



universität
uulm

Ulm University | 89069 Ulm | Germany

**Faculty of Engineering, Computer Science and
Psychology**
Neural Information Processing

Reinforcement learning with variational quantum algorithms for trajectory planning

Master thesis at Ulm University and the German Aerospace Center (DLR)

Submitted by:

M. Lautaro Hickmann
lautaro.hickmann@uni-ulm.de
953591

Reviewer:

Prof. Dr. Friedhelm Schwenker
Prof. Dr. Frank Köster

Advisers:

Dr. Hans-Martin Rieser
Dr. Bogusz Bujnowski

2022

Version from May 30, 2022

© 2022 Manuel Lautaro Hickmann

Satz: PDF- \LaTeX 2 ϵ

Abstract

English version

The goal of this thesis is to explore Reinforcement Learning (RL) with Variational Quantum Circuits (VQCs) with a focus on its applicability to different standard RL problems and lane change manoeuvres. The main aspects investigated covered feasibility, limitations, and possible advantages when comparing quantum enhanced systems with classical systems in RL. We could show that VQCs can solve multiple simple RL environments, achieving results similar to or even better than those of a classical agent. For the more complex lane change manoeuvre we achieved suboptimal results for the current setup with limited hyperparameter search. We further showed that environments with an observation vector size twice as large as previously published can be solved. We also found indications of possible quantum advantages in convergence rate and stability for discrete state-space environments. Furthermore, we also showed that the Q-learning algorithm implemented with a VQC is heavily prone to noise which leads to problems when using Noisy Intermediate-Scale Quantum (NISQ) hardware.

Deutsche Version

Das Ziel dieser Arbeit ist es, Reinforcement Learning (RL) mit Variational Quantum Circuits (VQCs) mit Fokus auf seine Anwendbarkeit auf verschiedene Standard-RL-Probleme und Spurwechselmanöver zu untersuchen. Die untersuchten Hauptaspekte umfassten Machbarkeit, Einschränkungen und mögliche Vorteile beim Vergleich quantenerweiterter Systeme mit klassischen Systemen in RL. Wir konnten zeigen, dass VQCs mehrere einfache RL-Umgebungen lösen können und ähnliche oder sogar bessere Ergebnisse erzielen als ein klassischer Agent. Für das komplexere Spurwechselmanöver erzielten wir suboptimale Ergebnisse für das aktuelle Setup mit eingeschränkter Hyperparametersuche. Wir haben ferner gezeigt, dass Umgebungen mit einer doppelt so großen Beobachtungsvektorgroße wie zuvor veröffentlicht gelöst werden können. Wir fanden auch Hinweise auf mögliche Quantenvorteile in der Konvergenzrate und Stabilität für diskrete Zustandsraumumgebungen. Darüber hinaus haben wir auch gezeigt, dass der mit einem VQC implementierte Q-Learning-Algorithmus stark rausch anfällig ist, was zu Problemen bei der Verwendung von Noisy Intermediate-Scale Quantum (NISQ)-Hardware führt.

Acknowledgement

I wish to express my sincere thanks to Prof. Dr. Friedhelm Schwenker and Prof. Dr. Frank Köster, for sharing expertise, guidance, and encouragement which helped me accomplish this thesis. Additionally, I am extremely grateful to Dr. Hans-Martin Rieser and Dr. Bogusz Bujnowski for their guidance, assistance, and dedicated involvement in every step throughout the process of making this thesis. Thanks should also go to Matthias Nichting for his support with the ADORe framework.

I also thank my parents for their constant encouragement, support, and attention.

Contents

1	Introduction	1
1.1	On Quantum Machine Learning	1
1.2	Quantum Hype	1
1.3	Problem statement and Contributions	2
1.4	Structure of the thesis	3
2	Theory Fundamentals	4
2.1	Machine Learning	4
2.1.1	Principles of Machine Learning	4
2.1.2	Deep Feedforward neural networks	5
2.1.3	Training	5
2.1.4	Optimizers	8
2.1.5	Batch and mini-batch training	8
2.2	Reinforcement Learning	8
2.2.1	Learning from interactions	8
2.2.2	The environment	9
2.2.3	Reward	10
2.2.4	Value-Based and Policy-Based learning	10
2.2.5	Q-learning	11
2.3	Connection to Quantum Computing	12
2.3.1	Postulates of Quantum Mechanic	12
2.3.2	Quantum Computing	14
2.4	Quantum Machine Learning (QML)	20
2.4.1	Ideas of Quantum Machine Learning	20
2.4.2	Input Encoding	21
2.4.3	Variational Quantum Circuits (VQCs)	23
3	A simulation study of Quantum Reinforcement Learning	28
3.1	Outline	28
3.1.1	Motivation	28
3.1.2	Related work	28
3.2	Proposed Implementation and Frameworks	29
3.2.1	Proposed Reinforcement Learning (RL) algorithm	29
3.2.2	Proposed VQC implementation	31
3.2.3	Hypotheses	33
3.2.4	Selected scenarios	34
3.2.5	Quantum vs Classic on Cart Pole	36

3.2.6	Quantum vs Classic on FrozenLake	37
3.2.7	Quantum scalability	38
3.2.8	Effects of noise	39
3.2.9	Lane change	40
4	Simulation Results	41
4.1	Results	41
4.1.1	Quantum vs Classic on CartPole	41
4.1.2	Quantum vs Classical on Frozen Lake	46
4.1.3	Quantum scalability	49
4.1.4	Effects of noise	51
4.1.5	Lane change	53
4.2	Discussion	54
4.2.1	Quantum vs Classic on Cart Pole	54
4.2.2	Quantum vs Classic on FrozenLake	57
4.2.3	Quantum Scalability	58
4.2.4	Effects of noise	59
4.2.5	Lane change	61
5	Conclusion	63
	Bibliography	66
A	Supplementary material for the experiments	72
A.1	Grid World like environments	72
A.2	Hyperparameter searches	72
A.2.1	Setup and objectives	72
A.2.2	Search spaces and selected hyperparameters	74
B	Supplementary material for the results	77
B.1	Quantum vs Classic on CartPole	77
B.2	Quantum vs Classic on Frozen Lake	77
B.3	Quantum scalability	78
B.4	Effects of noise	78

1 Introduction

1.1 On Quantum Machine Learning

In recent years, the field of Machine Learning (ML) has achieved many successes and has grown exponentially in popularity. More recently, with the advent of the first publicly available quantum computers, quantum computing has emerged and gained rapidly in popularity. By synergistically combining ML and quantum computing approaches, the new field of Quantum Machine Learning (QML) has emerged. This field promises to improve ML and open new learning possibilities.

In this work, we investigate the advantages and challenges of combining ML, the discipline of making computers solve problems by learning from data rather than explicitly coding the solution of the problem [1], with quantum computing. Quantum computing describes how to process information with devices based on the laws of quantum theory. We focus on the subfield of Reinforcement Learning (RL) that aims to solve an agent's task how to behave in an environment without a fixed training dataset. This combination is called Quantum Reinforcement Learning (QRL), a subfield of QML, and is expected to produce quantum advantages in the form of faster convergence or improved learning capabilities. We focus on a feasibility study and a constrained comparison with selected classical methods.

RL with Variational Quantum Circuits (VQCs) is in an early stage of development. Deep Q-learning algorithms have been shown to run on VQCs and have been tested on simple environments [2, 3, 4]. We investigated empirically the feasibility of QRL solving complex problems, expanding the understanding of what VQCs can solve. Furthermore, most related work only investigates the perfect case of noise-free quantum circuits. But, most current quantum computers are so-called Noisy Intermediate-Scale Quantum (NISQ) devices, with low number of qubits that do not necessarily interact with each other, have no error correction, and therefore produce only approximate results of the computations. We investigate how VQCs behave with respect to noise, to investigate if QRL can also be carried out on current real quantum hardware.

1.2 Quantum Hype

One reason for the current hype of QML is that optimisation problems, one of the mathematical core problems of ML, are seen as promising candidates where quantum computing could show significant improvements [5]. Furthermore, one big factor pushing the hype are promises of quantum speed-ups. A quantum speed-up is when a quantum computer can theoretically solve a computational problem faster, in the asymptotic runtime, than a classical computer. Another promise, especially for QML is quantum advantages in the amount of training data needed to solve a given task and faster algorithm convergence.

One problem of the hype is stated by Schuld and Petruccione: “[...] the sheer expense involved in the development of quantum hardware creates a strong incentive for research to motivate their studies with arguments along the lines

of superior quantum algorithms which led to the controversial term quantum supremacy for experiments which demonstrate a provable classical-quantum separation in computational complexity—however artificial the problem may be." [6]. This means that many investigations are portrayed in a manner in which a quantum advantage can be shown, but they have limited practical applications since the problems have been artificially constructed exactly in a way to produce quantum advantages.

Furthermore, current QML algorithms are subject to strict design limitations in order to perform even the smallest empirical benchmarks on simulators and NISQ hardware. Larger circuits cannot be simulated on classical hardware, and quickly become drowned in noise on real quantum devices. Thus, the algorithms must be small and use only a few qubits and gates. Additionally, quantum speed-ups are not as beneficial for QML as to quantum computing in general, since the important problem is the question what can be learnt. If something has been proven to be learnable, quantum speed-ups come into play to achieve faster runtimes.

To achieve a QML approach that is useful in practise, the following challenges must be overcome:

- The hype leads to trying to find fast improvements, by taking classical algorithms and porting some part of the computations to a quantum machine. However, this severely limits the potential of quantum computations. Instead, a slower approach of designing a quantum-classical hybrid algorithm bottom up could deliver better results.
- Most proposed algorithms are built on noise-free simulations, but real hardware is noisy. Thus, noise robust algorithms must be developed.
- Further hardware and software developments are needed to be able to run circuits large enough to solve most praxis relevant problems.
- A theoretical framework of QML is needed, to better achieve classical-quantum interfaces and especially how to encode classical data into quantum circuits efficiently. Such a framework would also be helpful to improve the explainability of QML algorithms.

However, the quantum hype enables the development of the quantum computing field and the QML field, opening the possibility of achieving significant breakthroughs on these challenges.

1.3 Problem statement and Contributions

The purpose of the thesis is to explore RL with VQCs with a focus on its applicability to different standard problems and lane change manoeuvres. The main results have been achieved on assessing feasibility, limitations, and possible advantages when comparing quantum enhanced systems with classical systems in RL. Furthermore, the effect of noisy circuits on RL tasks will be investigated.

We showed that a VQC can solve RL environments with continuous or discrete state spaces, discrete action spaces, and immediate or delayed reward. Furthermore, the behavior with respect to the observation vector size was investigated, showing that environments more than two times larger than previously tried are solvable. Achieving results similar to or even better than those of a classical agent. The results for discrete state-space environments hint at possible quantum advantages in the form of faster convergence. Additionally, we showed that the Q-learning algorithm implemented with a VQC is noise prone, and this could lead to problems when using NISQ hardware. We also show that we can solve the lane change environment, but achieve sub-optimal results compared to the classical agent. This should improve with a larger hyperparameter search.

1.4 Structure of the thesis

In Chapter 2, the theoretical fundamentals needed to understand this thesis are presented, starting by introducing the concepts of ML in Section 2.1 and RL in Section 2.2. Followed by a excerpt into quantum computing in Section 2.3 and its applicability to QML in Section 2.4. The methods implemented and the experiments performed in this thesis are then described in Chapter 3, after which the results are presented and discussed in Chapter 4. Finally, Chapter 5 covers the main conclusions and identifies both limitations of the experiments and recommendations for future research.

2 Theory Fundamentals

2.1 Machine Learning

2.1.1 Principles of Machine Learning

In this thesis, we focus on sub-symbolic Artificial Intelligence (AI) approaches, such as Machine Learning (ML) and Quantum Machine Learning (QML). This kind of approaches provide associative results by learning from large datasets, i.e. these methods establish correlations between input and output variables through learning of data without human intervention [7]. In contrast, symbolic AI methods refer to human-readable and explainable processes [7]. These methods are usually used for knowledge deduction, and are defined by the usage of symbolic techniques as are formal methods and programming languages.

Among the most common uses of sub-symbolic methods are prediction, clustering, pattern classification, object recognition, and natural language processing (NLP) tasks. Additionally, speech and text recognition, classification and categorization are also applications of sub-symbolic methods. A disadvantage of sub-symbolic methods is their lack of explainability.

ML is the data driven intersection of computer science, statistics, and mathematics. ML algorithms can be divided into three main categories: unsupervised, supervised and Reinforcement Learning (RL). Each category is characterized by the kind of input data it uses and how the data is generated.

- **Supervised learning:** The training data consist of input and target pairs. The targets are also called labels. The goal of the algorithm is to reproduce the targets from the inputs. Usually the targets are the results of applying a function to the inputs, and learning said unknown function is the task at hand.
- **Unsupervised learning:** The training data consist only of inputs without targets. The goal of the algorithm is to extract some information about the dataset, as in clustering [8] or dimensionality reduction [9].
- **RL:** There is no predefined dataset, but an agent generates a dataset from interactions with an environment. An interaction consists of the agent taking an action and receiving a response/observation from the environment. The agent has to learn an optimal interaction strategy by trial and error. The agent gets rewarded for its actions according to predefined environment rules. This work uses the RL machine learning paradigm which will be explained in more detail in Section 2.2.

A major source of difficulty in many real-world AI applications, is that many variable factors influence observable data [10]. For example, the colour of objects changes depending on lighting, dust on a camera produces a non-standard image or sensors produce noisy readouts. Therefore, most AI applications require the separation and filtering of variation factors. This is done so that only factors beneficial to achieving a given task are used. Often, discriminating these high-level, abstract features from raw data is very complicated, and a nearly human-level understanding of the data is required. In the case when obtaining a representation is as difficult as solving the original problem, learning a representation of the data does not appear helpful.

Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations [10]. Deep learning enables the systems to build complex concepts out of simpler concepts. It can be regarded as the study of models that involve a higher amount of composition of either learned functions or learned concepts than traditional machine learning does. Deep learning methods are therefore formed by the composition of multiple non-linear transformations, with the goal of yielding more abstract – and ultimately more useful – representations [11]. Nevertheless, deep learning is somewhat like a “black-box” and so far, there is no strict theoretical system to support it. As mentioned by Wang, Yao, and Zhao: “we have impressive performance using deep learning but we do not know why theoretically” [12].

Deep learning algorithms achieve state-of-the-art results and received much attention in the machine learning and applied statistics literature in recent years. Such algorithms consist of multiple layers of representation, where the greatest advantage over ML is that the features of each layer are not designed manually but are learned from the input data automatically [13].

Training of multi-layered networks was unsuccessful until 2006, when breakthroughs were made by [14, 15, 16] among others. Previously, only convolutional neural networks (CNN) were trained successfully. To achieve said breakthrough in multi-layered network training, a greedy layer-wise pre-training was proposed to initialize the weights of an entire network, in an unsupervised manner. After finalizing pre-training, the entire network is trained in a supervised fashion using back propagation.

2.1.2 Deep Feedforward neural networks

Deep Feedforward Neural Networks (DNNs), also called feed forward networks, or Multilayer Perceptrons (MLPs), are the quintessential deep learning models [10]. The goal is to approximate a function f^* by defining a mapping $y = f(x, \theta)$ for an input x and learning a set of parameters θ . This structure is called feed forward neural networks, since the information derived from the input is propagated forwards.

Feedforward neural networks are called networks because they are typically represented by composing together multiple different functions. The model is associated with a directed acyclic graph describing how the functions are composed together [10].

Most DNNs are organized as chained layers, with each layer $\mathbf{h}^{(i)}$ being a function of the precedent layer,

$$\mathbf{h}^{(1)} = g^{(1)} \left(\mathbf{W}^{(1)\top} \mathbf{x} + \mathbf{b}^{(1)} \right), \quad (2.1)$$

$$\mathbf{h}^{(i)} = g^{(i)} \left(\mathbf{W}^{(i)\top} \mathbf{h}^{(i-1)} + \mathbf{b}^{(i)} \right). \quad (2.2)$$

Here $\mathbf{h}^{(1)}$ is the initial layer, \mathbf{x} the input data point, $g^{(i)}$ is the activation function of the i -th layer (see Section 2.1.3) and $\mathbf{W}^{(i)\top}$ and $\mathbf{b}^{(i)}$ are the weights and bias of the layer i , respectively. The activation function of the last layer is often called the output function and denoted as $y = f(x, \theta) = \mathbf{h}^{(n)}$ for a n -layer network. Here θ represents the set of the weights and biases for all layers.

2.1.3 Training

The central challenge in machine learning is that an algorithm must perform well on new, previously unseen inputs, not just on the ones the model was trained on. This ability is called generalisation.

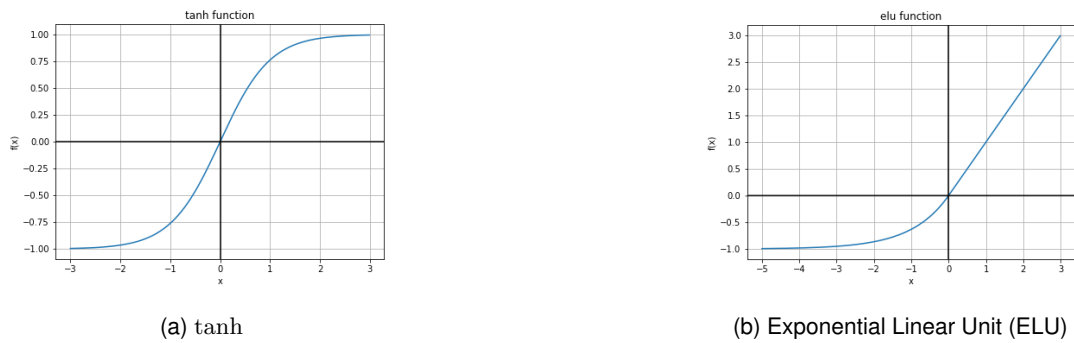


Figure 2.1: Activation functions

Typically, a machine learning model is trained on a training set of n input-target pairs $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$. From this set an error measurement, the so called training error, can be computed. Minimizing this error improves the models ability to match the desired input-output relationship. Up to this point this minimization process is only an optimisation task. The difference between machine learning and optimisation is the goal of also minimizing the generalisation error, that is defined as the expected value of the error on input that the model was not trained on. The expectation is taken across different possible inputs, drawn from the distribution of the expected inputs the system could encounter in practice. The generalization error is not used for training but only as an indication of how good the training is [10].

In almost all instances of deep learning, the objective function is a highly non-convex function of the parameters with potentially many distinct local minima in the model parameter space. The principal difficulty is that not all of these minima provide equivalent generalisation errors.

DNN are usually trained by optimizing the training error by stochastic gradient descent and backpropagation. The backpropagation algorithm [17] calculates the gradients of the network output with respect to its trainable parameters and propagates the partial derivatives in the opposite direction of the information propagation. This process is used to update the parameters of the network.

The two central challenges in machine learning are underfitting and overfitting [10].

- Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set.
- Overfitting occurs when the gap between the training error and test error is too large.

Whether a model is more likely to overfit or underfit can be controlled by altering its ability to fit a wide variety of functions [10]. Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties specific to the training set that do not serve well on the test set.

Activation functions

Activation functions introduce non-linear properties to the network and thus influence the expression capacity of a network [10]. In the following we list two common activation functions that are used in this work.

Hyperbolic Tangent (\tanh) is a nonlinear activation function that maps a real-valued number to the range $[-1, 1]$ as shown in Figure 2.1a. It offers the advantage that negative inputs will be mapped strongly negative, and the zero

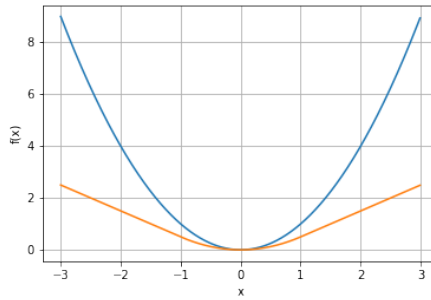


Figure 2.2: Loss functions as $Loss(x)$ with x representing the difference between target and prediction. In blue the Medium Squared Error (MSE) and in orange the Huber loss with $\delta = 1$

inputs will be mapped to zero. The Hyperbolic Tangent can be expressed in terms of the exponential function,

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (2.3)$$

It is often used as an output layer activation function, also known as output function.

The **Exponential Linear Unit (ELU)** function speeds up learning in deep neural networks and leads to higher classification accuracies [18]. It is defined as

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}. \quad (2.4)$$

with $\alpha > 0$. The form of the activation function is shown in Figure 2.1b. Since the Exponential Linear Unit (ELU) acts as identity for positive values it alleviates the vanishing gradient problem. The parameter α controls the value to which an ELU saturates for negative net inputs. The negativity of the function, and the fact that that the derivatives of ELU are small at negative inputs decreases the variation and the information that is propagated to the next layer [18]. Therefore, the representation is both noise-robust and low-complex.

Loss Functions

Most machine learning algorithms involve optimisation of some sort, i.e. the task of minimizing or maximizing an objective function. When the objective function is minimized it is referred to as a cost function or loss function as it measures the discrepancy between the output of the network and some target. For supervised learning, these targets are the labels. For unsupervised learning, the target can be represented by a condition, or property of the dataset, among many other possibilities.

One way of measuring the performance of the model is to compute the **Mean squared error (MSE)** of the model on a data set, given by:

$$MSE(y, \hat{y}) = \frac{1}{m} \sum_i^m (y_i - \hat{y}_i)^2 \quad (2.5)$$

where \hat{y} denotes the predictions of the model and y the targets of the dataset.

The **Huber** loss interpolates between the MSE and the Mean Absolute Error (MAE) for different error regimes and

is given by

$$\text{Huber}(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^m \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & , \text{ if } |y_i - \hat{y}_i| \leq \delta \\ \delta \cdot (|y_i - \hat{y}_i| - \frac{1}{2}\delta) & , \text{ otherwise} \end{cases} \quad (2.6)$$

When the error is smaller than a threshold δ , the Huber loss equals the MSE whereas it equals the MAE for errors larger than δ . It inherits the advantages of both loss functions as it is less sensitive to outlier error in the linear regime and leads to faster convergence in the quadratic regime [1].

2.1.4 Optimizers

The **Adaptive Movement Estimation (Adam)** optimizer is a method for efficient stochastic optimisation that only requires first-order gradients with little memory requirement [19]. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. It combines the advantages of AdaGrad [20], which works well with sparse gradients, and RMSProp [21], which works well in on-line and non-stationary settings.

An extension to the Adam optimizer is **AMSGrad**. AMSGrad is a stochastic optimization method that seeks to fix a convergence issue with Adam based optimizers by using the maximum of past squared gradients rather than the exponential average to update the parameters [22].

For a detailed algorithm and further explanation, the reader is advised to [19] for Adam and [22] for AMSgrad.

2.1.5 Batch and mini-batch training

There are two main settings to divide the available data for training. When all the data is used to compute the loss and its gradient with respect to the trainable parameters, it is called **batch** gradient descent. The other setting is using only a random subset of the data to compute the loss and its gradient with respect to the trainable parameters, which is called **mini-batch** gradient descent. An epoch of training is defined as the time when the training algorithm iterated over all available training data. How many data points are used is defined by the so called batch size B , usually being greater than one but less than the total dataset size. The impact of choosing B is mostly computational, i.e. larger B yield faster computation but requires visiting more examples in order to reach the same error, since there are fewer updates per epoch. For batch gradient descent only one update is realized per epoch, in contrast for mini-batch gradient descent multiple updates are done (as many as necessary to iterate over all mini-batches).

In theory the batch size should impact training time and not so much test performance. Therefore, it can be optimized separately after the other hyperparameters (except learning rate) have been selected, by comparing training curves (training and validation error vs amount of training time) [23].

2.2 Reinforcement Learning

2.2.1 Learning from interactions

Reinforcement Learning (RL) has been around since the 1950s, producing many interesting applications receiving moderate attention. In 2013 it exploded in popularity when it was demonstrated that a RL system could learn to play

Atari games from scratch [24], eventually outperforming humans in most games [25]. The agents used only raw pixels as input without any prior knowledge of the game rules. After further developments, in 2016 AlphaGo [26] was able to beat the best human players in the complex game of Go. These breakthroughs were achieved by combining the fields of deep learning and RL.

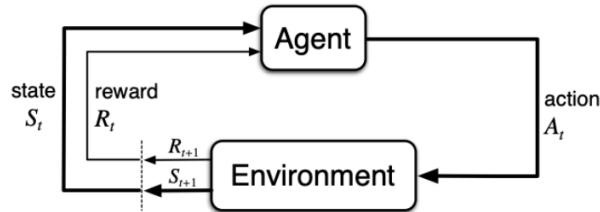


Figure 2.3: Agent-environment interaction scheme. Image reproduced from [27].

Learning from interaction is a fundamental idea underlying most theories of learning and intelligence [27]. RL is a goal oriented trial and error learning method. Where an agent, makes *observations* and takes *actions* in an *environment*, receiving a *reward* from the environment, this is called the RL loop and is shown in Figure 2.3. The agent’s objective is to learn how to behave in a way that will maximize its expected reward over time.

In contrast to other learning methods in RL there is no fixed training dataset, but the agent generates one by interacting with the environment. The data samples are not labelled, but have an associated reward given by the environment. The agent changes its behaviour while training, and continually interacts with the environment, this leads to a constantly changing data set.

2.2.2 The environment

Environments are defined by a state space \mathcal{S} , describing the current state of the environment, and an action space \mathcal{A} , describing which actions an agent can take to alter the state of the environment. State and action spaces can be arbitrarily complex, and can be either discrete or continuous. In a discrete state or action space only a finite set of elements exist, and each variable of the space can only take a number of discrete values. In contrast, in continuous states or action spaces, each variable of the space can take arbitrary many values, normally in a fixed value range. Furthermore, environments can be deterministic by always reacting the same to a given action, or stochastic by behaving probabilistically with respect to the action carried out, i.e. the actions do not always have the same effect.

An agent interacts with an environment at time step t by taking an action a_t in state s_t , and receives a reward r_{t+1} . A tuple of these elements together form a so-called *transition* $(s_t, a_t, r_{t+1}, s_{t+1})$. The transition function $P_{s_t s_{t+1}}^{a_t}$ gives the probabilities of transitioning from state s_t to s_{t+1} when performing a_t in an environment and is given by

$$P_{s_t s_{t+1}}^{a_t} = P(s_{t+1} | s_t, a_t). \tag{2.7}$$

When a state transition happens the agent gets a reward from the environment, if only very few state transitions return a non-zero reward then the environment is said to have sparse reward. On the other hand, if most of the transitions return a non-zero reward, it is denominated immediate reward.

Environments often break down into so-called episodes, these are sequences of interactions that reach a terminal state that reinitiates the environment. The horizon H represents the maximal number of steps that can be taken

until the episode terminates. Environments based on games are prominent examples of episodic environments, with an episode comprising one game played. The agent learns by playing several games one after the other.

2.2.3 Reward

The reward function is used to evaluate the quality of the actions taken by the agent in the environment based on the learning task at hand. The goal of the agent is to maximize the total expected reward over a sequence of times steps, this is called the return and is given by

$$G_t = \sum_{k=0}^H \gamma^k r_{t+k+1}, \quad (2.8)$$

with horizon H and discount factor $\gamma \in [0, 1]$. γ determines the present value of the future reward, i.e. a reward received k time steps in the future is only worth γ^{k-1} times what it would be worth if it were received immediately [27].

A modified version of the expected reward is often used as an evaluation metric, namely the cumulative reward. The cumulative reward of an episode is defined as the sum over all step rewards in an episode, it is also often called the episode score, and is given by

$$C_H = \sum_{t=0}^H r_t \quad (2.9)$$

with horizon H , i.e. there are maximally H steps in the episode [27].

2.2.4 Value-Based and Policy-Based learning

RL approaches can be divided into value-based and policy-based learning methods [27]. Both approaches follow the same goal of maximizing the return, but use different methods for this. In all cases, the function that models what action a_t the agent takes given the environment in state s_t is called the *policy* $\pi(a_t|s_t)$.

In general, the performance of a policy is given by a state value function

$$V_\pi(s) = \mathbb{E}_\pi [G_t | s_t = s] \quad (2.10)$$

which is the expected return starting from state s at an initial time step t and following policy π . The goal of RL algorithms is to learn the policy that maximises the expected return for each starting state, this policy is called the optimal policy π_* .

The main difference between Value-Based and Policy-Based learning methods is how the policy is realized. Value-based algorithms learn a value function which is used to create a policy, by selecting the action which yields the highest $V_\pi(s)$. In contrast, policy-based algorithms try to directly optimized a parametrized probability distribution that represents the policy [3]. Both methods can be combined to leverage the strengths of both approaches in the so called actor-critic setting [28].

In this work we will only focus on value-based algorithms. Especially on the Q-learning algorithm, presented in the next section.

2.2.5 Q-learning

In Q-learning the quality of a policy is given by a modified version of the value function (Equation 2.10), the action-value function

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi} [G_t \mid s_t = s, a_t = a], \quad (2.11)$$

which also describes the expected return when following a policy π , but additionally conditioned on an action a [1]. The optimal Q-function is given by $Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$, and the optimal policy is constructed by always executing the action with the highest Q-value:

$$\pi_*(a \mid s) = \operatorname{argmax}_a Q_*(s, a). \quad (2.12)$$

The goal in Q-learning is to learn an approximation $Q(s, a)$ of the optimal Q-function. Q-learning was originally proposed as a tabular learning algorithm, with a so called Q-table which stores the Q-values for each possible state-action pair [29]. A higher Q-value means a higher expected reward when taking action a in state s_t , thus the agent chooses its next action as follows:

$$a_t = \operatorname{argmax}_a Q(s_t, a). \quad (2.13)$$

For learning, it is important that the agent experiences a variety of transitions to explore the state and action spaces. Using always the argmax policy this would not happen, therefore to ensure enough exploration is carried out a so-called ϵ -greedy policy is used while training in Q-learning. For evaluation the argmax policy is used again. The ϵ -greedy policy choses with probability ϵ a random action and with probability $1 - \epsilon$ the action corresponding to the highest Q-value. To balance exploration and exploitation ϵ is usually decreased during training.

For learning the Q-values are updated with observations made in the environment by the following update rule

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \left[\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{expected optimal future return}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right] \quad (2.14)$$

with learning rate α . The update rule incorporates the direct environment feedback in the form of the reward, and the agent's expected optimal future return. Depending on γ the optimal Q-values for the same environment can take highly varying values. Therefore, they can be viewed as different learning environments [3]. For the tabular case, this update rule is proven to converge to the optimal Q-values in the limit of visiting all state-action pairs [30].

In praxis, it suffices to learn the order of Q-values to solve an environment, since the actions are selected following a maximum criterion (Equation 2.13). Thus, the learning task can be reformulated to learning the correct order of Q-values by observing rewards through interactions with the environment.

For moderate sized state and action spaces the tabular approach quickly becomes intractable. Therefore, the Q-tables are replaced by Q-function approximators, which return the Q-values for each state-action pair but do not store them individually. One such approximators are called deep Q-networks which use Artificial Neural Network (ANN) as the Q-function approximator [25].

In this thesis, we will use DNN and quantum algorithms as Q-value approximators. To this end in the next section quantum computing will be explained.

2.3 Connection to Quantum Computing

This section is a brief summary of the fundamental concepts of quantum mechanics. The focus of this thesis will be on quantum computing and therefore many distracting details will be omitted.

2.3.1 Postulates of Quantum Mechanic

Quantum mechanics describes the behaviour of systems at the scale of molecules, atoms and below. Although it has been experimentally verified to high accuracy, the principles and consequences of quantum mechanics are highly unintuitive. In contrast to classical mechanics, some properties of particles that we experience as continuous in our macroscopic world, can only take discrete values in the quantum world. They appear “quantized” which explains the name of the theory. Quantum mechanics is a probabilistic theory, which means that a possible outcome of a measurement of an observable quantity will not happen with certainty but will appear with some probability. In quantum mechanics all particles have wavelike properties and the equation that governs the evolution of these particle waves is the Schrödinger equation. Thus, it is not surprising that particles on the quantum scale exhibit interference behaviour which was one of the many unexplained phenomena that quantum mechanics was able to successfully describe theoretically [31]. A consequence of the wave-like nature of particles is that in quantum mechanics there exists a fundamental upper limit to measure certain observables simultaneously, which is known as the uncertainty principle. A prominent example of such conjugate variables are the position and the momentum of a particle.

Although many aspects of quantum mechanics might seem confusing to our classically trained eye, the basic rules of the quantum world can be stated in a few postulates. As we are often forced to leave behind our intuition, we can learn about quantum systems by applying the rules and then compare how quantum systems differ from classical systems. We now list the basic postulates and definitions that are necessary to explore the elements of quantum information processing and quantum computation in this thesis.

State space

The state space of a quantum mechanical system is a complex separable vector space, called the Hilbert space. In this work we are only concerned with systems where the dimension of the corresponding Hilbert space is finite, thus \mathcal{H} is isomorphic to \mathbb{C}^k . The elements of \mathcal{H} can be represented as complex-valued vectors $|\psi\rangle$ in the convenient Dirac notation and the corresponding covectors are written as $\langle\psi|$. The inner product on \mathcal{H} between two vectors $|\phi\rangle, |\psi\rangle$ of the Hilbert space is given by $\langle\phi|\psi\rangle$ with $\langle\phi|\psi\rangle^* = \langle\psi|\phi\rangle$, which implies that the norm of a vector is given by $\|\psi\| = \sqrt{\langle\psi|\psi\rangle}$.

For every state $|\psi\rangle \in \mathcal{H}$ there exists a complete orthonormal basis, $\{|e_k\rangle\}, k \in \mathbb{N}$ with $\langle e_i|e_j\rangle = \delta_{ij}$ such that:

$$|\psi\rangle = \sum_k |e_k\rangle \langle e_k|\psi\rangle = \sum_k \langle e_k|\psi\rangle |e_k\rangle, \text{ with } \sum_k |e_k\rangle \langle e_k| = \mathbb{1}. \quad (2.15)$$

Note that $|e_k\rangle \langle e_k|$ denotes a projector on the state $|e_k\rangle$. Every state $|\psi\rangle$ can be expanded in the basis states,

$$|\psi\rangle = \sum_k \alpha_k |e_k\rangle, \quad (2.16)$$

with $\alpha_k = \langle e_k | \psi \rangle \in \mathbb{C}$ and $\sum_k |\alpha_k|^2 = 1$. The weights α_k are called probability amplitudes. The probabilistic nature of quantum mechanics appears in the interpretation of the wave function. The modulus squared of the probability amplitudes represent the probability that the quantum object described by state $|\psi\rangle$ is in basis state k . To guarantee the interpretation as probability we need that $\sum |\alpha_k|^2 = 1$.

Observables

Observables of a quantum mechanical system are represented by Hermitian operators \mathcal{M} on \mathcal{H} . Hermitian operators have the property that they are equal to its complex-conjugate transpose, $\mathcal{M} = (\mathcal{M}^*)^T$. Thus, their eigenvalues must be real quantities and can represent real physical quantities.

Measurement

The possible outcomes of a measurement in quantum mechanics are the eigenvalues of hermitian operators. The eigenstates of a Hermitian operator form an orthonormal basis of the Hilbert space. In its eigenbasis

$$\mathcal{M} = \sum_k \mu_k |\mu_k\rangle \langle \mu_k| = \sum_k \mu_k P_k, \quad (2.17)$$

and is obviously diagonal. Here we introduced P_k as the projector to the eigenspace spanned by the eigenvector $|\mu_k\rangle$. Without going into further details, after measuring the eigenvalue μ_k of the observable \mathcal{M} , the state of the system collapses to the corresponding eigenvector $|\mu_k\rangle$. This can be described by the action of the projector onto the system state:

$$|\psi\rangle \rightarrow \frac{P_k |\psi\rangle}{\sqrt{\langle \psi | P_k | \psi \rangle}}. \quad (2.18)$$

Consequently the expectation value of an observable \mathcal{M} in the state $|\psi\rangle$ is defined as

$$\langle \mathcal{M} \rangle = \langle \psi | \mathcal{M} | \psi \rangle. \quad (2.19)$$

Time Evolution

The time evolution of a quantum mechanical system is described by the Schrödinger equation,

$$i\hbar \frac{d}{dt} |\psi\rangle = H |\psi\rangle, \quad (2.20)$$

where H is a special observable known as the Hamiltonian that represents the energy of a system and \hbar is Planck's constant. Solving the Schrödinger equation for time-independent Hamiltonians gives rise to the time-evolution described by the unitary operator:

$$U(t) = e^{-i\frac{t}{\hbar} H}. \quad (2.21)$$

A unitary operator has the property that its inverse is its complex conjugated, i.e. $U^{-1} = U^\dagger$ and represent norm preserving rotations in the Hilbert space.

Composite Systems

A quantum system Σ made up of multiple subsystems Σ_i , for simplicity here only Σ_1 and Σ_2 , is called a composite System. The states space of each subsystem is a Hilbert space \mathcal{H}_i with $\dim(\mathcal{H}_i) = K_i$, spanned by an orthonormal basis $\{e_j^i\}, j \in \mathbb{N}$. The Hilbert space \mathcal{H} of the total system Σ is given by the tensor product of the subsystems, i.e., $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$. The states in the composite system have $\dim(\mathcal{H}) = K_1 K_2$.

Using the Schmidt decomposition theorem, a generic state $|\psi\rangle$ in the composite system Σ is given by

$$|\psi\rangle = \sum_{j,k} c_{j,k} |f_j^1\rangle \otimes |f_k^2\rangle \quad (2.22)$$

with complex scalars $c_{j,k}$ that fulfil $\sum_{j,k} |c_{j,k}|^2 = 1$ and orthonormal bases $\{|f_j^1\rangle\}$ and $\{|f_k^2\rangle\}$ of \mathcal{H}_1 and \mathcal{H}_2 respectively.

A state $|\psi\rangle$ is called *separable* if it can be expressed as

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \quad (2.23)$$

with $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$. If a state cannot be expressed as above, the state is called *entangled*. Entangled states, where the state of one particle can not be described independently of the state of the other are a unique feature of quantum systems and have no classical analogous.

Open Quantum System

In reality, any finite physical system is always interacting with its environment. In the context of quantum computing this means that quantum computers are constantly subject to external perturbations. To study the effect of noise on the computation process the quantum system and the environment can be described as a composite system where Σ_1 represents the open quantum system of the quantum computer and Σ_2 represents the environment. The coupling with the environment introduces noise into the system. Most of the time it is not possible to treat the coupling between the two systems exactly as the exact state of the environment is unknown and hugely complicated. In this work, we will simulate the effect of noise from the interaction with the environment by using effective operators, that randomly change the state during quantum computation and thus disturb the outcome.

This represents the big challenge of using Noisy Intermediate-Scale Quantum (NISQ) devices. NISQ devices are the current, and first, prototypes of quantum computers. They consist of around 10 – 100 qubits that do not necessarily all interact with each other, have no error correction, and therefore produce only approximate results of computations [6]. Nevertheless, in principle they have the ability to test quantum computing advantages, but the need to limit algorithms to only a few qubits and gates has a profound impact on the design of quantum algorithms.

2.3.2 Quantum Computing

In this section, we link the introduced framework from the previous section to the field of quantum computation. According to Nielsen and Chuang: “Quantum computation and quantum information is the study of the information processing tasks that can be accomplished using quantum mechanical systems” [33]. We start by comparing the architecture differences between classical computers and quantum computers that are an unavoidable consequence when using quantum systems as processing unit.

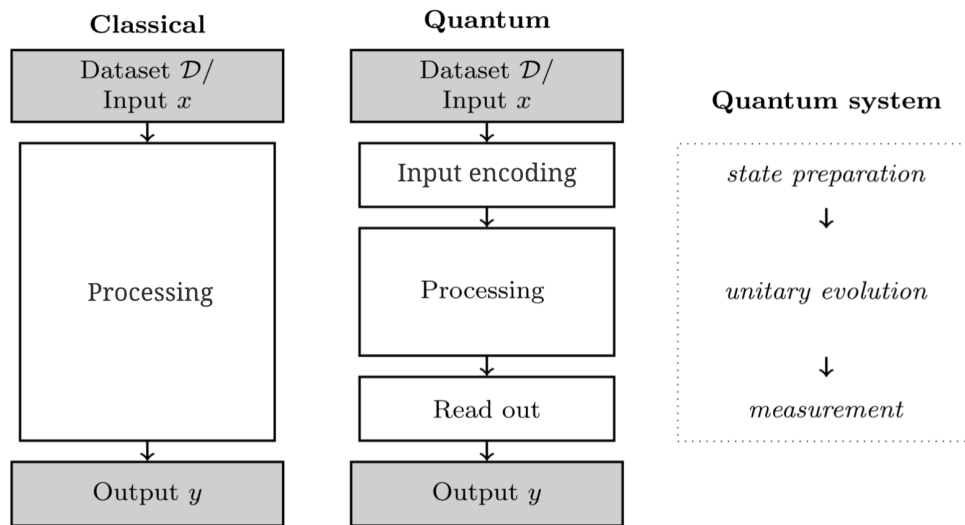


Figure 2.4: Diagram for processing tasks based on classical data. The encoding and readout steps can be highly non-trivial and take considerable runtime. Adapted from [32].

To process classical data, quantum computers require two additional steps, input encoding and the read out, as shown in Figure 2.4. For quantum computing it is crucial how the classic input is encoded into the device, since this strongly affects the computational power of the applicable quantum algorithms. The process of setting a quantum computer to an initial state is usually called state preparation. To be able to utilize classical data in quantum algorithms, the dataset must be represented by quantum states. Input encoding is often the bottleneck in quantum algorithms as efficient encoding strategies require a significant amount of quantum computing power [32].

In contrast to classical systems, quantum systems do not have a permanent storage and the data has to be encoded and read out of the quantum computer every time it goes through the processing stage. This is a consequence of the **no-cloning theorem** of quantum mechanics which states that it is impossible to create an independent and identical copy of an arbitrary unknown quantum state. Its time-reversed dual the **no-deleting theorem** of quantum information theory which states that given two arbitrary copies of some arbitrary quantum state, it is impossible to delete one of the copies [34].

To use classical data in a quantum algorithm, first a quantum state has to be prepared to then encode the classical data into the circuit. Thereafter, the quantum calculation can be carried out. In the last step a quantum measurement has to be performed to extract the classical result, i.e. to translate back the quantum information into the classical world. Due to the probabilistic nature of quantum mechanics, usually this process has to be carried out multiple times to get a useful result.

Qubits

In classical computation and information, the bit is the smallest unit of information. Quantum computation and information uses the quantum bit, or qubit as an alternative concept [33].

A quantum system that can exist in any quantum superposition of two independent (physically distinguishable) quantum states is called a two state quantum-system (also known as two-level system). Any two state quantum-

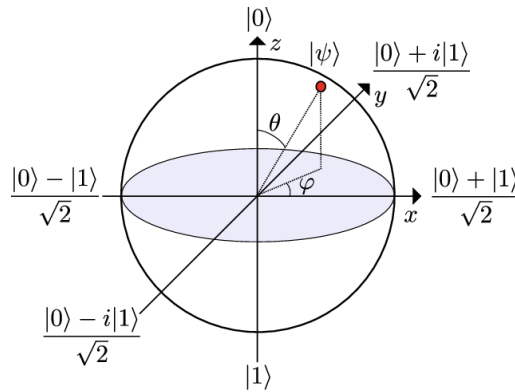


Figure 2.5: The Bloch sphere representation of a qubit state. Adapted from [41].

system can represent a qubit. Physical realisations of a qubit are for example the spin of an electron,¹ which can be either in the state up, $|\uparrow\rangle$, or down, $|\downarrow\rangle$. Another realisation is to use the polarizations of a single photon,² where two orthogonal polarizations (horizontal and vertical) encode a qubit. Most current NISQ computers use superconducting qubits [37].³ In quantum information the two orthonormal basis states of a qubit are usually denoted by

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

and are referred to as the computational basis states [6]. In contrast to the binary classical bit, the state of a qubit can be any superposition of the basis states:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \quad (2.24)$$

with $\alpha_i \in \mathbb{C}$. According to Equation 2.16 the outcome of measuring the state of the qubit in the computational basis gives $|0\rangle$ with probability $|\alpha_0|^2$, or $|1\rangle$, with probability $|\alpha_1|^2$, and $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

Equivalent in vector notation:

$$|\psi\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}; \quad \langle\psi| = \begin{pmatrix} \alpha_0^* & \alpha_1^* \end{pmatrix} \quad (2.25)$$

To get a geometric representation Equation 2.24 can be rewritten as:

$$|\psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right) \quad (2.26)$$

where $0 \leq \theta \leq \pi$, $0 \leq \varphi \leq 2\pi$, γ are real numbers [33] and the factor $e^{i\gamma}$ is a global phase.

Equation 2.26 implies a geometrical interpretation of a qubit as a unit vector in a three-dimensional vector space that points from the origin to the surface of the unit sphere as shown in Figure 2.5. This is usually referred to as the Bloch sphere representation of a qubit.

Using the computational basis states, the inner product of two single qubit states, $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ and $|\phi\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle$, reduces to :

¹Used by Intel [35]

²Used by Xanadu [36].

³Used by Google, IBM, and Rigetti among others [38, 39, 40].

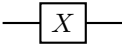


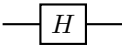
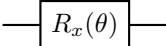
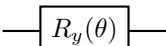
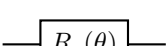
Gate	Circuit representation	Matrix representation
Pauli- $X = \sigma_x$		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli- $Y = \sigma_y$		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli- $Z = \sigma_z$		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Hadamard H		$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Pauli rotation $R_x(\theta) = e^{-i\frac{\theta}{2}\sigma_x}$		$\begin{pmatrix} \cos(\frac{\theta}{2}) & -i \sin(\frac{\theta}{2}) \\ -i \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$
Pauli rotation $R_y(\theta) = e^{-i\frac{\theta}{2}\sigma_y}$		$\begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$
Pauli rotation $R_z(\theta) = e^{-i\frac{\theta}{2}\sigma_z}$		$\begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$

Table 2.1: Examples of useful single-qubit logic gates, modified from [6, 33]

$$\langle \psi | \phi \rangle = \alpha_0^* \beta_0 + \alpha_1^* \beta_1 \quad (2.27)$$

and the outer product is described as:

$$|\psi\rangle \langle \phi| = \begin{pmatrix} \alpha_0 \beta_0^* & \alpha_0 \beta_1^* \\ \alpha_1 \beta_0^* & \alpha_1 \beta_1^* \end{pmatrix}. \quad (2.28)$$

An arbitrary computational basis state of a n -qubit system is the direct product of the single qubit states $|q_1\rangle, \dots, |q_n\rangle$. Any multi qubit state can be represented in this basis,

$$|\psi\rangle = \alpha_0 |0\dots 00\rangle + \alpha_1 |0\dots 01\rangle + \dots + \alpha_{2^n-1} |1\dots 11\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \quad (2.29)$$

with $\alpha_i \in \mathbb{C}$, and $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. The rightmost part of the equation represents an abbreviated representation of the computational basis states as the binary string of the integer i [6]. For example, $|5\rangle \equiv |0101\rangle$ for a 4-qubit system. For simplicity also the following notation is used:

$$|\phi\rangle = |q_1\rangle \otimes \dots \otimes |q_2\rangle, \text{ or in shorthand } |\phi\rangle = |q_1 \dots q_2\rangle, \text{ i.e. } |ab\rangle \equiv |a\rangle \otimes |b\rangle. \quad (2.30)$$

Gates

Quantum algorithms are commonly expressed as circuit models, where qubits take the place of classical bits, and quantum gates replace classical gates and perform computations on qubits [33]. The basic operations central to quantum computing are quantum logic gates and computation basis measurements (see Section 2.3.2). Quantum logic gates are realizations of unitary transformations and thus rotate the qubit state in the underlying Hilbert space [6]. Note that since all quantum gates are unitary all quantum computations are reversible in contrast to classical computing.

Gate	Circuit representation	Matrix representation
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
SWAP		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
controlled- Z		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$

Table 2.2: Examples of useful multi-qubit logic gates, modified from [6, 33]

The simplest gates possible are single qubit gates described by 2×2 unitary transformations. Table 2.1 shows commonly used single qubit gates. Two useful gates are the Pauli- X gate and the Hadamard gate. Pauli- X swaps the amplitudes of the $|0\rangle$ and $|1\rangle$ components and therefore is equivalent to the classical NOT gate, i.e. $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. The role of the single qubit Hadamard gate H is to create a superposition state:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \text{ and } H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle), \quad (2.31)$$

these states are the so called one-qubit Bell states. (Parametrised)-Rotation gates can also create superpositions depending on the angles.

In Table 2.1 more examples of useful gates can be found. One of the most important gates are the Pauli gates, represented by the Pauli matrices, $\sigma_x, \sigma_y, \sigma_z$.

Quantum gates can also be parametrized which means that a classical parameter determines the properties of the gate. The most important ones are the three Pauli rotations $R_x(\theta), R_y(\theta)$ and $R_z(\theta)$ where the classical parameter θ determines the rotation angle, in radians.

Gates can also operate on multiple qubits. A simple way to create a multiqubit gate is to combine single qubit gates through tensor products as described in Section 2.3.1. Frequently used multi qubit gates are controlled gates, that alter the state of a set of target qubits depending on the state of a set of control qubits. Often used multiqubit gates are two-qubit controlled gates (see Table 2.2). Such gates are often used to introduce or reduce the entanglement of the qubits in the system. A prototypical controlled gate is the $CNOT$ gate, where the X gate is applied to the target qubit if the control qubit is in state $|1\rangle$:

$$|00\rangle \mapsto |00\rangle, |01\rangle \mapsto |01\rangle, |10\rangle \mapsto |11\rangle, |11\rangle \mapsto |10\rangle. \quad (2.32)$$

More generally with U an arbitrary single qubit gate, a controlled- U operation is a two-qubit operation where U is applied to the target qubit $|t\rangle$ if the control qubit $|c\rangle$ is set, otherwise the target qubit is left unchanged, $|c\rangle|t\rangle \rightarrow |c\rangle U^c|t\rangle$ [33], i.e.

$$cU = |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes U. \quad (2.33)$$

Controlled gates with the control qubit in a superposition state are used to create entangled calculation branches, one side of the branch assuming the control qubit is set and the other assuming it isn't. The interference between

different branches can be used to calculate values based on both branches at the same time. This phenomenon is often used in quantum algorithms, for example in the swap, Hadamard, and inversion tests [6].

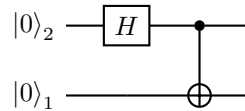
As an example of the introduced notation and gates, and to introduce quantum circuit diagrams we observe the evolution of a two qubit state when passing through a Hadamard and $CNOT$ gate. We start with two qubits in the state $|0\rangle_2 \otimes |0\rangle_1 \equiv |00\rangle_{21}$, where the qubits and gates have been labelled indicating on what qubit each gate is applied for clarity.

$$CNOT((H_2 \otimes \mathbb{1}_1)(|0\rangle_2 \otimes |0\rangle_1)) \tag{2.34}$$

$$=CNOT\left(\frac{1}{\sqrt{2}}(|0\rangle_2 \otimes |0\rangle_1)\right) + CNOT\left(\frac{1}{\sqrt{2}}(|1\rangle_2 \otimes |0\rangle_1)\right) \tag{2.35}$$

$$=\frac{1}{\sqrt{2}}(|0\rangle_2 \otimes |0\rangle_1 + |1\rangle_2 \otimes |1\rangle_1) = \frac{1}{\sqrt{2}}(|00\rangle_{21} + |11\rangle_{21}) \tag{2.36}$$

In the first line we apply the Hadamard gate to the second qubit, which is followed by the $CNOT$ gate in the second line. The created state is one of the famous Bell states [6], which are the maximally entangled quantum states of two qubits. The above calculation can be represented in the following circuit representation:



The \bullet symbol denotes the control and the \oplus symbol denotes the target qubit.

Measurement (Read out)

Most of the current quantum computing platforms only implement basis measurements, which measure whether the individual qubits are in the $|0\rangle$ or $|1\rangle$ state. More complicated observables can be implemented by applying a circuit before measuring.

The computational basis measurement of a generic qubit state $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, is determined by the projectors on the possible eigenspaces $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$ (see Section 2.3.1). The probability of obtaining each measurement is:

$$p(0) = \langle \psi | P_0 | \psi \rangle = |\alpha_0|^2 \text{ and } p(1) = \langle \psi | P_1 | \psi \rangle = |\alpha_1|^2. \tag{2.37}$$

The observable corresponding to a computational basis measurement is the Pauli Z observable (see Table 2.1). The eigenvalues of the observable correspond to the possible measurements, where $+1$ corresponds to the $|0\rangle$ observation and -1 to the $|1\rangle$ observation. The expectation value of a single-qubit measurement in the computational basis is $\langle \sigma_z \rangle \in [-1, 1]$. On multiple qubits, computational basis measurements can be interpreted as drawing a sample of a binary string of length n , with n the number of qubits, from a distribution defined by the quantum state.

Measuring a single qubit can be seen as equivalent to estimating the probability p when sampling from a Bernoulli distribution, therefore the number of samples S needed to estimate $\langle \mathcal{M} \rangle$ with error ϵ can be determined with conventional statistics. Using different methods it can be estimated that ≈ 15000 samples are needed to reach a 99% confidence and $\epsilon = 0.01$ [6].

2.4 Quantum Machine Learning (QML)

2.4.1 Ideas of Quantum Machine Learning

Quantum Machine Learning (QML) is the combination between quantum computing and Machine Learning (ML). Proposals to combine both fields exist since around the 1980s. In 1995, after Shor proved the potential power of quantum computers by introducing his famous prime factorisation algorithm [42], the first contributions looking at quantum models of neural networks were published [43]. In the early 2000s the question of statistical learning theory in a quantum setting was discussed, but received limited attention. In 2009, the QBoost algorithm was published and implemented on the first commercial quantum annealer, the D-Wave device [44]. Starting in the mid-2010s, the term “quantum machine learning” was coined and the interest in the topic started to grow significantly. Nowadays, QML is an active and established sub-discipline of quantum computing research [5]. QML changed the aim of quantum algorithm investigation from trying to provide speed-ups, to empirical research about what these algorithms can achieve.

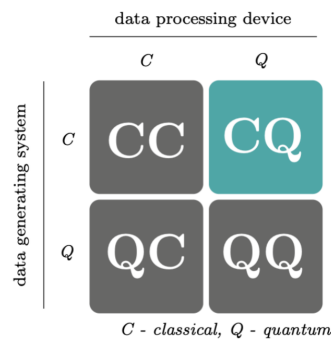


Figure 2.6: The four possibilities to combine quantum or classical data and quantum or classical processing devices. This work falls into the highlighted sector CQ. Image and description modified from [5].

The combination of classic (C) or quantum (Q) data, with classic (C) or quantum (Q) processing devices, is possible in four distinct ways as shown in Figure 2.6. The four modes of operation in the context of ML/QML can be explained as follows:

- *CC* refers to classical machine learning, i.e. classical data being processed classically.
- *QC* explores how classical ML can help the field of quantum computing. For example ML has been used to improve quantum state representation or the hardware setup of NISQ devices, and to discriminate quantum data [5].
- *CQ* is the case when classical data is fed into a quantum computer. In this work QML is used as a synonym for this configuration. Note that this operational mode requires a quantum-classical interface that introduces strict lower bounds to the total runtime of QML algorithms. Loading classical data points into a quantum computer takes linear time.
- *QQ* this last mode uses data generated by a quantum system and processes it with a quantum system.

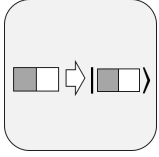
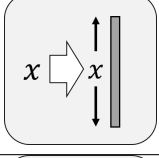
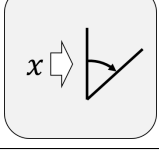
Encoding Pattern	Encoding	Required Qubits
	$x_i \approx \sum_{i=-k}^m b_i 2^i \mapsto b_m \dots b_{-k}\rangle$	$l = k + m$ per data-point
	$X \mapsto \sum_{i=0}^{n-1} x_i i\rangle$	$\lceil \log n \rceil$ per data-point
	$x_i \mapsto \cos x_i 0\rangle + \sin x_i 1\rangle$	1 per data-point

Table 2.3: Adapted from [45, 46]. x_i represents a scalar, X a vector and $b_i \in \{0, 1\}$. The graphics can be found in <https://quantumcomputingpatterns.org/>

2.4.2 Input Encoding

The encoding of classical inputs plays a critical role for QML. A central aspect for runtime evaluations are the interface components, such as software, hardware, and theoretical frameworks between classical memory and quantum devices. Since most quantum machine learning algorithms return probabilistic results, the whole routine, including state preparation and measurement, may have to be repeated several times [32].

Table 2.3 shows the most common input strategies that are explained in the following subsections.

As a starting point for the encoding strategies it can be assumed, that each algorithm starts with a n -qubit system in the ground state $|0 \dots 0\rangle$, and accessible classical data.

Computational Basis Encoding

Basis encoding is the most straightforward encoding method where a n -bit classical string is associated with a corresponding n -qubit computational basis state. To encode a real number $x \in \mathbb{R}$ into a quantum state $|x\rangle$, x is first expressed in binary representation, e.g. floating point representation, and then encoded into an appropriately sized qubit register. This is easily understood by observing the conversion of the integer 4 into a qubit representation, $4 \equiv 0100_2 \mapsto |4\rangle \equiv |0100\rangle$.

For a binary dataset \mathcal{D} , where each pattern is represented as a binary string of length n , $x^m \rightarrow (b_1^m, \dots, b_n^m) \in \mathcal{D}$, with $b_i^m \in \{0, 1\}$ and $m = 1, \dots, M$ as the label for the elements in the dataset, the state that encodes the complete dataset has the form,

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^M |\mathbf{x}^m\rangle. \quad (2.38)$$

Note that n is also the size of the qubits register. A detailed explanation and complexity analysis of the state preparation scheme for $|\mathcal{D}\rangle$ is given in References [6, 32].

If the result of an algorithm is also in computational basis encoding format, the value of the amplitude of each basis

state is associated to the probability of being measured. “The goal of a quantum algorithm is therefore to increase the probability or absolute square of the amplitude that corresponds to the basis state encoding of the solution.” [6]

A disadvantage of this encoding strategy is the high amount of qubits needed, and therefore it is seldom used in NISQ algorithms.

Amplitude Encoding

Amplitude encoding associates classical information with quantum amplitudes and is less common in quantum computing. A normalised vector $\mathbf{x} \in \mathbb{C}^{2^n}$ can be represented by the amplitudes of a quantum state $|\psi\rangle \in \mathcal{H}$:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix} \leftrightarrow |\psi_{\mathbf{x}}\rangle = \sum_{j=0}^{2^n-1} x_j |j\rangle \quad (2.39)$$

A complete dataset \mathcal{D} of elements \mathbf{x}^m ($m = 1, \dots, M$) encoded in superposition has the form:

$$|\psi_{\mathcal{D}}\rangle = \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} \sum_{i=0}^{2^n-1} x_i^m |i\rangle |m\rangle = \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} |\psi_{\mathbf{x}^m}\rangle |m\rangle. \quad (2.40)$$

This encoding has the restriction that only normalized classical vectors can be encoded, so that the quantum state representing the data has one less degree of freedom. In comparison to computational basis encoding, amplitude encoding uses less qubits to store the same amount of classical information at the expense of increasing the complexity of the encoding process [6].

Time evolution encoding

Time evolution encoding, also called rotation encoding, associates a scalar $x \in \mathbb{R}$ with the angle of a parametrised Pauli rotation. Formally, this is described as associating x with the time t in the unitary evolution of a Hamiltonian:

$$U(x) = e^{-ixH} \quad (2.41)$$

where the state after evolution, $|\psi(x)\rangle = U(x)|\psi_0\rangle$, encodes x depending on H . Any 2×2 matrix H can be decomposed into $H = a_0 \mathbb{1} + \sum_i a_k \sigma_k$ for $k \in \{x, y, z\}$ and $\alpha \in \mathbb{R}$. The most common realization of time evolution encoding is by parametrized Pauli rotations [32], hence the name rotation encoding.

Successive gates or evolutions of the form $U(x)$ can be used to encode a real-valued vector $\mathbf{x} \in \mathbb{R}^N$, i.e. $U(x) = R_i(x_0) \otimes \dots \otimes R_m(x_n)$ with $i, \dots, m \in x, y, z$. Rotation encoding produces sine/cosine structures in the amplitudes and leads to a relation with Fourier series (Section 2.4.3).

An advantage of this encoding method compared to the previous ones is that a small amount of qubits is needed, and the encoding circuits are simpler thus making it a common encoding strategy for NISQ devices.

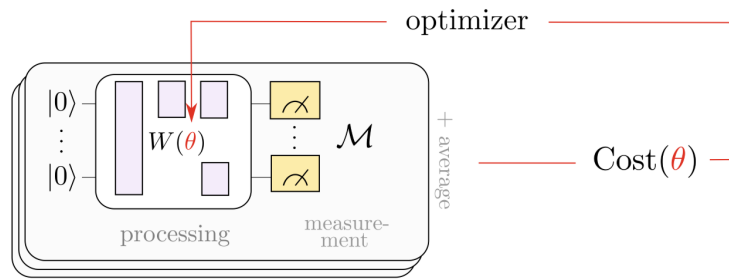


Figure 2.7: Principle of a VQC, the circuit depends on parameters and a cost function that evaluates the expected measurement for a given set of parameters. The computational problem is encoded as a cost minimisation. Normally, training is carried out iteratively and in each step the cost function is consulted to find better parameters θ . Usually, the optimisation is done on a classical computer. Image and description modified from [6]

Data Encoding as a Feature Map

From a mathematical perspective, encoding $x \in \mathcal{X}$ into a quantum system is a feature map, assuming that the map has an inner product, from input space \mathcal{X} to the state space of a quantum system [32]. A data encoding feature map can be interpreted as Dirac vectors representing feature vectors with the map:

$$\phi : x \rightarrow |\phi(x)\rangle \quad (2.42)$$

with $\langle \phi(x) | \phi(x') \rangle$ the inner product in the standard Dirac bra-ket notation.

This process can also be seen as embedding classical data in a quantum state space, as in natural language processing. A crucial aspect is that most data encoding strategies introduce a non-linear operation in the data. Nonlinearities can change the distance between data points and thus “change the intrinsic hardness of a machine learning problem for better or for worse” [32]. Once the data is encoded into quantum states, quantum algorithms can only apply unitary operations which do not change the distances between states. This highlights the importance of the embedding provided by the encoding strategy.

2.4.3 Variational Quantum Circuits (VQCs)

ML is a mostly empirical field. This presents a problem for QML, since current simulators and NISQ hardware limit algorithms strictly to a few qubits and gates. Large routines are at least very costly, or cannot be simulated even on state of the art classical hardware, and quickly become unfeasible on real quantum hardware due to high levels of noise. VQCs present a possibility to implement larger QML algorithms under these constraints by creating a hybrid quantum-classical algorithm. Figure 2.7 shows such an algorithm, where the model is realized with a quantum circuit and learning is carried out on a classical processor.

Variational Quantum Algorithms (VQAs) are designed to work on NISQ devices, and instead of having a fixed sequence of static gates, they are defined by an ansatz W , a pattern or “template” that describes which types of gates are applied to which qubits. Starting with all qubits in the basis state, the ansatz is repeated in layers and adapted to the number of qubits used. Usually, most of the gates used are parametrised, such as the Pauli rotations of Table 2.1, and these parameters are trained using a classical optimisation routine. Optimisation is implemented as a feedback scheme, comparable to neural network training. VQCs are therefore more precisely

a family of algorithms defined by $W(\theta)$, with ansatz W and parameters θ , from which the optimisation selects an optimal candidate [6].

VQCs were initially designed to find the ground states of quantum systems, i.e. the eigenstate with the lowest eigenvalue; such algorithms are called Variational Quantum Eigensolvers (VQE) [47]. Another early example of VQC algorithms is the so-called Quantum Approximate Optimisation Algorithm (QAOA) which was designed to solve combinatorial optimisation problems [48].

In a machine learning context, the terms Variational Quantum Algorithms (VQAs), Variational Quantum Circuits (VQCs), Parametrised Quantum Circuits (PQCs), or Quantum Neural Networks (QNNs) are used interchangeably depending on the literature consulted. In this thesis Variational Quantum Circuits (VQCs) will be used.

Model Interpretation

VQCs can be interpreted as deterministic machine learning models for classification tasks, as used in this work, but they have also been applied as probabilistic models for generative tasks.

A VQC can be interpreted as a discrete ML model with the function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that returns an output by measuring an observable \mathcal{O} for some classical input x . The VQC is represented by a quantum circuit $U(x, \theta)$ that depends on both, input data $x \in \mathcal{X}$ and trainable parameters $\theta \in \mathbb{R}^k$. Applying the circuit to the ground state, $U(x, \theta)|0\rangle$ prepares the state $|\psi(x, \theta)\rangle$. The function

$$f(x; \theta) = \langle \psi(x, \theta) | \mathcal{O} | \psi(x, \theta) \rangle, \quad (2.43)$$

therefore defines a deterministic VQC model [49]. To shorten the notation we will drop the state label ψ and only keep track of the parameters it depends on s.t. $f(x; \theta) = \langle \mathcal{O} \rangle_{x, \theta}$. The circuit can, in principle, have any internal structure, usually consisting of alternating trainable parametrised variational blocks $W(\theta)$ and data encoding blocks $S(x)$, as shown in Figure 2.8. Each block consists of gates that depend on either inputs or parameters.

Expressivity

The model function of a VQC with alternating variational and time-evolution encoding (Section 2.4.2) blocks, can be expressed as a partial Fourier series, as shown in Figure 2.8. The nomenclature *partial* indicates that only a subset of the Fourier coefficients is nonzero. For input features $\mathbf{x} = (x_1, \dots, x_N) \in \mathbb{R}^N$ and parameters $\boldsymbol{\theta} = (\theta_1, \dots, \theta_M)$ the quantum model function is thus given by

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sum_{\omega \in \Omega} c_{\omega}(\boldsymbol{\theta}) e^{i\omega \mathbf{x}}, \quad (2.44)$$

where $\omega \mathbf{x}$ is the inner product [50].

The eigenvalues of the data encoding gates used in the encoding blocks determine the integer frequency spectrum $\Omega \subset \mathbb{Z}^N$ of the Fourier series. The coefficients c_{ω} that a quantum model can realize depend on the remaining architecture, i.e. the variational blocks and the observable that defines the readout operation.

Representing quantum models as partial Fourier series determines the function families that a given class of quantum models can learn via two interrelated properties [50]. First, the frequency spectrum determines which functions $e^{i\omega \mathbf{x}}$ the model can use. Second the expressivity of the coefficients $\{c_{\omega}\}$ determines how the usable functions can be combined.

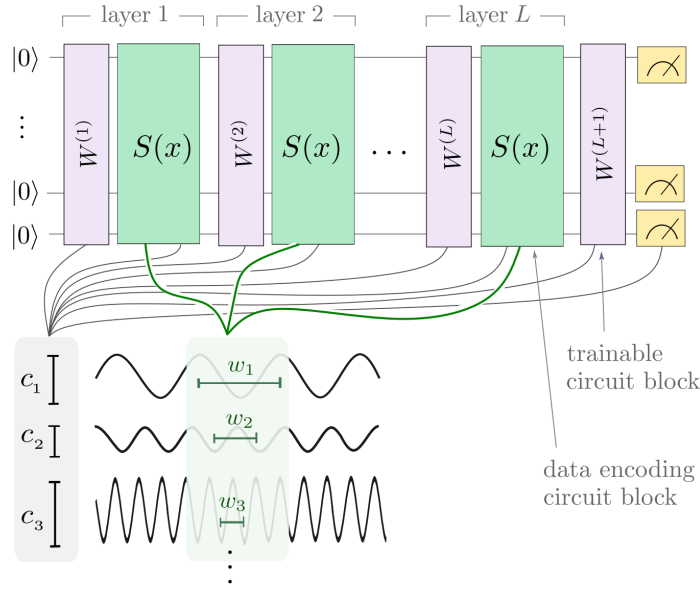


Figure 2.8: Illustration of the relation between the VQC structure and its expressivity based on partial Fourier series [50].

Using time-evolution data encoding with one parameterised Pauli rotation repeated r times sequentially or in parallel per input, only allows the model to access a frequency spectrum Ω consisting of r frequencies [50]. To increase expressivity, the encoding gates can be repeated in parallel by increasing the number of qubits used and repeatedly encoding the same data. Or in sequence by alternating variational and encoding blocks, increasing the depth of the circuit, this method is called data re-uploading [51].

In Reference [50] it was proved that for sufficiently flexible trainable variational blocks, there exist quantum models that can realise any possible set of Fourier coefficients. Thus, making such models universal function approximators, if the asymptotically accessible frequency spectrum is rich enough.

Fourier series are linear combinations of trigonometric functions. Thus, quantum models based on time-evolution encodings are periodic functions. This means that the scale of the input data must match the periodicity of the frequencies of the modelled function, since inputs differing by one period are indistinguishable. Otherwise, the model cannot fit the target function [49]. To map the data to a single period, making the input data scaling trainable has been proposed in References [4, 51].

Training of Variational Quantum Circuits (VQCs)

Training a VQC means finding parameters θ that minimise a data-dependent cost function $C(\theta)$, analogous to training a classical ANN. The difference to classical training is in the calculation of the partial derivatives with respect to the parameters of the circuit. The derivative of a parameter $\mu \in \theta$ for a quantum model $f(x; \theta)$ is given by the chain rule:

$$\partial_{\mu} C(\theta) = \frac{\partial C}{\partial f(x; \theta)} \frac{\partial f(x; \theta)}{\partial \mu}. \quad (2.45)$$

The first derivative $\partial_{f(x; \theta)} C$ is still a classical computation and can be computed using traditional auto differentiation methods. The second term depends on the quantum computation $f(x; \theta)$.

Using the finite difference method to calculate the partial derivative of the quantum expectation $f_\mu = \langle O \rangle_\mu$ with respect to a classical parameter μ leads to problems, since the results of a quantum circuit are sampled and therefore have a certain error (see Section 2.3.2). Therefore, the smaller the gradient, the more precise the estimate of the cost function must be, and more sampling shots are required, making numerical finite-difference methods unfeasible on NISQ devices [49].

Instead, quantum circuits offer an exact method of calculating derivatives, the parameter shift rule [52] given by

$$\partial_\mu f(x; \mu) = \sum_i a_i f(x; \mu + s_i) \quad (2.46)$$

with real scalar values $\{a_i\}$ and $\{s_i\}$. The shifts s_i are not necessarily small, usually $s_i = \frac{\pi}{2}$. On real quantum computers, the expectation value can only be approximated, thus the parameter-shift rule computes the *estimation of the analytical gradient*, whereas the finite-difference method only allows to compute the *estimation of the approximate gradient*.

For a VQC using parameterized Pauli gates, the gradient of the quantum circuit function, Equation 2.43, is given by [52]:

$$\nabla_\theta f(x; \theta) = \frac{1}{2} \left[f\left(x; \theta + \frac{\pi}{2}\right) - f\left(x; \theta - \frac{\pi}{2}\right) \right]. \quad (2.47)$$

A drawback of using the parameter-shift rule to compute the gradient is that for each trainable parameter the circuit has to be run twice, increasing the computation time. If a parameter occurs in multiple gates of the circuit, the usual product rule applies and the parameter-shift rule is applied to each gate independently.

Barren plateaus

A problem when training VQC that is somewhat similar to the vanishing gradient problem of classical ANN is the existence of so called barren plateaus. In contrast to classical ANN where the gradient can vanish exponentially with the number of layers, in a quantum circuit it can vanish exponentially in the number of qubits [53]. Barren plateaus are large regions in the parameter space of the cost function where its landscape is flat, i.e. the variance of the gradient is almost 0. A VQC initialised in such an area is not trainable using any gradient-based learning algorithm [53]. Vanishing gradients slow down the optimisation and are expensive in the sense that, to avoid random walks, the small gradient values must be measured with high precision. This is a problem when using NISQ devices where noise levels limit the achievable precision.

Barren plateaus can be caused by highly expressive VQC architectures with high dimensional Hilbert spaces [49]. For large architectures and vector spaces the average effect of single parameters on the measurement of expectation values are negligible, making it more probable to get a low variance. This makes the training of larger VQC more complicated, and highlights the importance of parameter initialisation to avoid barren plateaus.

Potential quantum advantages

A quantum speed-up is achieved when a quantum algorithm can theoretically solve a computational problem faster than all known classical algorithms in the asymptotic runtime [6]. This is investigated in the field of quantum complexity theory, developed as an extension of classical complexity theory. And is usually brought forward as the reason why quantum computing has potential.

For QML speed-ups are not so interesting, since what is important is what can be learnt. If something can be learnt, then quantum speed-ups can lead to faster computations. More interesting are other quantum advantages compared to classical methods. For example, generalisation performance, smaller number of data needed to train a model, faster convergence, using less trainable parameters, or solving problems that cannot be solved with classical ML methods.

Quantum advantages arise from the properties of quantum systems, especially entanglement and superposition, due to its enhanced access to the state space during computations.

Reinforcement Learning with VQC

Quantum Machine Learning (QML) has emerged as a research field suitable for applications of VQCs on NISQ devices [3]. From the three paradigms of ML, RL has received the least attention in the QML community [54]. Only recently, deep Q-learning algorithms with VQCs have been shown to solve simple environments [2, 3, 4].

Q-learning can be implemented as a supervised task, restating the problem as a loss minimisation between the predicted and target Q-values. Thus, using the deterministic interpretation of a VQC model $f(x; \theta)$ can be used to approximate Q-values, fulfilling the same function as a classical ANN. The VQC can be trained using gradient descent with the parameter-shift-rule analogous to backpropagation for classical models. Thus VQCs represent a tool for solving learning tasks in an analogous form to a classical ANN.

Using the theory described in this chapter, we combine classical and quantum paradigms to construct an experimental study of Quantum Reinforcement Learning (QRL) in the next chapter. Specifically, using the classical paradigms of Q-learning, the `tanh` output function, and loss functions to build the learning algorithm. Combining them with quantum computing ingredients such as qubits, gates, and measurements to build a VQC that uses rotation encoding and basis measurements to connect classical data with quantum computations to return the approximated Q-values. Furthermore, using the parameter-shift rule combined with classical optimisers, the complete model is trained with gradient descent. How the learning algorithm and the VQC are implemented will be explained in next Section (3.2).

3 A simulation study of Quantum Reinforcement Learning

3.1 Outline

3.1.1 Motivation

The aim of this thesis is to investigate the feasibility of Quantum Reinforcement Learning (QRL) to solve complex learning tasks in realistic environments like the problem of changing between lanes on a highway in an automated driving context. As QRL is in its early stage of development and the available simulated quantum resources are low, one of the goals was to investigate whether the QRL algorithm is able to solve several simple environments, such as Cart Pole, Frozen Lake, and Grid World, and finally to adapt what was learnt from this initial test to the more advanced environment of interest.

Since NISQ devices suffer from noise, we also investigate the impact of noise on the training of a QRL agent within a simplified noise model. Detailed noise studies are computationally costly and time-intensive and for this first study we aim to get a qualitative understanding rather than exact quantitative results.

The lane change task presents a larger and more complex scenario to test the initial learning capacity of QRL with a standard VQC. With the information learnt from this initial feasibility study, new methods and algorithms can be proposed that could potentially improve the learning capacity of QRL.

Besides a proof-of-concept, another goal of this work is to understand if there are any clear advantages using a quantum algorithm and if these are relevant for future use cases. To date, there are no clear comparisons to classical methods or they are compared in environments specifically designed to achieve a quantum advantage [3]. An interesting point is to see how the system's convergence changes when changing its size. Furthermore, the stability of the system with respect to its hyperparameters is also of importance.

3.1.2 Related work

A first study on the influence of architectural choices of VQCs for policy-based QRL algorithms was conducted in [55]. The authors highlight the crucial role of data encoding and readout strategies for policy-based QRL algorithms. For value-based QRL in discrete state spaces, a VQC has been shown to be able to solve two discrete state space environments with Q-learning [2]. This was the first time where a VQCs was used to approximate a Q-value function.

The original Q-learning algorithm can be extended to continuous state space environments [56]. For this, the potentially infinite range of input values in continuous environments was simplified by encoding the input into the angles of an initial layer of rotation gates. Qubit measurement statistics were used to represent the Q-values. Still,

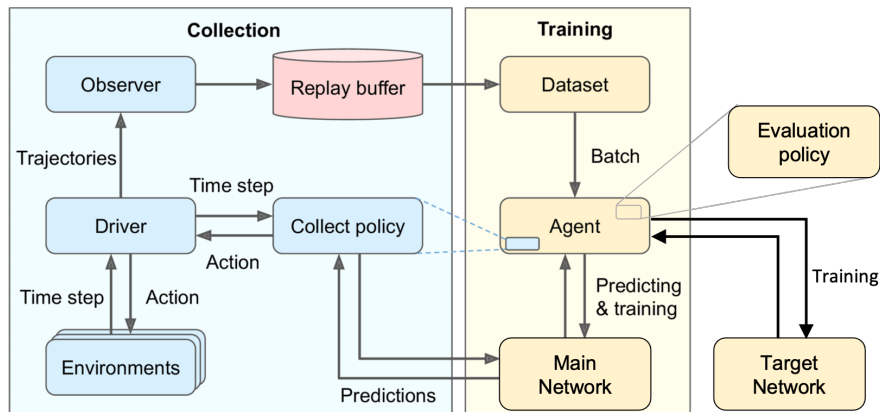


Figure 3.1: RL training architecture with classic environment, and either classic or hybrid (classic data input and quantum computation) Network, extracted from [1] and modified.

the algorithms with these modifications were not able to solve even simple problems with a continuous state space according to their specifications, as for example the cart pole environment.

Recently, a VQC structure was proposed that could solve both, discrete and continuous state space environments with a discrete action space, without restricting the state space values [3]. The authors proposed a VQC that can encode states of discrete and continuous RL environments and explained the intricate relationship between the environment specification and the requirements of the read-out operators of the quantum model [3]. To make the output scalable they added trainable output weights that enable the model to match the environment's requirements. With these improvements, both Frozen Lake and Cart Pole problems could be solved. This approach will be used for the experiments of this thesis with minimal changes.

For completeness, it is important to mention another branch of research that investigates potential quantum speed-ups for QRL. So far, the environments were always classical, and the observations had to be encoded into a quantum state. In contrast, this research branch focusses on creating a quantum-classical hybrid environment where the agent and the environment have both quantum and classical communication channels [57, 58, 59]. Quantum speed-ups have been demonstrated for simple environments. Since the environment is encoded in a quantum operator, the agent can explore the environment using an effective quantum search algorithm [60] which yields a proven advantage over classical search algorithms. This theoretical advantage has also been proven experimentally on real quantum hardware [61, 62, 63].

3.2 Proposed Implementation and Frameworks

3.2.1 Proposed RL algorithm

The Reinforcement Learning (RL) algorithm used for this work closely follows the previous work of [3, 25]. We use the same training algorithm for both quantum and classical agents and for all environments. We use a value-based, off-policy deep Q-learning approach with experience replay and fixed Q-value targets [25]:

- **experience replay:** Past transitions and their outcomes are stored in a memory, from which batches are sampled randomly to train the agent. Uniform memory sampling reduces the correlation between elements

of the dataset which improves the training, since two elements of a batch can be taken from different collection episodes.

- **fixed Q-value targets:** Adds a second network, the *target network* of the same structure as the agent's main network. The target network is used to calculate the expected Q-values for the update rule. Only the main network is trained, but sporadically, at fixed intervals, the parameters of the target network are updated with a copy of the parameters of the main network. This improves stability during training [25].

Figure 3.1 shows all components of the training algorithm, which can be separated into collection and training. In summary, the collection part explores the environment and collects training data, and the training side learns how to achieve a higher cumulative reward and updates the collect policy [1]. The complete loop is then repeated until the training is completed.

In detail, starting with the collection block, the agent uses the **collect policy** to collect training data. This is a ϵ -greedy policy with a cosine ϵ decay over the training steps. The **driver** manages the communication between the collect policy and multiple environments. Converting successive time steps (of the same environment) consisting of $(r_t, s_t, \text{done}_t)$ and the action taken to trajectories consisting of $(s_t, a_t, r_{t+1}, \text{done}_{t+1})$. Where for time t , s_t is the environment state, a_t the action taken, r_{t+1} the reward returned after taking action a_t in state s_t , and done_{t+1} describes whether the environment is in a terminal state. Multiple environments are used in parallel to better utilise hardware resources and to obtain less correlated training data. The **observer** manages the communication between the driver and the replay buffer. The **replay buffer** is an efficient FIFO queue used to store past trajectories. This is usually implemented as a server for distributed data collection. Since we use experience replay, the **dataset** is built from the replay buffer by uniformly sampling the replay buffer to build a batch of training data [25]. Each sample consists of two consecutive trajectories that are merged into $(s_t, a_t, r_{t+1}, s_{t+1}, \text{done}_{t+1})$. The difference between a trajectory and an element in the training dataset is that the latter also includes the state after taking the action. The size of the replay buffer is orders of magnitude larger than the batch size [1].

After data collection and dataset building, the agent is trained. The **agent** consists of two networks (since we use fixed Q-value targets) used to approximate Q-values, a collect policy, an evaluation policy, and an update rule for training the main network. Then, for data collection, the action with the highest Q-value is selected with probability $1 - \epsilon$, this is the collection policy. To evaluate the training progress, an **evaluation policy** is used where $\epsilon = 0$, that is, the action with the highest Q-value is always selected. The deep Q-learning agent used for this work is a custom implementation modifying the standard deep Q-learning agent provided by TF Agents [64].

The Q-value update rule given a dataset vector (Equation 2.14) is modified to use fixed Q-value targets as follows, for **main network** Q_θ and **target network** \hat{Q}_{target} ,

$$Q_\theta(s_t, a_t) \leftarrow Q_\theta(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \cdot \max_a \hat{Q}_{\text{target}}(s_{t+1}, a) - Q_\theta(s_t, a_t) \right]. \quad (3.1)$$

The learning task, updating the parameters θ of Q_θ , is then implemented as a supervised learning task by minimizing the loss between $Q_{\text{predicted}}$ and Q_{target} , with

$$\begin{aligned} Q_{\text{predicted}} &= Q_\theta(s_t, a_t), \\ Q_{\text{target}} &= r_{t+1} + \left(\gamma \cdot \max_a \hat{Q}_{\text{target}}(s_{t+1}, a) \right). \end{aligned} \quad (3.2)$$

The loss functions used were MSE and Huber loss (see Section 2.1.3) and the appropriate loss function was selected by a hyperparameter search. Finally, the parameters θ of Q_θ are copied to Q_{target} in fixed intervals, since we are using fixed Q-value targets.

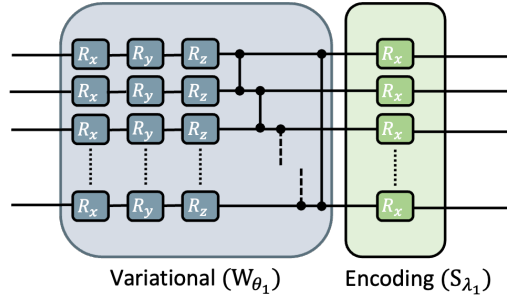


Figure 3.2: ansatz used in this work, also called layer, adapted from [3, 4, 65, 66].

3.2.2 Proposed VQC implementation

In this work, we use a Variational Quantum Circuit (VQC) for Q-function value approximation, based on [3, 4, 65, 66] shown in Figure 3.2.

Since we repeat the ansatz similar to layers in a neural network, we will refer to it as a layer, too. Each layer is divided into a variational part and an encoding part. The variational part consists of parametrised rotations with Pauli- X , Pauli- Y and Pauli- Z gates, where the rotation parameters correspond to the trainable weights of a conventional neural network. This structure is followed by a cyclical entangling using controlled Pauli- Z gates. In the encoding part the preprocessed input is fed into the VQC by rotation encoding through parametrised Pauli- X gates (see Section 2.4.2). Following the rotation encoding strategy, the number of qubits used is equal to the input vector size, i.e. the length of the observation vector defines how many qubits are used.

This ansatz is known to be highly expressive, but for a large number of qubits and layers it is susceptible to the barren plateau phenomenon. For the configurations used in this work with relatively small qubits and layers, this should not be an issue [3].

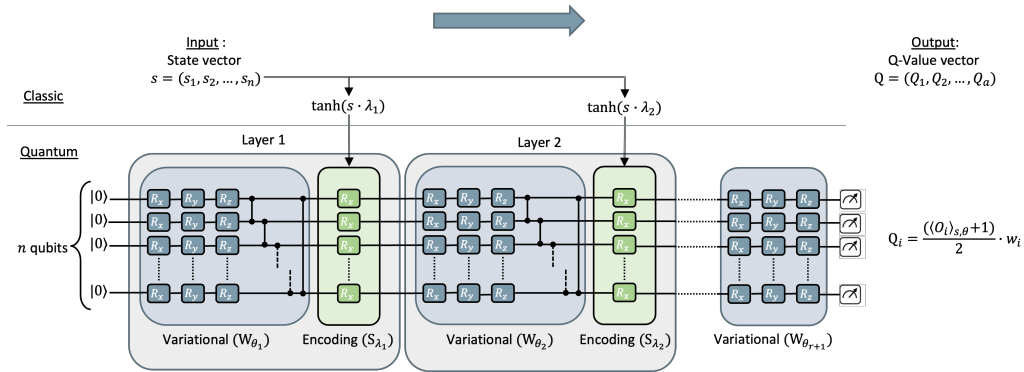


Figure 3.3: Variational Quantum Circuit (VQC) structure used for Q-value predictions. Adapted from [65]

Figure 3.3 shows the complete VQC used for the approximation of Q-values. We use a trainable classical scaling of the input data and data reuploading to improve the VQC expressivity [4, 50, 51]. The classical input state vector s is preprocessed by scaling with trainable classical weights λ and applying a \tanh nonlinearity. Therefore, the processed input is confined to $[-1, 1]$ and is then fed into the quantum circuit using parametrized VQC rotations. Note that each layer has a set of trainable classical rotation and scaling parameters (see Section 3.2.2). After r layers, a last variational block is applied before measurement.

The Q-values predicted by the VQC are expressed as the expectation values of observables, with one observable per action. The selection of observables is treated as a hyperparameter. A key difference between a classical neural network and VQCs, is that VQCs have a fixed output range defined by their measurement, while a classical network can adapt its output range during training depending on its weights and activation function [3]. To solve this issue and since Q values are generally positive, we rescale the expectation value from $[-1, 1]$ to $[0, 1]$ and then scale it by a trainable classical value [3]. For this work, we are using either single PauliZ observables or a multiplicative combination of them, so that for all observables the expectation value is in $[-1, 1]$.

For an input vector $s = (s_1, s_2, \dots, s_n)$ and a n -qubit system initialised in the ground state $|0^{\otimes n}\rangle$, the Q-value for the action i is given by:

$$Q(s, i) = \frac{(\langle 0^{\otimes n} | U_{\theta}(s)^{\dagger} O_i U_{\theta}(s) | 0^{\otimes n} \rangle) + 1}{2} \cdot w_{o_i}, \quad (3.3)$$

with $U_{\theta}(s)$ the parametrised quantum circuit, O_i the observable for action i and w_{o_i} the scaling weight for action i .

The scaling trainable parameters are initialised to 1 so that they have no effect at the start, and depending on training, they are adapted to correctly scale the input or output. This was also the main point of the VQC proposed in [3]. The variational parameters are randomly and uniformly initialised between 0 and π , due to their connection with rotation angles

We use the following libraries: Cirq [67] to specify the quantum circuit, TensorFlow Quantum (TFQ) [68] takes the quantum circuits and provides all the necessary tools for training them. It also provides the classical neural networks. TensorFlow Agents [64] and DeepMind's Reverb [69] libraries were used for the RL training loop and the Replay buffer, respectively. The Sacred [70] tool was used to record the experiments. The code is available upon request.

Trainable parameters

In our ansatz, one VQC is created for the main network and one for the target network. The main VQC is then optimised end-to-end with input s and the loss between $Q_{\text{predicted}}$ and Q_{target} . The VQC uses three different optimisers, one for updating the input weights, one for the variational parameters of the VQC and one for the output weights. The experiments in this thesis were carried out using analytical simulations with TFQ [68], thus the state of the system is available at all times. Therefore, no sampling is needed to obtain an approximation of the states. TFQ automatically combines the parameter-shift rule and backpropagation to update the trainable parameters.⁴

Each layer has $W_{\theta_j} = 3 * n$ variational parameters and $\lambda_j = n$ input encoding parameters. Additionally, the last variational block also has parameters $W_{\theta_{r+1}} = 3 * n$. And finally, for each action, we have a scaling parameter w_i .

A VQC with input $s = (s_1, s_2, \dots, s_n)$, r layers (i.e. data reuploads) and an output Q-value vector $Q = (Q_1, Q_2, \dots, Q_a)$ has the following number of trainable parameters:

$$\text{Variational:} = 3 \cdot n \cdot (r + 1). \quad (3.4)$$

$$\text{Encoding:} = n \cdot r. \quad (3.5)$$

$$\text{Output:} = a. \quad (3.6)$$

$$\text{In total:} = n \cdot (4 \cdot r + 3) + a. \quad (3.7)$$

⁴A downside of TFQ is that it cannot yet use GPU acceleration.

Variational parameters are initialised with uniform random values in $[0, \pi]$. The encoding and output parameters are all initialised with the value 1.

3.2.3 Hypotheses

The experiments studied in this thesis were designed to evaluate the capabilities of the implemented VQC to solve tasks using Q-learning. With the main focus on the feasibility of solving the lane change problem [71] using QRL. To achieve this, we set up the following three main hypotheses:

1. *The lane change problem presented in [71] can be solved using a QRL algorithm, specifically with the VQC quantum agent from [3].* The lane change environment is classified as solved when the performance is at least of the same order of magnitude as the classical agent.

The lane change problem is an environment far more complex than the standard problems that have been used. Therefore, this hypothesis generates three subordinate hypotheses that have to be answered beforehand:

- a) *The VQC is capable of solving RL tasks with a continuous state space.*
 - b) *The VQC is capable of solving RL tasks with sparse reward.*
 - c) *The VQC is robust to a scaling of the observation space to higher qubit numbers.*
2. *The implemented QRL algorithm is prone to noise.* This indicates whether the proposed QRL implementation can feasibly work on current NISQ devices.
 3. *If a QRL algorithm solves an environment, it does so with quantum advantages, e.g. faster convergence or less training data.* This would corroborate the claim of [3] that a quantum agent needs less training data, and converges faster than a comparable classical agent.

To solve the hypotheses experimentally, small environments that could be simulated were needed. We chose to use different environments with OpenAI Gym interfaces [72]. The environments are either provided directly by OpenAI Gym, or they provide a Gym-like interface; as is the case for the lane change environment proposed in [71].

For each hypothesis, a suitable environment was used, each containing partial properties of the target lane change environment. Starting with hypothesis 1a the Cart pole environment was used, since it has a continuous state space and a discrete action space, similarly to the lane change environment, but with a three-times smaller state space vector. Contrary to the lane change environment, it has immediate reward therefore, for hypothesis 1b the Frozen lake environment was chosen since this one has sparse reward. Since the previous environments have a small state space vector compared to the lane change environment, for hypothesis 1c, grid world like environments with different state space sizes were used to scale up to the target environment. Lastly, for the overall hypothesis 1 the lane change environment was used to test the current QRL capabilities. Hypothesis 2 was tested in all smaller environments and hypothesis 3 was investigated in all environments, corroborating the claims of Reference [3] was another reason to use the Cart Pole environment.

Having described the hypotheses, the environments are described in detail in the next section.



Figure 3.4: Standard environments used.

3.2.4 Selected scenarios

Cart Pole

The Cart Pole environment has a multidimensional continuous state space, a discrete action space, and immediate reward [27]. In this environment, the agent must learn to balance a pole upright on a cart that moves on a frictionless track, as shown in Figure 3.4a. The action space consists of two actions: pushing the cart to the left (action 0) or to the right (action 1). The state space is a vector of 4 elements described by Table 3.1. Exceeding the allowed values leads to the termination of the episode.⁵ At the beginning of each episode, the four variables in the environment state are assigned a uniformly random value in $(-0.05, 0.05)$ [73].

Num	Observation	Min	Max	Min allowed	Max allowed
0	Cart Position	-4.8	4.8	-2.4	2.4
1	Cart Velocity	-Inf	Inf	-Inf	Inf
2	Pole Angle (in radians)	-0.418	0.418	-0.2095	0.2095
3	Pole Angular Velocity	-Inf	Inf	-Inf	Inf

Table 3.1: The state space of the Cart Pole environment [27] is described by a 4-dimensional vector.

There are two versions of the Cart Pole environment, depending on the maximal episode length: $v0$, 200 steps and $v1$, 500 steps. The agent receives an immediate reward of $+1$ for each step taken before the end of the episode. The episode score is calculated as the cumulative reward of all steps taken in the episode. An episode is solved if the score is ≥ 195 for $v0$ and ≥ 475 for $v1$. The environment is solved when the agent can solve 100 consecutive evaluation episodes [73].

Frozen Lake

In contrast to the Cart Pole environment, the Frozen Lake environment has a discrete state space, a discrete action space, and sparse reward [74]. In this environment, the agent must learn to cross a frozen lake from start to finish without falling into any holes. The start, goal, and holes are always in the same position. Figure 3.4b shows the standard frozen lake map, with the agent in the starting position and the gift representing the goal. The action

⁵Exceeding the allowed values of the cart position means that a part of the cart is outside the visible display. For the Pole angle, it usually means that the state is physically unrecoverable.

space consists of 4-discrete actions: go left, go down, go right, and go up. If taking an action means going outside the 4×4 map, the agent remains in the same position, i.e. the agent bounces on the map walls.

The observation is a value of the current position of the agent as $current_row \cdot n_rows + current_col$ (where both the rows and the columns start at 0). For the frozen lake 4×4 map configuration used, the observations range from 0 to 15. This observation is then encoded as the 4-bit binary value vector of the integer, for example, $13 \mapsto (1, 1, 0, 1)_2$. The agent always starts in the top left position $0 \equiv (0, 0, 0, 0)_2$, and the goal is in the bottom right position $15 \equiv (1, 1, 1, 1)_2$.

The agent gets a reward of +1 when the goal is reached. Otherwise, it receives a reward of 0 per step. There are three episode termination conditions:

- The agent falls into a hole.
- The agent reaches the goal and gets a +1 reward.
- Maximum allowed steps are reached. This value is set to 100.

For the deterministic frozen lake environment, a solve condition can be defined as crossing the lake once. Repeating the crossing with the same agent will always return the same path.

Grid World

The Grid World-like environments in this work have been derived from Frozen Lake by reducing the holes to 0 (Figure 3.4c) and increasing the maximum number of steps to 300. The goal is still to cross the lake from start to finish, but in this case there is no danger of falling into the holes. The challenge of these environments comes from larger sizes of the maps. Examples of larger maps are shown in the Appendix A.1.

For maps of side length l , each square is enumerated from 0 to $(l^2 - 1)$ (in row-major order). These states are then encoded into a $\log_2(l^2)$ -bit binary representation as input to the networks. The actions remain the same as in the Frozen Lake case.

Lane change

Lastly, combining properties of the previous environments, the simulation environment for lane change was proposed in [71] and is based on the open source framework ADORe [75]. In this scenario, a vehicle has to merge into moving traffic from a slip road. The moving traffic is randomly generated. The task of the agent is to choose a suitable gap for the ego vehicle ,to change lanes. This gap is communicated to a classical algorithmic trajectory planner [76] that performs the manoeuvre, the trajectory planner is fixed and not learnable. Figure 3.5a shows an example of the manoeuvre. At the time of writing this work, not all modules needed to reproduce this environment have been made publicly available yet.

A distinct characteristic of the lane change environment is that the environment changes independently of the actions taken by the agent, i.e. the surrounding traffic adds randomness to the observations.

Identifiers are assigned to the surrounding traffic and the available gaps based on their longitudinal position with respect to the ego vehicle (Figure 3.5b). The longitudinal position is taken in the middle of each vehicle. Vehicle v_2 is always the closest vehicle longitudinally ahead (in the driving direction) of the ego car, on the target lane. If there is a vehicle in front of v_2 , it is labelled v_1 . If cars behind v_2 exist, they are labelled v_3 and v_4 , from closest

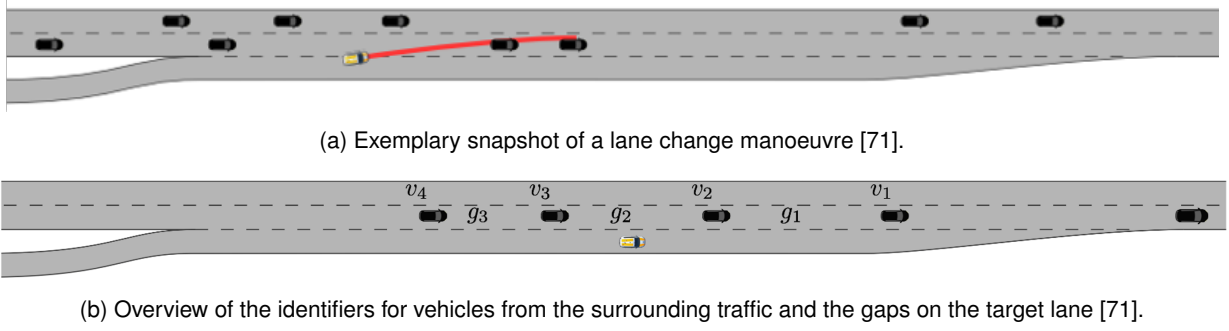


Figure 3.5: ADORe lane change environment. The yellow car represents the agent and the black cars the surrounding traffic. The ego vehicle is marked in yellow [71].

to farthest away. The state space is continuous and is described in Table 3.2. If some vehicles do not exist, their corresponding state values are set to 1 for v_1 and v_2 or to 0 for v_3 and v_4 [71].

s_0	position d_{ego} of the ego vehicle
s_1	velocity v_{ego} of ego vehicle
s_2	position $d_{1, \text{front}}$ of vehicle 1
s_3	velocity v_1 of vehicle 1
s_4	position $d_{1, \text{rear}}$ of vehicle 1
s_5, s_6, s_7	$d_{2, \text{front}}, v_2$ and $d_{2, \text{rear}}$ of vehicle 2
s_8, s_9, s_{10}	$d_{3, \text{front}}, v_3$ and $d_{3, \text{rear}}$ of vehicle 3
s_{11}, s_{12}, s_{13}	$d_{4, \text{front}}, v_4$ and $d_{4, \text{rear}}$ of vehicle 4

Table 3.2: Elements of the state vector, the values are normalised and bounded to $[0, 1]$ see [71].

The action space is discrete and consists of a_0 lane following and a_1, a_2, a_3 meaning to merge into gap g_1, g_2, g_3 , respectively. When a merging action is selected, the velocity of the ego vehicle is algorithmically adapted (not by the agent) between $7 \frac{\text{km}}{\text{h}}$ and $50 \frac{\text{km}}{\text{h}}$ to achieve the merge [71].

The ego vehicle always starts on the slip road and accelerates to the maximum speed allowed before reaching the merge situation [71]. The environment returns a sparse reward, for non-terminal states 0 and either $+50$ or -15 for terminal states, depending on whether the lane change is successful or not. Episodes terminate immediately when a lane change is successful. There are two failure settings, the first is when the ego vehicle reaches the end of the slipway without having changed lanes. The second situation occurs when no successful lane change has taken place within 7 steps after the first time that the agent returns an action other than lane following [71].

Using these environments selected for having properties that synergise with our hypotheses, we describe the experimental structure.

3.2.5 Quantum vs Classic on Cart Pole

Starting with the Cart Pole environment, we analyse whether the proposed VQC can solve an environment with continuous state space, discrete action space, and immediate reward. Furthermore, we compare the empirical performance of this quantum agent against two classical agents, especially whether quantum agents need fewer training episodes as claimed in the initial commit of [3]. While working on this thesis, an update was published in which the authors retracted said claim.

Quantum			Classic					
Layer	Output Shape	#	1-hidden layer			2-hidden layer		
Layer	Output Shape	#	Layer	Output Shape	#	Layer	Output Shape	#
InputLayer	[(None, 4)]	0	InputLayer	[(None, 4)]	0	InputLayer	[(None, 4)]	0
VQC	(None, 2)	92	Dense	(None, 13)	65	Dense	(None, 9)	45
Rescaling	(None, 2)	2	Dense	(None, 2)	28	Dense	(None, 4)	40
			Dense	(None, 2)	28	Dense	(None, 2)	10
Total trainable params (#): 94			Total trainable params (#): 93			Total trainable params (#): 95		

Table 3.3: Model structures of the agents used in the Cart Pole environment.

The classical agents are simple dense Feedforward Neural Networks with either one or two hidden layers to investigate whether the number of hidden layers influences the results. They are constrained to use the same input as the quantum agent. We fixed the number of trainable parameters for the quantum agents using the same number of layers as in [65], where an implementation of Reference [3] was proposed. For the classical agents, we chose the width of the layers such that the resulting number of trainable parameters are comparable to the trainable parameters of the quantum model, as shown in Table 3.3.

The cart pole environment has a four-dimensional state space vector. For the quantum agent, this means that 4 qubits will be used and for the classical agents, the input layer has dimension 4. Furthermore, all networks output a vector of dimension 2 with the Q-values for each possible action.

To also test the generalisation power of the agents, evaluations were carried out on two versions of the Cart-Pole environment:

- Cart pole v_0 : also used for training, with a maximum of 200 steps per epoch.
- Cart-Pole v_1 : used only for the generalisation test. Identical to v_0 , except for a maximum of 500 steps per epoch.

The idea is to test the agent on episodes that are longer than the ones it was trained on, and to see if it can balance the pole for longer, i.e. if the agent can generalise to longer episodes.

We conducted a hyperparameter search on all three models and then compared the best models. The hyperparameter search space and the selected hyperparameters are listed in the Appendix A.2.

The models were compared on their evaluation results (for v_0 and v_1), their convergence epoch, and how many trajectories each model needed until convergence. In this context, convergence means the epoch in which v_0 is perfectly solved for the last 100 consecutive evaluation episodes, i.e. the average of the cumulative reward is 200. 10 evaluation episodes are carried out every 10 training epochs. Repeating the evaluations per epoch makes sense, since the initial conditions of each episode are randomly sampled. To evaluate the initialisation stability of each model, we compare the rewards and convergence epoch for each of the best models as an average over 5, 10 and 20 trained models.

3.2.6 Quantum vs Classic on FrozenLake

Similarly to the previous scenario, we analysed whether the proposed VQC can solve an environment with discrete state space, discrete action space and sparse reward, like the Frozen Lake environment. Unlike the experiment

proposed in [3], we did not use a special input encoding for each environment, but used the same proposed VQC structure for all environments. This was done to make it comparable to the Cart Pole experiment and to investigate whether this “standard” VQC structure can be used for environments with different characteristics, specifically continuous or discrete state spaces and immediate or sparse reward. All environments used in this work have a discrete action space (otherwise, using Q-learning would be unsuitable).

For this environment, we used only one classical agent with one hidden layer for comparison, since we observed that multiple hidden layers did not significantly affect performance in the previous experiment, Section 3.2.5. Again, for the quantum agents we used the number of layers proposed in [65]. For the classical agent, we used the same dense layer width as in the Cart Pole scenario, where the trainable parameters were almost equal. But because the Frozen Lake environment has 4 actions, instead of 2, the difference in trainable parameters for the classic agent increased by 28 and for the quantum agent only by 2. For the classic agent with a hidden layer of 13 neurons, increasing the output layer by two neurons adds: 2 bias weights and $13 \times 2 = 26$ new connection weights. However, for the quantum agent, only two output scaling values are added (this is due to having the same input dimension as before and only changing the number of outputs). Resulting in 96 trainable parameters for the quantum agent and 121 trainable parameters for the classical model with one hidden layer, i.e. a difference of 25 trainable weights or in other words the classic agent has 26% more trainable parameters than the quantum agent. Still, the number of free parameters of the classical model is of the same order of magnitude as in the quantum model.

Following the proposed 4-bit binary encoding of the state values for Frozen Lake, the quantum agent used 4 qubits and the input layer of the classical agent has dimension 4. Both networks output a vector of dimension 4 with the Q-values for each possible action.

We conducted a hyperparameter search on both models and then compared the best models. The hyperparameter search space and the selected hyperparameters can be found in the Appendix A.2.

Similar to the Cart Pole experiment, the models were compared in terms of their evaluation results, their convergence epochs, how many trajectories each model needed until convergence, and additionally the path length of the solution policy. Here, convergence means the training epoch in which the goal is reached for 10 consecutive evaluation episodes. One evaluation episode is carried out every 10 training epochs. One episode suffices, since environment and model are deterministic with identical initial conditions for each rerun. As before, we compare the rewards and convergence epoch for each of the best models as an average over 5, 10 and 20 trained models. We also compared the different paths created by the trained policies to observe the variation between training repetitions.

3.2.7 Quantum scalability

For investigating the complexity limits of current quantum (simulated) hardware, we tested the ability of the proposed VQC to handle environments with larger state space, meaning more input qubits. In both previous experiments, the environment tested used at most 4 qubits. Building upon the previous experiment where it was shown that our agents could solve the frozen lake, we used a simplified version of the environment to test scalability by experimenting on Grid World-like environments. In this way, we make sure that the results depend mostly on the state vector size and not on the environment complexity, since Grid World environments do not have holes, and the only difference between them is the map size. Nevertheless, it is important to mention that we expect that with higher map size and thereby longer optimal paths, the environments become more complicated to solve.

We used the same agent structure as in the previous experiment. Table 3.4 shows the different map sizes used, their respective observation vector size, the length of the shortest possible path from start to goal, and the number

Map size (l)	4×4	5×5	8×8	11×11	16×16
Observation vector size $n = \lceil \log_2(l^2) \rceil$	4	5	6	7	8
Optimal path length	6	8	14	20	30
n.t.p. classical model	121	134	147	160	173
n.t.p. quantum model	96	119	142	165	188

Table 3.4: Description of Grid world-like environment used. With n.t.p.: number of trainable parameters.

of trainable parameters used by each type of model. Due to time and computation constraints, we did not test larger map sizes.

For all map sizes, the agent can perform four actions. The number of trainable parameters can be calculated based on the size of the observation vector n . n also represents the number of qubits used in the quantum agent and the input layer size for the classical agent. Due to their geometry, the number of trainable parameters scales different for classical and quantum models. For the quantum agent with 5 layers, and 4 output Q-values, the number of trainable parameters is, $tp_{\text{quantum}}(n) = n \cdot 23 + 4$, and for the classical network with a hidden layer of 13 neurons, it is $tp_{\text{classic}}(n) = n \cdot 13 + 69$. For environments with $n > 6$ the quantum agent has more trainable parameters than the classical agent.

We conducted hyperparameter searches for both model types on all map sizes. We then chose the best hyperparameters over all models such that both the quantum and the classical agents had the same training requirements, in the sense of how many data points are used per training iteration. In this way, we ensure that the performance differences came from the larger environments and not the model hyperparameters. All models used the same batch size, number of interactions with the environment before the main model is trained and that the same number of updates of the main model are done before updating the target mode. The hyperparameter search space and the selected hyperparameters can be found in the Appendix A.2.

The comparison of the models was done the same way as for the Frozen Lake environment.

3.2.8 Effects of noise

Current NISQ devices suffer from noise, see Section 2.3.1. To evaluate how the proposed VQC would work on real quantum hardware, we investigate the effects of noise on different environments. We used simulated noise since the computation time needed to train a model on real quantum hardware is not realistically available.

To simplify the noise experiments, we added after each gate the same noise gate with the same noise probability (see Figure 4.14). In real quantum devices, there is a difference in the noisiness of one- and two-qubit gates, with the two-qubit gate error being significantly higher. Gates of three or more qubits are not realized on real hardware and are composed of a combination of one and two qubit gates [77]. Furthermore, we neglected that depending on the placement of the qubits in hardware, some connections are more prone to noise than others. For simplicity, we use the same noise probability on all gates, and it should be interpreted as an upper bound when only using the two-qubit gate error.

For simulated noise, we follow the implementation proposed in TFQ, using depolarization gates [78]. The depolarization probability of these gates can be varied to simulate different levels of noise. TFQ uses Monte Carlo trajectory simulations with analytical calculations performed on each trajectory to estimate the noisy expectation values of observables [79]. These trajectory simulations are very computationally and time intensive, and therefore we only used 10 repetitions. Due to restrictions by the hardware used for the simulation, more repetitions lead to

highly degraded performance. This fits the requirements of the TFQ documentation, where 10 repetitions are used in QML contexts, as circuits will be called repeatedly during evaluations.⁶

Due to the extremely costly nature of noisy training we did not do a new hyperparameter search for each noise level. Instead, we used the best hyperparameters found in the noise-free experiments and retrained the models using noisy versions of their circuits.

We first investigated the Frozen Lake environment for noise levels increasing in 0.05% steps until the agent was unable to solve the environment. Due to the expensive nature of noisy simulation, we restricted Grid-World-like environments to the map sizes 4×4 , 5×5 and 8×8 with noise levels ranging from 0% to 5% in 1% steps. Finally, we compared the behaviour in the Frozen Lake environment and the Grid World-like environments since they present a similar environment but with very different termination conditions. Specifically, we wanted to see how an environment with termination conditions for bad actions, i.e. falling into a hole, compares to an environment where the only termination condition is running out of possible steps. The continuous state space problem of Cart Pole has been unsolvable with noisy circuits.

The models were compared under the same conditions as used for each environment without noise. However, 10 evaluation epochs were carried out every 10 training epochs because noisy models are no longer deterministic. Noise can cause two calculations with the same input to achieve different results.

3.2.9 Lane change

In order to investigate whether the quantum agent can also solve a more complex environment, the lane change environment proposed in [71] was used. Specifically, the environment has a continuous state space with more degrees of freedom, discrete actions, and sparse reward. Furthermore, this environment changes independently of the actions taken by the agent, i.e. the traffic changes independently of the agent's actions.

We trained only a quantum agent as we used the results from [71] as reference, and only a minimal hyperparameter search could be performed because the lane change environment is time- and computationally intensive for training. Nevertheless, we conducted a first proof-of-concept for solving the environment with a quantum agent.

Compared to the classical agent with 1334 trainable parameters from Ref. [71] our quantum agent has only 326 trainable parameters. We used the same number of layers as in all previous experiments, since we inferred from the Grid World experiments that this agent can be trained for comparable observation vector sizes and possible actions. Larger models could be susceptible to the barren plateau phenomenon [3].

Combining the described VQC and RL algorithm to solve the proposed hypotheses using the mentioned environments, we set up the experiments to be carried out. In the next chapter, the results of these will be presented and discussed.

⁶See the Noise tutorial <https://www.tensorflow.org/quantum/tutorials/noise>

4 Simulation Results

4.1 Results

4.1.1 Quantum vs Classic on CartPole

This section covers the results for the Quantum vs Classic experiment on the Cart Pole environment. After performing a hyperparameter search for each architecture, the model with the best results per architecture was selected and retrained 15 additional times (see Appendix A.2). After the training, the weights of the epoch with the highest cumulative reward for both Cart Pole v_0 and v_1 were restored for each model, and 100 evaluation episodes were carried out.

Following the definition of the Cart Pole environment, each episode score (see Section 3.2.4) is calculated as the cumulative reward of all steps taken in the episode (Equation 2.9). An episode is solved if the score is $\geq s_c$; with $s_{c_0} = 195$ for v_0 and $s_{c_1} = 475$ for v_1 . We define the solution of an environment as the consecutive solution of 100 episodes. Since the agent gets a reward $+1$ for each step in the episode, the cumulative reward of an episode is given by its length. The solving criteria for the Cart Pole environments can then be reformulated in terms of the minimal cumulative reward and T_r as the length of each episode:

$$\text{min. cumulative reward} = \min\{C_{T_0}, \dots, C_{T_{100}}\} = \min\left\{\sum_{t=0}^{T_0} r_t, \dots, \sum_{t=0}^{T_{100}} r_t\right\} = \min\{T_0, \dots, T_{100}\} \stackrel{?}{\geq} s_{c_i} \quad (4.1)$$

The length of each episode is determined by the environment and the quality of the policy of the evaluated model.

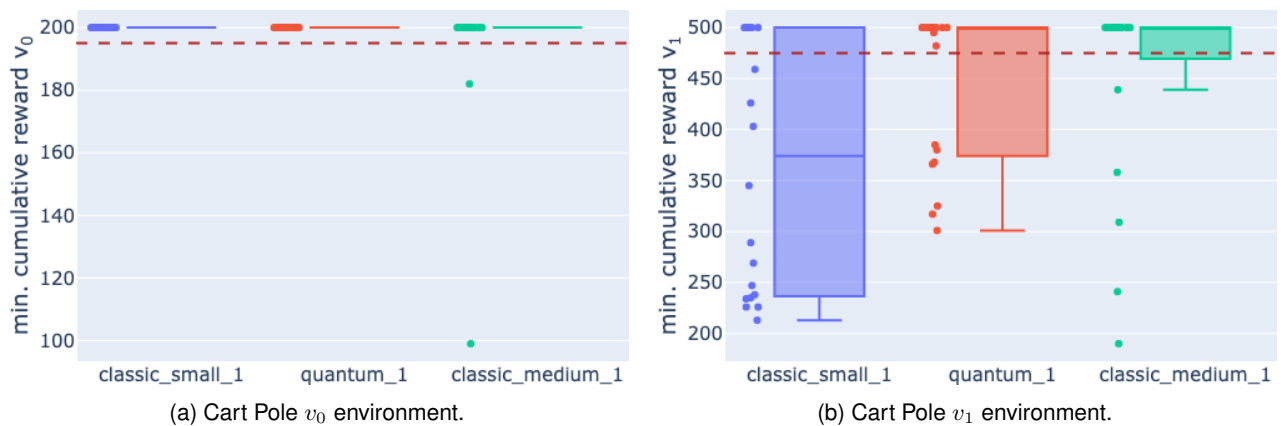


Figure 4.1: Model evaluations for the Cart Pole v_i environment after training. Showing the minimal cumulative reward over 100 evaluation episodes, see Equation 4.1. 20 training repetitions have been conducted per model type. The red dashed line represents the lower threshold for solving each environment according to its specification.

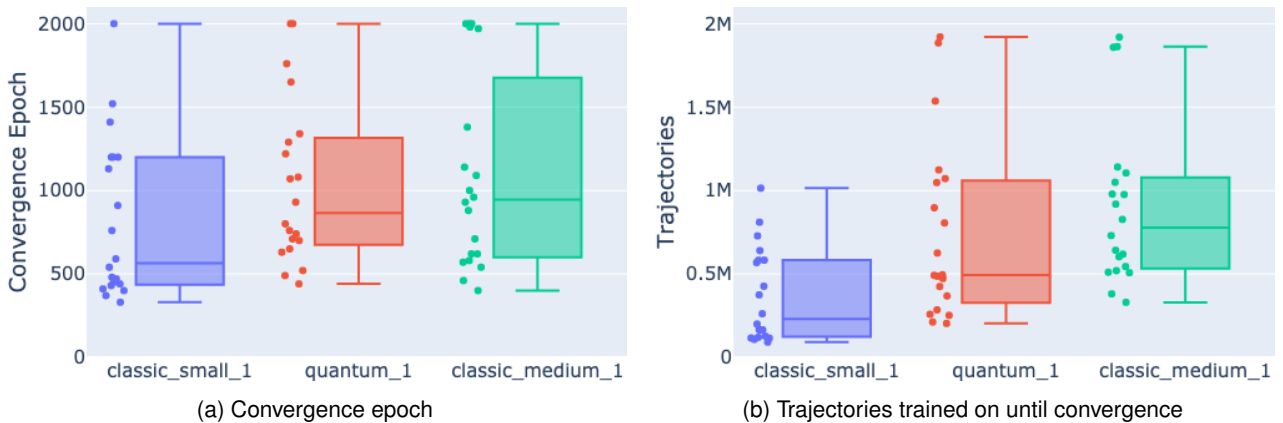


Figure 4.2: Training metrics over 20 models, per model type. **a)** epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the average cumulative reward over the last 100 evaluation episodes is ≥ 195 . Every 10 training epochs 10 evaluation episode are carried out. **b)** how many trajectories each model use until convergence, in other words how many training examples each model saw until convergence.

There are two possible termination conditions. Either the agent fails, or it reaches the maximal number of allowed steps.

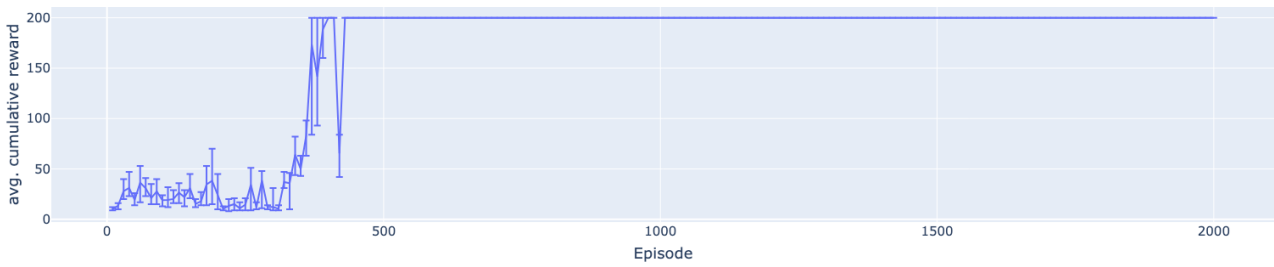
Figure 4.1 shows the minimal evaluation reward for Cart Pole v_0 and v_1 . As shown in Figure 4.1a the classical agent with one hidden layer and the quantum agent can solve the Cart Pole environment v_0 for all repetitions, and the classical agent with two hidden layers can solve the environment in 90% of the repetitions. Figure 4.1b shows the agents generalization ability. The classical agent with one hidden layer achieves the worst results with only a 35% success rate in the Cart Pole v_1 environment. Both the quantum agent and the classical agent with two hidden layers achieve significantly better results solving the environment in 65% and 75% percent of the cases, respectively. Overall, the classical agent with two hidden layers achieves the best generalization results, although it has the worst results in the training environment.

Model type	Perfect	Noisy	Forgetting	Not converging
Quantum	60%	35%	5%	-
Classical 1 hidden layer	35%	50%	15%	-
Classical 2 hidden layer	40%	25%	15%	20%

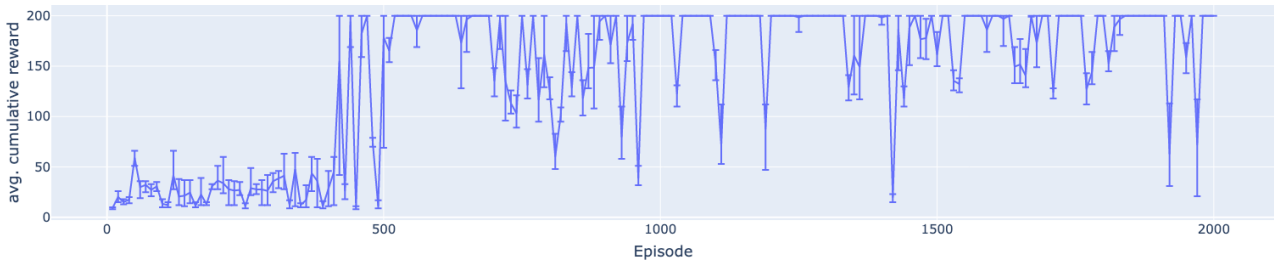
Table 4.1: For each model, the evaluation history for the training environment (v_0) can be grouped according to their behaviour. The classification was done based on visual inspection of the learning curves. Prototypical examples are shown in Figure 4.3. Per model 20 training repetitions were carried out.

The training convergence epoch is the training epoch where the minimum cumulative reward for the training environment over the last 100 evaluation episodes is 200. The convergence epochs are shown in Figure 4.2a. It can be observed that for all cases not all repetitions converged. This is due to the fact that the convergence criteria is stricter than the solution criteria, as it requires that all episodes are perfect. The best repetitions, i.e. with the lowest convergence epoch, for all three models are very similar with about 400 epochs. But the medians differ significantly, with the classical agent with one hidden layer performing the best. Both the quantum and the classical agent with two hidden layers behave slightly worse, but quite similar to each other. However, the interquartile range of the quantum agent is superior to the classical model with two hidden layers (see Figure 4.2b). Logically, the same behaviour can be observed in the number of trajectories needed until convergence, since more epochs equal

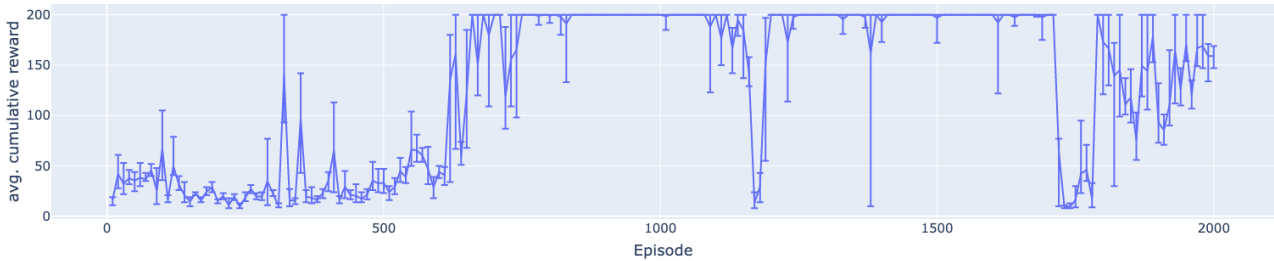
4 Simulation Results



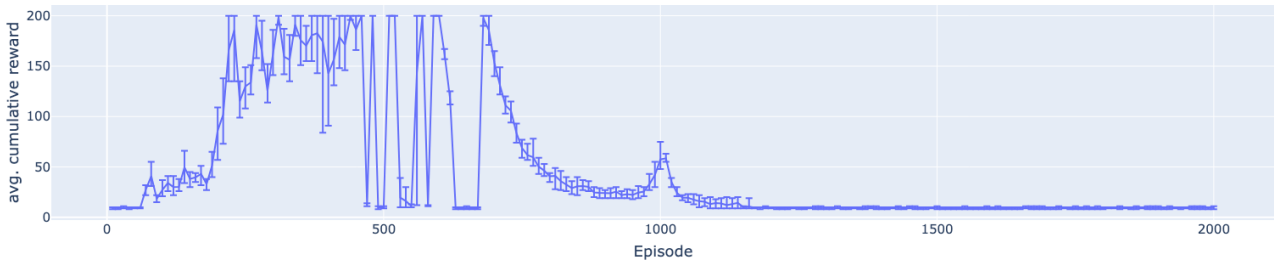
(a) Prototypical perfect run, here shown on an example of a quantum agent.



(b) Prototypical noisy run, here shown on an example of a quantum agent.



(c) Prototypical catastrophic forgetting run, here shown on an example of a quantum agent.

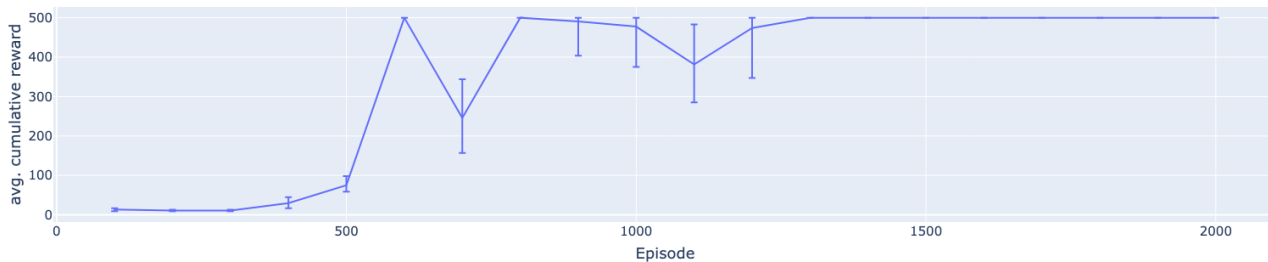


(d) Prototypical not converging run, here shown on an example of a classical agent with two hidden layers.

Figure 4.3: Prototypical examples of the evaluation history in the training environment (v_0) are shown grouped according to its behaviour. The classification was done based on visual inspection of the learning curves. Per model 20 training repetitions were carried out. Every 10 training epochs, 10 evaluation episodes were carried out. The lines represent the mean and the whiskers the maximal and minimal value over the 10 episodes.

more data. One notable difference is that in the worst case, the shallow classical agent needs half as many training trajectories compared to the other two model architectures. The exaggeration of the behaviour can be explained through hyperparameters, where the small model uses a batch size of 32 and the other two architectures a batch size of 64 (all three models use the same amount of collection steps between training epochs).

4 Simulation Results



(a) Prototypical converging run, here shown on an example of a quantum agent.



(b) Prototypical noisy run, here shown on an example of a quantum agent.

Figure 4.4: Prototypical examples of the evaluation history in the generalisation environment (v_1) are shown grouped according to its behaviour. The classification was done based on visual inspection of the learning curves. Per model 20 training repetitions were carried out. Every 10 training epochs, 10 evaluation episodes were carried out. The lines represent the mean and the error bars the maximal and minimal value over the 10 episodes.

For the evaluation history during training, we used the average cumulative reward:

$$\text{avg. cumulative reward} = \frac{1}{10} \sum_{r=0}^{10} C_{T_r} = \frac{1}{10} \sum_{r=0}^{10} \sum_{t=0}^{T_r} r_t = \frac{1}{10} \sum_{r=0}^{10} T_r \quad (4.2)$$

with T_r being the length of each episode. During training, every 10 training epochs 10 evaluation episodes were carried out (over which the average was calculated). The histories averaged over all repetitions are shown in Appendix B.1, with Figure B.1 showing the history for the training environment and Figure B.2 for the generalisation environment.

Model type	Converging	Noisy
Classical 1 hidden layer	30%	70%
Quantum	45%	55%
Classical 2 hidden layer	65%	35%

Table 4.2: For each model, the evaluation history for the generalisation environment (v_1) can be grouped according to their behaviour. Classification was done based on visual inspection of the learning curves. Prototypical examples are shown in Figure 4.4. Per model 20 training repetitions were carried out.

We (subjectively) classified all evaluation histories according to their behaviour into the following groups: perfect runs, noisy runs, or catastrophic forgetting runs. Catastrophic forgetting occurs when while exploring the environment, the agent updates its policy, but what it learns in one part of the environment may break what it learnt earlier in other parts of the environment [1]. In Figure 4.3 prototypical histories are shown for each class. For all architectures, similar behaviours could be observed. Table 4.1 shows the results of classifying 20 repeated trainings for each architecture. The table shows that the quantum model has the most perfect trials, and catastrophic forgetting

4 Simulation Results

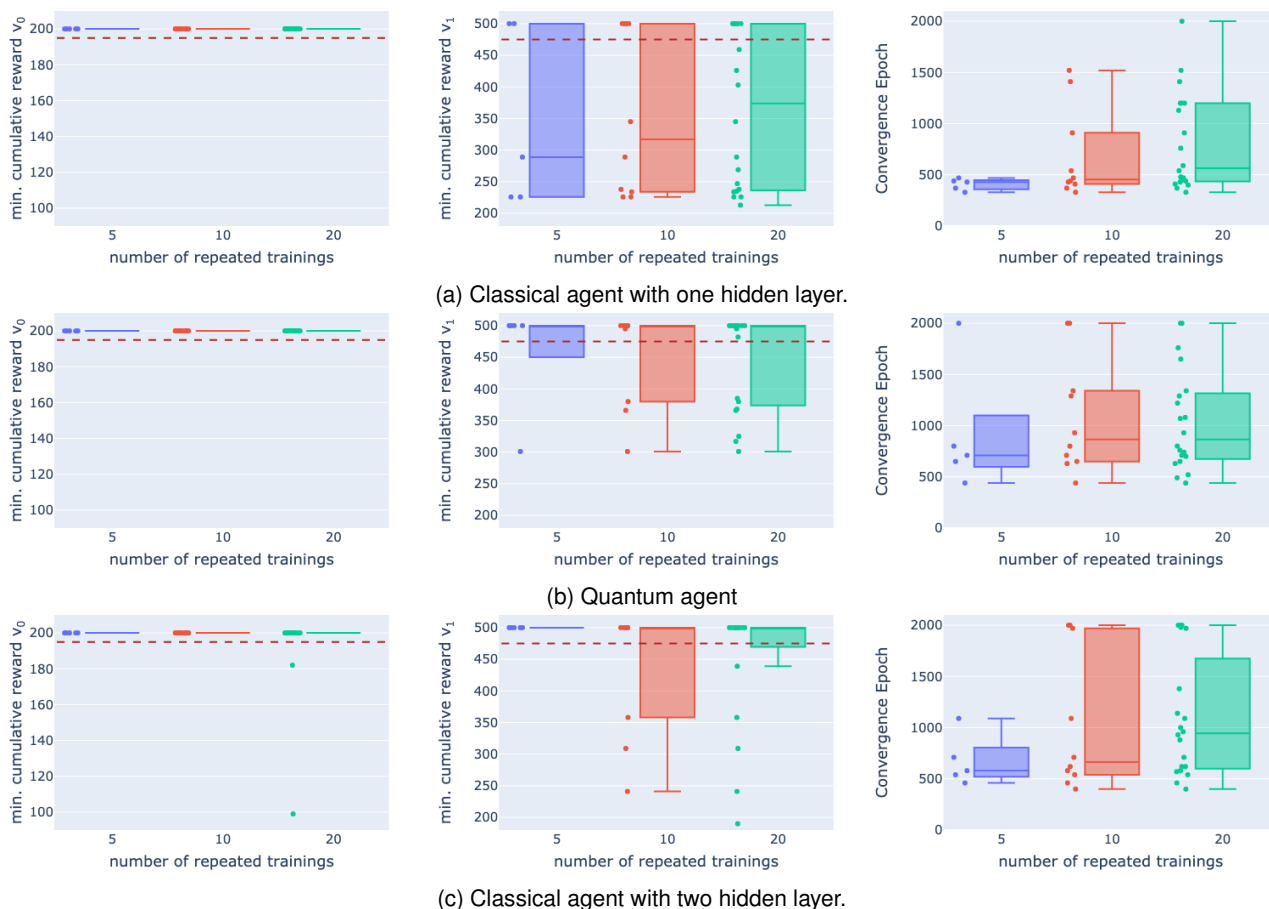


Figure 4.5: Initial condition study with 5, 10 and 20 models trained with the same hyperparameters. The average cumulative reward for training and generalisation environment and the convergence epoch are shown for different subsets of trained models.

occurs only rarely compared to the classical agents. On the contrary, this happens three times as often in the classical agents. For the classical agent v_1 with one hidden layer, half of its runs are noisy. The classical agent with two hidden layers is the only one with not converging runs.

Analogously, we conducted the same classification for the generalisation environment, shown in Figure 4.4. Table 4.2 shows the classification into converging runs and noisy runs. About half of the quantum agent runs are stable converging and the rest is noisy. For the classical agents with one hidden layer most runs are noisy and for the one with two hidden layers about two thirds are stable.

The results on the stability of training with respect to the parameters initialisation are shown in Figure 4.5. It can be seen that general trends depend highly on how many repetitions are performed for training which reflects the very unstable nature of RL and especially of Q-learning [1]. This presents the conundrum of finding the optimal balance between how many times to train each hyperparameter configuration and how many configurations can be tested given limited computation and time resources. For the hyperparameter search we used 5 repetitions, and for the results we re-trained the best results up to 20 repetitions to achieve a balance.

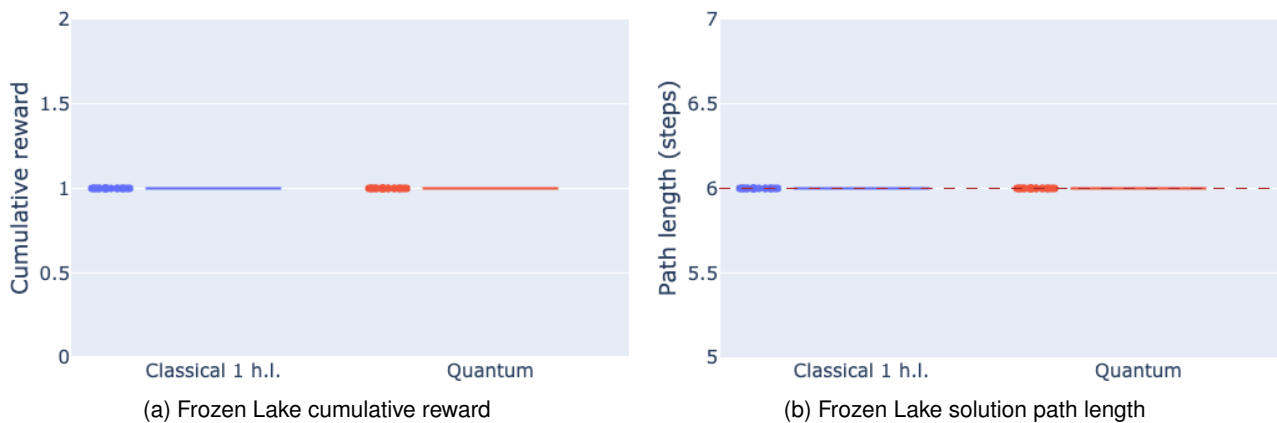


Figure 4.6: Model evaluations for the Frozen lake environment after training. 20 training repetitions per model type are shown. **a)** Showing the cumulative reward of each repetition, see Equation 2.9. Reward 1 means the environment was solved. **b)** Path length of the solution, with 6 being the optimal length and shown by the red dashed line.

4.1.2 Quantum vs Classical on Frozen Lake

For the Quantum vs Classical experiment on the Frozen Lake environment a hyperparameter search was done for each architecture. The model with the best results was selected and retrained to have 20 trials;⁷ see Appendix A.2 for details on the hyperparameter search. After training was complete, the weights of the epoch with the highest cumulative reward and the shortest solution paths were restored for each model, and an evaluation episode was carried out. Only one evaluation was necessary since repeated evaluations with identical starting conditions always lead to the same result as the environment and the model are both deterministic.

The Frozen Lake environment is solved if the agent reaches the goal. The cumulative reward of an episode is 1 iff the goal is reached, otherwise it is 0. The environment is solved independent of the chosen path as long as the goal is reached, but the path length can also be used as a metric to see if the agent achieves an optimal or suboptimal solution. Figure 4.6a shows the cumulative reward of the quantum and classical agents. From the cumulative reward it can be observed that both agents can solve the environment in all trials. Figure 4.6b further shows that all agents solved the environment with an optimal length solution (there are multiple different paths from start to finish with optimal length).

Model type	Perfect	Noisy	late converging
Quantum	60%	40%	-
Classical	20%	60%	20%

Table 4.3: For each model, the evaluation history for the training environment can be grouped according to their behaviour. The classification was done based on visual inspection of the learning curves. Prototypical examples are shown in Figure 4.8. Per model 20 training repetitions were carried out.

We define the training epoch where the minimal cumulative reward over 10 evaluation episodes is 1 as the convergence epoch. The convergence epochs are shown in Figure 4.7a, where it can be observed that the quantum agent converged approximately twice as fast as the classical agent. In the worst case, the quantum agent took around 1500 epochs of the 5000 epochs it was trained on. The classical agent took significantly longer with four repetitions converging very late during training, taking over 2800 training epochs. However, for both agents all 20

⁷the classical agent has one repetition less since one run had crashed due to hardware reasons and was not noticed until writing the results.

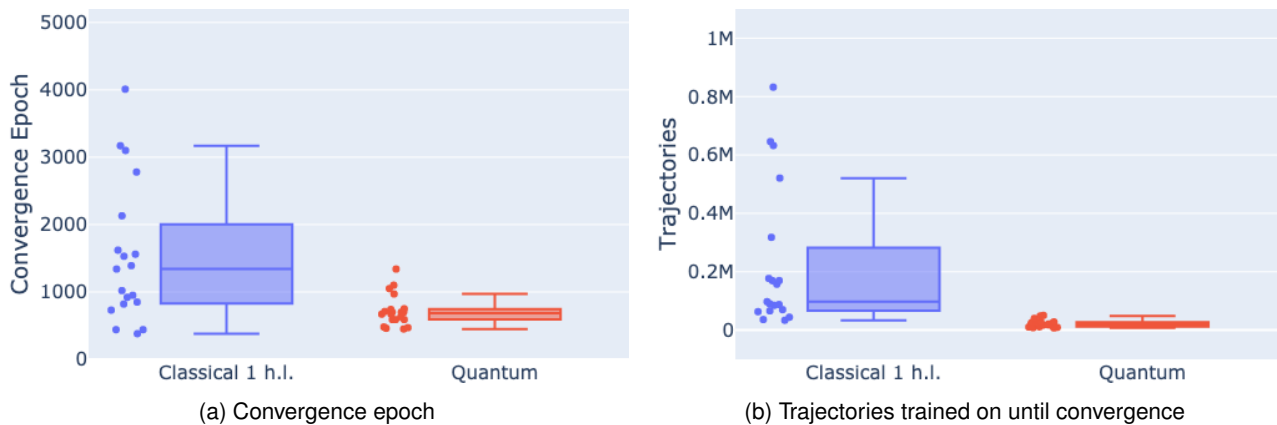


Figure 4.7: Training metrics over 20 models, per type. **a)** epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the cumulative reward over the last 10 evaluation episodes is $= 1$. Every 10 training epochs 1 evaluation episode was carried out. This means that convergence is fulfilled when for the last 100 training episodes the corresponding evaluation episode **b)** how many trajectories each model use until convergence, in other words how many training examples each model saw until convergence. This is influenced by hyperparameters, such as batch size.

repetitions converged before the training ended. In the best case, both architectures converged after around 400 training epochs. Logically, the number of trajectories needed until convergence follows the same behaviour as the convergence epochs, as is shown in Figure 4.7b. The behaviour is exaggerated due to the classical agent using a batch size of 32 and the quantum agent of 16, as this automatically leads to processing more data per training epoch.

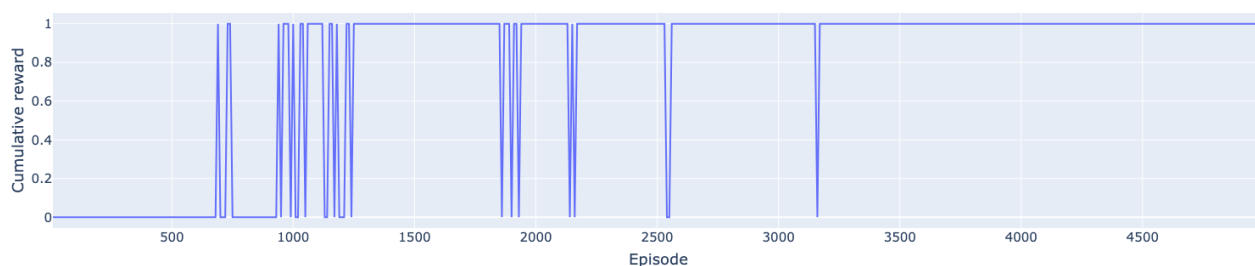
Model type	Converging	Noisy
Quantum	55%	45%
Classical	20%	80%

Table 4.4: Path convergence history for the training environment v_0 , grouped according to their behaviour. Classification was done based on visual inspection of the learning curves. Prototypical examples are shown in Figure 4.9.

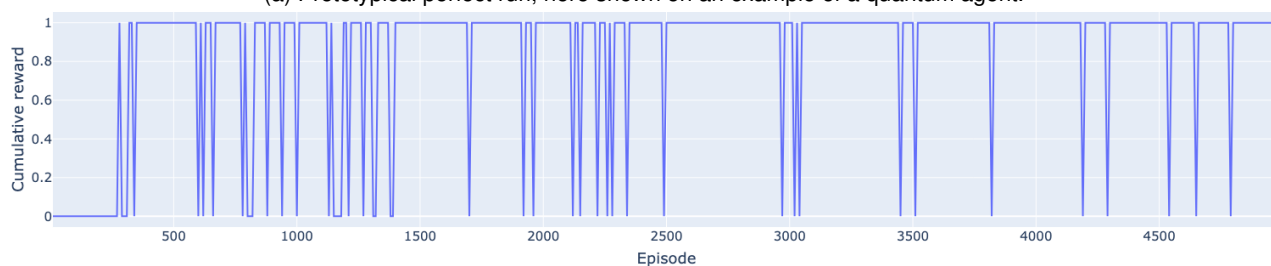
Following the same procedure as in the previous experiment we (subjectively) classified all evaluation histories into groups according to their behaviour, as shown in Figure 4.8. For all architectures, similar behaviours could be observed. Table 4.3 shows how the training repetitions are classified for each architecture. We see that the quantum model has the most perfect trials, and no late converging runs. The classical agent on the other hand presents a very noisy nature, additionally a 20% of the trials only start to learn after almost half of the maximal training epochs, at this point most quantum agents have already converged. Analogously, the same classification was carried out for the evaluation path length, shown in Figure 4.9. Paths shorter than 6 steps, represented by the dips under the stable line in the figure, are runs where the agent failed and fell into a hole, thereby terminating the episode in failure. Table 4.4 shows the classification percentages, again with the quantum agent showing more stable results than the classical agent, with half of the runs converging almost perfect. In contrast, for the classical agent most runs are very noisy. In Appendix B.2 the histories averaged over all repetitions are shown in Figure B.3 for the cumulative reward and for the path length in Figure B.4.

Similar to the preceding experiment, we conducted a study on the influence of initial conditions, the results are shown in Figure B.5 in Appendix B.2. For the Frozen Lake environment, the agents are not as susceptible to

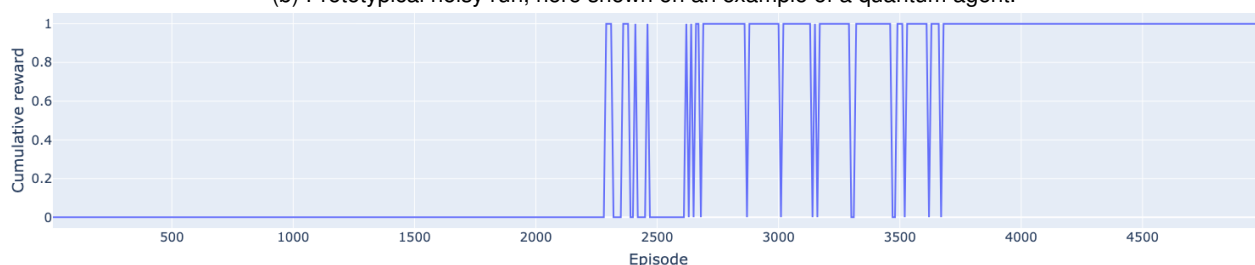
4 Simulation Results



(a) Prototypical perfect run, here shown on an example of a quantum agent.



(b) Prototypical noisy run, here shown on an example of a quantum agent.



(c) Prototypical late converging run, here shown on an example of a classical agent.

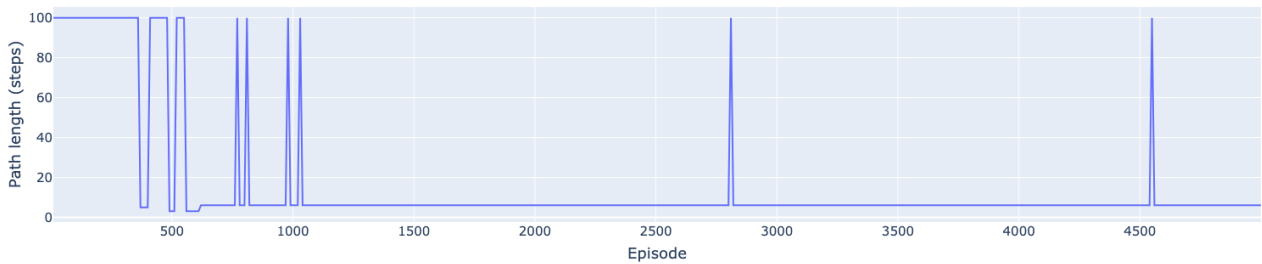
Figure 4.8: Prototypical examples of the cumulative reward evaluation history in the training environment are shown grouped according to their behaviour. The classification was done based on visual inspection of the learning curves. Per model 20 training repetitions were carried out. Every 10 training epochs, 1 evaluation episodes was carried out.

weights initializations, and especially the quantum agents varies only minimally between repetitions.

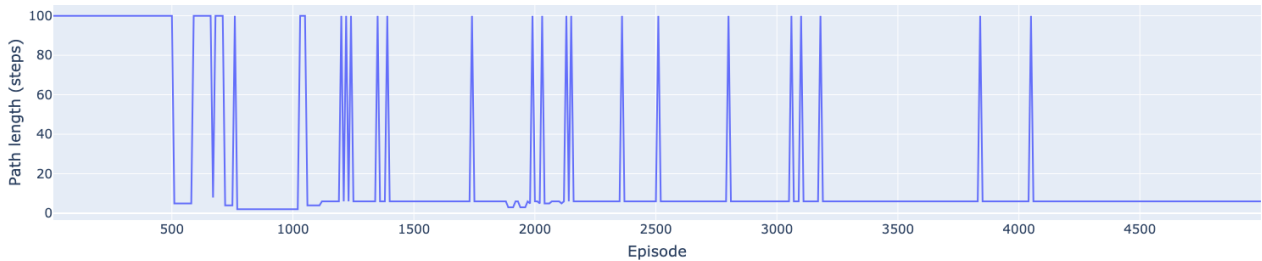
Figure 4.10 shows the different solution paths produced by trained agents, all paths are of optimal length. Both agents produced the same paths but with different frequencies, as shown in Table 4.5. The table shows that the quantum agent produced in 85% of the cases the same path, while the classical agent produced all three paths with almost equal probability.

Model type	Ideal path 1	Ideal path 2	Ideal path 3
Quantum	5%	85%	10%
Classical	40%	30%	30%

Table 4.5: The solution paths produced by the agents are grouped according to the path followed. The labelling is carried over from Figure 4.10



(a) Prototypical almost perfectly converging run, here shown on an example of a quantum agent.



(b) Prototypical noisy converging run, here shown on an example of a quantum agent.

Figure 4.9: Prototypical examples of the path length evaluation history in the training environment. The classification was done based on visual inspection of the learning curves. Per model 20 training repetitions were carried out. Every 10 training epochs, 1 evaluation episode was carried out.

4.1.3 Quantum scalability

This section presents the results for the Quantum scalability experiment (Section 3.2.7) on grid world-like environments. After performing a hyperparameter search for each architecture and map sizes 4×4 , 8×8 , and 16×16 , we chose the hyperparameters that worked the best on all environments while making sure that both quantum and classical agents had the same training requirements (in the sense of how many data points are used per training iteration). We then expanded the list of map sizes and added 5×5 and 11×11 so that we could test the models with observation vector size ranging from 4 to 8 in 1 steps, see Table 3.4. For the quantum agent and the map size 11×11 we had to redo the hyperparameter search because the agent failed to converge for no obvious reason. We only re-searched over output learning rate and ϵ decay steps (leaving the rest of the hyperparameters identical to the one used for the other map sizes). The final hyperparameter list can be found in Appendix B.3 in Table B.1. Once the hyperparameters were fixed, we retrained all models to have 20 trials. The details of the hyperparameter search can be found in Appendix A.2. After training was completed, the weights of the epoch with the highest cumulative reward and the shortest solution paths were restored for each model and map size. Then an evaluation episode was carried out.

The Grid world-like environments have the same solution and convergence criteria as the Frozen Lake environments discussed previously. Additionally, we added path convergence as criterion, given by the training epoch where the agent achieves solutions with optimal paths lengths over 10 consecutive evaluation epochs. Figure 4.11a shows the cumulative reward for varying map size. Both agents solve the environment perfectly until the map size 11×11 , where the classical agent solves all the cases, but the quantum agent does not solve the environment in 2 out of 20 cases. For the map size 16×16 both the quantum and the classical agents fail to solve the environment in 3 out of 20 cases. The same behaviour is observed in Figure 4.11b for the path lengths of the solutions.

In Figure 4.12 the training and path convergences are shown. Comparing both, it can be observed that the path convergences are always higher than the training convergence. Both agents converge later for larger map sizes,

4 Simulation Results

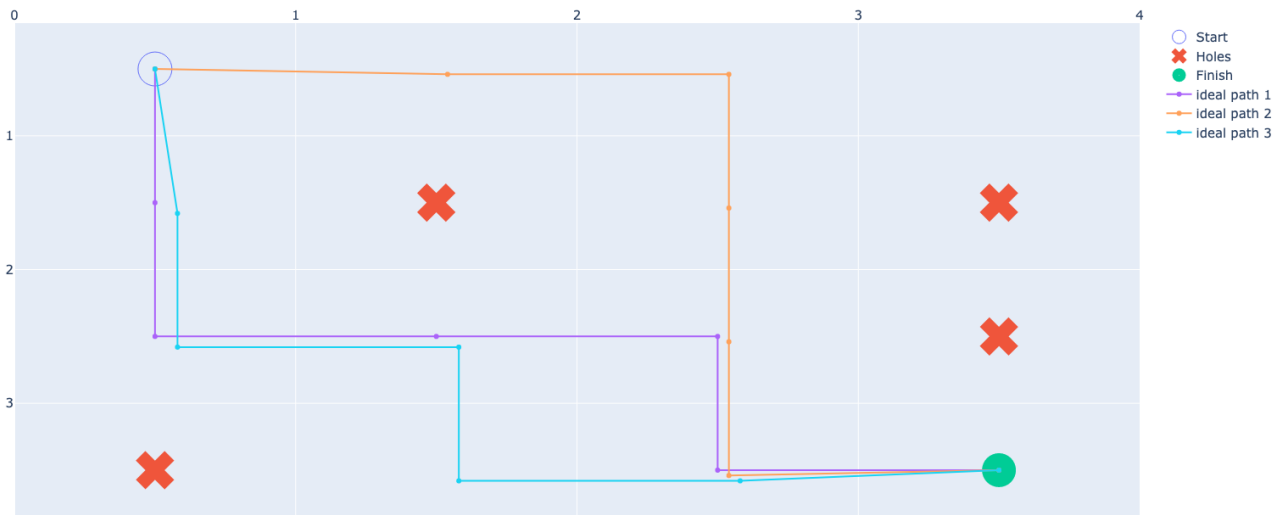
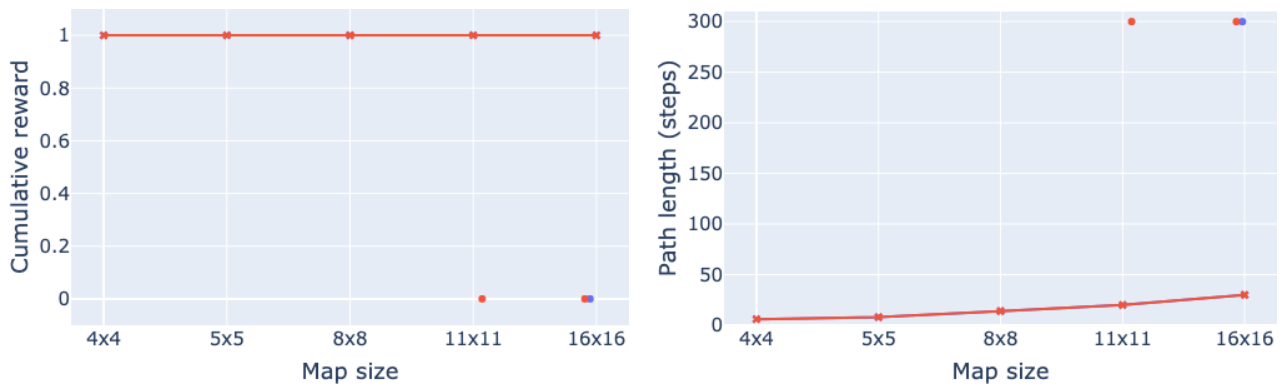


Figure 4.10: Different path produced by the policies of each architecture. Grouped according to the path taken.



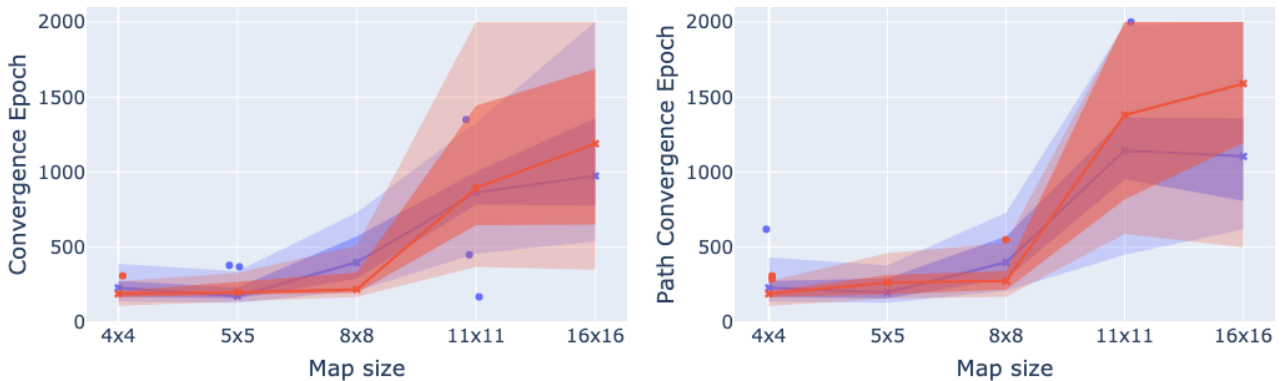
(a) Cumulative reward for Grid World environments of different map sizes.

(b) Path length of solutions over Grid World environments of different map sizes.

Figure 4.11: In red the quantum agent is shown and in blue the classical agent. Model evaluations for Grid World environments after training was completed, shown are 20 training repetitions per model type. **a)** shows the cumulative reward of each repetition, see Equation 2.9. **b)** Path length of the solution.

with the quantum agent converging faster until map size 8×8 . For larger map sizes it needs slightly more epochs, this behaviour coincides with the quantum agent having more trainable parameters than the classical agent for map sizes greater than 8×8 .

Figure 4.13 shows the behaviour of the number of trajectories needed until convergence, due to their direct connection to the convergence results the behaviour present is similar. Table 4.6 shows the trajectories used while training as a table to better observe the variation over map sizes. From the figure and the table it can be observed that both models behave similarly. For the two smallest maps the number of trajectories needed until convergence stays almost constant, and then for map size 8×8 this amount is duplicated. Between map size 8×8 and 11×11 the number of trajectories needed increases sevenfold for the quantum agent, while the classic agent only needs two and a half times more trajectories. Then, between map sizes 11×11 and 16×16 both need around 1.5 times more trajectories.



(a) Training convergence epoch for Grid World environments of different map sizes. (b) Path convergence epoch for Grid World environments of different map sizes.

Figure 4.12: In **red** the **quantum** agent and in **blue** the **classical** agent. Lines represent the median, dark shaded region represents the interquartile range, light shaded regions the min-max range, and points the outliers. Model evaluations for Grid World environments after training was completed. Over 20 training repetitions per model type. Every 10 training epochs 1 evaluation episode was carried out. This means that convergence is fulfilled when for the last 100 training episodes the corresponding evaluation episode fulfil the following criteria: **a)** epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the cumulative reward over the last 10 evaluation episodes is = 1. **b)** epoch when the model first achieved a solution with optimal length over the last 10 evaluation episodes.

Map size	4×4	5×5	8×8	11×11	16×16
classical model	$26 \cdot 10^3$	$26 \cdot 10^3$	$63 \cdot 10^3$	$160 \cdot 10^3$	$194 \cdot 10^3$
quantum model	$12 \cdot 10^3$	$15 \cdot 10^3$	$32 \cdot 10^3$	$242 \cdot 10^3$	$411 \cdot 10^3$

Table 4.6: Median of the number of trajectories processed until training convergence for different map sizes.

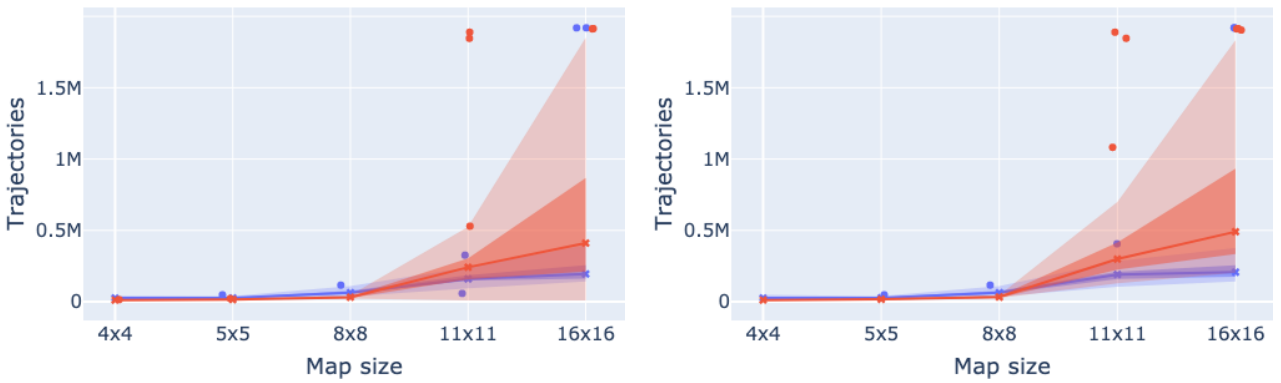
4.1.4 Effects of noise

For the investigation of the effects of noise on the VQC (Section 3.2.8) we added a depolarization gate after each gate, following the noise strategy proposed in [78]. Figure 4.14 shows part of the circuit with and without noise. The depolarization gate adds with the given probability, one random Pauli rotation to the circuit. Monte Carlo simulations are then used to calculate the expectation value of each observable.

Both environments with discrete state spaces could be solved under different noise levels, but the continuous state-space problem of Cart Pole was unsolvable with noisy circuits. Due to the long training times, instead of doing a hyperparameter search again, the best noise-free hyperparameters were retrained with different noise levels. Figure 4.15 and Figure 4.16 show the evaluation and training metrics for the quantum agent with the same hyperparameters trained on different noise levels, 10 times per noise level. After training, 100 evaluation episodes were carried out per model, as one evaluation episode is no longer enough because the agents are not deterministic any more. The convergence and environment solved criteria are the same as for Grid World-like environments. In Figure 4.15a it can be clearly observed that for noise levels $\leq 0.2\%$ the agent could solve the environment in the majority of the training repetitions. For higher noise levels, the agent could solve the environment only for a minority of the cases. And for noise levels $\geq 0.4\%$ the environment could not be solved at all. The same behaviour is reflected in the training epochs, shown in Figure 4.16a. It should be noted that small noise levels improved the training convergence.

In contrast to the cumulative reward, the results of the path length evaluation were not grouped over the 100 evalu-

4 Simulation Results



(a) Trajectories trained on until convergence for Grid World environments of different map sizes. (b) Trajectories trained on until path convergence for Grid World environments of different map sizes.

Figure 4.13: In **red** the **quantum** agent and in **blue** the **classical** agent. Lines represent the median, dark shaded region represents the interquartile range, light shaded regions the min-max range, and points the outliers. Trajectories evaluations for Grid World environments after training was completed, in other words how many training examples each model saw until convergence. Over 20 training repetitions per model type. **a)** training convergence. **b)** path convergence.

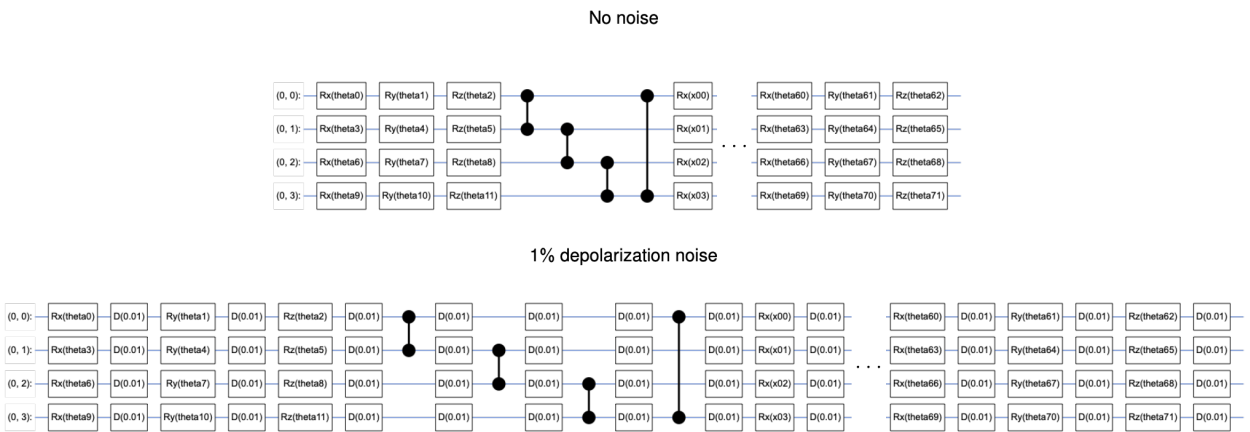


Figure 4.14: Example of the VQC ansatz with a 1% depolarization noise.

ation episodes. Instead, each individual path is shown, this was done to observe how the paths' behaviour change over noise, i.e. what happens more often: falling into holes, finding the goal, or getting stuck in a loop. Figure 4.15b shows the path length of each evaluation episode for each repetition, in total 1000 data points. Path length lower than the optimal length of 6 steps, means that the agent fell into a hole. The optimal path length is not marked since the median for all noise levels is near the optimal path length. Lastly, Figure 4.16b shows the path convergence, for all noisy levels no path convergence was achieved, only for the noise-free level 0%. Due to the non-deterministic nature of the noisy agent it was never possible to reach the goal perfectly in 100 consecutive training episodes. This indicates that the agent reached its goal, but took a suboptimal path. In Appendix B.4 Figure B.6 some examples of noisy policies are shown.

We also observed the effect of noise on grid world-like environments with map size 4×4 , 5×5 and 8×8 . All three environments could be perfectly solved with all noise levels we tested, as shown in Figure 4.17a. Although agents achieved a perfect cumulative reward, the path length of each solution suffered due to noise, as shown

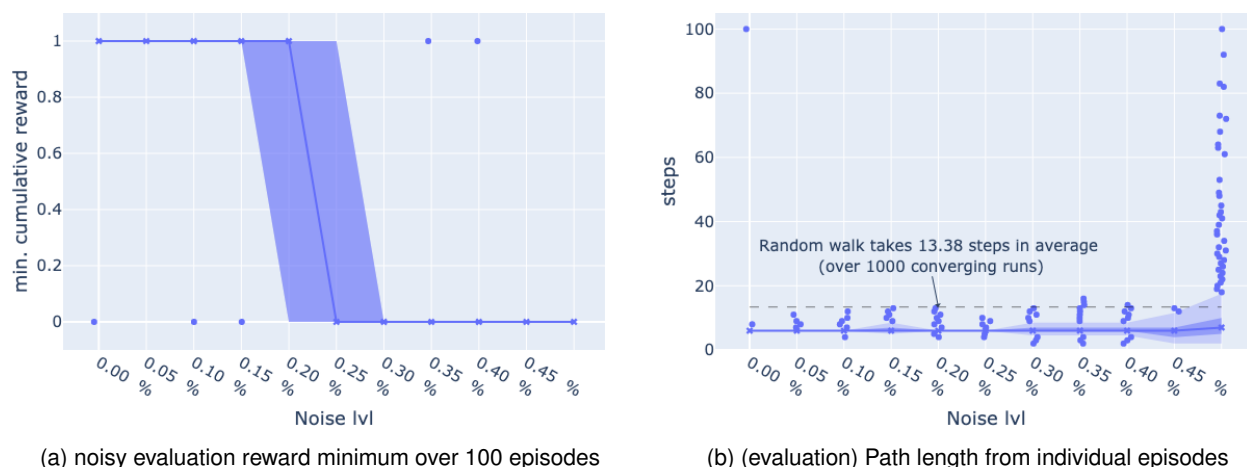


Figure 4.15: Model evaluations for the Frozen lake environment, using noisy agents after training was completed. Over 10 training repetitions per model type. The line represents the median, the dark shaded region the interquartile range, the lightly shaded region the min-max range, and outliers are represented as points. **a)** Showing the minimum cumulative reward over the episodes of each repetition (minimized since now the agents are not deterministic due to noise). Reward 1 means the environment was solved. **b)** Path length of the solution, with 6 being the optimal length. The dashed line shows how many steps a random policy would need to solve the environment.

in Figure 4.17b. Since there are no holes in the grid world environments, longer solution paths are possible. The figure shows that for the 4×4 map size, the median of the path length over all evaluation episodes increases linearly with noise levels. For both larger map sizes, the path length rises fast by about 50% for noisy agents, and then remain mostly stable. For the largest environment, both the maximum values and outliers are very high.

Figure 4.18a shows that noise improves the training convergence, in contrast Figure 4.18b shows that path convergence is not achieved for runs with noise.

In Appendix B.4 some examples of noisy policies are shown for grid world-like environments of size 4×4 , 5×5 and 8×8 in Figures B.7, B.8 and B.9 respectively.

4.1.5 Lane change

For the lane change environment, only a small hyperparameter search could be performed due to time and computation constraints. Figure 4.19 shows the evaluation history during the training of the best model (only the best repetition). After training, the weights of the training epoch with the best average cumulative reward were restored. Following the evaluation strategy of Reference [71], 3 evaluation episodes were performed every 10 training epochs, and the cumulative reward was averaged over these 3 evaluation episodes. The agent was then tested in 100 trials and was able to successfully do a lane change manoeuvre in 52% of the cases.

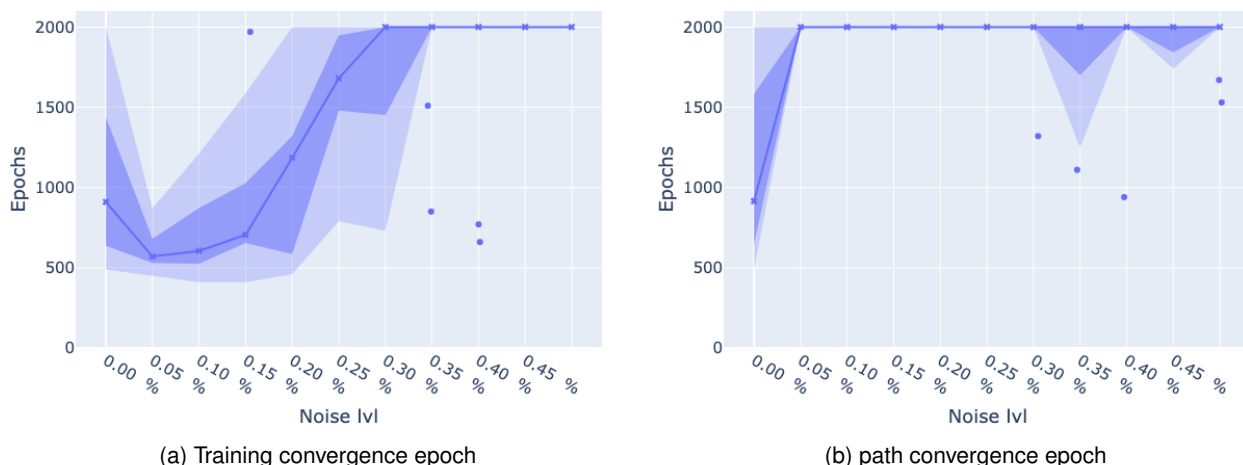


Figure 4.16: Convergence evaluations for the Frozen lake environment, using noisy agents after training was completed. Over 10 training repetitions per model type. The line represents the median, the dark shaded region the interquartile range, the lightly shaded region the min-max range, and outliers are represented as points. **a)** epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the min. cumulative reward over the last 100 evaluation episodes is $= 1$. Every 10 training epochs 10 evaluation episode was carried out. This means that convergence is fulfilled when for the last 100 training episodes the corresponding evaluation episode are all solved. **b)** epoch when the model first achieved a solution with optimal length over the last 100 evaluation episodes.

4.2 Discussion

4.2.1 Quantum vs Classic on Cart Pole

The results of Section 4.1.1 show that the quantum agent can solve the Cart Pole environment. We consider an environment as solvable by a certain architecture if the median of all models trained with the best hyperparameter choice solves this environment. It also generalises to episodes that are more than twice as long as the ones it was trained on. This confirms our hypothesis 1a that the VQC can solve environments with continuous state spaces. That a quantum agent could solve the Cart Pole environment v_1 was already shown in references [3, 65], but no previous work analysed if agents could generalise to episodes longer than they were trained on. The generalisation behaviour hints that the agent learnt the balancing task of the environment, and not just to maintain a precarious balance for the length of a training episode steps.

Due to Q-learning, even the classical results are quite unstable. Combining the instability of VQCs, especially on NISQ devices, with Q-learning leads to still more unstable results. Most notably, the results are very dependent on the hyperparameters selected, with only very few configurations achieving good results. Furthermore, they also depend heavily on the trainable parameter initialisation (as shown by the initial condition study). The instability of RL in general and particularly of ϵ -greedy policies presents a challenge. The decrease of the ϵ -greedy policy with a cosine decay improved the exploration versus exploitation balance and therefore improved long term stability. Due to the initial ϵ plateau it leads to an improved initial environment exploration, then quickly changes to stable exploitation.

Compared to the classical agents, the quantum agent behaves very similarly. All three agents solve the training environment perfectly, but as one of the classical agents solves the generalisation task and one does not, the performance difference hints at the importance of the architecture, i.e. number of hidden layers in this case. The

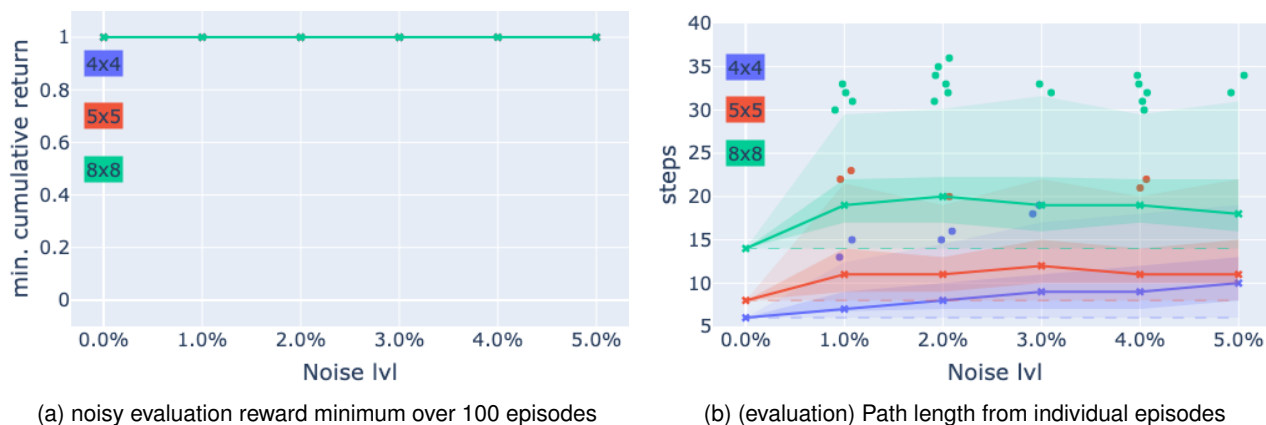


Figure 4.17: Model evaluations for grid world environments of different map sizes after training was completed. Over 10 training repetitions per model type. The line represents the median, the dark shaded region the interquartile range, the lightly shaded region the min-max range, and outliers are represented as points. **a)** Showing the minimum cumulative reward over the episodes of each repetition (minimized since now the agents are not deterministic due to noise). Reward 1 means the environment was solved. **b)** Path length of the solution, with the dashed lines marking the optimal path length for each environment size.

quantum agent can solve the generalisation environment, but a considerable number of trials still fail. In contrast to the classical agents, where the minimal cumulative reward is distributed mostly uniformly between around 200 and 500, the quantum agent's minimal cumulative reward is clustered in three groups. The clusters are narrowly distributed either at 500 for solved cases or around 375 and 325 for cases where it was not solved. This hints that the quantum agent converged to similar specific policies, but further investigation into the Q-values returned by each policy is needed to verify this, since comparing some sample output Q-values is not enough to determine the full Q-value function nor the underlying policy. No clear quantum advantage could be observed with regard to the training noisiness as the results in Table 4.1 and Table 4.2 are in a similar range with quantum models performing better in one case and classical ones in another. This contradicts our hypothesis 3.

We did not stop training when convergence was achieved to see if catastrophic forgetting occurred. For quantum agents, this happened very rarely, compared to the classical agents tested. When the quantum agents converged, they showed more stability to further training than the classical agents. The effects of catastrophic forgetting were not carried over into the evaluation since the best weights were restored. The environments tested were small enough that catastrophic forgetting did not play a role in solving them, but for more complex and longer environments this could become a significant difference and hints at a potential quantum advantage.

Of course, the comparison with the classical agent is somewhat artificial since the classical agents were constrained to the same number of trainable parameters, to use the same input and to a similar basic architecture. This was done to be able to compare both architectures, and was based on the comparisons proposed in [3]. Therefore, the comparison can not be extrapolated to quantum algorithms vs classical algorithms in general, but only applies to the selected architectures under the chosen constraints.

It can also be argued that using a classical RL learning algorithm and only changing how the Q-value approximation function is calculated from a classical Artificial Neural Network (ANN) to a VQC is also not suitable. Using an algorithm tailored to the quantum nature of VQCs would probably improve performance, but this is an open research question. However, Q-learning is a standard algorithm worth investigating due to its simplicity and it is known to work well, although it has stability caveats. To perform an initial investigation of QRL algorithms and for environ-

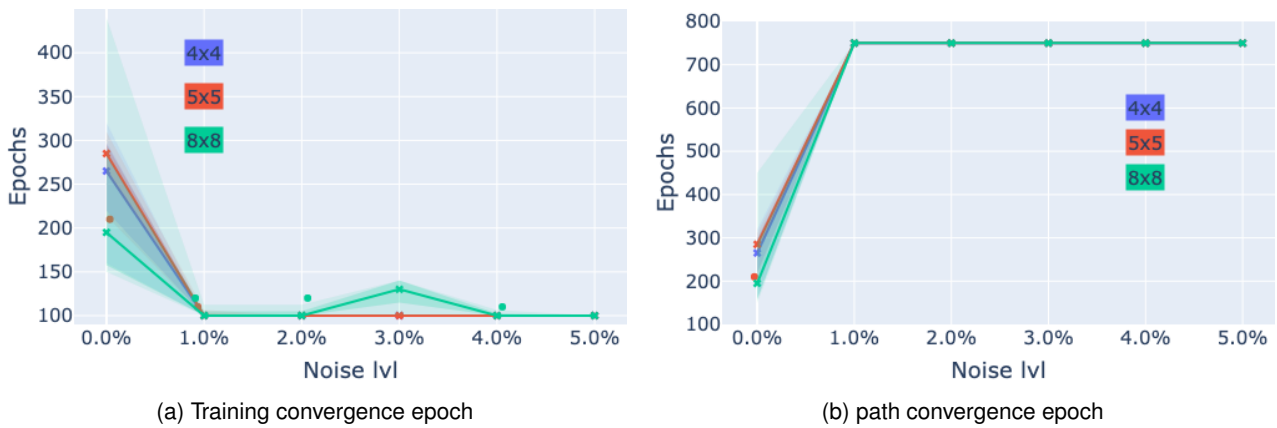


Figure 4.18: Convergence for grid world environments of different map sizes after training was completed. Over 10 training repetitions per model type. The line represents the median, the dark shaded region the interquartile range, the lightly shaded region the min-max range, and outliers are represented as points. **a)** epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the min. cumulative reward over the last 100 evaluation episodes is = 1. Every 10 training epochs 10 evaluation episode was carried out. This means that convergence is fulfilled when for the last 100 training episodes the corresponding evaluation episode are all solved. **b)** epoch when the model first achieved a solution with optimal length over the last 100 evaluation episodes.

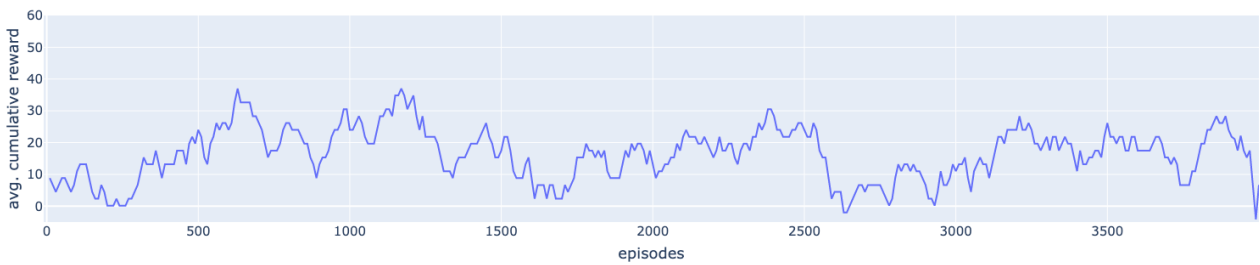


Figure 4.19: Cumulative reward per episode, averaged over the last 30 evaluations, following the reporting in [71]. Every 10 training epochs 3 evaluation episode were done, so averaging over 30 evaluation episodes means averaging over the last 100 training epochs.

ments with a discrete action space, Q-learning is still a viable option as it is able to solve these environments using only simple quantum Q-value approximators. Consistently, most of the related work uses the Q-learning algorithm [2, 3], including the lane change problem we want to solve [71].

In summary, the quantum agent could successfully solve the Cart Pole environment in both versions, with its continuous state space, discrete action space, and immediate reward, thus successfully achieving the goal of the experiment. Furthermore, the quantum agent achieved similar results to the tested classical algorithms, this corroborates our hypothesis 1a. Furthermore, it shows that Q-learning with VQCs works as expected. On the other hand, the results contradict the quantum advantage proposed in [3], and our hypothesis 3. Nevertheless, the results achieved allowed us to continue with the next subordinate hypothesis, hinting at the feasibility of solving the lane change problem.

4.2.2 Quantum vs Classic on FrozenLake

The results of Section 4.1.2 show that the quantum agent can solve the Frozen Lake environment, i.e. an environment with sparse reward which is in accordance with results of previous works and confirms hypothesis 1b.

Similar to the Cart Pole experiment, the agent is very susceptible to the choice of hyperparameters. Especially for environments with sparse reward, only very few hyperparameter combinations worked. But in contrast to the classical agents, where multiple configurations achieve results of medium quality, the quantum agent either worked excellent or not at all. This behaviour complicates the hyperparameter search since the Hyperband algorithm Ref. [80] tended to produce good results only in the last hyperparameter configurations tested. The Bayesian guided hyperparameter search failed because most configurations failed, and therefore no probability landscape could be built. On the other hand, this behaviour allows using fairly strict early stopping, since the models either converge fast or do not converge at all. If this behaviour carries on to other VQC algorithms, new types of hyperparameter searches have to be developed to account for it.

One important hyperparameter for sparse reward problems is the discount factor γ that changes how future reward is weighted, effectively setting how many steps into the future each action affects the outcome [1]. Since the optimal path length is 6 a relatively low $\gamma = 0.8$ produced the best results, i.e. the immediate rewards are preferred. This means that the agent does not search for long paths which correspond to the quite small optimal path length. Longer paths would mean to either bounce at a wall or to fall into a hole.

The quantum models converged faster and used four times fewer training examples. This can be seen as a hit towards a quantum advantage, as described in Ref. [3], since the quantum agent needs fewer data points to solve the environment. This confirms our hypothesis 3 for this environment, but not in general as for the Cart Pole environment no such advantage was found. The behaviour in the Frozen Lake environment hints at the quantum agent "remembering" better when an optimal path was found and not forgetting it. The improved "remembering" could be specific to types of environment with discrete states. However, this corresponds to the observations of reduced catastrophic forgetting events in the Cart Pole experiments.

How many trajectories are processed depends directly on how good the agent is and how fast it becomes good, since the episode length depends on the policy. If the policy is bad and the agent falls into holes quickly, the episodes are very short and few training iterations are done per epoch, this leads to the late converging classical runs. On the other hand, if the agent keeps bouncing against a wall, the episodes are long, and multiple training iterations are done, but without learning meaningful information. This leads to runs processing many trajectories and still converging late. In contrast, if the agent quickly finds the optimal path, few training iterations and trajectories are needed for convergence. The quantum agent mostly showed the latter behaviour.

As already mentioned in Section 4.2.1, the classical vs quantum comparison is only viable for the constraints imposed on the classical agent, and the learning algorithm being designed for classical agents. Additionally, the binary encoding of the state values presents the challenge that each element of the state vector is 0 or 1 for multiple different states. Thus, the agents must learn to rely on a combination of inputs and not to focus too much on single elements of the input vector. In the quantum agent, the only way in which information is shared between the elements of the input, i.e. the different quantum channels, is through entanglement as opposed to the classical agent, where each element of the hidden layer receives a weighted combination of all inputs. An adapted encoding scheme might improve the performance of the quantum agent.

In summary, the results show that the quantum agent could successfully solve the Frozen Lake environment, with its discrete state space, discrete action space, and sparse reward. Thus, successfully achieving the goal of the experiment. A potential quantum advantage of faster convergence and needing less training data was also

observed, but might depend on the specific characteristics of the architectures and hyperparameter choices. The results also hint at the importance of parameter initialisation and its instability for different environments. Lastly, the results proved our subordinate hypothesis 1b, bringing us one step closer to proving that the lane change is solvable. Furthermore, the potential quantum advantage indicates that for specific environments QRL can work better than classical RL, as was expected in our hypothesis 3.

4.2.3 Quantum Scalability

The results of Section 4.1.3 show that the quantum agent is able to solve grid world-like environments of different sizes. Thus, we confirm our hypothesis 1c. Most of the proposed VQCs architectures were tested on 4 qubits and wider circuits have not been analysed yet. In most works only the depth of the networks was analysed, i.e. layer repetition. We investigate for a fixed number of layers repetitions how the models behave when changing their width, i.e. the number of qubits used.

We use rotation input encoding (see Section 2.4.2), and therefore the number of qubit channels depends on the state vector size. Most of the environments tested in related work had state vectors of size ≤ 4 corresponding to the 4 qubit circuit width. Our results in Section 4.1.3 show that we could successfully train networks of up to 8-qubits. For bigger environments, the libraries used for simulation, especially TensorFlow Quantum (TFQ), are not yet optimised to run on acceleration devices. This leads to an exponentially growing need for computation time and resources when increasing the number of qubits. Combined with the difficult and extensive hyperparameter search, this lead to experiments using 10 and 12 qubits not converging in a reasonable time, which correspond to map sizes 32×32 and 64×64 respectively. Neither the classic nor the quantum agent could solve the 64×64 environment after a brief hyperparameter search. The reason for this convergence behaviour might be that the grid is so large that the initial random walk never finds the goal within the maximal allowed steps per episode. Nevertheless, we could successfully train an agent with an observation space twice as big as for the environments used in the previous experiments (Cart Pole and Frozen Lake).

We fixed the maximal allowed steps to be 10-times more than the optimal path length for the biggest map size tested. Since there are no holes, the optimal path is the stepped diagonal path from start to goal, this is equivalent to going along the outer walls, i.e. one example of an optimal path from top left to bottom right is always going right and then down repeatedly and this can be rearranged to going down until the bottom and then right until the goal. Therefore, the optimal path length can be calculated as $2(l - 1)$ with l the map size. We assumed that a maximum of 300 steps offers enough leeway so that the goal is still reached in exploration. While briefly experimenting with larger maps, we increased the limit 10-fold again, but this did not improve the performance.

The difference between the Frozen Lake and grid world-like environments is that the latter lack holes. This eliminates the termination condition for falling into holes, allowing the agent to reach the goal through suboptimal paths. The only two remaining terminal states are either reaching the goal or running out of steps. When comparing the training and path convergence epochs, the path convergences occurs always later than the training convergence. This means that the agent first finds a suboptimal path, but still reaches the goal, and then the solution is optimised to achieve an optimal solution path. Thus, if the agent solves the environment, it also finds the optimal path. We observed that for maps sizes larger than 8×8 the difference between path and training convergence doubles with respect to smaller maps. This behaviour could be caused by the fact that the quantum agent has more trainable parameters than the classical agent for map sizes greater than 8×8 .

Overall, both the quantum and classical agents show similar scaling behaviour. In contrast to the Frozen lake environment there is no apparent quantum advantage regarding convergence epochs. This contradicts hypothesis 3.

However, both agents can solve all environments while also finding the optimal path. For the largest two map sizes, some repetitions start to fail, hinting that the architectures used are reaching their limit. The results show that the quantum agent could successfully solve grid world-like environments with a scaling discrete state space, discrete action space, and sparse reward. Thus, successfully achieving the goal of the experiment and corroborated the last subordinate hypothesis 1c. It indicates that the agent can also process larger state space vectors.

4.2.4 Effects of noise

Before investigating the lane change problem, we first investigated the effect of noise on the VQC. We used the best hyperparameters found in the noise-free experiments and retrained the models using noisy versions of their circuits, as mentioned in Section 3.2.8. This leaves the possibility that hyperparameters that were not ideal for noise-free training could improve the results of noisy training. Furthermore, training with noise can induce barren plateaus, complicating training even more [81, 82]. The training hyperparameters might help to avoid barren plateaus [83]. Using the same hyperparameters and architecture for noise-free and noisy training depicts a possible scenario where due to the scarce access to quantum hardware: The development of an algorithm occurs in simulations and the algorithm is then tested on a NISQ quantum computer without an explicit noise examination.

The results of Section 4.1.4 show that the quantum agent can solve discrete state space environments under small noise levels, but for environments with continuous state space and higher noise levels it fails. Thus, we confirm our hypothesis 2, the QRL implementation is prone to noise.

In the Cart Pole environment, using noisy circuits led to greatly deteriorated results and to an extremely long execution time. We infer that a problem with continuous states cannot be solved with this setup on a NISQ device. For a fixed number of training epochs, the duration of training is mostly based on how many steps per episode are carried out for each environment. For the Cart Pole environment in the ideal case 200 steps per episode are taken, compared to between 6 for frozen lake and 14 for the biggest grid world used for noisy tests. In the noise experiments, the cart pole took in the median 74 steps for the noise level 0.1% and 17 steps for 1% noise. This represents both bad results and long execution times, since per step 10 Monte Carlo noise simulations were conducted.

In the Frozen Lake environment, noise can lead to falling into a hole by selecting an incorrect action and thus failing the episode. In contrast, in grid world environments the agent can do random actions until the maximal number of steps per episode is reached. In other words the agent can do some random steps at some point and still arrive at the goal, thus solving the environment, including paths with loops (see B.4). This is the main difference from the Frozen Lake environment and also the reason why grid world-like environments are still solvable under much higher noise levels. Furthermore, the lack of holes leads to improved training convergence with noise, since the random selection of actions can be seen as an exploration strategy. This is only possible because of the solving criteria only stating that the goal must be reached but not in how many steps, i.e., two paths with different length are rewarded equally if both reach the goal.

Reinforcement Learning (RL) and especially the combination of experience replay and fixed Q-values targets present additional challenges to learning with noisy circuits. Noise in a real quantum computer depends in part on the environment conditions, which leads to fluctuations over time. Therefore, for the same input, the Q-values produced by the same VQC at two times far apart can vary more than two predictions made in a short time. This leads to a possible difference between the Q-values predicted to select actions while collecting data and the Q-values produced at training time, even if the network weight did not change. This can induce problems when

experience replay is used, since in one training batch data from different episodes and therefore at different times is used. The time fluctuation behaviour is not present in the simulated noise used.

For fixed Q-value targets, the Q-values of two networks with different weights are compared. Different parameters values between both networks can cause noise to affect each computation in different ways, making it harder to learn how to correct it. For Q-learning in general, not having fixed targets leads to both the predicted value and the target value being affected by noise, making learning more difficult.

Furthermore, we did not simulate measurement noise, that is present in real hardware and usually even greater than the two-qubit gate error but only occurs once in the end. Overall, these additional error sources would lead to an even more unstable behaviour on real devices.

Hardware study The computing capabilities of NISQ devices are mainly described by their decoherence time, Quantum Volume (QV) and noise level. The decoherence time describes the time it takes for a qubit to decay from the excited state to the ground state. It relates to the time limit within which a meaningful program can be run on a NISQ device. The QV is a single-number metric that can be measured using a concrete protocol on NISQ computers [84]. The QV method quantifies the largest random circuit of equal width and depth that a quantum computer can successfully implement, and is defined as

$$\log_2 V_Q = \operatorname{argmax}_n \min(n, d(n)), \quad (4.3)$$

with n the number of qubits used, and $d(n)$ the achievable model circuit depth for a given model circuit width n , i.e. the maximal depth where the results are still useful [85]. For our analysis this can be simplified to that we need $V_Q \geq 2^l$ to even be able of encoding a state vector of size l .

The depth of a quantum circuit is defined by the qubit channel with the highest number of gates. For the implemented VQC (see Figure 3.3) with 5-layers (the number of layers used for all experiments), the circuit depth can be calculated based on the state space vector size l as follows:

$$\text{depth}(l) = \left(\underbrace{((3 + 1) \cdot 1\text{-qubit gates})}_{\text{variational + encoding block}} + \underbrace{l \cdot 2\text{-qubit gates}}_{\text{entangling structure}} \right) \cdot 5\text{-layers} + \underbrace{3 \cdot 1\text{-qubit gates}}_{\text{last variational block}} \quad (4.4)$$

$$\text{dept}(l) = 23 \cdot 1\text{-qubit gates} + 5l \cdot 2\text{-qubit gates}. \quad (4.5)$$

A single qubit gate takes much less time to run than two qubit gates. Nevertheless, we use the average gate time since before running a circuit on real hardware, it has to be decomposed into gates that can be implemented in hardware. For example, the “ibmq_montreal” quantum computer implements only the following gates: $CNOT, ID, RZ, SX, X$ [86]. The average gate time is taken over the time taken by all implementable gates, including single and two qubit gates. Therefore, it gives us a rough upper estimate of how long the circuit would take without having to exactly calculate how many single- and two-qubit gates the decomposed circuit has.

For the decoherence calculation we used the information of the IBM quantum computers with the highest QV for which the decoherence time and average gate time are available $V_Q = 128$.⁸ Thus, we can make a rough estimate of whether the decoherence time suffices to run a VQC with input vector size $l = \log_2 128 = 7$. We chose to use the data for the “ibmq_montreal” with $V_Q = 128$. The decoherence and gate time values for different

⁸See <https://cloud.ibm.com/quantum/resources/systems/> for a dashboard of available quantum computers

quantum computers fluctuate constantly. At the time of consulting the data the decoherence time was $101.12\mu s$ and the average gate time was $426.159ns$ [86]. If $\text{dept}(l) \cdot \text{average gate time} \leq \text{decoherence time}$ the circuit can be implemented from the decoherence time, but it still depends on how noisy the implemented gates are and how noise-prone the circuit is. For our rough estimate: $\text{dept}(l) \cdot \text{average gate time} = \text{dept}(7) \cdot 426.159ns = 58 \cdot 426.159ns = 24.72\mu s \leq 101.12\mu s = \text{decoherence time}$. This means that it could theoretically be implemented, but it depends on the gate and readout noise.

The newest IBM quantum computer has a $V_Q = 2^8 = 256$ [87], which means that an environment of $l = 8$ can potentially be encoded and, depending on the noise of the gates, a VQC can be implemented. The 16×16 grid world environment, the biggest we tested, is such an environment. But for encoding the lane change environment, a $V_Q = 2^{14} = 16384$ is needed to even encode the state space. It is not clear when this will be achievable since in the development maps only qubit numbers are disclosed and not the expected QV [88].

Overall, we could observe that the circuit is very prone to noise, which confirms our hypothesis 2. We also noted that the environment tested plays a significant role, especially the termination conditions for suboptimal actions. Particularly, we noted that for discrete state environments the agent is more robust to noise. Furthermore, the combination of Q-learning and noisy VQC is not ideal, since both systems on their own are already quite unstable. Lastly, for smaller VQC it is theoretically possible to run them in NISQ hardware, depending on the noisiness of the quantum computer. But, to encode larger vector sizes, as the lane change problem, the NISQ hardware has to improve a lot.

4.2.5 Lane change

After investigating the three subordinate hypotheses (1a, 1b and 1c), finally the first main hypothesis 1 was tackled. We tested whether the implemented VQC could solve the lane change problem using a Q-learning algorithm.

Only a small hyperparameter search could be performed due to time and computation constraints. The lane change environment has a 14 dimensional continuous state space, making the VQC the widest we trained and the one with the most trainable parameters. Having 326 trainable parameters is almost twice as much as the largest model trained in the quantum scalability experiments. The size of the VQC can lead to barren plateaus that hinder the training [3]. Due to the larger circuit it could be that more layer repetitions, i.e. data-reuploads, are needed. We only investigate how a discrete state space scales, and how a continuous state space scales should be investigated as future work.

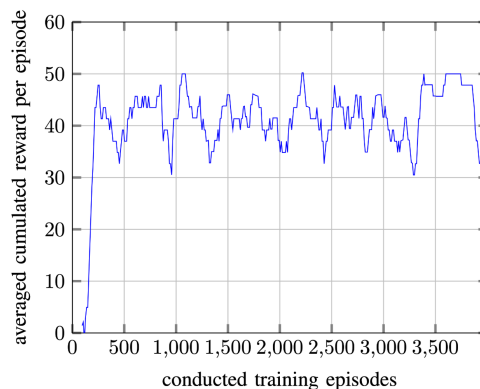


Figure 4.20: Figure 3 of Reference [71]. Cumulative reward per episode, averaged over the last 30 evaluations [71]

The results presented in Reference [71] are shown in Figure 4.20. Their proposed classical agent achieved a success rate of 93%. Our results in Section 4.1.5 show that although we did not achieve a comparable success rate, the peaks in the average cumulative reward are comparable to the median in the training curves from Reference [71]. Furthermore, the quantum agent could successfully merge in 52% of the episodes tested. The behaviour demonstrated by the quantum agent on this environment is similar to the behaviour presented for other environments when using suboptimal hyperparameters. We think that the quantum agent could solve the problem after a longer hyperparameter search is carried out. Furthermore, the hyperparameter search was performed using only half of the epochs used in Reference [71] to speed up training, and the best model found was then retrained for all 4000 epochs.

A distinct characteristic of the lane change environment is that the environment changes independently of the actions taken by the agent, i.e. the surrounding traffic adds randomness to the observations. This could also be a reason for the poor and unstable performance of the quantum agent.

Overall, we could observe a promising behaviour, and with a more profound hyperparameter search a quality comparable to the classical agent should be achievable. With the continuous development of hardware and software, the computational challenge of training and running the circuit necessary to solve this environment should decay.

5 Conclusion

The main aim of this thesis was to investigate the feasibility of solving complex learning tasks using a Quantum Reinforcement Learning (QRL) algorithm, specifically using Q-learning with a quantum deep Q-Network. We implemented a VQC suggested by related work and a Q-learning training loop with fixed Q-value targets and experience replay. Using the VQC as the Q-value approximation, we could solve different environments. Thus, building a “standard” quantum agent and training loop that can be used to solve different RL environments or tasks.

We showed that this quantum agent can solve environments with continuous or discrete state spaces, discrete action spaces, and immediate or delayed reward, and achieves similar or even better results than a classical agent on simple problems. The results for discrete state-space environments hint at the possible quantum advantage of faster convergence. We found that already moderate noise within our simplified noise model contributes significantly to the instability of the training process. This leads to problems when using NISQ hardware.

In particular, we first showed empirically that the quantum agent could successfully solve the Cart Pole training environment and also can generalise to episodes longer than it was trained on. In this way, we showed that environments with continuous state spaces and immediate reward can be solved by the quantum agent. We achieved results equal to those of comparable classical agents. However, no clear quantum advantage could be observed.

As the next step we showed that the quantum agent could successfully solve the Frozen Lake environment, with its discrete state space, discrete action space, and sparse reward. The agent achieved results better than a comparable classical agent, hinting at a potential quantum advantage of faster convergence and needing less training data. Furthermore, the potential quantum advantage indicates that for specific environments QRL can work better than classical RL.

We then showed that both the quantum and classical agents show similar scaling behaviour by investigating grid world environments with different state space sizes. Both the quantum and the classical agents solved and found the optimal path for all environment map sizes. We could train the VQCs successfully for state vector sizes up to 8, twice the maximal size used in previous works. For the largest maps, we observed a rapid increase in the number of runs that did not solve the environment, hinting that the architectures used were reaching their limit.

We then analysed how the quantum agent acted under noise, to simulate real NISQ hardware. We found that the characteristics of each environment play an important role in how stable the agent is with respect to noise. Especially if there are termination conditions for suboptimal actions, it makes training more difficult. We noted that the agent is more robust to noise for environments with discrete state. Furthermore, the combination of Q-learning and noisy VQCs is not ideal, since both systems on their own are already quite unstable. Lastly, for smaller VQC it is theoretically possible to run them on NISQ hardware, based on the needed QV and decoherence times. However, it ultimately depends on the gate and measurement noises of the quantum computer. However, to encode larger vector sizes that correspond to realistic environments, such as the lane change problem, the NISQ hardware has to improve significantly.

Lastly, we tackled the main objective of solving the lane change environment. We could observe a promising

behaviour, but due to time and computation constraints, no large enough hyperparameter search could be conducted, leading to suboptimal results. With a more profound hyperparameter search, a quality comparable to that of the classical agent should be achievable. Nevertheless, we showed that a 14 dimensional vector could be encoded using 14-qubits for the lane change problem and that we could also train this circuit. Thus, we encoded an environment three and a half times larger than the standard environments previously used in related work.

It was complicated to make a fair comparison between quantum and classical agents since the two systems are based on different principles. We decided to constrain the classical agents to the same input data and similar number of trainable parameters to make both approaches comparable. This has the effect that our conclusions cannot be easily extrapolated to quantum and classical agents in general.

The discussions of the results of this thesis were based on observations, and not on a fixed theoretical background. Due to the novelty of the field, there is no established formal mathematical approach that could be used to analyse the theoretical reasons why QML works. This is a complex and open research question, and most current work approaches the problem empirically. Furthermore, similarly to classical ML the explainability of QML is also an open question and an interesting field to investigate. Additionally, RL introduces instability further complicating the investigation of the reasons why the agent made certain decisions. However, explainable RL is an active research field that should be expanded to include QRL methods.

One problem in understanding the current QML algorithms is the difficulty of explaining which exact part influenced a decision made by the agent. The difficulty arises from quantum computations and measurement and their combinations with classical pre-and post-processing and classical optimisation. Even the simulation programs used, and even more constraints imposed by the NISQ hardware used to run the circuits (e.g., which gates can be implemented, number of qubits, QV, among others) have a high influence on the overall results. Especially when considering noise.

Another problem we encountered was the extremely long calculation time needed, especially for noisy simulation and sampling, since the circuits are not optimised to run on GPUs. But the field is rapidly advancing and the first tests to achieve acceleration on GPUs are already being carried out.⁹

We observed that the training stability of the quantum agents is very dependent on the selection of the correct hyperparameters. For suboptimal hyperparameters, repeatedly training a quantum agent on the same hyperparameter configuration leads to highly varying results, i.e. some runs fail and some converge. This hints at the importance of the initialisation of the trainable parameters. The efficient weight initialisation for VQCs is an open research question and in contrast to classical ML for VQCs only a few investigations have been carried out.

Based on these observations, we found several interesting open research questions that can be answered with the following proposed experiments. To better investigate the effect of weight initialisation a future work experiment would be to retrain a quantum agent with a fix set of initial conditions and observe if repeated training leads to different results or if they are always similar. It would be interesting to search for special initial conditions, such as purposely starting in a barren plateau, or to search through random initialisation until an excellent one is found and see if the good results are repeatable. With the objective of defining a good initialisation technique. Another option to avoid the vanishing gradient problem would be to use evolutionary learning instead of gradient base learning. Using evolutionary learning strategies in QRL is also an open research question.

The use of more complex RL algorithms is another open research question. These could improve overall performance and could also help to improve the stability with respect to noise. It would be interesting to investigate how to adapt the actor-critic paradigm to QRL and also to see if both networks can be implemented as a single

⁹See <https://github.com/tensorflow/quantum/pull/687>

circuit. Furthermore, policy gradient algorithms on VQCs have shown potential, these should also be investigated with respect to noise and bigger environments. For noisy experiments, the use of training methods proposed for supervised noisy learning and new hyperparameter searches for noisy models could improve the results.

Another interesting approach would be to combine classical environments with a quantum agent in a model-based RL approach, especially in a learning by latent imagination way. This would combine the work done on representing classical environments as quantum environments with classical model-based RL approaches.

Finally, we believe the field of QML has a lot of potential, but it is still in its infancy and more fundamental work must be done to fully understand where to look for potential advantages. In our opinion, the mere translation of classical algorithms into quantum algorithms is not going to work. But if the focus changes to developing classical-quantum hybrid algorithms from the ground up with the idea of combining classical and quantum advantages to solve ML tasks, potential advantages will emerge. It is important to learn from the enormous hype of early ML that led to the so called "AI-winters", and similarly the research waves observed in quantum computing, and not to repeat those errors with QML.

Bibliography

- [1] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. O'Reilly Media, Inc., 2019. ISBN: 1492032646.
- [2] Samuel Yen-Chi Chen et al. "Variational Quantum Circuits for Deep Reinforcement Learning". In: *IEEE Access* 8 (2020), pp. 141007–141024. DOI: 10.1109/ACCESS.2020.3010470.
- [3] Andrea Skolik, Sofiene Jerbi, and Vedran Dunjko. *Quantum agents in the Gym: a variational quantum algorithm for deep Q-learning*. 2021. DOI: 10.48550/ARXIV.2103.15084. URL: <https://arxiv.org/abs/2103.15084>.
- [4] Sofiene Jerbi et al. *Variational quantum policies for reinforcement learning*. 2021. DOI: 10.48550/ARXIV.2103.05577. URL: <https://arxiv.org/abs/2103.05577v1>.
- [5] Maria Schuld and Francesco Petruccione. "Introduction". In: *Machine Learning with Quantum Computers*. Cham: Springer International Publishing, 2021, pp. 1–21. ISBN: 978-3-030-83098-4. DOI: 10.1007/978-3-030-83098-4_1. URL: https://doi.org/10.1007/978-3-030-83098-4_1.
- [6] Maria Schuld and Francesco Petruccione. "Quantum Computing". In: *Machine Learning with Quantum Computers*. Cham: Springer International Publishing, 2021, pp. 79–146. ISBN: 978-3-030-83098-4. DOI: 10.1007/978-3-030-83098-4_3. URL: https://doi.org/10.1007/978-3-030-83098-4_3.
- [7] Eleni Ilkou and Maria Koutraki. "Symbolic Vs Sub-symbolic AI Methods: Friends or Enemies?" In: *Proceedings of the CIKM 2020 Workshops co-located with 29th ACM International Conference on Information and Knowledge Management (CIKM 2020), Galway, Ireland, October 19-23, 2020*. Ed. by Stefan Conrad and Ilaria Tiddi. Vol. 2699. CEUR Workshop Proceedings. CEUR-WS.org, 2020. URL: <http://ceur-ws.org/Vol-2699/paper06.pdf>.
- [8] Chunfeng Song et al. "Auto-encoder Based Data Clustering". en. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Ed. by David Hutchison et al. Vol. 8258. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 117–124. DOI: 10.1007/978-3-642-41822-8_15. URL: http://link.springer.com/10.1007/978-3-642-41822-8_15 (visited on 5/11/2019).
- [9] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. ISSN: 0036-8075. DOI: 10.1126/science.1127647. eprint: <https://science.sciencemag.org/content/313/5786/504.full.pdf>. URL: <https://science.sciencemag.org/content/313/5786/504>.
- [10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives". en. In: *arXiv:1206.5538 [cs]* (Apr. 2014). arXiv: 1206.5538. URL: <http://arxiv.org/abs/1206.5538> (visited on 5/11/2019).

- [12] Yasi Wang, Hongxun Yao, and Sicheng Zhao. "Auto-encoder based dimensionality reduction". In: *Neurocomputing* 184 (2016). RoLoD: Robust Local Descriptors for Computer Vision 2014, pp. 232–242. ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2015.08.104>. URL: <https://www.sciencedirect.com/science/article/pii/S0925231215017671>.
- [13] Haidong Shao et al. "A novel deep autoencoder feature learning method for rotating machinery fault diagnosis". en. In: *Mechanical Systems and Signal Processing* 95 (Oct. 2017), pp. 187–204. ISSN: 08883270. DOI: 10.1016/j.ymsp.2017.03.034. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0888327017301607> (visited on 5/11/2019).
- [14] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Comput.* 18.7 (July 2006), pp. 1527–1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. URL: <http://dx.doi.org/10.1162/neco.2006.18.7.1527>.
- [15] Yoshua Bengio et al. "Greedy Layer-wise Training of Deep Networks". In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. NIPS'06. Canada: MIT Press, 2006, pp. 153–160. URL: <http://dl.acm.org/citation.cfm?id=2976456.2976476>.
- [16] Marc'Aurelio Ranzato et al. "Efficient Learning of Sparse Representations with an Energy-based Model". In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. NIPS'06. Canada: MIT Press, 2006, pp. 1137–1144. URL: <http://dl.acm.org/citation.cfm?id=2976456.2976599>.
- [17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (6088 Oct. 1986), pp. 533–536. ISSN: 0028-0836. DOI: 10.1038/323533a0.
- [18] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". en. In: *arXiv:1511.07289 [cs]* (Feb. 2016). arXiv: 1511.07289. URL: <http://arxiv.org/abs/1511.07289> (visited on 5/11/2019).
- [19] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].
- [20] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: <http://jmlr.org/papers/v12/duchi11a.html>.
- [21] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.
- [22] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. "On the Convergence of Adam and Beyond". In: *ArXiv abs/1904.09237* (2018).
- [23] Yoshua Bengio. "Practical Recommendations for Gradient-Based Training of Deep Architectures". In: *Neural Networks: Tricks of the Trade* (2012), pp. 437–478. ISSN: 1611-3349. DOI: 10.1007/978-3-642-35289-8_26. URL: http://dx.doi.org/10.1007/978-3-642-35289-8_26.
- [24] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *ArXiv abs/1312.5602* (2013).
- [25] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (7540 Feb. 2015), pp. 529–533. ISSN: 0028-0836. DOI: 10.1038/nature14236.
- [26] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (7587 Jan. 2016), pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961.
- [27] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

- [28] Vijay Konda and John Tsitsiklis. “Actor-Critic Algorithms”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: <https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf>.
- [29] C.J.C.H. Watkins and King’s College (University of Cambridge). *Learning from Delayed Rewards*. Cambridge University, 1989. URL: <https://books.google.de/books?id=6MBgNwAACAAJ>.
- [30] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8 (3-4 May 1992), pp. 279–292. ISSN: 0885-6125. DOI: 10.1007/BF00992698.
- [31] Thomas Young. “I. The Bakerian Lecture. Experiments and calculations relative to physical optics”. In: *Philosophical Transactions of the Royal Society of London* 94 (1804), pp. 1–16. DOI: 10.1098/rstl.1804.0001. eprint: <https://royalsocietypublishing.org/doi/pdf/10.1098/rstl.1804.0001>. URL: <https://royalsocietypublishing.org/doi/abs/10.1098/rstl.1804.0001>.
- [32] Maria Schuld and Francesco Petruccione. “Representing Data on a Quantum Computer”. In: *Machine Learning with Quantum Computers*. Cham: Springer International Publishing, 2021, pp. 147–176. ISBN: 978-3-030-83098-4. DOI: 10.1007/978-3-030-83098-4_4. URL: https://doi.org/10.1007/978-3-030-83098-4_4.
- [33] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CB09780511976667.
- [34] AK Pati and SL Braunstein. “Impossibility of deleting an unknown quantum state”. English. In: *Nature* 404.6774 (Mar. 2000), pp. 164–165. ISSN: 0028-0836.
- [35] intel. *Quantum Computing | Achieving Quantum Practicality*. 2020. URL: <https://www.intel.com/content/www/us/en/research/quantum-computing.html> (visited on 13/05/2022).
- [36] Xanadu. *Photonic Quantum Hardware*. 2021. URL: <https://www.xanadu.ai/hardware> (visited on 13/05/2022).
- [37] John Clarke and Frank K. Wilhelm. “Superconducting quantum bits”. In: *Nature* 453 (7198 June 2008), pp. 1031–1042. ISSN: 0028-0836. DOI: 10.1038/nature07128.
- [38] Google Quantum AI. *Quantum Computer Datasheet*. 2021. URL: <https://quantumai.google/hardware/datasheet/weber.pdf> (visited on 13/05/2022).
- [39] IBM Research Blog. *Eagle’s Quantum Performance Progress*. 2021. URL: <https://research.ibm.com/blog/eagle-quantum-processor-performance> (visited on 13/05/2022).
- [40] Rigetti Computing. *Building Scalable, Innovative Quantum Systems*. 2021. URL: <https://www.rigetti.com/what-we-build> (visited on 13/05/2022).
- [41] Anton Frisk Kockum and Franco Nori. “Quantum Bits with Josephson Junctions”. In: *Fundamentals and Frontiers of the Josephson Effect*. Ed. by Francesco Tafuri. Cham: Springer International Publishing, 2019, pp. 703–741. ISBN: 978-3-030-20726-7. DOI: 10.1007/978-3-030-20726-7_17. URL: https://doi.org/10.1007/978-3-030-20726-7_17.
- [42] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM J. Comput.* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 0097-5397. DOI: 10.1137/S0097539795293172. URL: <https://doi.org/10.1137/S0097539795293172>.
- [43] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. “The quest for a Quantum Neural Network”. In: *Quantum Information Processing* 13.11 (Aug. 2014), pp. 2567–2586. DOI: 10.1007/s11128-014-0809-8. URL: <https://doi.org/10.1007/s11128-014-0809-8>.

- [44] Hartmut Neven et al. *Training a Large Scale Classifier with the Quantum Adiabatic Algorithm*. 2009. DOI: 10.48550/ARXIV.0912.0779. URL: <https://arxiv.org/abs/0912.0779>.
- [45] Manuela Weigold et al. “Expanding Data Encoding Patterns For Quantum Algorithms”. In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. 2021, pp. 95–101. DOI: 10.1109/ICSA-C52384.2021.00025.
- [46] Manuela Weigold et al. “Data Encoding Patterns for Quantum Computing”. In: *Proceedings of the 27th Conference on Pattern Languages of Programs*. PLoP ’20. Virtual Event: The Hillside Group, 2020. ISBN: 9781941652169.
- [47] Alberto Peruzzo et al. “A variational eigenvalue solver on a photonic quantum processor”. In: *Nature Communications* 5 (1 Sept. 2014), p. 4213. ISSN: 2041-1723. DOI: 10.1038/ncomms5213.
- [48] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A Quantum Approximate Optimization Algorithm”. In: (Nov. 2014).
- [49] Maria Schuld and Francesco Petruccione. “Variational Circuits as Machine Learning Models”. In: *Machine Learning with Quantum Computers*. Cham: Springer International Publishing, 2021, pp. 177–215. ISBN: 978-3-030-83098-4. DOI: 10.1007/978-3-030-83098-4_5. URL: https://doi.org/10.1007/978-3-030-83098-4_5.
- [50] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. “Effect of data encoding on the expressive power of variational quantum-machine-learning models”. In: *Phys. Rev. A* 103 (3 Mar. 2021), p. 032430. DOI: 10.1103/PhysRevA.103.032430. URL: <https://link.aps.org/doi/10.1103/PhysRevA.103.032430>.
- [51] Adrián P’erez-Salinas et al. “Data re-uploading for a universal quantum classifier”. In: *Quantum* 4 (2020), p. 226.
- [52] Kosuke Mitarai et al. “Quantum circuit learning”. In: *Physical Review A* (2018).
- [53] Jarrod R. McClean et al. “Barren plateaus in quantum neural network training landscapes”. In: *Nature Communications* 9.1 (Sept. 2018), p. 4812. ISSN: 2041-1723. DOI: 10.1038/s41467-018-07090-4. URL: <https://doi.org/10.1038/s41467-018-07090-4>.
- [54] A. Hamann, V. Dunjko, and S. Wölk. *Quantum-accessible reinforcement learning beyond strictly epochal environments*. 2020. DOI: 10.48550/ARXIV.2008.01481. URL: <https://arxiv.org/abs/2008.01481>.
- [55] Sofiène Jerbi et al. “Parametrized Quantum Policies for Reinforcement Learning”. In: *NeurIPS*. 2021.
- [56] Owen Lockwood and Mei Si. “Reinforcement Learning with Quantum Variational Circuit”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16.1 (Oct. 2020), pp. 245–251. URL: <https://ojs.aaai.org/index.php/AIIDE/article/view/7437>.
- [57] Matthias Wolff et al. “Towards a Quantum Mechanical Model of the Inner Stage of Cognitive Agents”. In: *2018 9th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. 2018, pp. 000147–000152. DOI: 10.1109/CogInfoCom.2018.8639892.
- [58] Alexey A. Melnikov, Adi Makmal, and Hans J. Briegel. “Projective simulation applied to the grid-world and the mountain-car problem”. In: (May 2014).
- [59] Hans J. Briegel and Gemma De las Cuevas. “Projective simulation for artificial intelligence”. In: *Scientific Reports* 2 (1 Dec. 2012), p. 400. ISSN: 2045-2322. DOI: 10.1038/srep00400.

- [60] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: <https://doi.org/10.1145/237814.237866>.
- [61] Th Sriarunothai et al. “Speeding-up the decision making of a learning agent using an ion trap quantum processor”. In: *Quantum Science and Technology* 4.1 (Dec. 2018), p. 015014. DOI: 10.1088/2058-9565/aaef5e. URL: <https://doi.org/10.1088/2058-9565/aaef5e>.
- [62] A. Hamann, V. Dunjko, and S. Wölk. “Quantum-accessible reinforcement learning beyond strictly epochal environments”. In: *Quantum Machine Intelligence* 3 (2 Dec. 2021), p. 22. ISSN: 2524-4906. DOI: 10.1007/s42484-021-00049-7.
- [63] V. Saggio et al. “Experimental quantum speed-up in reinforcement learning agents”. In: *Nature* 591 (7849 Mar. 2021), pp. 229–233. ISSN: 0028-0836. DOI: 10.1038/s41586-021-03242-7.
- [64] Sergio Guadarrama et al. *TF-Agents: A library for Reinforcement Learning in TensorFlow*. <https://github.com/tensorflow/agents>. [Online; accessed 25-June-2019]. 2018. URL: <https://github.com/tensorflow/agents>.
- [65] Tensorflow Quantum. *Parametrized Quantum Circuits for Reinforcement Learning*. 2022. URL: https://www.tensorflow.org/quantum/tutorials/quantum_reinforcement_learning#3_deep_q-learning_with_pqc_q-function_approximators (visited on 18/05/2022).
- [66] Abhinav Kandala et al. “Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets”. In: *Nature* 549 (7671 Sept. 2017), pp. 242–246. ISSN: 0028-0836. DOI: 10.1038/nature23879.
- [67] Cirq Developers. *Cirq*. Version v0.12.0. See full list of authors on Github: <https://github.com/quantumlib/Cirq/graphs/contributors>. Aug. 2021. DOI: 10.5281/zenodo.5182845. URL: <https://doi.org/10.5281/zenodo.5182845>.
- [68] Michael Broughton et al. *TensorFlow Quantum: A Software Framework for Quantum Machine Learning*. 2020. DOI: 10.48550/ARXIV.2003.02989. URL: <https://arxiv.org/abs/2003.02989>.
- [69] Albin Cassirer et al. *Reverb: A Framework For Experience Replay*. 2021. arXiv: 2102.04736 [cs.LG].
- [70] Klaus Greff et al. “The Sacred Infrastructure for Computational Research”. In: *Proceedings of the 16th Python in Science Conference*. Ed. by Katy Huff et al. 2017, pp. 49–56. DOI: 10.25080/shinma-7f4c6e7-008.
- [71] Matthias Nichting, Thomas Lobig, and Frank Köster. “Case Study on Gap Selection for Automated Vehicles Based on Deep Q-Learning”. In: *2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST)*. 2021, pp. 252–257. DOI: 10.1109/ICAICST53116.2021.9497818.
- [72] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).
- [73] Gym Documentation. *Cart Pole*. 2022. URL: https://www.gymnasium.ml/environments/classic_control/cart_pole/ (visited on 14/05/2022).
- [74] Gym Documentation. *Frozen Lake*. 2022. URL: https://www.gymnasium.ml/environments/toy_text/frozen_lake/ (visited on 15/05/2022).
- [75] Daniel Heß et al. *ADORE: Automated driving open research*. 2022. URL: <https://github.com/eclipse/adore> (visited on 15/05/2022).
- [76] Daniel Heß et al. “Fast Maneuver Planning for Cooperative Automated Vehicles”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 1625–1632. DOI: 10.1109/ITSC.2018.8569791.

- [77] Farrokh Vatan and Colin P. Williams. "Realization of a General Three-Qubit Quantum Gate". In: *arXiv: Quantum Physics* (2004).
- [78] Tensorflow Quantum. *Noise*. 2022. URL: <https://www.tensorflow.org/quantum/tutorials/noise> (visited on 18/05/2022).
- [79] Tensorflow Quantum. *Noisy Controlled Parametrized Quantum Circuits*. 2022. URL: https://www.tensorflow.org/quantum/api_docs/python/tfq/layers/NoisyControlledPQC (visited on 18/05/2022).
- [80] Lisha Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: (2016). DOI: 10.48550/ARXIV.1603.06560. URL: <https://arxiv.org/abs/1603.06560>.
- [81] Samson Wang et al. "Noise-induced barren plateaus in variational quantum algorithms". In: *Nature Communications* 12 (1 Dec. 2021), p. 6961. ISSN: 2041-1723. DOI: 10.1038/s41467-021-27045-6.
- [82] María Cerezo et al. "Cost-Function-Dependent Barren Plateaus in Shallow Quantum Neural Networks". In: *ArXiv abs/2001.00550* (2020).
- [83] Ankit Kulshrestha and Ilya Safro. "BEINIT: Avoiding Barren Plateaus in Variational Quantum Algorithms". In: *ArXiv abs/2204.13751* (2022).
- [84] Lev Bishop et al. "Quantum Volume". In: 2017.
- [85] Andrew W. Cross et al. "Validating quantum computers using randomized model circuits". In: *Physical Review A* (2019).
- [86] IBM quantum. *ibmq_montreal*. 2022. URL: https://cloud.ibm.com/quantum/resources/systems/ibmq_montreal (visited on 24/05/2022).
- [87] IBM quantum. *Pushing quantum performance forward with our highest Quantum Volume yet*. 2022. URL: <https://research.ibm.com/blog/quantum-volume-256> (visited on 24/05/2022).
- [88] IBM quantum. *Expanding the IBM Quantum roadmap to anticipate the future of quantum-centric supercomputing*. 2022. URL: <https://research.ibm.com/blog/ibm-quantum-roadmap-2025> (visited on 24/05/2022).
- [89] Tom O'Malley et al. *KerasTuner*. <https://github.com/keras-team/keras-tuner>. 2019.

A Supplementary material for the experiments

A.1 Grid World like environments

Additional Grid World environments used for the scalability and noise experiment. In Figure A.1 a grid world of size 8×8 is shown, and in Figure A.2 a grid world of size 16×16 .



Figure A.1: Grid world like environment with map size of 8×8

A.2 Hyperparameter searches

An extensive hyperparameter search was carried out for each scenario, using a Hyperband search algorithm [80] and the Keras Tuner library [89]. The code is available upon request.

A.2.1 Setup and objectives

For each hyperparameter configuration, the models were trained and evaluated five times using the metrics described in each experimental setup. Except for noise experiments, where it was repeated four times, and for the lane change experiments, where only two repetitions were carried out. The number of trained epochs depends on the Hyperband search algorithm. Using this search algorithm, the models are trained in a bracket competition fashion for a subset of epochs. The models with the highest objective metric are then selected and further trained [80]. The objective to maximize in the hyperparameter search was defined as follows for each environment with Q_1 the lower quartile.:

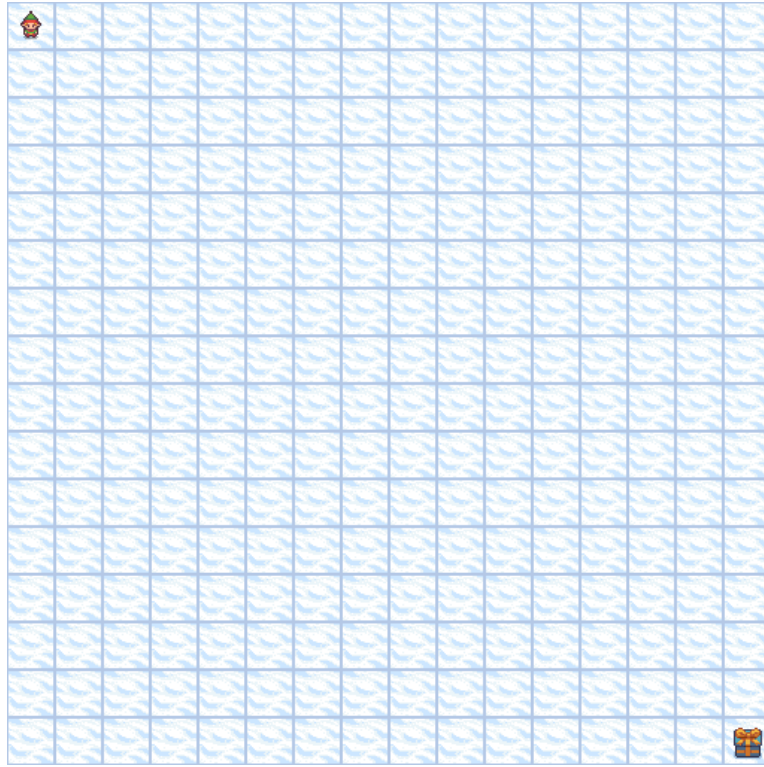


Figure A.2: Grid world like environment with map size of 16×16

Cart Pole environment

$$objective = \frac{Q_1(\text{min. cum. rew.})}{200} + \frac{Q_1(\text{min. cum. rew.})}{500} + \frac{Q_1(n\text{-epochs} - \text{convergence epoch})}{n\text{-epochs}}, \quad (\text{A.1})$$

min. cum. rew. means minium cumulative reward, see Equation 4.1. 200 and 500 represent the maximum cumulative reward of the cart pole environment v_0 and v_1 respectively.

Frozen Lake environment

$$objective = \frac{Q_1(\text{cum. rew.})}{1} + \frac{Q_1(100 - \text{path length})}{100} + \frac{Q_1(n\text{-epochs} - \text{convergence epoch})}{n\text{-epochs}}, \quad (\text{A.2})$$

cum. rew. means cumulative reward, see Equation 2.9. 100 is the maximal number of steps allowed in the Frozen lake environment.

Grid world environment

$$objective = \frac{Q_1(\text{cum. rew.})}{1} + \frac{Q_1(300 - \text{path length})}{300} + \frac{Q_1(n\text{-epochs} - \text{convergence epoch})}{n\text{-epochs}}, \quad (\text{A.3})$$

cum. rew. means cumulative reward, see Equation 2.9. 300 is the maximal number of steps allowed in the grid world like environments.

Lane change environment

In this case the objective was the percent of successful lane change manoeuvres, averaged over all training repetitions.

A.2.2 Search spaces and selected hyperparameters

Table A.1 shows the meaning of each hyperparameter over which was searched.

Hyperparameter explanation	
steps	Maximal number of steps per episode.
batch_size	number of samples shown to optimizer at each update.
decay_steps	Over how many steps the ϵ -decay happens.
gamma	discount factor for Q-learning
layers	Classical layer architecture
n_layers	Number of quantum layer repetitions
loss	loss function
lr_1	For classical agent the learning rate. For quantum agents learning rate for the input scaling
lr_2	Learning rate for the variational trainable parameters
lr_3	Learning rate for the output scaling
collect_steps_per_iteration	time steps after which the main model is updated
updates_per_target_update	main model updates after which the main model parameters are copied to the target model
observables_fn	Observables used for the quantum measurement

Table A.1: Description of hyperparameter considered in this work

Cart Pole environment

Hyperparameter	quantum	classical 1 hidden layer	classical 2 hidden layer
batch_size	[16, 32, 64]	[16, 32 , 64]	[16, 32, 64]
decay_steps	[150, 500 , 1000]	[150, 500, 1000]	[150, 500, 1000]
gamma	0.99	0.99	0.99
loss	[MSE, Huber]	[MSE, Huber]	[MSE, Huber]
lr_1	[0.01, 0.001 , 0.0001]	[0.01 , 0.001, 0.0001]	[0.01 , 0.001, 0.0001]
lr_2	[0.01, 0.001 , 0.0001]	-	-
lr_3	[0.1 , 0.01, 0.001]	-	-
collect_steps_per_iteration	[10 , 30, 90]	[10 , 30, 90]	[10 , 30, 90]
updates_per_target_update	[3, 10]	[3, 10]	[3, 10]
observables_fn	$a_0 = Z_0 Z_1, a_1 = Z_2 Z_3$	-	-

Table A.2: Hyperparameter used for the Cart Pole environment. In bold the best found hyperparameters

Frozen Lake environment

Table A.3 shows the hyperparameter search space and selected hyperparameters for the frozen lake environment.

Hyperparameter	quantum	classical 1 hidden layer
batch_size	[16 , 32, 64]	[16, 32 , 64]
decay_steps	[150, 500 , 1000]	[150 , 500, 1000]
gamma	[0.2, 0.5, 0.8, 0.95 , 0.99]	[0.2, 0.5, 0.8, 0.95 , 0.99]
loss	[Huber]	[MSE, Huber]
lr_1	[0.01, 0.001 , 0.0001]	[0.01 , 0.001, 0.0001]
lr_2	[0.01 , 0.001, 0.0001]	-
lr_3	[0.1, 0.01, 0.001]	-
collect_steps_per_iteration	[10 , 30,]	[10 , 30, 90]
updates_per_target_update	[3 , 10]	[3 , 10]
observables_fn	$a_0 = Z_0, a_1 = Z_1, a_2 = Z_2, a_3 = Z_3$	-

Table A.3: Hyperparameter used for the Frozen Lake environment. In bold the best found hyperparameters

Grid world environment

Table A.3 shows the hyperparameter search space for grid world environments, and Table B.1 shows the best hyperparameters selected.

Hyperparameter	quantum	classical 1 hidden layer
batch_size	[16, 32]	[16, 32, 64]
decay_steps	[150, 500, 1000]	[150, 500, 1000]
gamma	[0.8, 0.95]	[0.8, 0.95]
loss	[Huber]	[Huber]
lr_1	[0.001, 0.0001]	[0.01, 0.001, 0.0001]
lr_2	[0.01, 0.001]	-
lr_3	[0.1, 0.01, 0.001]	-
collect_steps_per_iteration	[10, 30,]	[10]
updates_per_targpdate	[3, 10]	[3, 10]
observables_fn	$a_0 = Z_0, a_1 = Z_1, a_2 = Z_2, a_3 = Z_3$	-

Table A.4: Hyperparameter used for grid world environments. In bold the best found hyperparameters

Lane change environment

Table A.5 shows the hyperparameter search space for the lane change environment.

Hyperparameter	quantum
batch_size	[16, 32, 64]
decay_steps	[150, 500, 1000]
gamma	[0.2, 0.5, 0.8, 0.95, 0.99]
loss	[MSE, Huber]
lr_1	[0.01, 0.001, 0.0001]
lr_2	[0.01, 0.001, 0.0001]
lr_3	[0.1, 0.01, 0.001]
collect_steps_per_iteration	[10, 30,]
updates_per_target_update	[3, 10]
observables_fn	$a_0 = Z_0 Z_1, a_1 = Z_2 Z_3, a_2 = Z_4 Z_5, a_3 = Z_6 Z_7$

Table A.5: Hyperparameter used for the Lane change environment.

B Supplementary material for the results

B.1 Quantum vs Classic on CartPole

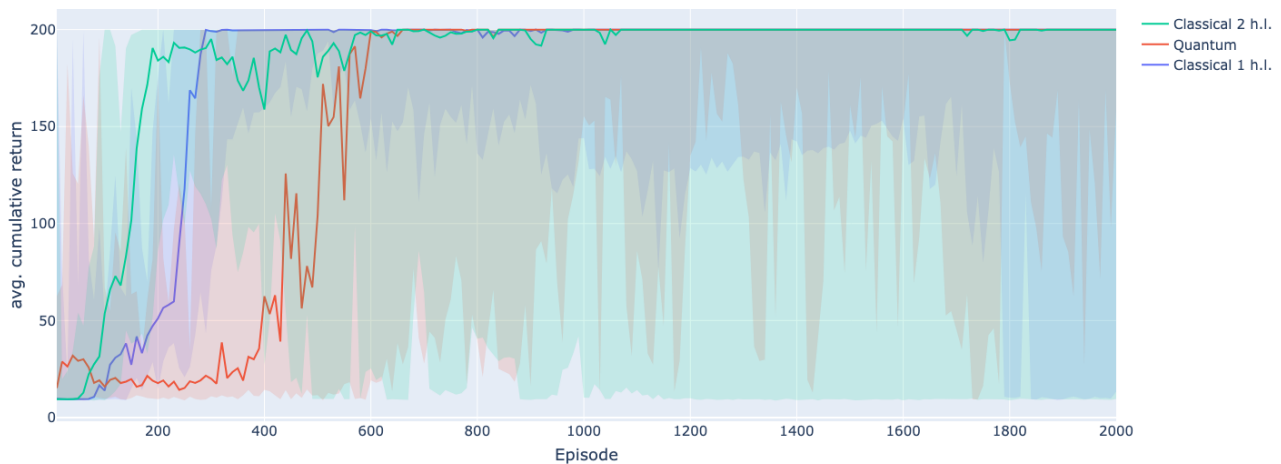


Figure B.1: For the 20 models trained per type, the median of the cumulative reward of the evaluations over the training history is shown, for the v_0 Cart Pole environment. Every 10 training epochs 10 evaluation episodes were carried out. The line represents the median over the 10 evaluation episodes over the 20 repetitions, and the shaded region represents the minimal and maximal values. In **green** the classical model with two hidden layers is represented, in **blue** the classical model with one hidden layers, and in **red** the quantum model.

Figure B.1 show the evaluation results during training, using the average cumulative reward, Equation 4.2. The line represents the median of the average cumulative reward for 20 repetitions. In this figure, it can be seen that both classical agents start to learn faster. However, the quantum agent remains more stable once it converges. Figure B.2 shows the same, but for the generalisation environment, cart pole v_1 . In this case, the quantum agent and the classical agent with two hidden layers achieve the best results.

B.2 Quantum vs Classic on Frozen Lake

Figure B.3 and Figure B.4 show the cumulative reward and path-length histories during training over all training repetitions. Figure B.5 shows the stability investigation for the Frozen Lake environment. All metrics remain mostly constant, independent of how many training repetitions are taken.

Figure B.3 show the evaluation results during training, using the cumulative reward, Equation 2.9. The line represents the median of the cumulative reward for 20 repetitions. In this figure, it can be seen that the quantum agent learns faster and converges faster than the classical agent. Figure B.4 shows the same but for the path length. In

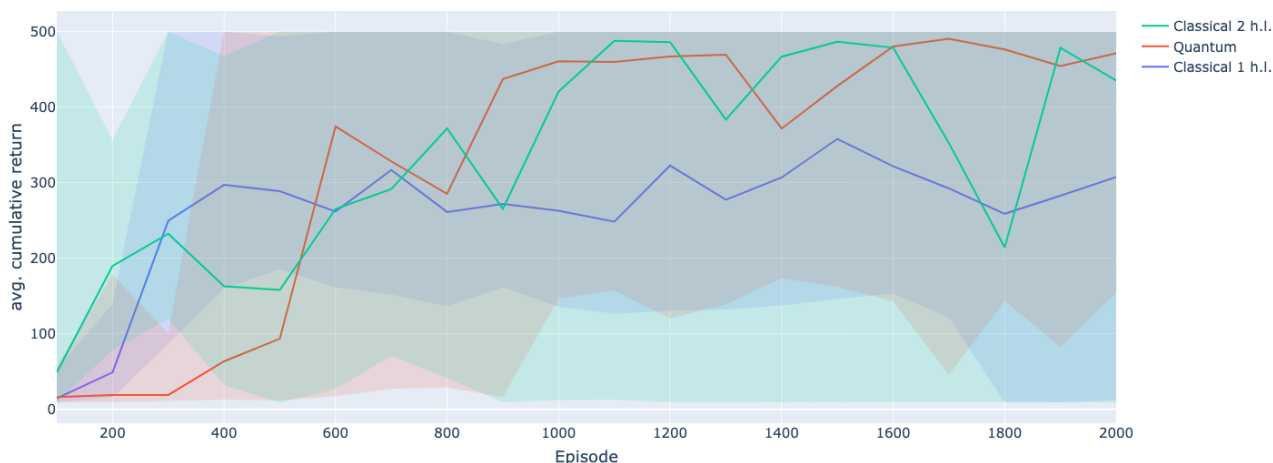


Figure B.2: For the 20 models trained per type, the median of the cumulative reward of the evaluations over the training history is shown, for the v_1 Cart Pole environment. Every 10 training epochs 10 evaluation episodes were carried out. The line represents the median over the 10 evaluation episodes over the repetitions 20, and the shaded region represents the minimal and maximal values. In **green** the classical model with two hidden layers is represented, in **blue** the classical model with one hidden layers, and in **red** the quantum model.

this case the quantum agent and the classical agent achieve a short path at comparable epochs, but the quantum agent converges to a lower path length faster.

B.3 Quantum scalability

Table B.1 shows hyperparameters used for both agent architectures in the scalability experiment.

B.4 Effects of noise

Here, examples of solutions for different policies are shown. For each noise level, the training repetition with a median objective value was selected. Two paths per training repetition per noise level are shown. Examples for the Frozen Lake environment are shown in Figure B.6, and for grid world-like environments of size 4, 5 and 8 in Figures B.7, B.8 and B.9 respectively.

type	classical	quantum (4x4, 5x5, 8x8, 16x16)	quantum (11x11)
amsgrad	TRUE		
steps	300		
batch_size	32		
collect_steps_per_iteration	10		
decay_steps	500		1000
epsilon	1		
epsilon_min	0.01		
eval_interval	10		
gamma	0.95		
initial_random_steps	100		
is_slippery	FALSE		
layers	[13]	-	
n_layers	-	5	
loss	huber		
lr_1	0.01	0.0001	
lr_2	-	0.01	
lr_3	-	0.001	0.01
n_actions	4		
n_episodes	2000		
n_state		Obs_vector_size	7
num_eval_episodes	1		
parallel_episodes	2		
quantum	FALSE	TRUE	
replay_buffer_max_length	10000		
updates_per_target_update	10		
use_gpu	FALSE		
entangle_func	-	entangling_layer_CZ	
observables_fn	-	obs_fn_Z0_a0_Z1_a1_Z2_a2_Z3_a3	

Table B.1: Best hyperparameters found for quantum and classical agents. The classical agent used the same hyperparameters for all map sizes. The quantum agent had to use a special set of hyperparameters for the map size 11.

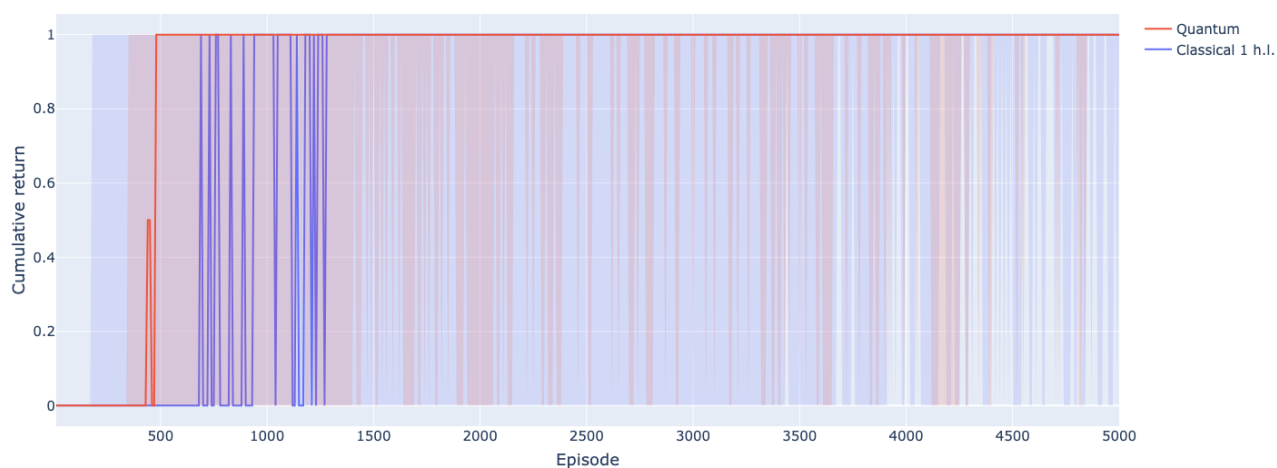


Figure B.3: For the 20 models trained per type, the median of the cumulative reward of the evaluations over the training history is shown, for the Frozen Lake environment. Every 10 training epochs 1 evaluation episode was carried out. The line represents the median over the evaluation episode of the 20 repetitions, and the shaded region represents the minimal and maximal values. In **red** the quantum model is represented, in **blue** the classical model.

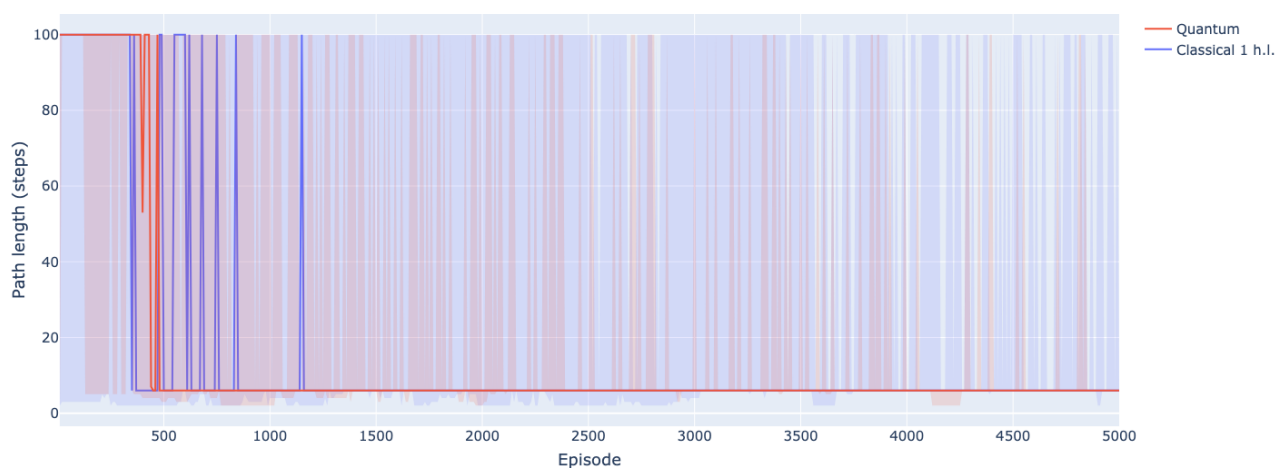


Figure B.4: For the 20 models trained per type, the median of the cumulative reward of the evaluations over the training history is shown, for the Frozen Lake environment. Every 10 training epochs an evaluation episode was performed. The line represents the median over the evaluation episode over the 20 repetitions, and the shaded region represents the minimal and maximal values. In **red** the quantum model is represented, in **blue** the classical model.

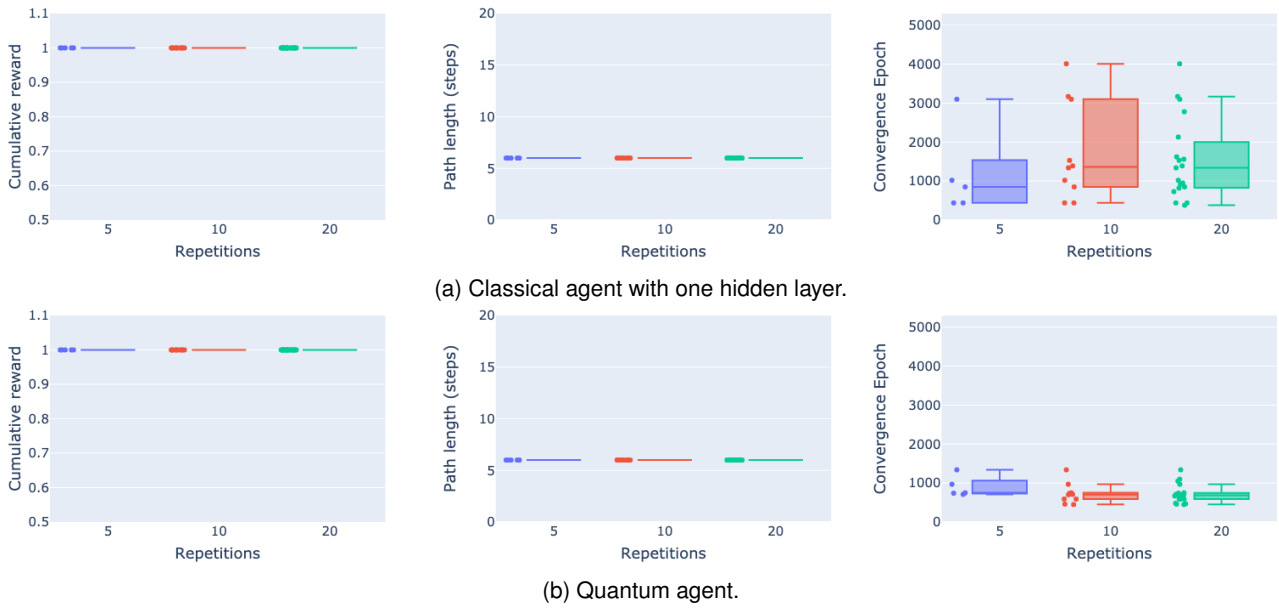


Figure B.5: Initial condition study with 5, 10 and 20 repetition. The final evaluation of cumulative reward, path length and the convergence epoch are shown for different subsets of trained models. For the Frozen Lake environment.

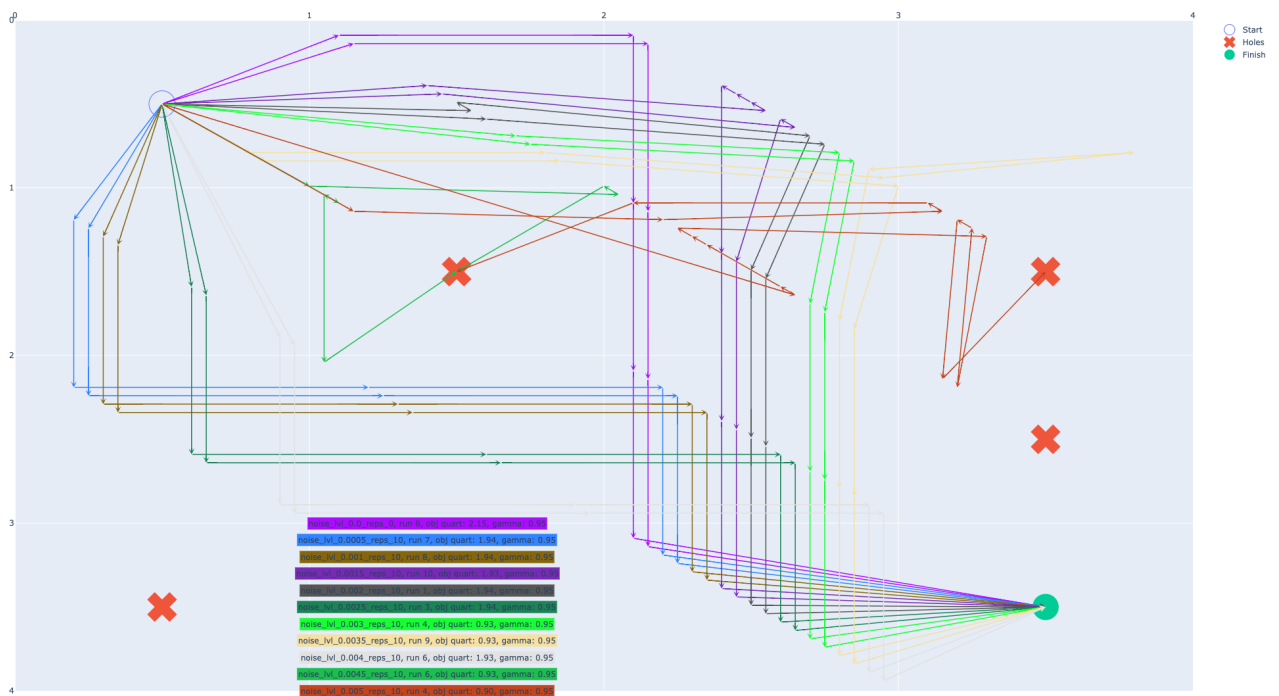


Figure B.6: Different path produced by the policies of each noise level. Only two examples are shown for each noise level. The arrows pointing to a hole means that the agent failed to solve the environment by falling into a hole.

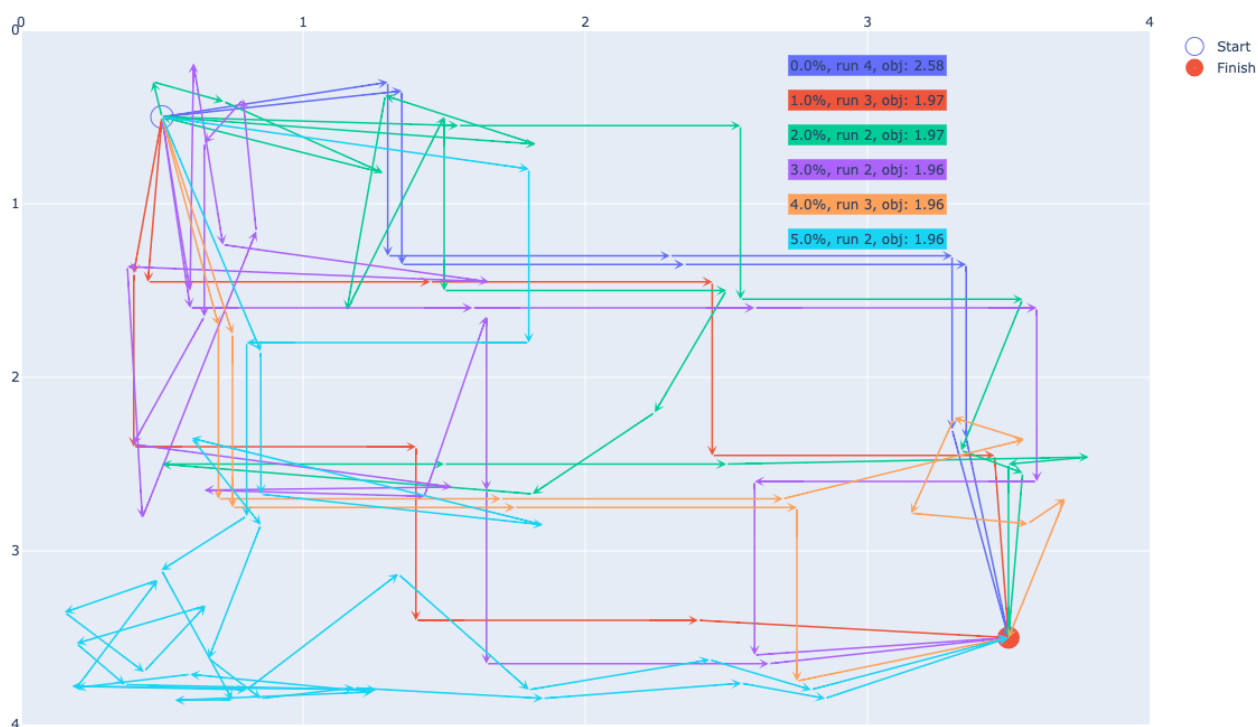


Figure B.7: Different path produced by the policies of each noise level, on the Grid World-like environment of map size 4. Only two examples are shown for each noise level.

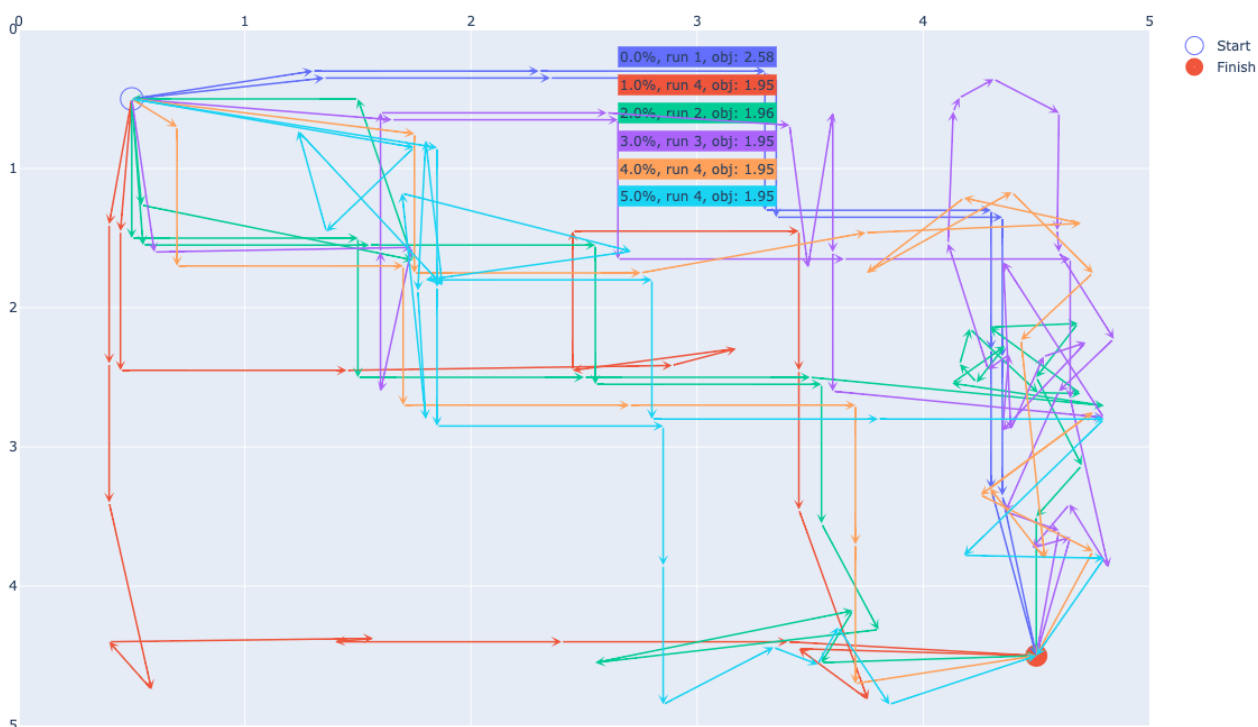


Figure B.8: Different path produced by the policies of each noise level, on the Grid World-like environment of map size 5. Only two examples are shown for each noise level.

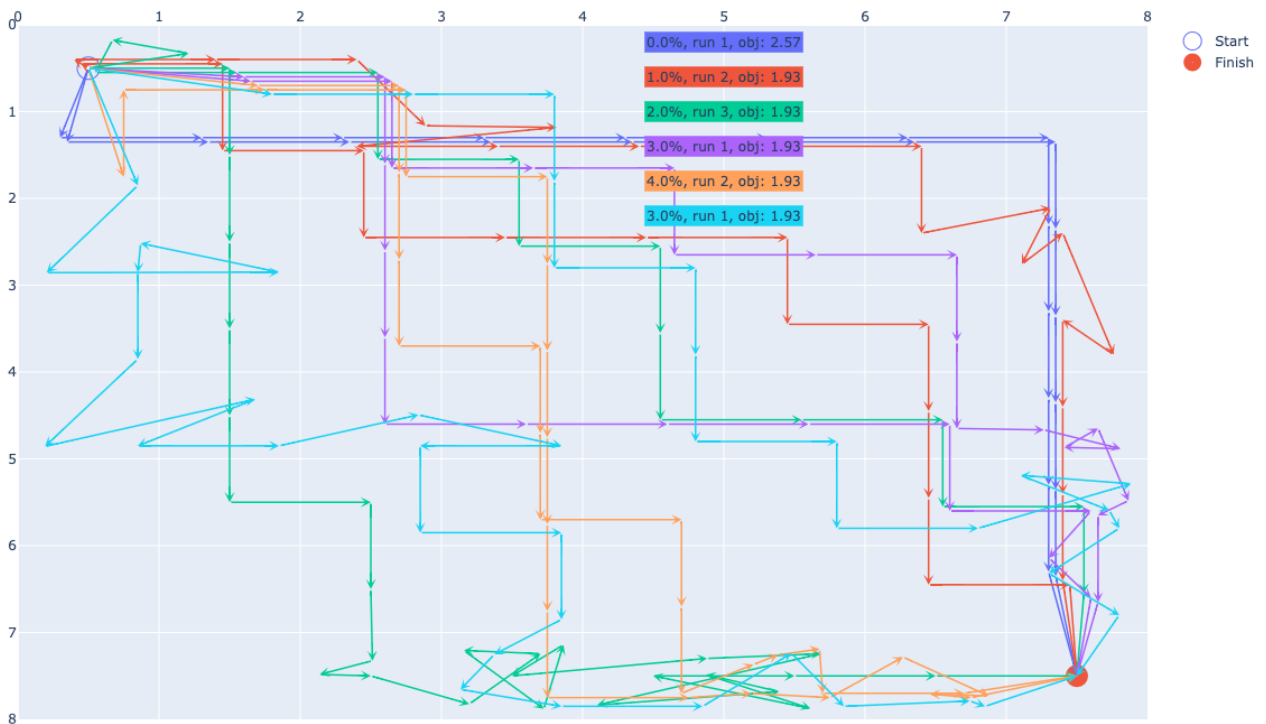


Figure B.9: Different path produced by the policies of each noise level, on the Grid World-like environment of map size 8. Only two examples are shown for each noise level.

List of Figures

2.1	Activation functions	6
2.2	Loss functions as $Loss(x)$ with x representing the difference between target and prediction. In blue the MSE and in orange the Huber loss with $\delta = 1$	7
2.3	Agent-environment interaction scheme. Image reproduced from [27].	9
2.4	Diagram for processing tasks based on classical data. The encoding and readout steps can be highly non-trivial and take considerable runtime. Adapted from [32].	15
2.5	The Bloch sphere representation of a qubit state. Adapted from [41].	16
2.6	The four possibilities to combine quantum or classical data and quantum or classical processing devices. This work falls into the highlighted sector CQ. Image and description modified from [5].	20
2.7	Principle of a VQC, the circuit depends on parameters and a cost function that evaluates the expected measurement for a given set of parameters. The computational problem is encoded as a cost minimisation. Normally, training is carried out iteratively and in each step the cost function is consulted to find better parameters θ . Usually, the optimisation is done on a classical computer. Image and description modified from [6]	23
2.8	Illustration of the relation between the VQC structure and its expressivity based on partial Fourier series [50].	25
3.1	RL training architecture with classic environment, and either classic or hybrid (classic data input and quantum computation) Network, extracted from [1] and modified.	29
3.2	ansatz used in this work, also called layer, adapted from [3, 4, 65, 66].	31
3.3	Variational Quantum Circuit (VQC) structure used for Q-value predictions. Adapted from [65]	31
3.4	Standard environments used.	34
3.5	ADORe lane change environment. The yellow car represents the agent and the black cars the surrounding traffic. The ego vehicle is marked in yellow [71].	36
4.1	Model evaluations for the Cart Pole v_i environment after training. Showing the minimal cumulative reward over 100 evaluation episodes, see Equation 4.1. 20 training repetitions have been conducted per model type. The red dashed line represents the lower threshold for solving each environment according to its specification.	41
4.2	Training metrics over 20 models, per model type. a) epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the average cumulative reward over the last 100 evaluation episodes is ≥ 195 . Every 10 training epochs 10 evaluation episode are carried out. b) how many trajectories each model use until convergence, in other words how many training examples each model saw until convergence.	42

4.3	Prototypical examples of the evaluation history in the training environment (v_0) are shown grouped according to its behaviour. The classification was done based on visual inspection of the learning curves. Per model 20 training repetitions were carried out. Every 10 training epochs, 10 evaluation episodes were carried out. The lines represent the mean and the whiskers the maximal and minimal value over the 10 episodes.	43
4.4	Prototypical examples of the evaluation history in the generalisation environment (v_1) are shown grouped according to its behaviour. The classification was done based on visual inspection of the learning curves. Per model 20 training repetitions were carried out. Every 10 training epochs, 10 evaluation episodes were carried out. The lines represent the mean and the error bars the maximal and minimal value over the 10 episodes.	44
4.5	Initial condition study with 5, 10 and 20 models trained with the same hyperparameters. The average cumulative reward for training and generalisation environment and the convergence epoch are shown for different subsets of trained models.	45
4.6	Model evaluations for the Frozen lake environment after training. 20 training repetitions per model type are shown. a) Showing the cumulative reward of each repetition, see Equation 2.9. Reward 1 means the environment was solved. b) Path length of the solution, with 6 being the optimal length and shown by the red dashed line.	46
4.7	Training metrics over 20 models, per type. a) epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the cumulative reward over the last 10 evaluation episodes is = 1. Every 10 training epochs 1 evaluation episode was carried out. This means that convergence is fulfilled when for the last 100 training episodes the corresponding evaluation episode b) how many trajectories each model use until convergence, in other words how many training examples each model saw until convergence. This is influenced by hyperparameters, such as batch size.	47
4.8	Prototypical examples of the cumulative reward evaluation history in the training environment are shown grouped according to their behaviour. The classification was done based on visual inspection of the learning curves. Per model 20 training repetitions were carried out. Every 10 training epochs, 1 evaluation episodes was carried out.	48
4.9	Prototypical examples of the path length evaluation history in the training environment. The classification was done based on visual inspection of the learning curves. Per model 20 training repetitions were carried out. Every 10 training epochs, 1 evaluation episode was carried out.	49
4.10	Different path produced by the policies of each architecture. Grouped according to the path taken.	50
4.11	In red the quantum agent is shown and in blue the classical agent. Model evaluations for Grid World environments after training was completed, shown are 20 training repetitions per model type. a) shows the cumulative reward of each repetition, see Equation 2.9. b) Path length of the solution.	50
4.12	In red the quantum agent and in blue the classical agent. Lines represent the median, dark shaded region represents the interquartile range, light shaded regions the min-max range, and points the outliers. Model evaluations for Grid World environments after training was completed. Over 20 training repetitions per model type. Every 10 training epochs 1 evaluation episode was carried out. This means that convergence is fulfilled when for the last 100 training episodes the corresponding evaluation episode fulfil the following criteria: a) epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the cumulative reward over the last 10 evaluation episodes is = 1. b) epoch when the model first achieved a solution with optimal length over the last 10 evaluation episodes.	51

4.13	In red the quantum agent and in blue the classical agent. Lines represent the median, dark shaded region represents the interquartile range, light shaded regions the min-max range, and points the outliers. Trajectories evaluations for Grid World environments after training was completed, in other words how many training examples each model saw until convergence. Over 20 training repetitions per model type. a) training convergence. b) path convergence.	52
4.14	Example of the VQC ansatz with a 1% depolarization noise.	52
4.15	Model evaluations for the Frozen lake environment, using noisy agents after training was completed. Over 10 training repetitions per model type. The line represents the median, the dark shaded region the interquartile range, the lightly shaded region the min-max range, and outliers are represented as points. a) Showing the minimum cumulative reward over the episodes of each repetition (minimized since now the agents are not deterministic due to noise). Reward 1 means the environment was solved. b) Path length of the solution, with 6 being the optimal length. The dashed line show how many steps a random policy would need to solve the environment.	53
4.16	Convergence evaluations for the Frozen lake environment, using noisy agents after training was completed. Over 10 training repetitions per model type. The line represents the median, the dark shaded region the interquartile range, the lightly shaded region the min-max range, and outliers are represented as points. a) epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the min. cumulative reward over the last 100 evaluation episodes is = 1. Every 10 training epochs 10 evaluation episode was carried out. This means that convergence is fulfilled when for the last 100 training episodes the corresponding evaluation episode are all solved. b) epoch when the model first achieved a solution with optimal length over the last 100 evaluation episodes.	54
4.17	Model evaluations for grid world environments of different map sizes after training was completed. Over 10 training repetitions per model type. The line represents the median, the dark shaded region the interquartile range, the lightly shaded region the min-max range, and outliers are represented as points. a) Showing the minimum cumulative reward over the episodes of each repetition (minimized since now the agents are not deterministic due to noise). Reward 1 means the environment was solved. b) Path length of the solution, with the dashed lines marking the optimal path length for each environment size.	55
4.18	Convergence for grid world environments of different map sizes after training was completed. Over 10 training repetitions per model type. The line represents the median, the dark shaded region the interquartile range, the lightly shaded region the min-max range, and outliers are represented as points. a) epoch when the model first fulfilled the solving criteria for the training environment, i.e. when the min. cumulative reward over the last 100 evaluation episodes is = 1. Every 10 training epochs 10 evaluation episode was carried out. This means that convergence is fulfilled when for the last 100 training episodes the corresponding evaluation episode are all solved. b) epoch when the model first achieved a solution with optimal length over the last 100 evaluation episodes.	56
4.19	Cumulative reward per episode, averaged over the last 30 evaluations, following the reporting in [71]. Every 10 training epochs 3 evaluation episode were done, so averaging over 30 evaluation episodes means averaging over the last 100 training epochs.	56
4.20	Figure 3 of Reference [71]. Cumulative reward per episode, averaged over the last 30 evaluations [71]	61
A.1	Grid world like environment with map size of 8×8	72
A.2	Grid world like environment with map size of 16×16	73

B.1	For the 20 models trained per type, the median of the cumulative reward of the evaluations over the training history is shown, for the v_0 Cart Pole environment. Every 10 training epochs 10 evaluation episodes were carried out. The line represents the median over the 10 evaluation episodes over the 20 repetitions, and the shaded region represents the minimal and maximal values. In green the classical model with two hidden layers is represented, in blue the classical model with one hidden layers, and in red the quantum model.	77
B.2	For the 20 models trained per type, the median of the cumulative reward of the evaluations over the training history is shown, for the v_1 Cart Pole environment. Every 10 training epochs 10 evaluation episodes were carried out. The line represents the median over the 10 evaluation episodes over the repetitions 20, and the shaded region represents the minimal and maximal values. In green the classical model with two hidden layers is represented, in blue the classical model with one hidden layers, and in red the quantum model.	78
B.3	For the 20 models trained per type, the median of the cumulative reward of the evaluations over the training history is shown, for the Frozen Lake environment. Every 10 training epochs 1 evaluation episode was carried out. The line represents the median over the evaluation episode of the 20 repetitions, and the shaded region represents the minimal and maximal values. In red the quantum model is represented, in blue the classical model.	80
B.4	For the 20 models trained per type, the median of the cumulative reward of the evaluations over the training history is shown, for the Frozen Lake environment. Every 10 training epochs an evaluation episode was performed. The line represents the median over the evaluation episode over the 20 repetitions, and the shaded region represents the minimal and maximal values. In red the quantum model is represented, in blue the classical model.	80
B.5	Initial condition study with 5, 10 and 20 repetition. The final evaluation of cumulative reward, path length and the convergence epoch are shown for different subsets of trained models. For the Frozen Lake environment.	81
B.6	Different path produced by the policies of each noise level. Only two examples are shown for each noise level. The arrows pointing to a hole means that the agent failed to solve the environment by falling into a hole.	81
B.7	Different path produced by the policies of each noise level, on the Grid World-like environment of map size 4. Only two examples are shown for each noise level.	82
B.8	Different path produced by the policies of each noise level, on the Grid World-like environment of map size 5. Only two examples are shown for each noise level.	82
B.9	Different path produced by the policies of each noise level, on the Grid World-like environment of map size 8. Only two examples are shown for each noise level.	83

List of Tables

2.1	Examples of useful single-qubit logic gates, modified from [6, 33]	17
2.2	Examples of useful multi-qubit logic gates, modified from [6, 33]	18
2.3	Adapted from [45, 46]. x_i represents a scalar, X a vector and $b_i \in \{0, 1\}$. The graphics can be found in https://quantumcomputingpatterns.org/	21
3.1	The state space of the Cart Pole environment [27] is described by a 4-dimensional vector.	34
3.2	Elements of the state vector, the values are normalised and bounded to $[0, 1]$ see [71].	36
3.3	Model structures of the agents used in the Cart Pole environment.	37
3.4	Description of Grid world-like environment used. With n.t.p.: number of trainable parameters.	39
4.1	For each model, the evaluation history for the training environment (v_0) can be grouped according to their behaviour. The classification was done based on visual inspection of the learning curves. Prototypical examples are shown in Figure 4.3. Per model 20 training repetitions were carried out.	42
4.2	For each model, the evaluation history for the generalisation environment (v_1) can be grouped according to their behaviour. Classification was done based on visual inspection of the learning curves. Prototypical examples are shown in Figure 4.4. Per model 20 training repetitions were carried out.	44
4.3	For each model, the evaluation history for the training environment can be grouped according to their behaviour. The classification was done based on visual inspection of the learning curves. Prototypical examples are shown in Figure 4.8. Per model 20 training repetitions were carried out.	46
4.4	Path convergence history for the training environment v_0 , grouped according to their behaviour. Classification was done based on visual inspection of the learning curves. Prototypical examples are shown in Figure 4.9.	47
4.5	The solution paths produced by the agents are grouped according to the path followed. The labelling is carried over from Figure 4.10	48
4.6	Median of the number of trajectories processed until training convergence for different map sizes.	51
A.1	Description of hyperparameter considered in this work	74
A.2	Hyperparameter used for the Cart Pole environment. In bold the best found hyperparameters	74
A.3	Hyperparameter used for the Frozen Lake environment. In bold the best found hyperparameters	75
A.4	Hyperparameter used for grid world environments. In bold the best found hyperparameters	75
A.5	Hyperparameter used for the Lane change environment.	76
B.1	Best hyperparameters found for quantum and classical agents. The classical agent used the same hyperparameters for all map sizes. The quantum agent had to use a special set of hyperparameters for the map size 11.	79

Name: M. Lautaro Hickmann

Matriculation number: 953591

Honesty disclaimer

I hereby affirm that I wrote this thesis independently and that I did not use any other sources or tools than the ones specified.

Ulm, 30/05/2022

M. Lautaro Hickmann