Quantum machine learning: on the design, trainability and noise-robustness of near-term algorithms

Skolik, A.

Quantum machine learning: On the design, trainability and noise-robustness of near-term algorithms

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr.ir. H. Bijl,
volgens besluit van het college voor promoties
te verdedigen op donderdag 7 december 2023
klokke 10.00 uur

door

Andrea Skolik

geboren te Karlsruhe, Duitsland
in 1988

Promotores:	Prof. Dr. V. Dunjko	
	Prof. Dr. T. H. W. Bäck	
Promotiecommissie:	Prof. Dr. K. J. Batenburg	
	Prof. Dr. A. Plaat	
	Prof. Dr. C. Macchiavello	(University of Pavia)
	Prof. Dr. M. J. Hartmann	(FAU Erlangen)
	Dr. S. Feld	(TU Delft)

This research is financially supported by the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 828826, the German Ministry for Education and Research (BMB+F) in the project QAI2-Q-KIS under grant 13N15587, and Volkswagen AG. The cover has been designed using assets from Freepik.com.

Abstract

The past decade has brought on tremendous progress in the quest to build a quantum computer, and cloud access to first devices is starting to get available. Although these devices are still very limited in terms of the number of qubits and coherence times, their emergence has focused attention on the question of how these near-term devices can be of value. A specific type of hybrid quantum-classical algorithms, namely variational quantum algorithms, has stood out as a candidate for providing value in these early stages of quantum computing. Machine learning is believed to be a field where variational quantum algorithms can be beneficial, and they have been studied in the context of various learning tasks. Theoretical guarantees are hard to obtain for these algorithms due to their heuristic nature, so progress in this field will heavily rely on empirical discovery and evaluation of algorithms. However, many fundamental questions about successfully running these types of algorithms are still open, like how to design individual components of the quantum circuits that are used, and how to avoid pitfalls in their training. In this thesis, we study various aspects of variational quantum algorithms for machine learning, with a focus on reinforcement learning. We study the steps of the whole pipeline in training a variational quantum machine learning model on near term devices, starting at the question of how to encode classical data and read out information from a quantum circuit, and how to design the circuit itself in a problem-tailored manner. We address the question of how the classical optimization routine in this model can be tailored to quantum-specific issues in optimization landscapes, as well as how various noise sources present on quantum hardware affect the training of these algorithms. With the above, this thesis aims at contributing to the knowledge of how to train variational quantum machine learning models, in order to foster further investigation of these types of algorithms once high-performance quantum hardware will become available.

Publications

This thesis is partially based on content that was previously published by the author and collaborators in the following manuscripts, with author contributions as listed below.

1. **Layerwise learning for quantum neural networks**
   Andrea Skolik, Jarrod R. McClean, Masoud Mohseni, Patrick van der Smagt, Martin Leib
   Quantum Machine Intelligence 3.1 (2021): 1-11

   AS conceived the idea for this work and the layerwise learning algorithm. AS and JRM performed the numerical simulations. JRM defined the cost model for measuring the performance of the studied models. AS and JRM wrote the first draft of the manuscript, all authors were involved in the final editing. MM, PS, and ML supervised the work.

2. **Quantum agents in the Gym: a variational quantum algorithm for deep Q-learning**
   Andrea Skolik, Sofiene Jerbi, Vedran Dunjko
   Quantum 6 (2022): 720

   AS and VD conceived the idea for this work. AS performed the numerical simulations, with the help of SJ. AS, SJ, and VD worked out theoretical separation between quantum and classical learners. AS wrote the manuscript, all authors were involved in the final editing. VD supervised the work.

3. **Equivariant quantum circuits for learning on weighted graphs**
   Andrea Skolik, Michele Cattelan, Sheir Yarkoni, Thomas Bäck, Vedran Dunjko
   npj Quantum Information 9 (1), 1-15

   AS conceived the idea for this work. AS and VD conducted theoretical analysis of the proposed ansatz. AS and MC performed numerical simulations and the analysis of their results. SY created the data set used in this work. AS, MC and SY wrote the first draft of the manuscript; all authors contributed to editing the final manuscript. VD and TB supervised the project.

4. **Robustness of variational quantum reinforcement learning under hardware errors**
   Andrea Skolik, Stefano Mangini, Thomas Bäck, Chiara Macchiavello, Vedran Dunjko
   EPJ Quantum Technology 10 (1), 1-43

   AS conceived the idea for this work and conducted the numerical experiments. SM performed analytical study on the effect of Gaussian noise and provided decoherence noise model. AS and VD proposed shot allocation algorithm. AS and SM wrote the first version of the manuscript, all authors contributed to the final editing.

The following publications were co-authored during the course of the PhD and are not included in this thesis.

5. **TensorFlow Quantum: A Software Framework for Quantum Machine Learning**
   Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, ..., Andrea Skolik, ..., Alan Ho, Hartmut Neven, Masoud Mohseni
   arXiv preprint arXiv:2003.02989

6. **Quantum approximate optimization of non-planar graph problems on a planar superconducting processor**
   Matthew P. Harrigan, Kevin J. Sung, Matthew Neeley, Kevin J. Satzinger, ..., Andrea Skolik, ..., Erik Lucero, Edward Farhi, Ryan Babbush
   Nature Physics 17.3 (2021): 332-336

7. **Beating classical heuristics for the binary paint shop problem with the quantum approximate optimization algorithm**
   Michael Streif, Sheir Yarkoni, Andrea Skolik, Florian Neukart, Martin Leib
   Physical Review A 104.1 (2021): 012403

8. **Hyperparameter optimization of hybrid quantum neural networks for car classification**
   Asel Sagingalieva, Andrii Kurkin, Artem Melnikov, Daniil Kuhmistrov, Michael Perelshtein, Alexey Melnikov, Andrea Skolik, David von Dollen
   arXiv preprint arXiv:2205.04878

Acknowledgements

Getting a PhD can be a long and hard journey, which is made tremendously more enjoyable by good company. I was blessed with this company over the past years, and would like to especially thank the following people.

First and foremost, I thank my supervisor Vedran Dunjko for being an endless source of support, advice, knowledge, and humor. I am impressed with his dedication to research and teaching, and very grateful for the many things I learned from him about quantum computing and research in general. I am also grateful for the welcoming and inclusive environment in the aQa group, that Vedran helped me to become a part of despite being an external student living in another country during a global pandemic. Even though the pandemic prevented me from spending as much time in Leiden as I would have liked, the time I did spend there I will always keep in good memory thanks to the amazing people at LIACS. I especially want to thank Yash Patel, Simon Marshall and Casper Gyurik for interesting discussions and sharing their insights about various topics, and Marie Kempkes, Adrián Pérez-Salinas, Charles Moussa and Stefano Polla for making the time I got to spend physically in Leiden so pleasant. I also thank Thomas Bäck for being my promotor.

One upside of being an external student is that one gets to be a part of two groups, and I thank the members of the Volkwagen quantum computing team – former and present – for the many fun discussions in- and outside of work, and for the trust and support that I received over the years. In particular I thank Sheir Yarkoni, Gabriele Compostella, Michael Streif, Martin Leib, and Florian Neukart. I am also very grateful for the support I received from Patrick van der Smagt at the beginning of my time at Volkswagen.

The work presented in this thesis is the outcome of collaborations with a number of people beyond the above two groups, whom I want to thank for the great

discussions and everything I learned from them. I thank Jarrod McClean and Masoud Mohseni for the joint work on layerwise learning, and their hospitality during my two visits to the Google Quantum AI team. I also thank Alan Ho for giving me the opportunity to contribute to the early stages of TensorFlow Quantum, and the whole TFQ team for making my life so much easier by implementing this framework. I thank Sofiene Jerbi for providing his insights about quantum reinforcement learning, and for sharing the woes of getting these algorithms to work. I also thank Stefano Mangini for randomly reaching out to me and starting a conversation about reinforcement learning, which eventually lead to a great collaboration, and his positive attitude that made this such a fun project.

Finally, I want to thank Marcus Voigt for embarking on this journey called "life" with me, and for his endless patience, support and love throughout the past decade. Let's start the next adventure!

Contents

CHAPTER 1

Introduction

> *"The history of classical computing teaches us that when hardware becomes available that stimulates and accelerates the development of new algorithms. There are many examples of heuristics that were discovered experimentally, which worked better than theorists could initially explain. We can anticipate that the same thing will happen with quantum computers."*
>
> – John Preskill

Since the first ideas for a quantum computer, a device that uses quantum mechanical effects to process information, were proposed in the 1980s, remarkable progress has been made in the quest to build these types of machines and to find tasks where they outperform their classical counterparts. While the first algorithm with a proven speed-up in a practically relevant task, namely Shor's factoring algorithm that can break a widely used encryption method [1], was already proposed in 1994, the first experimental demonstration of a quantum computer outperforming its classical counterpart was only due in late 2019. In this experiment, a team of researchers at Google and NASA showed that their superconducting quantum hardware could perform a sampling task much faster than any classical supercomputer existing to that date [2]. Just one year later, a similar result on another sampling task was shown on the Chinese photonic quantum device Jiuzhang [3]. In parallel to these experimental advances, theoretical investigation has lead to a multitude of quantum algorithms with guaranteed speed-ups. These algorithms operate under the assumption that one can execute them on a perfect quantum computer, also known as fault-tolerant quantum computing. Among these algorithms are for example Grover's search algorithm, which yields a quadratic speed-up for

unstructured search [4], and an algorithm for solving linear systems of equations that provides an exponential speed-up under certain conditions and in certain cases [5].

The latter has inspired investigation of using quantum computers in the context of machine learning, as the algorithm from [5] can be used to exponentially speed up some of the matrix operations underlying many classical machine learning algorithms [6, 7]. However, these speed-ups come with a set of caveats: there are specific constraints on the structure of the matrices, as well as the assumption that the classical data is stored in a so-called quantum random access memory (qRAM) [8], a memory that gives the algorithm access to data in quantum superposition. As there are numerous technical challenges in building a qRAM, as well as subtleties in when exactly these types of speed-ups exist [9, 10, 11], it is not believed that algorithms based on speeding up solving linear systems of equations will lead to a practical advantage over classical computers in the near future.

In recent years, an alternative to fault-tolerant algorithms has emerged in the so-called noisy intermediate-scale quantum (NISQ) setting, where algorithms for near-term quantum computers are studied [12, 13]. These algorithms are designed with the expectation that they are run on noisy quantum computers with a limited number of qubits and no error correction. One of the most popular approaches in this regime are variational quantum algorithms [14]. In a variational quantum algorithm (VQA), a classical and quantum part work in tandem, where the quantum part of the algorithms is defined in terms of a parametrized quantum circuit (PQC). The parameters of this circuit are then optimized by a classical subroutine in order to solve a certain task. These types of algorithms have been applied to various tasks such as optimization [15, 16, 17], chemistry and simulation [18, 19, 20, 21, 22], and machine learning [23, 24, 25, 26, 27, 28].

While variational algorithms are widely applicable in principle, theoretical statements about their performance and potential speed-ups are difficult to obtain. Especially in a machine learning context, giving rigorous statements in a variational setting is hard, as the performance of the algorithm strongly depends on several factors, like the classical optimizer and the learning task at hand. This resembles the situation in classical machine learning, where first theories of neural networks were developed in the 1940ies, however, the true capabilities of these models only became clear around seventy years later when the increase in computational resources enabled training of large-scale neural networks to perform practically

relevant tasks. While recent breakthroughs in classical machine learning, like beating a grandmaster in the game of Go [29], predicting the structure of proteins [30, 31], or turning natural language prompts into stunning artwork [32], were only possible because current hardware makes it feasible to train models with billions of parameters, these works were built on the basis of a firm understanding on how to design trainable and performant neural networks.

Variational quantum machine learning models are often described as the quantum analog of classical neural networks due to the similarity in their training procedure, and are therefore also referred to as quantum neural networks. Unlike for their classical counterparts however, there are still numerous open questions about how to design trainable and performant quantum neural networks. Examples of this include the questions of how to encode classical data into a quantum model, how to structure the operations that are used to implement models, and how to avoid pitfalls in the trainability of these models that are unique to the quantum setting. Assuming that similarly to the history of classical machine learning, the development of more performant quantum hardware will facilitate large-scale empirical studies on the usefulness of variational quantum machine learning, it is of key importance to build an understanding of how these models can be trained successfully. This thesis aims to contribute to this understanding by studying various aspects of training variational quantum machine learning models.

We start by giving a basic introduction to the topics of quantum computing, machine learning, and their intersection in Chapters 2 and 3, respectively. In Chapter 4, we study how a fundamental issue in the training of variational quantum circuits, namely barren plateaus in the training landscapes, can be addressed by the classical training algorithm to aid scaling up the size of quantum models. To this end, we provide a training scheme that alleviates the problem of barren plateaus for specific cases and compare it to standard training procedures in the existing literature. While this type of training procedure can in principle be used for arbitrary types of machine learning, we focus our attention on a specific type of learning in subsequent chapters, namely on reinforcement learning (RL). First, we study in Chapter 5 how the architectural choices made for a PQC-based quantum agent influence its performance on two classical benchmark tasks from RL literature, where we specifically consider the question of encoding data into, and reading information out of the quantum model. In addition, we establish a theoretical separation between classical and quantum models for the specific type

of RL algorithm that we use, and also perform an in-depth empirical comparison of the quantum model developed in our work to a classical neural network that performs the same task. In addition to the questions of how to encode data and read out information from a PQC, the third key question in the performance of a variational quantum machine learning model is how to design the structure of the circuit itself, also referred to as the ansatz. For this reason, we move on to study this question in Chapter 6 and introduce an ansatz that is tailored to a specific type of input data, namely to weighted graphs. To do this, we take inspiration from the classical field of geometric deep learning, and design a PQC that preserves an important symmetry in graph-based input data. We analytically study the expressivity of this type of circuit, and then go on to numerically compare it to ansatzes that are not tailored to the specific training data at hand. Finally, another important consideration in the study of algorithms for the NISQ era is how the given learning algorithms and models are influenced by quantum hardware-induced noise. In Chapter 7, we study this for two of the variational RL paradigms from recent literature. We investigate analytically and numerically how various types of errors, namely coherent, incoherent, and measurement-based errors, affect the training performance of variational RL algorithms and the robustness of the learned policies. In particular, this study includes an evaluation of the performance of the models we introduced in Chapter 5 and Chapter 6 under various types of noise that are expected to be present on near-term hardware.

With the above, this thesis aims to contribute to building a foundation of knowledge about how to successfully train variational quantum machine learning models, in the hope that similarly to classical machine learning, this knowledge will one day, when quantum hardware has sufficiently matured, aid demonstrations of the practical usefulness of these types of algorithms.

CHAPTER 2

Quantum computing

Quantum computing is a rapidly evolving field that has the potential to change the way we solve a number of complex computational problems. In recent years, there has been significant progress in the development of quantum computers, with researchers and companies around the world working to build these types of machines. In terms of current hardware implementations, there are two main approaches to quantum computing: quantum annealing and gate-based quantum computing. A quantum annealer is a special-purpose device tailored to solve combinatorial optimization problems, based on the idea to slowly evolve a system until it reaches its lowest-energy state, which represents the optimal solution to a given problem. Gate-based quantum computing, on the other hand, involves the use of quantum gates to manipulate qubits, the basic units of quantum information. This approach is more flexible and has the potential to perform a wider range of calculations than quantum annealing, but it is also more difficult to implement in practice due to the need to maintain the delicate quantum states of the qubits. In this thesis, we focus on the latter paradigm of quantum computing, and this chapter provides an introduction to the most important concepts in the gate-based formalism, and the specific type of algorithms we study in later chapters.

2.1 Gate model quantum computing

Quantum computers are devices that harness quantum mechanical effects to process information. In order to utilize these types of effects, one has to define a quantum system to perform operations on. A simple and broadly used approach to this are two-level systems called qubits, which form the building blocks of most quantum algorithms. Mathematically, a qubit can be represented as the quantum state

$|\psi\rangle \in \mathbb{C}^2$ with amplitudes α and β,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \tag{2.1}$$

where $|\alpha|^2 + |\beta|^2 = 1$. The squared modulus of α and β give us the probability of measuring zero and one, respectively. We henceforth adopt bra-ket notation, where the *bra* $\langle\psi| = (\alpha^*, \beta^*)$ denotes the complex conjugate of the state $|\psi\rangle$ (*ket*), $\langle\psi|\psi\rangle$ denotes an inner product, and $|\psi\rangle\langle\psi|$ denotes an outer product. The vectors $|0\rangle$ and $|1\rangle$ form an orthonormal basis of the Hilbert space $\mathbb{C}^2$, and are therefore referred to as *basis states*. Technically, any two orthonormal states can be used as basis states for the vector space of the qubit, however, the two states above are the most common and are called the *computational basis*. These basis states are the states that we can observe in the classical world, while the linear combination of basis states in Equation (2.1) is called a *superposition*, where the coefficients α and β define the probabilities with which each of the basis states can be observed. Unlike classical probabilities, those coefficients are complex-valued and can therefore also be negative. This means that they can either add up or cancel each other out, and this constructive and destructive *interference* of amplitudes plays an important role in many quantum algorithms.

In order to manipulate the state of a qubit, one uses unitary operators which are referred to as *quantum gates*. A commonly used set of gates that can be used to implement arbitrary unitary transformations on a single qubit are the so-called Pauli operators,

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ \sigma_y = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix},\ \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.2}$$

If one, for example, wants to implement a bitflip operation on a qubit, one can do this by applying the σ_x operator as follows,

$$\sigma_x\,|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |1\rangle\,. \tag{2.3}$$

Now, if we consider not just one but multiple qubits, the above can easily be extended by forming tensor products of the kets of a number of qubits. Let us take for example the state over the qubits $|\psi_A\rangle$, $|\psi_B\rangle$ and $|\psi_C\rangle$, then the complete state of this *qubit register* is

$$|\psi_{ABC}\rangle = |\psi_A\rangle \otimes |\psi_B\rangle \otimes |\psi_C\rangle\,, \tag{2.4}$$

and $|\psi_{ABC}\rangle$ is now an element of the Hilbert space $\mathbb{C}^{2^n}$ for $n = 3$ qubits. When we consider a register of multiple qubits, another important aspect of quantum algorithms can arise: *entanglement* between these qubits. Intuitively, entanglement means that the state of a quantum system can not simply be described by considering its individual parts. Formally, we call a state entangled if it is not a *separable state*. To understand the notion of a separable state, consider a bipartite quantum system on the Hilbert spaces $\mathcal{H}_A$ with basis $\{|a_i\rangle\}_{i=1}^k$ and $\mathcal{H}_B$ with basis $\{|b_j\rangle\}_{j=1}^l$, and a basis $\{|a_i\rangle \otimes |b_j\rangle\}$ for $\mathcal{H}_A \otimes \mathcal{H}_B$. Any pure state in this composite system can be written as

$$|\psi\rangle_{AB} = \sum_{i,j} c_{i,j}(|a_i\rangle \otimes |b_j\rangle). \tag{2.5}$$

If the state can be written as a simple tensor product of the two subsystems,

$$|\psi\rangle_{AB} = |\psi\rangle_A \otimes |\psi\rangle_B \,, \tag{2.6}$$

it is considered a separable state. Intuitively, this can be understood as a joint probability mass function that is the product of two independent marginals, i.e., $p(x, y) = p(x)p(y)$. However, the type of correlation that is present in non-separable states has no analog in the classical world and is therefore hard to understand intuitively. Entanglement is a crucial ingredient for many quantum algorithms, like the famous prime factorization algorithm by Shor [1].

So far we have discussed how to prepare and manipulate quantum states in the gate-model formalism. The final ingredient required to perform quantum computation is getting classical information out of the device, that is, performing a measurement of a given observable. Going back to the one-qubit example above, we have already discussed that the superposition state shown in Equation (2.1) can not be measured. Instead, we can only measure the basis state $|0\rangle$, which occurs with probability $|\alpha|^2$, and the basis state $|1\rangle$, with probability $|\beta|^2$. More formally, we define measurement observables in terms of a set of operators $\{M_m\}$ on the state space of the quantum system. In this thesis, we consider the special case where all M_m are orthogonal and Hermitian, called a *projective measurement*. The observable M has the spectral decomposition

$$M = \sum_m m P_m, \tag{2.7}$$

with P_m the projector onto the eigenspace of M with eigenvalue m. For the example of a computational basis measurement on one qubit $|\psi\rangle$ we have the measurement operators

$$P_0 = |0\rangle\langle 0|\,, P_1 = |1\rangle\langle 1|\,, \tag{2.8}$$

both with eigenvalue 1, and we get

$$\sum_m P_m^\dagger P_m = P_0^\dagger P_0 + P_1^\dagger P_1 = I, \tag{2.9}$$

where Equation (2.9) shows that this set of projectors satisfies the *completeness relation*, i.e., the set of projectors has to satisfy the condition that the probabilities of all measurement outcomes sum to one. After measuring outcome m, the state is then

$$|\psi_m\rangle = \frac{P_m\,|\psi\rangle}{\sqrt{p(m)}}, \tag{2.10}$$

where the probability $p(m)$ to measure outcome m is

$$p(m) = \langle\psi|\,P_m\,|\psi\rangle\,. \tag{2.11}$$

The above statement that the measurement outcome will be one of the eigenvalues of M and that the probability of measuring eigenvalue m is given by Equation (2.11) is also known as the *Born rule*. Once the observable is measured, the quantum state *collapses* and all information about its previous state is lost. Subsequent measurements of the resulting state will then always yield the same output.

Another common observable, which we will also use in this thesis, is the observable Z, also referred to as a *measurement in the Z basis*, with basis states $|0\rangle$ and $|1\rangle$, and eigenvalues 1 and -1, respectively,

$$Z = 1\,|0\rangle\langle 0| - 1\,|1\rangle\langle 1| = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.12}$$

After defining the main ingredients of quantum algorithms above, like qubits, quantum gates, superposition, interference and entanglement, we can start writing down our own algorithms. The formalism most commonly used for this, and the one we also use in this work, are quantum circuits. In a quantum circuit, qubits are represented as *wires*, horizontal lines read from left to right. Quantum gates are then placed on these wires to indicate the operations performed on each of the qubits in the register. This formalism can be used to write down arbitrary

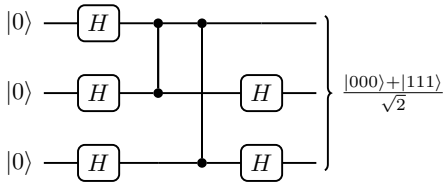


Figure 2.1: Example of a quantum circuit diagram depicting a circuit to create a Greenberger-Horne-Zeilinger (GHZ) state, an important type of entangled state. In this 3-qubit circuit, the GHZ state is prepared by applying Hadamard gates (H) and controlled phase gates (vertical lines).

quantum algorithms, and a simple example of a circuit diagram can be seen in Figure 2.1.

With the above, we are equipped to study arbitrary quantum algorithms, like the well-known prime factorization algorithm [1], or the quantum algorithm for solving linear systems of equations [5] mentioned above. In this thesis, however, we are interested in a special type of quantum algorithm tailored for near-term devices, which we will introduce in the following section.

2.2 Noisy intermediate-scale quantum computing

In Chapter 1, we already outlined the distinction between fault-tolerant algorithms, that are designed with perfect, large-scale quantum computers in mind, and so called noisy intermediate-scale quantum (NISQ) algorithms, which are more suitable for the error-prone and small quantum computers that we expect to have access to in the coming few decades. The term NISQ was coined by John Preskill in a keynote talk and accompanying article [12], and he defines these types of devices as quantum computers with around fifty to at most a few hundred qubits, which are subject to noise and which we have only imperfect control over. Apart from the number of qubits, he also emphasizes that the quality of qubits and how precisely operations can be performed on them plays a crucial role. This means that NISQ algorithms are not only restricted in the number of qubits they can use, but also in the *depth* and the total number of gates of the quantum circuits that are used. Additionally, as error correction techniques that are developed to aid

fault-tolerant quantum computation require a high overhead of additional qubits, NISQ devices are assumed to operate without error correction in order to fully utilize the small number of qubits that are available. Alongside all of the above, device-specific limitations like qubit connectivity and native gate sets have to be taken into account as well.

With these constraints, the algorithms that can successfully be run in the NISQ era are severely limited. In particular, algorithms of the type as the prime factorization and linear equation solving algorithms discussed in previous chapters [1, 5], that require the execution of a pre-defined and rigid gate sequence, are out of reach for these types of devices. However, an interesting class of algorithms specifically tailored to the limitations of these devices has emerged in the past few years: variational quantum algorithms (VQAs) [14]. VQAs are hybrid quantum-classical algorithms that outsource a large part of the heavy lifting to a classical computer. The basis for a VQA is a parametrized quantum circuit (PQC) that is run on the quantum device. Based on measurements of this quantum circuit and a given objective function, a classical optimizer computes updates of the circuit parameters. This procedure is repeated in an iterative fashion, until the circuit output matches the desired output for a given task. This approach is quite different from the one taken for designing the fault-tolerant algorithms we have discussed before. Instead of specifying a fixed algorithm in form of a quantum circuit that relies on the execution of a precise sequence of gates, in a VQA, designing this circuit is left more or less to the classical optimization routine. In general, only the structure of the parametrized gates is given in advance, and this can be tailored to the specific constraints of a certain device, like the available gate set or connections between qubits. In the following sections, we will address how the parameter optimization scheme in these types of models works, and in which areas they have shown to be promising to apply.

2.2.1 Variational quantum algorithms

A VQA consists of two components: the parametrized quantum circuit, also called *ansatz*, and the classical optimization routine that performs the parameter updates, as can be seen in Figure 2.2. In general, the structure of the ansatz can be chosen quite flexibly, and we denote a unitary parametrized by $\boldsymbol{\theta}$ as $U(\boldsymbol{\theta})$. Usually, the ansatz is structured in layers, and we write the unitary that represents a PQC of

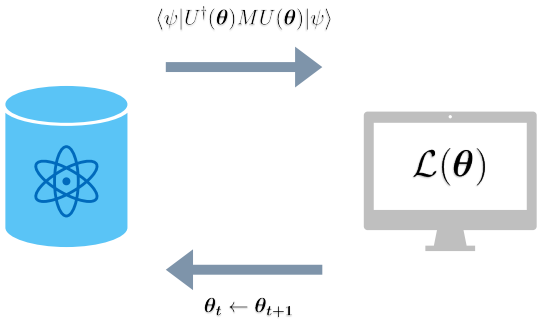


Figure 2.2: Depiction of a variational quantum-classical algorithm. The quantum computer on the left implements a quantum circuit parametrized by $\boldsymbol{\theta}$, and outputs expectation values of this circuit given an observable M. Based on this, the classical computer on the right computes a loss function $\mathcal{L}(\boldsymbol{\theta})$, and the updated parameters $\boldsymbol{\theta_{t+1}}$, which are fed into the quantum circuit for the next iteration. This process is repeated until the loss function reaches a desired value.

depth L as

$$U_L(\boldsymbol{\theta}) = \prod_{i=1}^{L} U_i(\boldsymbol{\theta}_i). \tag{2.13}$$

In the case where a PQC is used in a machine learning setting, there may also be additional parameters that serve to encode the training data into the circuit, and we will discuss these types of circuits in more detail in later chapters. For simplicity, we will omit these parameters in this chapter, as they do not influence the optimization procedure directly.

In order to optimize the parameters of the PQC given a certain task, a classical optimization routine is used. There is a wealth of options to choose from in the classical optimization literature, and indeed numerous different techniques have been explored for the optimization of PQCs [33, 34, 35, 36, 37]. One of the most popular approaches are gradient-based methods, which are the state-of-the-art optimizers used for classical neural networks. In their most basic form, called *stochastic gradient descent*, the parameters $\boldsymbol{\theta}$ of a function $f(\boldsymbol{\theta})$ are updated according to the following rule,

$$\theta_i \leftarrow \theta_i - \eta \nabla f(\theta_i), \tag{2.14}$$

where η is a step size that determines the magnitude of each update step, also called *learning rate* in the machine learning literature. There are many different versions of these types of algorithms, often with elaborate schedules to fine-tune the step size for various phases of the optimization. The Adam optimizer, for example, has additional momentum terms that tune the step size according to the steepness of the optimization landscape [38]. It depends on the given task which flavor of gradient descent is best suited, however, what they all have in common is that they require computation of partial derivatives of the function to be optimized.

2.2.1.1 Computing gradients

A simple possibility to compute gradients in PQCs is to use the *finite difference* method. With this, the gradient of an arbitrary function $f(\theta)$ can be approximated to precision $\mathcal{O}(\epsilon)$ by

$$\frac{df}{d\theta} \approx \frac{f(\theta+\epsilon) - f(\theta)}{\epsilon}. \tag{2.15}$$

With this method, the number of circuit evaluations required to compute the gradient of a PQC scales linearly with the number of parameters used in the circuit. However, this comes at the cost of only obtaining an approximation of the gradient, as well as introducing another hyperparameter, namely ϵ, to the optimization routine. Furthermore, in the case of noisy quantum hardware with imprecise control, and circuit evaluations based on a limited number of measurements, there are additional limitations on how small ϵ can be chosen in this setting.

This lack of precision in computing gradients will increase their variance during training and can therefore negatively impact the optimization routine. To alleviate the issue of only computing an ϵ-approximation of the gradients, one can also compute exact gradients for PQCs by using the so-called *parameter-shift rule* [39, 33, 40]. Consider a parametrized gate $U_G(\theta_i) = e^{-ia\theta_i G}$ with generator G that is a Hermitian linear operator, trainable parameter θ_i and a real constant a, that acts within an arbitrary unitary $U(\boldsymbol{\theta})$. If G has at most two distinct eigenvalues e_0 and e_1, the partial derivative of $\langle U(\boldsymbol{\theta})\rangle$ w.r.t. θ_i can be written as

$$\frac{d}{d\theta_i}\langle U(\boldsymbol{\theta})\rangle = r\left(\langle U(\boldsymbol{\theta}+\boldsymbol{\Delta\theta_i})\rangle - \langle U(\boldsymbol{\theta}-\boldsymbol{\Delta\theta_i})\rangle\right), \tag{2.16}$$

where $\langle U\rangle$ denotes the expectation value of U acting on some initial state and under measuring a given observable, and $\boldsymbol{\Delta\theta_i}$ is a vector of the same length as

$\boldsymbol{\theta}$ that is $\frac{\pi}{4r}$ at the i-th position, and zero everywhere else. In other words, the partial derivative of a PQC can be computed by the difference of two evaluations of the same PQC, with the parameter that is considered for the derivative shifted by $\frac{\pi}{4r}$ and $-\frac{\pi}{4r}$, respectively. The factor r depends on the eigenvalues of the generator as $r = \frac{a}{2}(e_1 - e_0)$ [40], and $r = \frac{1}{2}$ for the Pauli gates we commonly use in this thesis [39]. In the case that one parameter is shared between a number of gates, the parameter-shift rule has to be applied for each of the gates individually, and the derivative is then computed according to the product rule. The authors of [33] also show how the above can be extended for generators with arbitrary eigenvalues, at the cost of introducing one ancilla qubit, and for Gaussian gates in continuous-variable quantum computing. The above parameter-shift rule enables computation of exact derivatives of arbitrary quantum circuits, at the cost of two circuit evaluations per parameterized gate in each update step. These gradients can then be used to perform any gradient-based update routine, like the stochastic gradient descent method described above. The downside of the parameter-shift rule is that the number of circuit evaluations required to compute gradients scales linearly with the number of *parametrized gates*, instead of the *number of parameters* itself as in the finite difference method above.

As we have seen above, computing gradients for PQCs is relatively straightforward. However, there are a number of challenges in the optimization of PQC parameters, as we will describe in the next section.

2.2.1.2 Challenges in the optimization of PQCs

In Section 2.2.1.1, we described how the parameters in a quantum circuit can be optimized by using tools from the classical optimization literature. However, it turns out that there are a number of difficulties when optimization is done in a quantum landscape. A first fundamental issue in the optimization of PQCs was described in [41], where the authors introduce the notion of *barren plateaus*, vast saddle points in quantum circuit training landscapes where first and higher order derivatives vanish. These plateaus result from basic features of the geometry of high-dimensional spaces.

The authors of [41] examine the gradients for quantum circuit training in a model known as random PQCs. For this model, parameter updates for gradient-based optimization are calculated based on the expectation value of an observable measured on a state produced by a parametrized circuit, and the circuits are drawn

from a random distribution. The distribution of circuits amounts to choosing a set of gates uniformly at random, where some of the gates have continuous parameters amenable to differentiation. The authors of [41] show that a sufficient, but not necessary, condition for the vanishing of gradients with high probability is that for any arbitrary division of the circuit into two pieces, the two pieces are statistically independent, and that one of them approximately matches the fully random distribution up to the second moment, or in other words forms an approximate 2-design. Formally, a unitary t-design X is defined as,

$$\frac{1}{|X|}\sum_{U\in X} U^{\otimes t}\otimes (U^*)^{\otimes t} = \int_{U(d)} U^{\otimes t}\otimes (U^*)^{\otimes t} dU, \tag{2.17}$$

where the unitary t-design is an ensemble of unitaries that matches the fully random distribution of unitaries up to the t-th moment. The fully random distribution of unitaries is given by the *Haar measure*, which assigns a translation invariant volume on the sphere. To understand the above, it is important to note that in order to sample unitaries of a given size *uniformly* at random from the full Hilbert space, it is not enough to choose an appropriate parametrization, e.g., parametrized unitaries that represent arbitrary rotations, and then sample the *parameters* uniformly at random. This is because those unitaries act on a high-dimensional sphere, where differences between parameters are not proportional to the differences between the resulting points on the sphere, as these differences depend on their position on the sphere. The Haar measure assigns a volume on the sphere that is invariant to the position, and therefore sampling uniformly at random according to the Haar measure allows to sample unitaries uniformly at random from the full Hilbert space. Additionally, a t-design over unitaries generates a t-design of states in the given Hilbert space. While executing a circuit that implements such a Haar-random unitary takes time exponential in the number of qubits, there are efficient techniques to produce circuits that approximate the first and second moment of the Haar distribution, which is formalized by the notion of 2-designs. It has been shown in [42] that random PQCs form approximate 2-designs once they reach a certain depth, and the authors of [41] show how even a modest number of qubits and layers of random gates is enough for this. The depth of a quantum circuit required to reach this regime of barren plateaus depends on the number of qubits and allowed connectivity. It is thought that the depth required to reach a 2-design scales roughly as $\tilde{O}(n^{1/D})$, converging to a logarithmic required depth in the all-to-all connectivity limit [43], where D is the dimension of

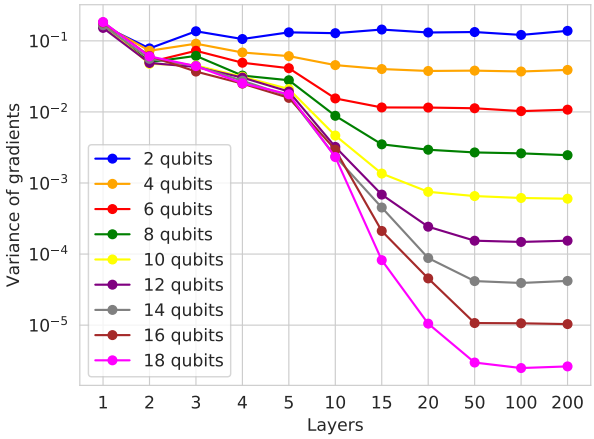


Figure 2.3: Concentration of variance of gradients of the expectation value of the readout qubit. For random parametric quantum circuits, as circuits of different sizes converge to a 2-design, gradient values necessary for training vanish with increasing number of qubits and layers.

the connectivity of the device, e.g., $D = 2$ for a square lattice, and n is the number of qubits.

As an approximation of a 2-design, a particular class of random circuits that were studied numerically in [41] were those with some discrete structure of gates determined by an underlying geometry (e.g., 1D line, 2D array, or all-to-all), and single qubit gates with a continuous parameter rotating around a randomly chosen axis. This assigns a parameter θ_i to each of these gates. To sample from the distribution of random circuits, each angle θ_i was drawn from the uniform distribution $\theta_i \in \mathcal{U}(0, 2\pi)$. Due to concentration of measure effects as described in [41], random PQCs with different sets of parameters will produce very similar outputs and their variance vanishes for sufficiently large circuits, i.e., those that reach approximate 2-designs, as shown in Figure 2.3. Each data point in the figure is calculated over 1000 randomly generated circuits with all-to-all connectivity in each layer, and an initial layer of Hadamard gates on each qubit, to avoid a bias of gradients with respect to the circuit structure caused by initializing in an all-zero state. Note that the average value of the gradient here is zero in all cases.

This represents a generic statement about the volume of quantum space for such

a circuit where one expects to find trivial values for observables. In other words, sufficiently deep, arbitrary random PQCs will produce very similar expectation values regardless of the set of individual parameters. Consequently, the partial derivatives of an objective function based on expectation values of a random PQC will have extremely small mean and variance. These properties make them essentially untrainable for gradient-based or related local optimization methods on quantum devices, as the training landscapes in these scenarios are characterized by large regions where the gradient is almost zero, but which do not correspond to a local or global minimum of the objective function. This decay of the variance of gradients is not only detrimental to parameter optimization, but more generally hinders extraction of meaningful values from quantum hardware, especially on near-term processors that are subject to noise.

Based on the above results by [41], trainability issues in PQCs have been studied extensively in the past few years. While the above results illustrate how with increasing depth, a PQC gets closer to the 2-design regime and is therefore more susceptible to barren plateaus, it was shown in [44, 45] that barren plateaus can also be present in shallow circuits when the cost function is global, in the sense that it is computed based on states in exponentially large Hilbert spaces, instead of local, where the cost function depends only on smaller subsets of the qubits in a circuit. It has also been shown that there is a relationship between the amount of entanglement in a given circuit and its susceptibility to barren plateaus, where high amounts of entanglement tend to lead to untrainability [46, 47]. Moving away from only considering circuits that are approximate 2-designs, the authors of [48] establish a connection between the expressivity of a PQC and barren plateaus, where the expressivity of a circuit is measured in terms of its distance to a 2-design. Finally, it was shown in [49, 50] that regardless of the specific structure of the ansatz, the noise present on quantum hardware can lead to untrainability issues for circuits where the depth grows at least linearly with the system size. While the above results focus on gradients and their variance, it has also been shown that the barren plateau phenomenon persists for higher-order derivatives [51], as well as gradient-free optimization [52]. The amount of negative results above seems discouraging. However, there has also been a tremendous effort to develop methods to address the above issues. The authors of [53] introduce a non-random parameter initialization strategy, where some parts of the circuit are initialized randomly, and the rest is then chosen so that the whole circuit will act as the identity. This will prevent initialization on a barren plateau. Another approach

that addresses the barren plateau issue by choosing a specific parametrization is that by [54], where a number of gates in a circuit share the same parameter and are therefore correlated. Other approaches focus on the circuit structure itself, instead of the parametrization of gates. The authors of [55] introduce an ansatz that is tailored to a specific type of learning problem, and a subsequent work shows that this problem-related structure prohibits the onset of barren plateaus in this ansatz [56]. In a similar vein, the authors of [57] show that circuits that preserve a certain symmetry are immune to barren plateaus. Another recent approach uses techniques related to shadow tomography to detect and avoid barren plateaus [58]. In Chapter 4 of this thesis, we will introduce another method to address the barren plateau problem, where the circuit is initialized and trained in a way that avoids utilizing the full Hilbert space, and thereby sidesteps going into the 2-design regime.

To summarize, there are a number of fundamental challenges in the training of PQCs which present large hurdles that have to be overcome to train VQAs on a large scale. At the same time, addressing and mitigating these challenges is a prosperous field of research, and recent results provide hope that especially problem-tailored ansatzes will enable successful training of large scale quantum models in the future.

2.2.2 Application areas and outlook

The VQA framework is extremely flexible and can be applied to a wide variety of problems. However, three main application areas of VQAs have gained traction in the past years: combinatorial optimization, quantum chemistry and simulation, and machine learning. For combinatorial optimization, the most commonly used technique is the quantum approximate optimization algorithm (QAOA) [59], where the solutions of an optimization problem are encoded as the eigenstates of a Hamiltonian, and the ground state represents the optimal solution. The parameters of the PQC are then optimized such that the circuit outputs the ground state with highest probability. In the limit of an infinitely deep circuit, this algorithm also has theoretical guarantees to find the ground state. Next to being studied in-depth in the context of the canonical Max-Cut problem [60, 61, 16, 62, 63], the QAOA has also been applied to a number of industrially relevant problems [64, 65, 66, 67], as well as being studied experimentally on a superconducting quantum device [68] and with Rydberg atoms [69].

A similar approach to find ground states of Hamiltonians in a quantum chemistry setting is called the variational quantum eigensolver (VQE) [18, 20]. Here, the goal is still to find the ground state of a given Hamiltonian, but the structure of the ansatz to do this is not given in advance as in the case of the QAOA, but can be chosen depending on the given problem. This approach has also been studied extensively in the past few years, both theoretically [70, 71, 72], as well as experimentally on real hardware [73]. The third application area where VQAs have gained much interest in recent years, and the one we focus on in this thesis, is machine learning. Similarly to the above two examples, there has been a wealth of results in this area in the past years, and we will discuss VQAs in this context in more detail in Section 3.3.

Overall, VQAs seem to be a promising road to demonstrate the usefulness of quantum computers on a task of practical interest. However, this road is not without obstacles. In addition to the challenges we described in Section 2.2.1.2, and even when VQAs turn out to be applicable to large-scale quantum systems, the question remains whether these algorithms are better than their classical counterparts. Due to their variational nature, it is hard to make rigorous statements about any type of quantum advantage for VQAs. Second, even if a task with a potential empirical gain in performance is found, it is hard to directly compare these types of algorithms to their classical analogs. In particular, there is the question which classical algorithm one compares to, and under which criteria. To give an example, it has been empirically observed in a number of studies that PQCs seem to be able to solve certain tasks with fewer parameters than classical neural networks [74, 75, 76, 77]. However, one can not directly proclaim quantum advantage from this fact, as it also has to be taken into account that the computation of gradients is more efficient in the classical setting and can be heavily parallelized, which is not the case in a VQA. And last but not least, while VQAs have been conceived with NISQ devices in mind, they too do suffer from the noise present on these devices. An open question is how one can efficiently combat these types of errors. While a number of error mitigation techniques have been proposed in the past years [78, 79, 80], recent results also show that they come with an exponential sampling overhead [81, 82]. So even if VQAs eventually turn out to be useful for practical tasks in the future, a number of roadblocks still have to be overcome until we get there.

CHAPTER 3

Machine learning

The field of machine learning (ML) is concerned with developing systems that learn to perform certain tasks without being explicitly programmed to do so. The methods used in this field encompass a wide range of techniques, ranging from simple linear regression to more complex approaches like neural networks [83, 84]. How the problem of learning a certain task is addressed can be broadly separated into three branches: supervised learning, unsupervised learning, and reinforcement learning. What all three of these methods have in common is that they are based on a *model*, the trainable part of the algorithm that will later execute the learned task. In the supervised setting, a model is trained based on a set of labeled data samples, to then label samples not seen during training. In unsupervised learning, there are no labels assigned to the training data, but the goal of the algorithm is to infer the underlying structure of the data, e.g., by sorting it into separate clusters. In reinforcement learning, there is no training data per se, but the algorithm learns in a trial-and-error fashion by interacting with an environment.

Depending on which type of learning is performed, one can also choose between a number of different models. In this thesis, we will study ML from the neural network perspective [85], as these types of models are most closely related to the VQAs described in Section 2.2.1. We will first describe neural networks and their training in Section 3.1, and then move on to introduce reinforcement learning in more depth in Section 3.2, which is the learning algorithm we mainly focus on in this thesis. Finally, we outline the intersection of machine learning and quantum computing, referred to as *quantum machine learning*, in Section 3.3.

3.1 Neural networks

3.1.1 Neurons, layers, and backpropagation

In its simplest form, a neural network (NN) consists of layers of artificial neurons that are loosely based on biological neurons, where the output of the k-th neuron, given an input vector $\boldsymbol{x}$ of length m, is of the form

$$f_k(\boldsymbol{x}) = \sigma\left(\sum_{j=0}^{m} w_{kj} x_j\right), \tag{3.1}$$

where σ denotes an *activation function* that serves the purpose of introducing nonlinearities to the NN. The w_{kj} represent trainable weights, which are optimized such that the whole network gives the desired output on a given input, similarly to the parameters $\boldsymbol{\theta}$ in the VQAs we introduced in Section 2.2.1. When NNs are trained with gradient descent, the activation function needs to be differentiable or at least allow the computation of subderivatives in order to compute gradients for the weight updates. We denote the weights between layers l and $l-1$ as $\boldsymbol{w^{(l)}}$, and the function that represents the action of the full layer as $\sigma^{(l)}$. Each layer in a NN can theoretically contain arbitrarily many neurons, where the input of the neurons in each layer is given by the output of the neurons in the previous layer. When a higher number of layers is involved, this is commonly referred to as *deep learning*. The full network with input $\boldsymbol{x}$ can then be written as the following composition of functions starting from the last layer L,

$$f(\boldsymbol{x}) = \sigma^{(L)}(\boldsymbol{w^{(L)}}\sigma^{L-1}(\boldsymbol{w^{(L-1)}}\sigma^{L-2}(\ \ldots\ \boldsymbol{w^{1}}\sigma^{0}(\boldsymbol{w^{0}x})\ \ldots\ ))). \tag{3.2}$$

A visualization of this type of NN can be seen in Figure 3.1. The above equation represents the simplest form of a NN, where the neurons in each layer are only connected to the neurons in neighboring layers, called a *feedforward* NN. The training data set consists of pairs of examples $\mathcal{X} = \{(x_i, y_i)\}$, and the goal is to adjust the weights of the network such that for each x_i, the network produces the desired output y_i. How close the model output $\hat{y}_i$ is to the target output, also called the *loss*, is quantified in terms of a *cost function*,

$$C(y_i, \hat{y}_i). \tag{3.3}$$

To optimize the weights $\boldsymbol{w}$, typically gradient descent with the *backpropagation algorithm* is used [86, 87]. This algorithm provides an efficient method to compute

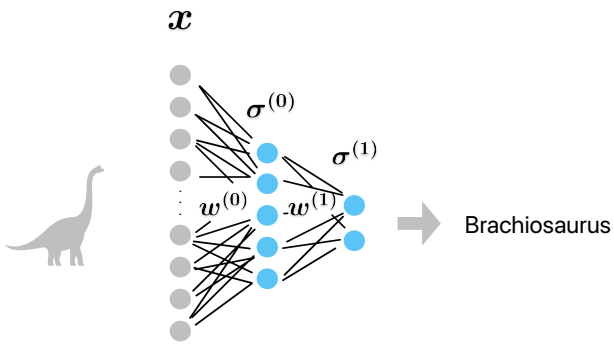


Figure 3.1: Depiction of a neural network that takes images of dinosaurs as input, and classifies them into two categories. The $\boldsymbol{w^{(l)}}$ are trainable weights, the $\sigma^{(l)}$ are layers of neurons with activation functions, and $\boldsymbol{x}$ is an input vector. Black lines show weights $w_{kj}^{(l)}$ between two neurons in adjacent layers.

gradients in NNs, utilizing their layered structure. Naively, one can compute the gradient of the cost function w.r.t. a given weight $w_{kj}^{(l)}$ via the chain rule. However, doing this will require redundant computations of partial derivatives as we work through the layers. Due to the fact that each layer only depends on its successors by how they affect the cost function, and does so in a linear manner, the computation of the gradient can be regarded layer-by-layer and performed without re-computing redundant derivatives. More formally, the loss of a NN with L layers on a pair of samples (x_i, y_i) is

$$C(y_i, \sigma^{(L)}(\boldsymbol{w^{(L)}}\sigma^{(L-1)}(\boldsymbol{w^{(L-1)}}\sigma^{(L-2)}(\ \ldots\ \boldsymbol{w^{(1)}}\sigma^{(0)}(\boldsymbol{w^{(0)}}\boldsymbol{x})\ \ldots\ )))). \tag{3.4}$$

The gradient of the cost function given input x is then the following,

$$\nabla_x C = \boldsymbol{w^{(0)T}} \cdot (\sigma^{(0)})' \circ \cdots \circ \boldsymbol{w^{(L-1)T}} \cdot (\sigma^{(L-1)})' \circ \boldsymbol{w^{(L)T}} \cdot (\sigma^{(L)})' \circ \nabla_{a^{(L)}} C, \tag{3.5}$$

where we denote the output of layer l as $a^{(l)}$, and the derivative of $\sigma^{(l)}$ as $(\sigma^{(l)})'$. Instead of computing the derivatives from left to right, where each computation at the i-th layer includes derivatives of the following layers of nested functions, in the backpropagation algorithm, derivatives are evaluated right to left, and the $a^{(l)}$ as well as the derivatives of each layer are cached along the way. In addition, propagating the error of a given training data pair backwards through the network constitutes multiplication of the vector of partial products of the derivatives with a matrix of weights for each layer. Performing the computation in the opposite

direction, on the other hand, constitutes of a multiplication of two matrices at each layer. These two changes in the procedure of computing gradients make the backpropagation algorithm much more efficient than a naive calculation that includes all the $L - i$ terms in every step, and is key to enable training large-scale NNs with billions of parameters. This is different to gradient computation in the VQAs we described in Section 2.2.1. There, gradients are obtained by the parameter-shift rule in Equation (2.16), which requires two circuit evaluations per trainable gate in the circuit. In particular, generic quantum circuits do not allow for a layer-wise computation of partial derivatives as in the backpropagation algorithm, where derivatives previously computed for other parameters in the circuit can be reused.

As mentioned before, the above describes computation of gradients in a feedforward NN, which is the simplest NN in terms of the connections between neurons. While gradients in this architecture can be computed efficiently, the all-to-all connectivity between each of the layers poses other problems for the trainability of the network. The most infamous of these problems is that of *vanishing* or *exploding gradients* [88]. This is a consequence of the layer-by-layer computation of gradients in the backpropagation algorithm. The gradient is computed by multiplying partial derivatives for each layer, so when derivatives are small, and in particular when they are smaller than one, multiplying several of these values will lead to a gradient that is vanishing exponentially in the number of terms that are multiplied. A similar effect is present when partial derivatives are large: multiplying increasingly large numbers in each layer leads to a blow-up of the derivative terms. In order to overcome this problem, weight initialization strategies [89], as well as activation functions [90] and more elaborate NN architectures [91] have been developed, and research in this area is still ongoing.

Despite these practical hurdles, NNs have become the most popular models used for large-scale ML due to their flexibility and the broad range of tasks they can be applied to. Indeed, theory shows that NNs can, in principle, approximate any function, a result that is known as the *universal approximation theorem* [92, 93]. There are actually a number of these universal approximation theorems, each pertaining different types of NN architectures: the *arbitrary width* case concerns networks with only one layer that contains many neurons [92, 94, 95], the *arbitrary depth* case is about networks that contain a limited number of neurons in each layer, but use several of those layers [96], and the *bounded width and depth* case is a

combination of the first two cases [97]. These results show that theoretically, NNs can represent many functions of interest, given a suitable set of weights. However, these theorems neither tell us anything about how to obtain these weights, nor how hard it is to find them. In order to make full use of the power of NNs, several challenges in high-dimensional optimization as the one mentioned above have to be overcome. In the following section, we will encounter one of those challenges, that results from a fundamental tradeoff between model complexity, and a model's ability to perform well on previously unseen data.

3.1.2 Generalization and overfitting

As briefly alluded to in the introduction of this chapter, the main goal of a machine learning algorithm is to learn to perform a given task without being explicitly programmed to do so. Information is inferred from a set of data samples or interactions with an environment, in order to use this information to process previously unseen data. In machine learning literature, the ability to use this information on unfamiliar data is referred to as *generalization*, and the failure to do so as the *generalization error*. Usually, these terms are formally defined in a supervised learning setting, where the algorithm is given a set of training data points $\mathcal{X} = \{(X = x_i, Y = y_i)\}$ sampled from some joint distribution of $X \times Y$, and the goal is to produce the output y_i given the input x_i, and the error between model output $\hat{y}_i$ and target label y_i is given by the cost function $C(y_i, \hat{y}_i)$, as in Section 3.1.1. The average difference between the output of a model parametrized by $\boldsymbol{\theta}$, denoted $\hat{y}_i^{(\boldsymbol{\theta})}$, and the target labels for a training data set of size N is then given as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} C(y_i, \hat{y}_i^{(\boldsymbol{\theta})}) \tag{3.6}$$

and is called the *empirical risk*. Naively, one could now define the goal of the ML model as finding the set of parameters $\boldsymbol{\theta}^*$ that minimizes the empirical risk on the given training data set,

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{3.7}$$

However, even under the assumption that there exists a set of parameters that yields a model with an empirical risk of zero, it is not clear that these parameters are also optimal for samples that were not included in the training data, i.e., whether this set of optimal parameters generalizes to unseen samples. Even worse, the parameters that are optimal for the training data may lead to higher error on

data not included in the training set, as the model is now overly fine-tuned on the training data. This effect is commonly referred to as *overfitting*.

To quantify the generalization performance of a model, it is helpful to specify the data that it was trained on in the definition of the risk,

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{X}) = \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} C(y_i, \hat{y}_i^{(\boldsymbol{\theta})}). \tag{3.8}$$

Now, assuming that we have access to the probability distribution that generated the training data, $p^*(x, y)$, we can define the *population risk*, also called *total risk*, as the theoretical expected loss

$$\mathcal{L}(\boldsymbol{\theta}; p^*) = \mathbb{E}_{p^*(x,y)}[C(y_i, \hat{y}_i^{(\boldsymbol{\theta})})]. \tag{3.9}$$

In practice, we usually do not have access to p^*, however, it lets us define the *generalization gap* as the difference between the population and the empirical risk, $\mathcal{L}(\boldsymbol{\theta}; p^*) - \mathcal{L}(\boldsymbol{\theta}; \mathcal{X})$. A large generalization gap, meaning that the empirical risk is low while the population risk is high, is a clear sign for overfitting. But how can we prevent the model from overfitting without knowing the population risk?

An empirical approach to prevent overfitting on the training data is to split the available data into three partitions: i) the training data that is used to fit the model, ii) a validation set that is used for hyperparameter tuning, and iii) a test set that is finally used to evaluate model performance. Further splitting the data into validation and test set is beneficial, as most ML models' performance strongly depends on the hyperparameter setting used. Hyperparameters in a NN are, for example, the learning rate η used to determine the step size of the gradient-based updates of weights, or the number of layers and neurons in each layer in the network. Once a well-performing set of hyperparameters is found based on the validation data set, the final model performance is evaluated on the test data set, which was not used in any of the previous steps of the model training and selection. There are many different techniques to perform this division of the data set that facilitate making the best use of the limited data that is available to train and evaluate a model. A commonly used technique is called *k-fold cross validation*, where the full data set is split into k parts, and the training and evaluation of a model is repeated multiple times where different partitions of the data act as training, validation and test sets, respectively.

Preventing overfitting in ML models is a highly non-trivial task, and a variety of different approaches have been developed for this. A models' capacity for overfitting is closely related to its complexity, where a rule of thumb is that the more complex a model is, the closer it can and will fit the training data exactly, unless measures to prevent this are taken. This results in a tradeoff between the model complexity and its generalization performance, where both, models that are too complex or not complex enough, will have bad generalization performance and one is required to find the model that has *just the right* level of complexity for a given learning task. This is referred to as the *bias-variance tradeoff* [98]. To make this formal, we consider the *bias-variance decomposition* of the expected error $\mathbb{E}_{p^*(x,y)}[(\hat{y} - y)^2|x]$, where for ease of notation we drop the explicit reference to $p^*(x, y)$ and x in the following, assuming that the cost function is the mean squared error between model prediction $\hat{y}$ and true label y,

$$\begin{aligned}
\mathbb{E}_{p^*(x,y)}[(\hat{y} - y)^2|x] &= \mathbb{E}[\hat{y}^2 - 2y\hat{y} + y^2] \\
&= \hat{y}^2 - 2\hat{y}\mathbb{E}[y] + \mathbb{E}[y^2] \\
&= \hat{y}^2 - 2\hat{y}\mathbb{E}[y] + \mathbb{E}[y]^2 + \mathrm{Var}[y] \\
&= (\hat{y} - \mathbb{E}[y])^2 + \mathrm{Var}[y] \\
&\triangleq (\hat{y} - y^*) + \mathrm{Var}[y],
\end{aligned} \tag{3.10}$$

where $y^* = \mathbb{E}[y]$ is the optimal prediction given x. If we now treat $\hat{y}$ as a random variable, where we repeatedly sample a data set from p^*, train the model, and generate predictions $\hat{y}$, we get the expected error

$$\begin{aligned}
\mathbb{E}[(\hat{y} - y)^2] &= \mathbb{E}[(\hat{y} - y^*)^2] + \mathrm{Var}[y] \\
&= \mathbb{E}[(y^*)^2 - 2y^*\hat{y} + \hat{y}^2] + \mathrm{Var}[y] \\
&= (y^*)^2 - 2y^*\mathbb{E}[\hat{y}] + \mathbb{E}[\hat{y}^2] + \mathrm{Var}[y] \\
&= (y^*)^2 - 2y^*\mathbb{E}[\hat{y}] + \mathbb{E}[\hat{y}]^2 + \mathrm{Var}[\hat{y}] + \mathrm{Var}[y] \\
&= \underbrace{(y^* - \mathbb{E}[\hat{y}])^2}_{\text{bias}} + \underbrace{\mathrm{Var}[\hat{y}]}_{\text{variance}} + \underbrace{\mathrm{Var}[y]}_{\text{Bayes error}}.
\end{aligned} \tag{3.11}$$

The bias term in Equation (3.11) represents the average error of the model predictions, while the second term informs us about the variance of model predictions on the training data. The third term is the variance over the true labels that we have no control over, and can therefore ignore in this discussion. High variance hints at an increased sensitivity of the model to small fluctuations in the training data, which can be a result of the model fitting the training data too closely, and

is therefore a sign of overfitting. The bias of the model results from erroneous predictions when the model misses crucial information in modeling the input-output relation, and is therefore a sign of *underfitting*. The above tells us that restricting the model by introducing a bias can actually be beneficial, as long as it reduces the variance. As described above, one difficulty at the heart of ML lies in finding an adequate balance between these two terms for the learning task at hand. In the following section, we will see how we can deliberately introduce an inductive bias based on knowledge about the training data into the model, in order to increase model performance and improve generalization.

3.1.3 Geometric deep learning

In the past years, the dimensionality of the input data of problems that are addressed with deep learning has kept increasing in size, and state-of-the-art NNs now contain billions of trainable weights. Finding a good set of weights poses an extremely high-dimensional optimization problem that comes with its own challenges. Most notably, we face the *curse of dimensionality* [99]: the data required to solve these high-dimensional learning problems often grows exponentially with the dimensionality. This means that learning about generic, unstructured data in high-dimensional spaces is generally infeasible (an effect we have already encountered for quantum models in Section 2.2.1.2). However, most of the problems that we are interested in solving with ML do have an underlying structure. If this knowledge about the structure of the learning task can be imposed on the model itself before training, this effectively reduces the dimensionality of the optimization problem. This problem-dependent imposing of structure is also referred to as the *inductive bias* of a model.

The field of *geometric deep learning* [100] is concerned with endowing models with an inductive bias from the perspective of geometry and symmetries. Most of the data that we are interested in learning about is generated by a process in our physical world, and many of these physical processes have underlying symmetries that make reasoning about them more easy. In fact, the study of symmetries is one of the cornerstones of modern physics, and underlies important theories like that of general relativity [101]. Therefore, it is not surprising that learning of functions in high-dimensional spaces can also be simplified by utilizing the symmetries that are present in the given training data. A key example of a NN with a problem-dependent inductive bias is the convolutional NN, that is used to process images

[102, 103, 104]. One symmetry present in images is that of translation invariance, i.e., shifting an object around in an image does not change the object itself. For example, a model that is used for object tracking in video data should recognize that a ball that is thrown and will therefore appear in different locations of successive frames of the video, is indeed the same ball. The layers of convolutional NNs are designed precisely so that they are invariant to translations. Another common data structure that has a corresponding symmetry-preserving NN architecture is that of graphs [105]. Graph NNs preserve permutation invariance, or the closely related permutation equivariance, of graph nodes, meaning that they are not affected by the order in which representations of graph nodes are fed into the network. Graph NNs can be used for many different types of learning tasks, like predicting properties of molecules [106], inferring relationships in social networks [107], or solving instances of combinatorial optimization problems [108]. The above types of geometric models have played a key role in enabling recent breakthroughs in deep learning, alongside the backpropagation algorithm we described in Section 3.1.1. In Chapter 6 of this thesis, we will go into more detail about geometric learning, and introduce a quantum model that is equivariant under permutation of graph nodes, similarly to the graph NNs described above.

3.2 Reinforcement learning

In this section, we focus on the branch of machine learning that is considered in most of the following chapters of this thesis: reinforcement learning (RL). In RL, an agent does not learn from a fixed data set as in other types of learning, but by making observations on and interacting with an environment [109]. This distinguishes it from the other two main branches of ML, supervised and unsupervised learning, and each of the three comes with its individual challenges. In a supervised setting, an agent is given a fixed set of training data that is provided with the correct labels, where difficulties arise mainly in creating models that do not overfit the training data and keep their performance high on unseen samples. In unsupervised learning, training data is not labeled and the model needs to discover the underlying structure of a given data set, and the challenge lies in finding suitable loss functions and training methods that enable this. RL also comes with a number of challenges: there is no fixed set of training data, but the agent generates its own samples by interacting with an environment. These samples are not labeled, but only come with feedback in form of a reward. Additionally, the training data keeps changing

throughout the learning process, as the agent constantly receives feedback from interacting with its environment. This often results in a high number of interactions that is required to successfully learn in a given environment, making RL the most sample-inefficient method out of the three described approaches.

An environment consists of a set of possible states $\mathcal{S}$ that it can take, and a set of actions $\mathcal{A}$ which the agent can perform to alter the environment's state. Both state and action spaces can be continuous or discrete. The function that models the agent's behavior in the environment is called the *policy* $\pi(a|s)$, which gives the probability of taking action a in a given state s. An agent interacts with an environment by performing an action a_t at time step t in state s_t, upon which it receives a reward r_{t+1}. A tuple (s, a, r, s') of these four quantities is called a *transition*, and a sequence of transitions is a *trajectory*. Another ingredient in the interaction between agent and environment are the transition probabilities from state s to s' after performing action a. They are represented by the *transition function* $P^a_{ss'}$, which for environments that can be described in terms of a Markov decision process (MDP) is defined as follows,

$$P^a_{ss'} = P(s'|s, a). \tag{3.12}$$

For environments that are not MDPs, the transition probabilities may depend on the whole trajectory instead of just the previous state and action. The transition function is a property of the environment, and one can distinguish between *model-free* algorithms as the ones we study in this work, where transition probabilities are not learned, and *model-based* environments which learn the transition function as a model of the environment. The quality of the agent's actions in the environment is evaluated by a *reward function* that is tailored to the learning task at hand, and the agent's goal is to learn a policy $\pi(a|s)$ that maximizes its total reward. The expected reward over a sequence of time steps starting at t is called the *return*,

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{3.13}$$

where $\gamma \in [0, 1)$ is a discount factor introduced to prevent divergence of the infinite sum. The return G_t should be viewed as the agent's expected reward when starting from time step t and summing the discounted rewards of potentially infinitely many future time steps, where maximizing the return at step t implies also maximizing the return of future time steps. Note that the task is to maximize an expected

value, and that the reward r_t in Equation (3.13), and therefore G_t are random variables. Environments often naturally break down into so-called *episodes*, where the sum in Equation (3.13) is not infinite, but only runs over a fixed number of steps called *horizon* H. An example of this are environments based on games, where one episode comprises one game played and an agent learns by playing a number of games in series.

Much of the theoretical work on RL is on so-called *tabular* algorithms, where the rewards for possible transitions are stored in a table. While this simplifies certain theoretical analyses such as proofs of convergence [110], these types of algorithms quickly become inefficient with growing state and action spaces. Therefore, different methods based on *function approximation* have been developed, where rewards are not stored explicitly anymore but approximately represented by a function. One example of RL with function approximation is deep reinforcement learning (DRL), where the function approximator is a NN. Since the advent of NNs in the past decades, much of RL research has focused on these types of algorithms, and we also study RL with function approximation in this thesis. For this reason, we only describe the most important concepts and techniques for RL with function approximators here, as a general introduction to the broad field of RL is out of the scope of this thesis. For more information on the history of RL, its connection to dynamic programming and the Bellman equation [111], and the various theoretical and practical formalisms developed in this field, the interested reader can refer to [109].

3.2.1 Value-based and policy-based learning

RL algorithms can be categorized into *value-based* and *policy-based* learning methods. Both approaches aim to maximize the return as explained above, but use different figures of merit to achieve this. Both approaches also have their disadvantages as we will see below, and which type of algorithm should be used depends on the environment at hand. The main difference between the two approaches is how the policy is realized. In general, performance is evaluated based on a state-value function $V_\pi(s)$,

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s], \tag{3.14}$$

which is the expected return when following policy π starting from state s at initial time step t, and the goal of a RL algorithm is to learn the optimal policy π_* which maximizes the expected return for each state.

A policy-based algorithm seeks to learn an optimal policy directly, that is, learn a probability distribution of actions given states. In this setting, the policy is implemented in form of a parametrized conditional probability distribution $\pi(a|s;\boldsymbol{\theta})$, and the goal of the algorithm is to find parameters $\boldsymbol{\theta}$ such that the resulting policy is optimal. The figure of merit in this setting is some performance measure $J(\boldsymbol{\theta})$ that we seek to maximize, that in the fixed-horizon setting is equal to the value function in Equation (3.14),

$$J(\boldsymbol{\theta}) := V_{\pi_{\boldsymbol{\theta}}}(s). \tag{3.15}$$

This performance measure is used to find a good policy, however, once the policy has been learned $J(\boldsymbol{\theta})$ is not required for action generation. Typically, these algorithms perform gradient ascent on an approximation of the gradient of the performance measure $\nabla J(\boldsymbol{\theta})$, which is obtained by Monte Carlo samples of policy rollouts (i.e., a set of observed interactions with the environment performed under the given policy), and are hence called *policy gradient* methods. This approach produces smooth updates on the policy (as opposed to value-based algorithms, where a small change in the value function can drastically alter the policy) that enable proofs of convergence to locally optimal policies [112]. However, it also suffers from high variance as updates are purely based on Monte Carlo samples of interactions with the environment [113]. A number of methods to reduce this variance have been developed, like adding a value-based component as described below to a policy-based learner in the so-called *actor-critic* method [114].

In a value-based algorithm, a value function as in Equation (3.14) is learned instead of the policy. The policy is then implicitly given by the value function: an agent will pick the action which yields the highest expected return according to $V_\pi(s)$. A concrete example of value-based learning is given in Section 3.2.2, where we describe the *Q-learning* algorithm that we use in later chapters of this thesis. While value-based algorithms do not suffer from high training variance as policy gradient learning does, they often require more episodes to converge. They also result in deterministic policies, as the agent always picks the action that corresponds to the highest expected reward, so this approach will fail when the optimal policy is stochastic and post-training action selection is performed according to the argmax policy.[1] Additionally, the policy resulting from a parametrized value function can change substantially after a single parameter update (i.e., a very small change in

[1] Consider for example a game of poker where bluffing is a valid action to scare other players into folding, but quickly becomes obvious when greedily done in every round.

the value function can lead to picking a different action after an update). This results in theoretical difficulties to prove convergence when a function approximator is used to parametrize the value function, hence there are even fewer theoretical guarantees for this approach than for policy gradient methods. On the other hand, it was the advent of Q-learning with function approximation that made it possible to solve extremely complex problems such as Go with a RL approach [29].

Both approaches have their own (dis-) advantages, and while the popularity of either method has surpassed the other at some point in the last decades, there is no clear winner. As mentioned above, an actor-critic approach combines a policy-based and value-based learner to leverage the advantages of both while alleviating the disadvantages, and this method is among the stat-of-the-art in classical RL literature [115]. Additionally, it can be easier to learn either the policy or the value function depending on a given environment. For this reason, both approaches are worth being studied independently.

3.2.2 Q-learning

In Q-learning, we are not interested in the state-value function as shown in Equation (3.14), but in the closely related action-value function $Q_\pi(s, a)$,

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a], \tag{3.16}$$

which also gives us the expected return assuming we follow a policy π, but now additionally conditioned on an action a. We call the optimal Q-function for all state-action pairs $Q_*(s, a) = \max_\pi Q_\pi(s, a)$, and an optimal policy can be easily derived from the optimal values by taking the highest-valued action in each step,

$$\pi_*(a|s) = \underset{a}{\operatorname{argmax}}\ Q_*(s, a). \tag{3.17}$$

The goal in Q-learning is to learn an estimate, $Q(s, a)$, of the optimal Q-function. In its original form, Q-learning is a tabular learning algorithm, where a so-called *Q-table* stores Q-values for each possible state-action pair [116]. When interacting with an environment, an agent chooses its next action depending on the Q-values as

$$a_t = \underset{a}{\operatorname{argmax}}\ Q(s_t, a), \tag{3.18}$$

where a higher value designates a higher expected reward when action a is taken in state s_t as opposed to the other available actions. When we consider learning

by interaction with an environment, it is important that the agent is exposed to a variety of transitions to sufficiently explore the state and action space. Intuitively, this provides the agent with enough information to tell apart good and bad actions given certain states. Theoretically, visiting all state-action transitions infinitely often is one of the conditions that are required to hold for convergence proofs of tabular Q-learning to an optimal policy [110]. Clearly, if we always follow an argmax policy, the agent may only get access to a limited part of the state and action space. To ensure sufficient exploration in a Q-learning setting, a so-called *ϵ-greedy policy* is used. That is, with probability ϵ a random action is performed and with probability $1 - \epsilon$ the agent chooses the action which corresponds to the highest Q-value for the given state as in Equation (3.18). Note that the ϵ-greedy policy is only used to introduce randomness to the actions picked by the agent during training, but once training is finished, a deterministic argmax policy is followed.

The Q-values are updated with observations made in the environment by the following update rule,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \tag{3.19}$$

where α is a learning rate, r_{t+1} is the reward at time $t + 1$, and γ is a discount factor. Intuitively, this update rule provides direct feedback from the environment in form of the observed reward, while simultaneously incorporating the agent's own expectation of future rewards at the present time step via the maximum achievable expected return in state s_{t+1}. In the limit of visiting all (s, a) pairs infinitely often, this update rule is proven to converge to the optimal Q-values in the tabular case [110].

Obviously, the tabular approach is intractable for large state and action spaces. For this reason, the Q-table was replaced in subsequent work by a Q-function approximator which does not store all Q-values individually [117, 118]. In the landmark work [119], the authors use a NN as the Q-function approximator which they call *deep Q-network* (DQN) and the resulting algorithm the *DQN algorithm*, and demonstrate that this algorithm achieves human-level performance on a number of arcade games. In this work, the agent chooses actions based on an ϵ-greedy policy as described above. Typically ϵ is chosen large in the beginning and then

decayed in subsequent iterations, to ensure that the agent can sufficiently explore the environment at early stages of training by being exposed to a variety of states. The authors of [119] also utilize two other methods to stabilize training these models: (i) *experience replay*: past transitions and their outcomes are stored in a memory, and the batches of these transitions that are used to compute parameter updates are sampled at random from this memory to remove temporal correlations between transitions, (ii) adding a second so-called *target network* to compute the expected Q-values for the update rule, where the target network has an identical structure as the DQN, but is not trained and only sporadically updated with a copy of the parameters of the DQN. Note that the target network is only used for computing parameter updates, but is not needed after the training procedure where only the Q-network is used. We refer to the original paper for further details on the necessity of these techniques to stabilize training of the DQN algorithm.

The DQN is then trained almost in a supervised fashion, where the training data and labels are generated by the DQN itself through interaction with the environment. At each update step, a batch $\mathcal{B}$ of previous transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ is chosen from the replay memory. To perform a model update, we need to compute $\max_a Q(s_{t+1}, a)$. When we use a target network, this value is not computed by the DQN, but by the target network $\hat{Q}$. To make training more efficient, in practice the Q-function approximator is redefined as a function of a state parametrized by $\boldsymbol{\theta}$, $Q_{\boldsymbol{\theta}}(s) = \boldsymbol{q}$, which returns a vector of Q-values for all possible actions instead of computing each $Q(s, a)$ individually. We now want to perform a supervised update of $Q_{\boldsymbol{\theta}}$, where the label is obtained by applying the update rule in Equation (3.20) to the DQN's output. To compute the label for a state s that we have taken action a on in the past, we take a copy of $Q_{\boldsymbol{\theta}}(s)$ which we call $\boldsymbol{q_\delta}$, and only the ith entry of $\boldsymbol{q_\delta}$ is altered where i corresponds to the index of the action a, and all other values remain unchanged. The estimated maximum Q-value for the following state s_{t+1} is computed by $\hat{Q}_{\boldsymbol{\theta}_\delta}$, and the update rule for the i-th entry in $\boldsymbol{q_\delta}$ takes the following form

$$\boldsymbol{q}_{\delta_i} = r_{t+1} + \gamma \cdot \max_a \hat{Q}_{\boldsymbol{\theta}_\delta}(s_{t+1}, a), \tag{3.20}$$

where $\boldsymbol{\theta}_\delta$ is a periodically updated copy of $\boldsymbol{\theta}$. The loss function $\mathcal{L}$ is the mean squared error (MSE) between $\boldsymbol{q}$ and $\boldsymbol{q}_\delta$ on a batch of sample transitions $\mathcal{B}$,

$$\mathcal{L}(\boldsymbol{q}, \boldsymbol{q}_\delta) = \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} (\boldsymbol{q}_b - \boldsymbol{q}_{\delta_b})^2. \tag{3.21}$$

Note that because $\boldsymbol{q}_\delta$ is a copy of $\boldsymbol{q}$ where only the i-th element is altered via the update rule in Equation (3.20), the difference between all other entries in those two vectors is zero. As Q-values are defined in terms of (s, a)-pairs, this approach does not naturally apply to environments with continuous action spaces. In this case, the continuous action space has to be binned into a discrete representation.

Q-values can take an arbitrary range, which is determined by the environment's reward function and the discount factor γ, which controls how strongly expected future rewards influence the agent's decisions and is in general specified by the environment definition. Depending on γ, the optimal Q-values for the same environment can take highly varying values, and can therefore be viewed as different learning environments themselves. In practice, it is not necessary that an agent learns the optimal Q-values exactly. As the next action at step t is chosen according to Equation (3.18), it is sufficient that the action with the highest expected reward has the highest Q-value for the sake of solving an environment presuming a deterministic policy. In other words, for solving an environment only the *descending order* of Q-values is important, and the task is to learn this correct order by observing rewards from the environment through interaction.

3.2.3 Policy gradients

As described above, when using a policy gradient method, the goal is to learn a parametrized policy directly based on a quantity $J(\boldsymbol{\theta})$, that in the fixed-horizon setting is equal to the value function (3.14),

$$J(\boldsymbol{\theta}) := V_{\pi_{\boldsymbol{\theta}}}(s). \tag{3.22}$$

In a gradient-based optimization procedure the parameters are updated according to

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\boldsymbol{\theta}_t), \tag{3.23}$$

with a learning rate α, i.e., we perform gradient ascent on the parameters to maximize the expected return. The policy gradient theorem [109] then states that the gradient of the performance measure can be written as

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &= \nabla V_{\pi_{\boldsymbol{\theta}}}(s) \\ &\propto \sum_s \mu(s) \sum_a \nabla \pi_{\boldsymbol{\theta}}(a|s) Q_\pi(s, a) \\ &= \mathop{\mathbb{E}}_{\pi_{\boldsymbol{\theta}}} \left[\sum_a \nabla \pi_{\boldsymbol{\theta}}(a|S_t) Q_\pi(S_t, a) \right], \end{aligned} \tag{3.24}$$

where $\mu(s)$ is the on-policy distribution under the current policy, which depends on the time spent in each state, and S_t in the third line of Equation (3.24) are states sampled under the policy π. Using this, we can now derive the REINFORCE algorithm, that is the basis of policy gradient based training.

Our goal is to perform gradient ascent on the parametrized policy purely from samples generated from said policy through interactions with the environment. The last line of Equation (3.24) still contains a sum over all actions a, which we can replace by the sample $A_t \sim \pi$ after multiplying and dividing the terms in the sum by $\pi_{\boldsymbol{\theta}}(a|S_t)$,

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &\propto \mathop{\mathbb{E}}_{\pi_{\boldsymbol{\theta}}}\left[\sum_a \pi_{\boldsymbol{\theta}}(a|S_t) Q_\pi(S_t,a)\frac{\nabla\pi_{\boldsymbol{\theta}}(a|S_t)}{\pi_{\boldsymbol{\theta}}(a|S_t)}\right] \\ &= \mathop{\mathbb{E}}_{\pi_{\boldsymbol{\theta}}}\left[Q_\pi(S_t,A_t)\frac{\nabla\pi_{\boldsymbol{\theta}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}}(A_t|S_t)}\right] \\ &= \mathop{\mathbb{E}}_{\pi_{\boldsymbol{\theta}}}\left[G_t\frac{\nabla\pi_{\boldsymbol{\theta}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}}(A_t|S_t)}\right], \end{aligned} \tag{3.25}$$

where G_t is the expected return from Equation (3.13). Now, by using the fact that $\nabla \log x = \frac{\nabla x}{x}$, we can write

$$\mathop{\mathbb{E}}_{\pi_{\boldsymbol{\theta}}}\left[G_t\frac{\nabla\pi_{\boldsymbol{\theta}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}}(A_t|S_t)}\right] = \mathop{\mathbb{E}}_{\pi_{\boldsymbol{\theta}}}\left[G_t \cdot \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)\right]. \tag{3.26}$$

This equation allows us to estimate the gradient of $J(\boldsymbol{\theta})$ simply by taking samples of interactions with the environment under the current policy $\pi_{\boldsymbol{\theta}}$, and leads us to the following parameter update in each iteration of the algorithm,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \, \nabla \log \pi_{\boldsymbol{\theta}}(A_t, S_t), \tag{3.27}$$

where α is again the learning rate, R_k is the reward, and T is the length of the episode. As mentioned above, value- and policy-based algorithms both have their strengths and shortcomings, and which of the two should be used depends on the environment. Indeed, both of these approaches, as well as their combination in the actor-critic framework, have been studied extensively classically as well as in a quantum setting, as we will see in the following section.

3.3 Quantum machine learning

After getting a basic understanding of quantum computing in Chapter 2, and the field of machine learning in this chapter, it is finally time to turn to the intersection

of these two fields: quantum machine learning (QML). As the focus of this thesis is on the near-term algorithms we described in Section 2.2.1, we will regard QML from the angle of VQAs, and will not discuss fault-tolerant algorithms, like the previously-mentioned one for solving linear systems of equations [5], in much detail. In Section 2.2.1, we have already seen how a gradient-based optimizer can be used to iteratively train the parameters of a quantum circuit, and how this can be applied to a number of different tasks in combinatorial optimization or quantum chemistry and simulation. When using these types of algorithms in a ML context, the parameter optimization scheme stays essentially the same. The only difference is that the parameters now have to be found for a set of training samples instead of just for the ground state of a single Hamiltonian, which introduces the same problems we have already studied for NNs above, like overfitting and the bias-variance tradeoff, to the realm of VQAs. Additionally, the quantum cousin of the vanishing gradient problem, the barren plateau phenomenon we have described in Section 2.2.1.2, remains to pose a challenge in the variational QML setting as well. Nonetheless, QML is a quickly evolving field [120, 121, 122], and these types of algorithms have been studied in a multitude of contexts, as we will see in the following section.

3.3.1 Near-term quantum machine learning

The QML algorithms that have been proposed for the NISQ-era can be broadly divided into two types: the VQAs we already know, which in this context are also referred to as quantum neural networks (QNNs)[1], and another approach that is based on the classical ideas of *kernel methods* [124]. Kernel methods are based on a similarity function called a *kernel* that maps data points into a *feature space*, and they make use of the so-called *kernel trick*. This trick allows computation of the similarity of data points in the potentially infinite-dimensional feature space given by the kernel, without ever having to compute the coordinates of the data points in that feature space directly. Instead, it suffices to only compute the inner products between the images of all pairs of data points to determine their similarity. A well-known example of a ML algorithm that utilizes kernels

[1]Note that there are also efforts to directly implement the classical neuron/perceptron model with a quantum approach [123], which is also sometimes referred to as a "quantum neural network." However, most of these are not suitable for NISQ devices so we do not consider them in this thesis. When we refer to QNNs, we mean parametrized quantum circuits that are trained with ML techniques to learn a certain task.

is the support vector machine (SVM) [125]. Simply put, the SVM is a linear classifier that learns to divide a set of points into two partitions, and does this by finding a separating hyperplane where the separation between data points in the two partitions is maximal. As this technique can only be used for linearly seperable data, it is combined with a kernel, where the kernel is chosen such that the given data points become linearly seperable in the high-dimensional feature space. The SVM then only has to effectively partition a linearly seperable data set. The idea of computing inner products of data points in a high-dimensional space suggests itself to be translated to the quantum domain, and indeed quantum kernel methods have enjoyed much popularity in the QML community in the past years [126, 26, 127, 128, 129]. One advantage of this type of model in the quantum setting is that the quantum device is only used to compute the kernel function, and therefore there is no need to variationally train circuit parameters. Furthermore, it has been shown that there exists a quantum kernel that allows for a rigorous separation between classical and quantum learners [128], even though it is for a contrived learning problem that does not directly translate to a task of practical relevance. Despite having no trainable parameters, however, it has been shown that quantum kernel methods suffer from a similar effect as the barren plateaus in VQAs, namely an exponential concentration of the inner products in the number of qubits [130].

In the classical literature, kernel methods were popular before the onset of deep learning that was a result of increased hardware performance in the early 2000's, and since then focus has substantially shifted to NNs in research as well as industry. Nonetheless, kernel methods are still interesting from the theoretical perspective, as they are much more amenable to obtaining rigorous results about their performance than NNs. Additionally, a certain type of kernel, called the *neural tangent kernel* (NTK), can be used to describe the evolution of certain NNs when they are trained with gradient descent [131]. With the NTK, theoretical results from kernel methods can be used to study NNs, and it was, for example, used to prove that wide enough NNs converge to a global minimum of the loss function [132, 133]. An analogous connection between QNNs and quantum kernel methods has been established as well [127, 134], and quantum versions of the NTK have also been studied [135, 136]. This connection between QNNs and kernel methods and its theoretical implications are a promising direction for future research.

QNNs themselves were studied in the context of a variety of different learning tasks in all the three main branches of ML. Early work in the field introduced QNNs for classification [137, 39, 138, 74], alongside classification with quantum kernel methods as mentioned above. Another well-studied area of QML is generative learning, a branch of unsupervised learning where the goal is to model a probability distribution that generated a given data set, to then produce similar samples that were not in the training set. As sampling from classically hard distributions is the basis of quantum supremacy experiments [2, 3], using PQCs to model probability distributions seems to be a promising direction of research. Quantum generative models can be broadly classified into three categories. The first are *quantum circuit born machines* (QCBMs) [139, 140, 28], that got their name from the fact that the model's probability to produce a certain sample is given by the Born rule. The second are quantum *generative adversarial networks* (GANs), inspired by classical GANs [141], where two models are trained together with opposing targets. The generator is trained to generate realistic samples of some data distribution, while the discriminator's goal is to detect the "fake" data generated by the generator. Several proposals for quantum GANs have been made [142, 143, 144, 145]. The third type of quantum generative model is related to classical Boltzmann machines [146], which are energy-based models that can be defined in terms of a Hamiltonian, and are closely related to Ising models and spin-glasses. Due to this connection to physics, Boltzmann machines naturally lend themselves to be adapted to the quantum domain, and indeed multiple proposals for variational quantum Boltzmann machines exist [147, 148], as well as general Hamiltonian-based models for learning thermal states [149].

Variational QML algorithms for the third branch of ML, reinforcement learning, are also an active area of research. Policy gradient methods, where the policy is parametrized in form of a PQC, have been explored in [150, 151, 76, 152], while Q-learning with PQCs as function approximators was studied in [153, 154, 75, 155, 156]. The combination of both types of models in the actor-critic framework was studied in [157, 158]. Variational RL is also the main type of learning explored in this thesis, where we study quantum Q-learning in Chapter 5, use it to train a model to solve instances of combinatorial optimization problems in Chapter 6, and evaluate the noise-robustness of variational proposals for Q-learning and policy gradients in Chapter 7, considering noise sources present on near-term quantum hardware.

Finally, it is also worth mentioning another type of variational QML that promises to be a fruitful direction for future research, namely hybrid quantum-classical algorithms. The term "hybrid" here does not refer to the fact that a classical model is used to find optimal parameters for a PQC, but is due to a classical and quantum model being trained together [159, 160, 161, 162, 163]. The idea behind this is that classical resources are leveraged as much as possible, and only those computations where a quantum computer is really beneficial are outsourced to the quantum model. As, especially for learning tasks that involve classical data, it can be expected that quantum resources will only be required for part of the overall computation, this approach enables getting the most out of the scarce quantum resources we expect to have in the near future. A key question in this setting is what exactly the classically hard parts in QML on a classical data set are or if they even exist in generic learning problems, for which so far, there is no clear answer, and this question is an interesting direction for future research.

3.3.2 Data encoding and the choice of ansatz

A major difference to VQAs used in the ML setting, as opposed to finding the ground states of Hamiltonians, is the fact that we now want to train a PQC given a set of training data samples. This raises the question of how to best provide the PQC with this data, which is especially important when we consider learning tasks on classical data, where this data has to be encoded into a quantum state that is accessible to the PQC.

One of the first ideas on how to efficiently store classical data in a quantum state was *amplitude encoding*, where a vector of n input values is stored in the amplitudes of a qubit register. This approach is efficient in the sense that it only requires logarithmically many qubits in the length of the input vector, so it is efficient in space. However, this comes at the cost of time, as in general, it takes time $\mathcal{O}(2^n)$ to prepare this state [164]. This means that amplitude encoding can require relatively deep circuits, so this approach is not practical for NISQ devices. Additionally, it turns out that classifying states resulting from amplitude encoded data can even be less efficient than classifying the original vectors directly [165], so care must be taken about the specific encoding that is used. When the data consists of bitstrings instead of real values, a simpler approach is to directly use these classical bitstrings as the inputs to a PQC by initializing each qubit in the register in the state $|0\rangle$ or $|1\rangle$, respectively, called *basis encoding*.

A third method, sometimes referred to as *qubit encoding* or *angle encoding* [166], inputs real-valued data as rotation angles of parametrized gates on the single- or multi-qubit level. While this method obviously does not share the advantage of being space-efficient, as in the case of amplitude encoding, qubit encoding can be performed with arbitrary parametrizable gates and shallow circuits, and it is therefore usually the number one choice in NISQ algorithms. However, care must be taken on how this encoding is performed, in order to ensure a high expressivity of the resulting QML model [167, 168, 169]. The authors of [167, 169] establish a connection between PQCs and Fourier series, where each PQC with input data x can be written as a truncated Fourier series $f(x)$ in the following way,

$$f(x) = \sum_{\omega \in \Omega} c_\omega e^{i\omega x}, \tag{3.28}$$

where Ω is the frequency spectrum of the Fourier series, which completely depends on the choice of gates used to encode the data. Therefore, the data encoding used in a QML model determines the complexity of the functions that this model can fit. If the encoding is too simple, for example when only a layer of parametrized X-rotations is used where the data is only encoded once, the model can only learn a sine function of its input, irrespective of the complexity of the rest of the circuit. One method to address this issue while keeping the encoding gates simple is to repeatedly encode the data in subsequent layers of the circuit (or equivalently in parallel on the level of the qubits), interspersed with the trainable parametrized gates [167, 168, 169]. This approach has been introduced under the name of *data re-uploading* in [168]. The number of repetitions of the data encoding directly influences the frequency spectrum of the Fourier series and, therefore, represents a hyperparameter that can be used to tune the model complexity in a PQC. While qubit encoding represents an easy and theoretically well-motivated method to encode data into a PQC, this direct correspondence between the dimension of the input vector and the number of qubits is somewhat problematic. Considering that state-of-the-art classical NNs can be trained on high-dimensional inputs, like thousands of pixels of an image, the question arises how these types of inputs can be efficiently encoded into PQCs without resorting to amplitude encoding. So far, there is no clear answer to this, and finding good data encoding techniques is an active area of research.

Along with the data encoding technique, a second architectural choice has to be made in designing a PQC for QML: the overall structure of the circuit. The

question of suitable ansatzes for QML is one that has only recently begun to be studied in the community. In lieu of ansatzes targeted for specific problems, an ansatz dubbed *hardware-efficient* (HWE) ansatz, that was originally introduced in [170], is often used. The idea behind this ansatz, as the name suggests, is to pick the circuit structure and gates in it according to the device it is targeted for, best utilizing the native connectivity of the quantum processing unit. Apart from this idea there is no clear definition of what a HWE ansatz looks like, but usually it consists of layers of parametrized single-qubit gates and additional entangling gates following a simple topology like a ring. While this ansatz can be a good choice for small-scale models, it is prone to suffer from barren plateaus (see Section 2.2.1.2) as the system size is scaled up. Therefore, it is clear that devising ansatzes that can be scaled up to a large number of qubits is of key importance.

One possibility to do this is to create problem-tailored ansatzes, and the nascent field of *quantum geometric learning*, inspired by the classical field of geometric deep learning introduced in Section 3.1.3, is a promising approach for doing this. The first ansatz of this type is called the quantum convolutional NN [55], in reference to classical convolutional NNs. The number of parameters in this ansatz grows only logarithmically with the input size, which makes it suitable for NISQ devices. In addition, it was proven that this ansatz does not suffer from the barren plateau phenomenon due to the shallowness of the resulting circuit [56]. A recent line of research also focuses on how to create circuits that are invariant or equivariant under certain group actions [171, 172, 173, 174, 175, 176, 57]. The permutation equivariant circuit that we introduce in Chapter 6 also belongs to this class of ansatzes. While these ansatzes are already promising, the search for scalable and performant problem-tailored quantum circuits has only just begun.

3.3.3 Is there potential for quantum advantage?

Possibly the most important question for near-term QML is whether we can expect any form of quantum advantage. As we have mentioned before, showing rigorous quantum advantage for VQAs is hard due to their heuristic nature, and empirical advantages are for now out of reach as the hardware is not there yet. In addition, quantum advantage, as opposed to quantum supremacy [2, 3], is not clearly defined so it is hard to give a definite answer to the above question. At least at the time of writing this thesis, there is no indication that variational QML will yield a speed-up on real-world, classical learning tasks.

Nonetheless, there are a number of interesting theoretical results that motivate further research in the field of QML. A prominent example is learning from *quantum data*, where data is fed into the QML model directly from a quantum source, i.e., an experiment or another quantum device. The idea behind this is similar to Richard Feynman's original quote that the most suitable tool to simulate a quantum system, is another quantum system. In the same vein, if you want to learn about quantum data, you better make the learning model quantum, too. It was shown that in this setting, there can be an exponential separation between classical and quantum learners [177]. Furthermore, the authors of [178] show that for some classically hard-to-compute functions, their outcomes can still be efficiently predicted by a classical ML model when it is given access to data generated by these functions.

In addition to speed-ups, the differences between classical and quantum NNs have also been studied from the angle of expressivity and the families of functions they can model. The authors of [179] investigate the expressivity of both types of models from an information geometric perspective, and introduce the effective dimension as a means to measure model complexity. They show that quantum models can have a higher effective dimension and a more favorable Fisher information spectrum compared to their classical counterparts. In a similar vein, but more specific to the generative learning setting, the authors of [28] show that there exist probability distributions that can be represented by a quantum circuit, but are hard to learn classically, and similar results were established in the context of the QAOA as well [180].

Looking beyond results that are deemed accessible to NISQ devices, there have been a number of works pertaining the families of functions that can be learned by quantum models. The authors of [128] establish en exponential separation between classical and quantum learners in the context on an SVM, where the data set is based on the discrete logarithm problem. In this setting, a quantum learner can efficiently compute the discrete logarithms of a given set of data points, and thereby turn a data set that looks essentially randomly labeled to a classical learner, into one that is linearly separable. This result has been extended to a RL setting for policy gradients in [150], showing the same exponential separation between classical and quantum RL agents, and we show in Section 5.2 that this separation also holds for certain types of environments in the Q-learning setting. Based on similar arguments from cryptography, the authors of [181] show an exponential separation for learning discrete distributions.

The above results show that there definitely exist functions that can not efficiently be learned with classical models. However, the settings in which this was shown are contrived and in a way reverse engineered by defining classically hard data sets on known classically hard tasks. The question remains whether these differences in quantum and classical models will eventually be beneficial on tasks that are practically relevant.

CHAPTER 4

Layerwise learning for quantum neural networks

A key aspect of successfully training the variational quantum machine learning models introduced in Section 3.3 is the classical outer loop that optimizes the circuit parameters. One of the most popular choices for parameter optimization in this context are gradient-based methods such as stochastic gradient descent, inspired by their extensive use in the optimization of classical NNs. While the gradient-based backpropagation algorithm [87] is one of the most successful methods used to train NNs today, its direct translation to quantum neural network (QNN)s has been challenging. For QNNs, parameter updates for minimizing an objective function are calculated by stochastic gradient descent, based on direct evaluation of derivatives of the objective with respect to parameters in a PQC, as described in Section 2.2.1. The PQC is executed on a quantum device, while the parameter optimization routine is handled by a classical outer loop. The outer loop's success depends on the quality of the quantum device's output, which in the case of gradient-based optimizers are the partial derivatives of the loss function with respect to the PQC.

As the authors of [41] have shown, gradients of random PQCs vanish exponentially in the number of qubits, as a function of the number of layers. Furthermore, the authors of [44] show that this effect also depends heavily on the choice of cost function, where the barren plateau effect is worst for global cost functions like the fidelity between two quantum states. Furthermore, it was shown that the partial derivatives vanish exponentially in the number of qubits under local Pauli noise, if the depth of the circuit grows linearly with the number of qubits [49]. When executed on a NISQ device, the issue is further amplified because small gradient

values can not be distinguished from hardware noise, or will need exponentially many measurements to do so. These challenges motivate the study of training strategies that avoid initialization on a barren plateau, as well as avoid creeping onto a plateau during the optimization process.

Indeed, a number of different optimization strategies for PQCs have been explored, including deterministic and stochastic optimizers [182, 23, 183, 184, 185, 186]. Regardless of the specific form of parameter update, the magnitudes of partial derivatives play a crucial role in descending towards the minimum of the objective function. Excessively small values will slow down the learning progress significantly, prevent progress, or even lead to false convergence of the algorithm to a sub-optimal objective function value. Crucially to this work, small values ultimately lead to a poor signal-to-noise ratio in PQC training algorithms due to the cost of information readout on a quantum device. Even if only sub-circuits of the overall circuit become sufficiently random during the training process, gradient values in a PQC will become exponentially small in the number of qubits [41]. Moreover, in quantum-classical algorithms there is a fundamental readout complexity cost of $\mathcal{O}(1/\epsilon)$ [187] as compared to a similar cost of $\mathcal{O}(\log(1/\epsilon))$ classically. This is because classical arithmetic with floating point numbers for calculating analytic gradients may be done one digit at a time, incurring a cost $\mathcal{O}(\log(1/\epsilon))$. In contrast, quantum methods that require estimation of expectation values by measurements, such as those utilized in NISQ algorithms, converge similarly to classical Monte Carlo sampling. This means that small gradient magnitudes can result in an exponential signal-to-noise problem when training quantum circuits, as the number of measurements required to precisely estimate a value is related to the magnitude of that value. As a consequence, gradients can become too small to be useful even for modest numbers of qubits and circuit depths, and a randomly initialized PQC will start the training procedure on a saddle point in the training landscape with no interesting directions in sight.

To utilize the capabilities of PQCs, methods that overcome this challenge have to be studied. Due to the vast success of gradient-based methods in the classical regime, this work is concerned with understanding how these methods can be adapted effectively for quantum circuits. We propose layerwise learning, where individual components of a circuit are added to the training routine successively. By starting the training routine in a shallow circuit configuration, we avoid the unfavorable type of random initialization described in [41] and Section 2.2.1.2,

which is inherent to randomly initialized circuits of even modest depth. In our approach, the circuit structure and number of parameters is successively grown while the circuit is trained, and randomization effects are contained to subsets of the parameters at all training steps. This does not only avoid initializing on a plateau, but also reduces the probability of creeping onto a plateau during training, e.g., when gradient values become smaller on approaching a local minimum.

We compare our approach to a simple strategy to avoid initialization on a barren plateau, namely setting all parameters to zero, and show how the use of a layerwise learning strategy increases the probability of successfully training a PQC with restricted precision induced by shot noise by up to 40% for classifying images of handwritten digits. Intuitively, this happens for reasons that are closely tied to the sampling complexity of gradients on quantum computers. By restricting the training and randomization to a smaller number of circuit components, we focus the magnitude of the gradient into a small parameter manifold. This avoids the randomization issue associated with barren plateaus, but importantly is also beneficial for a NISQ quantum cost model, which must stochastically sample from the training data as well as the components of the gradient. Simply put, with more magnitude in fewer components at each iteration, we receive meaningful training signal with fewer quantum circuit repetitions.

Another strategy to avoid barren plateaus was recently proposed by Grant et al. [53], where only a small part of the circuit is initialized randomly, and the remaining parameters are chosen such that the circuit represents the identity operation. This prevents initialization on a plateau, but only does so for the first training step, and also trains a large number of parameters during the whole training routine. Another way to avoid plateaus was introduced in [54], where multiple parameters of the circuit are enforced to take the same value. This reduces the overall number of trained parameters and restricts optimization to a specific manifold, at the cost of requiring a deeper circuit for convergence [188]. Aside from the context of barren plateaus, [189] investigates a layer-by-layer training scheme to speed up the learning process of a variational quantum eigensolver.

In the classical setting, layerwise learning strategies have been shown to produce results comparable to training a full network with respect to error rate and time to convergence for classical NNs [190, 191]. It has also been introduced as an efficient method to train deep belief networks, which are generative models that consist of restricted Boltzmann machines (RBMs) [192]. Here, multiple layers of

RBMs are stacked and trained greedily layer-by-layer, where each layer is trained individually by taking the output of its preceding layer as the training data. In classical NNs, [193] shows that this strategy can be successfully used as a form of pre-training of the full network to avoid the problem of vanishing gradients caused by random initialization. In contrast to greedy layerwise pre-training, our approach does not necessarily train each layer individually, but successively grows the circuit to increase the number of parameters and therefore its representational capacity.

4.1 Layerwise learning

In Section 2.2.1.2, we described why training random PQCs with growing system size becomes increasingly harder, as the variance of gradients vanishes exponentially in the number of qubits and layers for these types of circuits. This necessitates the search for i) non-random circuit structures that are immune to this issue, or ii) optimization techniques that can combat the problem of vanishing gradients. We will later focus our attention on i) in Chapter 6, while in this section, we introduce a training method for PQCs in the line of ii). We call this optimization technique layerwise learning (LL) for parametrized quantum circuits, a training strategy that creates an ansatz during optimization, and only trains subsets of parameters simultaneously to ensure a favorable signal-to-noise ratio. The algorithm consists of two phases.

Phase one: The first phase of the algorithm constructs the ansatz by successively adding layers. The training starts by optimizing a circuit that consists of a small number s of start layers, e.g. $s = 2$, where all parameters are initialized as zero. We call these the initial layers $l_1(\boldsymbol{\theta}_1)$:

$$l_1(\boldsymbol{\theta}_1) = \prod_{j=1}^{s} U_{1_j}(\boldsymbol{\theta}_{1_j}) W \ , \tag{4.1}$$

where $\boldsymbol{\theta}_1$ is the set of angles parametrizing unitary U_{1_j}, and contains one angle for each local rotation gate per qubit, and W represents operators connecting qubits. The number of start layers is a hyperparamter, and should be chosen so that the initial circuit is shallow, but still sufficiently deep in order to advance training. How many start layers are required to fulfill this depends strongly on the learning task.

After a fixed number of epochs, another set of layers is added, and the previous layers' parameters are frozen. We define one epoch as the set of iterations it takes the algorithm to see each training sample once, and one iteration as one update over all trainable parameters. E.g., an algorithm trained on 100 samples with a batch size of 20 will need 5 iterations to complete one epoch. The number of epochs per layer, e_l, is a tunable hyperparameter. Each consecutive layer $l_i(\boldsymbol{\theta}_i)$ takes the form

$$l_i(\boldsymbol{\theta}_i) = \prod_{j=1}^{p} U_{i_j}(\boldsymbol{\theta}_{i_j}) W \ , \tag{4.2}$$

with a fixed W, as the connectivity of qubits stays the same during the whole training procedure, and p denoting the number of layers added at once. All angles in $\boldsymbol{\theta}_i$ are set to zero when they are added, which provides additional degrees of freedom for the optimization routine without perturbing the current solution. The parameters added with each layer are optimized together with the existing set of parameters of previous layers in a configuration dependent on two hyperparameters p and q. The hyperparameter p determines how many layers are added in each step, and q specifies after how many layers the previous layers' parameters are frozen. E.g., with $p = 2$ and $q = 4$, we add two layers in each step, and layers more than four steps back from the last layer are frozen. This process is repeated either until additional layers do not yield an improvement in objective function value, or until a desired depth is reached. The final circuit that consists of L layers can then be represented by

$$U(\boldsymbol{\theta}) = \prod_{i=1}^{L} l_i(\boldsymbol{\theta}_i) \ . \tag{4.3}$$

Phase two: In the second phase of the algorithm, we take the pre-trained circuit acquired in phase one, and train larger contiguous partitions of layers at a time. The hyperparameter r specifies the percentage of parameters that is trained in one step, e.g., a quarter or a half of the circuit's layers. The number of epochs for which these partitions are trained is also controlled by e_l, which we keep at the same value as in phase one for the sake of simplicity, but which could in principle be treated as a separate hyperparameter. In this setting, we perform additional optimization sweeps where we alternate over training the specified subsets of parameters simultaneously, until the algorithm converges. This allows us to train larger partitions of the circuit at once, as the parameters from phase one provide a sufficiently non-random initialization. As the randomness is contained to shallower sub-circuits during the whole training routine, we also minimize the probability to

creep onto a plateau during training as a consequence of stochastic or hardware noise present in the sampling procedure.

In general, the specific structure of layers $l_i(\boldsymbol{\theta}_i)$ can be arbitrary, as long as they allow successively increasing the number of layers, like in the hardware-efficient ansatz introduced in [170]. In this work, we indirectly compare the quality of gradients produced by our optimization strategy with respect to the results described in section 2.2.1.2 through overall optimization performance, so we consider circuits that consist of layers of randomly chosen gates as used in [41]. They can be represented in the following form:

$$U(\boldsymbol{\theta}) = \prod_{l=1}^{L} U_l(\boldsymbol{\theta_l}) W \ , \tag{4.4}$$

where $U_l(\boldsymbol{\theta_l}) = \prod_{i=1}^{n} \exp(-i\theta_{l,i} V_i)$ with a Hermitian operator V_i, n is the number of qubits, and W is a generic fixed unitary operator. For ease of exposition, we drop the subscripts of the individual gates in the remainder of this work. We consider single qubit generators V which are the Pauli operators X, Y and Z for each qubit, parametrized by θ_l, while W are CZ gates coupling arbitrary pairs of qubits. An example layer is depicted in Figure 4.1.

The structure and parameters of a quantum circuit define which regions of an optimization landscape given by a certain objective function can be captured. As the number of parametrized non-commuting gates grows, this allows a more fine-grained representation of the optimization landscape [188]. In a setting where arbitrarily small gradients do not pose a problem, e.g. noiseless simulation of PQC training, it is often preferable to simultaneously train all parameters in a circuit to make use of the full range of degree of freedom in the parameter landscape. We will refer to this training scheme as complete-depth learning (CDL) from now on. In a noiseless setting, LL and CDL will perform similarly w.r.t. the number of calls to a quantum processing unit (QPU) until convergence and final accuracy of results, as we show in the appendix. This is due to a trade off between the number of parameters in a circuit and the number of sampling steps to convergence [188]. A circuit with more parameters will converge faster in number of training epochs, but will need more QPU calls to train the full set of parameters in each epoch. On the other hand, a circuit with fewer parameters will show a less steep learning curve, but will also need fewer calls to the QPU in each update step due to the reduced number of parameters. When we consider actual number of

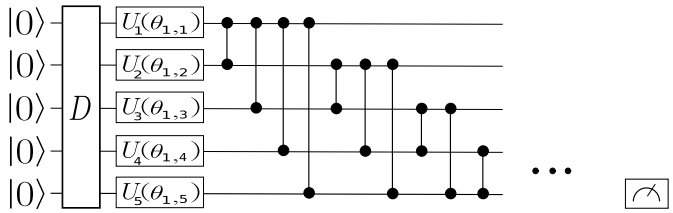


Figure 4.1: Sample circuit layout of the first layer in an LL circuit. D represents the data input which is only present once at the beginning of the circuit. The full circuit is built by successively stacking single rotation gates and two qubit gates to form all-to-all connected layers. For the classification example we show in 4.2, a measurement gate is added on the last qubit after the last layer.

calls to a quantum device until convergence as a figure of merit, training the full circuit and performing LL will perform similarly in a noise-free setting for this reason. However, this is not true when we enter a more realistic regime, where measurement of gradient values will be affected by stochastic as well as hardware noise, as we will show on the example of shot noise in Section 4.2. In such more realistic settings, the layerwise strategy offers considerable advantage in time to solution and quality of solution.

As noted in the appendix of [41], the convergence of a circuit to a 2-design does not only depend on the number of qubits, their connectivity and the circuit depth, but also on the characteristics of the cost function used. This was further investigated in [44], where cost functions are divided into those that are local and global, in the sense that a global cost function uses the combined output of all qubits (e.g., the fidelity of two states), whereas a local cost function compares values of individual qubits or subsets thereof (e.g., a majority vote). Both works show that for global cost functions, the variance of gradients decays more rapidly, and that barren plateaus will present themselves even in shallow circuits. As our training strategy relies on using larger gradient values in shallow circuit configurations, especially during the beginning of the training routine, we expect that LL will mostly yield an advantage in combination with local cost functions.

4.2 Results

4.2.1 Setup

To examine the effectiveness of LL, we use it to train a circuit with fully-connected layers as described in Section 4.1. While fully-connected layers are not realistic on NISQ hardware, we choose this configuration for our numerical investigations because it leads circuits to converge to a 2-design with the smallest number of qubits and layers [41], which allows us to reduce the computational cost of our simulations while examining some of the most challenging situations. To compare the performance of LL and CDL we perform binary classification on the MNIST data set of handwritten digits, where the circuit learns to distinguish between the numbers six and nine. We use the binary cross-entropy as the training objective function, given by

$$-\mathcal{L}(\boldsymbol{\theta}) = -\big(y \log\left(E(\boldsymbol{\theta})\right) + (1-y)\log\left(1-E(\boldsymbol{\theta})\right)\big) \,, \tag{4.5}$$

where log is the natural logarithm, $E(\boldsymbol{\theta})$ is given by a measurement in the Z-direction $M = Z_o$ on qubit o which we rescale to lie between 0 and 1 instead of -1 and 1, y is the correct label value for a given sample, and $\boldsymbol{\theta}$ are the parameters of the PQC. The loss is computed as the average binary cross entropy over the batch of samples. In this case, the partial derivative of the loss function is given by

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_i} = y\frac{1}{E(\boldsymbol{\theta})}\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} - (1-y)\frac{1}{1-E(\boldsymbol{\theta})}\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} \,. \tag{4.6}$$

To calculate the objective function value, we take the expectation value of the circuit of observable M,

$$E(\boldsymbol{\theta}) = \langle\psi| U^\dagger(\boldsymbol{\theta}) M U(\boldsymbol{\theta}) |\psi\rangle \,, \tag{4.7}$$

where $|\psi\rangle$ is the initial state of the circuit given by the training data set. The objective function now takes the form $\mathcal{L}(E(\boldsymbol{\theta}))$ and the partial derivative for parameter θ_i is defined using the chain rule as

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \frac{\partial \mathcal{L}}{\partial E(\theta_i)} \cdot \frac{\partial E(\theta_i)}{\partial \theta_i} \,. \tag{4.8}$$

To compute gradients of $E(\boldsymbol{\theta})$, we use the parameter-shift rule [39, 33] as described in Section 2.2.1.1. We note that in the numerical implementation, care must be

taken to avoid singularities in the training processes related to $E(\theta) = \{0, 1\}$ treated similarly for both the loss and its derivative (we clip values to lie in $[10^{-15}, 1 - 10^{-15}]$). We choose the last qubit in the circuit as the readout o, as shown in Figure 4.1. An expectation value of 0 (1) denotes a classification result for class sixes (nines). As we perform binary classification, we encode the classification result into one measurement qubit for ease of implementation. This can be generalized to multi-label classification by encoding classification results into multiple qubits, by assigning the measurement of one observable to one data label. We use the Adam optimizer [38] with varying learning rates to calculate parameter updates and leave the rest of the Adam hyperparameters at their typical publication values.

To feed training data into the PQC, we use qubit encoding in combination with principal component analysis (PCA), following [24]. Due to the small circuits used in this work, we have to heavily downsample the MNIST images. For this, a PCA is run on the data set, and the number of principal components with highest variance corresponding to the number of qubits is used to encode the data into the PQC. This is done by scaling the component values to lie within $[0, 2\pi)$, and using the scaled values to parametrize a data layer consisting of local X-gates. In case of 10 qubits, this means that each image is represented by a vector $\boldsymbol{d}$ with the 10 components, and the data layer can be written as $\prod_{i=1}^{10} \exp(-id_i X_i)$.

Different circuits of the same size behave more and more similarly during training as they grow more random as a direct consequence of the results in [41]. This means that we can pick a random circuit instance that, as a function of its number of qubits and layers, lies in the 2-design regime as shown in Figure 2.3, and gather representative training statistics on this instance. As noted in Section 4.1, an LL scheme is more advantageous in a setting where training the full circuit is infeasible, therefore we pick a circuit with 8 qubits and 21 layers for our experiments, at which size the circuit is in this regime. When using only a subset of qubits in a circuit as readout, a randomly generated layer might not be able to significantly change its output. For example, if in our simple circuit in Figure 4.1, $U_5(\theta_{1,5})$ is a rotation around the Z axis followed only by CZ gates, no change in $\theta_{1,5}$ will affect the measurement outcome on the bottom qubit. When choosing generators randomly from $\{X, Y, Z\}$ in this setting, there is a chance of 1/3 to pick an unsuitable generator. To avoid this effect, we enforce at least one X gate in each set of layers

that is trained. For our experiments, we take one random circuit instance and perform LL and CDL with varying hyperparameters.

4.2.2 Sampling requirements

To give insight into the sampling requirements of our algorithm, we have to determine the components that we need to sample. Our training algorithm makes use of gradients of the objective function that are sampled from the circuit on the quantum computer via the parameter shift rule as described in Section 4.2.1. The precision of our gradients now depends on the precision of the expectation values for the two parts of the r.h.s. in Equation 2.16. The estimation of an expectation value scales in the number of measurements N as $\mathcal{O}(\frac{1}{\epsilon^\alpha})$, with error ϵ and $\alpha > 1$ [187]. For most near-term implementations using operator averaging, $\alpha = 2$, resembling classical central limit theorem statistics of sampling. This means that the magnitude of partial derivatives $\frac{\partial E}{\partial \theta_i}$ of the objective function directly influences the number of samples needed by setting a lower bound on ϵ, and hence the signal-to-noise ratio achievable for a fixed sampling cost. If all of the magnitudes of $\frac{\partial E}{\partial \theta_i}$ are much smaller than ϵ, a gradient based algorithm will exhibit dynamics more resembling a random walk than optimization.

4.2.3 Comparison to CDL strategies

We compare LL to a simple approach to avoid initialization on a barren plateau, which is to set all circuit parameters in a circuit to zero followed by a CDL training strategy. We argue that considering the sampling requirements of training PQCs as described in Section 4.2.2, an LL strategy will be more frugal in the number of samples it needs from the QPU. Shallow circuits produce gradients with larger magnitude as can be seen in Figure 2.3, so the number of samples $1/\epsilon^2$ we need to achieve precision ϵ directly depends on the largest component in the gradient. This difference is exhibited naturally when considering the number of samples as a hyperparameter in improving time to solution for training. In this low sample regime, the training progress depends largely on the learning rate. A small batch size and low number of measurements will increase the variance of objective function values. This can be balanced by choosing a lower learning rate, at the cost of taking more optimization steps to reach the same objective function value. We argue that the CDL approach will need much smaller learning rates to compensate for smaller gradient values and the simultaneous update of all parameters in each

training step, and therefore more samples from the QPU to reach similar objective function values as LL. We compare both approaches w.r.t. their probability to reach a given accuracy on the test set, and infer the number of repeated re-starts one would expect in a real-world experiment based on that.

In order to easily translate the simulated results here to experimental impact, we also compute an average runtime by assuming a sampling rate of 10kHz. This value is assumed to be realistic in the near term future, based on current superconducting qubit experiments shown in [2] which were done with a sampling rate of 5kHz, not including cloud latency effects. The cumulative number of individual measurements taken from a quantum device during training is defined as

$$r_i = r_{i-1} + 2n_p mb \ , \tag{4.9}$$

where n_p is the number of parameters (taken times two to account for the parameter shift rule shown in Section 4.2.1), m the number of measurements taken from the quantum device for each expectation value estimation, and b the batch size. This gives us a realistic estimate of the resources used by both approaches in an experimental setting on a quantum device.

4.2.4 Numerical results

For the following experiments, we use a circuit with 8 qubits, 1 initial layer and 20 added layers, which makes 21 layers in total. As can be seen in Figure 2.3, this is a depth where a fully random circuit is expected to converge to a 2-design for the all-to-all connectivity that we chose. After doing a hyperparameter search over p, q and e_l, we set the LL hyperparameters to $p = q = 2$ and $e_l = 10$, with one initial layer that is always active during training. This means that three layers are trained at once in phase one of the algorithm, and 10 and 11 layers are trained as one contiguous partition in phase two, respectively. For CDL, the same circuit is trained with all-zero initialization.

We argue that LL not only avoids initialization on a plateau, but is also less susceptible to randomization during training. In NISQ devices, this type of randomization is expected to come from two sources: (i) hardware noise, (ii) shot noise, or measurement uncertainty. The smaller the values we want to estimate and the less exact the measurements we can take from a QPU are, the more often we have to repeat them to get an accurate result. Here, we investigate the robustness of both methods to shot noise. The hyperparameters we can tune are the

number of measurements m, batch size b and learning rate η. The randomization of circuits during training can be reduced by choosing smaller learning rates to reduce the effect of each individual parameter update, at the cost of more epochs to convergence. Therefore we focus our hyperparameter search on the learning rate η, after fixing the batch size to $b = 20$ and the number of measurements to $m = 10$. This combination of m and b was chosen for a fixed, small m after conducting a search over $b \in \{20, 50, 100\}$ for which both LL and CDL could perform successful runs that do not diverge during training. As we lower the batch size, we also increase the variance in objective function values similar to when the number of measurements is reduced, so these two values have to be tuned to match each other. In the remainder of this section we show results for these hyperparameters, and different learning rates for both methods. All of the results are based on 100 runs of the same hyperparameter configurations. We use 50 samples of each class to calculate the cross entropy during training, and another 50 samples per class to calculate the test error. To compute the test error, we let the model predict binary class labels for each presented sample, where a prediction ≤ 0.5 is interpreted as class 0 (sixes) and > 0.5 as class 1 (nines). The test error is then the average error over all classified samples.

Figure 4.2 shows average runtimes of LL and CDL runs that have a final average error on the test set that is less than 0.5, which corresponds to random guessing. We compute the runtime by computing the number of samples taken as shown in Section 4.2.3 and assume a sampling rate of 10kHz. Here, LL reaches a lower error on the test set on average, and also requires a lower runtime to get there. Compared to the CDL configuration with the highest success probability shown in Figure 4.3 (b) (red line), the best LL configuration (blue line) takes approximately half as much time to converge. This illustrates that LL does not only increase the probability of successful runs, but can also drastically reduce the runtime to train PQCs by only training a subset of all parameters at a given training step. Note also that the test error of CDL with $\eta = 0.05$ and $\eta = 0.01$ slowly increases at later training steps, which might look like overfitting at first. Here it is important to emphasize that these are averaged results, and what is slowly increasing is rather the percentage of circuits that have randomized or diverged at later training steps. The actual randomization in an individual run usually happens with a sudden jump in test error, after which the circuit can not return to a regular training routine anymore.

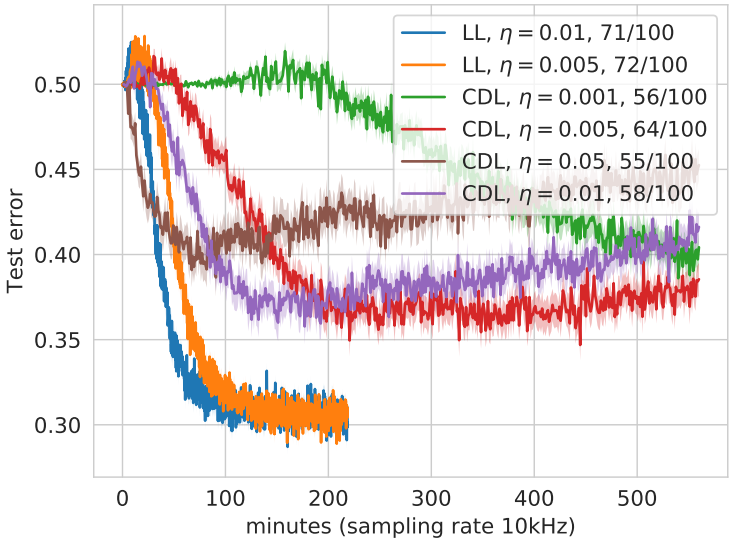
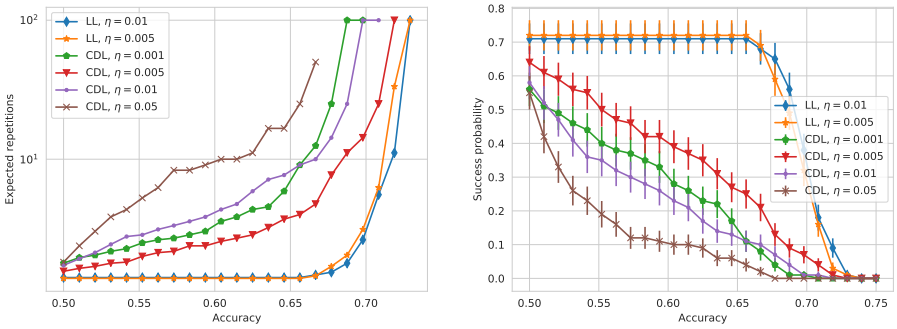


Figure 4.2: Average test error as a function of runtimes for runs that have a final average test error less than 0.5 (random guessing) over the last ten training epochs, assuming a sampling rate of 10kHz and number of samples taken as described in Section 4.2.3. Numbers in labels indicate how many out of 100 runs were included in the average, i.e. fraction of runs that did not diverge in training, exhibiting less than 50% error on the test set. Increasing test error for CDL runs with $\eta = 0.01$ and $\eta = 0.05$ is not due to overfitting, but due to a larger number of runs in the average that start creeping onto a plateau due to the increased learning rate.



(a) expected number of repetitions

(b) success probability

Figure 4.3: LL decreases expected run time and increases probability of success on random restarts. (a) Expected number of experiment repetitions needed until a given configuration reaches a certain accuracy defined as $(1 - \text{error}_{\text{test}})$, where $\text{error}_{\text{test}}$ is the average error on the test set, for LL and CDL with different learning rates. One experiment repetition constitutes in one complete training run of a circuit to a fixed number of epochs. Results are based on 100 runs for each configuration with $m = 10$, $b = 20$, and in case of LL, $e_l = 10$. LL circuits better avoid randomization during training, and therefore need less than two repetitions on average for learning rates with varying magnitudes. CDL is more susceptible to entering a plateau during training in a noisy environment, as all parameters are affected on a perturbative update. This effect becomes more pronounced as learning rates are increased. (b) Probability of reaching a certain accuracy on the test set for the same configurations shown in (a). Success probability of LL stays constant up to an accuracy of 0.65 and starts decaying from there, as fewer runs reach higher accuracies on average. All CDL configurations have a lower success probability than the LL configurations overall, which decays almost linearly as we demand a higher average accuracy. Notably, the CDL configurations with highest success probability are also the ones with the highest runtime, as shown in Figure 4.2.

Figure 4.3 (a) shows the number of expected training repetitions one has to perform to get a training run that reaches a given accuracy on the test set, where we define accuracy as $(1 - \text{error}_{\text{test}})$. One training run constitutes training the circuit to a fixed number of epochs, where the average training time for one run is shown in Figure 4.2. An accuracy of 0.5 corresponds to random guessing, while an accuracy of around 0.73 is the highest accuracy any of the performed runs reached, and corresponds to the model classifying 73% of samples correctly. We note that in a noiseless setting as shown in Chapter 8, both LL and CDL manage to reach accuracies around 0.9, and the strong reduction in number of measurements leads to a decrease in the final accuracy reached by all models. We find that LL performs well for different magnitudes of learning rates as $\eta = 0.01$ and $\eta = 0.005$, and that these configurations have a number of expected repetitions that stays almost constant as we increase the desired accuracy. On average, one needs less than two restarts to get a successful training run when using LL. For CDL, the number of repetitions increases as we require the test error to reach lower values. The best configurations were those with $\eta = 0.001$ and $\eta = 0.005$, which reach similarly low test errors as LL, but need between 3 and 7 restarts to succeed in doing so. This is due to the effect of randomization during training, which is caused by the high variance in objective function values, and the simultaneous update of all parameters in each training step. In Figure 4.3 (b), we show the probability of each configuration shown in (a) to reach a given accuracy on the test set. All CDL configurations have a probability lower than 0.3 to reach an accuracy above 0.65, while LL reaches this accuracy with a probability of over 0.7 in both cases. This translates to the almost constant number of repetitions for LL runs in Figure 4.3 (a). Due to the small number of measurements and the low batch size, some of the runs performed for both methods fail to learn at all, which is why none of the configurations have a success probability of 1 for all runs to be better than random guessing.

4.3 Conclusion and outlook

We have shown that the effects of barren plateaus in QNN training landscapes can be dampened by avoiding Haar random initialization and randomization during training through layerwise learning. While performance of LL and CDL strategies is similar when considering noiseless simulation and exact analytical gradients, LL strategies outperform CDL training on average when experimentally

realistic measurement strategies are considered. Intuitively, the advantage of this approach is drawn from both preventing excess randomization and concentrating the contributions of the training gradient into fewer, known components. Doing so directly decreases the sample requirements for a favorable signal-to-noise ratio in training with stochastic quantum samples. To quantify this in a cost effective manner for simulation, we reduce the number of measurements taken for estimating each expectation value. We show that LL can reach lower objective function values with this small number of measurements, while reducing the number of overall experiment repetitions until convergence to roughly half of the repetitions needed by CDL when comparing the configurations with highest success probability. This makes LL a more suitable approach for implementation on NISQ devices, where taking a large number of measurements is still costly and where results are further diluted by decoherence effects and other machine errors. While our approach relies on manipulating the circuit structure itself to avoid initializing a circuit that forms a 2-design, it can be combined with approaches that seek to find a favorable initial set of parameters as shown in [53]. The combination of these two approaches by choosing good initial parameters for new layers is especially interesting as the circuits grow in size. This work has also only explored the most basic training scheme of adding a new layer after a fixed number of epochs, which can still be improved by picking smarter criteria like only adding a new layer after the previous circuit configuration converged, or replacing gates in layers which provide little effect on changes of the objective function value. Moreover, one could consider training strategies which group sets of coordinates rather than circuit layers. These possibilities provide interesting directions for additional research, and we leave their investigation for future works.

CHAPTER 5

Quantum agents in the Gym: A variational quantum algorithm for deep Q-learning

The focus of Chapter 4 was to introduce a training method for VQAs that addresses the problem of barren plateaus, while reducing the number of trained parameters in ever update step of the optimization routine. While we demonstrated the feasibility and effectiveness of our method on a supervised learning task, the learning problem itself was not the focus of that work. In this chapter, we go deeper into investigating the use of VQAs for a specific type of ML algorithm. Many proposals for QML algorithms have been made in supervised [39, 74, 126, 26, 137] and unsupervised [194, 28, 148, 195, 143, 196] learning. In contrast, RL is a subfield of machine learning that has received less attention in the QML community [197, 198], and especially proposals for VQA-based approaches are only now emerging [153, 154, 199, 120, 150]. RL is essentially a way to solve the problem of optimal control. In a RL task, an agent is not given a fixed set of training data, but learns from interaction with an environment. Environments are defined by a space of states they can be in, and a space of actions that an agent uses to alter the environment's state. The agent chooses its next action based on a policy (probability distribution over actions given states) and receives a reward at each step, and the goal is to learn an optimal policy that maximizes the long-term reward the agent gets in the environment. State and action spaces can be arbitrarily complex, and it's an open question which types of models are best suited for these learning tasks. In classical RL, using NNs as function approximators for the agents' policy has received increased interest in the past decade. As opposed to learning exact functions to model agent behavior which is infeasible in large state and action spaces, this method of RL only approximates the optimal function. These types of RL algorithms have been shown to play Atari arcade games as well as human

players [119], and even reach super-human levels of performance on games as complex as Go [29], Dota [200] and StarCraft [201]. RL algorithms can be divided into *policy-based* and *value-based* methods, as described in Section 3.2. These two methods constitute related but fundamentally different approaches to solve RL tasks, and both have their own (dis-)advantages. Interestingly, these two methods can also be combined in a so-called *actor-critic* setting which leverages the strengths of both approaches [114]. Actor-critic methods are among the state-of-the-art in current RL literature [115], and therefore both value-based and policy-based algorithms are areas of active research.

RL is one of the hardest modes of learning in current ML research, and is known to require careful tuning of model architectures and hyperparameters to perform well. For NN-based approaches, one unfavorable hyperparameter setting can lead to complete failure of the learning algorithm on a specific task. Additionally, these hyperparameters and architectures are highly task dependent and there is no a-priori way to know which settings are best. Well-performing settings are found by experts via trial-and-error, and the ability to quickly find these settings is considered a "black art that requires years of experience to acquire" [202]. Thus a whole field of heuristics and numerical studies has formed on finding good sets of hyperparameters like NN architectures [203, 204, 205], activation functions [206, 207, 208], or learning rates and batch sizes [202, 209]. An increasingly investigated branch of research focuses on methods to automate the whole process of finding good architectures and hyperparameters, among which there is neural architecture search [210] and automated machine learning [211].

It is thus to be expected that quantum models in a VQA-based RL setting also need to be selected carefully. Even more so, it is still an open question whether VQAs are suitable for function approximation in RL at all. This question is directly related to choices made when defining an architecture for a VQA. There are three important factors to consider: the structure (or *ansatz*) of the model, the data-encoding technique, and the readout operators. For the choice of structure, there is a trade-off between the expressivity and trainability of a model, as certain structures are subject to the barren plateau phenomenon as described in Section 2.2.1.2. On the other hand, overparametrization has been observed to simplify optimization landscapes and lead to faster convergence for certain VQAs [212, 213]. Apart from that, the choice of structure is also limited by hardware constraints like the topology of a certain quantum device. While the model structure is an important

factor in training VQAs that has received much attention in the QML community [44, 49, 214, 47, 215, 216, 217], the authors of [169] have shown that the technique used to encode data into the model plays an equally important role, and that even highly expressive structures fail to fit simple functions with an insufficient data-encoding strategy.

A less explored architectural choice in the context of QML is that of the observables used to read out information from the quantum model. Considering that the readout operator of a quantum model fixes the range of values it can produce, this choice is especially important for tasks where the goal is to fit a real-valued function with a given range, as is the case in many RL algorithms. This is in contrast to NNs, which have no restriction on the range of output values and can even change this range dynamically during training. In Q-learning, the goal is to approximate the real-valued optimal Q-function, which can have an arbitrary range based on the environment. Crucially, this range can change depending on the performance of the agent in the environment, which is an impediment for quantum models with a fixed range of output values.

A first step to study the influence of architectural choices on PQCs for policy-based RL algorithms has been made in [150], who point out that data-encoding and readout strategies play a crucial role in these types of RL tasks, though they leave the open question if similar architectural choices are also required in a value-based setting. Previous work on Q-learning with PQCs has addressed certain other fundamental questions about the applicability of VQAs in a value-based context. A VQA for Q-learning in discrete state spaces was introduced in [153], where the quantum model's output is followed by a layer of additive weights, and it has been shown that the model successfully solves two discrete-state environments. A VQA for Q-learning in environments with continuous and discrete state spaces has been proposed in [154], who simplify the continuous environments' potentially infinite range of input values to a restricted encoding into angles of one initial layer of rotation gates, and use measurements in the Z-basis to represent Q-values. Notably, none of the models in [154] that were run for the continuous state-space environment Cart Pole reach a performance that is considered to be solving the environment according to its original specification [218], so it remains an open question whether a value-based algorithm that utilizes a PQC as the function approximator can solve this type of learning task.

These initial works prompt a number of vital follow-up questions related to the architectural choices that are required to succeed in arbitrary RL environments with a quantum Q-learning agent. We address these questions in form of our main contributions as follows: first, we propose a VQA which can encode states of discrete and continuous RL environments and explain the intricate relationship between the environment's specification and the requirements on the readout operators of the quantum model. We show how a quantum Q-learning agent only succeeds if these requirements are met. Second, to enable the model to match the environment's requirements on the range of output values, we make this range itself trainable by introducing additional weights on the model outputs. We show how the necessity of these weights can be inferred from the range that the optimal Q-values take in an environment. Third, we study the performance of our model on two benchmark environments from the OpenAI Gym [219], Frozen Lake and Cart Pole. For the continuous-state Cart Pole environment, we also study a number of data encoding methods and illustrate the benefit of previously introduced techniques to increase quantum model expressivity, like data re-uploading [168] or trainable weights on the input data [168, 150]. Additionally, the state space dimension of both environments is small enough so that inputs can be directly encoded into the quantum model without the use of a dimensionality reduction technique. This makes it possible to directly compare our model to a NN performing the same type of Q-learning algorithm to evaluate its performance. Specifically, we perform an in-depth comparison of the performance of PQCs and NNs with varying numbers of parameters on the Cart Pole environment. We show that recent results in classical deep Q-learning also apply to the case when a PQC is used as the function approximator, namely that increasing the number of parameters is only beneficial up to some point [220]. After this, learning becomes increasingly unstable for both PQCs and NNs. As an empirical comparison between PQCs and NNs can only give us insight into model performance on the specific environments we study, we also explain when recent separation results for policy gradient RL between classical and quantum agents [150] also hold in the Q-learning setting for restricted families of environments.

The remainder of this chapter is structured as follows: we give a description of our quantum Q-learning model in Section 5.1 and show when recent results for a separation between classical and quantum algorithms for policy-based learning also apply in the case of Q-learning in Section 5.2. In Section 5.3 we numerically evaluate the performance of our algorithm and compare it to a classical approach, and

finally discuss our findings in Section 5.4. The full code that was used to perform the numerical experiments in this work can be found on Github [221].

5.1 Quantum Q-learning

In this work, we adapt the DQN algorithm to use a PQC as its Q-function approximator instead of a NN. For this, we use a hardware-efficient ansatz [170] as shown in Figure 5.1. This ansatz is known to be highly expressive, and is susceptible to the barren plateau phenomenon for a large number of qubits and layers, although this is not an issue for the small state and action spaces we consider here. All other aspects of the Q-learning algorithm described in Section 3.2.2 stay the same: we use a target network, an ϵ-greedy policy to determine the agent's next action, and experience replay to draw samples for training the Q-network PQC. Our Q-network PQC is then $U_{\boldsymbol{\theta}}(s)$ parametrized by $\boldsymbol{\theta}$ and the target network PQC is $\hat{U}_{\boldsymbol{\theta}_\delta}(s)$, where $\boldsymbol{\theta}_\delta$ is a snapshot of the parameters $\boldsymbol{\theta}$ which is taken after fixed intervals of episodes δ and the circuit is otherwise identical to that of $U_{\boldsymbol{\theta}}(s)$. We now explain how environment states are encoded into our quantum model, and how measurements are performed to obtain Q-values.

5.1.1 Encoding environment states

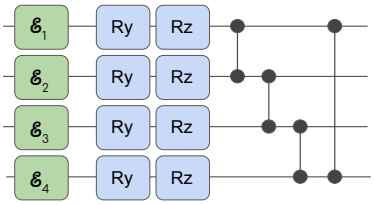


Figure 5.1: PQC architecture used in this work. Each layer consists of a parametrized rotation along the Y and Z axes on each qubit, and a daisy chain of CZ gates. The green boxes correspond to data encoding gates that encode data as parameters of X rotations. When data re-uploading is used, the whole circuit pictured is repeated in each layer, without data re-uploading only the variational part without the initial X rotations is repeated.

Depending on the state space of the environment, we distinguish between two different types of encoding in this work:

Discrete state space: Discrete states are mapped to bitstrings and then input into the model, where on an all-zero state the bits corresponding to ones in the input state are flipped.

Continuous state space: For continuous input states, we scale each component x of an input state vector $\boldsymbol{x}$ to $x' = \arctan(x) \in [-\pi/2, \pi/2]$ and then perform a variational encoding, which consists of X-rotations by the angles x'.

As shown in [169], when data is encoded into a PQC by local rotation gates along the X-axis, the PQC can only model simple sine functions of its input. To further increase the expressivity of the circuit, the data encoding can be repeated in two ways: either in parallel by increasing the number of qubits and duplicating the data encoding on them, or in sequence in an alternating fashion with the variational layers of the circuit. The latter is also referred to as *data re-uploading* in [168]. Where needed, we will introduce data re-uploading to our model in Section 5.3.

The formalism introduced in [169] establishes a connection between PQCs and partial Fourier series by showing that the functions a given PQC can model can be represented as a Fourier series, where the accessible frequency spectrum depends on the eigenvalues of the data encoding gates, and the coefficients depend on the architecture of the variational part of the PQC and the observable that defines the readout operation. They show that in models as ours, where data is encoded in form of Pauli rotations, only Fourier series up to a certain degree can be learned, where the degree depends on the number of times the encoding gate is repeated. Additionally, the scale of the input data must match the scale of the frequencies of the modeled function for the model to fit the target function exactly. Making the scaling of input data itself trainable to increase a PQC's expressivity has been suggested in [168, 150], which we will also use by introducing a weight $\boldsymbol{w}_d$ on the input data. The input value x'_i then becomes:

$$x'_i = \arctan(x_i \cdot w_{d_i}) \ , \tag{5.1}$$

where w_{d_i} is the weight for input x_i. We will illustrate the advantage of these enhanced data-encoding strategies numerically in Section 5.3.

5.1.2 Computing Q-values

The Q-values of our quantum agent are computed as the expectation values of a PQC that is fed a state s as

$$Q(s,a) = \left\langle 0^{\otimes n} \right| U_{\boldsymbol{\theta}}^{\dagger}(s) O_a U_{\boldsymbol{\theta}}(s) \left| 0^{\otimes n} \right\rangle, \tag{5.2}$$

where O_a is an observable and n the number of qubits, and our model outputs a vector including Q-values for each possible O_a as described in Section 3.2.2. Note that in practice, we can only compute an approximation of Equation (5.2) on a quantum device. The type of measurements we perform to estimate Q-values will be described in more detail in Section 5.3 for each environment. Before that, we want to highlight why the way Q-values are read out from the PQC is an important factor that determines the success at solving the environment at hand. A key difference between PQCs and NNs is that a PQC has a fixed range of output defined by its measurements, while a NN's range of output values can change arbitrarily during training depending on its weights and activation function. To understand why this is an important difference in a RL setting, we need to recall that Q-values are an estimate of the expected return

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{H-1} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right]. \end{aligned}$$

This quantity is directly linked to the performance of the agent in a given environment, so the model needs to have the ability to match the range of optimal Q-values in order to approximate the optimal Q-function. This means that the observables in a PQC-based Q-learning agent need to be chosen with care, and highly depend on the specific environment. To provide a simple example where an insufficient range prevents an agent from solving an environment, consider tabular learning in an environment that consists of a single state s and two actions a_1 and a_2, where the agent should learn to always pick a_1. One episode has a maximum length of $H = 10$ when the agent picks a_1 in each time step, and otherwise terminates when the agents picks action a_2. We consider a modification where the values in the Q-table are capped at 1, i.e., Q-values can not become larger than one, and both Q-values are initialized at zero. The environment is such that the reward for each action is 1 and the Q-value corresponding to the optimal action is > 1. For simplicity we set $\alpha = 1$ and $\gamma = 1$, which gives us an optimal value $Q_*(s, a_1) = 10$.

We now perform an update on both Q-values according to the update rule in Equation (3.19),

$$Q(s_t, a_t) \leftarrow r_{t+1} + \text{argmax}_a Q(s_{t+1}, a).$$

For action a_2, the transition from s leads to episode termination, so the update rule yields $Q(s, a_2) = r_{t+1} = 1$. For action a_1, we get $Q(s, a_1) = 2$, however, due to the capped Q-table, we also get $Q(s, a_1) = 1$ for this state-action pair. We see that after a single update according to this update rule, both Q-values will be one and due to the capped range of the Q-table the Q-values are already saturated. No further update can change the Q-values, which means that the agent can do no better than random guessing hereafter. This simple example illustrates why it is essential in a tabular Q-learning setting that the range of values in the Q-table accommodates the magnitude of optimal Q-values. Updates in the function approximation case like in the gradient-based DQN algorithm are more complex due to the regression task that the agent solves to perform parameter updates, however, a similar saturation can still occur as the update rule for Q-values is the same (see Equation (3.20)).

We have seen that it is crucial for a PQC-based Q-learning agent to have an output range that matches that of the optimal Q-values that it seeks to approximate. There are two ways to approach this issue: (i) multiply PQC outputs by a fixed factor to increase their range in a way that accommodates the theoretical maximum Q-value, (ii) make the output range itself a trainable model parameter. Multiplying the outputs of the PQC by a fixed factor increases the range of output values, but at the cost of potentially being close to the estimated maximum from the beginning, which makes this approach more sensitive to randomness in model parameter initialization. In particular, as Q-values are initialized randomly depending on the initial parameters of the PQC, the Q-values for actions of a specific state might have large differences. Considering that the reward which controls the magnitude of change given by the Q-value updates in Equation (3.20) is comparatively small and actions are picked based on the argmax policy $\text{argmax}_a Q(s, a)$, it may take a long time before subsequent updates of Q-values will lead to the agent picking the right actions. Even if we consider models that are initialized such that all Q-values are close to zero in the beginning, the actual changes in the rotation angles that the PQC needs to perform for Q-values of large ranges can become very small. Especially on NISQ devices, these changes might be impractically small to be reliably performed and measured on hardware. For these reasons, we focus on

option (ii). We add a trainable weight $w_o \in \mathbb{R}$ to each readout operation, so that the output Q-value $Q(s,a)$ becomes

$$Q(s,a) = \left\langle 0^{\otimes n} \right| U_{\boldsymbol{\theta}}(s)^\dagger O_a U_{\boldsymbol{\theta}}(s) \left| 0^{\otimes n} \right\rangle \cdot w_{o_a}, \tag{5.3}$$

and each action has a separate weight w_{o_a}. We make the weights multiplicative in analogy to weights in a NN. This gives the model the possibility to flexibly increase the magnitude of Q-values to match the given environment. Notably, the number of actions in an environment is usually small compared to the number of parameters in the model, so adding one extra weight corresponding to each action does not designate a large overhead. In Section 5.3.2.1, we numerically show that the approach of using a trainable weight on the output value outperforms multiplying the model output by a fixed factor that is motivated by the range of optimal Q-values.

5.2 Separation between quantum and classical Q-learning in restricted environments

In this section, we make formal statements about a separation between quantum and classical models for Q-learning in a restricted family of environments. These statements are based on recent results in supervised [128] and policy gradient based reinforcement learning [150]. The latter work constructs families of environments that are proven to be hard for any classical learner, but can be solved in polynomial time by a quantum learner in a policy learning setting. Learning policies is closely related to learning Q-values, however, Q-values contain more information about the environment per definition as they cover the whole state-action space. This means that it is not straightforward to generalize the results from [150] to a Q-learning setting. In this section, we will show under which conditions optimal Q-values can be inferred from optimal policies, so that the separation results in [150] also apply to the Q-learning case. The environments constructed in [150] are based on the supervised learning task introduced in [128], which are proven to be classically hard assuming the widely-believed hardness of the discrete logarithm problem, but can be solved by a quantum learner in polynomial time. To understand how a separation in supervised learning can be generalized to a RL setting, it is important to state that any classification task can be turned into an environment for RL. To do this, rewards in the environment are assigned according to the prediction

the agent makes. First examples of this were introduced in [222] for cases where the environment allows quantum access to its states. A classification task like the one proposed in [128] can be turned into a RL task by simply assigning a reward of 1 (-1) for a correct (incorrect) classification, and defining an episode as being presented with a set of training samples. In this section, we will briefly revise the separation results for supervised learning given in [128] and those for policy gradient RL given in [150], before we move on to characterize the types of environments that allow a generalization of the results in [150] to a Q-learning setting.

5.2.1 A classification task based on the discrete logarithm problem

The authors of [128] construct a classification task that is intractable for any classical learner, but can be solved by a quantum learner in polynomial time. The classification task is based on the discrete logarithm problem (DLP), and the separation relies on the the quantum learner's ability to perform the algorithm provided by Shor in [1] to solve the DLP efficiently.

Definition 5.1 (Discrete logarithm problem)**.** *Let* $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ *be the cyclic multiplicative group of integers modulo* p *for a large prime* p*, and* g *a generator of this group. The DLP is defined as computing* $\log_g x$ *for an input* $x \in \mathbb{Z}_p^*$*.*

It is widely believed that no classical algorithm can solve the DLP efficiently, however, it is proven that the algorithm provided by Shor can solve DLP in poly(n) time for $n = \lceil \log_2 p \rceil$ [1]. Based on this, [128] construct a classification task with a concept class $\mathcal{C} = \{f_s\}_{s \in \mathbb{Z}_p^*}$ and data points defined over the data space $\mathcal{X} = \mathbb{Z}_p^* \subseteq \{0,1\}^n$ as

$$f_s(x) = \begin{cases} +1, & \text{if } \log_g x \in [s, s + \frac{p-3}{2}] \\ -1, & \text{otherwise,} \end{cases} \tag{5.4}$$

where each concept $f_s : \mathbb{Z}_p^* \to \{-1, 1\}$ maps one half of the elements in $\mathbb{Z}_p^*$ to 1 and the other half to -1, which yields a linearly separable set of data points in log-space. A quantum learner can make use of the algorithm from [1] to compute the discrete logarithm and solve the resulting trivial learning task. However, if a classical learner could solve the above learning task this would imply that there exists an

efficient classical algorithm that solves the DLP. This is contrary to the widely believed conjecture that no efficient classical algorithm can solve the DLP, and [128] proves that no classical learner can do better than random guessing.

To connect these results to the RL setting, it is useful to be a bit more precise and define some terminology. The learning task is defined as finding a decision rule f^*, which assigns a label $y \in \{-1, 1\}$ to data point $x \in \mathcal{X}$[1]. f^* is learned on a set of labeled examples $S = \{x_i, y_i\}_{i=1,\dots,m}$ generated by the unknown decision rule, or *concept*, f. An efficient learner needs to compute f^* in time polynomial in n that agrees with the labeling given by f with high probability, or in other words reaches a high *test accuracy* on unseen samples,

$$\mathrm{acc}_f(f^*) = \Pr_{x \in \mathcal{X}}[f(x) = f^*(x)]. \tag{5.5}$$

The authors of [128] prove that no efficient classical learner can achieve

$$\mathrm{acc}_f(f^*) = \frac{1}{2} + \frac{1}{\mathrm{poly}(n)}$$

unless an efficient classical algorithm that solves the DLP exists, while there exists a quantum learner that achieves close to perfect accuracy with high probability in polynomial time.

5.2.2 Learning optimal policies in environments based on the DLP classification task

After stating the classification task based on the DLP in the previous section, we now briefly review how the authors of [150] construct families of environments based on the DLP classification task to transfer the separation results to RL. They show that (i) solving these environments is classically hard for any learner unless there exists an efficient classical algorithm that solves the DLP, (ii) there exists a quantum learner that can solve these environments in polynomial time. To understand how the DLP classification task can be used to construct a classically hard to solve RL environment, it is important to note again that any classification

[1] Note that we are adhering to the notation given in [128], where the asterisk stands for the learned decision rule and the function without an asterisk stands for the decision rule we seek to learn. This is the opposite of the notation used in Q-learning literature where Q_* stands for the optimal Q-values, which we have followed in previous sections. The authors of [150] have also adopted the latter notation in their paper to describe the DLP classification task. We will stick to denoting the learned decision rule with an asterisk in this section.

task can be trivially turned into a RL task by letting each data point $x \in \mathcal{X}$ denote a state in the environment, and giving rewards to the agent depending on whether it correctly assigns a state to its predefined label y. The rewards for the DLP classification task are 1 (-1) for a correct (false) classification. While [128] are interested in achieving a high test accuracy, in a RL setting we want to find an agent with close-to-optimal performance in the given environment. The authors of [150] measure this performance in terms of a value function $V_\pi(s)$ for policy π and state s,

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{H-1} \gamma^t r_t | s_t = s \right] \tag{5.6}$$

which is the expected reward for following policy π for an episode of length H in state s. Based on the DLP classification task from [128], the authors of [150] define three different environments that are classically hard to learn, where the value function of each of these environments is closely related to the accuracy in Equation (5.5) of the policy on the classification task. This allows them to get bounds on the value function as a function of bounds on the accuracy. Roughly speaking, by Theorem 1 of [128] no classical learner can achieve performance better than that of random guessing in poly(n) time on those environments, unless an efficient classical algorithm to solve the DLP exists. We will briefly explain the set-up of the quantum learner in [150], before going into more detail on one of the families of environments they construct to show a separation between classical and quantum learners for policy learning.

A RL agent can be trivially constructed from the classifier in [128], which is based on a classical support vector machine (SVM) that takes the samples that have been "decrypted" by a quantum feature map as an input. (This type of classifier is also referred to as an *implicit* SVM). However, to get a learner that more closely matches the parametrized training of a quantum learner done in [150], they use a model where the feature embedding and classification task are both solved by a PQC. This method is referred to as an *explicit* SVM. The explicit SVM comprises a feature-encoding unitary $U(x)$ applied on the all zero state, which they refer to as $|\phi(x)\rangle = U(x)\,|0^{\otimes n}\rangle$, a variational part $V(\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$, and an observable O. The feature-encoding unitary for the DLP task is the same as used in [128] so that feature states take the following form for $k = n - t \log n$ for a constant t related to noisy classification (we refer the reader to [150] for a detailed description

of classification under noise),

$$|\phi(x)\rangle = \frac{1}{\sqrt{2^k}} \sum_{i=0}^{2^k-1} \left|x \cdot g^i\right\rangle . \tag{5.7}$$

These states can be efficiently prepared on a fault-tolerant quantum computer by a circuit that uses the algorithm proposed by Shor in [1] as a subroutine. It was proven in [128] that for all concepts f_s the data points with labels 1 and -1, respectively, can be separated by a hyperplane with a large margin, and that this hyperplane always exists. The learning task of the PQC $V(\boldsymbol{\theta})$ is then to find this hyperplane. The hyperplanes are normal to states of the form

$$|\phi_{s'}\rangle = \frac{1}{\sqrt{(p-1)/2}} \sum_{i=0}^{(p-3)/2} \left|g^{s'+i}\right\rangle , \tag{5.8}$$

for $s' \in \mathbb{Z}_p^*$. A classifier $h_{s'}(x)$ for these data points can then be defined as

$$h_{s'}(x) = \begin{cases} 1, & \text{if } |\langle\phi(x)|\phi_{s'}|\phi(x)|\phi_{s'}\rangle|^2/\Delta \geq 1/2 \\ -1, & \text{otherwise,} \end{cases} \tag{5.9}$$

where $\Delta = \frac{2^k+1}{p-1}$ is the largest value the inner product $|\langle\phi(x)|\phi_{s'}|\phi(x)|\phi_{s'}\rangle|^2$ takes and is used to renormalize it to $[0,1]$. The variational circuit is defined as $V(\boldsymbol{\theta}) = \hat{V}(s')$ which is similar in implementation to $U(x_i)$ with $x_i = g^{s'}$ and $k \approx n/2$, and a measurement operator $O = |0^{\otimes n}\rangle\langle 0^{\otimes n}|$.

The simplest way of turning the DLP classification task into an environment is to define one episode as the agent being in a randomly chosen state corresponding to a training sample, performing an action which assigns the predicted label, and giving a reward of 1 (-1) for a correct (incorrect) classification. This family of environments is referred to as SL-DLP in [150]. While the family of SL-DLP environments is a straightforward way to generalize the results from [128] to policy learning, it lacks the characteristics typically associated with RL, namely a temporal structure in the state transitions, such that these depend on the actions taken by the agent. To construct a family of environments based on the DLP which includes this kind of structure, [150] introduce the family of Cliffwalk-DLP environments, inspired by the textbook Cliffwalk environment from [109]. Here, the goal is still to assign correct labels to given states, but now these states follow a randomly assigned but fixed order. The agent has to "walk along the edge of a cliff", where this edge is represented by the sequence of ordered states the

environment takes. A correct classification leads to the next state in the sequence, while an incorrect classification leads to “falling off the cliff" and immediate episode termination. The authors of [150] show that the quantum learnability results of the SL-DLP environment also hold for the family of Cliffwalk-DLP environments. In the following section we will generalize these results to Q-values by giving a definition of the types of environments where knowledge of an optimal policy lets us infer optimal Q-values.

5.2.3 Estimating optimal Q-values from optimal policies

In Section 5.2.2, we revised how [150] construct an efficient quantum agent that can achieve close-to-optimal policies in families of environments based on the DLP. Now, we turn to generalizing their results to the Q-learning setting. The classical hardness of the environment still holds irrespective of the learner that is used. The remaining question is now whether there exists an efficient algorithm to obtain optimal Q-values, given we have access to an optimal policy. Concretely, our goal is to compute optimal Q-values $Q_*(s,a)$ for state-action pairs from an environment, where s is given by the environment and a is determined by the optimal policy.

One could imagine that $Q_*(s,a)$ can be easily estimated using Monte Carlo sampling since the definition involves only the use of the optimal policy after the move (s,a) (cf. Equation (3.16)). However, in general it is not possible for an agent to get to arbitrary states s in poly time. We circumvent this problem by considering special cases of environments that are classically hard, where there are only two actions $\{a,a'\}$, and where the analytic values of $Q_*(s,a)$ and $Q_*(s,a')$ are known. The only unknown is which action a or a' is the optimal one. In this case it is clear that access to the optimal policy resolves the question.

As an example of such an environment, consider the SL-DLP family of environments from [150]. In each episode, the agent needs to classify one random sample from a set of samples corresponding to the DLP classification task from Section 5.2.1, where a correct (incorrect) classification yields a reward of 1 (-1). If we set $\gamma = 0$, the two possible Q-values for a given state and the two possible actions are simply the rewards corresponding to the result of the classification. To get the Q-value $Q_*(s,a)$, we query the policy $\pi_*(a|s)$ for the optimal action and assign the reward for a correct classification to the corresponding Q-value. (Note that we can also directly infer $Q_*(s,a')$ for the wrong action a' from this, as there are only two

distinct Q-values.) This can also be trivially extended to episodes with a horizon greater than one and $\gamma > 0$. After querying the policy for the optimal action given the initial state of the episode, the expected return is computed directly assuming optimal actions until the end of the episode is reached. I.e., we simply compute

$$Q_*(s_t, a_t) = \mathbb{E}_{\pi_*}\left[\sum_{k=0}^{H-1} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right]$$

for a_t given by the optimal policy, where all rewards are one from time step t onward. (For more details on settings with longer horizon and a discount factor larger than zero, and an analytic expression of the Q-values in these cases, see [150]).

In more general cases, the issue of approximation reduces to the problem of reaching the desired state s efficiently. When this is possible (i.e., it is possible to construct environments which allow this without becoming easy to learn), then so is estimating Q-values given an optimal policy. Note that for all of the above, the same caveat as in [150] applies, namely that this method of obtaining optimal Q-values does not resemble Q-learning in the sense that we use a tabular or DQN-type approach as shown in Section 3.2.2, and it is still an open question whether a rigorous quantum advantage can be shown in these settings for either policy-based RL or Q-learning.

5.3 Numerical results

In this section, we present results for our PQC model on two benchmark RL tasks from the OpenAI Gym [219], Frozen Lake v0 [223] and Cart Pole v0 [218] (see Figure 5.2). We ran an extensive hyperparameter search for both environments, and present our results for the best sets of hyperparameters. A detailed description of the hyperparameters we tuned and their best values can be found in Chapter 8. Our experiments were run with TensorFlow Quantum [224] and Cirq [225], the full code can be found on Github [221].

5.3.1 Frozen Lake

The Frozen Lake (FL) environment serves as an example for environments with a simple, discrete state space and with a reward structure that allows us to use an agent which performs measurements in the Z-basis to compute Q-values without

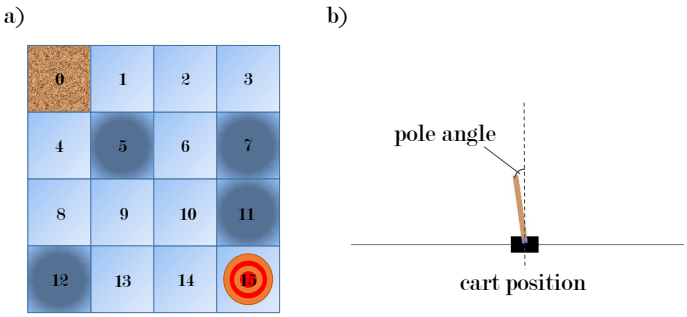
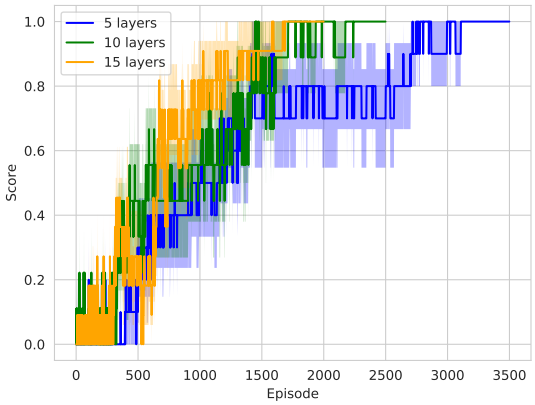


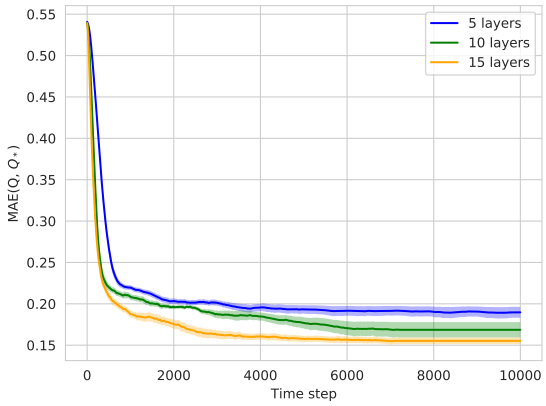
Figure 5.2: Gym environments solved by the quantum model. a) Frozen Lake environment, where an agent needs to learn to navigate from the top left of a grid to retrieve the Frisbee at the bottom right without falling into any of the holes (dark squares), b) Cart Pole environment, which consists of learning to balance a pole on a cart which moves left and right on a frictionless track.

the need for trainable weights to scale the output range. It consists of a 4x4 grid representing a frozen surface, where the agent can choose to move one step up, down, left or right. The goal is to cross the lake from the top left corner to the bottom right corner where the goal is located. However, some of the grid positions correspond to holes in the ice, and when the agent steps on them the episode terminates and it has to start again from the initial state. In each episode, the agent is allowed to take a maximum number of steps m_{max}. The episode terminates if one of the following conditions is met: the agent performs $m_{max} = 200$ steps, reaches the goal, or falls into a hole. For each episode in which the goal is reached the agent receives a reward of 1, and a reward of 0 otherwise. The environment is considered solved when the agent reaches the goal for 100 contiguous episodes. (See [223] for full environment specification.)

As the FL environment is discrete and the dimensions of the state and action spaces are small, there is no true notion of generalization in this environment, as all distinct state-action pairs are likely observed during training. On the other hand, generalization to unseen state-action pairs is one of the key reasons why function approximation was introduced to Q-learning. For this reason, environments like Frozen Lake are not a natural fit for these types of algorithms and we refrain from comparing to a classical function approximator. Note that we also refrain



(a) average scores



(b) mean absolute error with optimal Q-values

Figure 5.3: Agents with varying depth playing the Frozen Lake environment, and their closeness to the optimal Q-values. The environment is solved when the agent reaches the goal (receives a score of 1) for 100 contiguous episodes. a) Average score over 10 agents for circuits of depth 5, 10, and 15, respectively. All agents manage to solve the environment, higher circuit depth leads to lower time to convergence. Shaded area shows standard deviation from the mean. b) Mean absolute error between agents' Q-values and the optimal Q-values Q_* for all (s, a) pairs over time steps in episodes, where one time step corresponds to one transition in the environment. Shaded region shows standard error of the mean.

from comparing to the tabular approach, as this is (i) guaranteed to converge and (ii) not interesting beyond environments with very limited state and action spaces. However, this environment is interesting from another perspective: there are only 64 Q-values which we can compute exactly, and therefore we can directly compare the Q-values learned by our model to the optimal Q-values Q_*, which is not possible for the continuous-state Cart Pole environment that we study in Section 5.3.2.1. We show the difference between our agents' Q-values and the optimal Q-values during the course of training in Figure 5.3 b). Additionally, the FL environment serves as a nice example for environments where a PQC with simple measurements in the Z-basis can be used to solve a RL task, without requiring additional post-processing, as we describe below.

The FL environment has 16 states (one for each square on the grid) of which four are holes (marked as darker squares in Figure 5.2 a), and 4 actions (top, down, left, right). We encode each position on the grid as one of the computational basis states of a 4-qubit system, without use of trainable input data weights or data re-uploading. The optimal Q-values for each state-action pair can be computed as $Q_*(s,a) = \gamma^\beta$ (cf. Equation (3.16)), where β is the number of steps following the shortest path to the goal from the state s' that the agent is in after the transition (s,a). We will now motivate our choice of observables for the FL agent by studying the range the optimal Q-values can take. Note that these optimal Q-values are defined for the tabular case only, and serve as a reference for the Q-values we want our Q-function approximator to model. We know that only one transition, that from state 14 to the goal state 15, is rewarded. This corresponds to a Q-value $Q_*(14,R) = \gamma$. As the only other state adjacent to the goal (state 11) is a hole, no other transition in this environment is rewarded. Through the recursive Q-value update rule (see Equation (3.19)), all other Q-values depend on $Q_*(14,R)$, and are smaller due to the discount factor and the zero reward of all other transitions. In case of a function approximator, the Q-values may not be the same as the optimal values, but the relationship between $Q(14,R)$ and all other Q-values still applies as the update rule in Equation (3.19) changes values according to the observed reward and discounted expected reward. That is, if the function approximator outputs values that match the range of optimal Q-values and is not fundamentally limited in the updates that can be performed to it, the relationship above can be replicated. This means that we have an upper bound on the range of Q-values that we want to model which only depends on $\gamma \leq 1$ and stays constant over all episodes. Therefore we do not expect that Q-values need to become larger than γ

for our agent to solve the environment, and only become larger in practice if the initialization of our model happens to yield higher values for some state-action pairs. Motivated by this, we represent the Q-values for the four actions as the expectation values of a measurement with the operator Z_i for each of the four qubits $i \in \{1, \ldots, 4\}$, which we scale to lie between $[0, 1]$ instead of $[-1, 1]$. Note that even when parameter initialization yields Q-values higher than the largest optimal Q-value, they will still be close to this value as both optimal Q-values and those of our model are upper-bounded by 1. Figure 5.3 a) shows the average scores of ten agents, each configuration trained with a circuit depth of 5, 10, and 15 layers, respectively. All agents manage to solve the environment, and the time to convergence decreases as the number of layers increases. Figure 5.3 b) shows the averaged mean absolute error (MAE) between the optimal Q-values and the Q-values produced by the agents at each time step during training. The agents trained on circuits of depth 15 reach the lowest values and converge earlier to an average MAE that is roughly 0.05 lower than that of the agents trained on a circuit of depth 5. This illustrates that as we increase the complexity of the function approximator, the optimal Q-values can be more accurately modelled. However, the improvement between 10 and 15 layers is relatively small compared to that between 5 and 10 layers, similar to a saturation in performance w.r.t. number of parameters found in classical deep RL [220]. We will study this type of scaling behaviour more in-depth and compare it to that of NNs in Section 5.3.2.1. At the same time, we see that producing optimal Q-values is not necessary to solve an environment, as we argue in Section 3.2.2. In the following section, we study an environment where we are not able to compute the optimal Q-values analytically due to the continuous state space, but where we compare to a classical approach to assess the quality of our solution instead.

5.3.2 Cart Pole

In the previous section, we have seen that for an environment with discrete state space and a reward function that results in an upper bound of Q-values of one, a simple PQC without enhanced data encoding our readout strategies suffices to solve the environment. Now we turn to an environment that is slightly more complex: the continuous state space necessitates a more evolved data encoding strategy, while the reward function results in Q-values that far exceed the range of a Z-basis measurement. In the Cart Pole v0 environment, an agent needs to learn to balance a pole upright on a cart that moves on a frictionless track. The action

space consists of two actions: moving the cart left and right. Its state space is continuous and consists of the following variables: cart position, cart velocity, pole angle, and pole velocity at the tip. The cart position is bounded between ± 2.4, where values outside of this range mean leaving the space that the environment is defined in and terminating the episode. The pole angle is bounded between ± 41.8 degrees. The other two variables can take infinite values, but are bounded in practice by how episode termination is defined. An episode terminates if the pole angle is outside of ± 12 degrees, the cart position is outside of ± 2.4, or the agent reaches the maximum steps per episode $m_{max} = 200$. For each step of the episode (including the terminal step) the agent receives a reward of one. At the beginning of each episode, the four variables of the environment state are randomly initialized in a stable state within the range [-0.05, 0.05]. The episode score is computed as the cumulative reward of all steps taken in the episode. The environment is solved when the average score of the last 100 episodes is ≥ 195. (See [218, 109] for full environment specification.)

As in Section 5.3.1, we now motive our choice of observables depending on how rewards are received in this environment. For this, we recall that a Q-value gives us the expected return for a given state-action pair,

$$Q_\pi(s,a) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

Cart Pole is an episodic environment with a maximum number of time steps $H = 200$ in the version of the environment we study here, so the Q-value following optimal policy π_* from a stable state s is

$$Q_*(s,a) = \sum_{k=0}^{H-1} \gamma^k.$$

When following an arbitrary policy π and starting in a random stable state of the environment, the Q-value is

$$Q_\pi(s,a) = \sum_{k=0}^{h-1} \gamma^k,$$

where $h \leq H$ is the length of the episode which is determined by the policy. The longer the agent balances the pole, the higher h, with $h = H$ the maximum number of steps allowed in an episode. When not considering random actions taken by the ϵ-greedy policy, h depends solely on the performance of the agent, which changes

as the agent gets better at balancing the pole. Consequently, the Q-values we want to approximate are lower bounded by the minimum number of steps it takes to make the episode terminate when always picking the wrong action (i.e., the pole doesn't immediately fall by taking one false action alone), and upper bounded by the Q-values assuming the optimal policy, where $h = H$. We stress that this upper bound applies to the optimal policy in one episode only, and that in practice the upper bound of the magnitude of Q-values during training depends on the performance of the agent as well as the number of episodes played. Compared to the range of expectation values of computational basis measurements these values can become very high, e.g. for $\gamma = 0.99$ we get $\max Q_*(s,a) \approx 86$. Even when considering that Q-values need not necessarily be close to the optimal values to solve an environment, the range given by computational basis measurements is clearly too small compared to the frequency with which rewards are given and the number of episodes needed until convergence.

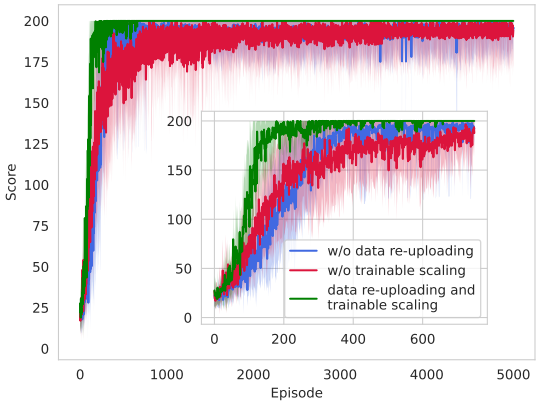
To give the agent the possibility to flexibly adjust it's output range, we add trainable weights on the output values as described in Section 5.1.2. The Q-values now take the form

$$Q(s,a) = \frac{\left\langle 0^{\otimes 4} \right| U_{\boldsymbol{\theta}}(s)^\dagger O_a U_{\boldsymbol{\theta}}(s) \left| 0^{\otimes 4} \right\rangle + 1}{2} \cdot w_{o_a}, \tag{5.10}$$

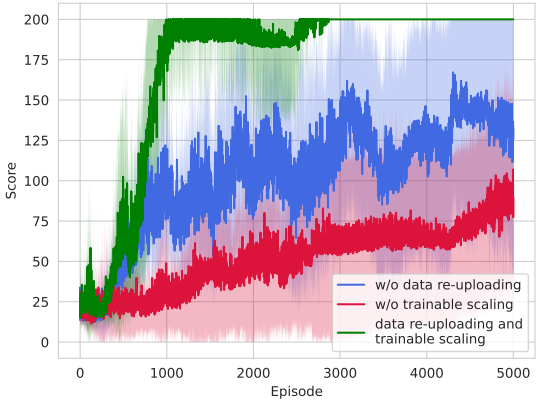
where $O_{a=L} = Z_1 Z_2$ and $O_{a=R} = Z_3 Z_4$ are Pauli-ZZ operators on qubits $(1,2)$ and $(3,4)$ respectively, corresponding to actions left and right. To further improve performance, we also use data re-uploading and add trainable weights on the input values as described in Section 5.1.1.

5.3.2.1 Comparison of data encoding and readout strategies

To illustrate the effect of data re-uploading and trainable weights on the input and output values, we perform an ablation study and assess the impact of each of these enhancements on learning performance. To illustrate that our proposed architecture (i) performs better overall, and (ii) is less sensitive to changes in hyperparameters, we show results for the best set of hyperparameters that were found for a circuit of depth five, as well as a sub-optimal set of hyperparameters with which it is less easy for the agents to solve the Cart Pole environment. The hyperparameters we optimize over are: batch size, learning rates and update frequencies of the Q-value-generating model and the target model (cf. Section 3.2.2)



(a) average scores with varying data encoding strategies for best set of hyperparameters



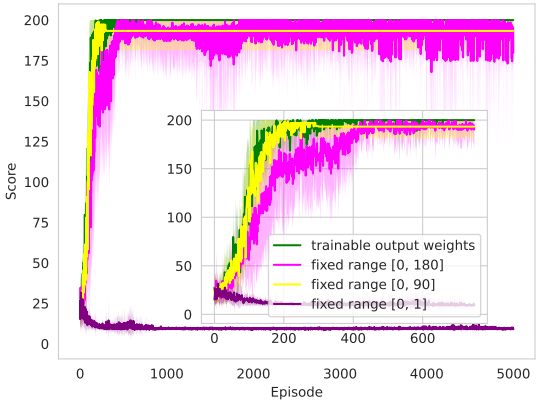
(b) average scores with varying data encoding strategies for sub-optimal set of hyperparameters

Figure 5.4: Comparison of data-encoding strategies for the optimal and one sub-optimal set of hyperparameters for agents training in the Cart Pole environment. The environment is solved when an agent has an average reward ≥ 195 for the past 100 episodes, after which training is stopped. Results are averaged over 10 agents each, where each agent consists of 5 layers of the circuit architecture depicted in Figure 5.1.

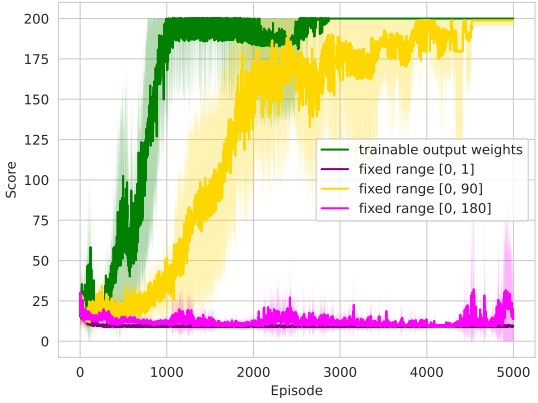
(see Chapter 8 for a detailed list of hyperparameter settings). Otherwise, we only vary the hyperparameters of the enhancements we want to study. The average performance of ten randomly initialized agents for each configuration is presented in Figure 5.4 and Figure 5.5. Once an agent solves the environment, we stop training and in the figures show the last encountered score for each agent in the averages (i.e., to form averages over equal lengths of episodes, we assume that each agent continues scoring the same value as it did in its last interaction with the environment).

Figure 5.4 a) and b) show the effects of varying data encoding strategies. While both data re-uploading and trainable weights on the input values alone do not produce agents that solve the environment in up to 5000 episodes for both the best and sub-optimal set of hyperparameters, combining both of these enhancements yields agents that solve Cart Pole in 3000 and 600 episodes at most on average, respectively. The fact that agents with trainable input weights and data re-uploading perform much better than those without, emphasizes the importance of matching the PQC's expressivity to the learning task at hand, as described in [169]. In Figure 5.5 a) and b), we compare agents with varying output ranges. Again, the green curves represent agents that are enhanced with a trainable weight corresponding to each Q-value that lets them flexibly adjust their output range during training, and these agents succeed with both sets of the remaining hyperparameters. The purple curves show agents with a fixed range of outputs of $[0, 1]$, all of which stay at an extremely low score during all 5000 episodes, as they fail to fit a good Q-function approximation regardless of hyperparameters. The yellow curves show agents with a fixed output range of maximally 90, which is motivated by the range of optimal Q-values. These agents also solve the environment on average, however, they are much more sensitive to parameter initialization and the remaining hyperparameters than agents with a trainable output range. The low final value of the yellow agents in Figure 5.5 a) is due to their last interaction with the environment achieving a relatively low score on average.

As described above, the magnitude of Q-values crucially depends on the agent's ability to balance the pole in each episode, and as a general trend it will increase over the course of training for agents that perform well. How large the final Q-values of a solving agent are therefore also depends on the number of episodes it requires until convergence, so a range which is upper bounded by 90 presumes agents that converge relatively quickly. Considering the range of final Q-values of agents in



(a) average scores with varying output ranges for best set of hyperparameters



(b) average scores with varying output ranges for sub-optimal set of hyperparameters

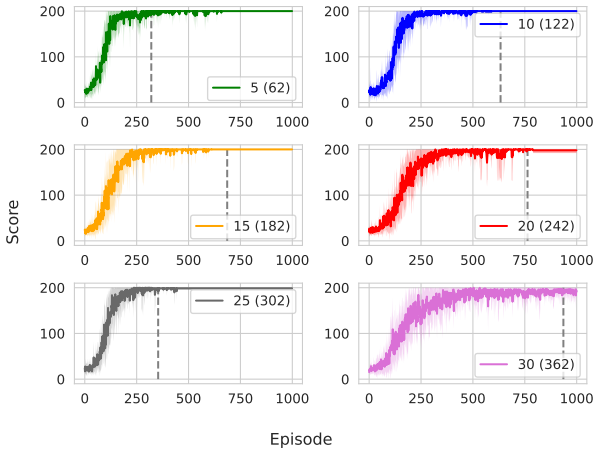
Figure 5.5: Comparison of different readout strategies of the same agents as in Figure 5.4 with the optimal and one sub-optimal set of hyperparameters.

the green curves, they can become as high as approximately 176 for agents that converge late. However, as we see for agents with a fixed output range of $[0, 180]$ (magenta curves), increasing the range to accommodate agents that converge later can lead to complete failure depending on the remaining hyperparameters.

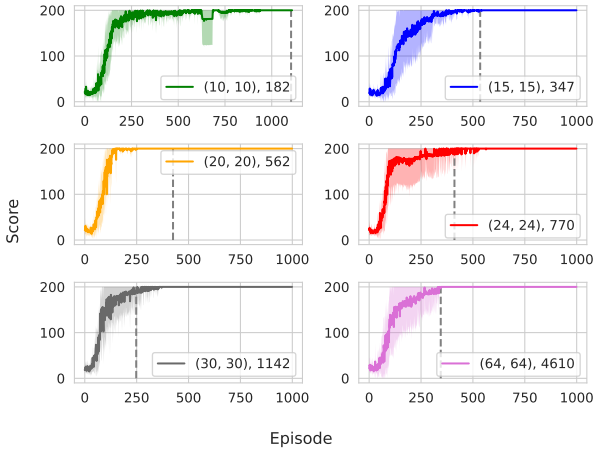
5.3.2.2 Comparison to the classical DQN algorithm

In addition to investigating the effects of varying data encoding and readout strategies, we compare the performance of our PQC model to that of the standard DQN algorithm that uses a NN as a function approximator. We do this for varying numbers of parameters for both the PQC and NN, and study how performance changes as the number of parameters increases. Note that because environments are strictly defined with a fixed number of input state variables, we cannot change the number of qubits arbitrarily for a certain environment. Studying varying system sizes in terms of qubits requires either artificially adjusting the data encoding to fit a certain number of qubits, or studying completely different environments all together. Therefore we focus on studying different model sizes in terms of number of parameters here. Additionally, the standard approach to increase model performance in supervised and unsupervised learning in the classical and quantum literature alike is often to add more parameters. However, it has been shown that this strategy does often not lead to success in classical deep RL due to the instability of training larger networks [220]. Instead, it is much more important to find good settings of hyperparameters (including the random initialization of model parameters), and it is preferable to use models which are less sensitive to changes in these settings.

To study whether this effect is also present when the function approximator is a PQC, we compare agents with up to 30 layers of the hardware efficient ansatz depicted in Figure 5.1. All agents use the enhancements which have shown to yield good performance in Figure 5.4 and Figure 5.5, namely data re-uploading and trainable input and output weights. The other hyperparameters that yield to the best performance for each depth are found through an extensive hyperparameter search and include the three different learning rates (Q-network, input and output weights), batch size, and update frequency of the Q-network and target network (see Chapter 8 for detailed settings). Figure 5.6 a) shows the average performance over 10 quantum agents of each configuration. We indeed observe that increasing the number of parameters is only efficient up to a certain point, after which



(a) PQCs, labels show: #layers (#parameters)



(b) NNs, labels show: (#units in hidden layer 1, 2), #parameters

Figure 5.6: Comparison of classical and quantum agents with varying numbers of parameters in the Cart Pole environment. Each sub-figure contains results averaged over ten agents, and the vertical dashed line marks the average number of episodes until solving the environment. We performed a hyperparameter optimization for each parameter configuration separately, and show the best setting for each. (See Chapter 8 for all settings and a list of hyperparameters that were searched over.)

additional layers lead to slower convergence. The best-performing configuration on average is a PQC with 25 layers and 302 parameters, which takes 500 episodes on average to solve the Cart Pole environment.

To investigate the performance of the classical DQN algorithm which uses a NN as the function approximator, we compare NNs with two hidden layers with varying numbers of units. As simply increasing the depth of the NNs has not been beneficial in a RL setting, it has been proposed to use shallow networks with increased width instead [220]. Therefore we keep the depth of our NNs fixed at two, and vary the width by changing the number of units in each hidden layer. This configuration is also inspired by well-performing agents on the official OpenAi Gym leaderboard [226].[1] We make the same observation for the NNs in Figure 5.6 b) as we did for the PQCs – increasing the number of parameters does not necessarily improve performance. The best-performing NN is one with 20 units in each of its hidden layers, which yields a network with 562 parameters overall that solves the Cart Pole environment in 250 episodes on average. Comparing the configurations of PQC and NN that perform best on average, the best NN configuration takes roughly half as many episodes on average to solve Cart Pole than the best PQC, and does this with roughly twice as many parameters. Notably, the PQCs seem to suffer more from an instability during training as the number of parameters is increased than the NNs do. We also show a comparison of the best individual (not averaged) PQC and NN agents in Figure 5.7. Here, the gap is relatively small: the best PQC (5 layers, 62 parameters) takes 206 episodes to solve Cart Pole, while the best NN (2 hidden layers with 30 units each, 1142 parameters) takes 186 episodes.

Finally, we note that unlike for the Frozen Lake environment, it is not straightforward to compute optimal Q-values for Cart Pole as its state space is continuous. A trained model that is known to implement the optimal policy (i.e., correct ordering of Q-values for all (s, a)-pairs) could be used as a baseline to compare other models to, but the magnitudes of Q-values can highly vary even among agents that solve the environment so this comparison will not provide much insight, which is why we refrain from including it here. Nonetheless, we provide a visualization of the Q-values learned by one of our best-performing quantum models in Chapter 8. We observe that these Q-values have a maximum value close to what we expected from an optimal agent (i.e., 86).

[1] However, we note that it is hard to find reliable benchmarks on the Cart Pole environment in classical literature, as it was already too small to be considered in state-of-the-art deep learning when the DQN algorithm was introduced in [119].

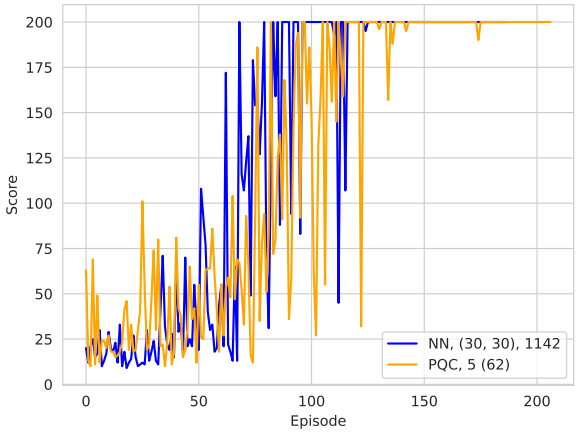


Figure 5.7: Best PQC and NN from the configurations we study in Figure 5.6. The best PQC (orange, 5 layers, 62 parameters) takes 20 episodes longer to solve Cart Pole than the best NN (blue, two hidden layers with 30 units each, 1142 parameters).

5.4 Conclusion

In this chapter, we have proposed a quantum model for deep Q-learning which can encode states of environments with discrete and continuous state spaces. We have illustrated the importance of picking the observables of a quantum model such that it can represent the range of optimal Q-values that this algorithm should learn to approximate. One crucial difference between PQCs and classical methods based on NNs, namely the former's restricted range of output values defined by its measurement operators, was identified as a major impediment to successfully perform Q-learning in certain types of environments. Based on the range of optimal Q-values, we illustrate how an informed choice can be made for the quantum model's observables. We also introduce trainable weights on the observables of our model to achieve a flexible range of output values as given by a NN and empirically show the benefit of this strategy on the Cart Pole environment by performing ablation studies. Our results show that a trainable output range can lead to better performance as well as lower sensitivity to the choice of hyperparameters and random initialization of parameters of the model. We also perform ablation studies on a number of data encoding techniques which enhance the expressivity of PQCs, namely data re-uploading [168] and trainable

weights on the input [168, 150]. We show the benefit of combining both approaches in the Cart Pole environment, where any of the two encoding strategies on its own does not suffice to reliably solve the environment. Our results illustrate the importance of architectural choices for QML models, especially for a RL algorithm as Q-learning that has very specific demands on the range of output values the model can produce.

Additionally, we investigated whether recent results in classical deep Q-learning also hold for PQC-based Q-learning, namely that increasing the number of parameters in a model might lead to lower performance due to instability in training. To evaluate the performance of our model compared to the classical approach where the same DQN algorithm is used with a NN as the Q-function approximator, we study the performance of a number of classical and quantum models with increasing numbers of parameters. Our results confirm that PQC-based agents behave similarly to their NN counterparts as the number of parameters increases. Performance only increases up to a certain point and then declines afterward. We find that in both cases, the hyperparameter settings (and in case of the PQC data encoding and readout strategies) are the determining factors for a model's success much more than the number of parameters. This is in contrast to previous results for training PQCs on supervised and unsupervised learning tasks, where additional layers are likely to increase performance [213, 212, 139]. The effect that an increased number of parameters hampers performance in Q-learning also seems to be more prominent in PQCs than in NNs, which raises the question whether we need additional mechanisms to increase learning stability in this setting than the ones from classical literature.

In addition to our numerical studies, we also investigated whether a recent proof of quantum advantage for policy gradient RL agents [150] implies a separation of classical and quantum Q-learning agents as well. We show how optimal Q-values for state-action pairs can be efficiently computed given access to an optimal policy in the SL-DLP family of environments from [150]. We explain additional requirements on the structure of states in a given environment that need to be fulfilled to allow efficiently inferring optimal Q-values from optimal policies in more general environments. However, the separation results in [150] only guarantee that quantum learners can be constructed in general, and not that the optimal policy can be learned by policy gradient methods directly. It is an interesting open question if a separation between classical and quantum agents can also be proven

for learning algorithms that use policy gradient or Q-value updates as shown in Equation (3.19). This opens up the path to future investigations of possible quantum advantages of these types of quantum agents in relevant settings.

CHAPTER 6

Equivariant quantum circuits for learning on weighted graphs

In Chapter 5, we described that the three key architectural choices one has to make for a variational QML model are i) the data encoding technique, ii) the circuit ansatz, and iii) the observable to measure. In that chapter, we have focused on how to encode data and pick suitable observables for a variational Q-learning algorithm. In this chapter, we turn to the question of how to design ansatzes that are tailored to a specific learning problem.

It is known that the right choice of ansatz is of key importance for the performance of these models. Much work has been dedicated to understand how circuits have to be structured to address problems in optimization [59, 227] or chemistry [71, 228]. For QML however, it is largely unknown which type of ansatz should be used for a given type of data. In absence of an informed choice, general architectures as the hardware-efficient ansatz [170] are often used [120]. It is known that ansatzes with randomly selected structure scale badly as the width and depth of the circuit grows, most prominently because of the barren plateau phenomenon [229, 52, 47] where the gradients of a PQC vanish exponentially as the system size grows and thus render training impossible, as we have described in detail in Section 2.2.1.2. This situation can be compared to the early days of NNs, where fully connected feedforward NNs were the standard architecture. These types of NNs also suffer from trainability issues that prevent their large-scale usage [230]. Recent breakthroughs in deep learning were in part possible because more efficient architectures that are directly motivated by the training data structure have been developed [29, 30, 231]. In fact, a whole field that studies the mathematical properties of successful NN architectures has emerged in the past decade, known as *geometric deep learning*.

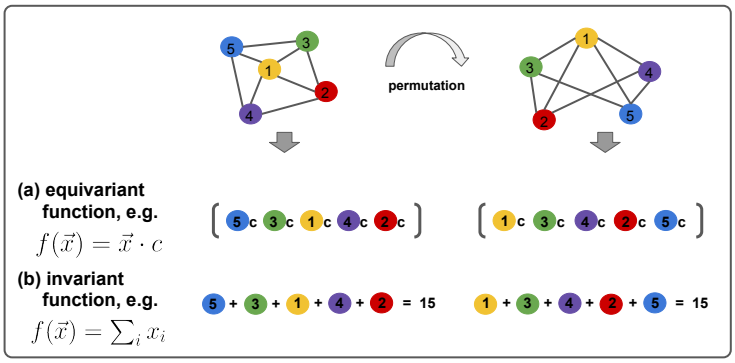


Figure 6.1: Depiction of two functions that respect important symmetries of graphs: a) The permutation *equivariant* function will yield the same output values for each graph permutation, but reordered according to the reordering of nodes. The above example shows a simple function that takes node features as an input and multiplies them with a constant. b) An *invariant* function will yield the same output, regardless of the permutation. The above example shows a simple function that takes node features as input and computes their sum. Which type of symmetry is preferable depends on the task at hand.

This field studies the properties of common NN architectures, like convolutional NNs or graph NNs, through the lens of group theory and geometry and provides an understanding of why these structured types of models are the main drivers of recent advances in deep learning. The success of these models can largely be attributed to the fact that they preserve certain symmetries that are present in the training data. Graph NNs, for example, take graph-structured data as input and their layers are designed such that they respect one of two important graph symmetries: invariance or equivariance under permutation of vertices [232], as depicted in Figure 6.1. Graph-structured data is ubiquitous in real-world problems, for example to predict properties of molecules [30] or to solve combinatorial optimization problems [108]. Even images can be viewed as special types of graphs, namely those defined on a lattice with nearest-neighbor connections. This makes graph NNs applicable in a multitude of contexts, and motivated a number of works that study quantum versions of these models [149, 233, 234, 172]. However, the key questions of how to design symmetry-preserving ansatzes motivated by a concrete input data structure and how these ansatzes perform compared to those that are structurally unrelated to the given learning problem remain open.

In this work, we address these open questions by introducing a symmetry-preserving ansatz for learning problems where the training data is given in form of weighted graphs, and study its performance both numerically and analytically. To do this, we extend the family of ansatzes from [172] to incorporate weighted edges of the input graphs and prove that the resulting ansatz is equivariant under node permutations. To evaluate this ansatz on a complex learning task where preserving a given symmetry can yield a significant performance advantage, we apply it in a domain where classical graph NNs have been used extensively: neural combinatorial optimization (NCO) [108]. In this setting, a model is trained to solve instances of a combinatorial optimization problem. Namely, we train our proposed ansatz to find approximate solutions to the Traveling Salesperson Problem (TSP). We numerically compare our ansatz to three non-equivariant ansatzes on instances with up to 20 cities (20 qubits), and show that the more the equivariance property of the ansatz is broken, the worse performance becomes and that a simple hardware-efficient ansatz completely fails on this learning task. Additionally, we analytically study the expressivity of our model at depth one, and show under which conditions there exists a parameter setting for any given TSP instance of arbitrary size for our ansatz that produces the optimal tour with the learning scheme that is applied in this work.

The neural combinatorial optimization approach presented in this work also provides an alternative method to employ near-term quantum computers to tackle combinatorial optimization problems. As problem instances are directly encoded into the circuit in form of graphs without the need to specify a cost Hamiltonian, this approach is even more frugal than that of the quantum approximate optimization algorithm (QAOA) [59] in terms of the requirement on the number of qubits and connectivity in cases where the problem encoding is non-trivial. For the TSP specifically, standard Hamiltonian encodings require n^2 variables where n is the number of cities (or $n\log(n)$ variables at the cost of increased circuit depth) [235], whereas our approach requires only n qubits and two-body interactions. We do note that the theoretical underpinnings and expected guarantees of performance of our method are very different and less rigorous than those of the QAOA, so the two are hard to compare directly. However, we establish a theoretical connection to the QAOA based on the structure of our ansatz, and in addition numerically compare QAOA performance on TSP instances with 5 cities to the performance of the proposed neural combinatorial optimization approach. We find that our ansatz at depth one outperforms the QAOA even at depth up to three. From a pragmatic

point of view, linear scaling in qubit numbers w.r.t. number of problem variables, as opposed to e.g. quadratic scaling as in the case of the TSP, dramatically changes the applicability of quantum algorithms in the near- to mid-term.

Our work illustrates the merit of using symmetry-preserving ansatzes for QML on the example of graph-based learning, and underlines the notion that in order to successfully apply variational quantum algorithms for ML tasks in the future, the usage of ansatzes unrelated to the problem structure, which are popular in current QML research, is limited as problem sizes grow. This work motivates further study of "geometric quantum learning" in the vein of the classical field of geometric deep learning, to establish more effective ansatzes for QML, as these are a prerequisite to efficiently apply quantum models on any practically relevant learning task in the near-term.

6.1 Geometric learning - quantum and classical

Learning approaches that utilize geometric properties of a given problem have lead to major successes in the field of ML, such as AlphaFold for the complex task of protein folding [30, 31] and have become an increasingly popular research field over the past few years. Arguably, the prime example of a successful geometric model is the convolutional NN (CNN), which has been developed at the end of the 20th century in an effort to enable efficient training of image recognition models [104]. Since then, it has been shown that one of the main reasons that CNNs are so effective is that they are translation invariant: if an object in a given input image is shifted by some amount, the model will still "recognize" it as the same object and thus effectively requires fewer training data [100]. While CNNs are the standard architecture used for images, symmetry-preserving architectures have also been developed for time-series data in the form of recurrent NNs [236], and for graph data with GNNs [107]. GNNs have seen a surge of interest in the classical machine learning community in the past decade [107, 232]. They are designed to process data that is presented in graph form, like social networks [107], molecules [106], images [237] or instances of combinatorial optimization problems [108].

The first attempt to implement a geometric learning model in the quantum realm was made with the quantum convolutional NN in [55], where the authors introduce a translation invariant architecture motivated by classical convolutional NNs. Approaches to translate the GNN formalism to QNNs were taken in [149], where

input graphs are represented in terms of a parametrized Hamiltonian, which is then used to prepare the ansatz of a quantum model called a *quantum graph neural network* (QGNN). While the approach in [149] yields promising results, this work does not take symmetries of the input graph into account.[1] The authors of [233] introduce the so-called *quantum evolution kernel*, where they devise a graph-based kernel for a quantum kernel method for graph classification. Again, their ansatz is based on alternating layers of Hamiltonians, where one Hamiltonian in each layer encodes the problem graph, while a second parametrized Hamiltonian is trained to solve a given problem. A proposal for a quantum graph convolutional NN was made in [234], and the authors of [238] propose directly encoding the adjacency matrix of a graph into a unitary to build a quantum circuit for graph convolutions. While all of the above works introduce forms of structured QML models, none of them study their properties explicitly from a geometric learning perspective or relate their performance to unstructured ansatzes.

The authors of [172] take the step to introduce an equivariant model family for graph data and generalize the QGNN picture to so-called *equivariant quantum graph circuits* (EQGCs). EQGCs are a very broad class of ansatzes that respect the connectivity of a given input graph. The authors of [172] also introduce a subclass of EQGCs called *equivariant quantum Hamiltonian graph circuits* (EH-QGCs), that includes the QGNNs by [149] as a special case. EH-QGCs are implemented in terms of a Hamiltonian that is constructed based on the input graph structure, and they are explicitly equivariant under permutation of vertices in the input graph. The framework that the authors of [172] propose can be seen as a generalization of the above proposals. Different from the above proposals, EQGCs use a post-measurement classical layer that performs the functionality of an aggregation function as those found in classical GNNs. In classical GNNs, the aggregation function in each layer is responsible for aggregating node and edge information in an equivariant or invariant manner. Popular aggregation functions are sums or products, as they trivially fulfill the equivariance property. In the case of EQGCs, there is no aggregation in the quantum circuit, and this step is offloaded to a classical layer that takes as input the measurements of the PQC. Additionally, the EQGC family is defined over unweighted graphs and only considers the adjacency matrix of the underlying input graph to determine the connectivity of the qubits. The authors of [172] also show that their EQGC outperforms a standard message

[1] However, in an independent work prepared at the time of writing this manuscript, one of the authors of [149] shows that one of their proposed ansatzes is permutation invariant [173].

passing neural network on a graph classification task, and thereby demonstrate a first separation of quantum and classical models on a graph-based learning task. A work on invariant quantum machine learning models was published by the authors of [173]. They prove for a number of selected learning tasks whether an invariant quantum machine learning model for specific types of symmetries exists. Their work focuses on group invariance, and leaves proposals for NISQ-friendly equivariant quantum models as an open question.

Our proposal is most closely related to EH-QGCs, but with a number of deviations. First, our model is defined on weighted graphs and can therefore be used for learning tasks that contain node as well as edge features. Second, the initial state of our model is always the uniform superposition, which allows each layer in the ansatz to perform graph feature aggregation via sums and products of node and edge features, as discussed in Section 6.3. Third, we do not require a classical post-processing layer, so our EQC model is purely quantum. Additionally, in its simplest form as used in this work, the number of qubits in our model scales linearly with the number of nodes in the input graph, while the depth of each layer depends on the graph's connectivity, and therefore it provides one answer to the question of a NISQ-friendly equivariant quantum model posed by [173].

6.2 Neural combinatorial optimization with reinforcement learning

The idea behind NCO is to use a ML model to learn a heuristic for a given optimization problem based on data. When combined with RL, this data manifests in form of states of an environment, while the objective is defined in terms of a reward function, as we described in Section 3.2. To do NCO in this setting, the reward function is defined such that maximizing the expected return (see Equation (3.16)) corresponds to finding the optimum of the given combinatorial optimization problem. In this work, we use a Q-learning agent as introduced in Section 5.1 as the learning model in this NCO scheme, and train it in an evironment that is specified to solve instances of the TSP.

6.2.1 Solving the Traveling Salesperson Problem with reinforcement learning

To evaluate the performance gains of an ansatz that respects certain symmetries relevant to the problem at hand, we apply our model to a practically motivated learning task on graphs. The TSP is a low-level abstraction of a problem commonly encountered in mobility and logistics: given a list of locations, find the shortest route that connects all of these locations without visiting any of them twice. Formally, given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and weighted edges $\mathcal{E}$, the goal is to find a permutation of the vertices such that the resulting tour length is minimal, where a tour is a cycle that visits each vertex exactly once. A special case of the TSP is the 2D Euclidean TSP, where each node is defined in terms of its x and y coordinates in Euclidean space, and the edge weights are given by the Euclidean distance between these points. In this work, we deal with the symmetric Euclidean TSP on a complete graph, where the edges in the graph are undirected. This reduces the number of possible tours from $n!$ to $\frac{(n-1)!}{2}$. However, even in this reduced case the number of possible tours is already larger than 100k for instances with a modest number of ten cities, and the TSP is a well-known NP-hard problem.

To solve this problem with a RL approach, we follow the strategy introduced in [239]. In this work, a classical GNN is used to solve a number of combinatorial optimization problems on graphs. The authors show that this approach can outperform dedicated approximation algorithms defined for the TSP, like the Christofides algorithm, on instances of up to 300 cities. One episode of this learning algorithm for the TSP can be seen in Figure 6.3, and a detailed description of the learning task as implemented in our work is given in Section 6.4.1.

6.2.2 Solving the TSP with the QAOA

The quantum NCO scheme that we propose in this work poses an alternative to the well known quantum approximate optimization algorithm (QAOA), and for this reason we provide a comparison to this algorithm in addition to the comparison to non-equivariant ansatzes. The QAOA is implemented as a PQC by a Trotterization of Adiabatic Quantum Computation (AQC) [59]. In general, for AQC, we consider a starting Hamiltonian H_0, for which both the formulation and the ground state are well known, and a final Hamiltonian H_P, that encodes the combinatorial

optimization problem to be solved. The system is prepared in the ground state of the Hamiltonian H_0 and then it is evolved according to the time-dependent Hamiltonian:

$$H(t) := (1 - s(t))H_0 + s(t)H_P,$$

where $s(t)$ is a real function called annealing schedule that satisfies the boundary conditions: $s(0) = 0$ and $s(T) = 1$, with T the duration of the evolution. To implement this as a quantum circuit we use the following approximation:

$$e^{A+B} \approx \left(e^{\frac{A}{r}} e^{\frac{B}{r}}\right)^r, r \to +\infty, \tag{6.1}$$

which is knwon as the Trotter-Suzuki formula. By using this formula to approximate the evolution according to $H(t)$ and by parameterizing time we obtain:

$$e^{-i\beta_p H_0} e^{-i\gamma_p H_P} \cdots e^{-i\beta_1 H_0} e^{-i\gamma_1 H_P}. \tag{6.2}$$

All of these matrices are unitary since the Hamiltonians in the argument of the exponential are all Hermitian. We define a parameter p (integer known as the depth, or level) of QAOA which has the same role as r in Equation (6.1). Increasing the depth p adds additional layers to the QAOA circuit, and thus more closely approximates the H(t) [59].

In QAOA, all qubits are initialized to $|+\rangle^{\otimes_n}$, which is the ground state of $H_0 = \sum_i \sigma_x^{(i)}$. Alternating layers of H_p and H_0 are added to the circuit (p times), parameterized by γ and β as defined in Equation (6.2). The values of γ and β are found by minimizing the expectation value of H_p, and thus approximate the optimal solution to the original combinatorial optimization problem. When using QAOA, we do not solve the TSP directly, but a QUBO representation of this problem. This representation is well-known, and can be found in [235]:

$$\sum_{(i,j)\in\mathcal{E}} \sum_{t=1}^{N} \frac{\varepsilon_{i,j}}{W} x_{i,t} x_{j,t+1} + \sum_{i\in\mathcal{V}} \left(1 - \sum_{t=1}^{N} x_{i,t}\right)^2 + \\ + \sum_{t=1}^{N} \left(1 - \sum_{i\in\mathcal{V}} x_{i,t}\right)^2 + \sum_{(i,j)\notin\mathcal{E}} \sum_{t=1}^{N} x_{i,t} x_{j,t+1}.$$

Here, $\varepsilon_{i,j}$ are the distances between two nodes $i, j \in \mathcal{V}$ and $W := \max_{(i,j)\in\mathcal{E}} \varepsilon_{i,j}$. The variables $x_{v,t}$ are binary decision variables denoting whether node v is visited at step t. We optimize the β and γ parameters for $p = 1$ by performing a uniform random search over the space $[0, 2\pi]^2$, and selecting the best configuration found.

6.3 Equivariant quantum circuit

In this section, we formally introduce the structure of our equivariant quantum circuit (EQC) for learning tasks on weighted graphs that we use in this work. Examples of graph-structured data that can be used as input in this type of learning task are images [231], social networks [240] or molecules [30]. In general, when learning based on graph data, there are two sets of features: node features and edge features. Depending on the specific learning task, it might be enough to use only one set of these features as input data, and the specific implementation of the circuit will change accordingly. As mentioned above, an example of an ansatz for cases where encoding node features suffices is the family of ansatzes introduced in [172]. In our case, we use both node and edge features to solve TSP instances. In case of the nodes, we encode whether a node (city) is already present in the partial tour at time step t to inform the node selection process described later in Definition 6.2. For the edges, we simply encode the edge weights of the graph as these correspond to the distances between nodes in the TSP instance's graph. In this work, we use one qubit per node in the graph, but in general multiple qubits per node are also possible. We discuss the details of this in **??**. We now proceed to define the ansatz in terms of encoding node information in form of $\boldsymbol{\alpha}$ (see Definition 6.1) and edge information in terms of the weighted graph edges $\varepsilon_{ij} \in \mathcal{E}$. For didactic reasons we relate the node and edge features to the concrete learning task that we seek to solve in this work, however, we note that this encoding scheme is applicable in the context of other learning tasks on weighted graphs as well.

6.3.1 Ansatz structure and equivariance

Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with node features $\boldsymbol{\alpha}$ and weighted edges $\mathcal{E}$, and trainable parameters $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^p$, our ansatz at depth p is of the following form

$$|\mathcal{E}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p \quad = \quad U_N(\boldsymbol{\alpha}, \beta_p) U_G(\mathcal{E}, \gamma_p) \quad \ldots \quad U_N(\boldsymbol{\alpha}, \beta_1) U_G(\mathcal{E}, \gamma_1) \, |s\rangle \,, \tag{6.3}$$

where $|s\rangle$ is the uniform superposition of bitstrings of length n,

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \,, \tag{6.4}$$

$U_N(\boldsymbol{\alpha}, \beta_j)$ with $\mathrm{Rx}(\theta) = e^{-i\frac{\theta}{2}X}$, is defined as

$$U_N(\boldsymbol{\alpha}, \beta_j) = \bigotimes_{l=1}^{n} \mathrm{Rx}(\alpha_l \cdot \beta_j), \tag{6.5}$$

and $U_G(\mathcal{E}, \gamma_j)$ is

$$U_G(\mathcal{E}, \gamma_j) = \exp(-i\gamma_j H_{\mathcal{G}}) \tag{6.6}$$

with $H_{\mathcal{G}} = \sum_{(i,j)\in\mathcal{E}} \varepsilon_{ij}\sigma_z^{(i)}\sigma_z^{(j)}$ and $\mathcal{E}$ are the edges of graph $\mathcal{G}$ weighted by ε_{ij}. A 5-qubit example of this ansatz can be seen in Figure 6.2.

For $p = 1$, we have

$$\begin{aligned}
|\mathcal{E}, \boldsymbol{\alpha}, \beta, \gamma\rangle_1 &= U_N(\boldsymbol{\alpha}, \beta) U_G(\mathcal{E}, \gamma) |s\rangle \\
&= \frac{1}{\sqrt{2^n}} \\
&\cdot \sum_{x\in\{0,1\}^n} \underbrace{\left(\cos\frac{\alpha_1\beta}{2} + \cdots - i\sin\frac{\alpha_l\beta}{2} - \cdots - i\sin\frac{\alpha_n\beta}{2}\right)}_{\text{weighted bitflip terms}} \\
&\underbrace{\cdot \exp\left(\sum_{(i,j)\in\mathcal{E}} \mathrm{diag}(Z_iZ_j)_{|x\rangle} \cdot -i\,\frac{\pi}{2}\gamma\varepsilon_{ij}\right)}_{\text{edge weights}} |x\rangle\,,
\end{aligned} \tag{6.7}$$

where $\mathrm{diag}(Z_iZ_j)_{|x\rangle} = \pm 1$ is the entry in the matrix corresponding to each Z_iZ_j term, e.g., $I_1 \otimes \cdots \otimes Z_i \otimes I_k \otimes \cdots \otimes Z_j \otimes \cdots \otimes I_n$, corresponding to the basis state $|x\rangle$. (E.g., the first term on the diagonal corresponds to the all-zero state, and so on.) We see that the first group of terms, denoted *weighted bitflip terms*, is a sum over products of terms that encode the node features. In other words, in the one-qubit case we get a sum over sine and cosine terms, in the two-qubit case we get a sum over products of pairs of sine and cosine terms, and so on. The terms in the second part of the equation denoted *edge weights* is the exponential of a sum over edge weight terms. As we start in the uniform superposition, each basis state's amplitude depends on all node and edge features, but with different signs and therefore different terms interfering constructively and destructively for every basis state. This can be regarded as a quantum version of the aggregation functions used in classical graph NNs, where the k-th layer of a NN aggregates information over the k-local neighborhood of the graph in a permutation equivariant way [100]. In a similar fashion, the terms in Equation (6.7) aggregate node and edge information and become more complex with each additional layer in the PQC.

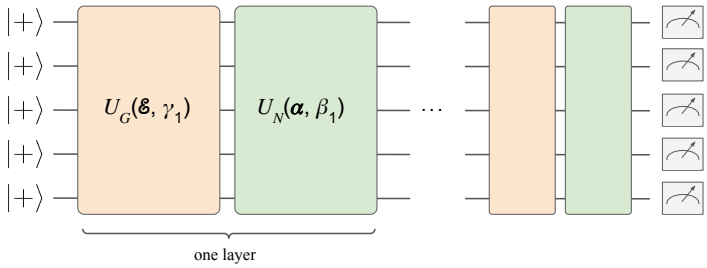


Figure 6.2: EQC used in this work. Each layer consists of two parts: the first part U_G encodes edge features, while the second part U_N encodes node features. Each of the two parts is parametrized by one parameter β_l, γ_l, respectively.

The reader may already have observed that this ansatz is closely related to an ansatz that is well-known in quantum optimization: that of the quantum approximate optimization algorithm [59]. Indeed, our ansatz can be seen as a special case of the QAOA, where instead of using a cost Hamiltonian to encode the problem, we directly encode instances of graphs and apply the "mixer terms" in Equation (6.5) only to nodes not yet in the partial tour. This correspondence will later let us use known results for QAOA-type ansatzes at depth one [241] to derive exact analytical forms of the expectation values of our ansatz, and use these to study its expressivity.

As our focus is on implementing an ansatz that respects a symmetry that is useful in graph learning tasks, namely an equivariance under permutation of vertices of the input graph, we now show that each part of our ansatz respects this symmetry.

Theorem 6.1 (Permutation equivariance of the ansatz)**.** *Let the ansatz of depth p be of the type as defined in Equation* (6.3) *with initial state $|+\rangle^{\otimes n}$ and parameters $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^p$, that represents an instance of a graph $\mathcal{G}$ with nodes $\mathcal{V}$ and the list of edges $\mathcal{E}$ with corresponding edge weights ε_{ij}, and node features $\boldsymbol{\alpha} \in \mathbb{R}^n$ with $n = |\mathcal{V}|$. Let σ be a permutation of the vertices in $\mathcal{V}$, $P_\sigma \in \mathbb{B}^{n \times n}$ the corresponding permutation matrix that acts on the weighted adjacency matrix A of $\mathcal{G}$, and $\tilde{P}_\sigma \in \mathbb{B}^{2^n \times 2^n}$ a matrix that maps the tensor product $|v_1\rangle \otimes |v_2\rangle \otimes \cdots \otimes |v_n\rangle$ with $|v_i\rangle \in \mathbb{C}^2$ to*

$\left|v_{\tilde{p}_\sigma(1)}\right\rangle \otimes \left|v_{\tilde{p}_\sigma(2)}\right\rangle \otimes \cdots \otimes \left|v_{\tilde{p}_\sigma(n)}\right\rangle$. *Then, the following relation holds,*

$$|\mathcal{E}_A, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p = \tilde{P}_\sigma \left|\mathcal{E}_{(P_\sigma^T A P_\sigma)}, P_\sigma^T \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\right\rangle_p, \tag{6.8}$$

where $\mathcal{E}_{(.)}$ denotes a specific permutation of the adjacency matrix A of the given graph. We call an ansatz that satisfies this property permutation equivariant.

Proof of Theorem 6.1. We want to prove that our ansatz is equivariant under permutations of the nodes of the input graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$,

$$|\mathcal{E}_{(P_\sigma^T A P_\sigma)}, P_\sigma^T \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p = \tilde{P}_\sigma |\mathcal{E}_A, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p. \tag{6.9}$$

For this, we have to prove that the unitaries that are used to construct the full circuit are permutation equivariant, i.e.,

$$\tilde{P}_\sigma U_G(\mathcal{E}_A, \gamma_l) \tilde{P}_\sigma^\dagger = U_G(\mathcal{E}_{(P_\sigma^T A P_\sigma)}, \gamma_l) \tag{6.10}$$

and

$$\tilde{P}_\sigma U_N(\boldsymbol{\alpha}, \beta_l) \tilde{P}_\sigma^\dagger = U_N(P_\sigma^T \boldsymbol{\alpha}, \beta_l). \tag{6.11}$$

We begin with the edge-encoding unitary U_G:

$$\tilde{P}_\sigma U_G(\mathcal{E}_A, \gamma_l) \tilde{P}_\sigma^\dagger = \tilde{P}_\sigma e^{-i\gamma_l H_\mathcal{G}} \tilde{P}_\sigma^\dagger \tag{6.12}$$

$$= e^{-i\gamma_l \tilde{P}_\sigma H_\mathcal{G} \tilde{P}_\sigma^\dagger} \tag{6.13}$$

$$= e^{-i\gamma_l H_{\mathcal{G}_{(P_\sigma^T A P_\sigma)}}} \tag{6.14}$$

$$= U_G(\mathcal{E}_{(P_\sigma^T A P_\sigma)}, \gamma_l), \tag{6.15}$$

where line (6.13) holds because for any unitary U we have $U e^{-iH_\mathcal{G}} U^\dagger = e^{-iUH_\mathcal{G}U^\dagger}$, and line (6.14) holds because $H_\mathcal{G} = \sum_{\varepsilon \in \mathcal{E}} \varepsilon_{ij} Z_i Z_j$ is defined completely through the adjacency matrix and the edge weights of the input graph $\mathcal{G}$, and $\tilde{P}_\sigma$ and P_σ are defined through permutations σ on the nodes of $\mathcal{G}$. Similarly, we get

$$\tilde{P}_\sigma U_N(\boldsymbol{\alpha}, \beta_l) \tilde{P}_\sigma^\dagger = \tilde{P}_\sigma \bigotimes_i^{|\mathcal{V}|} \mathrm{Rx}(\alpha_i, \beta_l) \tilde{P}_\sigma^\dagger \tag{6.16}$$

$$= \tilde{P}_\sigma \bigotimes_i^{|\mathcal{V}|} \exp\left(-i \frac{\alpha_i \beta_l}{2} X\right) \tilde{P}_\sigma^\dagger \tag{6.17}$$

$$= \bigotimes_i^{|\mathcal{V}|} \exp\left(-i \frac{\alpha_{\sigma^{-1}(i)} \beta_l}{2} X\right) \tag{6.18}$$

$$= U_N(P_\sigma^T \boldsymbol{\alpha}, \beta_l). \tag{6.19}$$

As each of the unitaries in the circuit is equivariant under permutations of the graph nodes, and the initial state is trivially permutation invariant $|+\rangle = \tilde{P}_\sigma |+\rangle$, we arrive at Equation (6.9). □

As mentioned before, our ansatz is closely related to those in [172], and the authors of this work prove permutation equivariance of unitaries that are defined in terms of unweighted adjacency matrices of graphs. In order to prove equivariance of our circuit, we have to generalize their result to the case where a weighted graph is encoded in the form of a Hamiltonian, and parametrized by a set of free parameters as described in Equation (6.3). In the non-parametrized case this is trivial, as edge weights and node features are directly permuted as a consequence of the permutation of the graph. When introducing parameters to the node and edge features, however, we have to make sure that the parameters themselves preserve equivariance, as the parameters are not tied to the adjacency matrix but to the circuit itself. To guarantee this, we make the parametrization itself permutation invariant by assigning one node and edge parameter per layer, respectively, and this makes us arrive at the QAOA-type parametrization shown in Equation (6.3). Another difference of our proof to that in [172] is that we consider a complete circuit including its initial state, instead of only guaranteeing that the unitaries that act on the initial state are permutation equivariant.

The above definition and proof are given in terms of a learning problem where we map one vertex to one qubit directly. However, settings where we require more than one qubit to encode node information are easily possible with this type of architecture as well. In order to preserve equivariance of our ansatz construction, three conditions have to hold: i) the initial state of the circuit has to be permutation invariant or equivariant, ii) the two-qubit gates used to encode edge weights have to commute, iii) the parametrization of the gates has to be permutation invariant. In the case where each vertex or edge is represented with more than just one gate per layer, one has freedom on how to do this as long as the above i)-iii) still hold. A simple example is when each vertex is represented by m qubits: i) the initial state remains to be the uniform superposition, ii) the topology of the two-qubit gates that represent edges has to be changed according to the addition of the new qubits, but ZZ-gates can still be used to encode the information, iii) the parametrization is the same as in the one-qubit-per-vertex case.

6.3.2 Trainability of ansatz

Our goal in this work is to introduce a problem-tailored ansatz for a specific data type that provides trainability advantages compared to unstructured ansatzes. One important question that arises in this context is that of barren plateaus, where the variance of derivatives for random circuits vanishes exponentially with the system size [229]. This effect poses challenges for scaling up circuit architectures like the hardware-efficient ansatz [170], as even at a modest number of qubits and layers a quantum model like this can become untrainable [44, 47, 52]. Therefore it is important to address the presence of barren plateaus when introducing a new ansatz. In a recent work [57], it has been proven that barren plateaus are not present in circuits that are equivariant under the symmetric group S_n, namely the group of permutations on n elements, in this case all permutations over the qubits. While our circuit is also permutation equivariant, we define permutations based on the input graphs and not the qubits themselves, so our approach differs from the equivariant quantum neural networks in [57] as a) the incorporation of edge weights into the unitaries prevents the unitaries from commuting with all possible permutations of *qubits*, and b) multiple qubits can potentially correspond to one vertex. While permutation equivariance poses some restrictions on the expressibility of the ansatz and one would expect a better scaling of gradients than in, e.g., hardware-efficient types of circuits, the results of [57] do not directly translate to our work for the above reasons.

To get additional insight, one can also turn to results on barren plateaus related to QAOA-type circuits, due to the structural similarity that our ansatz has to them. The authors of [242] investigate the scaling of the variance of gradients of two related types of ansatzes. They characterize ansatzes given by the following two Hamiltonians: the transverse field Ising model (TFIM),

$$H_{\text{TFIM}} = \sum_{i=1}^{n_f} Z_i Z_{i+1} + h_x \sum_{i=1}^{n} X_i, \tag{6.20}$$

where $n_f = n - 1$ ($n_f = n$) for open (periodic) boundary conditions, and a spin glass (SG),

$$H_{\text{SG}} = \sum_{i<j} h_i Z_i + J_{ij} Z_i Z_j + \sum_{i=1}^{n} X_i, \tag{6.21}$$

with h_i, J_{ij} drawn from a Gaussian distribution. Based on the generators of those two ansatzes, the authors of [242] show that an ansatz that consists of layers given

by the TFIM Hamiltonian has a favorable scaling of gradients. An ansatz that consists of layers given by H_{SG}, on the other hand, does not. Considering the results for the two above Hamiltonians, one can expect that whether our ansatz exhibits barren plateaus will strongly depend on the encoded graphs, i.e., the connectivity, edge weights and node features. Which types of graphs lead to a favorable scaling of gradients, and for what learning tasks our ansatz exhibits good performance at a number of layers polynomial in the input size, is an interesting question that we leave for future work.

Additionally to barren plateaus that are a result of the randomness of the circuit, there is a type of barren plateau that is caused by hardware noise, called noise-induced barren plateaus (NIBPs) [49]. This problem can not be directly mitigated by the choice of circuit architecture, as eventually all circuit architectures are affected by hardware noise, especially when they become deeper. We do not expect that our circuit is resilient to NIBPs, however, the numerical results in Section 6.5 show that the EQC already performs well with only one layer for the environment we study in this work as we scale up the problem size. This provides hope that, at least in terms of circuit depth, the EQC will scale favorably in the number of layers as the number of qubits in the circuit is increased, and therefore the effect of NIBPs will be less severe than for other circuit architectures with the same number of qubits.

Another important question for the training of ML models is that of data efficiency, i.e., how many training data points are required to achieve a low generalization error. Indeed, one of the key motivating factors behind the design of geometric models that preserve symmetries in the training data is to reduce the size of the training data set. In the classical literature, it was shown that geometric models require fewer training data and as a result often fewer parameters as models that do not preserve said symmetries [243]. Recent work showed that this is also true for S_n-equivariant quantum models [57], where the authors give an improved bound on the generalization error compared to the bounds that were previously shown to exist for general classes of PQCs [244]. However, the results from [57] do again not directly translate to our approach as stated in the context of barren plateaus above.

6.4 Quantum neural combinatorial optimization with the EQC

Combinatorial optimization problems are ubiquitous, be it in transportation and logistics, electronics, or scheduling tasks. These types of problems have also been studied in computer science and mathematics for decades. Many interesting combinatorial optimization problems that are relevant in industry today are NP-hard, so that no general efficient solution is expected to exist. For this reason, heuristics have gained much popularity, as they often provide high-quality solutions to real-world instances of many NP-hard problems. However, good heuristics require domain expertise in their design and they have to be defined on a per-problem basis. To circumvent hand-crafting heuristic algorithms, machine learning approaches for solving combinatorial optimization problems have been studied. One line of research in this area investigates using NNs to learn algorithms for solving combinatorial optimization problems [108, 245], which is known as NCO. Here, NNs learn to solve combinatorial optimization problems based on data, and can then be used to find approximate solutions to arbitrary instances of the same problem. First approaches in this direction used supervised learning to find approximate solutions based on NN techniques from natural language processing [246]. A downside of the supervised approach is that it requires access to a large amount of training data in form of solved instances of the given problem, which requires solving many NP-hard instances of the problem to completion. At large problem sizes, this is a serious impediment for the practicability of this method. For this reason, RL was introduced as a technique to train these heuristics. These RL-based approaches have been shown to successfully solve even instances of significant size in problems with a geometric structure like the convex hull problem [239], chip placement [247] or the vehicle routing problem [248]. To implement NCO in this work we use Q-learning as described in Section 3.2 and Section 5.1 following [239].

In this section, we formally define the NCO task that we address in this work, and the specific setup of the EQC and its observables. We show that each component of the QNCO scheme is equivariant under permutation of the vertices, and then analytically study the expressivity of our ansatz at depth one.

6.4.1 Formal definition of learning task and figures of merit

Our goal is to use the ansatz described in Section 6.3 to train a model that, once trained, implements a heuristic to produce tours for previously unseen instances of the TSP. The TSP consists of finding a permutation of a set of cities such that the resulting length of a tour visiting each city in this sequence is minimal. The heuristic takes as input an instance of the TSP problem in form of a weighted 2D Euclidean graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ vertices representing the cities and edge weights $\varepsilon_{ij} = \mathrm{d}(v_i, v_j)$, where $\mathrm{d}(v_i, v_j)$ is the Euclidean distance between nodes v_i and v_j. Specifically, we are dealing with the symmetric TSP, where the edges in the graph are undirected. Given $\mathcal{G}$, the algorithm constructs a tour in $n-2$ steps. Starting from a given (fixed) node in the proposed tour $T_{t=1}$, in each step t of the tour selection process the algorithm proposes the next node (city) in the tour. Once the second-before-last node has been added to the tour, the last one is also directly added, hence the tour selection process requires $n-2$ steps. This can also be viewed as the process of successively marking nodes in a graph as they are added to a tour. In order to refer to versions of the input graph at different time steps where the nodes that are already present in the tour are marked, we now define the *annotated graph*.

Definition 6.1 (Annotated graph)**.** *For a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, we call $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{\alpha}^{(t)})$ the annotated graph at time step t. The vector $\boldsymbol{\alpha}^{(t)} \in \{0, \pi\}^n$ specifies which nodes are already in the tour T_t ($\alpha_i^{(t)} = 0$) and which nodes are still available for selection ($\alpha_i^{(t)} = \pi$).*

In each time step of an episode in the algorithm, the model is given an annotated graph as input. Based on the annotated graph, the model should select the next node to add to the partial tour T_t at step t. The annotation can be used to partition the nodes $\mathcal{V}$ into the set of *available nodes* $\mathcal{V}_a = \{v_i | \alpha_i^{(t)} = \pi\}$ and the set of *unavailable nodes* $\mathcal{V}_u = \{v_i | \alpha_i^{(t)} = 0\}$. The node selection process can now be defined as follows.

Definition 6.2 (Node selection)**.** *Given an annotated graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{\alpha}^{(t)})$, the node selection process consist of selecting nodes in a tour in a step-wise fashion. To add a node to the partial tour T_t, the next node is selected from the set of available nodes $\mathcal{V}_a$. The unavailable nodes $\mathcal{V}_u$ are ignored in this process.*

After $n-2$ steps, the model has produced a tour T_n. A depiction of this process can be found in Figure 6.3. To assess the quality of the generated tour, we compare

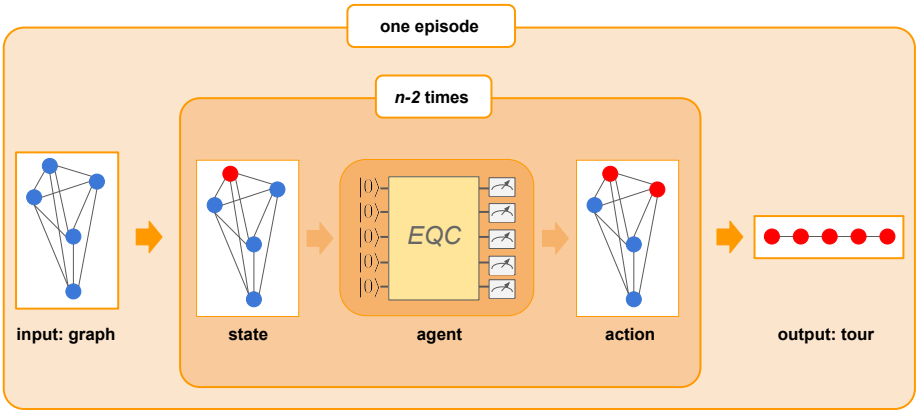


Figure 6.3: An illustration of one episode in the TSP environment. The agent receives a graph instance as input, where the first node is already added to the proposed tour (marked red), which can always be done without loss of generality. In each time step, the agent proposes which node should be added to the tour next. After the second-to-last node has been selected, the agent returns a proposed tour.

the tour length $c(T_n)$ to the length of the optimal tour $c(T^*)$, where

$$c(T) = \sum_{\{i,j\} \in \mathcal{E}_T} \varepsilon_{ij} \tag{6.22}$$

is the sum of edge weights (distances) for all edges between the nodes in the tour, with $\mathcal{E}_T \subset \mathcal{E}$. We measure the quality of the generated tour in form of the approximation ratio

$$\frac{c(T_n)}{c(T^*)}. \tag{6.23}$$

In order to perform Q-learning we need to define a reward function that provides feedback to the RL agent on the quality of its proposed tour. The rewards in this environment are defined by the difference in overall length of the partial tour T_t at time step t, and upon addition of a given node v_l at time step $t+1$:

$$r(T_t, v_l) = -c(T_{(t+1,v_l)}) + c(T_t). \tag{6.24}$$

Note that we use the negative of the cost as a reward, as a Q-learning agent will always select the action that leads to the maximum expected reward.

The learning process is defined in terms of a DQN algorithm, where the Q-function approximator is implemented in form of a PQC (which is described in detail in Section 6.3). Here, we define the TSP in terms of an RL environment, where the set of states $\mathcal{S} = \{\mathcal{G}_i(\mathcal{V}, \mathcal{E}, \boldsymbol{\alpha}^{(t)})$ for $i = 1, \ldots, |\mathcal{X}|$ and $t = 1, \ldots, n-1\}$ consists of all possible annotated graphs (i.e., all possible configurations of values of $\boldsymbol{\alpha}^{(t)}$) for each instance i in the training set $\mathcal{X}$. This means that the number of states in this environment is $|\mathcal{S}| = 2^{n-1}|\mathcal{X}|$. The action that the agent is required to perform is selecting the next node in each step of the node selection process described in Definition 6.2, so the action space $\mathcal{A}$ consists of a set of indices for all but the first node in each instance (as we always start from the first node in terms of the list of nodes we are presented with for each graph, so $\alpha_1^{(t)} = 0, \ \forall \ t$), and $|\mathcal{A}| = n - 1$.

The Q-function approximator gets as input an annotated graph, and returns as output the index of the node that should next be added to the tour. Which index this is, is decided in terms of measuring an observable corresponding to each of the available nodes $\mathcal{V}_a$. Depending on the last node added to the partial tour, denoted as v_{t-1}, the observable for each available node v_l is defined as

$$O_{v_l} = \varepsilon_{v_{t-1}, v_l} Z_{v_{t-1}} Z_{v_l} \tag{6.25}$$

weighted by the edge weight $\varepsilon_{v_{t-1}, v_l}$, and the Q-value corresponding to each action is

$$Q(\mathcal{G}_i(\mathcal{V}, \mathcal{E}, \boldsymbol{\alpha}^{(t)}), v_l) = \left\langle \mathcal{E}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{\beta}, \boldsymbol{\gamma} \right|_p O_{v_l} \left| \mathcal{E}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{\beta}, \boldsymbol{\gamma} \right\rangle_p , \tag{6.26}$$

where the exact form of $\left|\mathcal{E}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{\beta}, \boldsymbol{\gamma}\right\rangle_p$ is described in Section 6.3. The node that is added to the tour next is the one with the highest Q-value,

$$\text{argmax}_{v_l} Q(\mathcal{G}_i(\mathcal{V}, \mathcal{E}, \boldsymbol{\alpha}^{(t)}), v_l). \tag{6.27}$$

All unavailable nodes $v_l \in \mathcal{V}_u$ are not included in the node selection process, so we manually set their Q-values to a large negative number to exclude them, e.g.,

$$Q(\mathcal{G}_i(\mathcal{V}, \mathcal{E}, \boldsymbol{\alpha}^{(t)}), v_l) = -10000 \ \forall \ v_l \in \mathcal{V}_u.$$

We also define a stopping criterion for our algorithm, which corresponds to the agent solving the TSP environment for a given instance size. As we aim at comparing the results of our algorithm to optimal solutions in this work, we have access to a labeled set of instances and define our stopping criterion based on these. However,

note that the optimal solutions are not required for training, as a stopping criterion can also be defined in terms of number of episodes or other figures of merit that are not related to the optimal solution. In this work, the environment is considered as solved and training is stopped when the average approximation ratio of the past 100 iterations is < 1.05, where an approximation ratio of 1 means that the agent returns the optimal solution for the instances it was presented with in the past 100 episodes. We do not set the stopping criterion at optimality for two reasons: i) it is unlikely that the algorithm finds a parameter setting that universally produces the optimal tour for all training instances, and ii) we want to avoid overfitting on the training data set. If the agent does not fulfill the stopping criterion, the algorithm will run until a predefined number of episodes is reached. In our numerical results shown in Section 6.5, however, most agents do not reach the stopping criterion of having an average approximation ratio below 1.05, and run for the predefined number of episodes instead. Our goal is to generate a model that is, once fully trained, capable of solving previously unseen instances of the TSP.

6.4.2 Equivariance of algorithm components

We showed in Section 6.3.1 that our ansatz of arbitrary depth is permutation equivariant. Now we proceed to show that the Q-values that are generated from measurements of this PQC, and the tour generation process as described in Section 6.4.1 are equivariant as well. While the equivariance of all components of an algorithm is not a pre-requisite to harness the advantage gained by an equivariant model, knowing which parts of our learning strategy fulfill this property provides additional insight for studying the performance of our model later. As we show that the whole node selection process is equivariant, we know that the algorithm will always generate the same tour for every possible permutation of the input graph for a fixed setting of parameters, given that the model underlying the tour generation process is equivariant. This is not necessarily true for a non-equivariant model, and simply by virtue of giving a permuted graph as input, the algorithm can potentially return a different tour.

Theorem 6.2 (Equivariance of Q-values)**.** *Let $Q(\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{\alpha}), v_l) = Q(\mathcal{G}, v_l)$ be a Q-value as defined in Equation* (6.26)*, where we drop instance-specific sub- and superscripts for brevity. Let σ be a permutation of $n = |\mathcal{V}|$ elements, where the l-th element corresponds to the l-th vertex v_l and σ_Q be a permutation that reorders the set of Q-values $Q(\mathcal{G}) = \{Q(\mathcal{G}, v_1), \ldots, Q(\mathcal{G}, v_n)\}$ in correspondence*

to the reordering of the vertices by σ. Then the Q-values $Q(\mathcal{G})$ are permutation equivariant,

$$Q(\mathcal{G}) = \sigma_Q Q(\mathcal{G}_\sigma), \tag{6.28}$$

where $\mathcal{G}_\sigma$ is the permuted graph.

Proof. We know from Theorem 6.1 that the ansatz we use, and therefore the expectation values $\langle O_{v_l} \rangle$, are permutation equivariant. The Q-values are defined as $Q(\mathcal{G}, v_l) = \varepsilon_{ij} \langle O_{v_l} \rangle$ (see Equation (6.26)) and therefore additionally depend on the edge weights of the graph $\mathcal{G}$. The edge weights are computed according to the graph's adjacency matrix, and re-ordered under a permutation of the vertices and assigned to their corresponding permuted expectation values. □

As a second step, to show that all components of our algorithm are permutation equivariant, it remains to show that the tours that our model produces as described in Section 6.4.1 are also permutation equivariant.

Corollary 6.1 (Equivariance of tours)**.** *Let $T(\mathcal{G}, \boldsymbol{\beta}, \boldsymbol{\gamma}, v_0)$ be a tour generated by a permutation equivariant agent implemented with a PQC as defined in Equation* (6.3) *and Q-values as defined in Equation* (6.26)*, for a fixed set of parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$ and a given start node v_0, where a tour is a cycle over all vertices $v_l \in \mathcal{V}$ that contains each vertex exactly once. Let σ be a permutation of the vertices $\mathcal{V}$, and σ_T a permutation that reorders the vertices in the tour accordingly. Then the output tour is permutation equivariant,*

$$T(\mathcal{G}, \boldsymbol{\beta}, \boldsymbol{\gamma}, v_0) = \sigma_T T(\mathcal{G}_\sigma, \boldsymbol{\beta}, \boldsymbol{\gamma}, v_{\sigma(0)}). \tag{6.29}$$

Proof. We have shown in Theorem 6.2 that the Q-values of our model are permutation equivariant, meaning that a permutation of vertices results in a reordering of Q-values to different indices. Action selection is done by $v_{t+1} = \text{argmax}_v Q(\mathcal{G}_i^{(t)}, v)$, and the node at the index corresponding to the largest Q-value is chosen. To generate a tour, the agent starts at a given node v_0 and sequentially selects the following $n-1$ vertices. Upon a permutation of the input graph, the tour now starts at another node index $v_{\sigma(0)}$. Each step in the selection process can now be seen w.r.t. the original graph $\mathcal{G}$ and the permuted graph $\mathcal{G}_\sigma$. As we have shown in Theorem 6.1, equivariance of the model holds for arbitrary input graphs, so in particular it holds for each $\mathcal{G}$ and $\mathcal{G}_\sigma$ in the action selection process, and the output tour under the permuted graph is equal to the output tour under the original graph up to a renaming of the vertices. □

6.4.3 Analysis of expressivity

In this section, we analyze under which conditions there exists a setting of β, γ for a given graph instance $\mathcal{G}_i$ for our ansatz at depth one that can produce the optimal tour for this instance. Note that this does not show anything about constructing the optimal tour for a number of instances simultaneously with this set of parameters, or how easy it is to find any of these sets of parameters. Those questions are beyond the scope of this work. The capability to produce optimal tours at any depth for individual instances is of interest because first, we do not expect that the model can find a set of parameters that is close-to-optimal for a large number of instances if it is not expressive enough to contain a parameter setting that is optimal for individual instances. Second, the goal of a ML model is always to find similarities within the training data that can be used to generalize well on the given learning task, so the ability to find optimal solutions on individual instances is beneficial for the goal of generalizing on a larger set of instances. Additionally, how well the model generalizes also depends on the specific instances and the parameter optimization routine, and therefore it is hard to make formal statements about the general case where we find one universal set of parameters that produces the optimal solution for arbitrary sets of instances.

For our model at $p = 1$, we can compute the analytic form of the expectation values of our circuit as defined in Equation (6.25) and Equation (6.26) as the following, by a similar derivation as in [241],

$$\langle O_{v_l} \rangle = \varepsilon_{v_{t-1},v_l} \cdot \sin(\beta\pi)\, \sin(\varepsilon_{v_{t-1},v_l}\gamma) \cdot \prod_{\substack{(v_l,k)\in\mathcal{E} \\ k\neq v_{t-1}}} \cos(\varepsilon_{v_l,k}\gamma), \tag{6.30}$$

where v_{t-1} is the last node in the partial tour and v_l is the candidate node. Note that due to the specific setup of node features used in our work where the contributions of nodes already present in the tour are turned off, these expectation values are simpler than those given for Ising-type Hamiltonians in [241]. For a learning task where contributions of all nodes are present in every step, the expectation values of the EQC will be the same as those for Ising Hamiltonians without local fields given in [241], with the additional node features $\boldsymbol{\alpha}$. Due to this structural similarity to the ansatz used in the QAOA, results on the hardness to give an analytic form of these expectation values at $p > 1$ also transfer to our model. Even at depth $p = 2$ analytic expressions can only be given for certain types

of graphs [249, 250], and everything beyond this quickly becomes too complex. For this reason, we can only make statements for $p = 1$ in this work.

In order to generate an arbitrary tour of our choice, in particular also the optimal tour, it suffices to guarantee that for a suitable choice of (fixed) γ, at each step in the node selection process the edge we want to add next to the partial tour has highest expectation. One way we can do this is by controlling the signs of each sine and cosine term in Equation (6.30) such that only the expectation values corresponding to edges that we want to select are positive, and all others are negative.

To understand whether this is possible, we can leverage known results about the expressivity of the sine function. For any rationally independent set of $\{x_1, ..., x_n\}$ with labels $y_i \in (-1, 1)$, the sine function can approximate these points to arbitrary precision ϵ as shown in [251], i.e., there exists an ω s.t.

$$|\sin(\omega x_i) - y_i| < \epsilon \text{ for } i = 1, \ldots, n. \tag{6.31}$$

In general, the edge weights of graphs that represent TSP instances are not rationally independent.[1] However, in principle they can easily be made rationally independent by adding a finite perturbation ϵ_i' to each edge weight. The results in [251] imply that almost any set of points $x_1, \ldots, x_n$ with $0 < x_i < 1$ is rationally independent, so we can choose ϵ_i' to be drawn uniformly at random from $(0, \epsilon_{\max}]$. As long as these perturbations are applied to the edge weights in a way that does not change the optimal tour, as could be done by ensuring that $\epsilon_{\max}$ is small enough so that the proportions between edge weights are preserved, we can use this perturbed version of the graph to infer the optimal tour. (Such an $\epsilon_{\max}$ can be computed efficiently.) In this way we can guarantee that the ansatz at depth one can produce arbitrary labelings of our edges, which in turn let us produce expectation values such that only the ones that correspond to edges in the tour of our choice will have positive values. We note that in the analysis we assume real-valued (irrational) perturbations, which of course cannot be represented in the computer. However, by using the results of [251] and approximating ± 1 within a small epsilon, we can get a robust statement where finite precision suffices.

[1] The real numbers $x_1, \ldots, x_n$ are said to be rationally independent if no integers $k_1, \ldots, k_n$ exist such that $x_1 k_1 + \cdots + x_n k_n = 0$, besides the trivial solution $k_i = 0 \ \forall \ k$. Rational independence also implies the points are not rational numbers, so they are also not numbers normally represented by a computer.

Theorem 6.3 (Ansatz can generate optimal tours for rationally independent edge weights)**.** *There exists a setting $(\beta, \gamma)^*$ for each graph instance of the symmetric TSP such that the ansatz at depth one described in Section 6.3 will produce the optimal tour T^* with the node selection process described in Definition 6.2, given that the edge weights ε_{ij} of the graph are rationally independent and*

$$\varepsilon_{ij}\gamma \neq \frac{\pi}{4} + n\pi \;\forall\; n \in \mathbb{Z}.$$

Proof. As known from [251], we can find a parameter ω such that we can approximate an arbitrary labeling in $[-1, 1]$ for our rationally independent edge weights with the sine function. Given that this labeling exists, we now show how to use this labeling to generate the optimal tour with the EQC at depth one.

For $p = 1$, we can compute the analytic form of the expectation values of our circuit as defined in Equation (6.25) and Equation (6.26) as the following, by a similar derivation as in [241],

$$\langle O_{v_l} \rangle = \varepsilon_{v_{t-1},v_l} \cdot \sin(\beta\pi)\, \sin(\varepsilon_{v_{t-1},v_l}\gamma) \cdot \prod_{\substack{(v_l,k)\in\mathcal{E} \\ k\neq v_{t-1}}} \cos(\varepsilon_{v_l,k}\gamma), \tag{6.32}$$

where v_{t-1} is the last node in the partial tour and v_l is the candidate node. By the identity $\cos(\theta) = \sin(\frac{\pi}{2} - \theta)$ we can rewrite Equation (6.30) as

$$\langle O_{v_l} \rangle = \varepsilon_{v_{t-1},v_l} \cdot \sin(\beta\pi)\, \sin(\varepsilon_{v_{t-1},v_l}\gamma) \cdot \prod_{\substack{(v_l,k)\in\mathcal{E} \\ k\neq v_{t-1}}} \sin\left(\frac{\pi}{2} - \varepsilon_{v_l,k}\gamma\right). \tag{6.33}$$

Let us now assume that we want to construct a fixed (but arbitrary) tour T. First, we notice that the term $\sin(\beta\pi)$ does not depend on v_{t-1} or v_l and is the same for all v_l. This means that this term can merely flip the sign of all $\langle O_{v_l} \rangle$, and from now on w.l.o.g. we assume that β is such that the term is positive. Now we can again formulate the tour generation task in terms of a binary classification problem, where we want to find a configuration of labels for our remaining sin terms in Equation (6.33) s.t. the product will have the highest expectation value in each node selection step for the edge that produces the ordering we have chosen for T. Again, we can accomplish this for arbitrary settings of edge weights by only considering the sign of the resulting product. This means that we have to find an assignment of the edges ε_{ij} to the classes $f_\pm$ that at each step of the node selection process will lead to the node being picked that we specify in T. As all edges can occur in the above products multiple times during the node selection process, this is a non-trivial task. However, if we can guarantee that each $\langle O_{v_l} \rangle_t$ at

node selection step t contains at least one *unique term* that is only present in this specific expectation value, we can use this term to control the sign of this specific value. Each ε_{ij} occurs either in the leading term $\sin(\varepsilon_{ij}\gamma)$ (corresponding to the candidate edge to be potentially added in the next step) or in the product term as $\sin(\frac{\pi}{2} - \varepsilon_{ij}\gamma)$ (corresponding to an outgoing edge from the current candidate). We can easily see that the leading term *only* appears in the case when we ask for this specific ε_{ij} to be the next edge in the tour, and from Definition 6.2 we know that this only happens once in the node selection process. In all other expectations, ε_{ij} appears only with the "offset" of $\frac{\pi}{2}$. This means that this leading term is the unique term that we are looking for, as long as $\sin(\varepsilon_{ij}\gamma) \neq \sin(\frac{\pi}{2} - \varepsilon_{ij}\gamma)$, so as long as $\sin(\varepsilon_{ij}\gamma) \neq \cos(\varepsilon_{ij}\gamma)$. We know that $\cos(\theta) = \sin(\theta)$ for $\theta = \frac{\pi}{4} + n\pi$ with $n \in \mathbb{Z}$. So as long as

$$\varepsilon_{ij}\gamma \neq \frac{\pi}{4} + n\pi \;\forall\; n \in \mathbb{Z}, \varepsilon_{ij} \in \mathcal{E}, \tag{6.34}$$

and all ε_{ij} are unique, our ansatz can construct the desired tour T. In this case, we have a guarantee that we can construct the tour T for any configuration of edges that fulfills Equation (6.34). In particular, this means that we can construct the optimal tour in this way. □

However, we point out that the parameter γ that leads to the construction of the optimal tour can in principle be arbitrarily large and hard to find. We do not go deeper into this discussion since in fact we do not want to rely on this proof of optimality as a guiding explanation of how the algorithm works.

The reason for this is that in some way, this proof of optimality works *despite* the presence of the TSP graph and not because of it. This is similar in vein to universality results for QAOA-type circuits, where it can be shown that for very specific types of Hamiltonians, alternating applications of the cost and mixer Hamiltonian leads to quantum computationally universal dynamics, i.e., it can reach all unitaries to arbitrary precision [195, 252], but these Hamiltonians are not related to any of the combinatorial optimization problems that were studied in the context of the QAOA. While these results provide valuable insight into the expressivity of the models, in our case they do not inform us about the possibility of a quantum advantage on the learning problem that we study in this work. In particular, we do not know from these results whether the EQC utilizes the information provided by the graph features in a way in which the algorithm benefits from the quantumness of the model, at depth one or otherwise. As it is

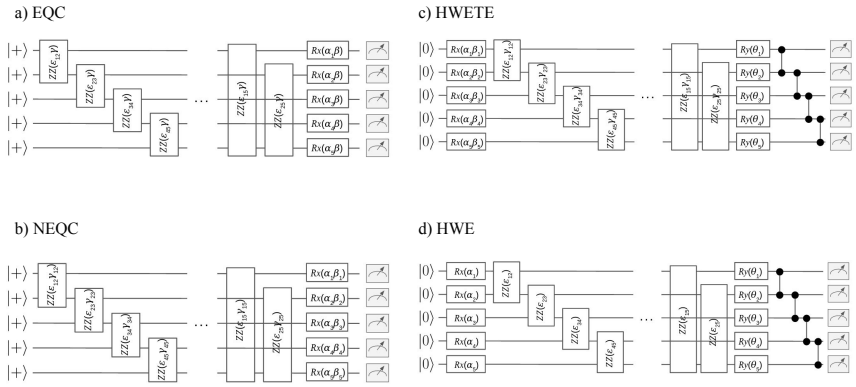


Figure 6.4: One layer of each of the circuits studied in this work. a) The EQC with two trainable parameters β, γ per layer. b) The same set of gates as in the EQC, but we break equivariance by introducing one individual free parameter per gate (denoted NEQC). c) Similar to the NEQC, but we start from the all-zero state and add a final layer of trainable one-qubit gates and a ladder of CZ-gates (denoted hardware-efficient with trainable encoding, HWETE). d) Same as the HWETE, but only the single-qubit Y-rotation parameters are trained (denoted HWE).

known that the QAOA applied to ground state finding benefits from interference effects, investigating whether similar results hold for our algorithm is an interesting question that we leave for future work.

Additionally, we note that high expressivity alone does not necessarily lead to a good model, and may even lead to issues in training as the well-studied phenomenon of barren plateaus [229], or a susceptibility to overfitting on the training data. In practice, the best models are those that strike a balance between being expressive enough, and also restricting the search space of the model in a way that suits the given training data. Studying and designing models that have this balance is exactly the goal of geometric learning, and the permutation equivariance we have proven for our model is a helpful geometric prior for learning tasks on graphs.

6.5 Numerical results

After proving that our model is equivariant under node permutations and analytically studying the expressivity of our ansatz, we now numerically study the training and validation performance of this model on TSP instances of varying size

in a NCO context. The training data set that we use is taken from [246], where the authors propose a novel classical attention approach and evaluate it on a number of geometric learning tasks.[1] To compute optimal solutions for the TSP instances with 10 and 20 cities we used the library [253].

We evaluate the performance of the EQC on TSP instances with 5, 10 and 20 cities (corresponding to 5, 10, and 20 qubits, respectively). As described in Section 6.4.1, the environment is considered as solved by an agent when the running average of the approximation ratio over the past 100 episodes is less than 1.05. Otherwise, each agent will run until it reaches the maximum number of episodes, that we set to be 5000 for all agents. Note that this is merely a convenience to shorten the overall training times, as we have access to the optimal solutions of our training instances. In a realistic scenario where one does not have access to optimal solutions, the algorithm would simply run for a fixed number of episodes or until another convergence criterion is met. When evaluating the final average approximation ratios, we always use the parameter setting that was stored in the final episode, regardless of the final training error. When variations in training lead to a slightly worse performance than what was achieved before, we still use the final parameter setting. We do this because as noted above, in a realistic scenario one does not have knowledge about the ratio to the optimal solutions during training. Unless otherwise stated, all models are trained on 100 training instances and evaluated on 100 validation instances.

As we are interested in the performance benefits that we gain by using an ansatz that respects an important graph symmetry, we compare our model to versions of the same ansatz where we gradually break the equivariance property. We start with the simplest case, were the circuit structure is still the same as for the EQC, but instead of having one β_l, γ_l in each layer, every X- and ZZ-gate is individually parametrized. As these parameters are now tied directly to certain one- and two-qubits gates, e.g. an edge between qubits one and two, they will not change location upon a graph permutation and therefore break equivariance. We call this the non-equivariant quantum circuit (NEQC). To go one step further, we take the NEQC and add a variational part to each layer that is completely unrelated to the graph structure: namely a hardware-efficient layer that consists

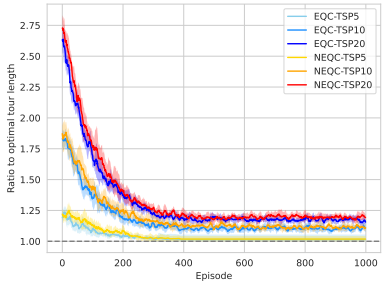
[1] We note that we have re-computed the optimal tours for all instances that we use, as the data set uploaded by the authors of [246] erroneously contains sub-optimal solutions. This was confirmed with the authors, but at the time of writing of this work their repository has not been updated with the correct solutions.

of parametrized Y-rotations and a ladder of CZ-gates. In this ansatz, we have a division between a *data encoding* part and a *variational* part, as is often done in QML. To be closer to standard types of ansatzes often used in QML, we also omit the initial layer of H-gates here and start from the all-zero state (which requires us to switch the order of X- and ZZ-gates)[1]. We denote this the hardware-efficient with trainable embedding (HWETE) ansatz. Finally, we study a third ansatz, where we take the HWETE and now only train the Y-rotation gates, and the graph-embedding part of the circuit only serves as a data encoding step. We call this simply the hardware-efficient (HWE) ansatz. A depiction of all ansatzes can be seen in Figure 6.4.

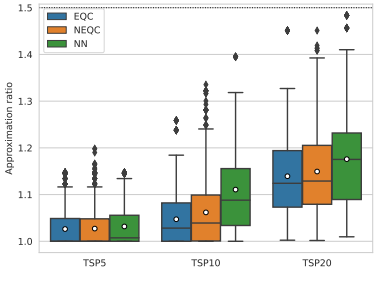
We start by comparing the EQC to the NEQC on TSP instances with 5, 10 and 20 cities. We show the training and validation results in Figure 6.5. To evaluate the performance of the models that we study, we compute the ratio to the optimal tour length as shown in Equation (6.23), as the instances that we can simulate the circuits for are small enough to allow computing optimal tours for.[2] To provide an additional classical baseline, we also show results for the nearest-neighbor heuristic. This heuristic starts at a random node and selects the closest neighboring node in each step to generate the final tour. The nearest-neighbor algorithm finds a solution quickly also for instances with increasing size, but there is no guarantee that this tour is close to the optimal one. However, as we know the optimal tours for all instances, the nearest-neighbor heuristic provides an easy to understand classical baseline that we can use. Additionally, we add the upper bound given by one of the most widely used approximation algorithms for the TSP (as implemented e.g. in Google OR-Tools): the Christofides algorithm. This algorithm is guaranteed to find a tour that is at most 1.5 times as long as the optimal tour [254]. In the case where any of our models produces validation results that are on average above this upper bound of the Christofides algorithm, we consider it failed, as it is more efficient to use a polynomial approximation algorithm for these instances. However, we stress that this upper bound can only serve to inform us about the failure of our algorithms and not their success, as in practice the Christofides algorithm often achieves much better results than those given by the upper bound. We also note that both the Christofides and nearest-neighbor algorithms are provided

[1] However, in practice it did not make a difference whether we started from the all-zero or uniform superposition state in the learning task that we study.

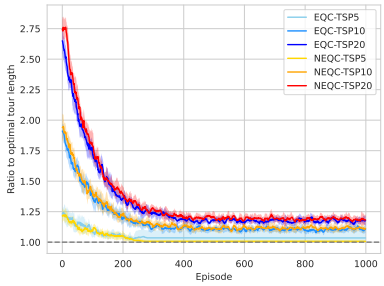
[2] For reference, the authors of [246], who generated the training instances that we use, stop comparing to optimal solutions at $n = 20$ as it becomes extremely costly to find optimal tours from thereon out.



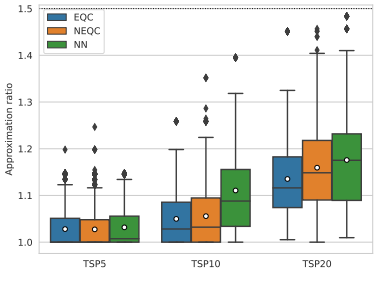
(a) Training performance, 1 layer



(b) Validation performance, 1 layer



(c) Training performance, 4 layers



(d) Validation performance, 4 layers

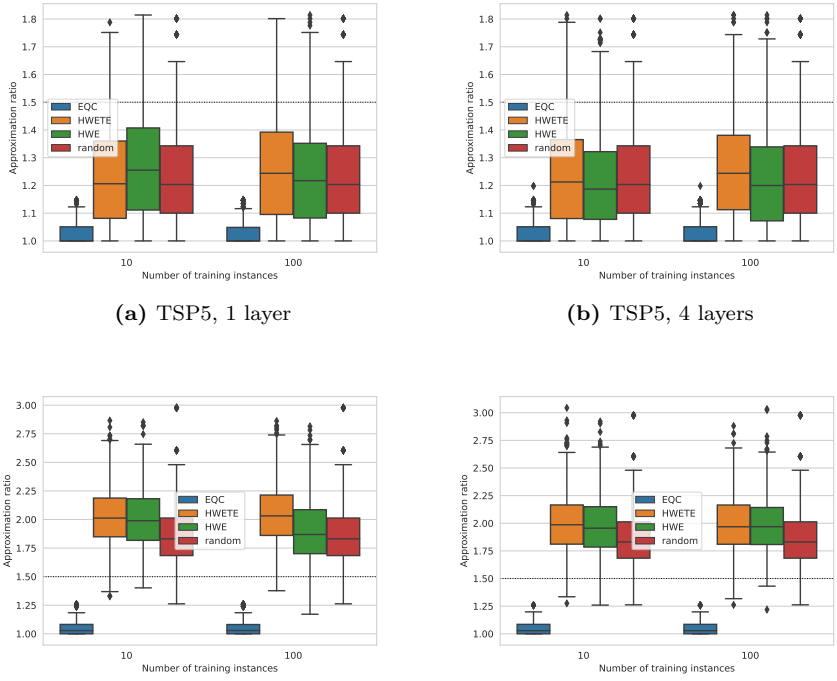
Figure 6.5: Comparison between the EQC and its non-equivariant version (NEQC) in terms of approximation ratio (lower is better) of ten trained models on a set of 100 previously unseen TSP instances for each instance size. The boxes show the upper quartile, median and lower quartile for each model configuration, the whiskers of the boxes extend to 1.5 times the interquartile range, and the black diamonds denote outliers. We additionally show the means of each box as white circles. In the NEQC each gate is parametrized separately but the ansatz structure is otherwise identical to the EQC, as described in Section 6.5. Results are shown on TSP instances with 5, 10 and 20 cities (TSP5, TSP10 and TSP20, respectively). To provide a classical baseline, we also show results for the nearest-neighbor heuristic (NN). a) and b) show the training and validation performance for both ansatzes with one layer, while c) and d) show the same for four layers. The dashed, grey line on the left-hand side figures denotes optimal performance. The dotted, black line on the right-hand side figures denotes the upper bound of the Christofides algorithm, a popular classical approximation algorithm that is guaranteed to find a solution that is at most 1.5 times as long as the optimal tour. Figures a) and c) show the running average over the last ten episodes.

here to assure that our algorithm produces reasonable results, and not to show that our algorithm outperforms classical methods as this is not the topic of the present manuscript. The bound is shown as a dotted black line in Figure 6.5 and Figure 6.6.

Geometric learning models are expected to be more data-efficient than their unstructured counterparts, as they respect certain symmetries in the training data. This means that when a number of symmetric instances are present in the training or validation data, the effective size of these data sets is decreased. This usually translates into models that are more resource-efficient in training, e.g. by requiring fewer parameters or fewer training samples. In our comparison of the EQC and the NEQC, we fix the number of training samples and compare the different models in terms of circuit depth and number of parameters to achieve a certain validation error and expect that the EQC will need fewer layers to achieve the same validation performance as the NEQC. This comparison can be seen in Figure 6.5. In Figure 6.5 a) and b), we show the training and validation performance of both ansatzes at depth one. For instances with five cities, both ansatzes perform almost identically on the validation set, where the NEQC performs worse on a few validation instances. As the instance size increases, the gap between EQC and NEQC becomes bigger. We see that even though the two ansatzes are structurally identical, the specific type of parametrizations we choose and the properties of both ansatzes that result from this make a noticeable difference in performance. While the EQC at depth one has only two parameters per layer regardless of instance size, the NEQC's number of parameters per layer depends on the number of nodes and edges in the graph. Despite having much fewer parameters, the EQC still outperforms the NEQC on instances of all sizes. Increasing the depth of the circuits also does not change this. In Figure 6.5 c) and d) we see that at a depth of four, the EQC still beats the NEQC. The latter's validation performance even slightly decreases with more layers, which is likely due to the increased complexity of the optimization task, as the number of trainable parameters per layer is $\frac{(n-1)n}{2} + n$, which for the 20-city instances means 840 trainable parameters at depth four (compared to 8 parameters in case of the EQC). This shows that at a fraction of the number of trainable parameters, the EQC is competitive with its non-equivariant counterpart even though the underlying structure of both circuits is identical. Compared to the classical nearest-neighbor heuristic, both ansatzes perform well and beat it at all instance sizes, and both ansatzes are also below the approximation ratio upper bound given by the Christofides algorithm on all validation instances. The

box plots in Figure 6.5 show a comparison of the EQC and NEQC in terms of the quartiles of the approximation ratios on the validation set. As it is hard to infer statistical significance of results directly from the box plots, especially when the distributions of data points are not very far apart, we additionally plot the means of the distributions and their standard error, and compute p-values based on a t-test to give more insight on the comparison of these two models in Chapter 8. To show statistical significance of the comparison of the EQC and NEQC, we perform a two-sample t-test with the null-hypothesis that the averages of the two distributions are the same, as is common in statistical analysis, and compute p-values based on this. The p-values confirm that there is indeed a statistical significance in the comparison between models for the 10- and 20-city instances, and that we can be more certain about the significance as we scale up the instance size. The average approximation ratios in case of the 5-city instances are roughly the same, as we can expect due to the fact that there exist only 12 permutations of the TSP graphs of this size. However, even for these small instances the EQC achieves the same result with fewer parameters, namely 2 per layer instead of the 15 per layer required in the NEQC.

Next, we compare the EQC to ansatzes in which we introduce additional variational components that are completely unrelated to the training data structure, as described above. We show results for the HWETE and the HWE ansatz in Figure 6.6. To our own surprise, we did not manage to get satisfactory results with either of those two ansatzes, especially at larger instances, despite an intensive hyperparameter search. Even the HWETE, which is basically identical to the NEQC with additional trainable parameters in each layer, failed to show any significant performance. To gauge how badly those two ansatzes perform, we also show results for an algorithm that selects a random tour for each validation instance in Figure 6.6. In this figure, we show results for TSP instances with five and ten cities for both ansatzes with one and four layers, respectively. Additionally, we show how the validation performance changes when the models are trained with either a training data set consisting of 10 or 100 instances, in the hopes of seeing improved performance as the size of the training set increases. We see that in neither configuration, the HWETE or HWE outperform the Christofides upper bound on all validation instances. Additionally, in almost all cases those two ansatzes do not even outperform the random algorithm. This example shows that in a complex learning scenario, where the number of permutations of each input instance grows combinatorially with instance size and the number of states



(a) TSP5, 1 layer

(b) TSP5, 4 layers

(c) TSP10, 1 layer

(d) TSP10, 4 layers

Figure 6.6: Comparison between EQC and two hardware-efficient ansatzes where we gradually break the equivariance of the original ansatz. We show results for TSP instances with five and ten cities (TSP5, TSP10 respectively) for models trained on 10 and 100 instances, and with one and four layers. Each box is computed over results for ten agents. The hardware-efficient ansatz with trainable embedding (HWETE) consists of trainable graph encoding layers as those in the EQC, with an additional variational part in each layer that consists of parametrized single-qubit Y-gates and a ladder of CZ-gates. The HWE ansatz is the same as the HWETE, but where the graph-embedding part is not trainable and only the Y-gates in each layer are trained. We also show approximation ratios of a random algorithm, where a random tour is picked as the solution to each instance. The dotted, black lines denote the upper bound of the Christofides algorithm. We see that the HWE ansatzes perform extremely badly and barely outperform picking random tours only in some cases.

in the RL environment grows exponentially with the number of instances, a simple hardware-efficient ansatz will fail even when the data encoding part of the PQC is motivated by the problem data structure. While increasing the size of the training set and/or the number of layers in the circuit seems to provide small advantages in some cases, it also leads to a decrease in performance in others. On the other hand, the EQC is mostly agnostic to changes in the number of layers or the training data size. Overall, we see that the closer the ansatz is to an equivariant configuration, the better it performs, and picking ansatzes that respect symmetries inherent to the problem at hand is the key to success in this graph-based learning task.

In Section 6.3 we have also pointed out that the EQC is structurally related to the ansatz used in the QAOA. The main difference in solving instances of the TSP with the NCO approach used in our work and solving it with the QAOA lies in the way in which the problem is encoded in the ansatz, and in the quantity that is used to compute the objective function value for parameter optimization. We give a detailed description of how the TSP is formulated in terms of a problem Hamiltonian suitable for the QAOA and how parameters are optimized in Section 6.2.2. As the QAOA is arguably the most explored variational quantum optimization algorithm at the time of writing, and due to the structural similarity between the EQC and the QAOA's ansatz, we also compare these two approaches on TSP instances with five cities.

For $p = 2$ and 3, we optimized the circuit parameters using Constrained Optimization BY Linear Approximation (COBYLA). In addition, similar to [255], we employed a p-dependent initialization technique for the circuit parameters. Specifically, $(p+1)$-depth QAOA circuit parameters were initialized with the optimal parameters from the p-depth circuit, as follows:

$$\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_{p'}, 0),$$
$$\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{p'}, 0).$$

This way we are allowing the parameter training procedure to start in a known acceptable state based on the results of the previous step. In Figure 6.7 we show our results for five-city instances of the TSP. The approximation ratio shown is derived by dividing the tour length of the best feasible solution, measured as the output of the trained QAOA circuit, by the optimal tour length of the respective instance. In addition, we compute results for two different $p = 3$ QAOA circuits: the first is trained in the procedure described above (where the parameters are

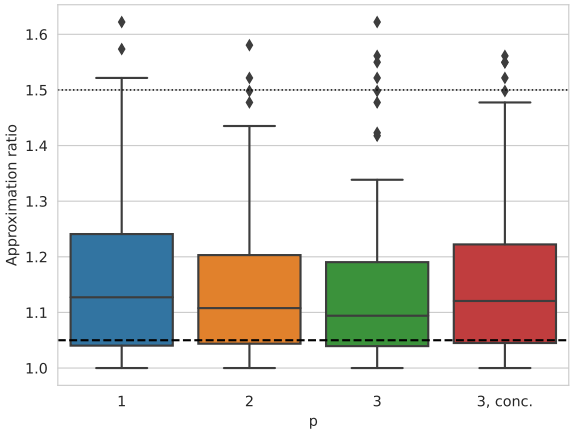


Figure 6.7: Approximation ratio of QAOA up to depth three. Dashed black line denotes average final performance of the EQC at depth one during the last 100 iterations of training on the same instances. Last box shows the results for the best parameters found for one instance at $p = 3$ applied to all training instances, following a parameter concentration argument. The dotted, black line denotes the upper bound of the Christofides algorithm.

trained for each instance). The second uses the parameters of the best QAOA circuit out of those for all instances evaluated at $p = 3$, following a concentration of parameters argument as presented in [61]. The second method is closer to what is done in a ML context, where one set of parameters is used to evaluate the performance on all validation samples.

Due to the number of qubits required to formulate a QUBO for the TSP, we were not able to run QAOA for all TSP instances. For example, an instance with six cities already requires 25 qubits (we can fix the choice of the first city to be visited without loss of generality, requiring only $(n-1)^2$ variables to formulate the QUBO). A different formulation of the QUBO problem presented in [256], that needs $\mathcal{O}(n \log(n))$ qubits, avoids this issue by modifying the circuit design. However, this proposal increases the circuit depth considerably and is therefore ill-suited for the NISQ era.

In Figure 6.7, we can see that finding a good set of parameters for QAOA to solve TSP is hard even for five-city instances. We note that the performance of QAOA improves with higher p, however, QAOA performance is still far from

matching the approximation ratios obtained by EQC even for $p = 3$, which can be seen in Figure 6.7 as a black dashed line. Furthermore, we note that significant computational effort is required to obtain these results: methods like COBYLA are based on gradient descent, which requires us to evaluate the circuit many times until either convergence or the maximum number of iterations is reached. We also note that due to the heuristic optimization of the QAOA parameters themselves, we are not guaranteed that the configuration of parameters is optimal, which may result in either insufficient iterations to converge or premature convergence to sub-optimal parameter values. In an attempt to mitigate this, we tested several optimizers (Adam, SPSA, BFGS and COBYLA) and used the best results, which were those found by COBYLA.

We see in Figure 6.7 that already on these small instances, the QAOA requires significantly deep circuits to achieve good results, that may be out of reach in a noisy near-term setting. The EQC on the other hand i) uses a number of qubits that scales linearly with the number of nodes of the input graph as opposed to the n^2 number of variables required for QAOA, and ii) already shows good performance at depth one for instances with up to 20 cities. In addition to optimizing QAOA parameters for each instance individually, we also show results of applying one set of parameters that performed well on one instance at depth three, on other instances of the same problem following the parameter concentration argument given in [61] and described in more detail in Section 6.2.2. While we find that parameters seem to transfer well to other instances of the same problem in case of the TSP, the overall performance of the QAOA is still much worse than that of the EQC.

6.6 Discussion

After providing analytic insight on the expressivity of our ansatz, we have numerically investigated the performance of our EQC model on TSP instances with 5, 10, and 20 cities (corresponding to 5, 10, 20 qubits respectively), and compared them to other types of ansatzes that do not respect any graph symmetries. To get a fair comparison, we designed PQCs that gradually break the equivariance property of the EQC and assessed their performance. We find that ansatzes that contain structures that are completely unrelated to the input data structure are extremely hard to train for this learning task where the size of the state space scales exponentially in the number of input nodes of the graph. Despite much

effort and hyperparameter optimization, we did not manage to get satisfactory results with these ansatzes. The EQC on the other hand works almost out-of-the box, and achieves good generalization performance with minimal hyperparameter tuning and relatively few trainable parameters. We have also compared using the EQC in a neural combinatorial optimization scheme with the QAOA, and find that even on TSP instances with only five cities the NCO approach significantly outperforms the QAOA. In addition to training the QAOA parameters for every instance individually, we have also investigated the performance in light of known parameter concentration results that state that in some cases, parameters found on one instance perform well on average for other instances of the same problem. While this is true in the case of the TSP instances we investigate here as well, the overall performance is still worse than that of the EQC.

Comparing our algorithm to the QAOA is also interesting from a different perspective. In Section 6.3 we have seen that our ansatz can be regarded as a special case of a QAOA-type ansatz, where instead of encoding a problem Hamiltonian we encode a graph instance directly, and in case of the specific formulation of the TSP used in this work, include mixing terms only for a problem-dependent subset of qubits. This lets us derive an exact formulation of the expectation values of our model at depth one from those of the QAOA given in [241]. For the QAOA, it is known that in the limit of infinite depth, it can find the ground state of the problem Hamiltonian and therefore the optimal solution to a given combinatorial optimization problem [59]. Additionally, it has been shown that even at low depth, the probability distributions generated by QAOA-type circuits are hard to sample from classically [180]. These results give a clear motivation of why using a quantum model in these settings can provide a potential advantage. While our model is structurally almost identical to that of the QAOA, in our case the potential for advantage is less clear. We saw in Equation (6.30) that at depth one, in each step the expectation value of each edge that we consider to be selected consists of i) a term corresponding to the edge between the last added node and the candidate node, and ii) all outgoing edges from the candidate node. So our model considers the one-step neighborhood of each candidate node at depth one. In the case of the TSP it is not clear whether this can provide a quantum advantage for the learning task as specified in Section 6.4.1. In terms of QAOA, it was shown that in order to find optimal solutions, the algorithm has to "see the whole graph" [257], meaning that all edges in the graph contribute to the expectation values used to minimize the energy. To alleviate this strong requirement on depth, a recursive version of

the QAOA (RQAOA) was introduced in [258]. It works by iteratively eliminating variables in the problem graph based on their correlation, and thereby gradually reducing the problem to a smaller instance that can be solved efficiently by a classical algorithm, e.g. by brute-force search. The authors of [258] show that the depth-one RQAOA outperforms QAOA with constant depth p, and that RQAOA achieves an approximation ratio of one for a family of Ising Hamiltonians.

The node selection process performed by our algorithm with the EQC used as the ansatz is similar to the variable elimination process in the RQAOA, where instead of merging edges, the mixer terms for nodes that have already been selected are turned off, therefore effectively turning expectation values of edges corresponding to unavailable nodes to zero. Furthermore, the specific setup of weighted Z_iZ_j-correlations (see Equation (6.25)) that we measure to compute Q-values in our RL scheme to solve TSP instances are of the same form as those in the Hamiltonian for the weighted MaxCut problem,

$$H_{\text{MaxCut}} = -\frac{1}{2}\sum_{ij} w_{ij}(1 - Z_iZ_j).$$

The MaxCut problem and its weighted variant have been studied in depth in the context of the QAOA, and it has been shown that it performs well on certain instances of graphs for this task [59, 259, 260, 61]. While the TSP and weighted MaxCut are clearly very different problems, the similarity between our algorithm and the RQAOA raises the interesting question whether the mechanisms underlying the successful performance of both models in those two learning tasks are related. Based on this, one may ask the broader question of whether QAOA-type ansatzes implement a favorable bias for hybrid quantum-classical optimization algorithms on weighted graphs, like the RQAOA or the quantum NCO scheme in this work. Specifically, by relating the mechanism underlying the variable elimination procedure in RQAOA, which eliminates variables based on their largest (anti-)correlation in terms of Z_iZ_j operators, to the node selection process in our algorithm that solves TSP instances, we can establish a connection between the EQC and known results for the (R)QAOA on weighted MaxCut. It is an interesting question whether results that establish a quantum advantage of the QAOA can be related to the EQC in a NCO context as we present here, and we leave this question for future work.

CHAPTER 7

Robustness of quantum reinforcement learning under hardware errors

One of the reasons that VQAs have gained increased interest in the past years is that their hybrid nature, where a large part of the computation is offloaded to a classical device, is hypothesized to make them robust to quantum hardware noise to some extent [261, 14]. This hypothesis is also inspired by classical neural networks, which are robust under certain types of noise. In the classical setting, one can broadly distinguish between two types of noise: benign noise that does not severely impact the training procedure or can even improve generalization [262, 263, 264, 265], and adversarial noise which is deliberately constructed to study where neural networks fail [266, 267, 268, 269]. Furthermore, we can distinguish between noise that is present during training, and noise that is present when using the trained model. Adversarial noise is usually of the latter case, where a trained neural network can produce completely wrong outputs due to small perturbations of the input data [270]. The benign type of noise mentioned above on the other hand is usually present at training time in form of perturbations of the input data, activation functions, weights or structure of the neural network, and has even been established as a method to combat overfitting in the classical literature [262, 263, 264, 265, 271].

These results inspired the hypothesis that variational quantum algorithms possess a similar robustness to certain types of noise and may even benefit from its presence when trained on a quantum device. However, thorough investigations that confirm such robustness of VQAs against hardware-related noise, or even a beneficial effect from it, are still lacking. In terms of negative results for the trainability of VQAs under noise, it has been shown that optimization landscapes of noisy

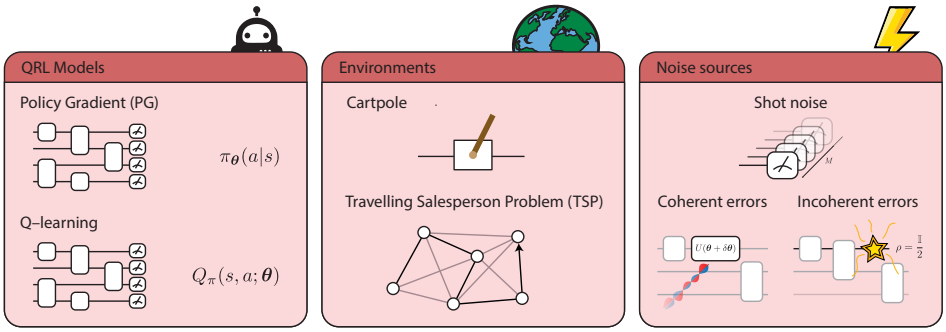


Figure 7.1: Summary of the scenarios analysed in the present work. We consider two models for quantum reinforcement learning (QRL) agents and test their performances on two environments, CartPole and the Travelling Salesperson Problem (TSP). We analyse the performances of the agents when these are trained and used in the presence of most common noise sources found on real quantum hardware, namely statistical fluctuations due to shot noise, coherent errors due to imperfect control or calibration of the device, and finally incoherent errors coming from the unavoidable interaction of the quantum hardware with its environment.

quantum circuits become increasingly flat at a rate that scales exponentially with the number of qubits under local Pauli noise when the circuit depth grows linearly with the number of qubits [49]. In the case of the variational quantum eigensolver, where the goal is to find the ground state of a given Hamiltonian, the presence of noise has been shown to lead to increasing deviation from the ideal energy [272]. Similar effects have been studied in the context of the quantum approximate optimization algorithm (QAOA) [59], where the goal is to find the ground state of a Hamiltonian that represents the solution to a combinatorial optimization problem [273, 68].

When it comes to QML, in-depth studies on the effect of noise on the trainability and performance of VQAs are scarce. Apart from the work mentioned above on noise-induced barren plateaus [49], the authors of [166] provided first insights into how the data encoding method used in a quantum classifier influences its resilience to varying types of noise. As for the potential benefit of noise, the authors of [274] show that the stochasticity induced by measurements in a QML model can help the optimizer to escape saddle points. The above results show that, on the one hand, too much noise will make the model untrainable, while on the other hand, modest amounts of noise can even improve trainability [274]. However, it remains

unclear how large the gap is between tolerable and harmful amounts of noise [261], and it is not expected that this can be answered in a general way for all different types of learning algorithms and noise sources.

In this chapter, we shed light on this question from the angle of variational quantum reinforcement learning. Classical reinforcement learning models have been shown to be sensitive to noise, either during training [275] or in the form of adversarial samples [276, 277]. Additionally, it is known that a bottleneck of RL algorithms is their sample inefficiency, i.e., many interactions with an environment are needed for training [278]. Still, RL resembles human-type learning most closely among the main branches of modern ML, and therefore motivates further studies in this area. Among these studies, RL with VQAs has been proposed and extensively investigated in the noise-free setting over the past few years [153, 154, 150, 75, 157, 199, 76, 279, 156]. These results provide promising perspectives, as quantum models have empirically been shown to perform similarly to neural networks on small classical benchmark tasks [75, 76], while at the same time an exponential separation between classical and quantum learners can be proven for specific contrived environments based on classically hard tasks [150, 75]. These results motivate further studies on how large the above-mentioned gap between tolerable and too much noise is in the case of variational RL algorithms, and how close the algorithm performance can get to the noise-free setting for various types of noise that can be present on near-term devices.

We investigate this for two types of variational RL algorithms, Q-learning and the policy gradient method, by performing extensive numerical experiments for both types of algorithms with two different environments, CartPole and the Travelling Salesperson Problem, and under the effect of a wide class of noise sources, namely shot noise, coherent and incoherent errors. In Figure 7.1 we summarise the approach of the present work showing the QRL models, environments and noise sources considered in the analysis. We start by considering the trade-off between the number of measurement shots taken for each circuit evaluation and the performance of variational agents. As the number of shots required by a QML algorithm can be a bottleneck on near-term devices and RL is known to require many interactions with the environment to learn, we propose a method for Q-learning to reduce the number of overall measurements by taking advantage of the structure of the underlying RL algorithm. Second, we model coherent errors with a random Gaussian perturbation of the variational parameters, and analytically study the

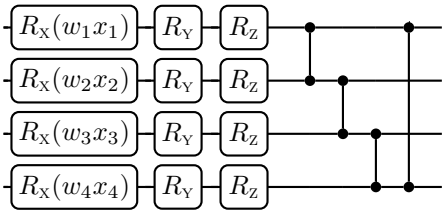
effect of these perturbations on the output of parameterised quantum circuits, similarly to [280]. We provide an upper bound on the perturbation induced by such Gaussian coherent noise based on the Hessian matrix of the circuit, and theoretically and numerically show that hardware-efficient ansätze may be particularly resilient against this type of error due to small second derivatives [51]. Finally, we analyse the performance of the above algorithms under the action of incoherent errors coming from the unavoidable interaction of the qubits with the environment which we have no control over. To study this type of noise, we start by investigating the effect of single-qubit depolarization channels. In addition, we consider a custom noise model that combines various types of errors present on hardware, and study the effect of this noise model with error probabilities that are present in currently available superconducting quantum hardware. Our results show that both policy gradient methods and Q-learning exhibit a robustness to noise that may enable successfully running them on near-term devices. This motivates further study in the quest to find a real-world problem of interest where a quantum advantage for variational RL could be possible.

7.1 Environments and implementation

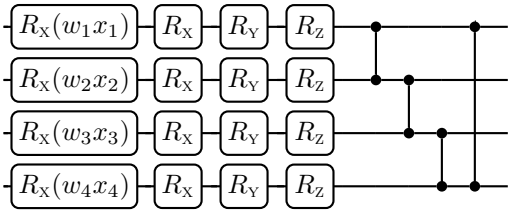
Our goal is to get insight into the effect of noisy training on quantum RL algorithms. For this, we consider quantum versions of the two main paradigms in RL that have been introduced in previous sections: value-based methods (see Section 3.2.2) and policy gradient methods (see Section 3.2.3). As we are interested in the effect of noisy training on models that have otherwise been proven to work well in the noise-free setting, we study models and environments that have been already investigated in this setting before [150, 75, 174]. In this way, we have evidence that the models and hyperparameters that we choose are suitable for the studied environments, and can focus our efforts on understanding the effect that noise has on the training and performance of these agents. The code that was used to generate the numerical results in this work can be found on Github [281].

7.1.1 CartPole

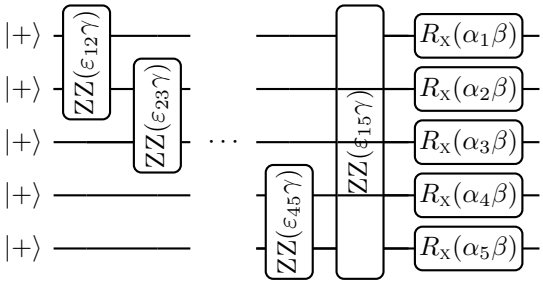
The first environment that we study is the CartPole environment from the OpenAI Gym [282] that was also studied in Chapter 5. For a detailed description of the environment and the implementation of the quantum Q-learning agent, we refer the reader to Section 5.1. For the policy gradient method, we follow the



(a) HWE-Q



(b) HWE-PG



(c) EQC

Figure 7.2: Parameterised circuits used in this work. (a) Hardware-efficient ansatz for Q-learning in the CartPole environment from [75], (b) hardware-efficient ansatz for policy gradient method in the CartPole environment from [150], (c) equivariant quantum circuit for Q-learning and policy gradient method in the TSP environment from [174]. For (a) and (b) we use 5 repetitions of the template shown above, while for (c) we use just one layer.

implementation used in [150] and made available at [283], which uses five layers of the same hardware-efficient ansatz used for the Q-learning agent, except that each layer has an additional trainable rotation around the x-axis on each qubit (see Figure 7.2(b)), and the action observables are defined as $O_L = Z_1Z_2Z_3Z_4$ and $O_R = \mathbb{I} - O_L$. As before, input features are multiplied with an additional trainable parameter each. Since the policy is a probability distribution, a final SoftMax layer is used to map the expectation values $\langle O_a \rangle_{s,\boldsymbol{\theta}} \in [-1, 1]$ to the appropriate range $[0, 1]$, and so probabilities for each action eventually become

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{e^{\beta \langle O_a \rangle_{s,\boldsymbol{\theta}}}}{\sum_{a'} e^{\beta \langle O_{a'} \rangle_{s,\boldsymbol{\theta}}}}, \tag{7.1}$$

where $\beta \in \mathbb{R}$ is a also a trainable parameter. A depiction of both circuits can be seen in Figure 7.2 a) and b).

7.1.2 Traveling Salesperson Problem

The second environment that we study is the TSP environment from Chapter 6, where again the Q-learning agent and the environment are implemented as described in Section 6.2 and the ansatz can be seen in Figure 7.2 c). For policy gradient agents the ansatz is the same as in the Q-learning case, but as the policy has to be a probability distribution we again use a final SoftMax layer with a trainable inverse temperature β on the observable, as in Equation (7.1). The authors of [150] have shown that using this type of final layer can be highly beneficial for policy gradient training, compared to only using the probability distribution resulting from the quantum state directly. This is due to the fact that the trainable inverse temperature enables the agent to tune its level of exploration of the state space. As the optimal solutions to TSP instances are deterministic, it is favourable in this environment to have a tunable inverse temperature that allows exploration of the large state space early in training, as well as close-to-deterministic decisions towards the end.

7.2 Shot noise

We start our studies with the type of noise that is arguably the simplest to characterize: noise induced by statistical errors that result from the probabilistic nature of quantum measurements. For each circuit evaluation, be it for action selection of the RL agent or for computing parameter updates via the parameter

shift rule, we take a fixed number of measurements M and compute the resulting expectation value. The precision ϵ of this expectation value depends on M and scales as $\epsilon \sim 1/\sqrt{M}$, as we will explain in more detail below.

Variational algorithms often require a very large number of measurements to be executed, and this problem is exacerbated in QML tasks that typically involve separate circuit evaluations for all training data points. For this reason, it is not only important to understand the effect of shot noise on the trainability and performance of QML models, but it is also desirable to develop methods that lead to a smaller shot footprint than simply assigning a fixed number of shots to each circuit evaluation. Depending on knowledge of the algorithm itself, it can be possible to make an informed decision on the number of shots that suffice in each step. In this section, we develop such a method specifically for Q-learning that is a natural extension to the original algorithm.

7.2.1 Reducing the number of shots in a Q-learning algorithm

As described in Section 3.2.2, a Q-learning agent selects actions based on the following rule (see Equation (3.18))

$$a_t = \text{argmax}_a Q_\pi(s_t, a; \boldsymbol{\theta}),$$

that is, it chooses actions according to the largest Q-value.[1] Now, consider a quantum agent that only has access to noisy estimates of the Q-values $\tilde{Q}(s_t, a_t; \boldsymbol{\theta})$ resulting from the statistical uncertainty of a measurement process involving a finite number of shots M. If the sample size is large enough $M \gg 1$, then by the central limit theorem each noisy Q-value can be described as a random variable

$$\tilde{Q}(s_t, a_t; \boldsymbol{\theta}) = Q(s_t, a_t; \boldsymbol{\theta}) + \epsilon\,, \tag{7.2}$$

where $Q(s_t, a_t; \boldsymbol{\theta})$ is the true noise-free value, and ϵ is a random variable sampled from a Gaussian distribution centered in zero $\mu_\epsilon = \mathbb{E}[\epsilon] = 0$, and with standard deviation inversely proportional to the square root of the number of measurement shots $\sigma_\epsilon = \text{Std}[\epsilon] \sim 1/\sqrt{M}$. Since actions are selected through an argmax function,

[1] In the ϵ-greedy policy (see Section 3.2.2) we consider here, the agent picks either the action corresponding to the argmax Q-value, or a random action. As no circuit evaluation is required to pick a random action, we only consider the steps with actual action selection by the agent in this section.

the perturbation ϵ will not affect the action selection process as long as the order between the largest and the remaining Q-values remains unchanged. Then, one may ask: is there a minimal number of shots that suffice to reliably distinguish the largest Q-value Q_{max} and the second-largest Q-value Q_2?

When the observables associated to the actions are non-commuting, they have to be estimated independently from each other, and one has the freedom of choosing how to allocate the measurement shots among the observables of interest, possibly in a clever way. In our case, the goal is to estimate which of the observables has the highest Q-value while trying to be shot-frugal, and this task can be related to the theory of multi-armed bandits [284]. The multi-armed bandit is a RL problem in which an agent can allocate only a limited amount of resources between a number of choices, e.g., a number of arms on a bandit machine, and is asked to determine which of these choices leads to the highest expected reward. There exists a trade-off between exploration (i.e., trying the different arms) and exploitation (always choosing the arm that appears best according to the current knowledge), and the upper confidence bound (UCB) [285, 286] algorithm shows how to use statistical confidence bounds to allocate exploratory resources. The UCB algorithm could be used in the scenario described above where a number of non-commuting observables have to be estimated, and we want to find the optimal strategy to allocate a fixed budget of measurement shots to the task of identifying the largest Q-value.

However, in the specific implementations of QRL agents based on recent literature that we study in this work [150, 75, 174], only commuting observables are used, hence it is not necessary to apply the UCB procedure to determine which one should be measured more often. Nonetheless, inspired by the UCB algorithm, we can still define a rather general simple heuristic that can be used to reduce the overall number of shots required to train the Q-learning models as those studied in this work. The idea is to use the knowledge about the scaling of the estimation error with respect to the number of measurements (see Equation (7.2)), to determine with confidence whether we have taken enough shots to determine the maximum Q-value.

The procedure goes as follows. First, we take a small number of initial measurements m_{init}, for example $m_{\text{init}} = 100$, of all observables to compute the estimates $\tilde{Q}_{m_{\text{init}}}(s_t, a),\ \forall\ a \in \mathcal{A}$. Based on these values, we compute the absolute difference between the largest and the second largest Q-values. If this difference is larger

Algorithm 1 Algorithm to reduce the number of measurements in Q-learning

Input m_{init}, m_{inc}, m_{max}

Output m_{est}

```
m_est ← m_init
while m_est < m_max do
    Q̃(s_t, a_1) = ⟨O_{a_1}⟩_{m_est}
    Q̃(s_t, a_2) = ⟨O_{a_2}⟩_{m_est}
    ΔQ̃ = |Q̃(s_t, a_1) − Q̃(s_t, a_2)|
    if ΔQ̃ < 2/√m_est then
        m_est ← m_est + m_inc
    else
        return min(m_est, m_max)
    end if
end while
return min(m_est, m_max)
```

than twice the estimation error $\epsilon = 2/\sqrt{m_{\text{init}}}$ (as both of the Q-values are noisy), we have found the largest Q-value with high confidence and we stop here. On the other hand, if the difference is smaller, we increment the sample size with additional m_{inc} measurements each, and recompute the estimated Q-values with the $m_{\text{inc}} + m_{\text{init}}$ shots. We again compute the absolute difference of the two largest Q-values and determine whether the number of measurements suffices based on the error $\epsilon = 2/\sqrt{m_{\text{init}} + m_{\text{inc}}}$. This measure-and-compare scheme is performed until either the two largest Q-values can be distinguished with high confidence, or a fixed shot budget m_{max} is reached. In Algorithm 1 we provide a description of this procedure, where for the sake of simplicity we describe the case where there are only two possible actions, and we therefore only have to find the larger of two Q-values. However, the scheme can be used for an arbitrary number of Q-values, as it is only important to distinguish between the highest and the second-highest Q-value with high confidence. The algorithm takes as input the number of initial measurements m_{init}, the number of additional measurements in every step m_{inc}, and the maximum number of measurements that are allowed in one run of the shot-allocation algorithm (i.e., finding the largest Q-value) m_{max}. The output is the number of measurements m_{est} that are sufficient to find the argmax Q-value with high confidence based on the rules above. The values $\langle O_{a_i} \rangle_{m_{\text{est}}}$ are the expectation values of observables O_{a_i} corresponding to action a_i, estimated

with m_{est} shots. Note that the proposed scheme works both for commuting or non-commuting observables, where in the former case one can spare shots by computing the observables from the same set of measurement outcomes. Moreover, note that we ignore the coefficients in the statistics of the Q-values coming from **??**, when considering the measurement stopping criterion. This choice has no impact on the effectiveness of the proposed method, as it is always found to be very well performing in the presented form.

While this algorithm can clearly determine the optimal number of shots in the action selection process in a methodical manner, one should check that this will not introduce errors in the remaining parts of the variational Q-learning model, i.e., during the parameter update step. Recall that each parameter update of the model is computed based on the output of the model itself (see Equation (3.19))

$$Q_\pi(s_t, a_t; \boldsymbol{\theta}) \leftarrow r_{t+1} + \gamma \max_a Q_\pi(s_{t+1}, a; \boldsymbol{\theta}),$$

which means that in the parameter update step we do not need to perform action selection, but instead care about the actual Q-values in order to compute the loss function. The question is now to what precision we need to approximate the Q-values in order to learn a good Q-function. Technically, even the noise-free Q-function is only an approximation of the true Q-function, which is the whole point of doing Q-learning with function approximators. This suggests that there is some leeway to make even the approximate function itself an approximation by taking only as many measurements as are necessary to find the argmax Q-value with high confidence. Indeed, it has been shown in [75] that even the Q-functions of agents that successfully solve an environment can produce Q-values that are far from the optimal Q-values, and that learning the correct order of Q-values is more important in this setting than approximating the optimal Q-value as precisely as possible. Consequently, when we compute the Q-values that are used to perform parameter updates, we use the same algorithm as that in Algorithm 1 to determine the number of measurements to take.

7.2.2 Numerical results

We now numerically compare the performance of agents in the CartPole and TSP environments in settings where a fixed number of shots is used in each circuit evaluation, and where the number of shots in each step is determined by the algorithm we introduced in Section 7.2.1. To give an overview of the number of

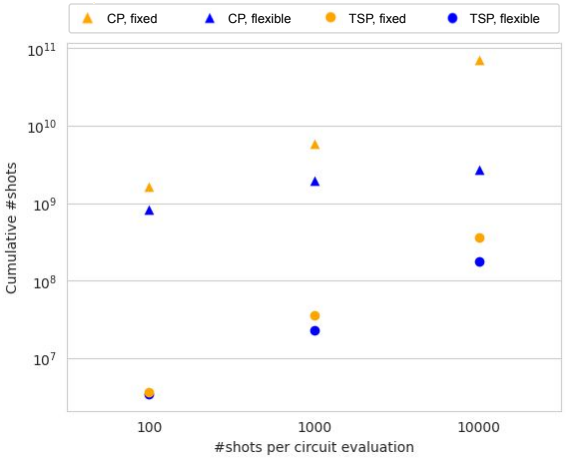


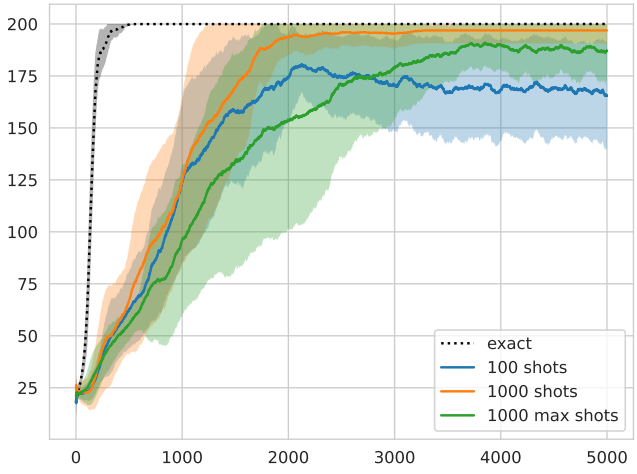
Figure 7.3: Comparison of the cumulative number of shots per observable over a full training run, for the flexible shot allocation technique (blue) and for a standard fixed measurement scheme using the same number of shots for every circuit evaluation (orange), both for CartPole (triangles) and TSP (circles). Each data point shows the average over ten trained agents.

shots used in one training run under varying hyperparameter settings, we show the average cumulative number of shots for different settings in Figure 7.3. For the CartPole environment (triangles), the number of cumulative shots grows quickly with the number of shots in each step in the fixed setting (orange). This is not true for the flexible shot allocation technique (blue), where for values of $m_{\max} \in \{100, 1000, 10000\}$ the cumulative number of shots is relatively similar. As we see in Figure 7.4 a), a low number of shots such as 1000 is already sufficient to achieve close to optimal performance in the CartPole environment. Therefore, we focus on comparing settings with 100 and 1000 (maximum) shots per circuit evaluation in that figure. Comparing the cumulative number of shots for $m_{\text{fixed}} = 100$ and $m_{\max} = 1000$ in Figure 7.3, we see that these two configurations use almost the same number of measurements overall. Still, the final performance of the agents trained with the flexible shot allocation technique is almost optimal, while those trained with a fixed number of shots in each circuit evaluation are below a final score of 175 on average. However, as we allow agents to use even less than 100 shots per evaluation with the flexible allocation method of Algorithm 1, performance starts to degrade, so at least 100 shots are required in this setting. To not clutter

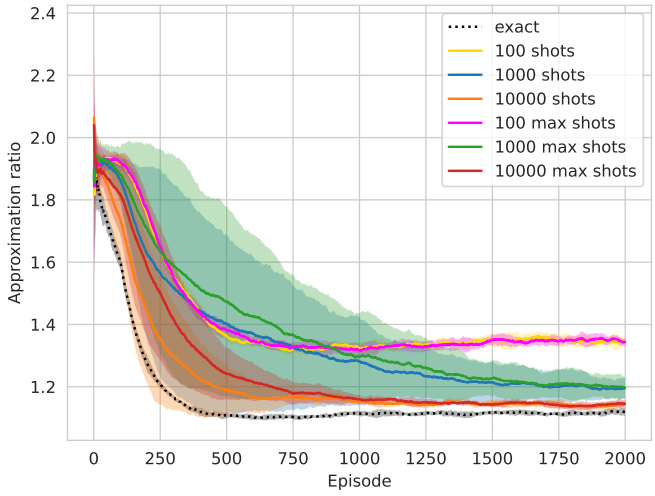
the figure we show the results for agents that use fewer than 100 shots per circuit evaluation in Figure 6 in the Appendix.

In the TSP environment, each step in an episode constitutes of a constant and (compared to CartPole) relatively low number of circuit evaluations. We still see that the higher the setting for the (maximum) number of shots is, the bigger the gap in average cumulative number of shots becomes. For agents trained in the TSP environment, shown in Figure 7.4 b), the final performance remains unchanged by the additional noise introduced by the flexible shot allocation technique, and agents reach the same accuracy of those trained with a corresponding but fixed number of shots per circuit evaluation. The only difference between the two approaches is that the agents using the flexible shot allocation method take slightly longer to converge in some cases. Independently from the estimation method used (flexible or fixed), it is clear from Figure 7.4 that it is the number of shots available that plays the major role in determining the performance of the noisy agents, as measured by the proximity to the average approximation ratios reached in the noise-free scenario, namely when agents have access to exact the expectation values ($M \to \infty$). In this environment, there is a trade-off between delayed convergence due to less precision in the approximation of the Q-function, and using a higher number of shots to arrive at the same final performance.

To summarize, we have seen that Q-learning models can be successfully trained even in the presence of statistical noise introduced by a measurement processes carried out with a limited number of shots. In addition, by leveraging the specifics of the Q-learning algorithm, we introduced an easy-to-implement and effective method that can be used to reduce the number of shots needed to train variational Q-learning agents. How many shots one can save during training with this method depends on the agents' resilience to shot noise, as well as the specific characteristics of the environment. In the CartPole environment, where one bad decision does not lead to immediate failure, the additional noise introduced by estimating expectation values with a low number of measurements and approximating an imprecise Q-function does not affect performance severely. In the TSP environment on the other hand, where one bad choice of the next city in the tour can lead to a much longer path, we observe that the number of measurements has to be relatively high to get close to optimal performance. However, even in this setting we can achieve a reduction in the overall number of measurements by taking an informed approach at when to measure an observable more often.



(a) CartPole



(b) Traveling Salesperson Problem

Figure 7.4: Comparison of Q-learning with shot noise using the informed shot-allocation method (labeled “max shots") proposed in this work, and a standard measurement scheme that simply assigns a fixed number of shots to each circuit evaluation (labeled “shots"). Results are averaged over ten agents for each configuration. (a) Shows results for agents trained in CartPole environment, (b) shows results for agents in the TSP environment.

7.3 Coherent noise

In this section, we turn our attention to coherent noise, that is, errors that preserve the unitary evolution of the quantum circuit but still change its output [287]. In our analysis, we model coherent noise as an *over-* or *under-rotation* of the parametrized gates, by adding a random Gaussian perturbation to the variational parameters in the considered circuits.

This type of error could occur in real quantum devices as a drift in the parameters for example due to an imperfect control of the system or a miscalibration of the hardware, and it is therefore an important component of the overall picture of an imperfect quantum device. Specifically, we assume that the perturbation remains unchanged during the estimation a given observable, i.e. it does not change considerably between repeated measurements on the same experimental setup. However the perturbation amount change whenever the experiment is changed, for example due to measuring a different observable, or using the circuit with a different set of parameters.

Gaussian coherent noise is also an interesting model because it lends itself very well to theoretical analysis, and one can estimate the effect of such an error on the output of a parameterised quantum circuit. In the following, we first proceed with an analytical treatment of the error introduced by Gaussian perturbations on variational circuits, and then proceed with the numerical results for the two environments considered in this work.

7.3.1 Effect of Gaussian coherent noise on circuit output

Consider a general parametrized quantum circuit acting on a system of n qubits, with unitary $U(\boldsymbol{\theta}) \in \mathbb{C}^{2n} \times \mathbb{C}^{2n}$ and parameter vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_M) \in \mathbb{R}^M$. Let O be on observable and $\rho = |0\rangle\langle 0|$ the initial state of the quantum system, the outcome of the variational circuit is the expectation value

$$f(\boldsymbol{\theta}) = \langle O \rangle_{\boldsymbol{\theta}} = \mathrm{Tr}\left[O\, U(\boldsymbol{\theta}) \rho U^\dagger(\boldsymbol{\theta})\right]. \tag{7.3}$$

Suppose that the parameters are affected by a noise process that adds a perturbation

$$\boldsymbol{\theta} \to \boldsymbol{\theta} + \delta\boldsymbol{\theta}\,, \tag{7.4}$$

where $\delta\boldsymbol{\theta} = (\delta\theta_1, \ldots, \delta\theta_M) \in \mathbb{R}^M$ are *i.i.d.* according to a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ with zero mean $\mu = 0$ and equal variance σ^2, namely

$$\begin{aligned} &\delta\theta_i \sim \mathcal{N}(0, \sigma^2)\,, \\ &\mathbb{E}[\delta\theta_i] = 0\,, \qquad\qquad \forall i \in \{1, \ldots, M\} \\ &\mathbb{E}[\delta\theta_i \delta\theta_j] = \sigma^2 \delta_{ij}\,. \end{aligned} \tag{7.5}$$

As discussed earlier, in our analysis in this section and in the numerical simulations in section 7.3.3.1, we assume that the perturbed parameters remain the same during the evaluation of a single expectation value. In a real experiment on quantum hardware, this would mean that for all measurements used to estimate the expectation value, the perturbations stay at least approximately unchanged. Of course, without this assumption, the resulting noise model could not be considered unitary, and one may then resort to a noise channel formulation of Gaussian noise as proposed in [261, 280]. Hence, in the following we restrict our attention to the setting described above.

The effect of Gaussian noise on the circuit can be evaluated by Taylor expanding the circuit around the unperturbed parameters $\boldsymbol{\theta}$. For ease of explanation, we hereby report only the main ideas and results, and we refer to Appendix 8 for a complete and detailed derivation of all the calculations performed in this section.

Let $f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})$ be the function evaluated on the perturbed parameters, its Taylor expansion up to fourth-order reads

$$\begin{aligned} f(\boldsymbol{\theta} + \delta\boldsymbol{\theta}) \approx & f(\boldsymbol{\theta}) + \sum_{i=1}^{M} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_i} \delta\theta_i + \frac{1}{2} \sum_{i,j=1}^{M} \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j} \delta\theta_i \delta\theta_j \\ & + \frac{1}{3!} \sum_{i,j,k=1}^{M} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j \partial \theta_k} \delta\theta_i \delta\theta_j \delta\theta_k + \mathcal{O}\big(\delta\theta^4\big)\,. \end{aligned} \tag{7.6}$$

With this expression one can evaluate the expected value of the noisy function $\mathbb{E}[f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})]$ over the distribution of the Gaussian perturbations, $\mathbb{E}(\cdot) = \mathbb{E}_{\delta\theta_i \sim \mathcal{N}(0,\sigma^2)}(\cdot)$. Since every odd moment of a Gaussian distribution vanishes, using

relations (7.5) in the expansion (7.6) one obtains

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}+\delta\boldsymbol{\theta})] &\approx f(\boldsymbol{\theta}) + \frac{1}{2}\sum_{ij}\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j}\mathbb{E}[\delta\theta_i\delta\theta_j] \\ &\approx f(\boldsymbol{\theta}) + \frac{1}{2}\sigma^2\sum_{ij}\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j}\delta_{ij} \\ &\approx f(\boldsymbol{\theta}) + \frac{1}{2}\sigma^2\,\mathrm{Tr}[H(\boldsymbol{\theta})] + \mathcal{O}(\sigma^4),\end{aligned} \tag{7.7}$$

where $\mathrm{Tr}[H(\boldsymbol{\theta})]$ denotes the trace of the Hessian matrix

$$H_{ij}(\boldsymbol{\theta}) = \frac{\partial^2 f(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j} \quad i,\, j = 1\ldots, M. \tag{7.8}$$

Thus, the first non-vanishing correction term caused by the noise is proportional to the noise variance σ^2, and the Hessian of the parametrized quantum circuit, which conveys geometric information about the curvature of the function landscape around the unperturbed point $\boldsymbol{\theta}$.

Higher-order terms in the expansion can be evaluated in a similar way, specifically making use of so-called *Wick's relations* for multivariate normal distributions as shown in Appendix 8. If all the derivatives of the function $f(\boldsymbol{\theta})$ are bounded, as it is the case for parametrized quantum circuits, then it is possible to derive an upper bound on the error induced by the perturbations which only depends on the noise strength σ^2 and the total number of parameters M, as we show in the following.

Using the parameter shift rule [33, 39], one can show that any derivative of a parametrized quantum circuit can be expressed as a linear combination of circuit outcomes evaluated at specific points in parameter space [280, 51]. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M) \in \mathbb{N}^M$ be a multi index keeping track of the order of partial derivatives, define the derivative operator

$$\partial^{\boldsymbol{\alpha}} := \frac{\partial^{|\boldsymbol{\alpha}|}}{\partial\theta_1^{\alpha_1}\cdots\partial\theta_M^{\alpha_M}}\,, \tag{7.9}$$

where $|\boldsymbol{\alpha}| := \sum_{i=1}^{M}\alpha_i$. By nested applications of the parameter shift rule, one can show that

$$\partial^{\boldsymbol{\alpha}} f(\boldsymbol{\theta}) = \frac{1}{2^{|\boldsymbol{\alpha}|}}\sum_{i=1}^{2^{|\boldsymbol{\alpha}|}} s_m\, f(\boldsymbol{\theta}_m)\,, \tag{7.10}$$

where $s_m \in \{\pm 1\}$ are signs, and $\boldsymbol{\theta}_m$ are parameters obtained shifting the parameter vector $\boldsymbol{\theta}$ along different directions. Now, since the measurement outcome of every circuit is bounded by the maximum absolute eigenvalue of the observable, i.e. $|f(\boldsymbol{\theta})| \leq \|O\|_\infty$, consequently it also holds that $|\partial^{\boldsymbol{\alpha}} f(\boldsymbol{\theta})| \leq \|O\|_\infty$ (see Appendix 8). Note that we only consider bounded observables here, like the Pauli operators commonly used in variational RL algorithms [153, 154, 150, 75].

Since all the derivatives of the function are bounded, it is possible to bound every term in the Taylor series and then compute an upper bound to the error caused by the perturbation. In fact, defining the absolute (average) error caused by the noise as

$$\varepsilon_{\boldsymbol{\theta}} := \left|\mathbb{E}[f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})] - f(\boldsymbol{\theta})\right|, \tag{7.11}$$

one can prove that this is upper bounded by (see Appendix 8)

$$\varepsilon_{\boldsymbol{\theta}} \leq \|O\|_\infty \left(e^{\sigma^2 M/2} - 1\right). \tag{7.12}$$

Note that since $\varepsilon_{\boldsymbol{\theta}} \leq 2\|O\|_\infty$ is always true, the bound is informative only as long as $e^{\sigma^2 M/2} - 1 < 2$.

This expression only depends on the noise strength σ^2, the total number of noisy parameters M, and the operator norm of the observable $\|O\|_\infty$, and it can be used to estimate a *sufficient* condition on the noise strength to guarantee a desired error threshold $\varepsilon_{\boldsymbol{\theta}}$. Rearranging Equation (7.12), a sufficient condition to have error $\varepsilon_{\boldsymbol{\theta}}$ not larger than ϵ, is to have Gaussian perturbations satisfying

$$\sigma \leq \sqrt{\frac{2}{M}} \log\left(1 + \frac{\epsilon}{\|O\|_\infty}\right). \tag{7.13}$$

As the allowable error is small $\epsilon \ll 1$, by approximating the logarithm $\log(1+x) \approx x$, one derives that the perturbations must follow the scaling

$$\sigma \in \mathcal{O}\left(\frac{\epsilon}{M^{1/2}\|O\|_\infty}\right). \tag{7.14}$$

Note that a similar scaling law was recently derived also in [280], though via a slightly different method based on the moment generating function of the probability distribution characterising the perturbations.

To provide an example, assume one is willing to tolerate an error of $\epsilon = 10\%$, that $\|O\|_\infty = 1$ as for measuring a Pauli operator and that the PQC consists of $M = 100$ noisy parametrized gates, then one can be sure of such accuracy if

$\sigma \sim 0.1/\sqrt{100} = 0.01$. However, we stress again that the scaling Equation (7.13) is only a *sufficient* but not necessary condition for achieving an error ϵ. In fact, apart from the requirement of bounded derivatives, Equation (7.13) is agnostic with respect to the specifics of the function, and such bound can be quite loose in real instances where a much larger noise level still causes a small error, as shown in Figure 7.5.

In Figure 7.5, we report simulation results obtained by simulating the parametrized ansatz depicted in Figure 7.2(b) subject to Gaussian coherent noise of increasing strength. It is clear that the output of the circuit closely follows the approximation of Equation (7.7) given by the Hessian even at moderately large value of the noise $\sigma \lessapprox 0.15$. When the noise is too strong ($\sigma > 0.2$), the circuit becomes essentially random, and the average expectation value when measuring a Pauli operator is zero. This is a consequence of PQCs often behaving like unitary designs upon random initialization of the parameters [229, 44], a fact which we discuss in detail in Sec. 7.3.2. At last, as discussed earlier, while the upper bound (7.12) holds, it is indeed very loose and only holds tightly at small $\sigma \lessapprox 0.01$.

We now proceed discussing why hardware-efficient parametrized quantum circuits can be resilient to Gaussian coherent noise. Roughly, this is because such circuits are found to behave like random unitaries upon random assignment of the parameters, which implies that the derivatives of such circuits tend to vanish as the system size grows large [51].

7.3.2 Resilience of Hardware-Efficient ansatzes to Gaussian coherent noise

The previous analysis showed that Gaussian perturbations induce an error depending on the Hessian of the circuit (see Equation (7.7)), so that up to fourth order in the perturbation it holds that

$$\mathbb{E}[f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})] \approx f(\boldsymbol{\theta}) + \frac{1}{2}\sigma^2 \operatorname{Tr}[H(\boldsymbol{\theta})]\,. \tag{7.15}$$

This equation tells us that if the optimization landscape is flat or close to being flat, then the Hessian is small, and so the perturbation will have little effect on the output of the circuit. On the contrary, in the presence of a very curved landscape, noise will have a great impact and the output of the circuit may change sensibly. It is known that the curvature of the optimization landscape produced by a PQC is

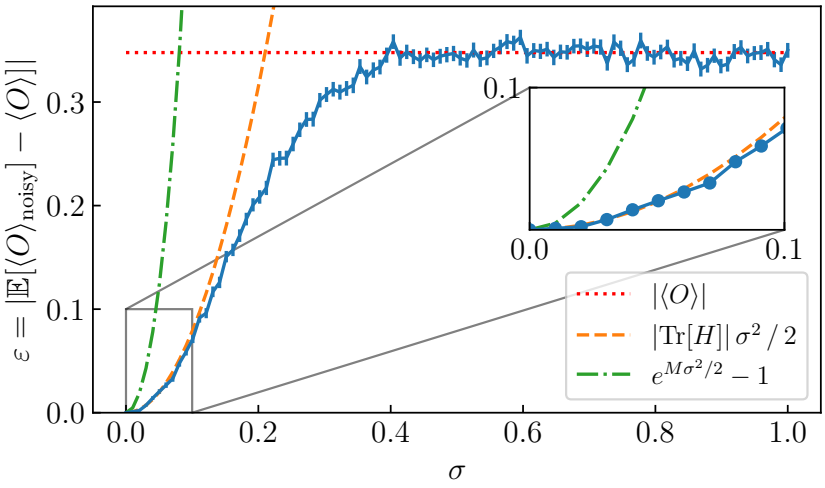


Figure 7.5: Effect of Gaussian coherent noise on the output of the parametrized quantum circuit shown in Figure 7.2(b). The plot is obtained by first choosing a parameter vector $\boldsymbol{\theta}_0 \in \mathbb{R}^{92}$ corresponding to a the ideal noise-free expectation value $f(\boldsymbol{\theta}_0) = \langle O \rangle$ with $O = Z^{\otimes 4}$. With this baseline fixed, random Gaussian perturbations are added to the angles $\boldsymbol{\theta}_{noisy} = \boldsymbol{\theta}_0 + \delta\boldsymbol{\theta}$, and the resulting noisy expectation vales $\langle O \rangle_{noisy}$ are computed. Each point in the plot is the average over $N = 10^5$ different perturbation vectors sampled from a multivariate Gaussian distribution of a given σ. The experiments are then repeated for increasing values of the noise strength σ. The error bars show the statistical error of the mean. For small noise levels, the output of the quantum circuit closely follows the behaviour predicted by Equation (7.7), where the Hessian is evaluated at the unperturbed value $H = H(\boldsymbol{\theta}_0)$. When the error is too large the circuit behaves as a random circuit whose output is on average zero, hence the error plateaus to the unperturbed expectation value $\varepsilon = |\langle O \rangle| = |f(\boldsymbol{\theta}_0)|$). The upper bound predicted by Equation (7.12) is very loose in general, and holds tightly only for very small values of $\sigma \lessapprox 0.01$.

closely related to the barren plateau phenomenon [229, 44, 48], where the variance of the first and second derivative vanishes exponentially in the number of qubits and layers in a random circuit. Additionally, the hardware-efficient ansatz we use for some of the environments in this work is known to suffer from barren plateaus when the system size is large. As the curvature of the optimization landscape of these types of circuits is very flat, it can also be expected that the type of noise induced by the Gaussian perturbations on parameters that we study in this work should not affect circuits that generally produce small first and second order derivatives. While circuits that are in the barren plateau regime are obviously undesirable as they quickly become untrainable, one can consider circuits of the size such that the variance in gradients is relatively small, but the circuit has not yet converged to an approximate 2-design, as shown in [229]. We make this statement more formal in the following.

We can use standard results on averages of unitary designs [288, 289] to characterize the Hessian of hardware-efficient circuits, and thus gain insight on their performance under Gaussian noise. We report the main results of our analysis here, full derivations can be found in Chapter 8. In the following, we suppose that sampling a random value of the parameter vector $\boldsymbol{\theta}$ in the parametrized circuit $U(\boldsymbol{\theta})$, is equivalent to sampling a unitary from a unitary 2-design, defined as a set of unitary matrices that match the Haar distribution up to the second moment. Also, we consider observables O being Pauli strings, so that $\mathrm{Tr}[O] = 0$ and $\mathrm{Tr}\big[O^2\big] = 2^n$. In order to distinguish from the previous notation where averages were computed over the Gaussian distribution of the perturbations, we use $\mathbb{E}_U[\cdot]$ and $\mathrm{Var}_U[\cdot]$ to denote average values and variances evaluated over the random unitaries.

Then, under reasonable and usual assumptions on parts of the parametrized quantum circuit being 2-designs, it is possible to show that the diagonal elements of the Hessian $H_{ii} = \partial^2 f(\boldsymbol{\theta})/\partial\theta_i^2$ satisfy [51] (see also Appendix 8 for an explicit derivation)

$$\mathbb{E}_U[H_{ii}] = 0\,, \quad \mathrm{Var}_U[H_{ii}] \in \mathcal{O}\left(\frac{1}{2^n}\right). \tag{7.16}$$

That is, in addition to first order derivatives, also second order derivatives of random parameterized quantum circuits are found to be zero on average, and with a variance which is exponentially vanishing.

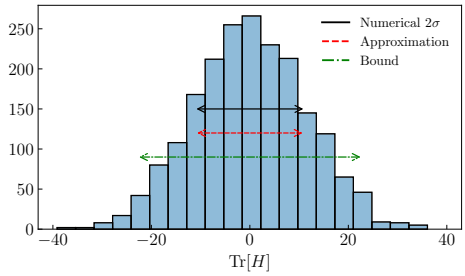


Figure 7.6: Simulation results of evaluating the trace of the Hessian matrix for the circuit shown in Fig. 7.2(b) with random assignments of the parameters and $O = Z^{\otimes 4}$. The simulations are performed by sampling 2000 random parameter vectors $\{\boldsymbol{\theta}_m\}_{m=1}^{2000}$ with $\theta_i \sim \text{Unif}[0, 2\pi[$ and then evaluating the trace of the corresponding Hessian matrix $\text{Tr}[H(\boldsymbol{\theta}_m)]$. These values are used to build the histogram showing the frequency distribution of $\text{Tr}[H]$. The length of the arrows are, respectively: "Numerical 2σ" (black solid line) twice the numerical standard deviation, "Approximation" (dashed red) twice the square root of the approximation in Eq. (7.18), "Bound" (dashed-dotted green) twice the square root of the upper bound in Eq. (7.17).

Starting from the results above, one can calculate the statistics of the trace of the Hessian, for which it holds

$$\mathbb{E}_U[\text{Tr}[H]] = 0\,, \quad \text{Var}_U[\text{Tr}[H]] \lessapprox \frac{M^2}{2^n}\,. \tag{7.17}$$

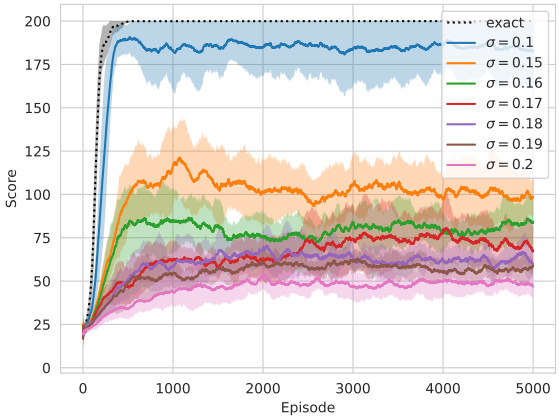
Furthermore, our numerical simulations suggest that the variance of the trace of the Hessian is actually smaller, and is well captured by the following expression

$$\text{Var}_U[\text{Tr}[H]] \approx \frac{M(M+1)}{4(2^n+1)} \approx \frac{1}{4}\frac{M^2}{2^n}\,, \tag{7.18}$$

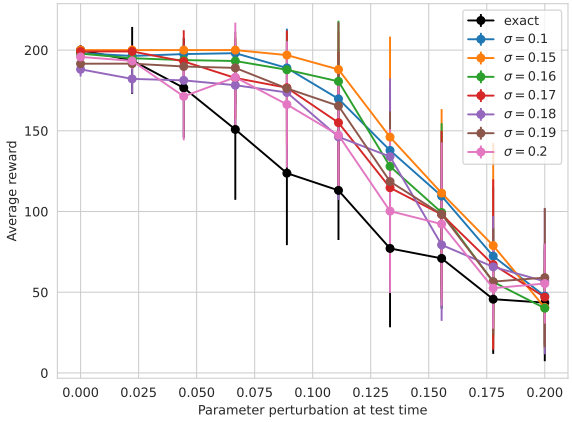
a fact which we justify and discuss in Chapter 8.

In Figure 7.6 we report simulation results of evaluating the trace of the Hessian matrix for the circuit shown in Figure 7.2(b). The histogram represents the frequency of obtaining a given value of the trace of the Hessian $\text{Tr}[H(\boldsymbol{\theta})]$ upon random assignments of the parameters. Indeed, there is a very good agreement between the variance obtained via numerical simulations (black solid line), and the one calculated with the approximation (7.18) (dashed red line).

The circuit used has $M = 92$ parameters and $n = 4$ qubits, and plugging these values in Equation (7.18) yields a standard deviation $\sigma_U = \text{Std}_U[\text{Tr}[H]] \approx 11$. Then, if the behaviour of the PQCs in practical scenarios is well described by its

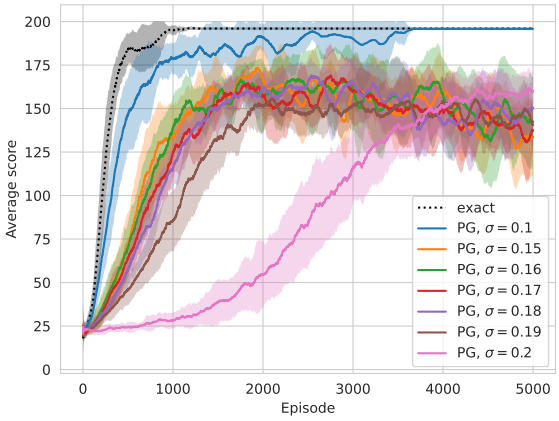


(a) training performance

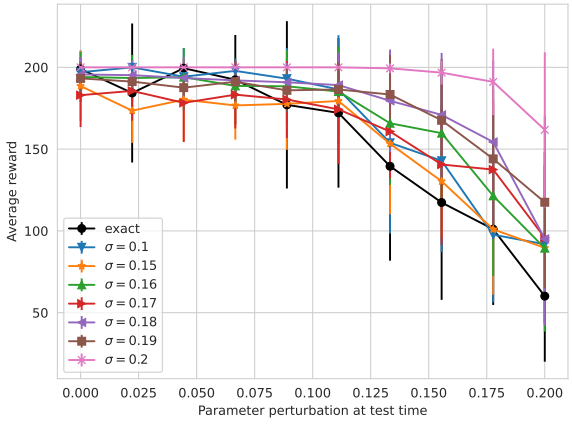


(b) evaluation performance

Figure 7.7: Q-learning agents on the CartPole environment trained and evaluated at varying perturbations σ. Panel (a) shows training performance, while panel (b) shows the performance of the same agents after training and evaluated under different perturbation levels than those present during training. Each point is computed as the average score of the 10 agents under the perturbation indicated on the x-axis.



(a) training performance



(b) evaluation performance

Figure 7.8: Policy gradient agents on the CartPole environment trained and evaluated at varying perturbations σ. Panel (a) shows training performance, while panel (b) shows the performance of the same agents after training and evaluated under different perturbation levels than those present during training. Each point is computed as the average score of the 10 agents under the perturbation indicated on the x-axis.

random parameter regime, one expects the trace of the Hessian to be on average zero and in general not much bigger (in absolute value) than $\sigma_U \approx 11$. With this order of magnitude for the trace, the first order correction Equation (7.15) even with a Gaussian noise level of $\sigma = 0.1$ is very small, as it amounts to

$$|\mathbb{E}[f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})] - f(\boldsymbol{\theta})| \approx \frac{1}{2}\sigma^2 |\mathrm{Tr}[H(\boldsymbol{\theta})]| \approx 0.05\,.$$

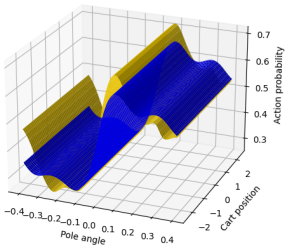
Summing up, for those PQCs whose cost landscape is close to being flat, then Gaussian perturbations on the variational parameters will have a limited impact on the output of the quantum circuit.

7.3.3 Numerical results

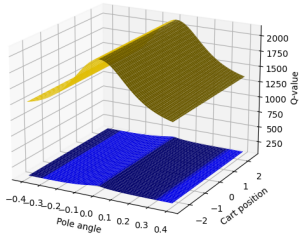
7.3.3.1 CartPole

First, we evaluate the performance of policy gradient and Q-learning algorithms when Gaussian perturbations are applied at each circuit evaluation during training. In Figure 7.7 (a) and (b), we show the training and evaluation performance, respectively, of Q-learning agents in the CartPole environment with perturbations in the range $\sigma \in \{0, 0.1, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2\}$. Only the agent trained with noise level $\sigma = 0.1$ learns the environment successfully and remains close to optimal performance. As suggested by our theoretical analysis in Section 7.3.1, performance starts to degrade as we consider higher perturbations of $\sigma > 0.1$, and none of those agents manage to achieve a better performance than a score of 125 on average. In Figure 7.7 (b) we evaluate the performance of trained agents when they act in an environment with different perturbation levels than those present when they were trained. Even agents that do not perform well during training achieve close to optimal performance when evaluated in the noise-free setting. This suggests that despite their bad training performance due to the added perturbations, these agents still learn a good Q-function. Notably, the agents trained without noise perform worst when they are evaluated under various levels of perturbations.

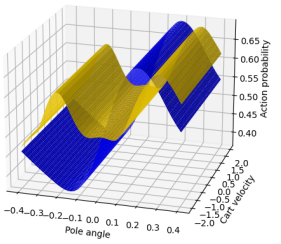
Results for agents trained with the policy gradient method are shown in Figure 7.8 (a). While again only the agents trained with a perturbation of $\sigma = 0.1$ perform well and even reach optimal performance, agents with higher perturbations also largely stay close to optimal performance with a final score of 125 on average. Even the agent trained with a relatively high $\sigma = 0.2$ is robust in this setting, even



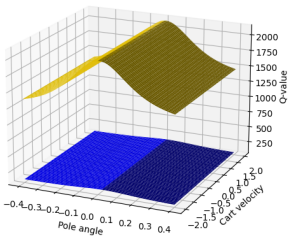
(a) pole angle, cart position (PG)



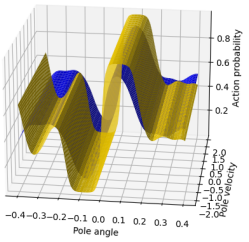
(b) pole angle, cart position (QL)



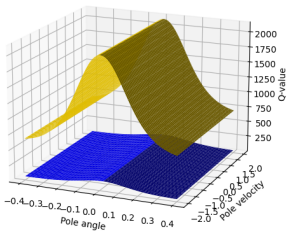
(c) pole angle, cart velocity (PG)



(d) pole angle, cart velocity (QL)



(e) pole angle, pole velocity (PG)



(f) pole angle, pole velocity (QL)

Figure 7.9: Comparison of average learned policies (PG) and Q-functions (QL) of agents from Figure 7.7 and Figure 7.8, in the noise-free setting (blue) and with a perturbation level $\sigma = 0.2$ (yellow).

though it requires by far the most training episodes to get to a good score. This positive trend is also visible in Figure 7.8(b), where we see that all agents achieve close to optimal performance when evaluated with perturbation levels $\sigma \leq 0.1$, which is again in line we our theoretical analysis in section 7.3.1. The difference between agents trained with Gaussian perturbations and those trained without is not as large as in the Q-learning setting, and at evaluation time both algorithms perform similarly. Another observation about the policy gradient agents is that those trained with $\sigma = 0.2$ achieve optimal or close to optimal performance in the environment under various perturbation levels at evaluation time, and are the most robust out of all agents trained in this setting. Overall, the policy gradient method shows a larger resilience to Gaussian noise in our experiments for the CartPole environment. It is an open question why this is the case, however, we did not observe better performance of the policy gradient algorithm under noise in general, as results in later sections will show.

In addition to studying the performance of Q-learning and policy gradient agents at training and evaluation time, we visualize the learned policies and Q-functions of both in the noisy and noise-free setting in Figure 7.9. As learned policies and Q-functions can look different even when training the same agent twice, we show averages of the ten agents shown in Figure 7.7 and Figure 7.8 for both algorithms, and for perturbation levels of $\sigma = 0$ (blue) and $\sigma = 0.2$ (yellow), respectively. The CartPole environment has four inputs: cart position and velocity, and pole angle and velocity. To visualize the learned policies and Q-functions, we show the probabilities and Q-values for taking the action "right" as a function of pairs of state values. The state inputs that are not in the figure are set to zero, and for the sake of clarity we do not apply perturbations to the parameters when visualizing the policy. In Figure 7.9 (a), (c), and (e), we see results for policy gradient agents. Overall, it can be seen that the agents trained without perturbations learn smoother policies, hence for most states there is a clear decision on which action to take. Training with perturbations makes the policies slightly more rippled, but they still mostly follow the contours of the policy learned under ideal conditions.

The approximated Q-functions can be seen in Figure 7.9 (b), (d), and (f). One observation we make here is that the range that Q-values take blows up considerably compared to the noise-free setting. This is due to the trainable output weights that the expectation values are multiplied with in the Q-learning setting (see Section 7.1) becoming considerably larger for agents trained in the noisy setting. However, as

we can see in the Appendix in Figure 7, the shapes of the learned Q-functions of the noise-free and noisy agents are still very similar, which explains why even the agents trained with $\sigma = 0.2$ perform almost optimally when evaluated without perturbations in Figure 7.7 (b). We also note that the range of Q-values of both the noisy and noise-free agents is much larger than the range of optimal Q-values given in [75]. This can be understood as the agent consistently overestimating the expected return, a problem known to arise in classical Q-learning, and which is exacerbated by noise [290]. However, the authors of [75] also point out that in the function approximation setting, it is more important to learn the order of Q-values for each state (i.e., preserving that the argmax Q-value corresponds to the optimal action) than learning a close representation of the optimal Q-values.

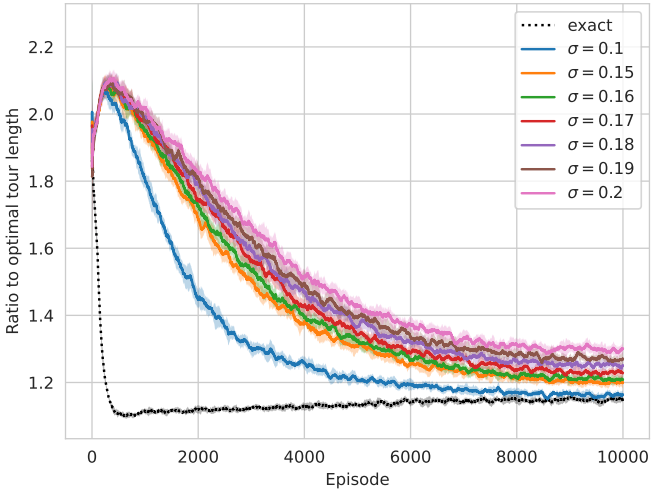
7.3.3.2 Traveling Salesperson Problem

In this section, we study the performance of Q-learning and policy gradient algorithms with Gaussian coherent noise in the TSP environment. Panels (a) and (b) in Figure 7.10 show the training and evaluation performance of Q-learning agents in this environment under perturbations in the range

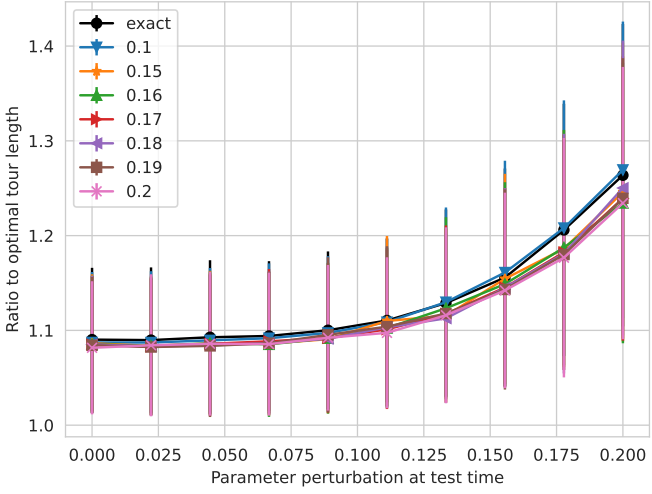
$$\sigma \in \{0, 0.1, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2\}.$$

We note that the Q-learning agents trained without noise already converge after 600 episodes on average, but to get an equal runtime in terms of episodes for all settings, we also let them run for 10000 episodes. This unnecessarily long runtime causes the optimizer to leave the local minimum again, which we ignore as an artifact here and consider the lowest average approximation ratio for the comparison with the other models.

For the TSP environment, we observe that with increasing levels of Gaussian perturbations, convergence of agents is delayed and their final approximation ratio becomes worse compared to the noise-free agents' performance. Still, all agents seem to learn very similar policies despite being trained with different settings of σ, as we can see by their almost identical performance at evaluation time shown in Figure 7.10 (b). Despite a drop in performance during training, the final performance of the models on a test set of previously unseen TSP instances stays almost unaffected by the noise present during training. While we see that agents trained with more noise seem to learn more noise-robust policies as in the case of the CartPole environment, this effect is not as pronounced here. Additionally,

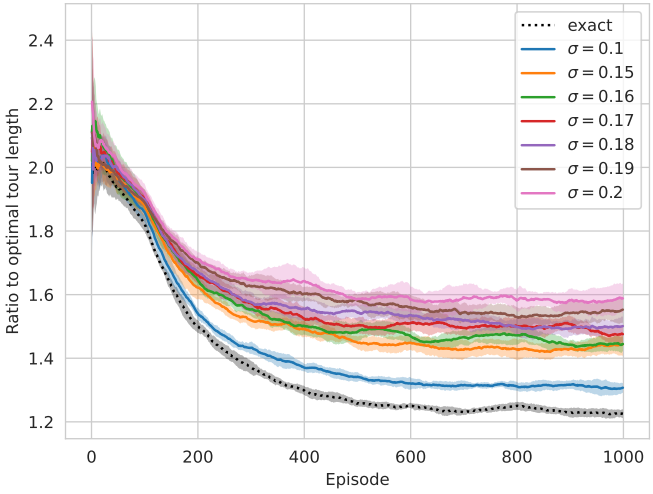


(a) training performance

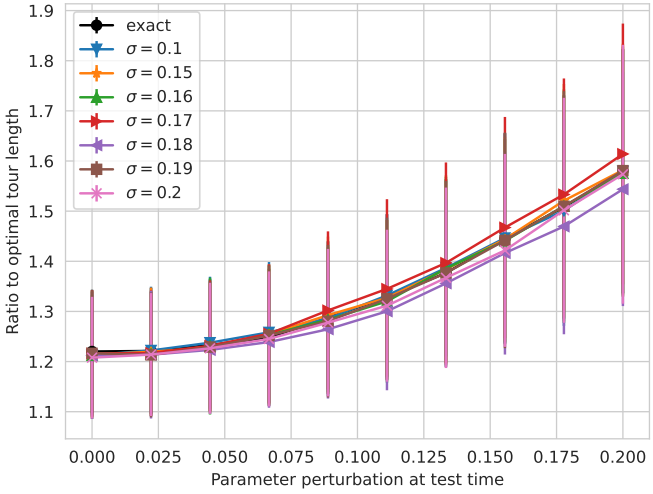


(b) evaluation performance

Figure 7.10: Training and evaluation of Q-learning agents in the TSP environment under various perturbations σ. Panel (a) shows the effect of perturbations during training, panel (b) shows results for the same agents evaluated on varying perturbation levels after training, different to those present at training time.



(a) training performance



(b) evaluation performance

Figure 7.11: Training and evaluation of policy gradient agents in the TSP environment under various perturbations σ. Panel (a) shows the effect of perturbations during training, panel (b) shows results for the same agents evaluated on varying perturbation levels after training, different to those present at training time.

we again see that performance of trained models in Figure 7.10 (b) starts to drop at $\sigma > 0.1$, as indicated by our theoretical analysis in Section 7.3.1. While the policy gradient method shows a certain robustness to noise during training in the CartPole environment, this is not the case for the TSP environment, as we show in Figure 7.11 (a). The only agent that gets close in performance to the noise free agent is the one trained with $\sigma = 0.1$, while higher perturbations yield agents that are relatively bad with an approximation ratio between 1.4 and 1.6 on average. However, again, all agents seem to learn similar policies as indicated by their test performance in Figure 7.11 (b). Similar to CartPole, the agents' performance on the test set under varying perturbation levels closely matches that of the noise-free agents, and again we see a large drop in performance for perturbations that are higher than $\sigma = 0.1$.

Overall, the Q-learning algorithm performs better in the TSP environment than the policy gradient method. The optimal tour for each TSP instance is deterministic, so using a stochastic policy as in the policy gradient approach introduces an additional source of error, as there is always a non-zero probability to chose a non-optimal action. This leads to an increased susceptibility to the Gaussian perturbations present during the evaluation of the policy gradient algorithm. This is not the case for Q-learning, where choices are made based on the argmax Q-value. Additionally, the ansatz that we use does not separate between data encoding and trainable parameters as described in Section 7.1. As the optimal tour of a TSP instance does not change upon small perturbations of the edge weights, this leads to a relative robustness of this ansatz used in conjunction with Q-learning to Gaussian coherent noise in this environment.

7.4 Incoherent noise

The Gaussian perturbation noise that we studied in Section 7.3 is well-suited to model coherent errors due to imprecision in the control of the quantum device, but it does not reflect noise that results from undesired interactions of the quantum system with its environment. To study the effect of this type of incoherent noise we perform additional experiments in this section.

We simulate this type of noise with TensorFlow Quantum (TFQ) [224], where they are implemented through a Monte-Carlo trajectory sampling method [291, 292] that approximates the effect of noise by averaging over state vectors generated from

a probabilistic application of the noise channel. This method of simulating noise essentially trades off the overhead in memory needed to store the $2^n \times 2^n$ sized density matrices necessary to simulate incoherent noise, with a runtime overhead. The precision of this approximation is determined by the number of repetitions, which specifies how many "noisy" state vectors are used. This adds a stochastic element to the simulation of the noise channels, and we get closer to simulating the exact noise model as the number of trajectories increases. Depending on the environments, we choose the number of trajectories so that it is possible to perform simulations in a reasonable time frame, and specify this number individually for each of the experiments below. We note that the runtime requirements for CartPole when simulating this type of noise are especially high, as the number of time steps in each episode, as well as the number of episodes itself depends strongly on the performance of the agent. In particular, agents that perform neither very well nor very poorly, which are exactly the noise configurations we are interested in studying here, take especially long to simulate, as they do not converge early by solving the environment, but still take on the order of 100 time steps in each episode. Therefore we focus our attention mainly on the TSP environment in this section.

7.4.1 Depolarizing noise

Depolarizing noise affects a quantum state by either replacing it with the completely mixed state with probability p, or leaving it untouched otherwise [293]. Let ρ be the density matrix of a qubit, then depolarizing noise is defined by the map

$$\mathcal{D}_p(\rho) = (1-p)\rho + p\frac{\mathbb{1}}{2}. \tag{7.19}$$

We model depolarization noise with Cirq [291] and TFQ [224] by appending a layer of local depolarizing channels to every qubit after each time step of the computation, where a time step is defined as the largest set of gates that can be implemented simultaneously. This implementation takes into account the possibility of cross-talk between qubits [294]. Also, note that while the use of depolarizing channels alone may not be a good approximation of real single qubits errors, it may become a good effective description of the overall noise process for the case where many qubits and layers are used [295].

In our simulations, we assume that both single- and two-qubits gates are noisy, and consist of a composition of the ideal gates followed by *local* depolarizing channels

of equal probability p, acting independently on each qubit. In particular, the application of a depolarizing noise channel is implemented by performing one out of four actions at each circuit execution (trajectory): do nothing with probability $1-p$, or apply at random one of the three Pauli operators with probability p, and then average over the results. We remark that the average gate error of single-qubit gates in currently available superconducting quantum computing hardware is of the order of $r \lessapprox 0.01$, with gate fidelities exceeding $> 99\%$. Finally, we note that one can relate the depolarisation strength p to the average gate error r over single qubit Cliffords, as measured by Randomized Benchmarking (RB) [296, 294, 296] and commonly reported for quantum devices [297, 298], via $r = p/2$. However, our circuits do not only use Cliffords, and moreover, estimates for the gate error in RB depend on the basis gates available on the device. Therefore, one should consider our simulations with depolarizing noise of strength p as a proxy for a quantum device whose average error rate r is of the same order of magnitude of p. While a single-qubit error noise model may not be accurate enough to closely mimic the behaviour of a real quantum device, it gives us the possibility to study the effect of single-qubit errors separately, before we go on to study a noise model that also includes two-qubit gate errors in section 7.4.2.

As mentioned above, simulating incoherent noise has high runtime requirements, so in the following we limit our studies to: (i) Q-learning in the CartPole environment, and (ii) the policy gradient method in the TSP environment. We pick these settings as they were the ones that were more sensitive to Gaussian coherent noise in our studies in Section 7.3, and in that sense represent the worst case instances from the previous section. To simulate the noisy quantum circuits, we use the Monte Carlo sampling as described above, where the number of trajectories used depends on the environment. As the CartPole environment requires a very high number of environment interactions (the better the agent, the more circuit evaluations are required per episode), we use 100 trajectories in this setting. In the TSP environment, the number of steps in each episode is constant and therefore we can use a higher number of 1000 trajectories and still perform simulations in a timely manner.

Figure 7.12 shows results of Q-learning agents trained in the CartPole environment with various error probabilities p. Agents with a realistic error probability of up to $p = 0.01$ still solve the environment in less than 2000 episodes on average. Agents trained with error probability $p = 0.005$ reach higher scores almost as quickly as

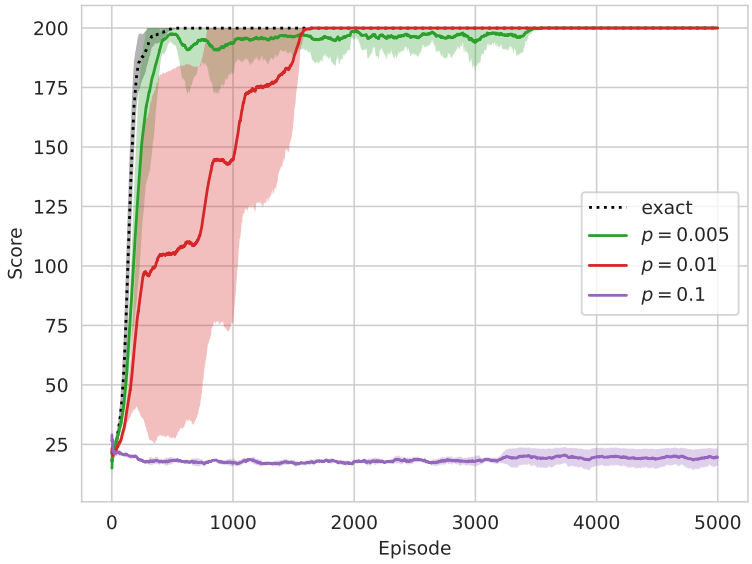


Figure 7.12: Q-learning agents trained with varying probabilities p of depolarization errors, and five layers of the circuit depicted in Figure 7.2 a). Noise is simulated with 100 Monte Carlo trajectories. The noisy curves are averaged over 5 agents, the exact one is averaged over 10 agents as in previous figures.

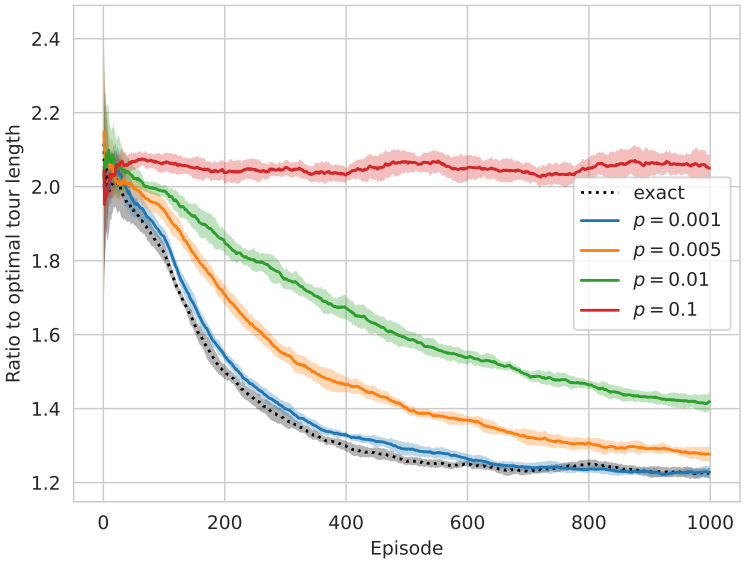


Figure 7.13: Policy gradient agents trained in the TSP environment with varying probabilities p of depolarization error, with one layer of the circuit depicted in Figure 7.2 c). Noise is simulated with 1000 Monte Carlo trajectories. All curves are averaged over 10 agents.

agents trained in the noise-free setting, but stay somewhat unstable until they solve the environment after 3500 episodes on average. When the noise probability is increased to $p = 0.1$, we see that agents fail to make any learning progress at all.

Figure 7.13 shows the performance of the policy gradient method under one-qubit depolarization errors in the TSP environment. In this setting, agents trained with error probability $p = 0.01$, as is a realistic assumption on current devices, perform noticeably worse than agents in the noise-free setting with a drop in approximation ration of around 0.2 on average. Only when we consider an error probability of $p = 0.001$ do we get performance that is almost exactly the same as that in the noise-free case. Similar to the results of the Q-learning agent in the CartPole environment, agents trained with an error probability of $p = 0.1$ show no meaningful learning progress.

7.4.2 Noise model based on current hardware

After studying the effect of single-qubit depolarization errors in Section 7.4.1, we now study the performance of the Q-learning algorithm in the TSP environment in the presence of a more realistic noise model that captures the behaviour of a near-term superconductive quantum device. The error sources we incorporate into this noise model are the following: single-qubit and two-qubit depolarization errors, single qubit amplitude damping error, and measurement noise. While hardware providers like IBM and Google offer the possibility of simulating noise models of specific devices, we do not want to take device-specific factors like qubit topology and native gate sets into account in this work, as the performance in these settings also depends strongly on the quality of the circuit compiled to the native gate set and qubit connectivity [299]. Instead, we define a custom noise model based on gate fidelities published by hardware vendors, but do not take the above details into account. To determine realistic settings for the error probability of each noise source, we use calibration data published by IBM [300] at the time of writing. The noise model used in our simulation is specified as follows:

- **Depolarization error:** Single qubit depolarization channels with $p = 0.001$ are applied after every single qubit gate. Two-qubit depolarization errors, defined by properly adjusting the definition in Equation (7.19), with $p_2 = 0.01$ are applied after every two-qubit gate on the corresponding pair of qubits.

Error source	**a)**	b)	c)	d)
Depolarization (1Q)	**0.001**	0.001	0.01	0.1
Depolarization (2Q)	**0.01**	0.01	0.1	0.2
Amplitude damping	**0.0003**	0.03	0.03	0.1
Bitflip (measurement)	**0.01**	0.01	0.1	0.1

Table 7.1: Error strengths for the configurations of the custom noise model used in Figure 7.14. Depolarization (1Q) indicates the single qubit depolarising channel applied after each single-qubit gate, and similarly for 2Q for two-qubit gates. Configuration a) in bold is based on error rates published by IBM at the time of writing, as described in the main text.

- **Amplitude damping error:** Amplitude damping channels with decay parameter $\gamma = 0.003$ are applied after each single- and two-qubit gate on the corresponding qubits. Such a decay rate is valid for real devices having single qubit gate durations of $t = 35$ns, and average qubit decay times $T_1 \approx 100\mu$s, which correspond to a decay parameter of $\gamma = 1 - \exp(-t/T1) \approx 0.0003$.
- **Measurement noise** Measurement errors are modeled by appending a bit-flip channel with probability $p = 0.01$ to every qubit right before the measurement process.

We recall that the circuit ansatz for the TSP environment is the one depicted in Figure 7.2(c), where input information about the edge weights of the TSP instance is encoded by means of two-qubit gates. We therefore chose to study this ansatz in the context of a noise model that incorporates two-qubit errors, as we expect that these types of errors will affect performance of an ansatz that encodes crucial information in two-qubit gates more severely. Additionally, it is hard to perform simulations in this setting for the CartPole environment in a reasonable amount of time, as discussed above. For these reasons, we restrict our attention to the TSP environment in this section.

Figure 7.14 shows results averaged over five Q-learning agents in the TSP environment for each of the error probability configurations of the custom noise model described above. We show the specific error probabilities used for the simulations in Table 7.1. Configuration a) corresponds to error probabilities that are consistent with those present on current quantum hardware as described above. Based on this, we specify three other error probabilities b) - d) by increasing the error on varying error sources. We note that while the error probabilities themselves in

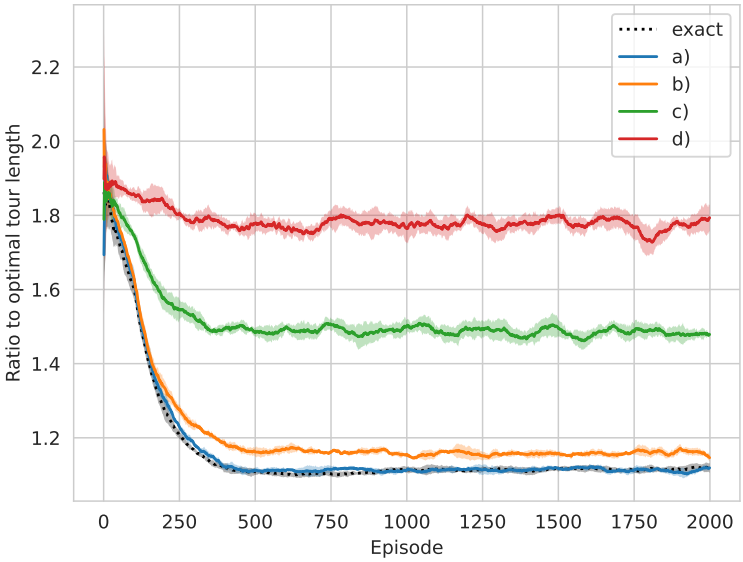


Figure 7.14: Q-learning agents trained in the TSP environment with one layer of the circuit depicted in Figure 7.2 c) and custom noise model, using 1000 Monte Carlo trajectories. The labels indicate the custom noise configurations defined in Table 7.1, results are averaged over five agents in each curve, except for the exact curve which is averaged over ten agents as done in previous figures.

configuration a) are consistent with those on current hardware, our simulation is only an approximation of this error due to the Monte Carlo trajectory sampling method described in Section 7.4. To perform simulations in a reasonable time frame, we use 1000 trajectories for each circuit evaluation. The circuit that we simulate has 145 gates (counting a ZZ-gate as two CNOTs and one Z gate), and for small error probabilities the chance of applying each of the noise channels is relatively small. This means that in each trajectory, a relatively small number of noise channels is applied. Hence we expect that the results in Figure 7.14 are slightly better than what we would get if the exact noise model was simulated (i.e., in the limit of a large number of trajectories, or by considering the full density matrix).

Looking at the results in Figure 7.14, we see that for configuration a) (blue), the performance of the agents matches those of the noise-free ones (dotted black)

almost exactly, and the noise model based on realistic error strengths of current devices does not affect training. We see a slight drop in performance when we increase the error probability of the amplitude damping channels from 0.0003 to 0.03 (orange), as described in Table 7.1, column b). For configuration c), we also increase the other remaining error sources' probabilities, which leads to a considerable drop in performance. In configuration d), we assume extremely high error probabilities for each of the noise channels, which leads to a complete failure of the agents to make any meaningful learning progress in this environment.

7.5 Conclusions

Our goal in this chapter was to evaluate the resilience of variational RL algorithms to various types of noise that are present on real quantum hardware. First, we investigated shot noise, which results from the probabilistic nature of quantum measurements. We introduced a method to reduce the number of shots to train a Q-learning agent, motivated by the specific structure of the underlying RL algorithm. Our shot allocation technique enables a more shot-frugal training of variational Q-learning models with little or no effect on the final performance of the agents.

After considering shot noise, we moved on to study the effect of Gaussian coherent errors that can arise on real hardware due to miscalibration of the device, or imprecise pulse sequences that implement the parameterised gates in the quantum circuit. We gave an analytic expression for how this type of noise affects the output of a quantum RL agent, and provided a bound on the standard deviation of the Gaussian error that elucidates the tolerable magnitude of the error on the output of a quantum model. We confirm this bound in our simulations, where we study the effect of various levels of Gaussian perturbations on the performance of training policy gradient and Q-learning agents in two different environments. For one of these environments, we find that agents trained with higher noise probabilities also learn more robust policies and Q-functions, in the sense that under evaluation of different perturbation levels, these agents achieve optimal or close to optimal performance more often.

Finally, we studied incoherent noise that emerges in real hardware due to undesired interactions of the qubits with the surrounding environment, as the device is not completely shielded from external effects. To this end, we consider single-qubit

depolarization errors, as well as a custom noise model that combines single- and two qubit depolarization errors, amplitude damping errors, and bitflip (measurement) errors. For the latter, we perform simulations with realistic error probabilities for each of the noise channels, in line with data published for IBM devices at the time of writing.

Overall, we find that the effect of noise on training variational RL algorithms for Q-learning and the policy gradient method depends strongly on the strength of the noise, as well as the type of noise itself. For some cases, like decoherence errors with realistic error probabilities of current devices, the drop in performance is relatively small. On the other hand, we find that large Gaussian perturbations as well as errors induced by the probabilistic nature of quantum measurements can affect performance in highly detrimental ways. Additionally, we find that for Gaussian coherent noise agents that are trained with higher perturbations learn more noise-robust policies in some cases, similar to results in classical literature, where noise is used as a regularization technique.

While our results were performed in a regime that is still efficiently simulable on classical computers, it is an interesting question for future work to consider the implications of noise-robustness of large-scale quantum models in light of recent results which show that in certain settings, the outputs of noisy quantum circuits can be efficiently approximated classically [50, 301]. This raises the question to what extent an inherent noise-robustness of hybrid variational quantum machine learning affects the possibility to achieve a quantum advantage with these types of models.

On the practical side, the optimization procedures that we used in this work were the same as those commonly used to train models in noise-free simulations and are not tailored to account for quantum hardware specific noise. This raises the question on how optimization methods that are tailored for the special characteristics of variational quantum models could further improve the performance of these types of models in a noisy setting. For the optimization of PQC parameters in the combinatorial optimization or quantum chemistry setting, it is known that some optmization methods, like simultaneous perturbation stochastic approximation (SPSA), actually become better with noise. It is an interesting area of future research to design quantum-specific optimization routines for machine learning that address or even combat specific types of noise, for example leveraging effective quantum error mitigation techniques [302, 303, 304]. This work motivates the study

of these types of optimization methods, as well as continued efforts to find learning tasks where variational RL algorithms can potentially provide an advantage.

CHAPTER 8

Conclusion

Throughout this thesis, we have investigated how to design and train variational quantum machine learning models, with a focus on reinforcement learning. We identified four main areas that influence the performance of VQAs in Chapter 1: i) the way in which data is fed in to and read out from the circuit, ii) the structure of the ansatz of the PQC that is used, iii) the classical optimization method used to find the parameters of the circuit, and iv) the influence of noise present on quantum hardware on the training of variational models. We dedicated a chapter to each of these areas.

In Chapter 4, we studied how the classical optimization routine used in a VQA can help to mitigate the barren plateau phenomenon for certain types of circuits. For this, we introduce a method to build the ansatz in an iterative fashion, and partition the set of parameters that are trained in the circuit in a way that prevents the onset of barren plateaus during the optimization routine. We compared this approach to the standard technique of training a fixed ansatz and updating all parameters in every step, and found that our algorithm achieves a lower error on the test set, as well as requiring less wall-clock time considering a realistic sampling rate of a current device.

We address the question of how the data-encoding technique, as well as the choice of observables to read out actions from a quantum Q-learning agent where the PQC is used as a Q-function approximator, influence its performance on two benchmark environments from classical RL literature in Chapter 5. In addition to this, we establish a theoretical separation between classical and quantum learners in a Q-learning setting, and perform an in-depth empirical comparison between quantum Q-learning and the classical DQN algorithm, where we find that the

quantum model achieves the same performance as its classical counterpart with a fraction of the parameters.

After this, we address the question of how to design problem-tailored ansatzes for a certain input data type, namely weighted graphs, in Chapter 6. In this chapter, we introduce an ansatz that is equivariant under permutations of the nodes of the input graphs, meaning that the outcome of the PQC does not depend on the order in which representations of nodes are fed into the circuit. We establish a connection to the field of classical geometric deep learning, that is concerned with the design of efficient NN architectures that preserve certain symmetries. We study our ansatz in the context of a learning task on graphs, where a QML model is trained to solve instances of a combinatorial optimization problem by using reinforcement learning. First, we theoretically study the expressivity of our model and find that for our ansatz at depth one, there exists a setting of parameters, for arbitrarily sized instances of the optimization problem that we study, so that our model produces the optimal solution. After establishing that our model is theoretically expressive enough to solve instances of the given optimization problem, we numerically compare our ansatz to general hardware-efficient ansatzes that are unrelated to the problem structure, and find that our equivariant ansatz outperforms them by a large margin.

Finally, we turn to the question of how noise that is present on quantum hardware influences the performance of variational RL models for policy gradients and Q-learning. This study is motivated by a common folklore in the QML community, that conjectures that variational QML models are robust to hardware noise to some degree, due to the classical parameter optimization scheme. In addition, certain results in the classical literature of training neural networks hint at the possibility that a small amount of noise can even be used as a method to combat overfitting. We analytically and numerically study the effect of various noise sources present on quantum hardware: shot noise that is based on the probabilistic nature of quantum measurements, coherent errors due to imperfect control of the quantum device, and incoherent errors that occur due to the device's interaction with its environment. We find that there indeed exists a regime where noise does not prevent quantum RL agents from successfully performing in a given environment, and that there exist cases when training under noisy conditions leads to more robust policies. Additionally, we provide an algorithm to flexibly determine the number of shots required for estimating Q-values, such that the overall number of shots is reduced

compared to setting a fixed number of measurements for each circuit evaluation when training a Q-learning agent.

As alluded to in Chapter 1, the goal of this thesis is to contribute to the knowledge of how to successfully train variational QML models, with the aim to foster empirical studies of areas where these types of models can eventually become useful in the future. This is motivated by theoretical results that show that there, indeed, are functions that can only be efficiently learned in a quantum setting, but which are somewhat contrived and not applicable to real-world problems, as well as the historical development of heuristics and machine learning, which showed that often large progress is made when improved hardware becomes available. We hope that, similarly to the development of classical deep learning, the availability of large-scale quantum hardware will lead to the discovery of interesting and valuable applications of quantum machine learning.

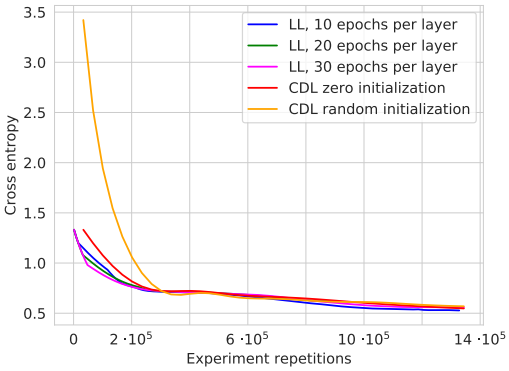
> *"We are only one creative algorithm away from valuable near-term applications."*
>
> – Arute et al. [2]

Appendix

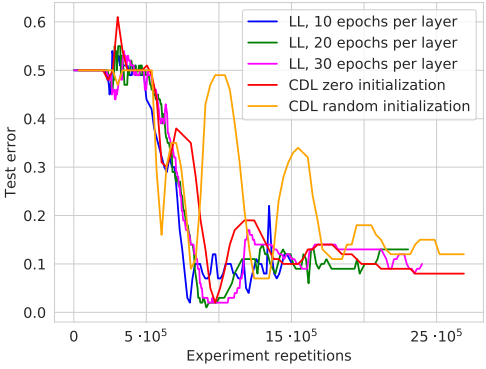
Layerwise learning for quantum neural networks

As alluded to in Section 4.1, LL and CDL perform similarly in a perfect simulation scenario, where we assume neither shot nor hardware noise. Figure 1 a) shows a comparison of LL and CDL under perfect conditions, i.e. infinite number of measurements and a batch size that corresponds to the number of samples, which enables computation of exact gradients. Here, the magnitude of gradients doesn't affect the learning process severely, as the Adam optimizer uses adaptive learning rates for each parameter and can therefore handle different ranges of gradient magnitudes well as long as there is some variance in the computed gradients. In this regime, both approaches show similar performance.

The convergence rate of a PQC increases proportionally to the number of parameters in a model [188], so the number of experiment repetitions is almost equal for LL and CDL. LL has less parameters and needs more epochs to converge due to this, whereas CDL needs more calls to the quantum device for one update step, but in turn needs less epochs to converge. In terms of cross entropy, both LL and CDL converge to a value of roughly 0.51. The corresponding test error of all approaches, except for the randomly initialized CDL, reaches almost 0 but doesn't converge there and settles around an error of roughly 0.1 eventually, as seen in Figure 1 b).



(a) cross entropy



(b) test error

Figure 1: a) Cross entropy of LL and CDL during training with exact gradient calculation corresponding to infinite number of measurements. When one assumes the unphysical situation of infinite measurements ($m = \infty$) all methods seem to perform similarly. In particular, we compare LL to CDL with zero and random initialization, where the initial parameters for the latter are chosen uniformly from $[0, 2\pi)$. The hyperparameters for all configurations were set to $m = \infty$, $b = 100$ and $\eta = 0.01$. (For computing the number of experiment repetitions as defined in Section 4.2.3, we drop m.) b) Test error corresponding to the runs shown in Figure 1. This further supports the observation that when one allows unphysical, arbitrary precision queries($m = \infty$), all tuned training strategies seem to perform similarly.

Quantum agents in the Gym: A variational quantum algorithm for deep Q-learning

Visualization of a learned Q-function

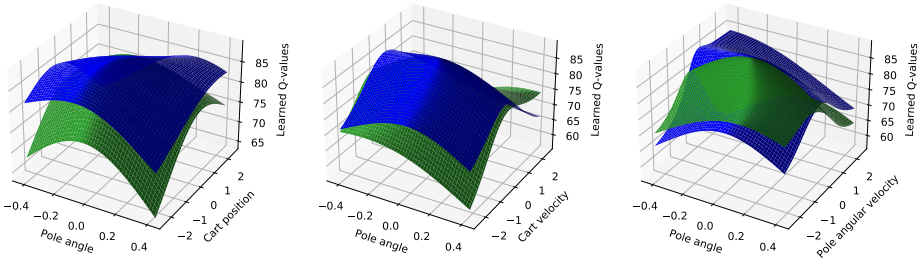


Figure 2: Visualization of the approximate Q-function learned by a quantum Q-learning agent solving Cart Pole. Due to the 4 dimensions of the state space in Cart Pole, we represent the Q-values associated to the actions "left" (green) and "right" (blue) on 3 subspaces of the state space by fixing unrepresented dimensions to 0 in each plot. As opposed to the analogue values (i.e., unnormalized policy) learned by policy-gradient PQC agents in this environment [150], the approximate Q-values appear nicely-behaved, likely due to the stronger constraints that Q-learning has on well-performing function approximations.

Model hyperparameters

In the following, we give a detailed list of the hyperparameters for each configuration in fig. 5.3, fig. 5.4, fig. 5.5, fig. 5.6 and fig. 5.7. The hyperparameters that we searched over for each model were the following (see explanations of each hyperparameter in table 1):

- *Frozen Lake v0*: update model, update target model, η
- *Cart Pole v0, quantum model*: batch size, update model, update target model, η, train w_d, train w_o, η_{w_d}, η_{w_o}
- *Cart Pole v0, classical model*: number of units per layer, batch size, update model, update target model, η

	Hyperparameter explanation
qubits	number of qubits in circuit
layers	number of layers
γ	discount factor for Q-learning
train w_d	train weights on the model input as defined in section 5.1.1
train w_o	train weights on the model output as defined in section 5.1.2
η	model parameter learning rate
η_{w_d}	input weight learning rate
η_{w_o}	output weight learning rate
batch size	number of samples shown to optimizer at each update
ϵ_{init}	initial value for ϵ-greedy policy
ϵ_{dec}	decay of ϵ for ϵ-greedy policy
ϵ_{min}	minimal value of ϵ for ϵ-greedy policy
update model	time steps after which model is updated
update target model	time steps after which model parameters are copied to target model
size of replay memory	size of memory for experience replay
data re-uploading	use data re-uploading as defined in section 5.1.1

Table 1: Description of hyperparameters considered in this work

	Frozen Lake v0, fig. 5.3	**Cart Pole v0, optimal**	**Cart Pole v0, sub-optimal**
qubits	4	4	4
layers	5, 10, 15	5	5
γ	0.8	0.99	0.99
train w_d	no	yes, no	yes, no
train w_o	no	yes, no	yes, no
η	0.001	0.001	0.001
η_{w_d}	–	0.001	0.001
η_{w_o}	–	0.1	0.1
batch size	11	16	16
ϵ_{init}	1	1	1
ϵ_{dec}	0.99	0.99	0.99
ϵ_{min}	0.01	0.01	0.01
update model	5	1	10
update target model	10	1	30
size of replay memory	10000	10000	10000
data re-uploading	no	yes, no	yes, no

Table 2: Hyperparameter settings of PQCs in fig. 5.3, fig. 5.4 and fig. 5.5

layers	5	10	15	20	25	30
qubits	4	4	4	4	4	4
γ	0.99	0.99	0.99	0.99	0.99	0.99
train w_d	yes	yes	yes	yes	yes	yes
train w_o	yes	yes	yes	yes	yes	yes
η	0.001	0.001	0.001	0.001	0.001	0.001
η_{w_d}	0.001	0.001	0.001	0.001	0.001	0.001
η_{w_o}	0.1	0.1	0.1	0.1	0.1	0.1
batch size	16	64	32	16	64	16
ϵ_{init}	1	1	1	1	1	1
ϵ_{dec}	0.99	0.99	0.99	0.99	0.99	0.99
ϵ_{min}	0.01	0.01	0.01	0.01	0.01	0.01
update model	1	10	10	10	10	10
update target model	1	30	30	30	30	30
size of replay memory	10000	10000	10000	10000	10000	10000
data re-uploading	yes	yes	yes	yes	yes	yes

Table 3: Hyperparameter settings of PQCs in fig. 5.6 a)

units in hidden layers	(10, 10)	(15, 15)	(20, 20)	(24, 24)	(30, 30)	(64, 64)
γ	0.99	0.99	0.99	0.99	0.99	0.99
η	0.001	0.001	0.001	0.001	0.001	0.001
batch size	64	16	64	64	64	16
ϵ_{init}	1	1	1	1	1	1
ϵ_{dec}	0.99	0.99	0.99	0.99	0.99	0.99
ϵ_{min}	0.01	0.01	0.01	0.01	0.01	0.01
update model	1	1	1	1	1	1
update target model	1	1	1	1	1	1
size of replay memory	10000	10000	10000	10000	10000	10000

Table 4: Hyperparameter settings of NNs in fig. 5.6 b)

Equivariant quantum circuits for learning on weighted graphs

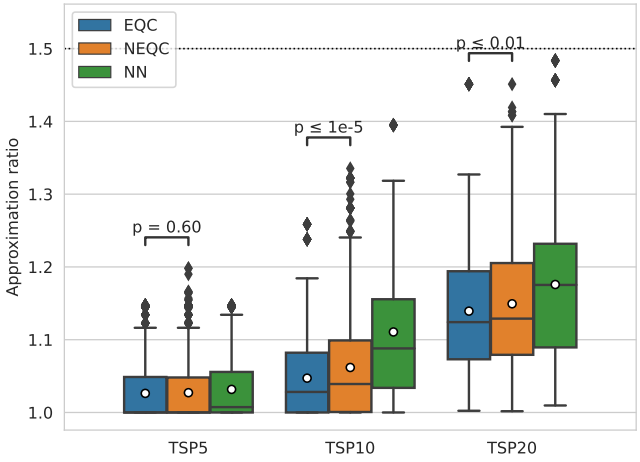
Additional results on statistical significance of comparison between EQC and NEQC

To make statements on the statistical significance of the difference between the performance of the EQC and NEQC shown in Figure 6.5, we perform a two-sample t-test on the two models for the same instance sizes (i.e., for the data in the two boxes for each instance size) with the null hypothesis that the averages of the two distributions are the same. Based on this, we compute p-values to quantify the statistical significance of the differences between models.

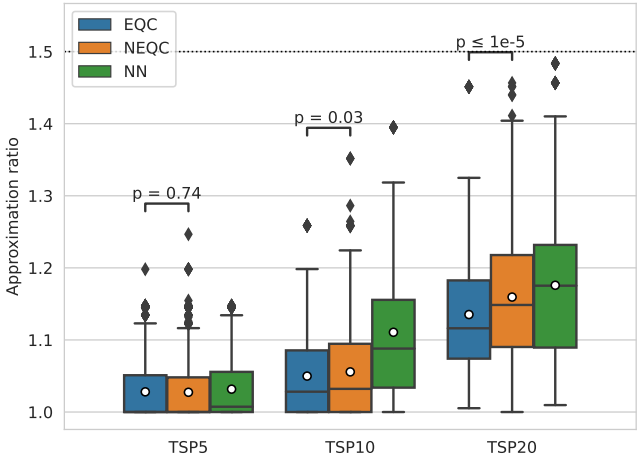
Figure 3 a) shows p-values for the depth-one EQCs and NEQCs from Figure 6.5 b). For the 5-city instances, we can not reject the null hypothesis. Indeed, it is already visible by looking at the boxes that the distributions are very similar, which can be expected as the number of permutations of a graph with five vertices is small. However, as we scale up the instance size to ten cities, the corresponding p-value is much smaller than 0.05, which means that we can reject the null hypothesis that the two distributions have the same average with high confidence. This is also the case for the instances with twenty cities, where the p-value is less than 0.01.

Figure 3 b) shows p-values for the depth four EQCs and NEQCs from Figure 6.5 d). Again, the p-value of the 5-city instances is very high with 0.74, so that we can not reject the null hypothesis. Also similarly to the above, the p-values get smaller as we scale up the instance size. For the depth-four ansatzes, the p-value is smallest for the twenty city instances, with a value much smaller than 0.05.

To provide additional insight, we also plot the means and their standard error for both the 1-layer (EQC-1, NEQC-1) and 4-layer (EQC-4, NEQC-4) models in Figure 6.5. As a rule of thumb, one can expect that when the error bars given by the standard errors of two means do not overlap, the p-value can be smaller than 0.05, while in the case that they do overlap, the p-value is likely much larger. The error bars in Figure 4 are in line with this statement, where we see that the error bars for the five-city instances overlap for both circuit depths, while this is not the case for the larger instance sizes and in addition the distance between the means increases for those instance sizes. Remarkably, we also see that the difference



(a) one layer



(b) four layers

Figure 3: P-values for comparison of EQCs and NEQCs at depth one and four from Figure 6.5 b) and d).

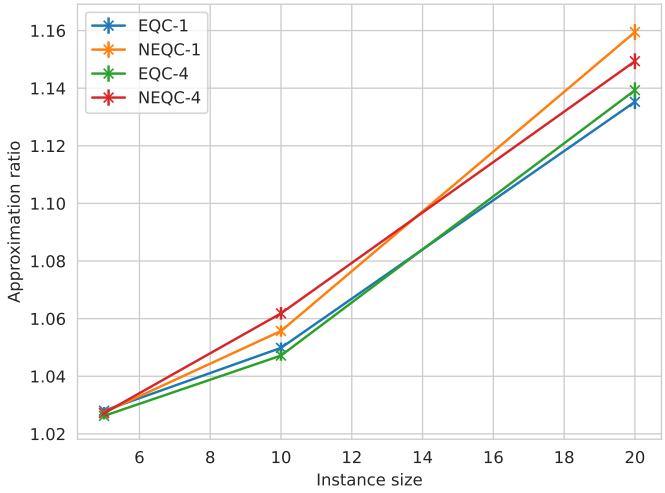


Figure 4: Mean and standard error of the mean for the one- and four-layer EQCs and NEQCs in Figure 6.5 b), d).

between the EQC at depths one and four is very small, and that increasing the circuit depth does not provide much benefit on this learning task.

Robustness of quantum reinforcement learning under hardware errors

Gaussian Noise Analysis

In this Appendix we perform the noise analysis of a scalar function whose parameters are corrupted by independently distributed Gaussian perturbations. Let $f : \mathbb{R}^M \to \mathbb{R}$ be the function under investigation, whose parameters $\boldsymbol{\theta} = (\theta_1, \dots, \theta_M) \in \mathbb{R}^M$ are corrupted by a Gaussian noise $\theta_i \to \theta_i + \delta\theta_i$ with zero mean and variance σ^2, i.e.

$$\begin{aligned} &\delta\theta_i \sim \mathcal{N}(0, \sigma^2) \quad \forall i = 1, \dots, M\,, \\ &\mathbb{E}[\delta\theta_i] = 0\,, \\ &\mathbb{E}[\delta\theta_i \delta\theta_j] = \sigma^2 \delta_{ij}\,. \end{aligned} \tag{1}$$

Since the perturbations are independently distributed and Gaussian, all higher order moments can be evaluated starting from two-point correlators of the form $\mathbb{E}[\delta\theta_i \delta\theta_j]$, as dictated by *Wick's formulas* for multivariate normal distributions [305]

$$\begin{aligned} &\mathbb{E}[\delta\theta_{i_1} \cdots \delta\theta_{i_{2n+1}}] = 0\,, \\ &\mathbb{E}[\delta\theta_{i_1} \cdots \delta\theta_{i_{2n}}] = \sum_{\mathcal{P}} \mathbb{E}[\delta\theta_{k_1} \delta\theta_{k_2}] \cdots \mathbb{E}[\delta\theta_{k_{2n-1}} \delta\theta_{k_{2n}}]\,, \end{aligned} \tag{2}$$

where with $\mathcal{P}$ we denote all the possible distinct $(2n-1)!!$ pairings of the n variables, as these can be used to express all higher order even moments in terms of products of second moments. Note that all the terms involving an odd number of perturbations $\delta\theta_i$ vanish, and only even moments remain. For example, expression (2) for the fourth-order moment ($n = 4$) amounts to

$$\begin{aligned} \mathbb{E}[\delta\theta_i \delta\theta_j \delta\theta_k \delta\theta_m] &= \mathbb{E}[\delta\theta_i \delta\theta_j]\, \mathbb{E}[\delta\theta_k \delta\theta_m] + \mathbb{E}[\delta\theta_i \delta\theta_k]\, \mathbb{E}[\delta\theta_j \delta\theta_m] + \mathbb{E}[\delta\theta_i \delta\theta_m]\, \mathbb{E}[\delta\theta_j \delta\theta_k] \\ &= \sigma^4 \big(\delta_{ij}\delta_{km} + \delta_{ik}\delta_{jm} + \delta_{im}\delta_{jk}\big)\,. \end{aligned} \tag{3}$$

We now proceed considering the multi dimensional Taylor expansion of the function

$f(\boldsymbol{\theta}+\delta\boldsymbol{\theta})$ around the noise-free point. Up to arbitrary order, this reads

$$f(\boldsymbol{\theta}+\delta\boldsymbol{\theta}) = f(\boldsymbol{\theta}) + \sum_{i=1}^{M} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_i}\delta\theta_i + \frac{1}{2!}\sum_{i,j=1}^{M} \frac{\partial^2 f(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j}\delta\theta_i\delta\theta_j + \frac{1}{3!}\sum_{i,j,k=1}^{M} \frac{\partial^3 f(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j\partial\theta_k}\delta\theta_i\delta\theta_j\delta\theta_k + \dots . \quad (4)$$

where we used the equal sign because we are considering the full Taylor series, and we assume that this converges to the true function (this statement can be made precise by showing that the remainder term of the expansion goes to zero as the order of expansion goes to infinity).

Before proceeding, we simplify the notation to make the calculation of the Taylor expansion easier to follow. First, we denote the partial derivatives with respect to parameter θ_i as $\partial_i := \partial/\partial\theta_i$, and similarly for higher order derivatives, for example $\partial_{ij} = \partial^2/\partial\theta_i\partial\theta_j$. Also, we suppress the explicit dependence of the function on $\boldsymbol{\theta}$, using the short-hand f instead of $f(\boldsymbol{\theta})$. At last, we make use of Einstein' summation notation where repeated indexes imply summation.

With this setup, using Eqs. (1), (2) and (3) in (4), one can evaluate the expectation value of the function over the perturbations' distributions as

$$\begin{aligned}
\mathbb{E}[f(\boldsymbol{\theta}+\delta\boldsymbol{\theta})] &= f(\boldsymbol{\theta}) + \partial_i f\,\mathbb{E}[\delta\theta_i] + \frac{1}{2}\partial_{ij} f\,\mathbb{E}[\delta\theta_i\delta\theta_j] + \frac{1}{3!}\partial_{ijk} f\,\mathbb{E}[\delta\theta_i\delta\theta_j\delta\theta_k] \\
&\quad + \frac{1}{4!}\partial_{ijkm} f\,\mathbb{E}[\delta\theta_i\delta\theta_j\delta\theta_k\delta\theta_m] + \dots \\
&= f(\boldsymbol{\theta}) + \frac{\sigma^2}{2}\partial_{ij} f\,\delta_{ij} + \frac{\sigma^4}{4!}\partial_{ijkm} f\,(\delta_{ij}\delta_{km} + \delta_{ik}\delta_{jm} + \delta_{im}\delta_{jk}) + \dots \\
&= f(\boldsymbol{\theta}) + \frac{\sigma^2}{2}\sum_i \frac{\partial^2 f}{\partial\theta_i^2} + \frac{\sigma^4}{4!}3\sum_{ij}\frac{\partial^4 f}{\partial\theta_i^2\partial\theta_j^2} + \dots
\end{aligned} \quad (5)$$

where in the last line we simplified the fourth order term as

$$\begin{aligned}
\mathbb{E}[f^{(4)}] &= \frac{\sigma^4}{4!}\partial_{ijkm} f\left(\delta_{ij}\delta_{km} + \delta_{ik}\delta_{jm} + \delta_{im}\delta_{jk}\right) \\
&= \frac{\sigma^4}{4!}\left(\sum_{ik}\frac{\partial^4 f}{\partial\theta_i^2\partial\theta_k^2} + \sum_{ij}\frac{\partial^4 f}{\partial\theta_i^2\partial\theta_j^2} + \sum_{im}\frac{\partial^4 f}{\partial\theta_i^2\partial\theta_m^2}\right) \\
&= \frac{\sigma^4}{4!}\,3\sum_{ij}\frac{\partial^4 f}{\partial\theta_i^2\partial\theta_j^2}\,.
\end{aligned}$$

Since the expectation values involving an odd number of perturbations vanish, only the even order terms survive, and these can be expressed as

$$\mathbb{E}[f^{(2n)}] = \frac{\sigma^{2n}}{(2n)!}(2n-1)!! \sum_{i_1,\dots,i_n} \frac{\partial^{2n} f(\boldsymbol{\theta})}{\partial\theta_{i_1}^2 \dots \partial\theta_{i_n}^2} \,. \tag{6}$$

where the coefficient $(2n-1)!!$ is the number of distinct pairings of $2n$ objects, which comes from Eq. Equation (1).

Thus, the full Taylor series can be formally written as

$$\begin{aligned} \mathbb{E}[f(\boldsymbol{\theta}+\delta\boldsymbol{\theta})] &= f(\boldsymbol{\theta}) + \sum_{n=1}^{\infty} \frac{\sigma^{2n}}{(2n)!}(2n-1)!! \sum_{i_1,\dots,i_n=1}^{M} \frac{\partial^{2n} f(\boldsymbol{\theta})}{\partial\theta_{i_1}^2 \dots \partial\theta_{i_n}^2} & (7) \\ &= f(\boldsymbol{\theta}) + \frac{\sigma^2}{2}\operatorname{Tr}[H(\boldsymbol{\theta})] + \sum_{n=2}^{\infty} \frac{\sigma^{2n}}{(2n)!}(2n-1)!! \sum_{i_1,\dots,i_n=1}^{M} \frac{\partial^{2n} f(\boldsymbol{\theta})}{\partial\theta_{i_1}^2 \dots \partial\theta_{i_n}^2} & (8) \end{aligned}$$

where we introduced the Hessian matrix $H(\boldsymbol{\theta})$, whose elements are given by $[H(\boldsymbol{\theta})]_{ij} = \partial_{ij} f(\boldsymbol{\theta})$, and we see that this term represent the first non-vanishing correction to the function caused by the perturbation.

Our goal is to bound the absolute error

$$\varepsilon_{\boldsymbol{\theta}} \;:=\; |\mathbb{E}[f(\boldsymbol{\theta}+\delta\boldsymbol{\theta})] - f(\boldsymbol{\theta})| = \left| \sum_{n=1}^{\infty} \frac{\sigma^{2n}}{(2n)!}(2n-1)!! \sum_{i_1,\dots,i_n=1}^{M} \frac{\partial^{2n} f(\boldsymbol{\theta})}{\partial\theta_{i_1}^2 \dots \partial\theta_{i_n}^2} \right| \tag{9}$$

caused by the gaussian noise, and we can do that by using the property that all the derivatives of most PQC (Parametrized Quantum Circuit) are bounded. In fact, for those circuits for which a *parameter-shift* rule holds [**?** **?**], one can show that any derivative of the function $f(\boldsymbol{\theta}) = \langle O \rangle = \operatorname{Tr}\big[O\,U(\boldsymbol{\theta})\,|0\rangle\langle 0|\,U^\dagger(\boldsymbol{\theta})\big]$ obeys

$$\left| \frac{\partial^{\alpha_1+\dots\alpha_M} f(\boldsymbol{\theta})}{\partial\theta_1^{\alpha_1} \dots \partial\theta_M^{\alpha_M}} \right| \leq \|O\|_\infty \,, \tag{10}$$

where $\|O\|_\infty$ is the infinity norm of the observable, namely its largest absolute eigenvalue. We give a proof of this below in Sec. 8.

Plugging this in Eq. (9), we can obtain an upper bound to the error $\varepsilon_{\boldsymbol{\theta}}$ as desired. Indeed, remembering that for even numbers the double factorial can be expressed

as $(2n-1)!! = (2n)!/(2^n n!)$, it holds

$$\begin{aligned}
\varepsilon_{\boldsymbol{\theta}} &= \left| \sum_{n=1}^{\infty} \frac{\sigma^{2n}}{(2n)!} (2n-1)!! \sum_{i_1,\ldots,i_n=1}^{M} \frac{\partial^{2n} f(\boldsymbol{\theta})}{\partial \theta_{i_1}^2 \ldots \partial \theta_{i_n}^2} \right| && (11) \\
&\leq \sum_{n=1}^{\infty} \frac{\sigma^{2n}}{(2n)!} (2n-1)!! \sum_{i_1,\ldots,i_n=1}^{M} \underbrace{\left| \frac{\partial^{2n} f(\boldsymbol{\theta})}{\partial \theta_{i_1}^2 \ldots \partial \theta_{i_n}^2} \right|}_{\leq \|O\|_\infty} \\
&\leq \sum_{n=1}^{\infty} \frac{\sigma^{2n}}{(2n)!} (2n-1)!! \, \|O\|_\infty \, M^n && (12) \\
&= \|O\|_\infty \sum_{n=1}^{\infty} \frac{1}{(2n)!} \frac{(2n)!}{2^n n!} (\sigma^2 M)^n = \|O\|_\infty \sum_{n=1}^{\infty} \frac{(M\sigma^2/2)^n}{n!} \\
&= \|O\|_\infty \left(e^{\sigma^2 M/2} - 1 \right) \\
&\quad \implies \varepsilon_{\boldsymbol{\theta}} = |\mathbb{E}[f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})] - f(\boldsymbol{\theta})| \leq \|O\|_\infty \left(e^{M\sigma^2/2} - 1 \right), && (13)
\end{aligned}$$

where in the last line we used the definition of the exponential function $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$.

One can see that the noise variance σ^2 must scale as the inverse of the number of parameters $\sigma^2 \in \mathcal{O}\big(M^{-1}\big)$ in order to have small deviations induced by the noise. Also, note that since the difference between the noise-free function $f(\boldsymbol{\theta})$ and its perturbed version $f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})$ cannot be larger than twice the maximum eigenvalue of O, $|f(\boldsymbol{\theta} + \delta\boldsymbol{\theta}) - f(\boldsymbol{\theta})| \leq |f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})| + |f(\boldsymbol{\theta})| = 2\|O\|_\infty$, the bound (11) is informative only as long as $\exp\big[M\sigma^2/2\big] - 1 < 2$.

It is worth noticing that an identical procedure can be used to bound the average error obtained by approximating the perturbed function with its first non-vanishing correction given by the Hessian. Indeed, starting from Eq. (8) are repeating the same calculation from above, one obtains

$$\left| \mathbb{E}[f(\boldsymbol{\theta} + \delta\boldsymbol{\theta})] - f(\boldsymbol{\theta}) - \frac{\sigma^2}{2} \operatorname{Tr}[H(\boldsymbol{\theta})] \right| \leq \|O\|_\infty \left(e^{M\sigma^2/2} - 1 - \frac{M\sigma^2}{2} \right). \qquad (14)$$

Parameter-Shift rule and bounds to the derivatives

Let $f(\boldsymbol{\theta}) = \operatorname{Tr}\big[O\, U(\boldsymbol{\theta})\, |0\rangle\langle 0|\, U^\dagger(\boldsymbol{\theta})\big]$ be the expectation value of an observable O on the parametrized state $|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})\, |0\rangle$ obtained with a parametrized quantum circuit $U(\boldsymbol{\theta})$. When the variational parameters $\boldsymbol{\theta} \in \mathbb{R}^M$ enter in the quantum circuit via rotation gates of the form $V(\theta_i) = \exp[-i\theta_i P/2]$ with $P^2 = \mathbb{1}$ being

Pauli operators, then the *parameter-shift* rule can be used to evaluate gradients of the expectation value [39, 33], as described in Section 2.2.1.1,

$$\frac{\partial f(\boldsymbol{\theta})}{\partial \theta_i} = \frac{1}{2}\Big(f\Big(\boldsymbol{\theta} + \frac{\pi}{2}\boldsymbol{e_i}\Big) - f\Big(\boldsymbol{\theta} - \frac{\pi}{2}\boldsymbol{e_i}\Big)\Big)\,, \tag{15}$$

where $\boldsymbol{e}_i$ is the unit vector with zero entries and a one in the $i-$th position corresponding to angle θ_i. Similarly, by applying the parameter-shift rule twice one can express second order derivatives as follows using four evaluations of the circuit [280, 306]

$$\frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_i \theta_j} = \frac{1}{2}\Big[\frac{\partial}{\partial \theta_i} f\Big(\boldsymbol{\theta} + \frac{\pi}{2}\boldsymbol{e_j}\Big) - \frac{\partial}{\partial \theta_i} f\Big(\boldsymbol{\theta} - \frac{\pi}{2}\boldsymbol{e_j}\Big)\Big] \tag{16}$$

$$= \frac{1}{4}[f(\boldsymbol{\theta} + \frac{\pi}{2}\boldsymbol{e_j} + \frac{\pi}{2}\boldsymbol{e_i}) - f(\boldsymbol{\theta} + \frac{\pi}{2}\boldsymbol{e_j} - \frac{\pi}{2}\boldsymbol{e_i}) \tag{17}$$

$$-f\Big(\boldsymbol{\theta} - \frac{\pi}{2}\boldsymbol{e_j} + \frac{\pi}{2}\boldsymbol{e_i}\Big) + f\Big(\boldsymbol{\theta} - \frac{\pi}{2}\boldsymbol{e_j} - \frac{\pi}{2}\boldsymbol{e_i}\Big)]. \tag{18}$$

In particular, for the diagonal elements $i = j$, one has

$$\begin{aligned}\frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_i^2} &= \frac{1}{4}[f(\boldsymbol{\theta} + \pi\boldsymbol{e_i}) - 2f(\boldsymbol{\theta}) + f(\boldsymbol{\theta} - \pi\boldsymbol{e_i})] \\ &= \frac{1}{2}[f(\boldsymbol{\theta} + \pi\boldsymbol{e_i}) - f(\boldsymbol{\theta})]\,,\end{aligned} \tag{19}$$

where we used the fact that $f(\boldsymbol{\theta} + \pi\boldsymbol{e_i}) = f(\boldsymbol{\theta} - \pi\boldsymbol{e_i})$. This last equality can be seen intuitively from the 2π periodicity of the rotation gates or by direct evaluation. In fact, let $U(\boldsymbol{\theta}) = U_2\, \exp[-i\theta_i P_i/2]\, U_1$ be a factorization of the parametrized unitary where we isolated the dependence on the parameter θ_i to be shifted. Then, since $\exp[-i\, 2\pi P/2] = \cos\pi\, \mathbb{I} - i \sin\pi\, P = -\mathbb{I}$, one has

$$\begin{aligned}|\psi(\boldsymbol{\theta} - \pi\boldsymbol{e}_i)\rangle &= U_2\, \exp[-i(\theta_i - \pi)P_i/2]\, U_1\, |0\rangle \\ &= U_2\, \exp[-i(\theta_i - \pi)P_i/2] \underbrace{-\exp[-i\, 2\pi\, P_i/2]}_{\mathbb{I}} U_1\, |0\rangle \\ &= -U_2\, \exp[-i(\theta_i - \pi + 2\pi)P_i/2] U_1\, |0\rangle \\ &= -\,|\psi(\boldsymbol{\theta} + \pi\boldsymbol{e}_i)\rangle\ ,\end{aligned} \tag{20}$$

and thus $\langle\psi(\boldsymbol{\theta} - \pi\boldsymbol{e}_i)|O|\psi(\boldsymbol{\theta} - \pi\boldsymbol{e}_i)\rangle = \langle\psi(\boldsymbol{\theta} + \pi\boldsymbol{e}_i)|O|\psi(\boldsymbol{\theta} + \pi\boldsymbol{e}_i)\rangle$.

Hence, using Eq. (19) it is possible to estimate the diagonal elements of the Hessian matrix with just two different evaluations of the quantum circuit.

By repeated application of the parameter-shift rule one can also evaluate arbitrary higher-order derivatives as linear combinations of circuit evaluations [280, 51]. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M) \in \mathbb{N}^M$ be a multi-index keeping track of the orders of derivatives, and let $|\boldsymbol{\alpha}| = \sum_{i=1}^{M} \alpha_i$. Then

$$\partial^{\boldsymbol{\alpha}} f(\boldsymbol{\theta}) := \frac{\partial^{|\boldsymbol{\alpha}|} f(\boldsymbol{\theta})}{\partial \theta_1^{\alpha_1} \ldots \partial \theta_M^{\alpha_M}} = \frac{1}{2^{|\boldsymbol{\alpha}|}} \sum_{m=1}^{2^{|\boldsymbol{\alpha}|}} s_m f(\tilde{\boldsymbol{\theta}}_m)\,, \tag{21}$$

where $s_m \in \{\pm 1\}$ are signs, and $\tilde{\boldsymbol{\theta}}_m$ are angles obtained by accumulation of shifts along multiple directions.

Since the output of any circuit evaluation is bounded by the infinity norm (i.e, the largest absolute eigenvalue) of the observable $\|O\|_\infty = \max\{|o_i|\,, O = \sum_i o_i \,|o_i\rangle\langle o_i|\}$

$$|f(\boldsymbol{\theta})| = |\mathrm{Tr}[O\,\rho(\boldsymbol{\theta})]| \leq \|O\|_\infty \|\rho(\boldsymbol{\theta})\|_1 = \|O\|_\infty \quad \forall \boldsymbol{\theta} \in \mathbb{R}^M\,, \tag{22}$$

then one can bound the sum in Eq. (21) simply as

$$|\partial^{\boldsymbol{\alpha}} f(\boldsymbol{\theta})| \leq \frac{1}{2^{|\boldsymbol{\alpha}|}} \sum_{m=1}^{2^{|\boldsymbol{\alpha}|}} \left| f(\tilde{\boldsymbol{\theta}}_m) \right| \leq \|O\|_\infty\,. \tag{23}$$

Average value of the Hessian of random PQCs

In this section we derive the formulas (7.16) and (7.17) for the expected value of the Hessian as shown in the main text. Consider a system of n qubits and a parametrized quantum circuit with unitary $U(\boldsymbol{\theta}) \in \mathcal{U}(2^n)$, where $\mathcal{U}(2^n)$ is the group of unitary matrices of dimension 2^n. Given a set of parameter vectors $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K\}$, one can construct the corresponding set of unitaries $\mathbb{U} = \{U_1, U_2, \ldots, U_K\}$, with $U_i = U(\boldsymbol{\theta}_i)$ and clearly $\mathbb{U} \in \mathcal{U}(2^n)$.

It is now well known that sampling a parametrized quantum circuit from a random assignment of the parameters is approximately equal to drawing a random unitary from the Haar distribution, a phenomenon which is at the root of the insurgence of barren plateaus (BPs) [48, 229, 44]. Specifically, it is numerically observed that parametrized quantum circuits behave like unitary 2-designs, that is averaging over unitaries U_i sampled from $\mathbb{U}$ yields the same result of averaging over Haar-random unitaries, up until second order moments.

As standard in the literature regarding BPs, in the following we assume that the considered parametrized unitaries (and parts of them) are indeed 2-designs, and so

we make use of the following relations for integration over random unitaries [289, 307, 288, 48, 44]

$$\mathbb{E}_U[UAU^\dagger] = \int d\mu(U)\, UAU^\dagger = \frac{\mathbb{1}\,\mathrm{Tr}[A]}{2^n} \tag{24}$$

$$\mathbb{E}_U[AUBU^\dagger CUDU^\dagger] = \frac{\mathrm{Tr}[BD]\,\mathrm{Tr}[C]A + \mathrm{Tr}[B]\,\mathrm{Tr}[D]AC}{2^{2n}-1} \tag{25}$$

$$-\frac{\mathrm{Tr}[BD]AC + \mathrm{Tr}[B]\,\mathrm{Tr}[C]\,\mathrm{Tr}[D]A}{2^n(2^{2n}-1)} \tag{26}$$

Statistics of the Hessian

Let $f(\boldsymbol{\theta}) = \mathrm{Tr}\big[OU(\boldsymbol{\theta})\,|0\rangle\langle 0|\,U(\boldsymbol{\theta})^\dagger\big]$ and assume that the observable O is such that $\mathrm{Tr}[O] = 0$ and $\mathrm{Tr}\big[O^2\big] = 2^n$, as is the case of measuring a Pauli string. As shown in Eq. (19), diagonal elements of the Hessian matrix H can be calculated as

$$H_{ii} = \frac{\partial^2 f(\boldsymbol{\theta})}{\partial\theta_i^2} = \frac{1}{2}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i) - f(\boldsymbol{\theta})]\,. \tag{27}$$

For simplicity, from now on we drop the explicit dependence on the parameter vector $\boldsymbol{\theta}$ when not explicitly needed. The variational parameters enter the quantum circuit via Pauli rotations $e^{-i\theta_i P_i/2}$ with $P_i = P_i^\dagger$ and $P_i^2 = \mathbb{1}$, and so the shifted unitary $U(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)$ can be rewritten as

$$U(\boldsymbol{\theta}+\pi\boldsymbol{e}_i) = U_L e^{-i\pi P_i/2} U_R = -i\, U_L P_i U_R\,, \tag{28}$$

where U_L and U_R form a bipartition of the circuit at the position of the shifted angle, so that $U(\boldsymbol{\theta}) = U_L U_R$.

Assuming that the set of unitaries $\mathbb{U}_L$ generated by U_L is at least a 1-design, one has that

$$\mathbb{E}_{U_L}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)] = \mathbb{E}_{U_L}\Big[\mathrm{Tr}\Big[O\,U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger\Big]\Big] \tag{29}$$

$$= \mathrm{Tr}\Big[O\,\mathbb{E}_{U_L}\Big[U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger\Big]\Big] \tag{30}$$

$$= \mathrm{Tr}\left[O\,\frac{\mathrm{Tr}\Big[P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i\Big]\mathbb{1}}{2^n}\right] = \frac{\mathrm{Tr}[O]}{2^n} = 0\,, \tag{31}$$

where in the first line we exchanged the trace and the expectation value since both are linear operations, and in the second line we made use of Eq. (24) for the first moment of the Haar distribution. Similarly, one can show that if $\mathbb{U}_R$ forms a

1-design, then averaging over it yields the same result, namely $\mathbb{E}_{U_R}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)] = 0$. The same calculation for $f(\boldsymbol{\theta})$ shows that $\mathbb{E}_{U_R}[f(\boldsymbol{\theta})] = \mathbb{E}_{U_L}[f(\boldsymbol{\theta})] = 0$.

Thus, for every diagonal element of the Hessian, if either $\mathbb{U}_L$ or $\mathbb{U}_\mathbb{R}$ is a 1-design (that is Eq. (24) hold), then its expectation value vanishes

$$\mathbb{E}_{U_R,U_L}[H_{ii}] = 0 \quad \forall i \quad \text{if either } \mathbb{U}_L \text{ or } \mathbb{U}_\mathbb{R} \text{ is a 1-design.} \tag{32}$$

The variance of the diagonal elements can be calculated in a similar manner, even though the calculation is more involved. Substituting Eq. (27) in the defition of the variance, one obtains

$$\begin{aligned}\mathrm{Var}[H_{ii}] &:= \mathbb{E}[H_{ii}^2] - \mathbb{E}[H_{ii}]^2 = \mathbb{E}[H_{ii}^2] \\ &= \frac{1}{4}\left[\mathbb{E}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)^2] + \mathbb{E}[f(\boldsymbol{\theta})^2] - 2\mathbb{E}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)f(\boldsymbol{\theta})]\right].\end{aligned} \tag{33}$$

In order to use Eq. (26) for second moment integrals, we can rewrite these expectation values as follows

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)^2] &= \mathbb{E}\left[\mathrm{Tr}\left[O\,U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger\right]^2\right] \\ &= \mathbb{E}\left[\mathrm{Tr}\left[O\,U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger\right]\langle 0|U_R^\dagger P_i U_L^\dagger O U_L P_i U_R|0\rangle\right] \\ &= \mathbb{E}\left[\mathrm{Tr}\left[O\,U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger O U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger\right]\right] \\ &= \mathrm{Tr}\left[\mathbb{E}[O\,U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger O U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger]\right],\end{aligned} \tag{34}$$

and similarly for the remaining two terms. Assuming that the set of unitaries $\mathbb{U}_L$ generated by U_L is a 2-design, then

$$\mathbb{E}_{U_L}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)^2] = \mathrm{Tr}\left[\mathbb{E}_{U_L}[O\,U_L \underbrace{P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i}_{B} U_L^\dagger O U_L \underbrace{P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i}_{B} U_L^\dagger]\right] \tag{35}$$

$$= \mathrm{Tr}\left[\frac{\mathrm{Tr}\left[B^2\right]\mathrm{Tr}[O]O + \mathrm{Tr}[B]^2 O^2}{2^{2n}-1} - \frac{\mathrm{Tr}\left[B^2\right]O^2 + \mathrm{Tr}[B]^2\,\mathrm{Tr}[O]O}{2^n(2^{2n}-1)}\right] \tag{36}$$

$$= \frac{\mathrm{Tr}[O]^2 + \mathrm{Tr}\left[O^2\right]}{2^{2n}-1} - \frac{\mathrm{Tr}\left[O^2\right] + \mathrm{Tr}[O]^2}{2^n(2^{2n}-1)} = \frac{1}{2^n+1}, \tag{37}$$

where in the second line we made use of Eq. (26), and the third line the used that $\mathrm{Tr}[B] = \mathrm{Tr}\left[B^2\right] = 1$ since $B = P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i$ is a projector, and that $\mathrm{Tr}[O] = 0$

and $\mathrm{Tr}\big[O^2\big] = 2^n$. Similarly, one can show that integration over $\mathbb{U}_R$ yields the same result. Also, the same calculation leads to $\mathbb{E}_{U_L}[f(\boldsymbol{\theta})^2] = \mathbb{E}_{U_R}[f(\boldsymbol{\theta})^2] = 1/(2^n+1)$. Thus, if either $\mathbb{U}_L$ or $\mathbb{U}_\mathbb{R}$ is a 2-design then

$$\mathbb{E}_{U_R,U_L}[f(\boldsymbol{\theta})^2] = \mathbb{E}_{U_R,U_L}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)^2] = \frac{1}{2^n+1} \quad \forall i \quad \text{if either } \mathbb{U}_L \text{ or } \mathbb{U}_\mathbb{R} \text{ is a 2-design.} \tag{38}$$

Now we evaluate the correlation term $\mathbb{E}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)f(\boldsymbol{\theta})]$. If $\mathbb{U}_L$ is a 2-design, then

$$\begin{aligned}
\mathbb{E}_{U_L}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)f(\boldsymbol{\theta})] &= \mathrm{Tr}\left[\mathbb{E}_{U_L}\left[O\,U_L P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger U_L^\dagger O U_L U_R\,|0\rangle\langle 0|\,U_R^\dagger P_i U_L^\dagger\right]\right] \\
&= \mathrm{Tr}\left[\frac{\mathrm{Tr}\Big[P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger\Big]^2 O^2}{2^{2n}-1} - \frac{O^2}{2^n(2^{2n}-1)}\right] \\
&= \frac{1}{2^{2n}-1}\left[2^n\ \mathrm{Tr}\Big[P_i U_R\,|0\rangle\langle 0|\,U_R^\dagger\Big]^2 - 1\right].
\end{aligned} \tag{39}$$

While if $\mathbb{U}_R$ is a 2-design instead it holds

$$\begin{aligned}
\mathbb{E}_{U_R}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)f(\boldsymbol{\theta})] &= \mathrm{Tr}\left[O\,U_L P_i\,\mathbb{E}_{U_R}\left[U_R\,|0\rangle\langle 0|\,U_R^\dagger U_L^\dagger O U_L U_R\,|0\rangle\langle 0|\,U_R^\dagger\right] P_i U_L^\dagger\right] \\
&= \mathrm{Tr}\left[O\,U_L P_i\,\frac{(2^n-1)U_L^\dagger O U_L}{2^n(2^{2n}-1)}\,P_i U_L^\dagger\right] \\
&= \frac{1}{2^n(2^n+1)}\,\mathrm{Tr}\Big[O\,U_L P_i U_L^\dagger O U_L P_i U_L^\dagger\Big].
\end{aligned} \tag{40}$$

If both of them are 2-designs, then continuing from Eq. (40), one obtains

$$\begin{aligned}
&\mathbb{E}_{U_L,U_R}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)f(\boldsymbol{\theta})] = \frac{1}{2^n(2^n+1)}\,\mathrm{Tr}\left[\mathbb{E}_{U_L}\left[O\,U_L P_i U_L^\dagger O U_L P_i U_L^\dagger\right]\right] \\
&= \frac{1}{2^n(2^n+1)}\,\mathrm{Tr}\left[\frac{\mathrm{Tr}[P_i]^2 O^2 + \mathrm{Tr}\big[P_i^2\big]\,\mathrm{Tr}[O]O}{2^{2n}-1} - \frac{\mathrm{Tr}\big[P_i^2\big]O^2 + \mathrm{Tr}[P_i]^2\,\mathrm{Tr}[O]O}{2^n(2^{2n}-1)}\right] \\
&= -\frac{1}{2^n(2^n+1)}\,\frac{\mathrm{Tr}\big[P_i^2\big]\,\mathrm{Tr}\big[O^2\big]}{2^n(2^{2n}-1)} = -\frac{1}{(2^n+1)(2^{2n}-1)} \in \mathcal{O}\big(2^{-3n}\big)
\end{aligned} \tag{41}$$

Finally, plugging Eqs. (39), (40) and (41) in Eq. (33), one has $\forall i = 1, \dots, M$

$$\begin{aligned}\mathrm{Var}_{U_L,U_R}[H_{ii}] &= \frac{1}{2}\mathbb{E}[f(\boldsymbol{\theta})^2] - \frac{1}{2}\mathbb{E}[f(\boldsymbol{\theta}+\pi\boldsymbol{e}_i)f(\boldsymbol{\theta})] \\ &= \frac{1}{2(2^n+1)} - \frac{1}{2}\begin{cases} \dfrac{1}{2^{2n}-1}\left[2^n\ \mathrm{Tr}\left[P_i U_R\left|0\right\rangle\!\left\langle 0\right| U_R^\dagger\right]^2 - 1\right] & \forall i\,, \text{if } \mathbb{U}_L \text{ 2-design} \\ \dfrac{1}{2^n(2^n+1)}\,\mathrm{Tr}\left[O\,U_L P_i U_L^\dagger O U_L P_i U_L^\dagger\right] & \forall i\,, \text{if } \mathbb{U}_R \text{ 2-design} \\ -\dfrac{1}{(2^n+1)(2^{2n}-1)} & \forall i\,, \text{if } \mathbb{U}_L, \mathbb{U}_R \text{ 2-designs} \end{cases}\end{aligned} \tag{42}$$

where $\mathbb{U}_R = \mathbb{U}_R^{(i)}$ and $\mathbb{U}_L = \mathbb{U}_L^{(i)}$ are defined as in Eq. (28) and actually depend on the index i of the parameter.

Not surprisingly, as it happens for first order derivatives, also second order derivatives of PQCs are found to be exponentially vanishing [51, 48], as from Eq. (42) one can check that $\mathrm{Var}[H_{ii}] \in \mathcal{O}(2^{-n})$.

Statistics of the trace of the Hessian

The average value of the trace of the Hessian is easily found to be zero using Eq. (32), in fact

$$\mathbb{E}_{U_R,U_L}[\mathrm{Tr}[H]] = \sum_{i=1}^{M} \mathbb{E}_{U_R^{(i)},U_L^{(i)}}[H_{ii}] = 0\,, \tag{43}$$

where we assume that for every parameter i either $\mathbb{U}_R^{(i)}$ or $\mathbb{U}_L^{(i)}$ is a 1-design. The variance of the trace is instead

$$\mathrm{Var}_{U_R,U_L}[\mathrm{Tr}[H]] = \mathrm{Var}\left[\sum_{i=1}^{M} H_{ii}\right] = \sum_{i=1}^{M}\mathrm{Var}[H_{ii}] + 2\sum_{i<j}^{M}\mathrm{Cov}[H_{ii}H_{jj}]\,. \tag{44}$$

We can upper bound this quantity using the covariance inequality [308],

$$|\mathrm{Cov}[H_{ii}, H_{jj}]| \leq \sqrt{\mathrm{Var}[H_{ii}]\,\mathrm{Var}[H_{jj}]} \approx \mathrm{Var}[H_{ii}]\,,$$

were we assumed that $\mathrm{Var}[H_{ii}] \approx \mathrm{Var}[H_{jj}]\,\forall i,j$. Using that $\mathrm{Var}[H_{ii}] \in \mathcal{O}(2^{-n})$ one finally has

$$\mathrm{Var}_{U_R,U_L}[\mathrm{Tr}[H]] \leq \sum_{i=1}^{M}\mathrm{Var}[H_{ii}] + 2\sum_{i<j}^{M}\mathrm{Var}[H_{ii}] \in \mathcal{O}\left(\frac{M^2}{2^n}\right). \tag{45}$$

Alternatively, one can obtain a tighter yet qualitative approximation by explicitly considering the nature of the sums in Eq. (44). First, by using Eq. (27), the covariance term is explicitly

$$\begin{aligned}\mathrm{Cov}[H_{ii}, H_{jj}] &= \mathbb{E}[H_{ii}H_{jj}] \\ &= \frac{1}{4}\mathbb{E}[(f_i - f)(f_j - f)] \\ &= \frac{1}{4}\mathbb{E}[f^2] + \frac{1}{4}\mathbb{E}[f_i f_j] - \frac{1}{4}\mathbb{E}[f_i f] - \frac{1}{4}\mathbb{E}[f_j f]\,,\end{aligned} \tag{46}$$

where for ease of notation we defined $f_{i,j} = f(\boldsymbol{\theta} + \pi \boldsymbol{e}_{i,j})$ and $f = f(\boldsymbol{\theta})$. Note that except for the first term which is always positive, all remaining correlations terms can be both positive and negative. Also, all of these terms are bounded from above by the same quantity, as via Cauchy-Schwarz it follows

$$|\mathbb{E}[f_i f_j]| \leq \sqrt{\mathbb{E}[f_i^2]\mathbb{E}[f_j^2]} = \frac{1}{2^n + 1} \quad \text{and} \quad |\mathbb{E}[f_i f]| \leq \sqrt{\mathbb{E}[f_i^2]\mathbb{E}[f^2]} = \frac{1}{2^n + 1}\,, \tag{47}$$

where we have used $E[f^2] = E[f_i^2] = 1/(2^n + 1)$ from Eq. (38). Then, the variance can be written as

$$\begin{aligned}&\mathrm{Var}_{U_R,U_L}[\mathrm{Tr}[H]] = \sum_{i=1}^{M} \mathrm{Var}[H_{ii}] + 2\sum_{i<j}^{M} \mathbb{E}[H_{ii}H_{jj}] \\ &= \sum_{i=1}^{M} \frac{\mathbb{E}[f^2] - \mathbb{E}[f_i f]}{2} + 2\sum_{i<j}^{M} \frac{\mathbb{E}[f^2] + \mathbb{E}[f_i f_j] - \mathbb{E}[f_i f] - \mathbb{E}[f_j f]}{4} \\ &= \frac{1}{2}\left(\sum_{i=1}^{M} + \sum_{i<j}^{M}\right)\mathbb{E}[f^2] - \frac{1}{2}\left(\sum_{i=1}^{M}\mathbb{E}[f_i f] + \sum_{i<j}^{M}\mathbb{E}[f_i f] + \sum_{i<j}^{M}\mathbb{E}[f_j f]\right) + \frac{1}{2}\sum_{i<j}^{M}\mathbb{E}[f_i f_j] \\ &= \frac{M(M+1)}{4}\mathbb{E}[f^2] - \underbrace{\frac{M}{2}\sum_{i=1}^{M}\mathbb{E}[f_i f] + \frac{1}{2}\sum_{i<j}^{M}\mathbb{E}[f_i f_j]}_{\Delta}\,.\end{aligned} \tag{48}$$

Numerical simulations In addition to Figure 7.6 in the main text, in Figure 5 we report numerical evidence for the trace of the Hessian for two common hardware-efficient parametrized quantum circuit ansatzes. The histograms represent the frequency of obtaining a given value of the trace of the Hessian $\mathrm{Tr}[H(\boldsymbol{\theta})]$ upon random assignments of the parameters. The length of the arrows are, respectively: "Numerical 2σ" (black solid line) twice the statistical standard deviation computed from the numerical results, "Approximation" (dashed red) twice the square root of

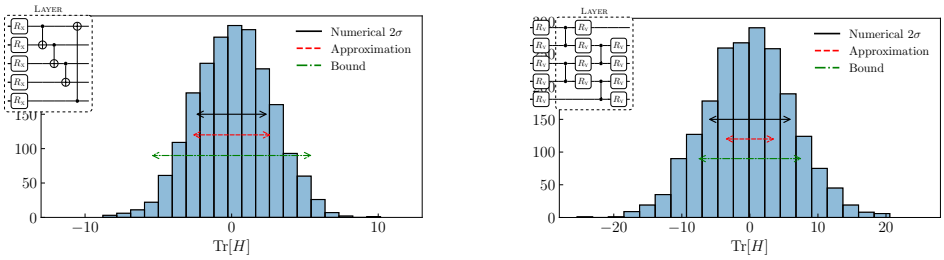


Figure 5: Simulation results of evaluating the trace of the Hessian matrix for two different hardware-efficient ansatzes with random values of the parameters. The plot on the left is obtained using the layer template shown in the figure for $n = 6$ qubits and $l = 6$ layers. The plot on the right instead with $n = 5$ and $l = 5$ layers of the template shown in the corresponding inset. The simulations are performed by sampling 2000 random parameter vectors $\boldsymbol{\theta}_m$ with $\theta_i \sim \text{Unif}[0, 2\pi[$, evaluating the trace of the Hessian matrix $\text{Tr}[H(\boldsymbol{\theta})]$, and then building the histogram to show its frequency distribution. In both experiments the measured observable is $Z^{\otimes n}$. The length of the arrows are respectively: "Numerical 2σ" (black solid line) twice the numerical standard deviation, "Approximation" (dashed red) twice the square root of the approximation in Eq. (49), "Bound" (dashed-dotted green) twice the square root of the upper Bound in Eq. (45). These parametrized circuits correspond to the templates `BasicEntanglinLayer` and `Simplified2Design` defined in Pennylane [**?**], and used for example in [44] to study barren plateaus.

the Eq. (48) with $\Delta = 0$, “Bound" (dashed-dotted green) twice the square root of the upper Bound in Eq. (45).

All simulations confirm the bound (45), and, more interestingly, both the circuit on the left of Fig. 5 and the one in Fig. 7.6 in the main text, have a numerical variance which is very well approximated by Eq. (48) with $\Delta = 0$. We conjecture this is due to the fact that all correlation terms in Eq. (48) are roughly of the same order of magnitude (see Eq. (47)), and can be either positive and negative, depending on the parameter and the specifics of the ansatz. Thus, one can expect the whole contribution to either vanish $\Delta \approx 0$, or be negligible with respect to the leading term. If this is the case, then substituting $\mathbb{E}[f^2] = 1/(2^n + 1)$, the variance of the Hessian is approximately

$$\mathrm{Var}_{U_R,U_L}[\mathrm{Tr}[H]] \approx \frac{M(M+1)}{4}\mathbb{E}[f^2] = \frac{M(M+1)}{4(2^n+1)} \approx \frac{1}{4}\frac{M^2}{2^n}\,, \tag{49}$$

which is four times smaller then the upper bound Eq. (45), but clearly has the same scaling. While we numerically verified it also at other number of qubits, more investigations are needed to understand if and when this approximation holds, and we leave a detailed study of this phenomenon for future work.

Additional results for flexible vs. fixed number of shots in Q-learning

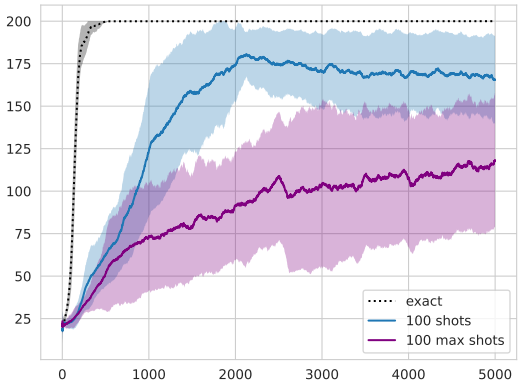
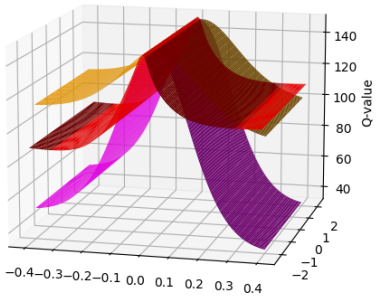
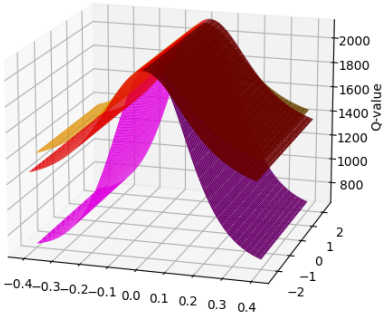


Figure 6: Performance of agents trained with a fixed number of 100 shots (blue) and $m_{\max} = 100$ with flexible shot allocation (purple), compared to model trained without shot noise (black dotted curve).

Visualization of CartPole policies obtained with Q-learning



(a) $\sigma = 0$



(b) $\sigma = 0.2$

Figure 7: Visualization of the Q-functions learned in the noise-free (a) and noisy (b) settings. The red surface shows Q-values for pole angle and cart position, orange for pole angle and cart velocity, and magenta for pole angle and pole velocity.

Bibliography

[1] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[3] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[4] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[5] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

[6] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013.

[7] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[8] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical review letters*, 100(16):160501, 2008.

[9] Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291–293, 2015.

[10] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, pages 217–228, 2019.

[11] Ainesh Bakshi and Ewin Tang. An improved classical singular value transformation for quantum machine learning. *arXiv preprint arXiv:2303.01492*, 2023.

[12] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[13] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S Kottmann, Tim Menke, et al. Noisy intermediate-scale quantum (nisq) algorithms. *arXiv preprint arXiv:2101.08448*, 2021.

[14] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[15] E. Farhi, J. Goldstone, and S. Gutmann. A Quantum Approximate Optimization Algorithm. *Preprint at arXiv:1411.4028*, 2014.

[16] Zhihui Wang, Stuart Hadfield, Zhang Jiang, and Eleanor G Rieffel. Quantum approximate optimization algorithm for maxcut: A fermionic view. *Physical Review A*, 97(2):022304, 2018.

[17] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor Rieffel, Davide Venturelli, Rupak Biswas, Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor G. Rieffel, Davide Venturelli, and Rupak Biswas. From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz. *Algorithms*, 12(2):34, feb 2019.

[18] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014.

[19] M-H Yung, J Casanova, Antonio Mezzacapo, J McClean, L Lamata, A Aspuru-Guzik, and E Solano. From transistor to trapped-ion computers for quantum chemistry. *Scientific Reports*, 4(3589):1–7, 2014.

[20] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[21] P. J. J. O'Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding, et al. Scalable quantum simulation of molecular energies. *Physical Review X*, 6(3):031007, 2016.

[22] Raffaele Santagati, Jianwei Wang, Antonio A. Gentile, Stefano Paesani, Nathan Wiebe, Jarrod R. McClean, Sam Morley-Short, Peter J. Shadbolt, Damien Bonneau, Joshua W. Silverstone, David P. Tew, Xiaoqi Zhou, Jeremy L. O'Brien, and Mark G. Thompson. Witnessing eigenstates for quantum simulation of hamiltonian spectra. *Science Advances*, 4:1, 2018.

[23] Jonathan Romero, Jonathan P Olson, and Alan Aspuru-Guzik. Quantum autoencoders for efficient compression of quantum data. *Quantum Science and Technology*, 2(4):045001, 2017.

[24] Edward Grant, Marcello Benedetti, Shuxiang Cao, Andrew Hallam, Joshua Lockhart, Vid Stojevic, Andrew G. Green, and Simone Severini. Hierarchical quantum classifiers. *npj Quantum Information*, 4(1):65, dec 2018.

[25] Jin-Guo Liu and Lei Wang. Differentiable learning of quantum circuit Born machines. *Physical Review A*, 98(6):062324, dec 2018.

[26] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[27] Marcello Benedetti, Edward Grant, Leonard Wossnig, and Simone Severini. Adversarial quantum circuit learning for pure state approximation. *New Journal of Physics*, 21(4):043023, apr 2019.

[28] Brian Coyle, Daniel Mills, Vincent Danos, and Elham Kashefi. The born supremacy: Quantum advantage and training of an ising born machine. *npj Quantum Information*, 6(1):1–11, 2020.

[29] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[30] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[31] Kathryn Tunyasuvunakool, Jonas Adler, Zachary Wu, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, et al. Highly accurate protein structure prediction for the human proteome. *Nature*, 596(7873):590–596, 2021.

[32] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[33] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3):032331, mar 2019.

[34] Kevin J Sung, Jiahao Yao, Matthew P Harrigan, Nicholas C Rubin, Zhang Jiang, Lin Lin, Ryan Babbush, and Jarrod R McClean. Using models to improve optimizers for variational quantum algorithms. *Quantum Science and Technology*, 5(4):044008, 2020.

[35] Xavier Bonet-Monroig, Hao Wang, Diederick Vermetten, Bruno Senjean, Charles Moussa, Thomas Bäck, Vedran Dunjko, and Thomas E O'Brien. Performance comparison of optimization methods on variational quantum algorithms. *arXiv preprint arXiv:2111.13454*, 2021.

[36] Aidan Pellow-Jarman, Ilya Sinayskiy, Anban Pillay, and Francesco Petruccione. A comparison of various classical optimizers for a variational quantum linear solver. *Quantum Information Processing*, 20(6):1–14, 2021.

[37] Samuel Yen-Chi Chen, Chih-Min Huang, Chia-Wei Hsing, Hsi-Sheng Goan, and Ying-Jer Kao. Variational quantum reinforcement learning via evolutionary optimization. *Machine Learning: Science and Technology*, 3(1):015025, 2022.

[38] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[39] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

[40] Gavin E Crooks. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. *arXiv preprint arXiv:1905.13311*, 2019.

[41] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):4812, dec 2018.

[42] Aram W. Harrow and Richard A. Low. Random Quantum Circuits are Approximate 2-designs. *Communications in Mathematical Physics*, 291(1):257–302, oct 2009.

[43] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, jun 2018.

[44] Marco Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature communications*, 12(1):1–12, 2021.

[45] AV Uvarov and Jacob D Biamonte. On barren plateaus and cost function locality in variational quantum algorithms. *Journal of Physics A: Mathematical and Theoretical*, 54(24):245301, 2021.

[46] Zoë Holmes, Andrew Arrasmith, Bin Yan, Patrick J Coles, Andreas Albrecht, and Andrew T Sornborger. Barren plateaus preclude learning scramblers, arxiv preprints. *arXiv preprint arXiv:2009.14808*, 2020.

[47] Carlos Ortiz Marrero, Mária Kieferová, and Nathan Wiebe. Entanglement-induced barren plateaus. *PRX Quantum*, 2(4):040316, 2021.

[48] Zoë Holmes, Kunal Sharma, Marco Cerezo, and Patrick J Coles. Connecting ansatz expressibility to gradient magnitudes and barren plateaus. *PRX Quantum*, 3(1):010313, 2022.

[49] Samson Wang, Enrico Fontana, Marco Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J Coles. Noise-induced barren plateaus in variational quantum algorithms. *Nature communications*, 12(1):1–11, 2021.

[50] Daniel Stilck França and Raul Garcia-Patron. Limitations of optimization algorithms on noisy quantum devices. *Nature Physics*, 17(11):1221–1227, 2021.

[51] Marco Cerezo and Patrick J Coles. Higher order derivatives of quantum neural networks with barren plateaus. *Quantum Science and Technology*, 6(3):035006, 2021.

[52] Andrew Arrasmith, M Cerezo, Piotr Czarnik, Lukasz Cincio, and Patrick J Coles. Effect of barren plateaus on gradient-free optimization. *Quantum*, 5:558, 2021.

[53] Edward Grant, Leonard Wossnig, Mateusz Ostaszewski, and Marcello Benedetti. An initialization strategy for addressing barren plateaus in parametrized quantum circuits. *Quantum*, 3:214, 2019.

[54] Tyler Volkoff and Patrick J Coles. Large gradients via correlation in random parameterized quantum circuits. *Quantum Science and Technology*, 6(2):025008, 2021.

[55] Iris Cong, Soonwon Choi, and Mikhail D Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, 2019.

[56] Arthur Pesah, M Cerezo, Samson Wang, Tyler Volkoff, Andrew T Sornborger, and Patrick J Coles. Absence of barren plateaus in quantum convolutional neural networks. *Physical Review X*, 11(4):041011, 2021.

[57] Louis Schatzki, Martin Larocca, Frederic Sauvage, and M Cerezo. Theoretical guarantees for permutation-equivariant quantum neural networks. *arXiv preprint arXiv:2210.09974*, 2022.

[58] Stefan H Sack, Raimel A Medina, Alexios A Michailidis, Richard Kueng, and Maksym Serbyn. Avoiding barren plateaus using classical shadows. *PRX Quantum*, 3(2):020365, 2022.

[59] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[60] Gavin E Crooks. Performance of the quantum approximate optimization algorithm on the maximum cut problem. *arXiv preprint arXiv:1811.08419*, 2018.

[61] Fernando G. Brandao, Michael Broughton, Edward Farhi, Sam Gutmann, and Hartmut Neven. For fixed control parameters the quantum approximate optimization algorithm's objective function value concentrates for typical instances. *arXiv preprint arXiv:1812.04170*, 2018.

[62] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor G Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, 2019.

[63] Gian Giacomo Guerreschi and Anne Y Matsuura. Qaoa for max-cut requires hundreds of qubits for quantum speed-up. *Scientific reports*, 9(1):1–7, 2019.

[64] Román Orús, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4:100028, 2019.

[65] Michael Streif, Sheir Yarkoni, Andrea Skolik, Florian Neukart, and Martin Leib. Beating classical heuristics for the binary paint shop problem with the quantum approximate optimization algorithm. *Physical Review A*, 104(1):012403, 2021.

[66] Andre Luckow, Johannes Klepsch, and Josef Pichlmeier. Quantum computing: Towards industry reference problems. *Digitale Welt*, 5(2):38–45, 2021.

[67] Constantin Dalyac, Loïc Henriet, Emmanuel Jeandel, Wolfgang Lechner, Simon Perdrix, Marc Porcheron, and Margarita Veshchezerova. Qualifying quantum approaches for hard industrial optimization problems. a case study in the field of smart-charging of electric vehicles. *EPJ Quantum Technology*, 8(1):12, 2021.

[68] Matthew P Harrigan, Kevin J Sung, Matthew Neeley, Kevin J Satzinger, Frank Arute, Kunal Arya, Juan Atalaya, Joseph C Bardin, Rami Barends, Sergio Boixo, et al. Quantum approximate optimization of non-planar graph problems on a planar superconducting processor. *Nature Physics*, 17(3):332–336, 2021.

[69] Sepehr Ebadi, Alexander Keesling, Madelyn Cain, Tout T Wang, Harry Levine, Dolev Bluvstein, Giulia Semeghini, Ahmed Omran, J-G Liu, Rhine Samajdar, et al. Quantum optimization of maximum independent set using rydberg atom arrays. *Science*, page eabo6587, 2022.

[70] Ying Li and Simon C Benjamin. Efficient variational quantum simulator incorporating active error minimization. *Physical Review X*, 7(2):021050, 2017.

[71] Harper R Grimsley, Sophia E Economou, Edwin Barnes, and Nicholas J Mayhall. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature communications*, 10(1):1–9, 2019.

[72] Arthur G Rattew, Shaohan Hu, Marco Pistoia, Richard Chen, and Steve Wood. A domain-agnostic, noise-resistant, hardware-efficient evolutionary variational quantum eigensolver. *arXiv preprint arXiv:1910.09694*, 2019.

[73] Google AI Quantum, Collaborators*†, Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B Buckley, et al. Hartree-fock on a superconducting qubit quantum computer. *Science*, 369(6507):1084–1089, 2020.

[74] Maria Schuld, Alex Bocharov, Krysta M Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *Physical Review A*, 101(3):032308, 2020.

[75] Andrea Skolik, Sofiene Jerbi, and Vedran Dunjko. Quantum agents in the gym: a variational quantum algorithm for deep q-learning. *Quantum*, 6:720, 2022.

[76] André Sequeira, Luis Paulo Santos, and Luís Soares Barbosa. Variational quantum policy gradients with an application to quantum control. *arXiv preprint arXiv:2203.10591*, 2022.

[77] Asel Sagingalieva, Andrii Kurkin, Artem Melnikov, Daniil Kuhmistrov, Michael Perelshtein, Alexey Melnikov, Andrea Skolik, and David Von Dollen. Hyperparameter optimization of hybrid quantum neural networks for car classification. *arXiv preprint arXiv:2205.04878*, 2022.

[78] Kristan Temme, Sergey Bravyi, and Jay M Gambetta. Error mitigation for short-depth quantum circuits. *Physical review letters*, 119(18):180509, 2017.

[79] Suguru Endo, Simon C Benjamin, and Ying Li. Practical quantum error mitigation for near-future applications. *Physical Review X*, 8(3):031027, 2018.

[80] Philippe Suchsland, Francesco Tacchino, Mark H Fischer, Titus Neupert, Panagiotis Kl Barkoutsos, and Ivano Tavernelli. Algorithmic error mitigation scheme for current quantum processors. *Quantum*, 5:492, 2021.

[81] Ryuji Takagi, Suguru Endo, Shintaro Minagawa, and Mile Gu. Fundamental limits of quantum error mitigation. *npj Quantum Information*, 8(1):1–11, 2022.

[82] Yihui Quek, Daniel Stilck França, Sumeet Khatri, Johannes Jakob Meyer, and Jens Eisert. Exponentially tighter bounds on limitations of quantum error mitigation. *arXiv preprint arXiv:2210.11505*, 2022.

[83] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[84] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[85] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[86] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[87] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986.

[88] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[89] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*, 2017.

[90] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[91] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in neural information processing systems*, 21, 2008.

[92] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[93] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[94] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[95] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[96] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017.

[97] Vitaly Maiorov and Allan Pinkus. Lower bounds for approximation by mlp neural networks. *Neurocomputing*, 25(1-3):81–91, 1999.

[98] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[99] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[100] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[101] David J Gross. The role of symmetry in fundamental physics. *Proceedings of the National Academy of Sciences*, 93(25):14256–14259, 1996.

[102] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[103] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[104] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[105] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[106] Elman Mansimov, Omar Mahmood, Seokho Kang, and Kyunghyun Cho. Molecular geometry prediction using a deep generative graph neural network. *Scientific reports*, 9(1):1–13, 2019.

[107] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[108] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.

[109] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[110] Francisco S Melo. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.

[111] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[112] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer, 1999.

[113] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.

[114] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[115] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[116] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[117] Long-Ji Lin. *Self-supervised Learning by Reinforcement and Artificial Neural Networks*. PhD thesis, Carnegie Mellon University, School of Computer Science, 1992.

[118] Francisco S Melo and M Isabel Ribeiro. Q-learning with linear function approximation. In *International Conference on Computational Learning Theory*, pages 308–322. Springer, 2007.

[119] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[120] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.

[121] Vedran Dunjko and Peter Wittek. A non-review of quantum machine learning: trends and explorations. *Quantum Views*, 4:32, 2020.

[122] M Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2(9):567–576, 2022.

[123] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13:2567–2586, 2014.

[124] Bernhard Schölkopf, Alexander J Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[125] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[126] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Physical review letters*, 122(4):040504, 2019.

[127] Maria Schuld. Supervised quantum machine learning models are kernel methods. *arXiv preprint arXiv:2101.11020*, 2021.

[128] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. A rigorous and robust quantum speed-up in supervised machine learning. *Nature Physics*, pages 1–5, 2021.

[129] Evan Peters, João Caldeira, Alan Ho, Stefan Leichenauer, Masoud Mohseni, Hartmut Neven, Panagiotis Spentzouris, Doug Strain, and Gabriel N Perdue. Machine learning of high dimensional data on a noisy quantum processor. *npj Quantum Information*, 7(1):1–5, 2021.

[130] Supanut Thanasilp, Samson Wang, Marco Cerezo, and Zoë Holmes. Exponential concentration and untrainability in quantum kernel methods. *arXiv preprint arXiv:2208.11060*, 2022.

[131] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[132] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. arxiv e-prints, art. *arXiv preprint arXiv:1811.03962*, 2018.

[133] SS Du, JD Lee, H Li, L Wang, and X Zhai. Gradient descent finds global minima of deep 774 neural networks. *CoRR abs/1811.03804*, 775, 2018.

[134] Sofiene Jerbi, Lukas J Fiderer, Hendrik Poulsen Nautrup, Jonas M Kübler, Hans J Briegel, and Vedran Dunjko. Quantum machine learning beyond kernel methods. *arXiv preprint arXiv:2110.13162*, 2021.

[135] Norihito Shirai, Kenji Kubo, Kosuke Mitarai, and Keisuke Fujii. Quantum tangent kernel. *arXiv preprint arXiv:2111.02951*, 2021.

[136] Junyu Liu, Francesco Tacchino, Jennifer R Glick, Liang Jiang, and Antonio Mezzacapo. Representation learning via quantum neural tangent kernels. *PRX Quantum*, 3(3):030323, 2022.

[137] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.

[138] Edward Grant, Marcello Benedetti, Shuxiang Cao, Andrew Hallam, Joshua Lockhart, Vid Stojevic, Andrew G Green, and Simone Severini. Hierarchical quantum classifiers. *npj Quantum Information*, 4(1):1–8, 2018.

[139] Jin-Guo Liu and Lei Wang. Differentiable learning of quantum circuit born machines. *Physical Review A*, 98(6):062324, 2018.

[140] Marcello Benedetti, Delfina Garcia-Pintos, Oscar Perdomo, Vicente Leyton-Ortega, Yunseong Nam, and Alejandro Perdomo-Ortiz. A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Information*, 5(1):1–9, 2019.

[141] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[142] Pierre-Luc Dallaire-Demers and Nathan Killoran. Quantum generative adversarial networks. *Physical Review A*, 98(1):012324, 2018.

[143] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 5(1):1–9, 2019.

[144] He-Liang Huang, Yuxuan Du, Ming Gong, Youwei Zhao, Yulin Wu, Chaoyue Wang, Shaowei Li, Futian Liang, Jin Lin, Yu Xu, et al. Experimental quantum generative adversarial networks for image generation. *Physical Review Applied*, 16(2):024051, 2021.

[145] Murphy Yuezhen Niu, Alexander Zlokapa, Michael Broughton, Sergio Boixo, Masoud Mohseni, Vadim Smelyanskyi, and Hartmut Neven. Entangling quantum generative adversarial networks. *Physical Review Letters*, 128(22):220505, 2022.

[146] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[147] Nathan Wiebe and Leonard Wossnig. Generative training of quantum boltzmann machines with hidden units. *arXiv preprint arXiv:1905.09902*, 2019.

[148] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Variational quantum boltzmann machines. *Quantum Machine Intelligence*, 3(1):1–15, 2021.

[149] Guillaume Verdon, Trevor McCourt, Enxhell Luzhnica, Vikash Singh, Stefan Leichenauer, and Jack Hidary. Quantum graph neural networks. *arXiv preprint arXiv:1909.12264*, 2019.

[150] Sofiene Jerbi, Casper Gyurik, Simon Marshall, Hans Briegel, and Vedran Dunjko. Parametrized quantum policies for reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[151] Dániel Nagy, Zsolt Tabi, Péter Hága, Zsófia Kallus, and Zoltán Zimborás. Photonic quantum policy learning in openai gym. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 123–129. IEEE, 2021.

[152] Nico Meyer, Daniel D Scherer, Axel Plinge, Christopher Mutschler, and Michael J Hartmann. Quantum policy gradient algorithm with optimized action decoding. *arXiv preprint arXiv:2212.06663*, 2022.

[153] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. *IEEE Access*, 8:141007–141024, 2020.

[154] Owen Lockwood and Mei Si. Reinforcement learning with quantum variational circuit. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 245–251, 2020.

[155] Dirk Heimann, Hans Hohenfeld, Felix Wiebe, and Frank Kirchner. Quantum deep reinforcement learning for robot navigation tasks. *arXiv preprint arXiv:2202.12180*, 2022.

[156] Maja Franz, Lucas Wolf, Maniraman Periyasamy, Christian Ufrecht, Daniel D Scherer, Axel Plinge, Christopher Mutschler, and Wolfgang Mauerer. Uncovering instabilities in variational-quantum deep q-networks. *arXiv preprint arXiv:2202.05195*, 2022.

[157] Qingfeng Lan. Variational quantum soft actor-critic. *arXiv preprint arXiv:2112.11921*, 2021.

[158] Won Joon Yun, Yunseok Kwak, Jae Pyoung Kim, Hyunhee Cho, Soyi Jung, Jihong Park, and Joongheon Kim. Quantum multi-agent reinforcement learning via variational quantum circuit design. *arXiv preprint arXiv:2203.10443*, 2022.

[159] Chen Zhao and Xiao-Shan Gao. Qdnn: Dnn with quantum neural network layers. *arXiv preprint arXiv:1912.12660*, 2019.

[160] Andrea Mari, Thomas R Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. *Quantum*, 4:340, 2020.

[161] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, and Tristan Cook. Quanvolutional neural networks: powering image recognition with quantum circuits. *Quantum Machine Intelligence*, 2(1):1–9, 2020.

[162] Tong Dou, Kaiwei Wang, Zhenwei Zhou, Shilu Yan, and Wei Cui. An unsupervised feature learning for quantum-classical convolutional network with applications to fault detection. In *2021 40th Chinese Control Conference (CCC)*, pages 6351–6355. IEEE, 2021.

[163] Samuel Yen-Chi Chen, Chih-Min Huang, Chia-Wei Hsing, and Ying-Jer Kao. An end-to-end trainable hybrid classical-quantum classifier. *Machine Learning: Science and Technology*, 2(4):045021, 2021.

[164] Emanuel Knill, Raymond Laflamme, and Gerald J Milburn. A scheme for efficient quantum computation with linear optics. *nature*, 409(6816):46–52, 2001.

[165] Nathan Wiebe. Key questions for the quantum machine learner to ask themselves. *New Journal of Physics*, 22(9):091001, 2020.

[166] Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Physical Review A*, 102(3):032420, 2020.

[167] Francisco Javier Gil Vidal and Dirk Oliver Theis. Input redundancy for parameterized quantum circuits. *Frontiers in Physics*, 8:297, 2020.

[168] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, 2020.

[169] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A*, 103(3):032430, 2021.

[170] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[171] Han Zheng, Zimu Li, Junyu Liu, Sergii Strelchuk, and Risi Kondor. Speeding up learning quantum states through group equivariant convolutional quantum ansatze. *arXiv preprint arXiv:2112.07611*, 2021.

[172] Péter Mernyei, Konstantinos Meichanetzidis, and İsmail İlkan Ceylan. Equivariant quantum graph circuits. *arXiv preprint arXiv:2112.05261*, 2021.

[173] Martín Larocca, Frédéric Sauvage, Faris M. Sbahi, Guillaume Verdon, Patrick J. Coles, and M. Cerezo. Group-invariant quantum machine learning. *arXiv preprint arXiv:2205.02261*, 2022.

[174] Andrea Skolik, Michele Cattelan, Sheir Yarkoni, Thomas Bäck, and Vedran Dunjko. Equivariant quantum circuits for learning on weighted graphs. *arXiv preprint arXiv:2205.06109*, 2022.

[175] Johannes Jakob Meyer, Marian Mularski, Elies Gil-Fuster, Antonio Anna Mele, Francesco Arzani, Alissa Wilms, and Jens Eisert. Exploiting symmetry in variational quantum machine learning. *arXiv preprint arXiv:2205.06217*, 2022.

[176] Quynh T Nguyen, Louis Schatzki, Paolo Braccia, Michael Ragone, Patrick J Coles, Frederic Sauvage, Martin Larocca, and M Cerezo. Theory for equivariant quantum neural networks. *arXiv preprint arXiv:2210.08566*, 2022.

[177] Hsin-Yuan Huang, Michael Broughton, Jordan Cotler, Sitan Chen, Jerry Li, Masoud Mohseni, Hartmut Neven, Ryan Babbush, Richard Kueng, John Preskill, et al. Quantum advantage in learning from experiments. *Science*, 376(6598):1182–1186, 2022.

[178] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. Power of data in quantum machine learning. *Nature communications*, 12(1):1–9, 2021.

[179] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, 2021.

[180] Edward Farhi and Aram W Harrow. Quantum supremacy through the quantum approximate optimization algorithm. *arXiv preprint arXiv:1602.07674*, 2016.

[181] Ryan Sweke, Jean-Pierre Seifert, Dominik Hangleiter, and Jens Eisert. On the quantum versus classical learnability of discrete distributions. *Quantum*, 5:417, 2021.

[182] Dave Wecker, Matthew B Hastings, and Matthias Troyer. Progress towards practical quantum variational algorithms. *Physical Review A*, 92(4):042303, 2015.

[183] Giacomo Nannicini. Performance of hybrid quantum-classical variational heuristics for combinatorial optimization. *Physical Review E*, 99(1):013304, 2019.

[184] Ken M Nakanishi, Keisuke Fujii, and Synge Todo. Sequential minimal optimization for quantum-classical hybrid algorithms. *Physical Review Research*, 2(4):043158, 2020.

[185] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D Lukin. Quantum approximate optimization algorithm: performance, mechanism, and implementation on near-term devices. *Preprint at arXiv:1812.01041*, 2018.

[186] Kevin Jeffery Sung, Jiahao Yao, Matthew Harrigan, Nicholas Rubin, Zhang Jiang, Lin Lin, Ryan Babbush, and Jarrod McClean. Using models to improve optimizers for variational quantum algorithms. *Quantum Science and Technology*, 2020.

[187] Emanuel Knill, Gerardo Ortiz, and Rolando D. Somma. Optimal quantum measurements of expectation values of observables. *Physical Review A*, 75(1):012328, jan 2007.

[188] Bobak Toussi Kiani, Seth Lloyd, and Reevu Maity. Learning Unitaries by Gradient Descent. 2020.

[189] Chufan Lyu, Victor Montenegro, and Abolfazl Bayat. Accelerated variational algorithms for digital quantum simulation of many-body ground states. *Quantum*, 4:324, 2020.

[190] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. *Advances in Neural Information Processing*, 2:524—-532, 1990.

[191] Chris Hettinger, Tanner Christensen, Ben Ehlert, Jeffrey Humpherys, Tyler Jarvis, and Sean Wade. Forward Thinking: Building and Training Neural Networks One Layer at a Time. In *31st Conference on Neural Information Processing Systems*, 2017.

[192] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, jul 2006.

[193] Yoshua Bengio, Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing*, 2007.

[194] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[195] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Physical review letters*, 121(4):040502, 2018.

[196] Shouvanik Chakrabarti, Huang Yiming, Tongyang Li, Soheil Feizi, and Xiaodi Wu. Quantum wasserstein generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 6781–6792, 2019.

[197] A Hamann, V Dunjko, and S Wölk. Quantum-accessible reinforcement learning beyond strictly epochal environments. *arXiv preprint arXiv:2008.01481*, 2020.

[198] Sofiene Jerbi, Lea M Trenkwalder, Hendrik Poulsen Nautrup, Hans J Briegel, and Vedran Dunjko. Quantum enhancements for deep reinforcement learning in large spaces. *PRX Quantum*, 2(1):010328, 2021.

[199] Shaojun Wu, Shan Jin, Dingding Wen, and Xiaoting Wang. Quantum reinforcement learning in continuous action space. *arXiv preprint arXiv:2012.10711*, 2020.

[200] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[201] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[202] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[203] Ziyu Ye, Andrew Gilman, Qihang Peng, Kelly Levick, Pamela Cosman, and Larry Milstein. Comparison of neural network architectures for spectrum sensing. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2019.

[204] Hao Yu, Tiantian Xie, Michael Hamilton, and Bogdan Wilamowski. Comparison of different neural network architectures for digit image recognition. In *2011 4th International Conference on Human System Interactions, HSI 2011*, pages 98–103. IEEE, 2011.

[205] F Cordoni. A comparison of modern deep neural network architectures for energy spot price forecasting. *Digital Finance*, 2:189–210, 2020.

[206] Tomasz Szandała. Review and comparison of commonly used activation functions for deep neural networks. In *Bio-inspired Neurocomputing*, pages 203–224. Springer, 2021.

[207] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[208] Sebastian Urban. *Neural network architectures and activation functions: A gaussian process approach*. PhD thesis, Technische Universität München, 2018.

[209] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[210] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[211] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[212] Bobak Toussi Kiani, Seth Lloyd, and Reevu Maity. Learning unitaries by gradient descent. *arXiv preprint arXiv:2001.11897*, 2020.

[213] Roeland Wiersema, Cunlu Zhou, Yvette de Sereville, Juan Felipe Carrasquilla, Yong Baek Kim, and Henry Yuen. Exploring entanglement and optimization within the hamiltonian variational ansatz. *PRX Quantum*, 1(2):020319, 2020.

[214] Andrea Skolik, Jarrod R McClean, Masoud Mohseni, Patrick van der Smagt, and Martin Leib. Layerwise learning for quantum neural networks. *Quantum Machine Intelligence*, 3 (1):1–11, 2021.

[215] Sukin Sim, Peter D Johnson, and Alán Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12):1900070, 2019.

[216] Sukin Sim, Jhonathan Romero Fontalvo, Jérôme F Gonthier, and Alexander A Kunitsa. Adaptive pruning-based optimization of parameterized quantum circuits. *Quantum Science and Technology*, 2021.

[217] Xiaoyuan Liu, Anthony Angone, Ruslan Shaydulin, Ilya Safro, Yuri Alexeev, and Lukasz Cincio. Layer vqe: A variational approach for combinatorial optimization on noisy quantum computers. *arXiv preprint arXiv:2102.05566*, 2021.

[218] OpenAI. Openai gym wiki, cartpole v0 https://github.com/openai/gym/wiki/cartpole-v0, 2021.

[219] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[220] Kei Ota, Devesh K Jha, and Asako Kanezaki. Training larger networks for deep reinforcement learning. *arXiv preprint arXiv:2102.07920*, 2021.

[221] Andrea Skolik. Code used in this work https://github.com/askolik/quantum_agents, 2021.

[222] Vedran Dunjko, Yi-Kai Liu, Xingyao Wu, and Jacob M Taylor. Exponential improvements for quantum-accessible reinforcement learning. *arXiv preprint arXiv:1710.11160*, 2017.

[223] OpenAI. Openai gym wiki, frozen lake v0 https://github.com/openai/gym/wiki/frozenlake-v0, 2021.

[224] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J Martinez, Jae Hyeon Yoo, Sergei V Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, et al. Tensorflow quantum: A software framework for quantum machine learning. *arXiv preprint arXiv:2003.02989*, 2020.

[225] Google. Cirq, https://quantumai.google/cirq, 2021.

[226] OpenAI. Openai gym leaderboard https://github.com/openai/gym/wiki/leaderboard, 2021.

[227] Tavis Bennett, Edric Matwiejew, Sam Marsh, and Jingbo B Wang. Quantum walk-based vehicle routing optimisation. *Frontiers in Physics*, page 692, 2021.

[228] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014.

[229] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature communications*, 9(1):1–6, 2018.

[230] Patrick van der Smagt and Gerd Hirzinger. Why feed-forward networks are in a bad shape. In *International Conference on Artificial Neural Networks*, pages 159–164. Springer, 1998.

[231] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation.

In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[232] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[233] Louis-Paul Henry, Slimane Thabet, Constantin Dalyac, and Loïc Henriet. Quantum evolution kernel: Machine learning on graphs with programmable arrays of qubits. *Physical Review A*, 104(3):032416, 2021.

[234] Jin Zheng, Qing Gao, and Yanxuan Lü. Quantum graph convolutional neural networks. In *2021 40th Chinese Control Conference (CCC)*, pages 6335–6340. IEEE, 2021.

[235] Andrew Lucas. Ising formulations of many np problems. *Frontiers in physics*, 2:5, 2014.

[236] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.

[237] Jianwu Long et al. A graph neural network for superpixel image classification. In *Journal of Physics: Conference Series*, volume 1871, page 012071. IOP Publishing, 2021.

[238] Yanhu Chen, Cen Wang, Hongxiang Guo, et al. Novel architecture of parameterized quantum circuit for graph convolutional network. *arXiv preprint arXiv:2203.03251*, 2022.

[239] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*, 2017.

[240] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.

[241] Asier Ozaeta, Wim van Dam, and Peter L McMahon. Expectation values from the single-layer quantum approximate optimization algorithm on ising problems. *Quantum Science and Technology*, 2022.

[242] Martin Larocca, Piotr Czarnik, Kunal Sharma, Gopikrishnan Muraleedharan, Patrick J Coles, and M Cerezo. Diagnosing barren plateaus with tools from quantum optimal control. *Quantum*, 6:824, 2022.

[243] Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Learning with invariances in random features and kernel models. In *Conference on Learning Theory*, pages 3351–3418. PMLR, 2021.

[244] Matthias C Caro, Hsin-Yuan Huang, Marco Cerezo, Kunal Sharma, Andrew Sornborger, Lukasz Cincio, and Patrick J Coles. Generalization in quantum machine learning from few training data. *Nature communications*, 13(1):1–11, 2022.

[245] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[246] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015.

[247] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

[248] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:1802.04240*, 2018.

[249] Kunal Marwaha. Local classical max-cut algorithm outperforms $p = 2$ qaoa on high-girth regular graphs. *Quantum*, 5:437, 2021.

[250] Mario Szegedy. What do qaoa energies reveal about graphs? *arXiv preprint arXiv:1912.12277*, 2019.

[251] Gilbert H Harman, Sanjeev R Kulkarni, and Hariharan Narayanan. $\sin(\omega x)$ can approximate almost every finite set of samples. *Constructive Approximation*, 42(2):303–311, 2015.

[252] Mauro ES Morales, Jacob D Biamonte, and Zoltán Zimborás. On the universality of the quantum approximate optimization algorithm. *Quantum Information Processing*, 19(9):1–26, 2020.

[253] Python-TSP.

[254] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[255] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D Lukin. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *Physical Review X*, 11(2):021067, 2020.

[256] Adam Glos, Alexandra Krawiec, and Zoltán Zimborás. Space-efficient binary optimization for variational computing. *arXiv preprint arXiv:2009.07309*, 2020.

[257] Edward Farhi, David Gamarnik, and Sam Gutmann. The quantum approximate optimization algorithm needs to see the whole graph: A typical case. *arXiv preprint arXiv:2004.09002*, 2020.

[258] Sergey Bravyi, Alexander Kliesch, Robert Koenig, and Eugene Tang. Obstacles to variational quantum optimization from symmetry protection. *Physical review letters*, 125(26):260505, 2020.

[259] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D Lukin. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *Physical Review X*, 10(2):021067, 2020.

[260] Ruslan Shaydulin, Phillip C Lotshaw, Jeffrey Larson, James Ostrowski, and Travis S Humble. Parameter transfer for quantum approximate optimization of weighted maxcut. *arXiv preprint arXiv:2201.11785*, 2022.

[261] Laura Gentini, Alessandro Cuccoli, Stefano Pirandola, Paola Verrucchi, and Leonardo Banchi. Noise-resilient variational hybrid quantum-classical optimization. *Physical Review A*, 102(5):052414, 2020.

[262] Kam-Chuen Jim, C Lee Giles, and Bill G Horne. An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on neural networks*, 7(6):1424–1438, 1996.

[263] Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. Regularizing deep neural networks by noise: Its interpretation and optimization. *Advances in Neural Information Processing Systems*, 30, 2017.

[264] Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

[265] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[266] Emilio Rafael Balda, Arash Behboodi, and Rudolf Mathar. Adversarial examples in deep neural networks: An overview. *Deep Learning: Algorithms and Applications*, pages 31–65, 2020.

[267] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[268] Justin Gilmer, Nicolas Ford, Nicholas Carlini, and Ekin Cubuk. Adversarial examples are a natural consequence of test error in noise. In *International Conference on Machine Learning*, pages 2280–2289. PMLR, 2019.

[269] Florian Jaeckle and M Pawan Kumar. Generating adversarial examples with graph neural networks. In *Uncertainty in Artificial Intelligence*, pages 1556–1564. PMLR, 2021.

[270] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[271] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[272] Jinfeng Zeng, Zipeng Wu, Chenfeng Cao, Chao Zhang, Shi-Yao Hou, Pengxiang Xu, and Bei Zeng. Simulating noisy variational quantum eigensolver with local noise models. *Quantum Engineering*, 3(4):e77, 2021.

[273] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh. Analysis of quantum approximate optimization algorithm under realistic noise in superconducting qubits. *arXiv preprint arXiv:1907.09631*, 2019.

[274] Junyu Liu, Frederik Wilde, Antonio Anna Mele, Liang Jiang, and Jens Eisert. Noise can be helpful for variational quantum algorithms. *arXiv preprint arXiv:2210.06723*, 2022.

[275] Jingkang Wang, Yang Liu, and Bo Li. Reinforcement learning with perturbed rewards. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 6202–6209, 2020.

[276] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[277] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452*, 2017.

[278] Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.

[279] Owen Lockwood and Mei Si. Playing atari with hybrid quantum-classical reinforcement learning. In *NeurIPS 2020 Workshop on Pre-registration in Machine Learning*, pages 285–301. PMLR, 2021.

[280] Kosuke Ito, Wataru Mizukami, and Keisuke Fujii. Universal noise-precision relations in variational quantum algorithms. *arXiv preprint arXiv:2106.03390*, 2021.

[281] Andrea Skolik and Stefano Mangini. Code that was used for training of noisy quantum agents. https://github.com/askolik/noisy_qrl, 2022.

[282] OpenAI. Openai gym. https://github.com/openai/gym/wiki, 2022. Accessed: 06-09-2022.

[283] Tensorflow quantum rl tutorial. https://www.tensorflow.org/quantum/tutorials/quantum_reinforcement_learning. Accessed: 06-09-2022.

[284] Aleksandrs Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[285] Tze Leung Lai, Herbert Robbins, et al. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[286] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[287] Zhenyu Cai, Xiaosi Xu, and Simon C Benjamin. Mitigating coherent noise using pauli conjugation. *npj Quantum Information*, 6(1):1–9, 2020.

[288] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *Nature Physics*, 16(10):1050–1057, 2020.

[289] Zbigniew Puchała and Jarosław Adam Miszczak. Symbolic integration with respect to the haar measure on the unitary group. *arXiv preprint arXiv:1109.4244*, 2011.

[290] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[291] Google. Documentation of depolarizing channel in cirq. https://quantumai.google/reference/python/cirq/depolarize, 2022.

[292] Sergei V. Isakov, Dvir Kafri, Orion Martin, Catherine Vollgraff Heidweiller, Wojciech Mruczkiewicz, Matthew P. Harrigan, Nicholas C. Rubin, Ross Thomson, Michael Broughton, Kevin Kissell, Evan Peters, Erik Gustafson, Andy C. Y. Li, Henry Lamm, Gabriel Perdue, Alan K. Ho, Doug Strain, and Sergio Boixo. Simulations of quantum circuits with approximate noise using qsim and cirq, 2021.

[293] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge, UK, 2010.

[294] Timothy Proctor, Stefan Seritan, Kenneth Rudinger, Erik Nielsen, Robin Blume-Kohout, and Kevin Young. Scalable randomized benchmarking of quantum computers using mirror circuits. *Physical Review Letters*, 129(15):150502, 2022.

[295] Joseph Vovrosh, Kiran E Khosla, Sean Greenaway, Christopher Self, Myungshik S Kim, and Johannes Knolle. Simple mitigation of global depolarizing errors in quantum simulations. *Physical Review E*, 104(3):035309, 2021.

[296] Easwar Magesan, Jay M Gambetta, and Joseph Emerson. Characterizing quantum gates via randomized benchmarking. *Physical Review A*, 85(4):042311, 2012.

[297] C Ryan-Anderson, NC Brown, MS Allman, B Arkin, G Asa-Attuah, C Baldwin, J Berg, JG Bohnet, S Braxton, N Burdick, et al. Implementing fault-tolerant entangling gates on the five-qubit code and the color code. *arXiv preprint arXiv:2208.01863*, 2022.

[298] IBM. Ibmquantum. https://quantum-computing.ibm.com/, 2022.

[299] Elijah Pelofske, Andreas Bärtschi, and Stephan Eidenbenz. Quantum volume in practice: What users can expect from nisq devices. *arXiv preprint arXiv:2203.03816*, 2022.

[300] IBM Quantum Experience. IBM Quantum Experience. https://quantum-computing.ibm.com/services/resources?tab=systems, 2022.

[301] Xun Gao and Luming Duan. Efficient classical simulation of noisy quantum computation. *arXiv preprint arXiv:1810.03176*, 2018.

[302] Ryan LaRose, Andrea Mari, Sarah Kaiser, Peter J Karalekas, Andre A Alves, Piotr Czarnik, Mohamed El Mandouh, Max H Gordon, Yousef Hindy, Aaron Robertson, et al. Mitiq: A software package for error mitigation on noisy quantum computers. *Quantum*, 6:774, 2022.

[303] Vincent Russo, Andrea Mari, Nathan Shammah, Ryan LaRose, and William J Zeng. Testing platform-independent quantum error mitigation on noisy quantum computers. *arXiv preprint arXiv:2210.07194*, 2022.

[304] Samson Wang, Piotr Czarnik, Andrew Arrasmith, Marco Cerezo, Lukasz Cincio, and Patrick J Coles. Can error mitigation improve trainability of noisy variational quantum algorithms? *arXiv preprint arXiv:2109.01051*, 2021.

[305] Gian-Carlo Wick. The evaluation of the collision matrix. *Physical review*, 80(2):268, 1950.

[306] Patrick Huembeli and Alexandre Dauphin. Characterizing the loss landscape of variational quantum circuits. *Quantum Science and Technology*, 6(2):025011, 2021.

[307] Motohisa Fukuda, Robert König, and Ion Nechita. Rtni—a symbolic integrator for haar-random tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 52(42):425303, 2019.

[308] Robert W. Keener. *Theoretical Statistics: Topics for a Core Course*. Springer Texts in Statistics. Springer, 1 edition, 2010.

Summary

Variational quantum machine learning models are often described as the quantum analog of classical neural networks due to the similarity in their training procedure, and are therefore also referred to as quantum neural networks. Unlike for their classical counterparts however, there are still numerous open questions about how to design trainable and performant quantum neural networks. Examples of this include the questions of how to encode classical data into a quantum model, how to structure the gates in the circuits that are used to implement models, and how to avoid pitfalls in the trainability of these models that are unique to the quantum setting. Assuming that similarly to the history of classical machine learning, the development of more performant quantum hardware will facilitate large-scale empirical studies on the usefulness of variational quantum machine learning, it is of key importance to build an understanding of how these models can be trained successfully. This thesis aims to contribute to this understanding by studying various aspects of training variational quantum machine learning models.

We start by giving a basic introduction to the topics of quantum computing, machine learning, and their intersection in Chapters 2 and 3, respectively. In Chapter 4, we study how a fundamental issue in the training of variational quantum circuits, namely barren plateaus in the training landscapes, can be addressed by the classical training algorithm to aid scaling up the size of quantum models. To this end, we provide a training scheme that alleviates the problem of barren plateaus for specific cases and compare it to standard training procedures in the existing literature. While this type of training procedure can in principle be used for arbitrary types of machine learning, we focus our attention on a specific type of learning in subsequent chapters, namely on RL. First, we study in Chapter 5 how the architectural choices made for a PQC-based quantum agent influence its performance on two classical benchmark tasks from RL literature, where we

specifically consider the question of encoding data into, and reading information out of the quantum model. In addition, we establish a theoretical separation between classical and quantum models for the specific type of RL algorithm that we use, and also perform an in-depth empirical comparison of the quantum model developed in our work to a classical neural network that performs the same task. In addition to the questions of how to encode data and read out information from a PQC, the third key question in the performance of a variational quantum machine learning model is how to design the structure of the circuit itself, also referred to as the ansatz. For this reason, we move on to study this question in Chapter 6 and introduce an ansatz that is tailored to a specific type of input data, namely to weighted graphs. To do this, we take inspiration from the classical field of geometric deep learning, and design a PQC that preserves an important symmetry in graph-based input data. We analytically study the expressivity of this type of circuit, and then go on to numerically compare it to ansatzes that are not tailored to the specific training data at hand. Finally, another important consideration in the study of algorithms for the NISQ era is how the given learning algorithms and models are influenced by quantum hardware-induced noise. In Chapter 7, we study this for two of the variational RL paradigms from recent literature. We investigate analytically and numerically how various types of errors, namely coherent, incoherent, and measurement-based errors, affect the training performance of variational RL algorithms and the robustness of the learned policies. In particular, this study includes an evaluation of the performance of the models we introduced in Chapter 5 and Chapter 6 under various types of noise that are expected to be present on near-term hardware.

With the above, this thesis aims to contribute to building a foundation of knowledge about how to successfully train variational quantum machine learning models, in the hope that similarly to classical machine learning, this knowledge will one day, when quantum hardware has sufficiently matured, aid demonstrations of the practical usefulness of these types of algorithms.

Samenvatting

Variational quantum machine learning modellen worden vaak gezien als het quantum analoog van klassieke neurale netwerken vanwege de gelijkenis in hun trainingsprocedure. Ze worden daarom ook wel quantum neurale netwerken genoemd. In tegenstelling tot hun klassieke tegenhangers zijn er echter nog veel open vragen over hoe trainbare en performante quantum neurale netwerken ontwerpen kunnen worden. Bijvoorbeeld, hoe kan je standaard data in een quantum model representeren; hoe moeten operaties in de circuits gestructureerd worden; of hoe kunnen optimaliseringsprobelem, die specifiek voor quantum circuits zijn, verhinderd worden. Net als klassieke hardware een game changer was voor de ontwikkeling van machine learning, kan een verbetering van quantum hardware een grote invloed hebben op de ontwikkelen van variational quantum machine learning modellen. Het is daarom van groot belang een begrip te ontwikkelen van hoe deze modellen met succes getrained kunnen worden. Dit proefschrift draagt bij aan dit begrip door zulke aspecten van variational quantum machine learning modellen te bestuderen.

We beginnen met een inleiding tot de quantum computing, machine learning en hun interactie in hoofdstukken 2 en 3. In hoofdstuk 4 bestuderen we hoe een fundamenteel probleem in het trainen van variational quantum circuits, namelijk het ontstaan van „barren plateaus" in de traininglandschappen, kan worden aangepakt door een bestaande leermethoden, die dan tot een vergroting van quantum modellen gebruikt kan worden. Hiertoe bieden we een trainingsprocedure dat het probleem van „barren plateaus" voor specifieke gevallen verlicht en vergelijken we het met standaard trainingsprocedures uit de literatuur.

Hoewel dit type trainingsprocedures in principe voor alle soorten machine learning kan worden gebruikt, richten we onze aandacht in de volgende hoofdstukken specifiek op een bepaald type, namelijk op reinforcement learning (RL). Allereerst

bestuderen we in hoofdstuk 5 hoe de architecturale keuzes voor een quantumagent gebaseerd op een parameterized quantum circuit (PQC) de prestaties beïnvloeden in twee benchmarks uit de RL-literatuur, waarbij we specifiek kijken naar de vraag hoe gegevens te coderen zijn in, en informatie te lezen is uit het quantummodel. Daarnaast stellen we een theoretische scheiding vast tussen klassieke- en quantummodellen voor het specifieke type RL-algoritme dat we gebruiken, en vergelijken het quantummodel empirisch met een klassiek neuraal netwerk. Naast de vragen hoe gegevens te coderen en te lezen uit een PQC, is de derde belangrijke vraag hoe de structuur van een variational quantum machine learning model ontworpen moet worden. Daarvoor gaan we in hoofdstuk 6 verder met het onderzoeken van deze vraag en introduceren we een structuur die is aangepast aan een specifiek type input, namelijk aan gewogen grafen. Hiervoor nemen we inspiratie uit het gebied van geometrical deep learning, en ontwerpen we een PQC dat een belangrijke symmetrie behoudt in graaf-gebaseerde input. We bestuderen de expressiviteit van dit type circuit analytisch, en vergelijken het daarna numeriek met structuren die niet aangepast zijn aan de specifieke input. Ten slotte is een andere belangrijk overweging bij het bestuderen van algoritmen voor het NISQ-tijdperk hoe de gegeven trainingsprocedures en modellen worden beïnvloed door de ruis veroorzaakt door de quantumhardware. In hoodstuk 7 bestuderen we dit voor twee van de variational RL-paradigma's uit recente literatuur. We onderzoeken analytisch en numeriek hoe verschillende soorten fouten, namelijk coherente, incoherente en op metingen gebaseerde fouten, de trainingsprestaties van variational RL-algoritmen en de robuustheid van de geleerde strategie beïnvloeden. In het bijzonder bevat dit onderzoek een evaluatie van de prestaties van de modellen die we in hoodstuk 5 en hoodstuk 6 hebben ingevoerd onder verschillende soorten ruis die verwacht worden aanwezig te zijn op quantumhardware.

Met bovenstaande hoopt dit proefschrift bij te dragen aan het opbouwen van een basis van kennis over hoe variational quantum machine learning modellen met succes opgeleid kunnen worden, in de hoop dat, zoals bij klassieke machine learning, deze kennis ooit, als de quantumhardware voldoende performant is geworden, zal helpen bij de praktische bruikbaarheid van dit soorten algoritmen.

About the author

Andrea Skolik received a Bachelors degree in Computer Science at the Hochschule der Medien in Stuttgart in 2011, and her Masters degree in Computer Science at Ulm University in Germany in 2015, with a focus on machine learning and robotics. In between and after graduating, Andrea worked as a Software Engineer in various areas such as logistics and finance, before joining the quantum computing team at Volkswagen in 2018 to pursue reasearch in this field. She started her studies as a PhD student at Leiden University in 2020 under the supervision of Vedran Dunjko and Thomas Bäck, with a focus on the question of how quantum computers can be used in conjunction with machine learning. Since 2022, Andrea is a full-time quantum computing researcher at Volkswagen, where she continues to work at the intersection of quantum computing and machine learning.