is given by the completely positive layer-to-layer transition map:

$$\mathcal{E}^l(X_{l-1}) = \text{Tr}_{l-1}\left(U^l \left(X_{l-1} \otimes |0 \ldots 0\rangle_l \langle 0 \ldots 0|\right) U^{l\dagger}\right) \tag{4.6}$$

with the partial trace Tr_{l-1} over all qubits in the input layer $l-1$, and the state of the qubits in the input layer X_{l-1} before applying the layer unitary U^l. The above equation explains why this QNN architecture is called dissipative: after applying the layer unitary, all qubits from the input layer are traced out, i.e., the qubits from previous layers are no longer used, but only the information that propagated into the next layer is preserved. The following notation as an iterative application of the map $\mathcal{E}^l$ and the propagation of the information from the input state ρ_{in} through all network layers highlights the feedforward nature of the DQNN:

$$\rho_{\text{out}} = \mathcal{E}^{L+1}\left(\mathcal{E}^L\left(\ldots \mathcal{E}^1\left(\rho_{\text{in}}\right)\right)\right) \tag{4.7}$$

with the output state ρ_{out} as the state of the qubits in the output layer $l = L + 1$.

The output state of the DQNN can also be written in a non-iterative fashion by combining all quantum perceptrons to one network unitary $\mathcal{U}$ and tracing out all qubits corresponding to the input layer and the hidden layers afterwards:

$$\rho_{\text{out}} = \text{Tr}_{\text{in,hid}}\left(\mathcal{U}\left(\rho_{\text{in}} \otimes |0 \ldots 0\rangle_{\text{hid,out}} \langle 0 \ldots 0|\right)\mathcal{U}^\dagger\right) \tag{4.8}$$

with the network unitary $\mathcal{U} = U^{L+1}U^L \dots U^1$. Please not that each layer unitary U^l only acts on the qubits from layers $l-1$,l and is extended by identities for the other qubits. The notation from Equation (4.8) goes well with the interpretation as a quantum circuit as it highlights the necessary steps in the correct order: first initialise the input qubits in ρ_{in} and the remaining qubits in the vacuum state $|0\rangle$, then apply all unitary operations combined in $\mathcal{U}$, and finally, measure only the output qubits. However, an interpretation of the DQNN as a quantum circuit will be the topic of section 4.3.

Quantum approximate optimisation algorithm

The quantum approximate optimisation algorithm (QAOA) [27], sometimes called *quantum alternating operator ansatz* with the same acronym, has initially been proposed for finding approximate solutions to combinatorial optimisation problems, such as the constraint satisfaction problem [76] and the MaxCut problem [77]. However, the QAOA can also be very easily interpreted and used as a general QNN ansatz. How this can be done is discussed in the following. This lays the foundation for implementing the QAOA for the execution on NISQ devices and comparing it to the previously discussed DQNN.

Typically, the combinatorial problem is encoded into a problem Hamiltonian H_P whose ground state than represents the optimal solution to the problem. To prepare this ground state, an additional mixer Hamiltonian H_M is introduced which is used together with the problem Hamiltonian H_P to generate unitary operators. By alternately applying the unitary operators $e^{-it_l H_P}$ and $e^{-i\tau_l H_M}$ for time steps $t_l,\tau_l \in \mathbb{R}$ to some initial state $|\psi_0\rangle$, the parameterised state $|\psi\left(\vec{\tau},\vec{t}\right)\rangle$ is prepared:

$$|\psi\left(\vec{\tau},\vec{t}\right)\rangle = e^{-i\tau_L H_M} e^{-it_L H_P} \dots e^{-i\tau_1 H_M} e^{-it_1 H_P} |\psi_0\rangle \,. \tag{4.9}$$

The goal of the QAOA is to find parameters $\vec{\tau},\vec{t}$ that minimise the following loss function:

$$\mathcal{L} = \langle\psi\left(\vec{\tau},\vec{t}\right)| H_P |\psi\left(\vec{\tau},\vec{t}\right)\rangle \tag{4.10}$$

which is usually done via the typical gradient descent method for VQAs.

However, the idea of alternating operators can also be generalised to other tasks without a specific problem Hamiltonian. In this case, for a problem that can be represented by m qubits, two 2^m-dimensional hermitian matrices A, B are randomly generated from the gaussian unitary ensemble [78]. Then, the QAOA consists of the following sequence of alternating unitary operators:

$$\mathcal{U} = e^{-i\tau_L B} e^{-it_L A} \dots e^{-i\tau_1 B} e^{-it_1 A} \tag{4.11}$$

with $t_l,\tau_l \in \mathbb{R}$. The resulting unitary $\mathcal{U}$ can be interpreted as the network unitary consisting of L quantum perceptrons of the form $U^l = e^{-i\tau_l B} e^{-it_l A}$. Although

there is no immediate notion of neurons and layers for the QAOA, the unitary U^l can be thought of as acting on the neurons of an imagined $(l-1)^{\text{th}}$ layer and storing the resulting state in the neurons of the subsequent l^{th} layer. As the QAOA actually acts on the same m qubits all the time, the notion of storing the resulting state in the next layer's neurons comes for free.

What are specific definitions for quantum neural networks?

A specific quantum neural network defines a **map, which repeatedly implements some quantum perceptron** and a set of qubits the map acts on. These quantum perceptrons can either always act on the same set of qubits or act on different qubits for different network layers. The latter is commonly called a dissipative ansatz. One example for the first approach is the **quantum approximate optimisation algorithm** which implements alternating unitary operators generated by two different Hamiltonians on the same qubits. An example for the second approach is the **dissipative quantum neural network** which employs completely positive layer-to-layer transition maps between different qubits.

4.3 Implementation as quantum circuits

Quantum circuit for the DQNN

The DQNN intends to realise each neuron as a separate qubit. Thus, implementing the DQNN as a quantum circuit requires $M = \sum_{l=0}^{L+1} m_l$ qubits. This results in a 2^M-dimensional Hilbert space $\mathcal{H}^{\otimes M}$ which is the tensor product of M single qubit Hilbert spaces.

The main task of implementing the DQNN described by equation (4.8) is to find an appropriate realisation of the quantum perceptron U_j^l which is a general unitary acting on $m_{l-1}+1$ qubits. For the classical simulation of the DQNN done by [24], it was sufficient to abstractly define the unitary matrix and update its entries during the training. However, to execute the DQNN on a quantum computer, a concrete realisation in the form of parameterised quantum gates is necessary to build. Once the parameterised quantum gates for representing the quantum perceptron are chosen, the full PQC can be built by composing the respective quantum perceptrons from all layers. When thinking about possible candidates for parameterised quantum gates, two objectives have to be well-balanced: on the one hand, the final realisation of the quantum perceptron should be as universal as possible, while on the other hand, the number of quantum gates and parameters should be kept as small as possible. If either one of these objectives is neglected, the DQNN will not perform as well as its classically simulated model.

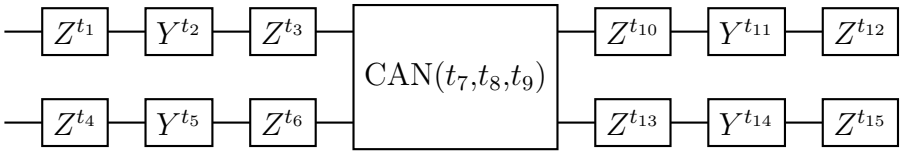
Any arbitrary two-qubit unitary can be expressed by a two-qubit canonical gate and twelve single-qubit gates [79]. The two-qubit canonical gate is defined via three parameters as:

$$\begin{aligned} \mathrm{CAN}(t_x,t_y,t_z) &= e^{-i\frac{\pi}{2}t_x X\otimes X} e^{-i\frac{\pi}{2}t_y Y\otimes Y} e^{-i\frac{\pi}{2}t_z Z\otimes Z} \\ &= \mathrm{RXX}(t_x\pi)\ \mathrm{RYY}(t_y\pi)\ \mathrm{RZZ}(t_z\pi) \end{aligned} \tag{4.12}$$

where $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $Y = \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right)$, $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ are the Pauli matrices, the RXX/RYY/RZZ gates are parameterised two qubit gates commonly available in quantum computing libraries, and $t_{x,y,z} \in \mathbb{R}$ are the parameters. The necessary single qubit gates are parameterised Pauli-Y and Pauli-Z operators. These are equivalent to the following rotations around the y- and the z-axis:

$$\begin{aligned} Y^t &\simeq R_Y(\pi t) = e^{-i\frac{\pi}{2}tY} \\ Z^t &\simeq R_Z(\pi t) = e^{-i\frac{\pi}{2}tZ} \end{aligned} \tag{4.13}$$

up to a phase factor which is indicated by $\simeq$. By executing the two-qubit canonical gate in addition to prepending and appending three single qubit gates to each qubit in the following form:



(4.14)

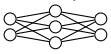
any arbitrary two-qubit gate can be performed. As a graphical simplification, the used sequence Z-Y-Z of single-qubit gates can be expressed as the commonly used single qubit gate $u(t_1,t_2,t_3)$:

$$u(t_1,t_2,t_3) = R_Z(t_2)R_Y(t_1)R_Z(t_3) = \begin{pmatrix} \cos(t_1/2) & -e^{it_3}\sin(t_1/2) \\ e^{it_2}\sin(t_1/2) & e^{i(t_2+t_3)}\cos(t_1/2) \end{pmatrix} \tag{4.15}$$

where the different parameterisation compared to Equation (4.13) should be noted.

The quantum perceptron is not, in general, a two-qubit unitary, therefore the universal two-qubit gate from 4.14 can not directly be used. When thinking about implementing the universal two-qubit gate, it is helpful to think about the task fulfilled by the quantum perceptron, which is to process the states of its input qubits and change the output qubit's state accordingly. This motivates the application of separate two-qubit gates on each input-output qubit pair. However, numerical studies have shown that it is sufficient and often advantageous to refrain from using the single-qubit sequence from Equation (4.15) and only use the two-qubit canonical gate as the direct realisation of the quantum perceptrons. In addition to realising the entire layer unitary U^l, i.e., all quantum perceptrons corresponding to layer l, the three-parameter single-qubit gate u is prepended to all input qubits

and appended to all output qubits. To append single-qubit gates on the input qubits is pointless, as these are no longer used. To prepend single-qubit gates on the input qubits has proven unnecessary in numerical studies.

The interpretation of the DQNN as a quantum circuit employing the previously discussed methods looks as follows. The first m_0 qubits are initialised in a given, possibly mixed state ρ_{in}, while all remaining qubits are initialised in the computational basis state $|0\rangle$. The quantum circuit and the general DQNN architecture are structured layer-wise and will therefore be described accordingly. The u gates are applied first, layer by layer ($l = 1, \ldots, L+1$), to the respective m_{l-1} input qubits. After that, the layer unitary $U^l = \prod_{j=m_l}^{1} U_j^l$ is applied to all input and output qubits. Here, U_j^l is a sequence of m_{l-1} CAN gates where the i^{th} CAN gate acts on the i^{th} input and the j^{th} output qubit. After each layer l, the m_{l-1} input qubits are neglected, i.e., they are just ignored for the rest of the quantum circuit. This layer's m_l output qubits serve as the input qubits for the next layer $l+1$. By this, the partial trace of Equation (4.8) is realised. After the output layer $L+1$, again, u gates are applied to the remaining m output qubits. Thus, the quantum circuit consists of $N_p = 3m + 3\sum_{l=1}^{L+1} m_{l-1}(1+m_l)$ parameters. This procedure is exemplary illustrated in Fig. 1b for a  DQNN.

Quantum circuit for the QAOA

The QAOA uses a constant number of qubits that depend not on the network architecture but only on the problem size. If the problem can be represented by m qubits, implementing the QAOA as a quantum circuit only requires m qubits. This results in a 2^m-dimensional Hilbert space $\mathcal{H}^{\otimes m}$ which is the tensor product of m single qubit Hilbert spaces. Thus, the Hilbert space on which the QAOA operates is always smaller than the DQNN's.

Implementing the QAOA as a quantum circuit is straightforward, and unlike for the DQNN, no preliminary considerations are required. The m qubits are initialised in the given, possibly mixed state ρ_{in} with dimension $d = 2^m$. After this, only the unitary operations from Equation (4.11) have to be realised. The number of alternating unitary operator pairs L as a hyper-parameter can be chosen freely according to the problem at hand. As a default, L is chosen to be $d^2/2$ because for the problem of learning an arbitrary unitary transformation, this guarantees the QAOA to converge to an optimal solution [78]. By using the hermitian matrices A,B which are randomly generated from the gaussian unitary ensemble, the individual unitary pairs $e^{-it_l A}$, $e^{-i\tau_l B}$ can be generated according to the parameters $\vec{t}$, $\vec{\tau}$. These unitaries are directly applied in the quantum circuit from right to left according to the operator notation in Equation (4.11). As a side note: there is no obvious way of interpreting each of these unitaries as another sequence of parameterised quantum gates similar to the DQNN; these are already parameterised via a single parameter, and it is essential to use the unitaries as generated by

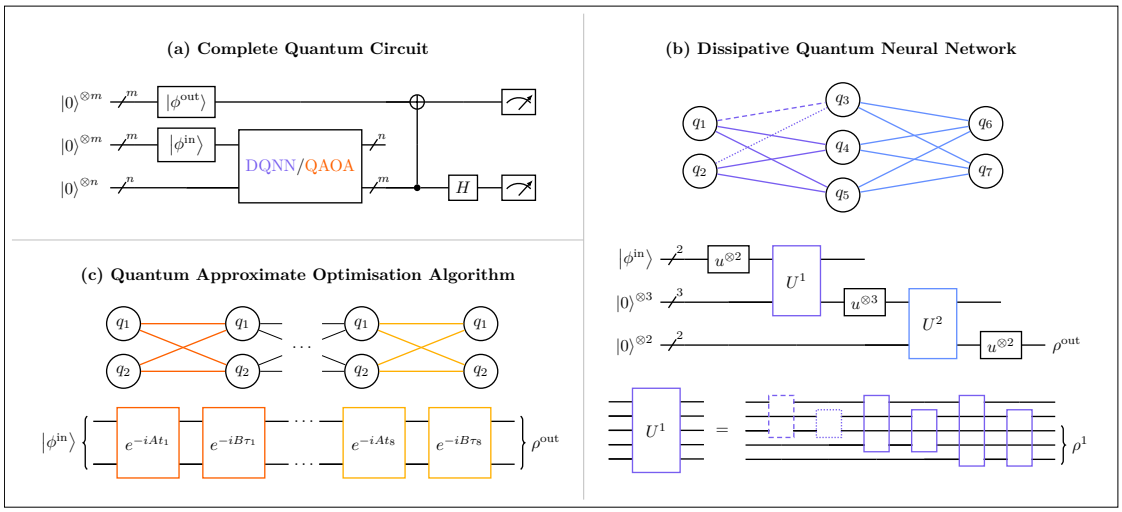


Figure 4.1: Quantum circuit implementation of the DQNN and the QAOA. The complete quantum circuit (a) consists of three parts: i) the initialisation of the first two m qubits in the target output state $|\phi_{\text{out}}\rangle$ and input state $|\phi_{\text{in}}\rangle$, respectively, ii) a QNN circuit that implements either the DQNN ($n \geq m$) or the QAOA ($n = 0$), and iii) the fidelity measurement between the first m and the last m qubits via the destructive swap test. Here, the variable $n > 0$ corresponds to the dissipative nature of the DQNN, i.e., its output state is stored in different qubits than the input state, while $n = 0$ covers the case of the QAOA which acts on only one set of m qubits. The DQNN (b) features one input layer, L hidden layers, and one output layer with m_l neurons, respectively. Each neuron is represented by an individual qubit. Two neighboured layers, $l-1$ and l, are connected through the layer unitary U^l. The QAOA (c) consists of a sequence of alternating unitary operators e^{iAt_l}, $e^{iB\tau_l}$ with $l = 1, \ldots, N$. It can be interpreted as an $(N+1)$-layer QNN with m neurons per layer where the operator pair l connects the layers $l-1$ and l. The quantum circuit only requires m qubits such that all unitaries act on the same qubits. The figure has already been published in the collaborative work [63].

the Hamiltonians. These unitaries are individually, and for each parameter choice differently, decomposed into quantum gates that can be executed later on a quantum computer. The quantum circuit consists of $N_p = d^2 = 4^m$ parameters. An exemplary quantum circuit for $m = 2$ is depicted in Fig. 1c.

Learning algorithm and measurement

A supervised learning algorithm will be explained for the remaining part of this section, where the QNN is treated as a black box. The learning algorithm can be used for both networks, the DQNN and the QAOA. The black box QNN expects a 2^m-dimensional, in this case, pure input state $|\phi^{\text{in}}\rangle$, and outputs an, in general, 2^m-dimensional mixed state ρ^{out}. It is parameterised via N_P parameters $\vec{\theta} \in \mathbb{R}^{N_P}$.

The task is to learn an unknown unitary transformation $V \in U(2^m)$ from a given training set $\{|\phi_x^{\text{in}}\rangle, |\phi_x^{\text{out}}\rangle\}_{x=1}^{N_T}$ where $|\phi_x^{\text{out}}\rangle = V|\phi_x^{\text{in}}\rangle$. This task is generally very interesting because it tests whether the QNNs are computationally universal in that they can compute any arbitrary unitary transformation. However, the

same learning algorithm could be employed if the training set would not represent a unitary transformation but any other unknown map.

As it is usual for VQAs, the learning task needs to be inscribed into a loss function, which can be used to train the parameterised black box QNN. The goal of training the QNN is that it maps each input state $|\phi_x^{\text{in}}\rangle$ to the corresponding output state $|\phi_x^{\text{out}}\rangle$. Thus, the training is successful if the QNN's output state ρ_x^{out} is close to the desired output state $|\phi_x^{\text{out}}\rangle$ for all x. Fortunately, there is a natural quantity that evaluates the closeness of two quantum states, namely the fidelity:

$$\mathcal{F}(\rho, |\phi\rangle) = \langle\phi|\,\rho\,|\phi\rangle \in [0,1] \tag{4.16}$$

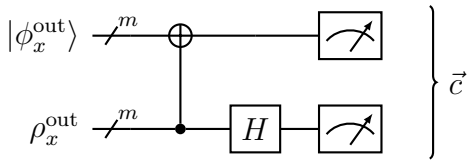
between an, in general mixed state ρ, and a pure state $|\phi\rangle$. The fidelity reaches its maximum $\mathcal{F} = 1$ if the two states are equal and reaches its minimum $\mathcal{F} = 0$ if the two states are orthogonal. Averaging over the fidelity between all QNN output states ρ_x^{out} and their corresponding desired output states $|\phi_x^{\text{out}}\rangle$ yields the loss function:

$$\mathcal{L}(\vec{\theta}) = \frac{1}{N_T}\sum_{x=1}^{N_T} \langle\phi_x^{\text{out}}|\,\rho_x^{\text{out}}(\vec{\theta})\,|\phi_x^{\text{out}}\rangle \tag{4.17}$$

where N_T is the number of training pairs, and $\rho_x^{\text{out}}(\vec{\theta})$ intrinsically depends on the QNN parameterisation $\vec{\theta}$. It is important to realise, that for this choice of a loss function, the training's objective will be to maximise it and not to minimise it as assumed in the discussion around Equation (4.1).

The loss function from Equation (4.17) is, in fact, a good choice as it meets the conditions discussed in Section 4.1. It is meaningful to the problem as it reaches its maximum if and only if the problem is solved. Additionally, it can be expected that the loss function offers a non-vanishing gradient almost everywhere, however, this has to be tested numerically for every specific QNN implementation. Finally, the loss function can be computed efficiently on a quantum computer due to its quantum mechanical nature and more efficiently than on a classical computer for high-dimensional systems.

Just now, it has been claimed that the loss function from Equation (4.17) can be computed efficiently on a quantum computer, however, it is not obvious how to measure the fidelity between two quantum states. First of all, in addition to the QNN's output state ρ_x^{out}, it is also necessary to have access to the corresponding desired output state $|\phi_x^{\text{out}}\rangle$. For this, another m ancillary qubits are necessary to be initialised in the state $|\phi_x^{\text{out}}\rangle$. If these requirements are met, the fidelity between both states can be calculated via the *destructive swap test* [80, 81] using the following circuit:



(4.18)

with the first m qubits in the state $|\phi_x^{\text{out}}\rangle$, the second m qubits in the state ρ_x^{out}, m CNOT gates and m Hadamard gates as described in Table 2.1, measurements of all $2m$ qubits in the computational basis, and classical post-processing of the measurement results indicated by $\vec{c}$. The classical post-processing works as follows. First, the measurement results are ordered from top to bottom into m pairs of qubits corresponding to one of the states respectively, i.e., the qubit measurements are ordered as $q_1, q_{m+1}, q_2, q_{m+2}, \ldots, q_{2m-1}, q_{2m}$ with q_i being the i^{th} qubit from the top. Due to the nature of quantum mechanics which only allows for discrete measurement results in the computational basis $|0\rangle$ or $|1\rangle$, the measurement is executed N times with the effect of quantum projection noise decreasing for larger N. The ratio of encountering the different measurement results is saved into $\vec{p}$ following the order of binary numbers, exemplary shown for $m = 1$: 00, 01, 10, 11. Using this definition, the components of $\vec{p}$ sum up to 1. The fidelity is then given by the inner product of the classical post-processing vector $\vec{c} = (1,1,1,-1)^{\otimes m}$ with the measurement results $\vec{p}$ as:

$$\mathcal{F}(\rho_x^{\text{out}}, |\phi_x^{\text{out}}\rangle) = \langle\phi_x^{\text{out}}| \, \rho_x^{\text{out}} \, |\phi_x^{\text{out}}\rangle = \vec{p} \cdot \vec{c}. \tag{4.19}$$

One of the advantages of the destructive swap test is its constant circuit depth of 2 as the m CNOT gates and the m Hadamard gates can be executed in parallel. The classical post-processing can be performed in linear time with respect to m. Another advantage is that the destructive swap test requires no ancillary qubits in addition to the $2m$ qubits, which represent the quantum states.

The parameterised black box QNN is trained iteratively by gradient descent. At the beginning of the learning algorithm, the N_P independent network parameters are initialised as $\vec{\theta}_0 = (\theta_1, \ldots, \theta_{N_P})^T$. At each epoch, all parameters are updated such that they maximise the loss function from Equation (4.17). According to the gradient descent method, the new parameters are then given by $\vec{\theta}_{t+1} = \vec{\theta}_t + d\vec{\theta}$. The update step is defined as $d\vec{\theta} = \eta \vec{\nabla} \mathcal{L}(\vec{\theta}_t)$ with the learning rate η. The gradient of the loss function is approximated parameter wise by:

$$\nabla_k \mathcal{L}(\vec{\theta}_t) = \frac{\mathcal{L}(\vec{\theta}_t + \epsilon \vec{e}_k) - \mathcal{L}(\vec{\theta}_t - \epsilon \vec{e}_k)}{2\epsilon} + \mathcal{O}(\epsilon^2) \tag{4.20}$$

where $\vec{e}_k$ has components $e_k^j = \delta_k^j$, $k,j = 1,\ldots,N_p$ and $\epsilon > 0$. The use of the parameter-shift rule from Equation (4.3) is waived because the necessary assumptions can not be made for an unknown black box QNN, and particularly not for the QAOA. However, by using a small ϵ the error of Equation (4.20) can be neglected. In the following, ϵ and η are called the hyper-parameters of the QNN training and may vary widely for different QNN choices.

In addition to the loss function Equation (4.17) for training the QNN, commonly called the *training loss*, it is important to define an additional *validation loss*. The purpose of the validation loss is to test whether the QNN training

generalises to unseen data with the same internal properties or if the QNN only performs on the data from the training set. For this, an additional validation set $\{|\phi_x^{\text{in}}\rangle, |\phi_x^{\text{out}}\rangle\}_{x=1}^{N_V}$ is built in the same way as the training set. The validation loss is defined analogously to the training loss:

$$\mathcal{L}_V = \frac{1}{N_V} \sum_{x=1}^{N_V} \langle \phi_x^{\text{out}} | \rho_x^{\text{out}} | \phi_x^{\text{out}} \rangle. \tag{4.21}$$

Now that all tools for training the black box QNN are defined, the supervised learning algorithm can be put together. It involves three crucial steps.

i. Initialisation : First, a set of training pairs and a set of validation pairs, if not externally given, are generated from a random target unitary $V \in u(d)$. These sets implicitly define the problem that the respective QNN should solve. The hyper-parameters are set according to the problem and network at hand. The network parameters $\vec{\theta}_0$ are initialised randomly.

ii. Learning: The following steps are repeated for a previously fixed or in real time adjusted number of epochs. The full quantum circuit consists of three parts and is depicted in Fig. 1a. In the first part, a set of m qubits is initialised in the desired output state $|\phi_x^{\text{out}}\rangle$ and another m qubits in the corresponding input state $|\phi_x^{\text{in}}\rangle$. The remaining qubits, if necessary, are initialised in $|0\rangle$ as operational qubits for a QNN with dissipative architecture. The second part features the network's evaluation of $|\phi_x^{\text{in}}\rangle$. In the last part, the fidelity between the network's output ρ_x^{out} and the desired output $|\phi_x^{\text{out}}\rangle$ is calculated using the destructive swap test circuit from 4.18. The full circuit is executed for all pairs x in the training set to calculate the training loss. At the end of the epoch t, the network parameters are updated using the gradient descent update step $\vec{\theta}_{t+1} = \vec{\theta}_t + d\vec{\theta}$.

iii. Validation: The black box QNN is evaluated using the state pairs from the validation set. These states are unknown to the QNN and uncorrelated to the training set. The resulting validation loss yields a measure for the QNN's generalisation capabilities.

What is the concrete implementation of a quantum neural network and the training algorithm as a quantum circuit?

Each quantum neural network requires the **implementation of its map as a set of quantum gates** that can be modified with the network parameters. Furthermore, it has to be defined on which qubits the network's input state is initialised and which qubits represent the output state of the network. In the case of supervised learning, the **destructive swap test** can be used to evaluate the **fidelity between the output state and the desired state** prepared on another set of qubits. In that way, the loss function and its gradient can be calculated and used for classically training the quantum neural network.

4.4 Results

After designing quantum circuits to implement the DQNN, the QAOA and a learning algorithm in the previous section, both QNNs have been tested numerically and compared in different areas with the results already published in the collaborative work [63]. A part of these results will be reviewed and discussed for the remainder of this chapter. The focus lies on a numerical study of the effects of quantum gate noise on the QNN performances and its execution on actual quantum computers hosted by IBM [8]. A presentation of the analysis of the generalisation capabilities will be dispensed in this work, but the interested reader is politely referred to the original publication [63]. All results have been obtained using the open source SDK Qiskit [82].

Setup

The general setup for the numerical studies is the task to learn an unknown unitary transformation V. In this case, one or several target unitaries of dimension $d = 4$ are chosen randomly. Thus, both networks have $m = 2$ input and output qubits, respectively. The DQNN is chosen to have no hidden layers ($L = 0$) which results in the architecture  . The QAOA is chosen to have the default number of $L = d^2/2 = 8$ alternating operator pairs as this guarantees the convergence to the optimal solution for learning a unitary transformation via gradient descent [78]. These shallow network architectures are well suited for analysing both network types as they implement the minimum non-trivial architectures for which successful training may be expected, and they already exhibit the effects of quantum entanglement. This enables a generalisation of the main results to more extensive networks. These networks are implemented using 6 (4) qubits for the DQNN (QAOA), including the initialisation and measurements as described in the previous Section 4.3. Training these networks requires the training of 24 (16) parameters, respectively. The DQNN network parameters are randomly initialised in the range $[0,2\pi)$, while the QAOA network parameters are randomly initialised in the range $[-1,1]$. Due to the small number of parameters and qubits, the barren plateau phenomenon is not expected to affect network performances strongly.

The specific choice for the hyper-parameters ϵ and η has to be adapted to both the network type and the dimension of the problem and can not be reliably estimated a priori. Therefore several previous test runs have been made to find well-performing hyper-parameters for both experiments, the gate noise analysis and the execution on NISQ devices. For the DQNN, the optimal hyper-parameters were found to be $\epsilon = 0.25, \eta = 0.5$ for the gate noise analysis, and $\epsilon = 0.5, \eta = 1.0$ for the NISQ device execution. For the QAOA, different hyper-parameters $\epsilon = 0.05, \eta = 0.05$ for the gate noise analysis, and $\epsilon = 0.15, \eta = 0.1$ for the NISQ device execution were found.

The implementation and execution of the corresponding quantum circuits have been carried out using the open-source SDK *Qiskit* [82]. Qiskit allows a decomposition of the parameterised quantum circuits discussed in Section 4.3 to a predefined set of elementary gates yielding a quantum circuit that the specific IBM quantum devices execute. While the required basis gates for the initialisation (6-12 single-qubit gates and 6 CNOT gates) and the fidelity measurement (2 single-qubit gates and 2 CNOT gates) is independent of the network type, the required qubits for implementing the core network circuit differs for the DQNN and the QAOA. For example, mapping the network's circuit to the H-topology of ibmq_casablanca [8] results in 57 (97) single-qubit gates and 63 (57) CNOT gates for the DQNN (QAOA).

The training of the QNNs has been executed in a hybrid manner. At each epoch, the loss from Equation (4.17) is evaluated by the repetitive execution of the respective quantum circuits, which was in turn used to update the parameters classically. To guarantee the training's success, it was necessary to use a training set that is large enough to determine the transformation of the target unitary V. For an $m = 2$ network ($V \in U(4)$), this is achieved using four training pairs [83] as has also been confirmed numerically with the generalisation analysis in [63].

Gate noise analysis

The performance of the DQNN and the QAOA is strongly affected by the noise entering through the quantum circuit execution on the actual quantum devices. The primary sources are readout noise, and quantum gate noise [84]. Both networks are equally influenced by readout noise as they are evaluated by the same measurement method. As the focus is on comparing both networks, the readout noise is neglected from here on. The influence of quantum gate noise, however, does differ because the QNNs consist of different gates.

To test the influence of quantum gate noise on both networks' performances, the DQNN and the QAOA are trained using a depolarising quantum error channel [3]. The depolarising channel $\mathcal{E}_{\mathrm{DC}}$ implements the depolarisation of some input state ρ with probability $\lambda \in [0,1]$ as:

$$\mathcal{E}_{\mathrm{DC}}(\rho) = \frac{\lambda I}{2} + (1-\lambda)\rho \tag{4.22}$$

with the completely mixed state, $I/2$. By assigning a depolarising probability λ to a quantum gate, the resulting state becomes the completely mixed state, $I/2$, with probability λ and the correct state according to the quantum gate with probability $1-\lambda$.

The depolarising channel in the gate noise analysis is parameterised with depolarisation probabilities $\lambda^g = k\lambda_0^g$ with basis gates $g = \mathrm{CNOT}, \mathrm{SX}, \mathrm{RZ}$ and scaling factor k. The parameter λ_0^g has been chosen to match the gate error

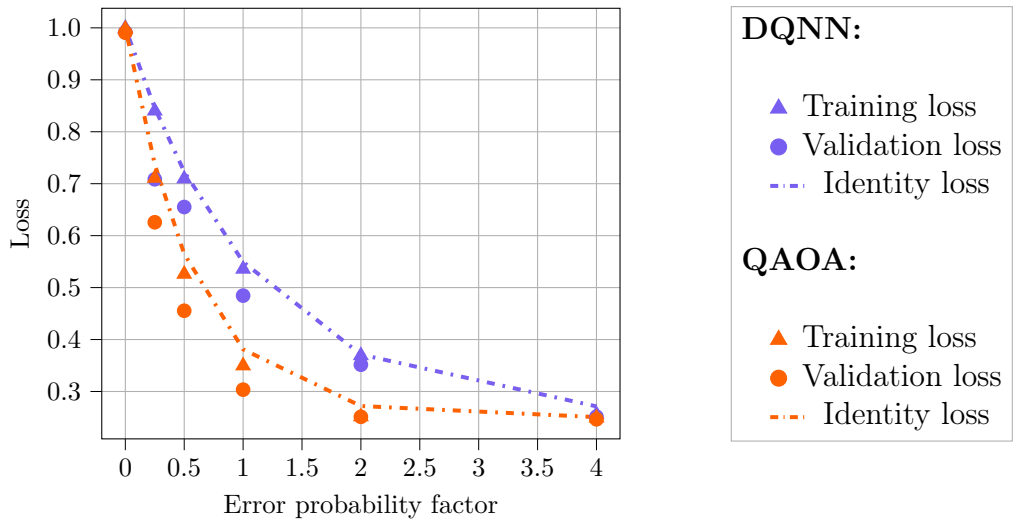


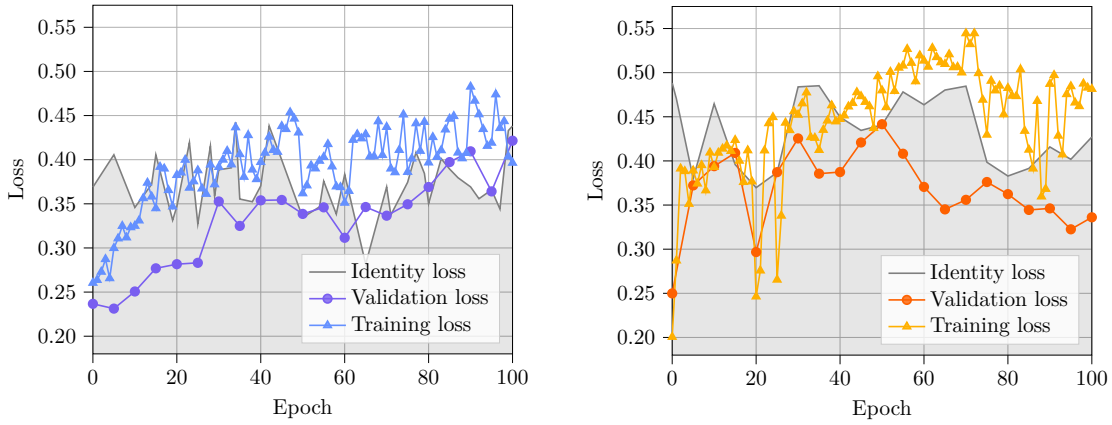
Figure 4.2: Gate noise analysis. The DQNN  ($L = 8$ QAOA) training is simulated for varying quantum gate noise via the depolarising quantum error channel using the qubit coupling map of *ibmq_16_melbourne* for $\epsilon = 0.25$, $\eta = 0.5$ ($\epsilon = 0.05$, $\eta = 0.05$). The training loss (▲) and the validation loss (●) are shown after training both networks with $N_T = 4$ training pairs until the validation loss converges. The identity loss (-·-·-) gives an estimate for the loss of an ideally trained network. Here, $k = 1$ approximately corresponds to the quantum gate noise of currently available NISQ devices. Especially in this region, the DQNN performs better. The quantities are averaged over 5 training sessions with different target unitaries V and start parameters $\vec{\theta}_0$.

probabilities of current NISQ devices. For the present analysis, the probabilities $\lambda_0^{\text{CNOT}} = 3.14 \times 10^{-2}$, $\lambda_0^{\text{SX}} = 1.18 \times 10^{-3}$, $\lambda_0^{\text{RZ}} = 0$ have been chosen as an approximation [85] for *ibmq_16_melbourne* [8] at the time of conducting the analysis. Additionally, the qubit coupling map of *ibmq_16_melbourne* has been used.

A relevant quantity beside the training and validation loss is the networks' identity loss. The identity loss is calculated using $V = 1$ and parameters such that the respective network acts as the identity. Each quantum gate is still applied and adds noise to the circuit. Thus, the identity cost gives an estimate for the best possible cost of an ideally trained network. In the noise-less case, the identity loss takes on the value 1.

Figure 4.2 shows the numerical results comparing the DQNN and the QAOA training for different error probability factors k. Note that $k = 1$ approximately corresponds to the quantum gate noise of currently available NISQ devices. For $k = 0$, i.e., a noise-free quantum device, both QNNs perform equally well. In the presence of gate noise, however, the DQNN has a higher identity cost, which in each case also yields a higher training and validation cost. This indicates that the DQNN is more suitable for learning unitaries on NISQ devices than the QAOA and will keep this advantage even for decreasing gate noise levels ($k < 1$).

Training on quantum computers



(a) The DQNN on *ibmq_ casablanca* for $\epsilon = 0.5$ and $\eta = 1.0$.

(b) The $L = 8$ QAOA on *ibmq_ casablanca* for $\epsilon = 0.15$ and $\eta = 0.1$.

Figure 4.3: Training on quantum computers. This figure shows the relevant quantities measured while training the DQNN (a) and the QAOA (b). Both networks have been trained for 100 epochs. Each epoch, the training loss (▲) is calculated via four 4 training pairs. Every fifth epoch, the validation loss (●) is measured using 4 validation pairs, as well as the identity loss (▬) using 4 output training states.

Although the effects of quantum noise have been considered, it is not sufficient to show that QNNs can be trained with simulated quantum circuits. Instead, as actual proof of concept and to benefit from their advantages, they have to be executed on actual NISQ devices, too. To demonstrate both the challenges and the potential of training the DQNN and the QAOA on today's quantum computers, no averages but the results from single training sessions are presented, respectively. That way, the effects of all present noise sources on the training performance can directly be observed. Both QNNs are trained using the IBM 7-qubit device *ibmq_ casablanca* [8]. The relevant loss functions during training the DQNN and the QAOA to learn the same unknown unitary $V \in U(d)$ are shown in Figure 4.3.

In the noise-free case, the training loss should always be monotone increasing for well-chosen hyperparameters, i.e., a small enough ϵ so that the error in the gradient estimation from Equation (4.20) can be neglected and a small enough η so that the parameter update step does not overshoot the global maximum. However, the variance in the training loss indicates the networks' response to gate and measurement noise.

Interestingly, the training and validation loss seem to be correlated to the varying identity loss. This again suggests that the variance of the identity loss is not only a statistical error but arises from a noise drift of the quantum computer itself. What is more, in the previous gate noise analysis, the identity loss reliably served as the upper limit for the training and validation loss. This observation is mainly explained by the definition of identity loss as the expected loss for a QNN

that perfectly mimics the target unitary V. The observation holds for the noisy case in the previous analysis, at least when encountering purely probabilistic gate noise. However, here the training loss exceeds the identity loss after a few epochs for both QNNs. This shows the networks' surprising ability to factor in the noise circumstances of the actual quantum computer, and it indicates that the device noise is partly structural.

Having said this, the high validation to identity loss ratio demonstrates the remarkable capability of both networks to generalise the provided information despite the high noise levels. Conclusively, it is justified to claim that the DQNN learns an unknown unitary slightly better than the QAOA, as quantified by the validation loss.

Chapter 5

Quantum Generative Adversarial Networks

Finally, with a working implementation of a quantum neural network at hand, it is possible to apply the concept of generative adversarial learning to the quantum case. This chapter translates the ideas from classical to quantum generative adversarial networks by answering the following questions: *What are the basic concepts of generative adversarial learning? How can quantum generative adversarial networks be defined?*

To lay a foundation, Section 5.1 discusses the general notions of generative adversarial learning for the classical case. The focus is on conveying an understanding of how training two networks simultaneously can lead to desired features for a generative model in an unsupervised learning setting. As a mathematical description, an objective function describing a two-player minimax game is introduced. In particular, the desired final state of the two-player minimax game is discussed and under which conditions it can be reached. The definition of another objective function follows this to cope with typical challenges. In Section 5.2, as a transition to the quantum case, some proposals for quantum generative adversarial networks are discussed, and the key ingredients for a meaningful implementation are highlighted. This is followed by a general discussion of a quantum objective function's structure and possible training settings. Essential is the definition of a generative adversarial learning algorithm for quantum neural networks. This is followed by two concrete proposals for quantum objective functions, one of which is based on a fidelity measurement while the other implements the Hilbert-Schmidt distance. In Section 5.3, the obtained numerical results are presented. After presenting single executions for supervised and unsupervised training of quantum generative adversarial networks, a numerical investigation of different network architectures is shown. The best performing networks are used to compare the training performance under different objective functions and training sets.

5.1 General notions of generative adversarial learning

Generative adversarial networks (GANs) resemble an advanced application of neural networks to learn the internal features of a given data distribution and produce similar data that favourably extends the initial distribution. On the contrary to other generative models, GANs do not require Markov chains or interference during training [31]. Additionally, GANs can represent sharp distributions and there are statistical advantages due to not training the generative model directly via data samples [31]. Since the first proposal in 2014 [31], GANs have successfully proven their vast capabilities through implementations on a variety of tasks such as image generation, image super-resolution, classification, and many more [86].

Traditional GANs belong to the class of unsupervised machine learning as they do not need access to labelled data. Instead, their training is based on a two-player minimax game, the two players being a generator network and a discriminator network adversarially and alternately trained. The discriminator network's goal is to classify its inputs correctly as real (from the training set) or as fake (from the the generator network). On the contrary, the generator network's goal is to produce data instances that fool the discriminator network into classifying them wrongly, i.e., as real. By alternately training the generator and the discriminator network, the generator eventually can produce a data distribution that leads to a Nash equilibrium [87], i.e., the discriminator cannot distinguish real and fake data samples anymore [31].

The basic process of GAN training can be explained playfully through the picture of a fraudster (generator) who produces counterfeit coin money and a government (discriminator) that tries to distinguish the counterfeit from the real money. In the beginning, the government might have a relatively easy job in detecting the counterfeit money. However, during the years, the fraudster learns to produce better fakes based on the government's evaluations of the counterfeit and the real money. After many feedback loops and iterations of the fraudster improving the counterfeit technique and the government improving their classification methods, the fraudster achieves to produce fake coins that can not be distinguished from the real money anymore. Both players adversarially improving their strategies leads to the generating player being able to resemble the initial data distribution closely.

In particular, the GAN training is described via a parameterised generator $G(\boldsymbol{z},\vec{\theta}_G)$ and a parameterised discriminator $D(\boldsymbol{x},\vec{\theta}_D)$ playing a two-player minimax game with respect to an objective function $\mathcal{L}(G,D)$ [31]:

$$\min_G \max_D \mathcal{L}(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\boldsymbol{z})))] \quad (5.1)$$

where $\boldsymbol{x}$ is a data sample according to the real data distribution $p_{\text{data}}(\boldsymbol{x})$, $\boldsymbol{z}$ is a sample from the latent space according to $p_{\boldsymbol{z}}(\boldsymbol{z})$, and $\mathbb{E}$ means the expectation value corresponding to the respective data distribution. For example, the data samples $\boldsymbol{x}$

could be images of paintings where the data distribution $p_{\text{data}}(\boldsymbol{x})$ provides a realistic composition of the art created in the 19th century. Accordingly, the latent sample $\boldsymbol{z}$ could be some random real-numbered vector whose components the generator could associate with different drawing styles and color schemes. Thus, the latent space can be interpreted as a compressed space containing different features of the data samples (paintings) where similar latent samples correspond to similar decompressed data samples.

The discriminator $D(\boldsymbol{y}) \in [0,1]$ attempts to maximise the objective function $\mathcal{L}(G,D)$. This is achieved if the discriminator assigns high values to samples from the real data distribution and small values to generated samples $G(\boldsymbol{z})$ as this maximises the first and the second term in Equation (5.1), respectively. Consequently, $D(\boldsymbol{y})$ can be interpreted as the discriminator's estimated probability for $\boldsymbol{y}$ being a real sample. In contrary, the generator G tries to minimise the objective function $\mathcal{L}(G,D)$. For this the generator aims to produce a distribution p_G via $G(\boldsymbol{z})_{\boldsymbol{z}\sim p_z(\mathbf{z})}$ that the discriminator wrongly classifies as real with the goal of minimising the second term in Equation (5.1). The traditional generative adversarial learning algorithm proposed by [31] is shown as Algorithm 1.

The two-player minimax game described by Equation (5.1) has a global optimum at $p_G = p_{\text{data}}$, i.e., it reaches the Nash equilibrium if the generator exactly produces the desired data distribution. A proof of this statement is given in [31] and will be sketched in the following. The desired optimum for the generated distribution p_G is given in the case of an optimal discriminator D. Analysing the properties of the objective function $\mathcal{L}(G,D)$ yields the optimal discriminator $D^*_G(\boldsymbol{x})$ for a fixed generator G:

$$D^*_G(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}. \tag{5.2}$$

Inserting the optimal discriminator $D^*_G(\boldsymbol{x})$ into Equation (5.1) yields the following virtual objective function that the generator aims to minimise:

$$\min_G \mathcal{L}(G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_G}\left[\log \frac{p_G(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right]. \tag{5.3}$$

Assuming $p_G = p_{\text{data}}$, the objective function becomes $\mathcal{L}(G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{1}{2}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_G}\left[\log \frac{1}{2}\right] = -\log 4$. By adding $\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log 2] + \mathbb{E}_{\boldsymbol{x}\sim p_G}[\log 2] - \log 4 = 0$ to the right hand side of Equation (5.3), the virtual objective function can be written in terms of the Jensen-Shannon divergence:

$$\min_G \mathcal{L}(G) = -\log 4 + JSD(p_{\text{data}} \| p_G). \tag{5.4}$$

The Jensen-Shannon divergence between two probability distributions is zero if and only if the two distributions are equal and is always non-negative otherwise [88].

This concludes that $\mathcal{L}(G,D^*_G) = -\log 4$ is the global minimum of Equation (5.3) and that it is unique for $p_G = p_{\text{data}}$.

Alternately training the generator G and the discriminator D converges to the global optimum, i.e., the generator produces the distribution $p_G = p_{data}$ under some further conditions discussed below. G and D must have enough capacity to produce the distribution p_{data} and to represent a sufficient discriminator, respectively. During training, the discriminator must be allowed to reach its optimum D^*_G for any given generator G. The generator is updated to minimise Equation (5.1) with $D = D^*_G$ via gradient descent using sufficiently small update steps, which finally guarantees the convergence of G such that $p_G = p_{\text{data}}$.

In practice, there are no convergence guarantees for the widely adopted GAN training algorithm from [31] which is based on alternately training the generator and the discriminator. G and D do not have unlimited capacity but are given by parameterised neural networks $G(\boldsymbol{z},\vec{\theta}_G)$ and $D(\boldsymbol{x},\vec{\theta}_D)$ which can not necessarily produce arbitrary distributions p_{data} or represent the optimal discriminator D^*_G. However, networks such as the multilayer perceptron can be expressible enough to be successfully trained by Algorithm 1. The discriminator training period T_D as one of the free parameters has to be chosen such that the discriminator training in line 6 yields a good approximation of the optimal discriminator D^*_G. If this is fulfilled, the discriminator D yields a useful gradient for the generator training in line 9. By repeating the alternate training of G and D N times, the generator can often produce a distribution $p_G \approx p_{\text{data}}$.

One of the usual difficulties during training GANs is mode collapse, which often is the cause of a not sufficiently trained discriminator [32]. Mode collapse describes the failure mode that the generator network only produces a small fraction of the target distribution, i.e., the generator maps many different latent space samples to the same data sample. In principle, this behaviour is expected as the generator's best option to fool a non-optimal discriminator is to mimic a delta function at the positions in the target space where the discriminator assigns the highest values. The problem becomes permanent if the discriminator cannot detect these preferred modes or can only remove these by provoking new modes. A solution to prevent mode collapse would be to train the discriminator to the optimum such that the gradient determining the generator training yields fewer preferred modes. However, training the discriminator to the optimum when using Equation (5.1) can lead to vanishing gradients for the generator training, thus being unpractical.

Some ansatzes address the problem of mode collapse using objective functions or algorithms different from Equation (5.1) or Algorithm 1, respectively. The main idea of *Wasserstein* GANs [32] is to use an objective function that in the case of a fully trained discriminator does not undergo vanishing gradients. In particular, for training the networks, the following objective function as an estimate of the

Algorithm 1: Minibatch stochastic gradient descent training of generative adversarial networks from [31]. Please note that gradient descent is used for the parameter optimisation, however, different gradient-based learning rules can be applied.

Data: Training data distribution $p_{\text{data}}(\boldsymbol{x})$, latent space distribution $p_z(\boldsymbol{z})$, generator network $G(\boldsymbol{z},\vec{\theta}_G)$, discriminator network $D(\boldsymbol{x},\vec{\theta}_D)$, discriminator training period T_D, learning rates $\eta_{G,D}$, batch size m, and number of training epochs N.

Initialise $\vec{\theta_G}$ and $\vec{\theta_D}$ randomly.
for $n \leftarrow 0$ **to** N **do**
  // Discriminator training
  for $i \leftarrow 0$ **to** T_D **do**
    Sample minibatch $\{\boldsymbol{x}^k\}_{k=1}^m$ from data distribution $p_{\text{data}}(\boldsymbol{x})$.
    Sample minibatch $\{\boldsymbol{z}^k\}_{k=1}^m$ from latent distribution $p_z(\boldsymbol{z})$.
    Update the discriminator by ascending its stochastic gradient:

$$\vec{g}_D \leftarrow \nabla_{\vec{\theta}_D} \frac{1}{m} \sum_{k=1}^{m} \log D(\boldsymbol{x}^k) + \log\left(1 - D(G(\boldsymbol{z}^k))\right)$$
$$\vec{\theta}_D \leftarrow \vec{\theta}_D + \eta_D \vec{g}_D$$

  end
  // Generator training
  Sample minibatch $\{\boldsymbol{z}^k\}_{k=1}^m$ from latent distribution $p_z(\boldsymbol{z})$.
  Update the generator by descending its stochastic gradient:

$$g_G \leftarrow \nabla_{\vec{\theta}_G} \frac{1}{m} \sum_{k=1}^{m} \log\left(1 - D(G(\boldsymbol{z}^k))\right)$$
$$\vec{\theta}_G \leftarrow \vec{\theta}_G - \eta_G \vec{g}_G$$

end
Use $G(\boldsymbol{z},\vec{\theta}_G)$ to generate new data samples from latent space $p_z(\boldsymbol{z})$.

Wasserstein distance [89] is used:

$$\min_G \max_C \mathcal{L}(G,C) = \frac{1}{m}\sum_{k=1}^{m}\left(C(\boldsymbol{x}^k) - C(G(\boldsymbol{z}^k))\right) \tag{5.5}$$

with a parameterised generator network $G(\boldsymbol{z},\theta_G)$ and critic network $C(\boldsymbol{x},\theta_C)$. Here the critic fulfils a slightly different task than the previously discussed discriminator, i.e., the critic $C(\boldsymbol{y}) \in \mathbb{R}$ does not assign probabilities to the input $\boldsymbol{y}$ being from the training set or the generator but instead tries to increase the difference between its output for real and fake samples. The algorithm of Wasserstein GANs intends the critic to be trained to the optimum such that the generator, in turn, is provided with the best possible gradient towards $p_G = p_{\text{data}}$. Another method to prevent mode collapse is given by *Unrolled* GANs [90]. The main idea is to use a special application of backpropagation that foresees the effects of specific generator updates to possible future discriminator versions and considers these when training the generator. This method exploits the fact that a well-trained discriminator can easily learn to recognise mode collapsed generator samples and reject them. Using this rejection information already during the generator training, mode collapse can be prevented from the beginning.

What are the basic concepts of generative adversarial learning?

The general setting features **two adversarial networks**. The **discriminator** network's task is to distinguish fake samples from real training samples, while the **generator** network's task is to fool the discriminator by producing fake samples similar to the training samples. This behaviour is controlled by an objective function that defines a **two-player minimax game**. Both networks are **trained alternately** to optimise for their respective task. This unsupervised learning algorithm can train a generator network to extend a given training set.

5.2 Quantum generative adversarial networks

Quantum generative adversarial networks (QGANs), in analogy to classical GANs, as their predecessor, represent an advanced application of QNNs to learn the internal features of a given class of quantum states and train a generative model to produce new quantum states within this class. The aim of QGANs is to adopt the advantages of classical GANs while preserving the advantages of QNNs as their basic components. The desired advantages of their predecessor include the possibility to learn from unlabelled data, the non-necessity to know any internal features of the training class, and the high diversity in generated states due to

not training the generative model directly via the class' samples. However, during designing QGANs and a corresponding training algorithm, it is very likely that not only the advantages but some challenges of classical GANs will be adopted as well. Therefore a broad study of possible QGAN designs is necessary to harness the full potential of GANs' general ideas.

Literature review

There are some concrete designs for QGANs in the literature, two of which will be reviewed here briefly, especially featuring different objective functions for training. In [91], some general notions of quantum adversarial learning are introduced, including the case of learning quantum data with fully quantum generator and discriminator networks. This work shows that a quantum discriminator can always discriminate training samples from generated samples with a probability larger than $\frac{1}{2}$ if the generator produces statistics different from the training data, and in turn, the quantum generator can learn to match the training data statistics such that the unique equilibrium is reached where the quantum discriminator can no longer differentiate between the training data and the generated data. To take these theoretical results into practice, the companion paper [92] proposes a concrete QGAN design including quantum circuits for both networks and an objective function $\mathcal{L}(G, D)$ for the generator and discriminator training:

$$\min_G \max_D \mathcal{L}\left(\vec{\theta}_G,\vec{\theta}_D\right) = \frac{1}{2} + \frac{1}{4\Lambda}\sum_{\lambda=1}^{\Lambda} \mathrm{Tr}\left(\left[\rho_\lambda^T(\vec{\theta}_G) - \rho_\lambda^G(\vec{\theta}_G,\vec{\theta}_D,|z\rangle)\right] Z\right) \tag{5.6}$$

with networks parameters $\vec{\theta}_G,\vec{\theta}_D$, the operator $Z \equiv |\text{real}\rangle\langle\text{real}| - |\text{fake}\rangle\langle\text{fake}|$, random latent states $|z\rangle$, and discriminator output states ρ_λ^T and ρ_λ^G given Λ samples from the training data and the generator, respectively. Both networks can additionally be granted access to exact copies of a label state $|l\rangle$ that can be used to learn states from differently labelled classes. In [93], the *Entangling* QGAN is proposed, including the definition of an alternative objective function for training and a specific discriminator network. It claims to solve some problems occurring in [92], namely, non-convergence due to the oscillation between a finite set of states and the convergence to a non-unique Nash equilibrium. The general idea is to provide the discriminator access to a training state and a generated state simultaneously and thus, enable the discriminator to entangle these states. For training the networks, the following objective function $\mathcal{L}(G, D)$ is used:

$$\min_G \max_D \mathcal{L}\left(\vec{\theta}_G,\vec{\theta}_D\right) = 1 - D_\sigma\left(\vec{\theta}_D, \rho(\vec{\theta}_G)\right) \tag{5.7}$$

with networks parameters $\vec{\theta}_G,\vec{\theta}_D$, the in general mixed training state σ, the generated state $\rho(\vec{\theta}_G)$, and the discriminator output $D_\sigma\left(\vec{\theta}_D, \rho(\vec{\theta}_G)\right)$ which is the result of a parameterised swap test between σ and $\rho(\vec{\theta}_G)$.

The previous QGAN proposals and, in particular, their applications do not yet grasp the broader idea of generative adversarial learning, which is to not only replicate but extend the class of training states. By alternately training the generator and the discriminator circuits with respect to Equation (5.6), [92] succeeded in training their generator circuit to produce the labelled states $\rho_A = |0\rangle\langle 0|$ and $\rho_B = |1\rangle\langle 1|$. They claim not to need any random latent state $|z\rangle$ to train the generator successfully. However, this abandons the fundamental idea of generative adversarial learning to not only replicate the training data but to produce new samples within the desired class - which is implicitly given through multiple training states. Waiving the generator's ability to produce different states based on random input states $|z\rangle$ means that their QGAN training becomes equivalent to a more basic supervised training procedure with access to two pairs of input and output states. As opposed to Equation (5.6) which at least offers the possibility to use random input states for the generator, the objective function given by Equation (5.7) from [93] does not even include the notion of random input states or the possibility to train the QGAN with multiple training states. The absence of random input states for the generator can not be viewed as a feature but instead means a fundamental cut into the possibilities that quantum generative adversarial learning may have to offer.

Definition

General role of the objective function

As previously discussed for the current literature on QGANs, the definition of the objective function is crucial in that it must provide the required tools for a useful application of generative adversarial learning to QNNs. In particular, the objective function has to incorporate access to a set of training states which implicitly define the desired class of quantum states and to a pool of random latent states. Here, a latent state is a pure quantum state from the latent space and enables the generator QNN to randomly produce different outputs using a random latent state as an input.

Before proposing a specific objective function for the realisation of QGANs, a placeholder objective function will be given here. The placeholder serves as an intermediate definition that clarifies the required quantities. It will be used to explain possible training sets, the corresponding QNN architectures and the quantum generative adversarial learning algorithm before, finally, giving two exact definitions for objective functions. The placeholder objective function $\mathcal{L}$ is given as:

$$\min_G \max_D \mathcal{L}\left(\mathcal{E}_G, \mathcal{E}_D, T, Z\right) \tag{5.8}$$

with the following variable parameters: a parameterised generator QNN $\mathcal{E}_G$, a parameterised discriminator QNN $\mathcal{E}_D$, a training set T, and a set of random latent

states Z. The generator aims to minimise $\mathcal{L}$ by a modification of its parameters $\vec{\theta}_G$ which consequently changes the map $\mathcal{E}_G$. On the contrary, the discriminator aims to maximise $\mathcal{L}$ by modifying its parameters $\vec{\theta}_D$ and thereby changing the map $\mathcal{E}_D$.

The basic idea of the objective function $\mathcal{L}$ is the same as for the classical case, which has been thoroughly discussed in Section 5.1 and will be shortly repeated using the terms of the quantum case. The objective function describes a two-player minimax game between a parameterised generator QNN $\mathcal{E}_G$ and a parameterised discriminator QNN $\mathcal{E}_D$. The objective function evaluates how well the discriminator distinguishes fake states produced by the generator from real states taken from the training set T. The better the discriminator performs, the larger is $\mathcal{L}$. Therefore, during training, the discriminator aims to maximise the objective function $\mathcal{L}$. In turn, the generator tries to produce fake states that the discriminator can not distinguish from the real states. The better the generator performs, the smaller is $\mathcal{L}$. Thus, the generator tries to minimise the objective function $\mathcal{L}$ during training.

There are two critical requirements for designing the objective function for a QGAN: it provides useful optimisation measures for both networks, and it is efficiently computable on a quantum device. Concerning the first requirement, it is helpful to learn from classical GANs such as the first successful GAN implementation (see Equation (5.1)) and the improved Wasserstein GAN (see Equation (5.5)). Regarding the second requirement, it is necessary to test different objective functions and optimise for an efficient computation while ensuring the training success. At the end of this section, two different objective functions $\mathcal{L}^F$ and $\mathcal{L}^{HS}$ will be proposed with the two mentioned classical objective functions as role models.

In principle, the generator QNN and the discriminator QNN can be optimised using the same objective function $\mathcal{L}$ as given by Equation (5.8). However, it turns out that it is more efficient to use slightly different objective functions $\mathcal{L}_G$ and $\mathcal{L}_D$ for training the generator QNN and the discriminator QNN, respectively.

Training set and network architecture

This work's proposal for generative adversarial learning with QNNs provides a general framework for learning different types of training sets. In particular, labelled as well as unlabelled training sets can be used. Within the scope of this work, a general training set $\mathcal{T} = \{|\phi_k^T\rangle, |l_k\rangle\}_{k=1}^{N_T}$ consists of N_T pure training states $|\phi_k^T\rangle$ and corresponding pure label states $|l_k\rangle$. In principle, mixed training and label states could be used as well.

The dimensions of the training set determine some boundary conditions for the used generator and discriminator QNNs and will therefore be laid out in the following. First, it is necessary to establish how many qubits are needed to represent the training and label states, respectively. For d_T-dimensional training states, $n_T = \log_2(d_T)$ qubits are required to represent one training state. Similarly, if the training set contains d_{label}-dimensional label states, an additional

$n_{\text{label}} = \log_2(d_{\text{label}})$ are required to represent a label state. Here, it is assumed without loss of generality that d_T is a power of 2 with a natural exponent. If the training set is unlabelled, no label qubits are needed ($n_{\text{label}} = 0$). Additionally, it is possible to use $n_{\text{latent}} \in \mathbb{N}^+$ qubits to represent random latent states of dimension $d_{\text{latent}} = 2^{n_{\text{latent}}}$. Here, $n_{\text{latent}} = 0$ corresponds to a QGAN without access to random latent states.

If and in which way the QGAN is granted access to a latent space may depend on the training set. If the training set is unlabelled, the generator QNN must access random latent states as inputs to be able to generate more than one state. In the case of a labelled training set, it may be useful to deny access to random latent states ($n_{\text{latent}} = 0$) and only use the label states as inputs to the generator. This results in exactly one generator output to each label state. This setting will be exploited as a proof-of-concept in Section 5.3 for once. However, as discussed previously, this does not resemble a profound use of generative adversarial learning.

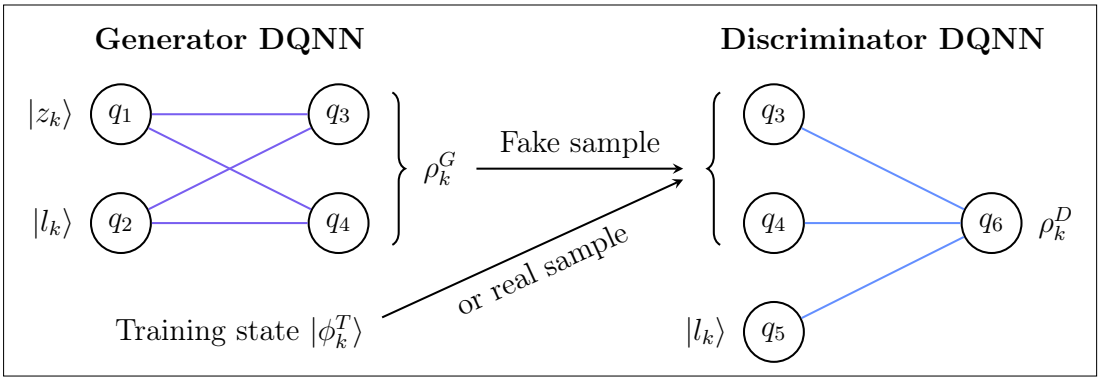


Figure 5.1: QGAN architecture. This work's QGAN features a generator DQNN and a discriminator DQNN. In the general case, the generator DQNN takes as input a pure random latent state vector $|z_k\rangle$ and a pure label state $|l_k\rangle$. The generated state ρ_k^G is given as the generator DQNN output qubits' state. The discriminator DQNN takes as input a generated state (fake sample) or a training state (real sample) and the previously mentioned pure label state $|l_k\rangle$. If the discriminator needs a fake sample as an input it does so by using the output qubits of the generator as input qubits. The discriminator outputs some, in general mixed state ρ_k^D which is used to evaluate the QGAN objective function. Here, a minimal example of a generator DQNN and a discriminator DQNN without any hidden layers for a labelled training set with $n_T = 2$, $n_{\text{label}} = 1$, and $n_{\text{latent}} = 1$ is presented.

The generator QNN and the discriminator QNN can both be implemented straightforwardly by a DQNN as defined in Sections 4.2 and 4.3. In what follows, when talking about the generator, a DQNN with map $\mathcal{E}_G$ parameterised by $\vec{\theta}_G$ is meant. Analogously, the discriminator is a DQNN with map $\mathcal{E}_D$ parameterised by $\vec{\theta}_D$. Both networks' input and output layers depend on some conditions clarified below, while the use and structure of any hidden layers can be chosen arbitrarily. The generator takes as input a random latent state and a label state which requires $n_{\text{latent}} + n_{\text{label}}$ input qubits. The generator's output corresponds to the dimension of the training states, thus, the generator features n_T output neurons.

The discriminator takes as input a real or fake training state and a label state, and therefore requires $n_T + n_{\text{label}}$ input qubits. The number of output qubits for the discriminator is generally a free parameter. However, it is highly dependent on the objective function and is usually set to 1. A minimal example of the generator and the discriminator without any hidden layers for a labelled training set with $n_T = 2$, $n_{\text{label}} = 1$, and $n_{\text{latent}} = 1$ is shown in Figure 5.1.

Generative adversarial learning algorithm

The algorithm for generative adversarial learning with QNNs is very similar to the originally proposed algorithm for classical neural networks shown in Algorithm 1. The main difference lies in the objective function used for training the generator and the discriminator, respectively. Before discussing two concrete proposals for objective functions for the quantum case, the general algorithm will be presented using the placeholder objective function from Equation (5.8). As a start of the iterative training process, the generator $\mathcal{E}_G$ and the discriminator $\mathcal{E}_D$ are initialised randomly with parameters $\vec{\theta}_G$ and $\vec{\theta}_D$, respectively. Throughout the algorithm, the network maps $\mathcal{E}_G, \mathcal{E}_D$ are always implicitly defined by the mentioned parameters.

Algorithm 2: Generative adversarial learning algorithm for DQNNs. If the objective functions $\mathcal{L}_G^F$,$\mathcal{L}_D^F$ are used, this algorithm is referred to as F-QGAN. If the objective functions $\mathcal{L}_G^{HS}$,$\mathcal{L}_D^{HS}$ are used, this algorithm is referred to as HS-QGAN.

Data: Learning rates $\eta_{G,D}$, training set $\mathcal{T}$, objective functions $\mathcal{L}_G, \mathcal{L}_D$, network maps $\mathcal{E}_{G,D}\left(\theta_{G,D}\right)$, training period T_D, batch size m, and number of epochs N.

Initialise $\vec{\theta}_G$ and $\vec{\theta}_D$ randomly.
for $n \leftarrow 0$ **to** N **do**
  Sample training batch T of size m from all training states $\mathcal{T}$.
  Randomly sample latent batch $Z = \{|z_k\rangle\}_{k=1}^m$.
  // Discriminator training
  for $i \leftarrow 0$ **to** T_D **do**
    $\vec{g}_D \leftarrow \nabla_{\vec{\theta}_D} \mathcal{L}_D\left(\mathcal{E}_G, \mathcal{E}_D, T, Z\right)$
    $\vec{\theta}_D \leftarrow \vec{\theta}_D + \eta_D \cdot \vec{g}_D$
  end
  // Generator training
  $\vec{g}_G \leftarrow \nabla_{\vec{\theta}_G} \mathcal{L}_G\left(\mathcal{E}_G, \mathcal{E}_D, T, Z\right)$
  $\vec{\theta}_G \leftarrow \vec{\theta}_G - \eta_G \cdot \vec{g}_G$
end
Use $\mathcal{E}_G$ with parameters $\vec{\theta}_G$ to sample new states from latent space.

As shown in Algorithm 2, the main structure of the proposed algorithm is to

alternately train the generator and the discriminator for a previously fixed or in real-time adjusted number of training epochs N. The following steps are repeated for each epoch. First, a training batch T of size m is sampled from the full training set $\mathcal{T}$, possibly including label states. Additionally, a latent batch Z of the same size is randomly generated from the latent space if $n_{\text{latent}} \neq 0$. In what follows, the discriminator is trained for T_D times by evaluating the discriminator's objective function. For this purpose, the generator gets the following inputs if available in each case: the k^{th} latent state from Z and the k^{th} label state from T. By execution of the corresponding map $\mathcal{E}_G$, a fake sample ρ_k^G is generated. By using the fake sample ρ_k^G as an input for the discriminator on the one hand and a real sample $|\phi_k^T\rangle$ from T as an input on the other hand, and repeating this for all elements of T and Z, the discriminator's objective function $\mathcal{L}_D$ can be evaluated. Generally speaking, $\mathcal{L}_D$ is larger if the discriminator distinguishes the real and fake samples better. By evaluating the gradient $\nabla_{\vec{\theta}_D}\mathcal{L}_D$, the discriminator parameters $\vec{\theta}_D$ can be updated via gradient descent such that the discriminator is optimised. After the discriminator training, a similar procedure is used to train the generator. The main differences are that now the generator's objective function $\mathcal{L}_G$ is used and is minimised instead of maximised. Furthermore, the generator is only trained once per epoch. In the optimal case, after N training epochs, i.e., $N \times (T_D + 1)$ training steps, the generator becomes so good that the discriminator can no longer distinguish the generated fake states from the real training states. If this is the case, the generator can sample new states from the latent space that extend the training set.

Objective function via fidelity

The first proposal for a QGAN objective function is inspired by Equation (5.1) and uses the fidelity as a measure for closeness between two quantum states, here, a pure state $|\phi\rangle$ and an, in general, mixed state ρ as $\mathcal{F}(|\phi\rangle, \rho) = \langle\phi|\rho|\phi\rangle$. It is defined as:

$$\begin{aligned}\min_G \max_D \mathcal{L}^F(\mathcal{E}_G, \mathcal{E}_D, T, Z) = \frac{1}{m}\sum_{k=1}^{m} \langle 1|\mathcal{E}_D\left(|\phi_k^T\rangle\langle\phi_k^T| \otimes |l_k\rangle\langle l_k|\right)|1\rangle \\ + \frac{1}{m}\sum_{k=1}^{m} \langle 0|\mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)|0\rangle\end{aligned} \tag{5.9}$$

with a generator $\mathcal{E}_G$, a discriminator $\mathcal{E}_D$, the generator output states $\rho_k^G = \mathcal{E}_G(|z_k\rangle\langle z_k| \otimes |l_k\rangle\langle l_k|)$, a set of possibly labelled training states $T = \{(|\phi_k^T\rangle, |l_k\rangle)\}_{k=1}^m$, and a set of random latent states $Z = \{|z_k\rangle\}_{k=1}^m$. Analogous to the classical case the discriminator is optimised to maximise the objective function. In this case the discriminator $\mathcal{E}_D$ aims to output the basis state $|1\rangle$ when fed with the training states $|\phi_k^T\rangle$ and the basis state $|0\rangle$ when fed with the generated

states ρ_k^G. The generator is optimised to minimise the objective function by learning to generate output states ρ_k^G which yield discriminator output states near $|1\rangle$. If the training set T is not labelled, the tensor products $\otimes |l_k\rangle \langle l_k|$ can be ignored throughout the objective function.

Here, the so-called *fidelity objective function* as given in Equation (5.9) can be split into two, one for the discriminator to maximise and one for the generator to minimise. This is more efficient because it is unnecessary to compute the full objective function each time. The discriminator's objective function is equivalent to Equation (5.9):

$$\begin{aligned} \max_D \mathcal{L}_D^F(\mathcal{E}_G,\mathcal{E}_D,T,Z) = \frac{1}{m}\sum_{k=1}^{m} \langle 1| \mathcal{E}_D\left(|\phi_k^T\rangle\langle\phi_k^T| \otimes |l_k\rangle\langle l_k|\right)|1\rangle \\ + \frac{1}{m}\sum_{k=1}^{m} \langle 0| \mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)|0\rangle \end{aligned} \tag{5.10}$$

and evaluates the discriminator's output for both, the training data and the generated data. On the contrary the generator's objective function only evaluates the discriminator's output for the generated data:

$$\min_G \mathcal{L}_G^F(\mathcal{E}_G,\mathcal{E}_D,T,Z) = \frac{1}{m}\sum_{k=1}^{m} \langle 0| \mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)|0\rangle \tag{5.11}$$

.

Both objective functions can be easily evaluated on a quantum device as Equations (5.10) and (5.11) only need $2m$ and m circuits with one measurement of the discriminator's output qubit in the computational basis $\{|0\rangle, |1\rangle\}$, respectively. A QGAN using the fidelity objective functions will be referred to as F-QGAN.

Objective function via Hilbert-Schmidt distance

The second QGAN objective function is inspired by Equation (5.5) and uses the Hilbert-Schmidt distance as a measure for closeness between two mixed states ρ and σ as $d_{\text{HS}}(\rho,\sigma) = \text{Tr}\left((\rho-\sigma)^2\right)$. It is defined as:

$$\begin{aligned} \min_G \max_D \mathcal{L}^{HS}(\mathcal{E}_G,\mathcal{E}_D,T,Z) &= \frac{1}{m}\sum_{k=1}^{m} d_{\text{HS}}\left(\mathcal{E}_D\left(|\phi_k^T\rangle\langle\phi_k^T| \otimes |l_k\rangle\langle l_k|\right), \mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)\right) \\ &= \frac{1}{m}\sum_{k=1}^{m} \text{Tr}\left(\left(\mathcal{E}_D\left(|\phi_k^T\rangle\langle\phi_k^T| \otimes |l_k\rangle\langle l_k|\right) - \mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)\right)^2\right) \end{aligned} \tag{5.12}$$

with a generator $\mathcal{E}_G$, a discriminator $\mathcal{E}_D$, the generator output states $\rho_k^G = \mathcal{E}_G(|z_k\rangle\langle z_k| \otimes |l_k\rangle\langle l_k|)$, a set of possibly labelled training states $T =$

$\{\left(|\phi_k^T\rangle, |l_k\rangle\right)\}_{k=1}^m$, and a set of random latent states $Z = \{|z_k\rangle\}_{k=1}^m$. Analogous to the classical case the discriminator aims to maximise the objective function by increasing the Hilbert-Schmidt distance between its output for fake generated states and real training states. In turn, the generator aims to minimise the objective function by decreasing the mentioned Hilbert-Schmidt distance.

As for the previously discussed objective function, the so-called *Hilbert-Schmidt objective function* as given in Equation (5.12) can be split into two, as again, the generator's optimisation does not require the computation of all terms. The discriminator's objective function is equivalent to Equation (5.12) and can be rewritten for the evaluation on a quantum device using three terms:

$$\begin{aligned}\max_D \mathcal{L}_D^{HS}(\mathcal{E}_G,\mathcal{E}_D,T,Z) = \frac{1}{m}\sum_{k=1}^m \mathrm{Tr}\left(\mathcal{E}_D\left(|\phi_k^T\rangle\langle\phi_k^T| \otimes |l_k\rangle\langle l_k|\right)^2\right) \\ - \frac{2}{m}\sum_{k=1}^m \mathrm{Tr}\left(\mathcal{E}_D\left(|\phi_k^T\rangle\langle\phi_k^T| \otimes |l_k\rangle\langle l_k|\right) \cdot \mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)\right) \\ + \frac{1}{m}\sum_{k=1}^m \mathrm{Tr}\left(\mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)^2\right).\end{aligned} \tag{5.13}$$

The computation of the Hilbert-Schmidt objective function requires much more computational resources than the fidelity objective function, as it is impossible to compute the discriminator's output with separate circuits for the generated and training data. Although the whole term can be evaluated with $3m$ circuits, each circuit comes at a higher cost as it requires two simultaneous evaluations of the discriminator and possibly the generator. In particular, more qubits are needed to perform these circuits. Additionally, the evaluation of the objective function no longer just contains one measurement in the computational basis, but each trace is calculated via the destructive swap test as shown in 4.18. Additionally, each circuit needs more qubits as two simultaneous evaluations of the discriminator, and possibly the generator are needed for each term. The generator's objective function comes with a slightly smaller computational cost as it just consists of two terms and, thus, can be evaluated using $2m$ circuits:

$$\begin{aligned}\max_G \mathcal{L}_G^{HS}(\mathcal{E}_G,\mathcal{E}_D,T,Z) = -\frac{2}{m}\sum_{k=1}^m \mathrm{Tr}\left(\mathcal{E}_D\left(|\phi_k^T\rangle\langle\phi_k^T| \otimes |l_k\rangle\langle l_k|\right) \cdot \mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)\right) \\ + \frac{1}{m}\sum_{k=1}^m \mathrm{Tr}\left(\mathcal{E}_D\left(\rho_k^G \otimes |l_k\rangle\langle l_k|\right)^2\right).\end{aligned} \tag{5.14}$$

A QGAN using the Hilbert-Schmidt objective functions will be referred to as HS-QGAN.

How can quantum generative adversarial networks be defined?

The fundamental definition of quantum generative adversarial networks is very similar to the classical case. The **generator** and the **discriminator** can both be **implemented as DQNNs** with a variety of possible architectures. The **quantum objective function represents a two-player minimax game** as well, and both networks are trained alternately. It should be possible to efficiently compute the objective function on a quantum computer whereby different measures such as the **fidelity or the Hilbert-Schmidt distance** may be used. Especially important is to **allow the generator DQNN to access a random latent space** to be able to extend the class of given training states.

5.3 Results

The training of QGANs is made difficult through various hyper-parameters whose choice is a priori not clear at all. The majority of hyper-parameters is just set and not further expanded on. Nevertheless, this chapter aims to create a fundamental understanding by thoroughly presenting two simple applications. Additionally, the crucial choice of the generator's and discriminator's network architectures is explained. Just like in Section 4.4, all results have been obtained using the open source SDK Qiskit [82]. However, here a statevector simulator is used which calculates the objective functions directly instead of exploiting the destructive swap test from 4.18.

Supervised QGAN training as a proof-of-concept

As a first proof-of-concept for the proposed QGAN algorithm, a labelled training set $\mathcal{T} = \{(|\phi_k^T\rangle, |l_k\rangle)\}_{k=1}^3$ will be learned without the use of any random latent states. Although generative adversarial learning is usually unsupervised, this concrete setting can be interpreted as supervised generative adversarial learning. To each training state, a unique label state $|l_k\rangle$ is assigned. Therefore the label states can be interpreted as the input states, while the training states $|\phi_k^T\rangle$ can be seen as the corresponding output states.

This setting can easily be compared to the supervised learning of a unitary transformation from Section 4.3. Here, the generator $\mathcal{E}_G$ takes the role of the QNN that should learn the unitary transformation from the input states to the output states. The major difference is the definition of the supervised loss function from Equation (4.17) compared to the definition of the F-QGAN objective function from Equation (5.9). The supervised loss function has direct access to the QNN's output state and evaluates its fidelity to the desired output state via the destructive

swap test. The F-QGAN (as well as the HS-QGAN) objective function does not directly compare the generator's output with the desired output. Instead, the discriminator accesses both states separately and learns to evaluate whether a state is correct with respect to the label state. So to say, the discriminator and the QGAN objective function can be seen as a substitute for the destructive swap test and the supervised loss function.

The training set features three of the four two-qubit maximally entangled Bell states with corresponding two-qubit label states:

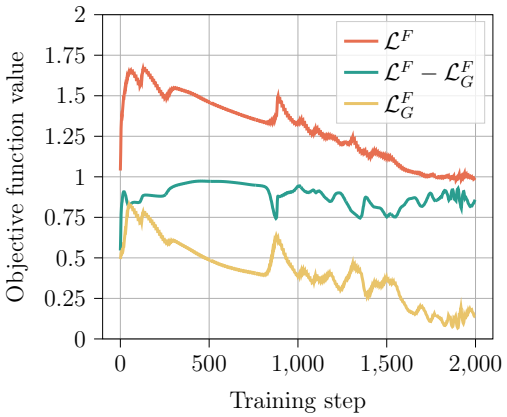
$$\begin{aligned} |\phi_1^T\rangle = |\Phi^+\rangle &= \frac{|00\rangle + |11\rangle}{\sqrt{2}}, \quad |l_1\rangle = (1,0,0,0)^T \\ |\phi_2^T\rangle = |\Phi^-\rangle &= \frac{|00\rangle - |11\rangle}{\sqrt{2}}, \quad |l_2\rangle = (0,1,0,0)^T \\ |\phi_3^T\rangle = |\Psi^+\rangle &= \frac{|01\rangle + |10\rangle}{\sqrt{2}}, \quad |l_3\rangle = (0,0,1,0)^T . \end{aligned} \tag{5.15}$$

The generator's goal for this supervised setting is to take a label state $|l_k\rangle$ as an input and produce the corresponding output state $\rho_k^G = \mathcal{E}_G(|l_k\rangle\langle l_k|) \approx |\phi_k^T\rangle$. Similar to the supervised learning from Section 4.3 an additional state pair is introduced

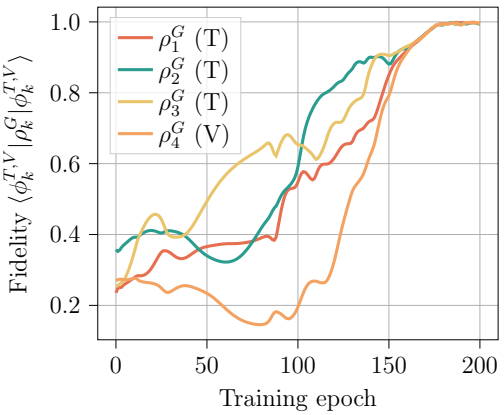
$$|\phi_4^V\rangle = |\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}, \quad |l_4\rangle = (0,0,0,1)^T \tag{5.16}$$

to test the generalisation of the training. It is important to note that the QGAN algorithm has no access to the fourth Bell state $|\phi_4^V\rangle = |\Psi^-\rangle$ but it is only used for validation. As the four Bell states and their label states are orthogonal it can be expected that a successfully trained generator maps the fourth label state to the fourth Bell state. To monitor the training's success, the fidelity between the desired states $|\phi_k^{T,V}\rangle$ and the corresponding generated states ρ_k^G is calculated. It is important to note that these fidelities are only calculated on the side and do not influence the QGAN training at all.

The results of the F-QGAN training are shown in Figure 5.2 for a discriminator training period $T_D = 9$ and 200 training epochs. This results in a total of 2000 training steps. The generator and the discriminator are defined as the minimal working DQNNs, namely a 2-2 generator and a 4-1 discriminator. The fidelity objective function is depicted in Figure 5.2a in three versions: the full objective function $\mathcal{L}^F$, the part based on the training states $\mathcal{L}^F - \mathcal{L}_G^F$, and the part based on the generated states $\mathcal{L}_G^F$. At first, the objective function $\mathcal{L}^F$ increases as the discriminator successfully distinguishes the real from the fake states. Finally, the objective function $\mathcal{L}^F$ decreases to 1 as the discriminator can not distinguish the states anymore and just accepts almost all generated states. The latter observation can be understood by the composition of the objective function from the parts



(a) The fidelity objective function $\mathcal{L}^F$ is used for training the generator and the discriminator. It is composed of two parts which evaluate the discriminator's success in correctly classifying the generated states as fake ($\mathcal{L}^F_G$) and the training states as real ($\mathcal{L}^F - \mathcal{L}^F_G$), respectively. At first, the discriminator classifies both types very well. Over time, the classification accuracy of the training states remains very good while the generated states are classified worse and worse. Finally, the discriminator classifies the training states and the generated states as real. The final convergence of $\mathcal{L}^F$ to 1 corresponds to the desired equilibrium of generative adversarial learning.



(b) The training's success is monitored by the fidelity between the desired training and validation states $|\phi_k^{T,V}\rangle$ and the corresponding generated states ρ_k^G. The fidelities are shown for the three generated states $\rho_{1,2,3}^G$ that are actually used for training (marked by a T) and for an additional generated state ρ_4^G which is never used during training (marked by a V). It is important to note that although $\rho_{1,2,3}^G$ are used for training, their fidelities with the desired states $|\phi_k^T\rangle$ are not. At the end of the training all fidelities converge to 1 which shows the training success.

Figure 5.2: Supervised QGAN training. The F-QGAN algorithm is used with a 2-2 generator DQNN and a 4-1 discriminator DQNN to learn a labelled training set $\mathcal{T} = \{|\phi_k^T\rangle, |l_k\rangle\}_{k=1}^3$ consisting of three of the four 2-qubit Bell states (see 5.15). This is a special case of generative adversarial learning as no latent states are used and the learning setting is supervised. The training is shown for 200 epochs where each epoch the discriminator is trained $T_D = 9$ times and the generator is trained once.

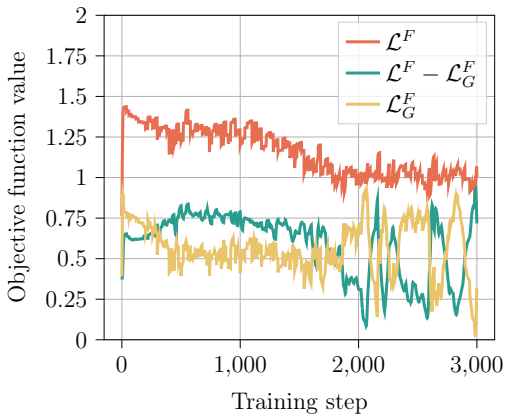
based on the real and generated states, respectively. The objective function's convergence to 1 is correlated with the fidelities' convergence to 1 as shown in Figure 5.2b. Interestingly, this is not only the case for the three training states but also the validation state. These results serve as a proof-of-concept for the QGAN algorithm, especially with using the fidelity objective function. However, it must be remembered that this only covers the special case of supervised generative adversarial learning and the unsupervised case remains to be tested.

Unsupervised QGAN training

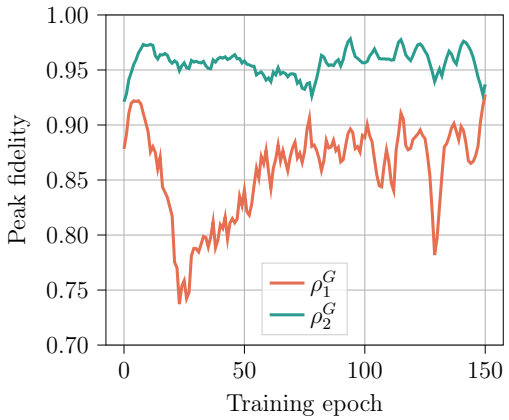
As a fundamental test of unsupervised QGAN training, an unlabelled training set will be learned with the use of random latent states. The training set features one-qubit states randomly distributed over a quarter of the Bloch sphere's equator. In particular, the training set is given as:

$$\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^8 \quad \text{with} \quad |\phi_k^T\rangle = R_Z(\varphi_k) H |0\rangle, \ \varphi_k \in \left[0, \frac{\pi}{2}\right] \tag{5.17}$$

with random angles φ_k and is shown in Figure 5.4a. In this case, the goal of the QGAN training is to obtain a generator DQNN which produces states on the same quarter of the Bloch sphere's equator when provided with random latent states as inputs.



(a) The fidelity objective function $\mathcal{L}^F$ is used for training the both networks and converges to 1 after approximately 2000 training steps. The objective function's oscillation at the end of the training indicates that the discriminator alternates its strategy between accepting and rejecting all input states.



(b) The training's success is monitored by the peak fidelity for two fixed one-qubit latent states. With respect to these latent states, the generator produces the fake states ρ^G_1 and ρ^G_1, respectively. The peak fidelity describes the fidelity between the generated state and the nearest training state.

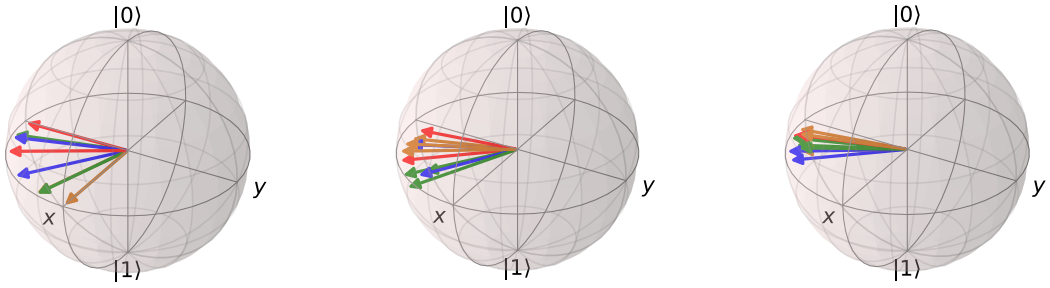
Figure 5.3: Unsupervised QGAN training. The F-QGAN algorithm is used with a 1-1 generator DQNN and a 1-1 discriminator DQNN to learn an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle, |l_k\rangle\}_{k=1}^8$ consisting of 10 one-qubit states randomly distributed over a quarter of the Bloch sphere's equator (see 5.4a). The training is shown for 150 epochs where each epoch the discriminator is trained $T_D = 19$ times and the generator is trained once.

The results of the F-QGAN training are shown in Figure 5.3 for a discriminator training period $T_D = 19$ and 150 epochs training epochs. This results in a total of 3000 training steps. The generator and the discriminator are defined as the minimal working DQNNs, namely, a 1-1 generator taking a one-qubit latent state as input and a 1-1 discriminator with a generated or training state as its input. The objective function value is shown in Figure 5.3a. As opposed to the previous analysis, there is no obvious quantity to evaluate the training success because the setting is now unsupervised, and there is no exactly right solution for the problem. Here, the *peak fidelity* is proposed to enable the monitoring of the training. It is defined as:

$$\text{peak fidelity} = \max\left(\{\langle\phi_k^T|\rho^G|\phi_k^T\rangle\}_{k=1,\ldots,N_T}\right) \tag{5.18}$$

with N_T training states $|\phi_k^T\rangle$, the generated state $\rho^G = \mathcal{E}_G\left(|z\rangle\langle z|\right)$, and a validation latent state $|z\rangle$. Here, two different validation latent states are randomly sampled in the beginning and used for the peak fidelity calculation as shown in Figure 5.3b. Although the objective function reaches its equilibrium state after approximately 2000 training steps, the peak fidelity does not reach 1. However, this behaviour is expected because a convergence to 1 would mean that the generator only learns to

exactly produce the training states but not to extend the training set.



(a) Training states. Randomly distributed states on a quarter of the Bloch sphere's equator.

(b) Epoch 87. Diverse set of generated states from a cherry-picked generator.

(c) Epoch 148. Non-diverse set of generated states with a generator suffering from mode collapse.

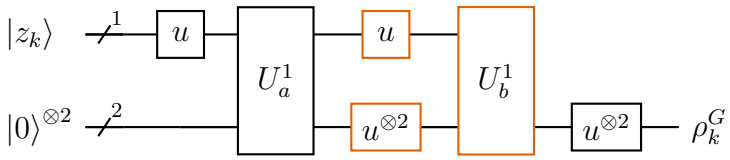
Figure 5.4: Bloch sphere representations of training and generated states. The F-QGAN algorithm is used with a 1-1 generator DQNN and a 1-1 discriminator DQNN to learn the unlabelled training set shown in (a). Two different sets of generated states using the same latent states are shown at different epochs 87 and 148 in (b) and (c), respectively.

To evaluate the performance of generative adversarial learning it is often useful to look at the generated states themselves. Therefore, two sets of generated states are shown on the Bloch sphere in Figures 5.4b and 5.4c, respectively. Both sets have been generated while the objective function $\mathcal{L}^F$ already converged to 1. The first set from Figure 5.4b shows the generated states during the training epoch 87 and has been cherry-picked to show the generator's capability to produce a diverse set of new states with the internal features of the training set. The second set from Figure 5.4c shows the generated states during the late training epoch 148. Here, as well, the generated states are within the desired class of states. However, during this epoch the generator suffers from a weak mode collapse, i.e., it produces a small subset of the desired class of states although provided with a diverse set of latent states. It can also happen that the generator suffers from a rather strong mode collapse such that it ignores the input latent state completely and always generates the same state.

Different generator and discriminator networks

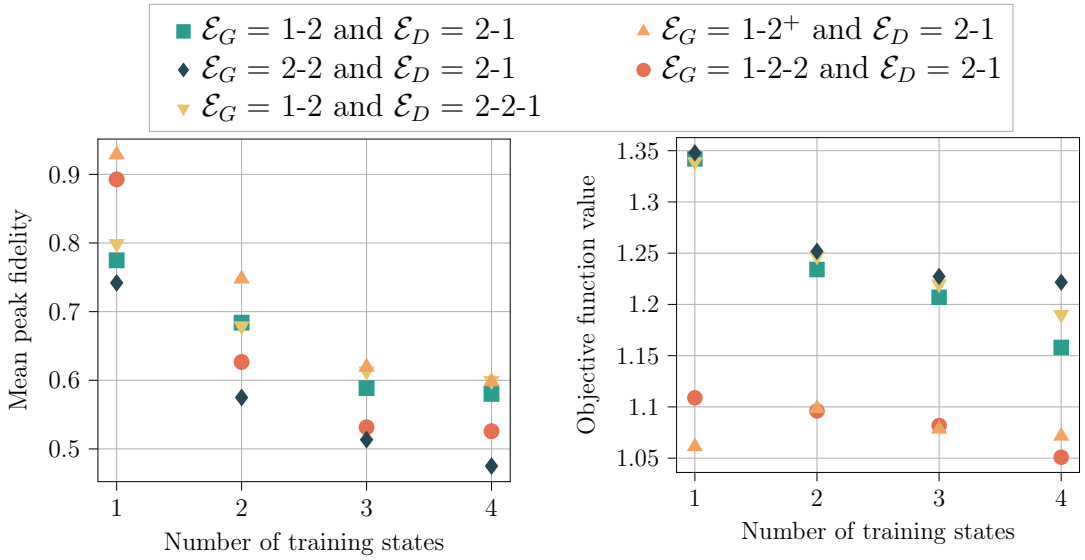
After ensuring the general functionality of the proposed F-QGAN algorithm, it is important to analyse different QNN architectures for the generator and the discriminator. As a setting for this experiment, an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^{N_T}$ with randomly sampled 4-dimensional pure states $|\phi_k^T\rangle$ is used. This narrows down the possible generator DQNNs to those with 2 output qubits and the discriminator DQNNs to those with 2 input qubits and 1 output qubit. The remaining degrees of freedom are, therefore, the number of latent state qubits $n_{\text{latent}} \geq 1$, the structure of the generator's and discriminator's hidden layers, and fundamental modifications to the DQNN's quantum perceptron.

The minimal working generator is a 1-2 DQNN while the simplest discriminator is a 2-1 DQNN. The most obvious modification of these networks is to add a hidden layer such that the generator would become a 1-2-2 DQNN while the discriminator would become a 2-2-1 DQNN. These modifications will be tested individually while using the other network's minimal working version. Instead of adding a hidden layer, which requires additional qubits, the quantum perceptron could be modified as well such that the DQNN's layer-to-layer transition gets computationally more powerful. This modification will be denoted by 1-2^+ with the corresponding DQNN defined as:



(5.19)

with the additional coloured quantum gates compared to only the black quantum gates which constitute the standard 1-2 DQNN. The layer unitaries U_a^1 and U_b^1 share the same structure as defined in Section 4.3 but are parameterised independently. A further modification of the generator network is possible by increasing the number of latent state qubits to $n_{\text{latent}} = 2$ which results in a 2-2 DQNN.



(a) The mean peak fidelity is averaged over all generated states from the last 20 training epochs. It quantifies the generator's capability to produce states close to at least one of the training states.

(b) The objective function is averaged over the last 20 training epochs. The F-QGAN's desired equilibrium is at 1, the HS-QGAN's is at 0. A larger value shows that the discriminator outperforms the generator.

Figure 5.5: Comparison of different generator and discriminator networks. The F-QGAN algorithm is used with different choices for the generator and discriminator networks, respectively, to learn an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^{N_T}$ with randomly sampled 4-dimensional pure states $|\phi_k^T\rangle$. The results are averaged over 10 runs with 250 epochs, respectively.

The numerical results of the F-QGAN training for different numbers of training

states $N_T = 1, \ldots, 4$ are shown in Figure 5.5 for a discriminator training period $T_D = 9$ and 250 epochs, respectively. The shown properties are averaged over the last 20 epochs for each training run, and over 10 training runs in total. Here, the *mean peak fidelity* is used to enable the numerical investigation of the different networks as shown in Figure 5.2b. It is defined as:

$$\text{mean peak fidelity} = \frac{1}{N_V} \sum_{n=1}^{N_V} \max \left(\{ \langle \phi_k^T | \rho_n^G | \phi_k^T \rangle \}_{k=1,\ldots,N_T} \right) \tag{5.20}$$

with N_V validation latent states $|z_n\rangle$, N_T training states $|\phi_k^T\rangle$ and the generated states $\rho_n^G = \mathcal{E}_G \left(|z_n\rangle \langle z_n| \right)$. Here, $N_V = 4$ validation latent states are randomly sampled in the beginning and fixed for all epochs. In addition to the mean peak fidelity, the objective function value is shown in Figure 5.5b and offers a second quantity to evaluate the training success.

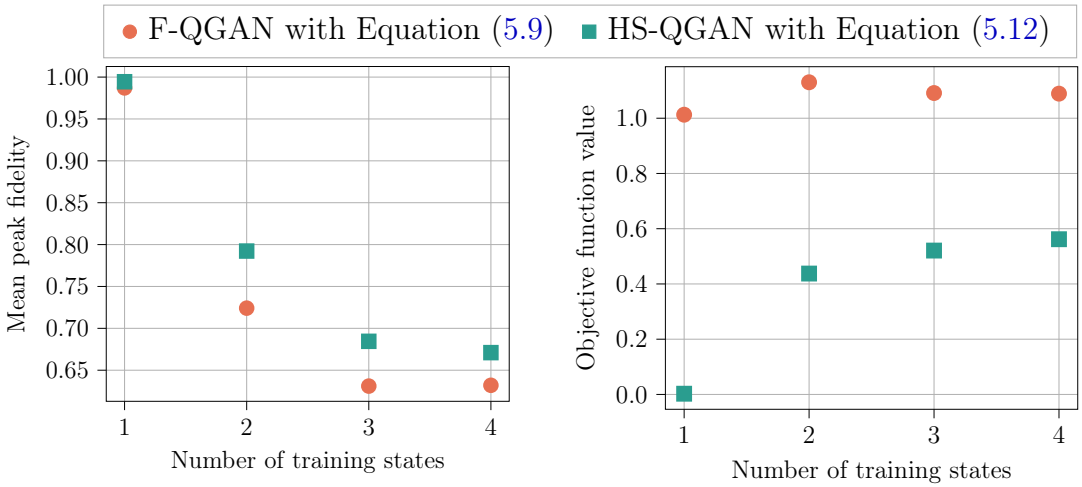
The first relevant observation is that for the case of a single training state, the 1-2 and 2-2 generators achieve much smaller peak fidelities than the 1-2^+ and the 1-2-2 generators. This is due to their inability to ignore a random input state while creating the desired training state. By contrast, the latter generators have this capability thanks to their more frequent application of quantum gates (see 5.19) or an additional layer. The difference in the generator's computational capabilities can also be observed in the objective function values. These are generally smaller for the 1-2^+ and the 1-2-2 generator which shows that these generators fool the respective discriminator more successfully. A large objective function value at the end of training means that the discriminator outperforms the generator.

For each network pair, the mean peak fidelity decreases for an increasing number of training states N_T. As the number of training states increases, the generator has to learn to generate a broader class of states from the random latent states, i.e., the generator is faced with the task to use some feature of the latent state to decide which type of state to produce. The more different training states, the harder it is for the generator to find enough different latent state features to exploit.

From the tested network configurations, the most successfully trained combination is a 1-2^+ generator with a 2-1 discriminator. It achieves some of the highest fidelities for all different numbers of training states. Especially, the comparison with the standard 1-2 generator shows the advantage gained by modifying the DQNN's layer-to-layer transition. The usage of a 2-2-1 compared to a 2-1 discriminator marginally improves the performance. The use of larger generators such as the 1-2-2 or the 2-2 DQNNs worsens the performance, probably due to poorer trainability. Conclusively, this analysis in addition to thoughts about efficiency suggest the use of the 1-2^+ DQNN generator with a 2-1 DQNN discriminator for the further investigation of QGANs.

Comparison of Hilbert-Schmidt and fidelity objective functions

The following analysis concerns the choice of the objective function, which results in the previously used F-QGAN algorithm or the alternative HS-QGAN algorithm, respectively. The setting for this comparison is based on the previous result, which encourages the use of a 1-2^+ generator and a 2-1 discriminator for learning an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^{N_T}$ with randomly sampled 4-dimensional pure states $|\phi_k^T\rangle$. Here, the F-QGAN and the HS-QGAN are compared for $N_T = 1, \ldots, 4$ random two-qubit training states.



(a) The mean peak fidelity is averaged over all generated states from the last 20 training epochs. It quantifies the generator's capability to produce states close to at least one of the training states.

(b) The objective function is averaged over the last 20 training epochs. The F-QGAN's desired equilibrium is at 1, the HS-QGAN's is at 0. A larger value shows that the discriminator outperforms the generator.

Figure 5.6: Comparison of F-QGAN and HS-QGAN training. Both QGAN algorithms are used with a 1-2^+ generator DQNN and a 2-1 discriminator DQNN to learn an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^{N_T}$ with randomly sampled 4-dimensional pure states $|\phi_k^T\rangle$. The results are averaged over 10 runs with 250 epochs, respectively.

The numerical results are shown in Figure 5.6 as an average over 10 separate runs with different training sets. The HS-QGAN reaches higher peak fidelities than the F-QGAN for all numbers of training states. The objective function values are not directly comparable. While the F-QGAN's desired equilibrium state is at a function value of 1, the HS-QGAN's is at 0. Therefore, both QGAN's reach their desired final state for $N_T = 1$, i.e., a single training state, but the generators are outperformed by the discriminator for $N_T > 1$.

However, it is not so clear whether to choose the HS-QGAN or the F-QGAN. Different criteria must be taken into account. First, a higher mean peak fidelity does not necessarily prove a better QGAN performance. It just shows the generator's capability to produce states close to one of the training states but does not account for the diversity of the generated states. Second, the HS-QGAN re-

quires more computational resources regarding qubits and quantum gates for the objective function evaluation. Therefore, the effects of noise in actual quantum computers are expected to weaken the performance of the HS-QGAN more than for the F-QGAN. Which of the objective functions is advantageous in the end is unclear and may also depend on the specific task at hand.

Different types of training sets

The final analysis aims to understand which kind of training data is particularly suitable for QGAN training. The interest is especially on generator DQNNs with one-qubit latent states. These networks consist of completely positive layer-to-layer transition maps with a 2-dimensional latent space as their input space, i.e., only 2 linearly independent input states are available. The conjecture is that a generator network with access to a 2-dimensional latent space can only efficiently produce a set of states with maximally 2 linearly independent states regardless of the output state's dimension. To test this conjecture, the QGAN will be faced with training sets of the following form:

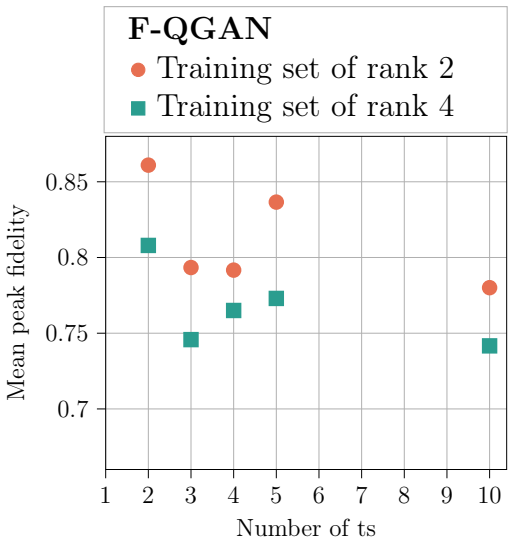
$$\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^{N_T} \quad \text{with} \quad |\phi_k^T\rangle = \frac{\sum_{n=1}^{\mathrm{R}} p_n |\psi_n\rangle}{\sqrt{\sum_{n=1}^{\mathrm{R}} p_n^2}},\ p_n \in [0,1] \tag{5.21}$$

with R random d-dimensional states $|\psi_n\rangle$, and random probabilities p_n. The training set is built by random linear combinations of R, in general, linearly independent states. Thus, the resulting training set has a maximal rank R when written in a matrix form.

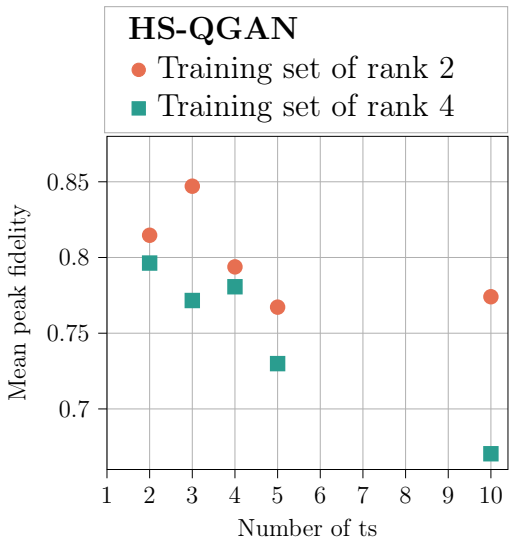
Both the F-QGAN and the HS-QGAN are trained with two training sets of different ranks 2 and 4 with 8-dimensional training states. An analogous numerical analysis with 4-dimensional training states is shown in Appendix B. The generator is given by a 1-3^+ DQNN while the discriminator is realised as a 3-1 DQNN. As the training set rank should not affect the training performance for only one training state, the number of training states is iterated over $N_T = 2,3,4,5,10$. In Figure 5.7 the results are shown in terms of the mean peak fidelity from Equation (5.20) averaged over 10 runs with 250 training epochs, respectively.

Both QGANs reach higher mean peak fidelities for the training set with rank 2 than for the training set with rank 4. The difference is particularly evident for a large number of training states. Here, the generator network seems to have more difficulty to extract information from the latent state to generate states close to the training states. The generator has an easier job to map the 2-dimensional latent states to a training set featuring only 2 instead of 4 linearly independent states.

One example for a training set of rank 2 with 8 training states is given by Equation (5.17). Here, the generator uses the 2-dimensional latent state to interpolate between the 2-dimensional training states (see Figure 5.4b). This can very



(a) The mean peak fidelity is averaged over all generated states from the last 20 training epochs. It quantifies the generator's capability to produce states close to at least one of the training states.



(b) The mean peak fidelity is averaged over all generated states from the last 20 training epochs. It quantifies the generator's capability to produce states close to at least one of the training states.

Figure 5.7: Different ranks for 8-dimensional training states. The F-QGAN (a) and the HS-QGAN algorithm (b) are used with a 1-3$^+$ generator DQNN and a 3-1 discriminator DQNN to learn an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^{N_T}$ with 8-dimensional pure states $|\phi_k^T\rangle$ of ranks 2 and 4 according to Equation (5.21). The results are averaged over 10 runs with 250 epochs, respectively.

likely be generalised to higher-dimensional training states such that the generator tries to interpolate between the 8-dimensional training states of rank 2 and rank 4. The results indicate that in the case of a generator with access to a 2-dimensional latent state, this works out better for a training set with only 2 linearly independent states. However, as mentioned before, the mean peak fidelity does not take into account the generator's diversity (compare Figures 5.3 and 5.4). It should therefore not be viewed as the QGAN's ultimate measure of success but rather as an instrument to monitor its general performance.

Chapter 6

Conclusion

The goal of this thesis has been to explore the notions of generative adversarial learning with quantum neural networks and establish a foundation for its application and further research. To keep this thesis self-consistent, it started with explaining the general concepts of quantum computing and classical neural networks before discussing the realisation of quantum neural networks. Having understood the basics of quantum neural networks, the thesis finally moved on to the definition of quantum generative adversarial networks and their investigation.

In the beginning, the most relevant features of quantum computing were introduced. This includes, in particular, a thorough explanation of the qubit along with its critical properties, namely superposition and entanglement. It has been explained that quantum computers apply quantum gates on the qubits by executing quantum circuits. The most important quantum gates have been defined, and their effects have been shown within an educational exemplary quantum circuit along with the Bloch sphere representations of the qubits' states. Moreover, some ideas of quantum advantage have been outlined, including quantum parallelism and how to exploit this phenomenon by employing Deutsch's algorithm. The challenges and the status quo of building actual quantum computers have been reported resulting in a discussion of the limitations for today's noisy intermediate-scale quantum devices.

Furthermore, the general ideas of classical neural networks have been reviewed. The foundation is given by the perceptron, which is the fundamental building block of a neural network and connects the neurons of adjacent network layers. The perceptron can be refined to a similar but computationally more powerful version, namely, the sigmoid neuron. By using the sigmoid neuron, it has been argued that a neural network can, in principle, compute any continuous function. To exploit this valuable property, the general structure of supervised training is explained, which features at its heart a loss function. The loss function guides the

training by evaluating the neural network's capability to solve a specific task. The explicit training of the neural network's parameters is performed using the gradient descent method with the backpropagation algorithm. Finally, some challenges of classical neural networks are discussed, especially finding good hyper-parameters, avoiding overfitting, and resolving the vanishing gradient problem.

Moreover, as the first part containing self-gained knowledge, the combination of quantum computing and classical neural networks to quantum neural networks is examined. As a start, the general notions of already existing quantum neural networks has been discussed. Here too, the definition of the loss function is at the heart of training a quantum neural network. The loss function has to fulfil some conditions, the most important being that it can be efficiently computed on a quantum computer to justify the use of quantum neural networks. While the gradient descent method can be used to train quantum neural networks, the backpropagation algorithm for the gradient calculation can not be used. Instead, the loss function's gradient is generally calculated via the central difference approximation for each network parameter separately. Different definitions of the fundamental building block, the quantum perceptron, are reviewed, including those which result in the dissipative quantum neural network (DQNN) and the quantum approximate optimisation algorithm (QAOA). Both named ansatzes are explicitly implemented as quantum circuits.

The quantum circuit implementation of the DQNN corresponds to this work's proposal for a quantum neural network. It features a completely positive layer-to-layer transition map whereby the different layers are built from different sets of qubits. The explicit definition of the quantum perceptron is based on the fact that any universal two-qubit quantum gate can be composed into a series of single-qubit gates and a two-qubit canonical gate. This results in a layer-to-layer transition map that employs canonical gates to connect all qubits of adjacent layers and additional single-qubit gates to improve the map's computational power.

The DQNN and the QAOA have been compared in the task of learning an unknown unitary transformation. The networks are provided with a training set of input and output pairs of quantum states and a corresponding supervised loss function to accomplish this task. First and foremost, an analysis of the influence of gate noise on both networks' performances has been carried out, ranging from a noise-free simulation to a simulation including noise similar to today's quantum computers. The numerical results indicate that the DQNN is more suitable for learning unitaries on today's quantum computers than the QAOA. Additionally, both networks have been trained on an actual quantum computer. Besides showing the network's surprising ability to factor in the noise circumstances of the used quantum computer, the results also show that the DQNN is slightly advantageous.

Finally, the notions of generative adversarial learning have been applied to the quantum case resulting in quantum generative adversarial networks (QGANs) consisting of two DQNNs. The main idea is to alternately train a generator DQNN

and a discriminator DQNN with the ultimate goal of obtaining a generator that produces quantum states that extend a given training set. During the unsupervised training, the discriminator's task is to distinguish fake states from real training states, while the generator's task is to fool the discriminator by producing states similar to the training states. To control the training, an objective function is used, which describes a two-player minimax game between the generator and the discriminator.

One of the main contributions of this work is the proposal of two objective functions for training QGANs. The first proposal is called the fidelity objective function, which consists of two parts: a fidelity measurement between $|0\rangle$ and the discriminator's output when fed with generated states and a fidelity measurement between $|1\rangle$ and the discriminator's output when fed with training states. The second proposal is called the Hilbert-Schmidt objective function and can not be separated into two independent parts. Instead, it calculates the Hilbert-Schmidt distance between the respective discriminator's output when fed with generated and training states. In comparison, the required computational resources for the Hilbert-Schmidt objective function are much larger in the number of qubits and quantum gates. However, it also comes with fewer restrictions for the discriminator's optimal behaviour.

The proposed QGAN algorithm has been applied to various training sets and analysed concerning its most essential properties. As a proof-of-concept, the QGAN has been trained successfully with a labelled training set resulting in a generator that produces the four Bell states from different label states. This learning setting is supervised and thus, does not support the broader idea of generative adversarial learning to generate states that extend the training set. Therefore, the successful QGAN training of an unlabelled training set is presented to convey an understanding of the unsupervised learning behaviour and the resulting generator. Here, the undesired phenomenon of mode collapse is introduced, which counteracts a high diversity of the generator's states. An analysis of different generator and discriminator network architectures has been conducted. It turns out that for 4-dimensional training states, the modified 1-2^+ DQNN with a one-qubit latent state makes a good generator while the 2-1 DQNN constitutes a well-performing discriminator. Using these networks, the fidelity objective function and the Hilbert-Schmidt objective function have been compared in learning random training sets of different sizes. The Hilbert-Schmidt objective function achieves higher mean peak fidelities, however, this result does not consider the generator's diversity and the larger required computational resources for the Hilbert-Schmidt objective function. As a final analysis, the QGAN has been trained on training sets with different numbers of linearly independent training states. The analysis shows the QGAN's better performance on a training set with 2 instead of 4 linearly independent training states. This allows conclusions on how the generator exploits the latent state to interpolate between different training states and to extend the training set.

Due to the high complexity of training QGANs, many exciting questions remain unanswered. The QGAN algorithm features many hyper-parameters which could not be analysed to the last detail, including the discriminator's training period and the learning rates. In addition to that, more complex network architectures might be advantageous for specific learning tasks. Furthermore, it is highly interesting how the QGAN performs on other training sets, e.g., featuring different levels of entanglement [94]. As a logical continuation of this work, other measures for the QGAN's performance should be proposed, possibly taking into account the generator's diversity. Of course, the future of QGANs lies in their application on actual quantum computers. Therefore, the QGAN's noise dependence, especially for the different objective functions, plays a crucial part in the future use of the techniques developed in this work.

The code for the implementation of the DQNN and the QAOA from Chapter 4 and the QGANs from Chapter 5 is available at `https://github.com/qigitphannover/DeepQuantumNeuralNetworks`.

Bibliography

[1] R. R. Schaller, IEEE spectrum **34**, 52 (1997).

[2] R. P. Feynman, Int. J. Theor. Phys **21** (1982).

[3] M. A. Nielsen and I. Chuang, American Journal of Physics **70**, 558 (2002).

[4] P. W. Shor, in *Proceedings 35th annual symposium on foundations of computer science* (Ieee, 1994) pp. 124–134.

[5] L. K. Grover, in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (1996) pp. 212–219.

[6] J. Preskill, Quantum **2**, 79 (2018).

[7] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, Nature **574**, 505 (2019).

[8] IBM Quantum team, IBM Quantum Experience, `https://quantum-computing.ibm.com` (2021).

[9] P. Shor, in *Proceedings of 37th Conference on Foundations of Computer Science* (1996) pp. 56–65.

[10] M. I. Jordan and T. M. Mitchell, Science **349**, 255 (2015).

[11] J. Golbeck, C. Robles, and K. Turner, in *CHI'11 extended abstracts on human factors in computing systems* (2011) pp. 253–262.

[12] S. Shalev-Shwartz, S. Shammah, and A. Shashua, (2016), arXiv:1610.03295 [cs.AI] .

[13] M. W. Libbrecht and W. S. Noble, Nature Reviews Genetics **16**, 321 (2015).

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).

[15] M. A. Nielsen, *Neural networks and deep learning*, Vol. 25 (Determination press San Francisco, CA, 2015).

[16] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, Phys. Rev. A **98**, 032309 (2018).

[17] J. Stokes, J. Izaac, N. Killoran, and G. Carleo, Quantum **4**, 269 (2020).

[18] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, Phys. Rev. A **99**, 032331 (2019).

[19] M. Ostaszewski, E. Grant, and M. Benedetti, arXiv:1905.09692 (2019), arXiv:1905.09692 .

[20] K. Bu, D. E. Koh, L. Li, Q. Luo, and Y. Zhang, (2021), arXiv:2101.06154 .

[21] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, Quantum Sci. Technol. **4**, 043001 (2019).

[22] Y. Du, M.-H. Hsieh, T. Liu, and D. Tao, (2018), arXiv:1810.11922 .

[23] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, *et al.*, (2020), arXiv:2012.09265 .

[24] K. Beer, D. Bondarenko, T. Farrelly, T. J. Osborne, R. Salzmann, D. Scheiermann, and R. Wolf, Nat commun **11**, 1 (2020).

[25] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe, Phys. Rev. A **101**, 032308 (2020).

[26] K. Sharma, M. Cerezo, L. Cincio, and P. J. Coles, (2020), arXiv:2005.12458 .

[27] E. Farhi, J. Goldstone, and S. Gutmann, (2014), arXiv:1411.4028 .

[28] Y. Li and S. C. Benjamin, Physical Review X **7**, 021050 (2017).

[29] X. Yuan, S. Endo, Q. Zhao, Y. Li, and S. C. Benjamin, Quantum **3**, 191 (2019).

[30] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. Oâbrien, Nature communications **5**, 1 (2014).

[31] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, (2014), arXiv:1406.2661 [stat.ML] .

[32] M. Arjovsky, S. Chintala, and L. Bottou, (2017), arXiv:1701.07875 [stat.ML] .

[33] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, (1999), arXiv:quant-ph/9906054 .

[34] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, Physical review A **52**, 3457 (1995).

[35] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani, Physical review letters **73**, 58 (1994).

[36] D. Deutsch, Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences **400**, 97 (1985).

[37] D. Deutsch and R. Jozsa, Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences **439**, 553 (1992).

[38] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **454**, 339 (1998).

[39] I. L. Chuang, L. M. Vandersypen, X. Zhou, D. W. Leung, and S. Lloyd, Nature **393**, 143 (1998).

[40] S. Gulde, M. Riebe, G. P. Lancaster, C. Becher, J. Eschner, H. Häffner, F. Schmidt-Kaler, I. L. Chuang, and R. Blatt, Nature **421**, 48 (2003).

[41] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms* (Addison-Wesley Professional, 2014).

[42] J. P. Buhler, H. W. Lenstra, and C. Pomerance, in *The development of the number field sieve* (Springer, 1993) pp. 50–94.

[43] D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, Physical Review A **54**, 1034 (1996).

[44] I. M. Georgescu, S. Ashhab, and F. Nori, Reviews of Modern Physics **86**, 153 (2014).

[45] Y. Manin, Sovetskoye Radio, Moscow **128** (1980).

[46] W. Gerlach and O. Stern, Zeitschrift für Physik **9**, 349 (1922).

[47] J. J. Sakurai and E. D. Commins https://doi.org/10.1119/1.17781 (1995).

[48] D. P. DiVincenzo, Science **270**, 255 (1995).

[49] F. Rosenblatt, Psychological review **65**, 386 (1958).

[50] M. W. Gardner and S. Dorling, Atmospheric environment **32**, 2627 (1998).

[51] K. Hornik, M. Stinchcombe, and H. White, Neural networks **2**, 359 (1989).

[52] X. Zhang, Y. Yu, L. Wang, and Q. Gu, in *The 22nd international conference on artificial intelligence and statistics* (PMLR, 2019) pp. 1524–1534.

[53] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, in *International Conference on Machine Learning* (PMLR, 2019) pp. 1675–1685.

[54] L. Bottou, in *Proceedings of COMPSTAT'2010* (Springer, 2010) pp. 177–186.

[55] M. Li, T. Zhang, Y. Chen, and A. J. Smola, in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014) pp. 661–670.

[56] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, Tech. Rep. (California Univ San Diego La Jolla Inst for Cognitive Science, 1985).

[57] I. V. Tetko, D. J. Livingstone, and A. I. Luik, Journal of chemical information and computer sciences **35**, 826 (1995).

[58] L. Prechelt, in *Neural Networks: Tricks of the trade* (Springer, 1998) pp. 55–69.

[59] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, The journal of machine learning research **15**, 1929 (2014).

[60] S. J. Nowlan and G. E. Hinton, Neural computation **4**, 473 (1992).

[61] S. Hochreiter, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**, 107 (1998).

[62] A. F. Agarap, (2018), arXiv:1803.08375 .

[63] K. Beer, D. List, G. Müller, T. J. Osborne, and C. Struckmann, (2021), arXiv:2104.06081 [quant-ph] .

[64] A. Mari, T. R. Bromley, and N. Killoran, Physical Review A **103**, 10.1103/physreva.103.012405 (2021).

[65] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, Nature **549**, 242 (2017).

[66] E. Farhi and A. W. Harrow, (2016), arXiv:1602.07674 .

[67] S. Hadfield, Z. Wang, B. OâGorman, E. G. Rieffel, D. Venturelli, and R. Biswas, Algorithms **12**, 34 (2019).

[68] S. Lloyd, (2018), arXiv:1812.11075 [quant-ph] .

[69] J. Biamonte, Physical Review A **103**, 10.1103/physreva.103.l030401 (2021).

[70] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, Nature Commun **9**, 1 (2018).

[71] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, and P. J. Coles, (2020), arXiv:2001.00550 .

[72] E. Grant, L. Wossnig, M. Ostaszewski, and M. Benedetti, Quantum **3**, 214 (2019).

[73] M. Alam, A. Ash-Saki, and S. Ghosh, (2019), arXiv:1907.09631 .

[74] C. Xue, Z.-Y. Chen, Y.-C. Wu, and G.-P. Guo, (2019), arXiv:1909.02196 .

[75] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P. J. Coles, (2020), arXiv:2007.14384 .

[76] C. Y.-Y. Lin and Y. Zhu, (2016), arXiv:1601.01744 .

[77] Z. Wang, S. Hadfield, Z. Jiang, and E. G. Rieffel, Phys. Rev. A **97**, 022304 (2018).

[78] B. T. Kiani, S. Lloyd, and R. Maity, (2020), arXiv:2001.11897 .

[79] G. E. Crooks, (2019), arXiv:1905.13311 .

[80] L. Cincio, Y. Subaşi, A. T. Sornborger, and P. J. Coles, New J. Phys. **20**, 113022 (2018).

[81] J. C. Garcia-Escartin and P. Chamorro-Posada, Phys. Rev. A **87**, 052330 (2013).

[82] H. Abraham *et al.*, Qiskit: An open-source framework for quantum computing (2019).

[83] K. Poland, K. Beer, and T. J. Osborne, (2020), arXiv:2003.14103 .

[84] B. Nachman, M. Urbanek, W. A. de Jong, and C. W. Bauer, npj Quantum Information **6**, 1 (2020).

[85] E. Magesan, J. M. Gambetta, and J. Emerson, Physical Review A **85**, 10.1103/physreva.85.042311 (2012).

[86] H. Alqahtani, M. Kavakli-Thorne, and G. Kumar, Archives of Computational Methods in Engineering , 1 (2019).

[87] J. Nash, Annals of Mathematics **54**, 286 (1951).

[88] A. K. Wong and M. You, IEEE transactions on pattern analysis and machine intelligence , 599 (1985).

[89] L. N. Vaserstein, Problemy Peredachi Informatsii **5**, 64 (1969).

[90] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, (2017), arXiv:1611.02163 [cs.LG] .

[91] S. Lloyd and C. Weedbrook, Physical review letters **121**, 040502 (2018).

[92] P.-L. Dallaire-Demers and N. Killoran, Physical Review A **98**, 012324 (2018).

[93] M. Y. Niu, A. Zlokapa, M. Broughton, S. Boixo, M. Mohseni, V. Smelyanskyi, and H. Neven, (2021), arXiv:2105.00080 .

[94] L. Schatzki, A. Arrasmith, P. J. Coles, and M. Cerezo, (2021), arXiv:2109.03400 [quant-ph] .

[95] D. S. Abrams and S. Lloyd, Physical Review Letters **83**, 5162 (1999).

[96] O. Higgott, D. Wang, and S. Brierley, Quantum **3**, 156 (2019).

[97] D. Poulin, A. Qarry, R. Somma, and F. Verstraete, Physical review letters **106**, 170501 (2011).

[98] D. Bondarenko and P. Feldmann, Phys. Rev. Lett. **124**, 130502 (2020).

[99] T. Achache, L. Horesh, and J. Smolin, (2020), arXiv:2012.14714 .

[100] K. H. Wan, O. Dahlsten, H. Kristjánsson, R. Gardner, and M. Kim, npj Quantum Inf **3**, 1 (2017).

[101] J. Romero, J. P. Olson, and A. Aspuru-Guzik, Quantum Science and Technology **2**, 045001 (2017).

[102] A. Pepper, N. Tischler, and G. J. Pryde, Physical review letters **122**, 060501 (2019).

[103] M. Schuld and N. Killoran, Phys. Rev. Lett. **122**, 040504 (2019).

[104] K. Zhang, M.-H. Hsieh, L. Liu, and D. Tao, (2020), arXiv:2011.06258 .

[105] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, Nature **567**, 209 (2019).

[106] M. R. Geller, Z. Holmes, P. J. Coles, and A. Sornborger, (2021), arXiv:2104.03295 [quant-ph] .

Appendix A

Applications of QNNs

Applications

The framework of VQAs, i.e., using the computational power of a PQC and training its parameters classically, is shown to be universal for quantum computing [69]. This motivates the application of VQAs to a wide range of problems, and in fact, these models have been successfully implemented on a variety of tasks - a few of these will be reviewed in the following.

The first proposal for a VQA in the literature aimed to find the ground state $|\psi_G\rangle$ of a given Hamiltonian H, namely the *variational quantum eigensolver* [30]. Although there exist other quantum algorithms to solve this task using quantum phase estimation [95], these require many more quantum gates than NISQ devices are able to execute with reasonable noise. Instead, a VQA can be used with the objective to minimise the loss function

$$\mathcal{L}(\vec{\theta}) = \langle\psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle \tag{A.1}$$

where a PQC is used to variationally produce the state $|\psi(\vec{\theta})\rangle = U(\vec{\theta})\,|\psi_0\rangle$ with some initial state $|\psi_0\rangle$. The loss is minimised for $|\psi(\vec{\theta}^*)\rangle \approx |\psi_G\rangle$ and in this case equals the ground energy E_G of the Hamiltonian H. By adjusting the Hamiltonian to $H' = H + a\,|\tilde{\psi}_G\rangle\langle\tilde{\psi}_G|$ with $|\tilde{\psi}_G\rangle = |\psi(\vec{\theta}^*)\rangle$ and some $a > 0$ much larger than the energy difference between the first excited state and the ground state of H, the VQA can be extended to compute an excited state as well [96]. Training the VQA with the adjusted loss function

$$\mathcal{L}(\vec{\theta}) = \langle\psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle + a\,\langle\psi(\vec{\theta})|\tilde{\psi}_G\rangle\,\langle\tilde{\psi}_G|\psi(\vec{\theta})\rangle \tag{A.2}$$

now yields the first excited state of H. This method can be extended iteratively to find higher excited states.

Another application of VQAs is to simulate the dynamical evolution of a quantum mechanical system described by some Hamiltonian H. Here, the general motivation to exploit a VQA is similar to before: other quantum algorithms such as the Trotter-Suzuki product formula [97] require deep quantum circuits, which are not suitable to be executed on NISQ devices. The specific VQAs for this task are based on different ideas [23], one example is an iterative approach where a PQC creates a state $|\psi(\vec{\theta})\rangle$ which is evolved through time by the evolution of $\vec{\theta}$ [28, 29].

A further application of VQAs that is concerned with quantum mechanical properties is that of *quantum autoencoders* to denoise [98, 99] or compress [100–102] quantum data. One of the present goals in [98] is to denoise entangled states, which are an important resource for quantum computing but not easily prepared experimentally. This is achieved by training the QNN from [24] in a supervised manner to process mixed input states ρ_{in} such that they are close to a noiseless pure reference state $|\psi^{\text{ref}}\rangle$. To prevent the QNN from just copying the input state to the output, a bottleneck within the QNN is introduced that forces the extraction of relevant information from the input state. By following this procedure, they succeeded in denoising Greenberger-Horne-Zeilinger states affected by spin-flip errors and random unitary noise.

One more application of VQAs that is well known from classical machine learning is the task of classification [25, 103, 104]. This is usually a supervised learning problem, where the training data is given in many pairs of an input state x and a corresponding class label y. [25] claims to have found numerical evidence that their QNN architecture from Equation (4.4) achieves a smaller model size than classical models. As non-linearity in classification models is known to be important in the classical regime, [103] realises the encoding of inputs into quantum states via non-linear feature maps and [105] discusses the use of quantum-enhanced feature spaces in general.

Finally, VQAs have also been successfully exploited to learn random unitary transformations $\mathcal{U}$ [24, 78, 106], especially as a general test of a model's computational possibilities. This is always done in a supervised manner where the training data is given in pairs of an input state $|\psi_{\text{in}}\rangle$ and an output state $|\psi_{\text{out}}\rangle = \mathcal{U}|\psi_{\text{in}}\rangle$. The exact procedure from [24] to employ a QNN for learning a unitary transformation is reviewed thoroughly in Section 4.2. As a demonstration of entanglement-enhanced machine learning, [106] has shown that a single entangled training data pair can be sufficient to learn the unitary $\mathcal{U}$. In [78], it is investigated which complexity level of the QAOA is necessary to learn an arbitrary d-dimensional unitary.

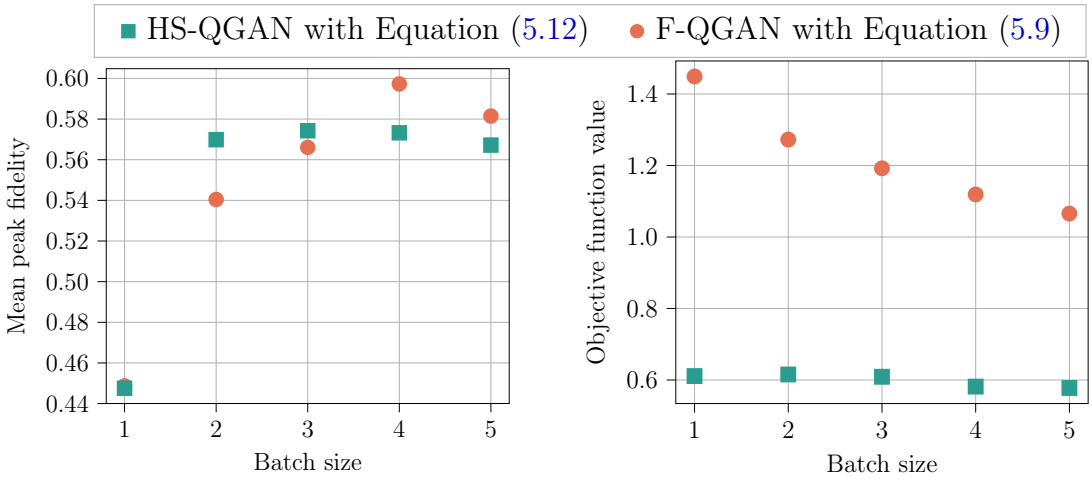
Appendix B

Further numerical results for QGANs

Analysis of different batch sizes

Another hyper-parameter of the QGAN algorithm is the batch size m, which determines how many training states are used for training the generator and the discriminator at each epoch. By default, the batch size is chosen such that all N_T training states are used during each epoch. However, this might be unnecessary for the training's success, and a smaller batch size $m < N_T$ could decrease the computing time. The setting for this analysis is an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^5$ with 5 randomly sampled 4-dimensional pure states $|\phi_k^T\rangle$. A 1-2^+ generator and a 2-1 discriminator are used for both the F-QGAN and the H-QGAN algorithm.

Figure B.1 shows the mean peak fidelities and the objective function values for batch sizes $m = 1, \ldots, 5$ averaged over 12 runs. For both QGAN algorithms, the peak fidelity is the worst for a batch size of $m = 1$. This might be a result of high training fluctuations, i.e., the discriminator learns to accept another random training state each epoch such that the generator fluctuates between specific states instead of learning the features of the complete training set. The F-QGAN algorithm reaches the best peak fidelity for a large batch size $m = 4$ which interestingly is slightly smaller than the complete training set with $N_T = 5$ and corresponds to the dimension of the training states. However, this finding is relativised by the decreasing objective function value for larger batch size as this corresponds to a better generator in relation to the discriminator. The HS-QGAN algorithm shows a similar performance for all batch sizes $2 \leq m \leq 5$ which indicates the possibility to use the HS-QGAN with a rather small batch size to immensely decrease the computing time.



(a) The mean peak fidelity is averaged over all generated states from the last 20 training epochs. It quantifies the generator's capability to produce states close to at least one of the training states.

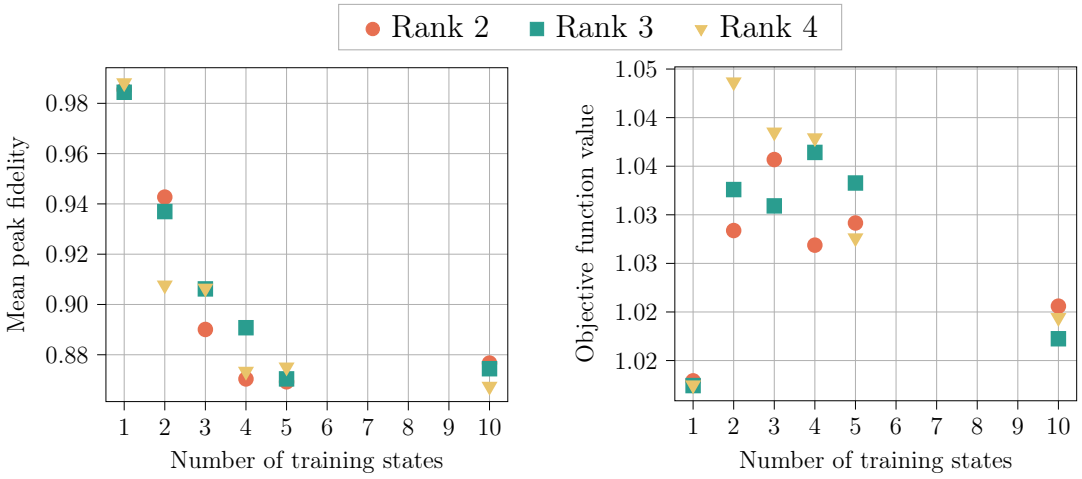
(b) The objective function is averaged over the last 20 training epochs. The F-QGAN's desired equilibrium is at 1, the HS-QGAN's is at 0. A larger value shows that the discriminator outperforms the generator.

Figure B.1: Analysis of different batch sizes. Both QGAN algorithms are used with a 1-2^+ generator DQNN and a 2-1 discriminator DQNN to learn an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^5$ with 5 randomly sampled 4-dimensional pure states $|\phi_k^T\rangle$. Different batch sizes $m = 1,\ldots,5$ are used. The results are averaged over 12 runs with 250 epochs, respectively.

Different types of 4-dimensional training sets

Similar to the analysis of different training state ranks with 8-dimensional training states in Section 5.3, the analysis can be performed for training sets with 4-dimensional training states. Here, the range of interesting ranks is rank $= 2,\ldots,4$ as the minimum non-trivial training set is given by rank $= 2$ and the full Hilbert space is obtained with rank $= 4$. However, it must be remembered that the procedure of generating the training set can usually not fill the complete Hilbert space.

The numerical results averaged over 20 runs for an F-QGAN using a 1-2^+ generator DQNN and a 2-1 discriminator DQNN are shown in figure B.2. As opposed to the 8-dimensional training sets, the different ranks do not significantly affect the performance in terms of the peak fidelity. This may be due to the smaller Hilbert space.



(a) The mean peak fidelity is averaged over all generated states from the last 20 training epochs. It quantifies the generator's capability to produce states close to at least one of the training states.

(b) The objective function is averaged over the last 20 training epochs. The F-QGAN's desired equilibrium is at 1, the HS-QGAN's is at 0. A larger value shows that the discriminator outperforms the generator.

Figure B.2: Different ranks for 4-dimensional training states. The F-QGAN algorithm is used with a 1-2^+ generator DQNN and a 2-1 discriminator DQNN to learn an unlabelled training set $\mathcal{T} = \{|\phi_k^T\rangle\}_{k=1}^{N_T}$ with 4-dimensional pure states $|\phi_k^T\rangle$ of ranks 2 to 4. The results are averaged over 20 runs with 400 epochs, respectively.

List of Figures

List of Tables

Acknowledgements

First of all I thank Tobias J. Osborne for introducing me to the exciting research field of quantum machine learning and guiding me through a year full of valuable lessons, both on a scientific and a human level. I am especially thankful to him for providing the possibility to publish my first scientific paper as part of a wonderful team and to all his tips that were beyond the scope of the paper. Finally, I would like to thank him for offering me and accepting various paths for further cooperation and my scientific orientation.

I would like to thank Reinhard F. Werner for his priceless honesty and humour during the weekly group meetings. Especially significant to my further scientific life will be his experience and advice concerning scientific publishing.

I would like to thank Kerstin Beer for letting me be part of her fascinating research and creating an environment for discussions and questions that was always pleasant and helpful.

It is highly important to me to thank my friend and fellow student Christian Struckmann with whom I shared an exciting and sometimes hard path through five years of university concluding in a close and fruitful cooperation during the work for this thesis. I will never forget our exhausting library sessions, the shared coffees, and the many problems solved together. While I am at it, I would like to equally thank Simon Bittner as a companion and friend for countless shared stories.

In the end, I would like to thank my partner for supporting me through all the past years and my parents and brothers for being my safe harbour throughout my whole life.