OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

LABOR FÜR
DIGITALISIERUNG

Technical University of Applied Sciences Regensburg
Laboratory for Digitalisation

Master Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science (M.Sc.)

Uncovering Instabilities in Variational-Quantum Deep Q-Networks

Student Name: Maja Franz
Student Number: 3310113

Primary Supervising Professor: Prof. Dr. Wolfgang Mauerer
Secondary Supervising Professor: Prof. Dr. Kai Selgrad

Submission Date: May 23, 2023

Abstract

Quantum computing holds the promise of achieving computational speed-ups by exploiting the principles of quantum mechanics. While certain quantum algorithms are proven to have theoretical speed-ups, the practical utility of these algorithms is limited on current noisy intermediate-scale quantum (NISQ) devices due to imperfections, noise and limited qubit capacity. Despite these limitations, variational hybrid quantum-classical algorithms are often proposed as a means to reap quantum advance from NISQ devices.

This thesis focuses on the empirical analysis of a recent class of variational hybrid approaches to quantum reinforcement learning, specifically variational quantum deep Q-Learning (VQ-DQL). We show that VQ-DQL approaches are subject to instabilities that cause the learned policy to diverge, study the extent to which this afflicts reproduciblity of established results based on classical simulation, and perform systematic experiments to identify potential explanations for the observed instabilities. Additionally, we validate the VQ-DQL algorithm on an actual quantum processing unit and investigate differences in behaviour between simulated and physical quantum systems, which suffer from imperfections. The experiments demonstrate that it is inconclusive whether known quantum approaches, especially on current NISQ devices, provide an advantage over classical approaches. Nevertheless, these findings highlight promising areas for future research, particularly in exploring the potential of a *hardware-software co-design* approach.

Kurzfassung

Quantencomputing verspricht, durch die Nutzung der Prinzipien der Quantenmechanik, Beschleunigungen gegenüber klassischen Rechnungen zu erzielen. Obwohl bestimmte Quantenalgorithmen theoretisch effizienter sind als klassische Ansätze, ist die praktische Anwendbarkeit auf aktuellen, sog. *Noisy Intermediate-Scale Quantum* (NISQ) Geräten aufgrund von Imperfektionen auf Grund von Rauschen und begrenzter Qubit-Kapazität eingeschränkt. Trotz dieser Einschränkungen werden variationale hybride quanten-klassische Algorithmen häufig als Möglichkeit vorgeschlagen, um von NISQ-Systemen einen quantenbasierten Vorteil zu erreichen.

In dieser Arbeit wird eine Klasse von variationalen hybriden Ansätzen für quantenbasiertes *Reinforcement Learning*, i.e. *Variational Quantum Deep Q-Learning* (VQ-DQL) experimentell untersucht. Es wird gezeigt, dass VQ-DQL-Ansätze instabiles Verhalten aufzeigen und zu einer Divergenz einer gelernten reinforcement learning Strategie führen können. Es werden systematische Experimente durchgeführt, um potenzielle Erklärungen für die beobachteten Instabilitäten zu identifizieren und Auswirkungen auf die Reproduzierbarkeit etablierter Ansätze abzuschätzen. Zudem wird der VQ-DQL-Algorithmus auf einem realen Quantencomputer validiert und es werden Unterschiede im Verhalten zwischen simulierten und physischen Quantensystemen untersucht. Die Experimente zeigen, dass es nicht eindeutig entschieden werden kann, ob bekannte Ansätze für VQ-DQL, insbesondere auf aktuellen NISQ-Systemen, einen Vorteil gegenüber klassischen Ansätzen bieten. Dennoch zeigen diese Ergebnisse vielversprechende Richtungen für zukünftige Forschung auf, insbesondere in der Erforschung des Potenzials eines *Hardware-Software-Co-Design*-Ansatzes.

Contents

1. Introduction **1**

2. Background on Quantum Computing **3**

2.1. Qubits and Superpositions . . . 3
2.2. Multiple Qubits and Entanglement . . . 4
2.3. Quantum Computation . . . 5
2.3.1. Single-Qubit Gates . . . 6
2.3.2. Multi-Qubit Gates . . . 7
2.4. Variational Quantum Algorithms . . . 8
2.4.1. Classical Data Encoding . . . 9
2.4.2. Quantum Data Extraction . . . 10
2.4.3. Gradient Calculation . . . 10
2.5. Modelling Noise . . . 11
2.5.1. Mixed Quantum States . . . 11
2.5.2. Operations in Noisy Quantum Systems . . . 12
2.5.3. Noise Models . . . 12

3. Background on Quantum Reinforcement Learning **15**

3.1. Markov Decision Process . . . 15
3.1.1. Rewards . . . 15
3.1.2. Policies . . . 16
3.2. Q-Learning . . . 16
3.2.1. Dynamic Programming . . . 16
3.2.2. Temporal-Difference Learning . . . 17
3.3. Deep Q-Learning . . . 17
3.3.1. Updating Weights with Gradient Descent . . . 18
3.3.2. Experience Replay . . . 18
3.3.3. Target Network . . . 19
3.4. Variational Quantum Deep Q-Learning . . . 19
3.4.1. Input Encoding . . . 19
3.4.2. Q-Value Extraction . . . 20

4. Related Work **21**

4.1. Deep Q-Learning and its instabilities . . . 21
4.2. Quantum Reinforcement Learning . . . 21

5. Experimental Analysis: Variational Quantum Deep Q-Learning **23**

5.1. The `CartPole` Environment . . . 23

5.2. Reproduction study 24
5.3. Methodology 26
5.4. Experiments with Ideal Quantum Simulators 27
5.4.1. Encoding and Extraction Methods 28
5.4.2. Cross-Validation 29
5.5. Comparison to a Classical Neural Network 32
5.6. Experiments with Noisy Quantum Systems 33
5.6.1. Validation on IBM Quantum Device 33
5.6.2. Training with Noisy Quantum Simulators 35

6. Discussion and Outlook **39**

7. Conclusion **41**

A. Validation Results **43**

B. Cross-Validation – Full Results **45**
B.1. Baseline without data re-uploading 45
B.2. Baseline with data re-uploading 53

List of Figures **61**

List of Tables **61**

Bibliography **63**

1. Introduction

The field of quantum computing aims to exploit the properties of quantum mechanics in order to achieve computational speed-ups. It has indeed been shown that certain problems can theoretically be solved more efficiently using quantum algorithms than classical algorithms. Two well-known examples are Grover's algorithm [1] and Shor's algorithm [2], which respectively provide a quadratic speed-up for the exploration of unstructured search spaces and efficiently solve prime factorization and discrete logarithm problems.

Quantum computers, utilising diverse physical implementations, have already been built by various vendors. However, the current generation of quantum computers, called noisy intermediate-scale quantum (NISQ) systems [3], is prone to errors, such as decoherence caused by interactions with the environment and errors that arise during the execution of quantum operations. These errors limit the capabilities of current quantum devices. Additionally, the number of qubits in current machines is limited, imposing restrictions on the size of problems that can be solved on them.

Nonetheless, some quantum approaches offer potential advantages over classical approaches, even in the NISQ era. Specifically, variational hybrid quantum-classical algorithms are well-suited for gate-based quantum machines in the near term [4]. These algorithms perform only a limited number of steps on a quantum computer and the remaining steps on classical machines, making it more feasible to take advantage of quantum computing while using current quantum systems.

For instance, one particular class of these variational hybrid algorithms is quantum machine learning (QLM). Moving parts of a classical machine learning algorithm to a quantum computer could reduce parameter and sampling complexity, as studies suggest [5], [6]. It is however unlikely that quantum computing will be able to handle large amounts of data in the near future due to the absence of quantum RAM [4], [7]. Therefore, quantum reinforcement learning (QRL) techniques, which only require small data points for training, but often need to explore large search spaces, are promising machine learning methods for achieving quantum advantage on NISQ devices.

Despite significant advancements in classical reinforcement learning (RL) [8] over the past decade [9]–[16], it requires large computational resources to match or exceed human performance even on simple tasks like playing arcade video games. For instance, Badia et al. [17] spent approximately 53,000 hours of training, distributed over 256 machines, to achieve superhuman performance on all 57 Atari games of the Arcade Learning Environment benchmark [18]. The learning dynamics of these methods are not yet fully understood, and they remain a subject of current research [19]–[22]. Therefore, it is natural to explore whether quantum computing could offer a potential speed-up in this domain.

The goal of this thesis is to analyse quantum *deep Q-Learning* [6], [23]–[25], i.e. a class of recent variational hybrid approaches to QRL, and compare it to its classical counterpart. To investigate the approach under the limitations of current NISQ devices, comparative experiments are performed using

noise models [26], which mimic the imperfections of near-term quantum systems.

Structure The remainder of this thesis is structured as follows: Chapter 2 gives an overview on quantum computing and further describes how the noise of currently available quantum systems can be modelled. Chapter 3 introduces the RL approach of deep Q-Learning, together with its quantum analogon. Chapter 4 describes how this work relates to prior research. Chapter 5 describe the experiments conducted using these RL approaches and present results, which are discussed in Chapter 6. Finally, the work is summarised in Chapter 7.

Credit This work, specifically the Chapters 3, 4, 5 and 6 all share material with an article, published in the Journal of the Franklin Institute, "Uncovering Instabilities in Variational-Quantum Deep Q-Networks", authored by Maja Franz, Lucas Wolf, Maniraman Periyasamy, Christian Ufrecht, Daniel D. Scherer, Axel Plinge, Christopher Mutschler and Wolfgang Mauerer [6].

2. Background on Quantum Computing

This chapter provides an overview of the main concepts of quantum computing. It begins by introducing single qubit systems and superpositions in Section 2.1 and expands the concept to multiple qubits and entanglement in Section 2.2. Section 2.3 gives an overview of quantum computation, including quantum gates. Section 2.4 describes the algorithmic family of variational hybrid quantum-classical algorithms that is used in this work. Finally Section 2.5 introduces the concepts of noise modelling, which are used to mimic the limitations of near-term quantum systems.

2.1. Qubits and Superpositions

The fundamental unit of information in quantum computing is the quantum bit, also known as the *qubit*. While a classical bit can only exist in either state 0 or 1, a qubit can can exist in a superposition of $|0\rangle$ and $|1\rangle$ [26]. This superposition is a linear combination of the computational basis states $|0\rangle$ and $|1\rangle$, which can be expressed as:

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

where $\alpha, \beta \in \mathbb{C}$ are referred to as *amplitudes*.

Dirac-Notation The notation for quantum states above is referred to as *Dirac-notation* [27]. Here, a so called *ket* $|\psi\rangle$, which is labeled by ψ, refers to a vector representing the state of a quantum system [28]. The ket is a linear combination of vectors such that

$$|\psi\rangle = a_1\,|s_1\rangle + a_2\,|s_2\rangle + ... + a_n\,|s_n\rangle\,, \tag{2.2}$$

where $a_i \in \mathbb{C}$ is the amplitude for vectors $|s_i\rangle$ [28]. A quantum system can be expressed as a linear combination of vectors, which form a basis $\mathbb{B}$ for the system and allow each state to be uniquely represented. Additionally, a basis in quantum computing typically is required to be *orthonormal* [28], which is explained in the following.

The conjugate transpose of a ket $|\psi\rangle$ is called *bra* $\langle\psi|$, with $|\psi\rangle = \begin{pmatrix} a_1 \\ a_2 \\ ... \\ a_n \end{pmatrix}$ and $\langle\psi| = \begin{pmatrix} \overline{a_1} & \overline{a_2} & ... & \overline{a_n} \end{pmatrix}$. Two vectors, $|s_1\rangle$ and $|s_2\rangle$, are said to be *orthogonal* if the inner product $\langle s_1|s_2\rangle$ is zero. A set of vectors $\mathbb{B}$ is called *orthonormal* if each $|\psi\rangle \in \mathbb{B}$ is of unit length, i.e. $\langle\psi|\psi\rangle = 1$, and and all elements of $\mathbb{B}$ are mutually orthogonal [28]. One possible set of state vectors to express all single-qubit systems is given by the computational basis

$$\mathbb{B} = \{|0\rangle\,, |1\rangle\} = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}, \tag{2.3}$$

which fulfills the orthonormality condition [28].

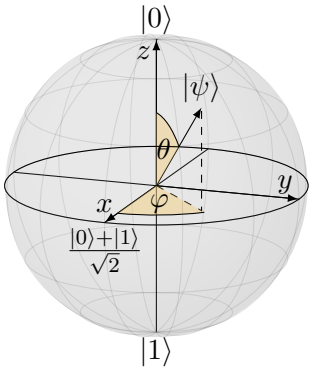


Figure 2.1.: Bloch sphere representation of a single qubit.

According to the principles of quantum mechanics, when the state of $|\psi\rangle$ is measured, a superposition collapses into one of the computational basis states. The probability of the collapse is determined by the square of its corresponding amplitude, which is $|\alpha|^2$ for $|0\rangle$ or $|\beta|^2$ for $|1\rangle$ for the quantum state described in Equation 2.1. As the sum of probabilities for the superposition collapse must be one, the normalisation condition $|\alpha|^2 + |\beta|^2 = 1$ must be fulfilled [26].

As described in Ref. [26] Equation 2.1 can also be expressed as:

$$|\psi\rangle = e^{i\gamma} \left(\cos\frac{\theta}{2} |0\rangle + e^{i\varphi} \sin\frac{\theta}{2} |1\rangle \right), \tag{2.4}$$

with $\gamma, \theta, \varphi \in \mathbb{R}$. The factor $e^{i\gamma}$ represents a global phase, which can be disregarded in the following as it does not have an observable impact. Every single-qubit quantum state, characterised by parameters θ and φ can be visualised on the surface of the unit three-dimensional sphere, often referred to as *Bloch sphere*, which is depicted in Figure 2.1. While the visualisation with the Bloch sphere can be useful for the representation of the state of a single qubit, there is no simple generalisation for multi-qubit systems [26]. The upcoming section describes the characteristics of multi-qubit systems, which offer crucial characteristics for quantum computation.

2.2. Multiple Qubits and Entanglement

In the case of two classical bits, they can be in four possible states, namely 00, 01, 10, and 11. Similarly, for two qubits, there are four computational basis states denoted by $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. However, unlike classical bits, a pair of qubits can also exist in superpositions of these states. Thus, as described in Ref. [26] the quantum state of a two qubits system can be expressed by

$$|\psi\rangle = \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle, \tag{2.5}$$

where $\alpha_{ij} \in \mathbb{C}$ with $i, j \in \{0, 1\}$ are the respective probability amplitudes for the computational basis states. As for single-qubit systems, the normalisation condition requires the squared amplitudes to sum up to one. In general, a state of a quantum system with n qubits can be described by 2^n amplitudes. Consequently, the information that can be stored in such a quantum state is far greater than that in a classical state. Thus, quantum computation aims to exploit the substantial information capacity of quantum systems [26].

As described in Ref [26], quantum computing can exploit a certain phenomenon that is exclusive to quantum mechanics, which is *entanglement*. For instance, the Bell state or EPR pair [29], [30], represented by

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}, \tag{2.6}$$

is a so-called *entangled* two-qubit state, which has the peculiar property that upon measuring the first qubit, one obtains two possible results. A measurement of the second qubit always gives the same result as the measurement of the first qubit, resulting in correlated measurement outcomes. In contrast to a classical state space, a quantum system's state cannot always be characterised by considering the states of its separate components [28]. The phenomenon of entanglement serves as the basis for numerous quantum computing approaches [26].

Analogous, to the vector notation for single-qubit systems, a vector space of 2^n dimensions can be used to describe quantum states of multi-qubit systems with n qubits. Generally, as described in detail in Ref. [26], vector spaces V and W can be combined by the so-called *tensor product*. A basis of the combined vector space is given by $|i\rangle \otimes |j\rangle$, where $|i\rangle$ and $|j\rangle$ are the orthonormal bases of W and V respectively. The states of the combined vector space can be expressed as linear combinations of $|i\rangle \otimes |j\rangle$. An abbreviation for a tensor product of multiple states such as $|v_1\rangle \otimes |v_2\rangle \otimes ... \otimes |v_n\rangle$ is commonly written as $|v_1 v_2 ... v_n\rangle$. The Kronecker product can be used to represent this operation for matrices and vectors, as described in Ref. [26]. For instance, when using the computational basis state vector for $|0\rangle$ for two operands, the operation can be represented as follows:

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \cdot 1 \\ 1 \cdot 0 \\ 0 \cdot 1 \\ 0 \cdot 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{2.7}$$

which is one of the computational basis states for two-qubit systems.

2.3. Quantum Computation

In this section, we provide an overview of quantum computation. Specifically, the process of transforming these quantum states into other quantum states using so-called *quantum gates* is described.

Quantum Gates A crucial aspect of quantum computing is manipulating the state of a quantum system. This can be achieved by rotating the quantum state vector in the complex vector space using a transformation, such as applying a matrix. To perform valid state transformations according to the dynamics of quantum mechanics the corresponding matrix U is required to fulfill the condition

$$U^\dagger U = \mathbb{1}, \tag{2.8}$$

where $U^\dagger$ is the conjugate transpose of U and $\mathbb{1}$ is the identity matrix, i.e. U must be *unitary* [26]. This ensures reversibility and also guarantees that all state vectors are kept normalized in every transformation step. Practically, quantum computers perform quantum state transformations using gates that are connected by wires and form circuits, with gates representing unitary matrices [26].

Two sample quantum circuits, which are also used in this work, are shown in Chapter 5 in Figure 5.1. The following two subsections provide an overview of important single-qubit and multi-qubit gates.

2.3.1. Single-Qubit Gates

In classical computing, there is only one non-trivial member in the class of single bit gates, the *NOT* gate, which flips a bit, i.e., transforms 0 to 1 and vice versa. In contrast, quantum computing employs multiple single-qubit gates. The quantum equivalent to the classical *NOT* gate is the X gate [26], which is represented by

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2.9}$$

Applying X on an arbitrary single-qubit state $\alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$ using the computational basis produces

$$X(\alpha|0\rangle + \beta|1\rangle) = X\left(\alpha\begin{pmatrix}1\\0\end{pmatrix} + \beta\begin{pmatrix}0\\1\end{pmatrix}\right) = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{pmatrix}\alpha\\\beta\end{pmatrix} = \begin{pmatrix}\beta\\\alpha\end{pmatrix} = \beta|0\rangle + \alpha|1\rangle, \tag{2.10}$$

and thus performs a negation. In general, the first column of a single-qubit matrix demonstrates the transformation of the computational basis state $|0\rangle$, while the second column represents the transformation of $|1\rangle$.

The X gate is one of the *Pauli* gates, which are among the most commonly used single-qubit transformations. The three Pauli matrices [28] are given by:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.11}$$

According to Ref. [28], the Pauli matrices possess a unique characteristic of being not just unitary, but also Hermitian matrices. A matrix A is considered to be Hermitian when it satisfies the following condition:

$$A = A^\dagger. \tag{2.12}$$

As described in Ref. [28], a special property of Hermitian matrices is that their eigenvalues are real. When Hermitian operators are used for measurements, they are often referred to as *observables*. The possible post-measurement states are represented by the eigenvectors of the operator A. For instance, as the Pauli-Z operator has the eigenvectors $|0\rangle$ and $|1\rangle$ with the eigenvalues 1 and -1, respectively, Z is commonly used for measurements in the computational basis. Hermitian operators and especially the Pauli-Z gate are essential in the context of hybrid quantum-classical algorithms, including QRL approaches.

The Pauli matrices, when exponentiated, generate three important classes of unitary matrices known as *rotation operators* around the x, y, and z axes in the Bloch sphere representation [26]. As outlined in Ref. [26], these rotation gates are defined by:

$$
\begin{aligned}
R_x(\theta) = e^{-i\theta X/2} &= \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \\
R_y(\theta) = e^{-i\theta Y/2} &= \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \\
R_z(\theta) = e^{-i\theta Z/2} &= \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}.
\end{aligned} \tag{2.13}
$$

As a rotation gate can be parameterised by a value θ, it is often used as one of the building blocks for variational quantum circuits (VQCs), which is described further in Section 2.4.

2.3.2. Multi-Qubit Gates

To utilise the phenomenon of entanglement in quantum computation, operations or gates can be performed on multiple qubits to create a correlation between them. The prototypical gate to introduce entanglement between two qubits is the so-called controlled-NOT or $CNOT$ gate, which operates on two qubits, with the first referred to as the *control qubit* and the second as the *target qubit*. The state of the target qubit is determined by the control qubit. If the control qubit is measured to be in state $|1\rangle$, the target qubit will be flipped; otherwise, the state of the target qubit remains unchanged [26]. To summarise, the $CNOT$ gate performes the following operations on the computational basis states:

$$
|00\rangle \to |00\rangle \qquad |01\rangle \to |01\rangle \qquad |10\rangle \to |11\rangle \qquad |11\rangle \to |10\rangle \tag{2.14}
$$

The $CNOT$ gate is defined by the following unitary matrix:

$$
CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.15}
$$

In general, any single-qubit unitary operation U can be used in a two-qubit *controlled*-U operation, employing a control and a target qubit. If the control qubit is measured to be in state $|1\rangle$, then U will be applied on the target qubit, otherwise the state of the target qubit stays the same [26]. The circuit symbols for the $CNOT$ and contolled-Pauli-Z (CZ) gate, which are used in the QRL algorithms later are depicted in Figure 2.2.

(a) CNOT **(b)** CZ

Figure 2.2.: Circuit symbols for the controlled-NOT ($CNOT$) and the controlled-Pauli-Z (CZ) gate.

This section covered the basic single-qubit and multi-qubit gates, including Pauli-rotation and entangling controlled gates. These gates serve as building blocks for variational quantum algorithms

(VQAs), which are discussed in the next section.

2.4. Variational Quantum Algorithms

This section introduces the family of variational hybrid quantum-classical algorithms. Concrete variational algorithms for QRL are described in the next chapter in Section 3.4.

According to Ref. [31], variational quantum algorithms (VQAs) aim to approximate a specific target function by minimising a corresponding cost function C that depends on parameters $\vec{\theta}$. such that

$$\vec{\theta}^* = \arg\min_{\vec{\theta}} C(\vec{\theta}), \tag{2.16}$$

where $\vec{\theta}^*$ represent the optimal parameters. The VQA employs a quantum circuit containing gates with tunable parameters, e.g. Pauli-rotation gates. Similar to classical machine learning approaches, these parameters are adjusted in an optimisation or training process. After feeding an input state through such a variational quantum circuit (VQC), the output is measured and the parameters are optimised using classical optimisation techniques, such as gradient descent [32]. This process is repeated until the desired target function is approximated with sufficient quality.

As depicted in Figure 2.3 a VQC consists of three parts. In the first part, a quantum state is prepared to represent the classical input data $\vec{x}$. A unitary operator $U_e(\vec{x})$ is applied on the initial quantum state, which by convention, is assumed to be $|0\rangle$ for each qubit [26]. In the second part, the quantum state is then transformed by applying a parameterised unitary $U(\vec{\theta})$, which is also referred to as a variational layer. Finally, classical information $\langle\hat{O}\rangle$ is obtained from the quantum circuit by measuring the state. The notation $\langle\hat{O}\rangle$ refers to the *expectation value* of an observable $\hat{O}$ and is discussed in Section 2.4.2. The classical algorithm interacts with the VQC by feeding input data $\vec{x}$ to the encoding component and by passing parameters $\vec{\theta}$ to the variational layer [33]. The upcoming three subsections provide a more detailed description of the main concepts of VQCs: classical data encoding, quantum data extraction and the calculation of gradients, which are often required by classical optimisers.

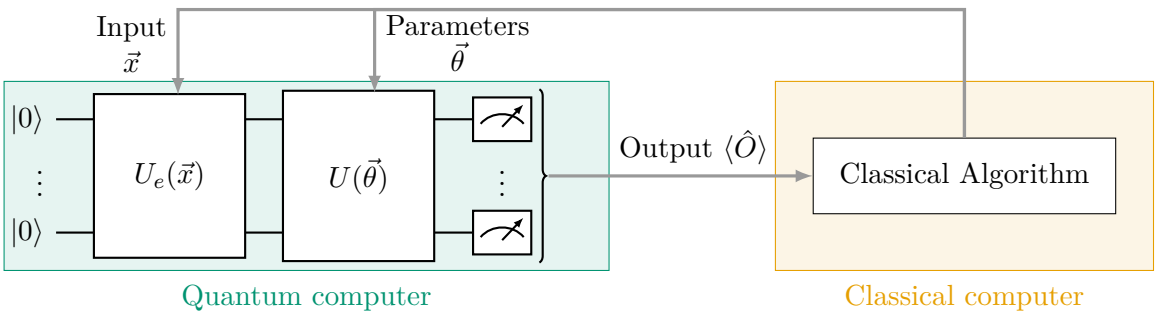


Figure 2.3.: The structure of a variational quantum-classical algorithm using a variational quantum circuit (VQC).

2.4.1. Classical Data Encoding

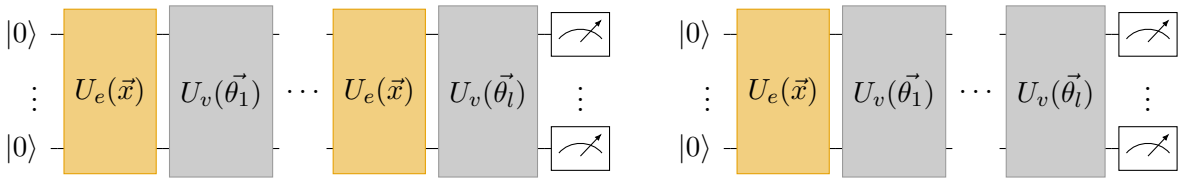
As stated prior, classical data $\vec{x}$ is prepared as a quantum state using a unitary $U_e(\vec{x})$. The structure and gates that build this unitary depend on the encoding strategy that is applied. Three of the most common strategies to encode the classical data, proposed by Weigold et al. [34], [35], are listed in the following:

- **Bitwise encoding** maps one classical bit to one qubit. The encoding unitary $U_{e_b}(\vec{x})$ comprises Pauli-X gates, which get applied to a qubit if the corresponding classical bit is set to one. However, since data is typically represented by multiple bits, such as 32 bits for a single-precision floating-point number, encoding even a small amount of data can quickly exceed the capacity of currently available gate-based quantum systems. For instance, the maximum number of qubits offered by the gate-based quantum computers from IBM as of 2023 is 127 [36].

- **Amplitude encoding** is a method that allows denser encoding by mapping the real values, such as integers or floating point numbers, to the amplitudes of a quantum state. This technique can encode 2^n values using n qubits with the limitation that the amplitudes must sum up to one, according to the normalisation condition. Although amplitude encoding provides the densest encoding, the state preparation using a corresponding unitary U_{e_a} requires an exponential number of operations to encode 2^n data values in the general case [34], [37]. We will therefore not consider amplitude encoding in this work.

- **Angle encoding** uses a Pauli-rotation gate to encode one real value into one qubit. The corresponding unitary $U_{e_\angle}(\vec{x})$ can comprise one [25], [34] or multiple [23], [24] parameterised Pauli rotation gates per qubit. There are various methods proposed in the literature for encoding input data $\vec{x}$ as angles for rotation gates, given that these gates are periodic and the input must be scaled to an interval smaller than 4π. For instance, Lockwood and Si [24] propose two encoding schemes: Scaled (S) encoding, which determines a rotation angle by scaling finite-domain input values to $[0, 2\pi]$, and Directional (D) encoding, which encodes infinite-domain inputs by rotating the qubit by π if the input is greater than 0. Skolik et al. [25] additionally present Continuous (C) encoding, which computes rotation angles as the arctan of the respective input value [6]. A comparative analysis of the underlined encoding schemes in a QRL algorithm can be found in Chapter 5. As angle encoding only requires one qubit for each input value, it will be used in this work.

Up until this point, the standard encoding method follows a traditional neural network setup, where the encoding unitary, i.e. the input component, generally preceeds the variational layers, as shown in Figure 2.4b. However, gate-based VQCs do not have a theoretical limitation on the maximum number of repetitions of input features that can be introduced into the circuit. Therefore, the encoding unitary can be reintroduced at multiple instances in the VQC, as shown in Figure 2.4a. This approach, known as *data re-uploading* [38], is suggested to increase the expressivity of a VQC [6], i.e. the capability to express information. Reintroducing the encoding circuit increases the expressivity of the model [39]. It was shown by Schuld et al. [39] that the functions represented by VQCs are Fourier sums. In this functions, the variational layers determine the amplitudes of the Fourier sum and the encoding layer fixes the frequency spectrum. Therefore, introducing more encoding layers via data re-uploading, increases the frequency spectrum represented by the VQC and thus the expressivity of the represented function class. In Chapter 5, the standard encoding and the encoding method with data re-uploading



(a) VQC architecture with data re-uploading. **(b)** Standard VQC architechture.

Figure 2.4.: VQC architectures with and without data re-uploading for an encoding unitary $U_e(\vec{x})$ representing the classical data $\vec{x}$ and a variational unitary $U_v(\vec{\theta}_i)$, parameterised by $\vec{\theta}_i$ with $i \in [1, l]$ and l variational layers.

are compared through experiments.

2.4.2. Quantum Data Extraction

To extract classical information from a quantum state $|\psi\rangle$, the *expectation value* $\langle\hat{O}\rangle$ [40] of an observable $\hat{O}$ needs to be used, which can be computed by

$$\langle\hat{O}\rangle = \frac{\langle\psi|\,\hat{O}\,|\psi\rangle}{\langle\psi|\psi\rangle}. \tag{2.17}$$

As only normalised quantum states are considered in this work, the denominator containing the inner product $\langle\psi|\psi\rangle$ can be disregarded as it always equals one. Since measurements are probabilistic and the probability of obtaining a particular result depends on the amplitude of the associated quantum state, expectation values can be estimated by averaging over multiple runs and measurements of the circuit. As described above in Subsection 2.3.1, measurements using Pauli-observables are often used in VQAs [40].

2.4.3. Gradient Calculation

To solve the optimisation problem from Equation 2.16 the parameters $\vec{\theta}$ can be trained using gradient-base approaches. In this section a VQC is considered, where the parameterised unitary $U(\vec{\theta})$ that depends on m real gate parameters $\vec{\theta}$ transforms the initial state $|\psi\rangle$, followed by a measurement of the observable $\hat{O}$. According to Ref. [41], in a hybrid VQA the calculation of the VQC can then be expressed as a function $f : \mathbb{R}^m \to \mathbb{R}$, mapping the gate parameters to an expectation:

$$f(\vec{\theta}) := \langle\hat{O}\rangle = \langle\psi|\,U^\dagger(\vec{\theta})\hat{O}U(\vec{\theta})\,|\psi\rangle\,. \tag{2.18}$$

For a parameterised quantum gate that is generated by a Hermitian operator with two unique eigenvalues, the partial derivative of $f(\vec{\theta})$ with respect to $\mu \in \vec{\theta}$ can be evaluated by the *parameter-shift rule* [5], [41]. Considering that the parameterised gates in unitary $U(\vec{\theta})$ are Pauli rotation gates, the following equation [31] holds:

$$\begin{aligned}\frac{\partial f(\vec{\theta})}{\partial\mu} &= \frac{1}{2}\left(\langle\psi|\,U^\dagger(\vec{\theta}_{\mu,+\frac{\pi}{2}})\hat{O}U(\vec{\theta}_{\mu,+\frac{\pi}{2}})\,|\psi\rangle - \langle\psi|\,U^\dagger(\vec{\theta}_{\mu,+\frac{\pi}{2}})\hat{O}U(\vec{\theta}_{\mu,-\frac{\pi}{2}})\,|\psi\rangle\right)\\ &= \frac{1}{2}\left(f(\vec{\theta}_{\mu,+\frac{\pi}{2}}) - f(\vec{\theta}_{\mu,-\frac{\pi}{2}})\right)\end{aligned} \tag{2.19}$$

The notation $\vec{\theta}_{\mu,s}$ denotes a parameter shift of s applied to the parameter μ, meaning that the shifted parameter $(\mu + s)$ belongs to the set of parameters $\vec{\theta}_{\mu,s}$. Unlike numerical differentiation techniques, such as using *finite differences* [42], the parameter-shift rule allows for analatic evaluation of gradients, as shown in Ref. [41]. For a general cost function $C(\vec{\theta})$, which further processes the expectation of the VQC, the gradient can be obtained by using the chain rule [31].

Similar to a classical neural network [43], a VQC is also proven to be a universal approximator [38], i.e. it is capable of reproducing any continous function, given a sufficient number of parameters. This characteristic makes the VQC a promising choice for a variety of optimisation and machine learning algorithms, which rely on a function approximator as a fundamental concept. Additionally, as the number of qubits and the circuit depth is typically controllable, the VQC approach is highly versatile and suitable for NISQ devices [4].

This section introduced VQAs that can use gradient-based optimisation techniques on ideal quantum devices or quantum simulators. In the following section we discuss how noise models can be used to simulate noisy quantum circuits.

2.5. Modelling Noise

This section presents a theoretical overview of how noise affects quantum calculations, including VQAs. The description is presented in a manner that is precise enough to draw dependable conclusions for quantum software and algorithms, without requiring an in-depth understanding of quantum physics. The content of this section is based on the description of quantum noise in Ref. [44].

2.5.1. Mixed Quantum States

In earlier sections, we represented a quantum state $|\psi\rangle \in \mathbb{C}^{2^n}$ as a unit vector in a 2^n dimensional complex vector space. This notation can only express states that are a linear combination of basis states, which are associated with an amplitude. However, such states, also known as *pure states*, cannot fully capture the state of a system when external influences such as noise are present. For instance, if a qubit ψ_i is flipped with probability p_i, the system could transform into the state $|\neg\psi_i\rangle$ with probability p_i or remain in state $|\neg\psi_i\rangle$ with probability $1 - p_i$. In this case, the system state can be described using a probability distribution of possible states, which is referred to as a *mixed state*. Mixed states entail two sources of probabilistic behavior. First, the outcomes of measurements are stochastic due to the fundamental properties of quantum theory. Second, the external influence of noise introduces a level of uncertainty due to a lack of knowledge on the systems behaviour.

When noise manipulates a quantum state, classical uncertainty arises that can be expressed using classical probabilities p_i. With the *density matrix* formalism [26], a mixed state can be described as multiple pure quantum states $|\psi_i\rangle$ mixed with classical probabilities p_i by

$$\rho = \sum_i p_i \left|\psi_i\right\rangle \left\langle\psi_i\right| . \tag{2.20}$$

The upcoming subsection utilises transformations of the density matrix to illustrate the evolution of quantum states under the influence of noise.

2.5.2. Operations in Noisy Quantum Systems

As discussed in previous sections, the evolution of a quantum state can be described by a unitary U in a closed quantum system, which does not suffer from any unwanted interactions with the environment [26]. If the system's initial state is $|\psi_i\rangle$ with probability p_i, after applying U the system will be in the state $U|\psi_i\rangle$ with the same probability p_i. This leads to the following evolution equation [26] of the density matrix:

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| \xrightarrow{U} \sum_i p_i U |\psi_i\rangle \langle\psi_i| U^\dagger = U\rho U^\dagger. \tag{2.21}$$

In the case of a noisy system, the quantum system is open to an external environment. To model such noisy open systems, the environment is included, resulting in a larger but closed quantum system. Let ρ denote the state of the open quantum system of interest, called the *principal system* and ρ_{env} the state of the *environment*, which together form a closed quantum system with input state $\rho \otimes \rho_{env}$ [26]. The evolution when applying unitary U can be described by $U(\rho \otimes \rho_{env})U^\dagger$. The final state of the principal system can then be expressed by the *quantum operator* $\mathcal{E}(\rho)$ that may not be related by a unitary transformation to the initial state ρ. Tracing out the environment reveals the evolution of ρ under noise:

$$\mathcal{E}(\rho) = tr_{env}\left[U(\rho \otimes \rho_{env})U^\dagger\right]. \tag{2.22}$$

The partial trace tr_a that is described in detail in Ref. [26] is a tool in the density matrix formalism to discard certain parts of a quantum mechanical systems.

Suppose that $\{|e_k\rangle\}_k$ is an orthonormal basis for the environment and without loss of generality that $\rho_{env} = |e_0\rangle \langle e_0|$ is the initial state of the environment. As described in Ref. [26], Equation 2.22 can then be rewritten as the *operator-sum representation* of $\mathcal{E}$, which is expressed as

$$\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger, \tag{2.23}$$

with E_k being the operator on the state space of the principal system, known as the *operation elements* for the operation $\mathcal{E}$ [26]. In the following subsection, we describe some instances of this general noise modelling principle.

2.5.3. Noise Models

In a quantum system, noise can arise from different sources such as decoherence, where quantum information is lost over time, or from imperfect calculations and measurements. This work focuses on gate errors, which are errors that occur when performing calculations. In the following, we describe two kinds of gate errors, which are evaluated experimentally for QRL in Section 5.

- **Bit/Phase Flip:** A probabilistic quantum bit flip [26] can be defined by

$$\mathcal{E}(\rho) = pX\rho X^\dagger + (1-p)\mathbb{1}\rho\mathbb{1}^\dagger. \tag{2.24}$$

  With this operator, a Pauli-X gate is applied to the single-qubit system with state ρ with a probability of p, and otherwise, the state remains unchanged. The equation above, is one way to describe the operator elements E_k from Equation 2.23. Similary, the bit-phase flip error or the phase flip error can be constructed, by replacing the Pauli-X with Pauli-Y or Pauli-Z gates,

respectively [26].

- **Depolarisation:** An error commonly used in simulations is the completely depolarising operator [26] applied to a single qubit, which randomly applies the Pauli-X, Y, or Z operators with probability p, and leaves the qubit unchanged otherwise:

$$\mathcal{E}(\rho) = \frac{p}{4}(\mathbb{1}\rho\mathbb{1}^\dagger + X\rho X^\dagger + Y\rho Y^\dagger + Z\rho Z^\dagger) + (1-p)\mathbb{1}\rho\mathbb{1}^\dagger. \tag{2.25}$$

  This Equation can be simplified to

$$\mathcal{E}(\rho) = \frac{p}{2}\mathbb{1} + (1-p)\mathbb{1}\rho\mathbb{1}^\dagger, \tag{2.26}$$

  where $\frac{p}{2}\mathbb{1}$ is the density representing the state of a system being in every basis state with equal probability. As a result, the system will either remain unchanged or have all information destroyed with a probability of p. For a system consisting of n qubits, the depolarising operator can be expressed as [26]

$$\mathcal{E}(\rho) = \frac{p}{2^n}\mathbb{1} + (1-p)\mathbb{1}\rho\mathbb{1}^\dagger. \tag{2.27}$$

To sum up, this chapter has presented the fundamental principles of quantum computing that are relevant to this thesis. These concepts are be used to develop and test quantum reinforcement learning methods in simulations, considering both ideal and noisy quantum models. The upcoming chapter introduces the concepts of reinforcement learning and explore how they can be adapted for quantum-based approaches.

3. Background on Quantum Reinforcement Learning

In this chapter, the main concepts of classical reinforcement learning (RL) and quantum reinforcement learning (QRL) are introduced. The formulation of RL environments using a Markov decision process (MDP) is described in the following subsection. Subsection 3.2 outlines a specific class of RL approaches known as *Q-learning*, which is further specified in Subsection 3.3 for *deep Q-learning* (DQL). Finally, an overview of a quantum version of DQL is provided in Subsection 3.4.

3.1. Markov Decision Process

Most formulations of RL center around the notion of a *Markov decision process* (MDP) [8], where an *agent* interacts with an *environment*[1] at discrete time steps t. In each time step, the current configuration of the agent in the environment is summarised by the *state* $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. Based on this information, the agent selects an *action* A_t from a set of possible actions $\mathcal{A}$ according to a *policy* $\pi(s, a) = \mathbb{P}[A_t = a|S_t = s]$, which gives the probability $\mathbb{P}$ of taking action a in state s. Executing the selected action causes a *transition* of the environment to a next state $S_{t+1} \in \mathcal{S}$; simultaneously, the agent receives a scalar *reward* $R_{t+1} \in \mathcal{R}$ that quantifies the contribution of the selected action towards solving the task, with $\mathcal{R} \subset \mathbb{R}$ being the set of all rewards. Such a transition can be defined as a tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ and is graphically depicted in Figure 3.1.

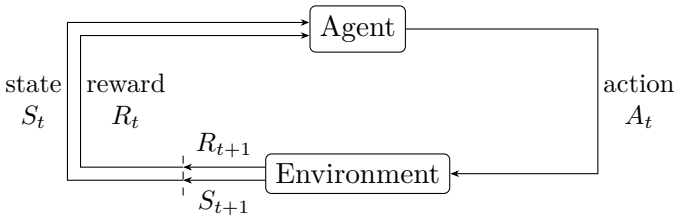


Figure 3.1.: Transition $(S_t, A_t, R_{t+1}, S_{t+1})$ in a MDP [8].

3.1.1. Rewards

The agent's goal is to maximise the return [8]

$$G_t = \sum_{t'=t}^{T} \gamma^{t'} R_{t'}, \tag{3.1}$$

[1] The term "environment" used in the context of RL should not be confused with the "environment" defined earlier in the text for noise modelling.

i.e. the discounted sum of rewards, until a terminal state S_T is reached. In that, the discount factor γ controls how much the agent favors immediate over future rewards. The period between the initial time step and S_T is often referred to as an *episode*. Both S_{t+1} and R_{t+1} are assumed to obey the Markov property [45], i.e., conditional independence of previous states and actions given S_t, A_t. However, the MDP's *dynamics*, $\mathbb{P}[S_{t+1}, R_{t+1}|S_t, A_t]$, are typically unknown to the agent, which necessitates learning a policy by trial-and-error.

3.1.2. Policies

A policy π can be evaluated with the *action-value* function

$$q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a], \tag{3.2}$$

which calculates the expected return, starting from state $s \in \mathcal{S}$ when following policy π under the constraint that action $a \in \mathcal{A}$ was executed. The results of the action-value function are also referred to as *Q-values* [8].

To solve a RL task, a policy must be found that achieves the highest possible reward over time. Using the function from Equation 3.2 an order of possible strategies can be established. A policy π is called better than a policy π' if $q_\pi(s, a) > q_{\pi'}(s, a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$. The optimal action-value function q_*

$$q_*(s, a) = \max_\pi q_\pi(s, a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{3.3}$$

is maximised for the optimal policy π_*, If q_* is known, then π_* can be derived by [25]

$$\pi_*(a, s) = \arg\max_\pi q_\pi(s, a). \tag{3.4}$$

The goal of RL approaches is to determine or approximate the optimal policy with sufficient quality. This thesis focuses on the *Q-Learning* approach, which is described further in the upcoming sections.

3.2. Q-Learning

Q-Learning [46] is a technique used to approximate the optimal action-value function defined in Equation 3.3. This algorithm combines ideas from *dynamic programming* and *temporal-difference* learning, both of which is discussed in this section.

3.2.1. Dynamic Programming

In dynamic programming, value functions are commonly used to facilitate the search for optimal policies [8]. To this end, *Bellman optimality equations* [8] can be converted into update rules for improving approximations of the desired value functions. The action-value function in Equation 3.3

can be expressed as a Bellman equation

$$\begin{aligned} q_*(s,a) &= \max_\pi q_\pi(s,a) \\ &= \max_\pi \mathbb{E}[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a\right], \end{aligned} \tag{3.5}$$

which relates the current state to its subsequent state. In a sequence of Q-values $(q_0, q_1, \dots)$, for which q_0 is randomly initialised, the subsequent Q-value q_{k+1} to q_k with $k \in \mathbb{N}$ can be calculated by

$$q_{k+1}(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') | S_t = s, A_t = a\right]. \tag{3.6}$$

The policy π_k, which can be derived from q_k, converges against the optimal policy π_* for $k \to \infty$ [47].

However, to calculate the expectation value in Equation 3.6, a complete and accurate model of the environment, i.e. a model of its reward and next-state probability distribution, is required, which is not practical for most environments [8]. Therefore, Q-learning also considers experience, i.e. sequences of states, actions, and rewards from actual or simulated interaction with the environment [8]. The following subsection outlines, how experience can be utilised in so-called temporal-difference learning.

3.2.2. Temporal-Difference Learning

Temporal-difference learning allows for building new approximations of the action-value function based on previous approximations. Q-learning is a specific approach to temporal-difference learning that was originally proposed as a tabular learning algorithm [46]. This means that the Q-values for each action-value pair encountered during learning are stored in a table. The algorithm starts by arbitrarily initialising this table. It then collects experience by exploring the environment and updates each Q-value for an explored action-value pair using the following update rule [8]:

$$q(S_t, A_t) \leftarrow (1-\alpha) \underbrace{q(S_t, A_t)}_{\text{old}} + \alpha \left(\underbrace{R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a')}_{\text{new}} \right), \tag{3.7}$$

where $\alpha \in [0, 1]$ is a constant step-size. As proven in Ref. [48], in the tabular Q-learning approach, $q(s,a)$ converges to the optimal Q-values $q_*(s,a)$ when all state-action pairs are visited infinitely often. However, in applications with many possible transitions, storing the Q-value for all states and actions is inefficient in terms of memory usage and difficult to implement. Therefore, in so-called (classical) *deep Q-learning* [47], a neural network is used to approximate the action-value function, which is outlined in the following section.

3.3. Deep Q-Learning

The fundamental idea of deep Q-learning (DQL) [47] is to learn the optimal Q-values from Equation 3.3 using a multi-layered neural network that for any given state s outputs a vector of Q-values $q(s, a; \vec{\theta})$,

where $\vec{\theta}$ are the parameters of the neural network. If the state space has n dimensions and the action space has m actions, then the neural network is a function from R^n to R^m [19].

The neural network is then trained to minimise the temporal difference error, i.e. the difference between the "old" and "new" value from Equation 3.7 under some loss function [47]. One possible loss function is e.g. the *mean squared loss function* [49]

$$\mathbb{L}_i(\theta_i) = \mathbb{E}_{(S_t, A_t, R_{t+1}, S_{t+1})}\left[\left(Y_i - q(S_t, A_t; \vec{\theta}_i)\right)^2\right], \tag{3.8}$$

which is evaluated in each training step $i \in \mathbb{N}$ on samples of transitions $(S_t, A_t, R_{t+1}, S_{t+1})$. The target Y_i [19] is defined as

$$Y_i \equiv R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a'; \vec{\theta}_i). \tag{3.9}$$

The remainder of this section describes the update of the parameters based on the loss, as well as common techniques to address known problems in training convergence, namely *experience replay* and a *target network*.

3.3.1. Updating Weights with Gradient Descent

The parameters, also called *weights*, of the deep neural network can be updated using gradient-based optimisation, such as stochastic gradient descent [32],

$$\theta_{i+1}^k = \theta_i^k - \alpha \frac{\partial}{\partial \theta_i^k} \mathbb{L}_i(\vec{\theta}_i), \tag{3.10}$$

where $\alpha \in [0, 1]$ is a step size, $\theta_i^k \in \vec{\theta}_i$ is the parameter at index $k \in \mathbb{N}$ in parameter set $\vec{\theta}_i$ at training step $i \in \mathbb{N}$ and $\frac{\partial}{\partial \theta_i^k} \mathbb{L}_i(\vec{\theta}_i)$ the corresponding gradient. The gradients can be obtained via *backpropagation* [50], i.e. applying the chain rule for calculating derivatives with respect to the loss function $\mathbb{L}_i$.

As Mnih et al. [51] point out, learning q_* with a high-capacity function approximator leads to convergence problems. To this end, DQL makes use of a target network and experience replay, which is described in the following two subsections.

3.3.2. Experience Replay

Experience replay, introduced in Ref. [52], involves storing the agent's experience, which consists of transitions denoted by $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$, at a time step t in a so-called *replay buffer* $D_t = \{e_1, \ldots, e_t\}$ across multiple episodes. During training, the parameters are updated based on the loss function from Equation 3.8 by uniformly sampling mini-batches of experience $e \sim U(D)$ that are randomly drawn from the stored samples [47]. Experience replay is used to achieve data efficiency and to reduce correlation between samples. For a detailed discussion on the advantages of using experience replay the reader is referred to Ref. [47].

In DQL, the transitions are sampled using an off-policy approach, i.e. instead of applying the current greedy policy, an ε-greedy behavior policy that selects a random action with probability ε is chosen.

Decaying ε over the course of training allows the agent to explore the environment, while guaranteeing that the behavior policy and target policy converge eventually.

3.3.3. Target Network

To further enhance the stability of DQL, a second modification involves using a separate network to generate the targets Y_i [19]. Specifically the standard neural network, in this context called *online network*, is cloned every $c \in \mathbb{N}$ time steps to obtain a *target network*, which is parameterised by $\vec{\theta}^-$. Therefore, instead of calculating the target values with the online network as defined in Equation 3.9, the target values in training step $i \in \mathbb{N}$ are calculated using the target network:

$$Y_i \equiv R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a'; \vec{\theta}_i^-). \tag{3.11}$$

As discussed in Ref. [47], this modification makes the algorithm more stable compared to standard DQL, where an update that increases $q(S_t, A_t; \vec{\theta})$ often leads to an increase in succeeding Q-value approximations and thus also increases the target Y_i, which can cause policy oscillations or divergence. By generating the targets using an older set of parameters, a delay is introduced between the time an update to q is made and the time it affects the targets Y_i, thereby reducing the likelihood of divergence or oscillations.

This section provided an overview of the main concepts of classical DQL. These concepts serve as the foundation for a quantum version of DQL, with some modifications specific to the quantum scenario. The upcoming section delves into these quantum specific changes.

3.4. Variational Quantum Deep Q-Learning

A specific class of recent hybrid quantum-classical RL methods is that of variational quantum deep Q-learning (VQ-DQL) [6], [23]–[25]. In this approach, the traditional deep neural network in DQL is replaced by a variational quantum circuit (VQC). As outlined in Section 2.4, a VQC has the capability to serve as a universial function approximator, making it a suitable machine learning model. The parameters of the VQC can be optimised using gradient descent and the parameter-shift rule, which is described in Section 2.4.3. The remainder of this section gives an overview of how the input encoding and the output extraction in the VQC differ compared to the classical approach.

3.4.1. Input Encoding

To input a classical MDP state $s \in \mathcal{S}$ to the VQC, the state must be represented as a quantum state $|\psi(s)\rangle$ using the available qubits. Various encoding strategies discussed in Section 2.4.1 can be employed for this purpose. The choice of a suitable encoding strategy depends on the structure of the classical input components. Therefore, Section 5.1 describes the input components of the MDP state in the `CartPole` benchmark environment [53] and Section 5.4.1 explore the encoding methods, which can be applied to it.

3.4.2. Q-Value Extraction

For a given input MDP state, the VQC predicts Q-values for all $|\mathcal{A}|$ actions simultaneously by taking the expectation value of a measurement, using Pauli-Z observables of a corresponding number of output qubits. The resulting expectation values lie within $[1; 1]$; thus obtaining valid action values can require additional processing, for instance by scaling the measured results by a learned multiplicative factor, which is further described for the particular MDP environment `CartPole` in Section 5.4.1.

By replacing the classical neural network with a VQC, there is potential to incorporate the power of quantum computation into the RL domain. While there is no proof that VQ-DQL models are superior to classical DQL models, several experiments [6], [23]–[25] have been conducted that suggest advantages in terms of sampling or parameter complexity. In Chapter 5, we describe some of these experiments.

4. Related Work

This chapter gives an overview of the existing research and literature, which serves as a foundation for this work.

4.1. Deep Q-Learning and its instabilities

Deep Q-Learning (DQL) dates back to Watkin's Q-Learning [46] and has seen a lot of interest over the years due to its immense potential in learning capabilities. DQL is itself an active field of research because of its versatility in applications. Nevertheless, as versatile as the applications are, the algorithm possesses space for improvements in its stability and speed of convergence to a solution [17], [19], [54]–[59]. In particular, the Q-learning approaches, i.e., off-policy learning with function approximation and bootstrapping, are known to diverge in certain scenarios. This divergence occurs more often when the Q-value is approximated using a non-linear function approximator such as a deep neural network. However, the root causes are still unknown [19], [60]–[62].

4.2. Quantum Reinforcement Learning

Over the past few years, there have been several attempts to improve the performance of reinforcement learning algorithms via possible "quantum advantage" using quantum computing. Like in the classical realm, no method has emerged as the superior approach in performance or generality. The first known quantum reinforcement learning (QRL) algorithm has been proposed by Dong et al. [63], which uses a modified version of Grover's algorithm [1] to learn a state-value function. As in the classical reinforcement learning family, whose members vary in algorithm and methodology, various algorithms for QRL have been studied [64]–[68].

The VQ-DQL approach was originally proposed by Chen et al. [23] where the authors have used VQCs to solve two different discrete environments, namely, "cognitive radio" [69] and "frozen lake" [53]. Both these environments are discrete environments where the state space is finite. Lockwood and Si [24], further investigated the VQ-DQL algorithm by utilizing a VQC to tackle environments with both continuous and discrete state spaces. Another study that analyses the learning performance and behavior of VQ-DQL was conducted by Skolik et al. [25]. Here the authors explore the effects of having a VQC as a Q-value approximator along with techniques like data re-uploading and a hybrid machine learning model, which combines components of a classical neural network with a VQC. The performance of these variational approaches [24], [25] is discussed in detail in our reproduction study in Section 5.2.

While there have been claims in the literature about the potential advantages of VQ-DQL in terms of sampling and parameter complexity [6], [23]–[25], the practicality of implementing these approaches

on current NISQ devices is limited due to noise susceptibility. However, a study conducted by Skolik et al. [70] suggests that VQ-DQL methods exhibit a certain level of robustness against (simulated) noise. Additionally, insights from other quantum machine learning domains [71] support the notion that incorporating noise during the training process can achieve comparable performance to the ideal case. Based on these discoveries, in our study, we further explore the performance of VQ-DQL under specific noise models in Section 5.6. The goal is to assess the resilience of VQ-DQL in the presence of noise. By examining the effects of noise on VQ-DQL, we can gain valuable insights into the practicality and viability of these approaches in real-world quantum computing scenarios.

5. Experimental Analysis: Variational Quantum Deep Q-Learning

This chapter presents our experimental analysis of Variational Quantum Deep Q-Learning (VQ-DQL). The chapter is organised as follows: We begin by introducing the `CartPole` environment, which serves as the benchmark for our experiments in the upcoming section. We then discuss the reproduction study conducted to validate previous research on VQ-DQL in Section 5.2. In Section 5.3, we describe the methodology employed in our experimental analysis, which we use for our experiments with ideal quantum simulators in Section 5.4. Next, in Section 5.5, we compare the performance of VQ-DQL with its classical counterpart. Finally, in Section 5.6, we investigate the impact of noise on the performance of VQ-DQL.

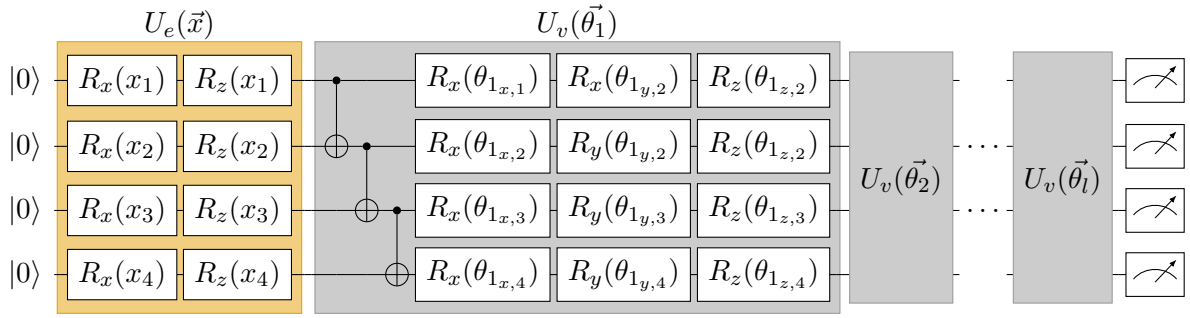
5.1. The `CartPole` Environment

The `CartPole` environment [53] has emerged as a widely recognised benchmark in the field of RL. It poses a non-trivial task due to its continuous state-space. In the environment, the goal is for an agent to learn how to balance a pole upright on a cart that moves along a frictionless track. The action-space for the agent is discrete and includes two actions: moving the cart to the left or to the right. The continuous state-space comprises four observations, which are listed in Table 5.1.

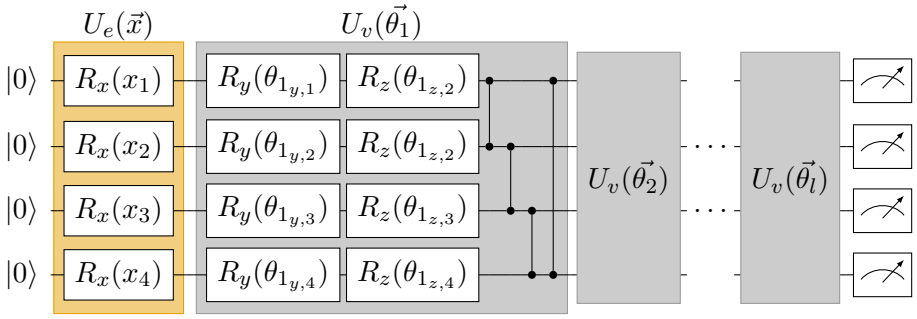
Table 5.1.: State observations in `CartPole`

Observation	Minimum	Maximum
Cart Position	-4.8	4.8
Cart Velocity	$-\infty$	∞
Pole Angle	$-24°$	$24°$
Pole Angular Velocity	$-\infty$	∞

An episode terminates when the pole angle exceeds $\pm 12°$, the cart position surpasses ± 2.4, or the agent reaches a predefined maximum number of steps per episode. OpenAI [53] provides two variants of `CartPole`: `CartPole-v0`, where the maximum number of steps in an episode is 200 and `CartPole-v1`, where it is 500. At the start of each episode, the four variables of the environment state are randomly initialised within a stable range of $[-0.05, 0.05]$. Throughout the episode, the agent receives a reward of one at each step. The episode score is computed as the cumulative sum of rewards obtained during the episode. Consequently, the episode score is directly tied to the length of the episode, with a maximum achievable score of 200 (`CartPole-v0`) or 500 (`CartPole-v1`).



(a) VQC architecture by Lockwood and Si [24]



(b) VQC architecture by Skolik et al. [25]

Figure 5.1.: VQC architectures from Refs. [24] and [25].

5.2. Reproduction study

To gauge the learning capability of VQ-DQL, we first reproduce the results published by Lockwood and Si [24] on the `CartPole-v1` task. We train five VQ-DQL agents and evaluate their performance during training using the source code[1] published by the authors, in which the VQC architecture depicted in Figure 5.1a is used. The results are visualised in Figure 5.2a. The blue line indicates episode returns. The red line represents a moving average of the (up to) 20 previous returns[2]. While our measurements reproduce the computational outcome of the published results, we identify two notable methodological aspects that require careful consideration and interpretation:

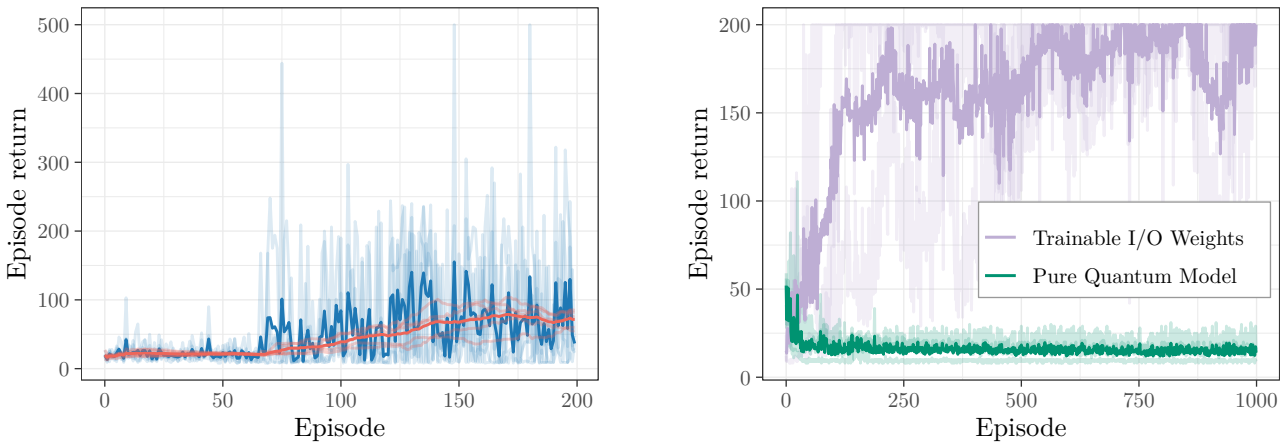
- **Training frequency**—A step of mini-batch gradient descent is carried out only once per episode (namely, after its termination). This differs substantially not only from the original DQL algorithm, but also from the pseudo-code provided by Lockwood and Si [24], where training is executed in regular intervals after a set number of trajectories has been sampled by the agent. We are not aware of other approaches in the literature that pursue or analyse this approach, and conjecture that it might have a detrimental effect on learning, since the distribution of transitions in the replay buffer grows faster than the amount of data that the agent perceives. The adaptation also complicates the comparison between independent runs of the algorithm, depending on the length of the experienced episodes.
- **Performance evaluation**—Measuring agent performance in terms of a moving average over previous runs is not a good indicator for learning success: Averaged returns have been generated

[1] Available on GitHub (link in PDF).
[2] Note that these statistics have been measured with the original source code, without modification. Superficial differences in visual appearance are caused by the plot aesthetic settings.

by different policies, that is, trained on increasing numbers of transitions at different stages of ε-decay. Further, the averaging approach shadows any underlying instabilities as indicated by the raw episode returns: In all five runs, the blue line oscillates strongly between low and high return values, indicating that the underlying policy network/circuit fails to converge towards an optimal policy. Note that in complex environments, DQL convergence *can* be non-monotonic in terms of measured returns (see, e.g., Ref. [72]). Observing oscillations of this magnitude on `CartPole` (which can be learnt in an approximately monotonic fashion by a simple neural network with DQL, refer to Section 5.5) does not give a promising outlook on capability of VQ-DQL to generalise to more challenging tasks.

Besides, we would like to explicitly point out that the experiment is based on `CartPole-v1` where return values of up to 500 can be achieved. In contrast, returns in `CartPole-v0` cannot exceed 200, which is important to take into account when judging closeness to optimality of particular approaches, especially when the visual display of episode return time series uses clipped axes.

One other study which overcame these instabilities using a VQ-DQL algorithm to solve the `CartPole` environment is conducted by Skolik et al. [25]. Here the authors have used slightly different gate connectivity in their VQC compared to Lockwood and Si [24]. Apart from the change in VQC architecture, which is depicted in Figure 5.1b, the authors also perform a gradient descent optimization step after a fixed number of sampling steps. They also present their total reward attained in each episode averaged over ten different agents rather than presenting a moving average. Skolik et al. [25] have studied and tested various combinations of pure and quantum-classical hybrid VQC architectures in their work. However, the pure VQC model did exhibit the same instabilities exhibited by Lockwood



(a) Results from Ref. [24], reproduced using the published source code. The light blue lines indicate the total reward collected in an episode, using the greedy policy for each agent. The light red lines represent a moving average of the (up to) 20 previous episode returns. Results are averaged over five experiments, which is represented by the strong red and strong blue lines. The experiments are based on the `CartPole-v1` environment, where the maximum achievable return value is 500.

(b) Replication of the results from Ref. [25]. Here we reproduced the quantum-classical model with classical trainable weights applied to the input and output and the pure quantum model, both with data re-uploading. The experiments are based on the `CartPole-v0` environment, where the maximum achievable return value is 200.

Figure 5.2.: Reproduced results from Refs. [24] and [25]

and Si's model. Skolik et al. [25] used a VQC model where the inputs to and outputs from the VQC were multiplied with classical weights along with the data re-uploading strategy [38] (cf. Section 2.4.1) to overcome these instabilities.

Figure 5.2b presents the outcomes of our replication experiment, which aims to reproduce the findings of Skolik et al. [25][3]. The experimental setup followed the parameters specified in the Appendix section of Ref. [25]. The measurement results shown in Figure 5.2b confirm the published results that the quantum-classical model exhibited a stable learning behavior, while the pure quantum model, which had no classical weights on the input and output parameters of the VQC, failed to learn the given task.

In order to gain deeper insights into the effects of different influencing factors on training performance, such as classical training weights and encoding schemes, the subsequent sections provide a comprehensive description of our experimental setup and methodology.

5.3. Methodology

Previous implementations of the VQ-DQL approach show various methodological issues [24] that we have discussed in the previous section. For having a stable and uniform VQ-DQL framework that coincides with the classical RL practices and to provide a replication of existing results on top of mere reproduction, we re-implement the original deep Q-learning algorithm as described in [47], [72] in Tensorflow-Quantum [73](TFQ). In contrast to the previous implementations, which use TFQ too, our re-implementation allows to conventiently integrate extensions and has a higher degree of configurability of hyperparameters. Furthermore we included a flexible validation mechanism, which is used to evaluate the performance of a current policy. Since in previous implementations a fair comparison between different studies was difficult due to several meanings of return values (e.g. averaging over past episodes as in Ref. [24] vs. taking a single episodes return value as in Ref. [25]), we designed our validation mechanism to allow a uniform comparison of different classical and quantum RL approaches. (Section 5.3 discusses implementation details). This section covers experiments, which were conducted using the quantum simulators of the TFQ framework. Apart from our implementation in TFQ, we have also adapted the code to the Qiskit framework [74] to conduct experiments on IBM Quantum devices and utilise IBM quantum noise models [36] (cf. Section 5.6.1). The process of porting between frameworks, as we elaborate on in in Ref. [75], is relatively straightforward and enables us to leverage the unique features of each framework.

To describe our methodology, let us first set the employed conventions: By *sampling steps*, we refer to the transitions sampled from the ε-greedy behavior policy. By *training step*, we understand one iteration of gradient descent. Words in `monospaced` font indicate configurable parameters of the algorithms.

To ensure comparability between our different experimental setups, and especially between previous research and our dedicated experiments, we choose sampling steps as fundamental unit of training time. Each experiment is run for 50 000 sampling steps. We deliberately use a long time horizon to capture

[3] The associated source code published by the authors of Ref. [25] is available on GitHub (link in PDF). Since this implementation was not published during the reproduction process in the paper [6] associated with this thesis, we independently replicated the results reported by Skolik et al. [25] using our own custom implementation.

any phenomena that may materialise late in the learning process caused by slow convergence, but retain the possibility to terminate successful runs prematurely, as described in detail below. Initially, the replay memory is pre-filled with `train_after=1000` sampling steps, corresponding to at least five full episodes, using a uniform random policy with $\varepsilon = 1$.

A sampling step does not necessarily entail a training step; instead, a training step is carried out every `train_every` sampling steps. As backpropagation on quantum devices is computationally intensive due to gradients being estimated via the parameter-shift rule [5], [41], we introduced this parameter as a means to keep the number of training steps per episode feasible. We note, however that in this thesis and in the accompanying paper [6], we only report validation results on quantum hardware, while the agent has been trained in simulation. Similarly, we update the target network parameters to match the policy network parameters every `update_every` sampling steps. After the initial warm-up phase, we decay ε linearly over `epsilon_duration` sampling steps in total, starting at a value of `epsilon_start=1`, and ending at a value of `epsilon_end=0.01`. Keeping $\varepsilon > 0$ ensures continued exploration with a near-greedy policy.

Since performance on the ε-greedy policy is not indicative of learned performance when ε is large [76], we estimate the expected return achieved by the current greedy policy in regular intervals. Specifically, we measure return over a single episode on a copy of the training environment every `validate_every=100` sampling steps (note that the parameter does not influence the actual training process, and is just used for performance monitoring). If the average validation return over the past consecutive 25 validation steps reaches 196 (recall that the maximum return is 200, and that we need to allow for some jitter), we regard the task as solved and terminate training early. While this differs from the official `CartPole-v0` benchmark [53] that necessitates an average score of at least 195 over 100 episodes, we find that training is very unlikely to diverge past this point, given that ε has decayed sufficiently. This early stopping criterion is particularly beneficial for experiments conducted on the experimental IBM quantum device, where a reduced number of episodes is crucial for practical feasibility. To showcase the effectiveness of the early stopping criterion, we present a collection of results for VQ-DQL agents trained using this criterion and validated against the solving criterion of the official benchmark [53]. The results, which can be found in Appendix A, demonstrate that the majority of trained agents successfully solved the `CartPole-v0` environment according to the official benchmark.[4]

5.4. Experiments with Ideal Quantum Simulators

Using our TFQ-implementation, we run a set of experiments to systematically evaluate the observed instabilities. Throughout all our experiments, we used the `CartPole-v0` environment to ensure comparability with [25] and [24], and also to keep computational cost at bay. Section 5.4.1 investigates the effects of the chosen input encoding and Q-value extraction method on performance and stability. Using these insights, we run an extensive cross-validation study described in Section 5.4.2.

[4] PLease note that, for the purpose of data visualization during averaging, any remaining steps in the training process will be substituted with the optimal return value of 200 if the training was halted prematurely due to the early stopping criterion.

5.4.1. Encoding and Extraction Methods

In this subsection, we delve into the various approaches for mapping input parameters onto quantum states, as previously outlined in Section 2.4.1; we consider the following approaches, which are special cases of angle enciding:

- **Continuous (C)**: continuous encoding applied to all input components.
- **Scaled & Continuous (SC)**: scaled encoding applied to finite-domain input components (i.e. cart position, pole angle), continuous encoding otherwise (i.e. cart velocity, pole angular velocity; cf. Table 5.1).
- **Scaled & Directional (SD)**: scaled encoding applied to finite-domain input components, directional encoding otherwise.

Along with the encoding strategies, we also investigate the impact of different Q-value extraction methods on agent performance. This is necessary due to the mismatch between VQC outputs and Q-values. In particular, we distinguish between:

- **Local Scaling**: each output is scaled by a dedicated trainable weight as described in Ref. [25].
- **Global Scaling (GS)**: all outputs are scaled by a single trainable weight.
- **Global Scaling with Quantum Pooling (GSP)**: *quantum pooling* as described in Ref. [24], followed by global scaling. Quantum pooling involves incorporating additional rotation and entangling gates into the circuit to compress the observation space to match the action space. However, the overall advantages of utilizing this pooling operation are not explicitly discussed in Ref. [24]. Therefore, we empirically evaluate its effectiveness in our experiments.

Initial Experiment

We conducted experiments for each combination of input encoding, Q-value extraction method and circuit architecture, totalling in 18 runs. To this end, we adapted hyperparameters from Ref. [25] to our slightly modified algorithm described in Section 5.3 (without data re-uploading). VQC weights are initialised to zero to avoid *barren plateaus* [77], i.e. the vanishing gradient problem as suggested in Ref. [78] and classical weights are initialised to one. Throughout all our experiments, we employed five variational layers based on the VQC architectures illustrated in Figure5.1.

Results are shown in Figure 5.3. As is apparent, instabilities occur in every run and are not tied to a specific encoding-/extraction setting. Nevertheless, some models only achieve comparatively low returns on average: In particular, runs involving directional encoding tend to perform sub-par, which we attribute to the high information-loss incurred by the encoding scheme. Directional encoding is therefore not considered in further experiments.

To minimise the number of classical parameters, we focus on global scaling (with and without pooling) in further experiments. While local scaling has not performed worse or less stable, the additional classical parameters increase model capacity, and might therefore shadow deficiencies on the quantum parts.

As described by Mnih et al. [51], Q-Learning is known to be instable, when a nonlinear function

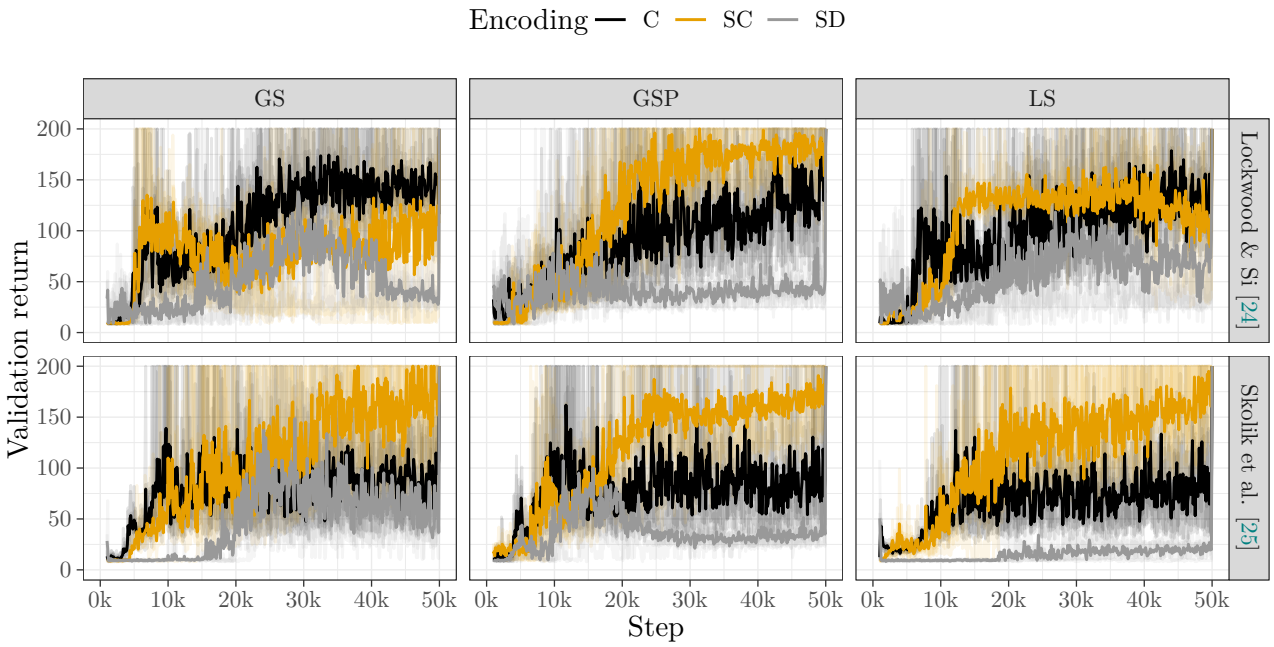


Figure 5.3.: Validation returns for using the VQC-layer structure as in Ref. [24] (top) and Ref. [25] (bottom) with different input encoding strategies. The columns correspond to the extraction strategy (ltr. Global Scaling (GS), Global Scaling with Quantum Pooling (GSP), Local Scaling (LS), cf. section 5.4.1). Results are averaged over five experiments each. These experiments are based on the `CartPole-v0` environment, where the maximum achievable return value is 200.

approximator, such as a classical neural network or a VQC, is used to represent the state-action value function. Our implementation already incorporates mechanisms suggested by Mnih et al. to support convergence in Q-Learning. However, these mechanisms do not guarantee a stable behaviour and we can not rule out that the instabilities in VQ-DQL are caused by classical algorithmic constraints. In section 5.5 we compare the VQC to a classical neural network using the same algorithm. Our results support the hypothesis that the reason for instabilities could be classical. Therefore, in the next subsection, we study the effect of classical hyperparameters on the training process of VQ-DQL.

5.4.2. Cross-Validation

As instabilities persist throughout our experiments, we turn to hyperparameters as a source of instabilities. To this end, we re-utilise the previous setting (C, SC/GS, GSP) with hyperparameters from Ref. [25] as a starting point. Following recommendations [23], [79]–[81] from classical supervised learning, we add a linear decay to the learning rate η. In particular, we decrease η over a period of `eta_duration` training steps from `eta_start` towards a target value of `eta_end=0.01*eta_start`. Additionally, we progressively increase the `update_every` parameter as learning progresses. This choice is motivated by the observation that the delta between target and policy network decreases as the agent becomes more proficient on the task. Finally, to optimise resource utilization and minimise training time, we increase the batch size from 16 to 32, since this does not have a major impact on the agent's performance [82].

Table 5.2.: Hyperparameter settings for cross-validation.

Hyperparameter	Description	Default value
	Fixed parameters throughout cross validation runs	
`num_steps`	#sampling steps	50 000
`train_after`	#sampling steps before first training step	1 000
`train_every`	#sampling steps between training steps	10
`update_every_start`	initial #sampling steps between target network updates	30
`update_every_end`	final #sampling steps between target network updates	500
`update_every_duration`	#sampling steps for `update_every` increase	35 000
`replay_capacity`	max. #transitions in replay buffer	50 000
`optimiser`	Loss-function optimiser	Adam [83]
`batch_size`	batch size for gradient descent	32
`loss`	TD error loss function	L_2
`epsilon_start`	initial value for ε decay	1.0
`epsilon_end`	final value for ε decay	0.01
`validate_every`	#sampling steps between validation runs	100
`eta_end`	final value for learning rate η	$0.01 * $ `eta_start`
	Hyperparameters subject to cross validation	
`eta_start` (η_s)	initial value for learning rate η	{0.001, 0.01, 0.1}
`eta_duration` (η_d)	#training steps for learning rate decay	{2 000, 4 000}
`epsilon_duration` (ε_d)	#sampling steps for ε decay	{10 000, 20 000, 30 000}
`gamma` (γ)	discount factor γ	{0.99, 0.999}

We cross-validate over the following hyper-parameter choices: `eta_start` (i.e., the initial learning rate) $\in \{10^{-3}, 10^{-2}, 10^{-1}\}$, `eta_duration` (learning rate decay duration) $\in \{2000, 4000\}$, `epsilon_duration` $\in \{10000, 20000, 30000\}$, `gamma` $\in \{0.99, 0.999\}$. The remaining parameters have been kept fixed over all experiments and are listed in Table 5.2. The following subsections describe our results obtained on the baseline setting (without data re-uploading), and a modified variant with data re-uploading.

Baseline

Results for the baseline case are depicted in Figure 5.4 (top two rows) and Table 5.3. The results in Figure 5.4 show the runs with the best-performing hyperparameter configuration for each combination of encoding/extraction method and VQC architecture. The full set of results is illustrated in Appendix B. As evident from the figure, almost every model was able to achieve stable optimal performance (according to our early-stopping criterion). Generally, the SC encoding tends to converge faster as compared to models with continuous encoding; in the best case (Skolik et al./SC/GS), optimal performance is reached after a mere 97 validation steps. This shows that VQ-DQL is in fact capable of learning a stable optimal policy, albeit hyperparameter tuning is a sensitive influence factor.

Baseline with data re-uploading

From Figure 5.4, it is evident that the performance of the VQ-DQL algorithm also suffers due to the choice of encoding strategy used along with the bad choice of hyperparameters. For example, the

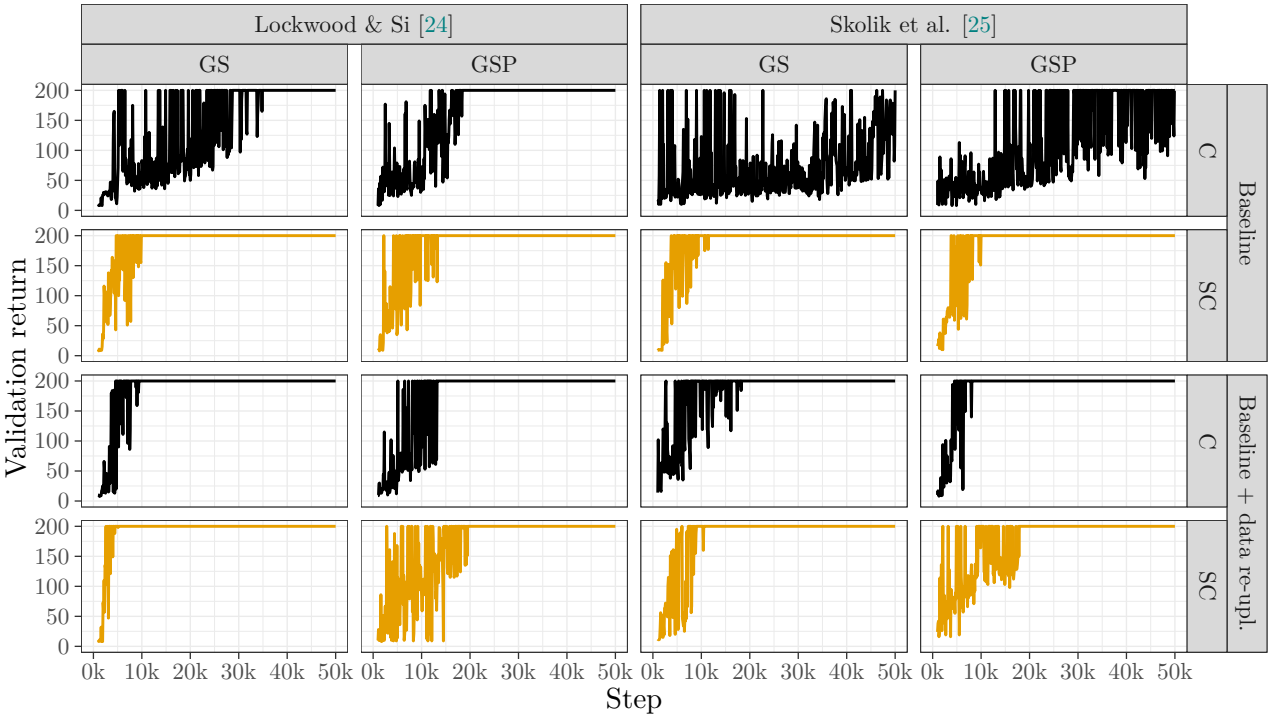


Figure 5.4.: Validation returns for our best-performing hyperparameter constellations in the baseline configurations. The figure considers the results for the VQC architecture described in Ref. [24] (left) and Ref. [25] (right). Columns correspond to the extraction method (ltr. Global Scaling (GS), Global Scaling with Quantum Pooling (GSP)), rows correspond to the input encoding strategy (Continuous (C), Scaled & Continuous (SC), cf. Section 5.4.1) and whether data re-uploading was applied. The experiments are based on the `CartPole-v0` environment, where the maximum achievable return value is 200.

Table 5.3.: Best-performing hyperparameters identified in cross-validation. The table provides values for `eta_start` (η_s), `eta_duration` (η_d), `epsilon_duration` (ε_s), and `gamma` (γ). Encodings C, SC, GS, and GSP as defined in Section 5.4.1.

Architecture	Baseline				Baseline + data re-uploading			
	η_s	η_d	ε_d	γ	η_s	η_d	ε_d	γ
[24]/C/GSP	0.01	2 000	20 000	0.99	0.01	2 000	30 000	0.99
[24]/C/GS	0.001	4 000	20 000	0.99	0.01	2 000	30 000	0.999
[24]/SC/GSP	0.01	2 000	20 000	0.99	0.1	2 000	20 000	0.999
[24]/SC/GS	0.01	4 000	30 000	0.99	0.01	2 000	30 000	0.99
[25]/C/GSP	-	-	-	-	0.01	2 000	30 000	0.999
[25]/C/GS	-	-	-	-	0.01	2 000	10 000	0.99
[25]/SC/GSP	0.01	2 000	10 000	0.999	0.01	2 000	10 000	0.99
[25]/SC/GS	0.01	4 000	30 000	0.99	0.01	2 000	10 000	0.99

agent with the continuous encoding format does not learn an optimal policy in many cases. Here to increase the expressivity of the model, we can use techniques such as data re-uploading [25], [38]. The results for the baseline case with data re-uploading are also depicted in Figure 5.4 (bottom two rows) and Table 5.3. As in Section 5.4.2, we only present a selection of the best-performing hyperparameter constellations (cf. Appendix B for the full set of results). From the results shown in Figure 5.4, we can conclude that the data re-uploading strategy does not significantly increase the VQ-DQL algorithm's performance. Though it increases the expressive power of the model, which in turn allows the agent to learn optimal behavior in some cases (for example, agent with Continuous (C) encoding), the performance change is negligible or even negative in some cases. Moreover, the data re-uploading strategy increases the gate count in the VQC architecture, which is not ideal for NISQ devices due to noise.

5.5. Comparison to a Classical Neural Network

A popular "quantum advantage" claimed by a good fraction of the literature in QRL is that the VQC has better state-action pair representation, samples efficiently, and learns an optimal policy faster than the classical neural network [23]–[25]. Hence to compare the sample efficiency of a VQ-DQL-agent trained on an ideal simulator against a classical neural network, we trained a simple fully-connected network with one hidden layer to solve the `CartPole-v0` environment. To ensure a fair comparison, we restricted the total number of parameters of the network to 58, which is approximately in the same order of magnitude as the number of parameter in the VQC. For reference, the comparative VQ-DQL approach, which is based on the "Skolik et al./SC/GS" architechture/encoding/extraction configuration without data re-uploading (cf. Section 5.4.2), employs 41 parameters in total (40 "quantum" parameters for the VQC and one classical trainable parameter for global scaling). Furthermore, we conducted cross-validation on the same set of hyperparameters as explained in Sec. 5.4.2.

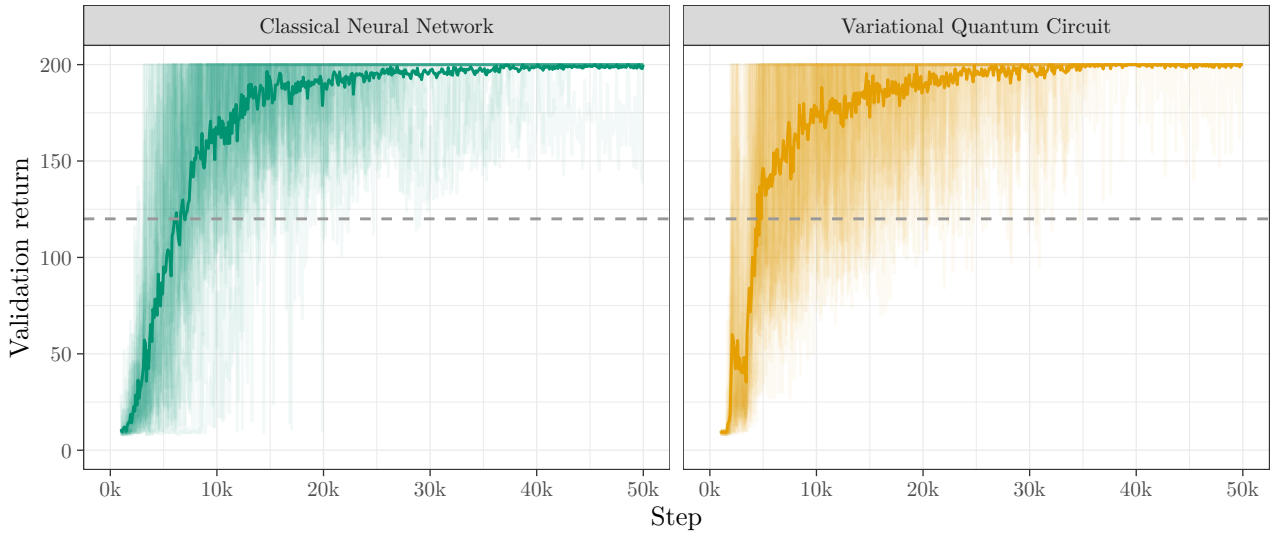


Figure 5.5.: Comparison between a VQC and classical neural network averaged over 30 different agents. The dashed line represents the threshold value V_{thresh} used in the significance test. The experiments are based on the `CartPole-v0` environment, where the maximum achievable return value is 200.

The results shown in Fig. 5.5 indicate that initially, the VQC seems to learn faster than the neural network. For a more rigorous discussion we resort to Ref. [84], where sample efficiency of an algorithm is defined for an online learning setting as the number of time steps from which on an agent trained by the algorithm perceives an average reward exceeding a certain threshold V_{thresh} with high probability.

For a weaker statement adapted to a numerical treatment, we propose to use significance testing under the null hypothesis of mean reward being smaller than V_{thresh}. Thus, we define sample efficiency as the number of time steps from which on the null hypothesis is rejected with respect to the given threshold. As statistical test we propose to use a one-sample t-test [85], [86], in particular its one-sided version as we compare the performance of a particular algorithm against a given threshold. Thus, we perform sufficiently many independent runs of each algorithm and fix the significance level at $\alpha = 0.05$.

With respect to this metric the variational quantum circuit indeed crosses $V_{\text{thresh}} = 120$ faster than the classical network. This indicates a potential improvement in terms of sampling complexity and also parameter complexity compared to the classical neural network, considering that we utilised slightly fewer parameters for the VQC. However, for larger threshold values, no definitive conclusion can be drawn. Furthermore, these experiments were performed using an ideal quantum simulation for the VQ-DQL algorithm. The impact of noise on the VQ-DQL agents performance is examined in the next section.

5.6. Experiments with Noisy Quantum Systems

In this section, we explore the influence of noise on the performance of VQ-DQL. We conduct experiments using both real hardware, specifically the IBM quantum device `ibmq_ehningen` [36], and noisy quantum simulators. In the following subsection, we present validation results on the IBM quantum device for an agent trained using an ideal simulator. These results provide valuable insights into the performance of VQ-DQL under realistic noise conditions. Subsequently, we introduce specific noise into the training process to evaluate the robustness of VQ-DQL when noise is present during both training and validation stages.

5.6.1. Validation on IBM Quantum Device

Results from Sec. 5.4.2 and Sec. 5.4.2 illustrate that a VQC can learn a stable policy to solve the `CartPole-v0` environment using the VQ-DQL algorithm if the right set of hyperparameters is used. In order to gauge the detrimental influence of device noise on an agent trained using an ideal simulator in solving the environment, we tested the trained model in an actual IBM quantum device [36]. As a first step, we had to port the VQ-DQL algorithm from the TFQ API [73] to the Qiskit API [74] as the IBM quantum devices [36] use the Qiskit API as their primary programming library. There is one significant difference between the Qiskit API and the TFQ API to be noted here. The TFQ API calculates the expectation value analytically, whereas the Qiskit API estimates the expectation value by simulating the ideal quantum device and measuring its outcomes. Likewise, the expectation values are estimated in the IBM quantum device [36] by measuring the outcome multiple times. Further, we trained the best-performing model without data re-uploading from Sec. 5.4.2 (specifically the "Skolik et al./SC/GS" configuration) using Qiskit `qasm_simulator` [74] and verified the correctness of our implementation in comparison to the results from Sec. 5.4.2. We chose a model without

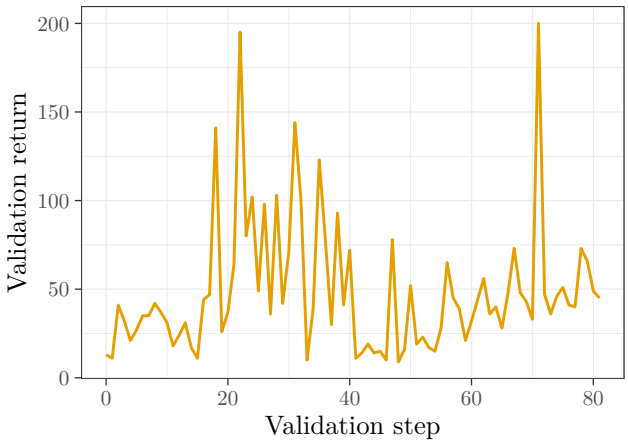


Figure 5.6.: Results of our validation run on `ibmq_ehningen` [36]. The experiments are based on the `CartPole-v0` environment, where the maximum achievable return value is 200.

data re-uploading due to the fact that the quantum devices available right now are prone to noise. Hence adding more gates via data re-uploading in NISQ devices seems counter-productive. Once the correctness was verified, we uploaded the weights trained using the `qasm_simulator` to the IBM quantum (`ibmq_ehningen`) device and validated the learned policy. The results of these validation runs are shown in Fig. 5.6.

Though the agents trained in the ideal simulator learned an optimal policy to solve the `CartPole-v0` environment, testing the trained agent in the `ibmq_ehningen` device did not reproduce the optimal behavior. This degradation in behavior is due to the noise present in the IBM quantum device. An agent trained in the IBM quantum device from scratch might reduce the effect of noise and learn a policy close to the optimal policy. Additionally, different types of error mitigation techniques can be employed to reduce the effects of noise at the cost of additional overhead. However, when we attempted to train the agent from scratch on the IBM quantum device, the training turned out to be infeasible due to the following practical issues:

1. We observed waiting times in the queue to start a job execution (referred to as fair-share queue for jobs in IBM quantum systems) in the cloud-based IBM quantum device that were typically two orders of magnitude (or more) larger than the actual job execution time. As (roughly speaking) a single action selection corresponds to a single job in the fair share queue, even completion of a single episode takes a substantial amount of time.

2. The overall time it takes to achieve low-variance estimators of expectation values can become quite large due to the large number of shots (i.e., measurement samples) taken for a single circuit instance.

Here, the first hindrance can be overcome in time as the availability of quantum devices and resources is expected to increase in the near future. As improvements in hardware and orchestration of quantum and classical computational resources progress, we might also witness an increased number of circuit layer operations per second (CLOPS) [87]. When we started the training process in the `ibmq_ehningen` device, the job execution time for each action selection took between 15 to 30 seconds, and each training step took around 3 minutes (as the training step performs gradient decent via parameter-shift rule). These long execution and waiting times make the training process in real quantum devices

impractical for training algorithms like VQ-DQL, where the agent has to interact with the environment sequentially.

Due to the limitations discussed above, conducting a complete training of a VQ-DQL agent on an IBM quantum device is beyond the scope of this thesis. However, IBM provides a set of noise models in their Qiskit API [74] to simulate the noise present in their quantum devices. In the following subsection, we delve into the experiments conducted using different simulated noise models.

5.6.2. Training with Noisy Quantum Simulators

In order to further assess the impact of noise on the performance of the VQ-DQL algorithm, we incorporate noise into the training process using specific noise models. In this subsection, we initially examine the separate effects of individual elementary noise types, which can offer insights for the design of future quantum systems. Subsequently, we explore the influence of a noise model provided by the Qiskit API [74], which characterises current real hardware.

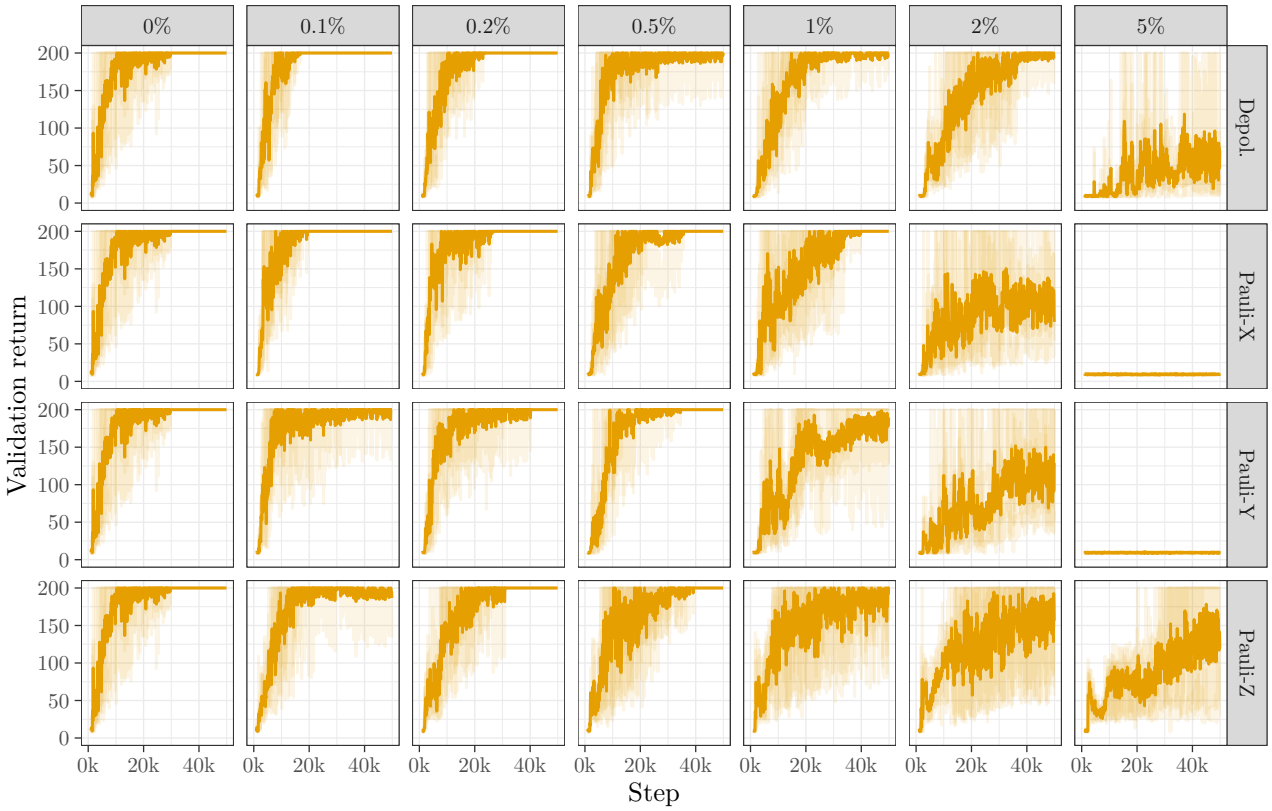


Figure 5.7.: Validation returns of VQ-DQL during the training process under different noise variants and gate error rates. Noise was introduced during both training and validation stages. Results are averaged over five experiments each. The experiments are based on the `CartPole-v0` environment, where the maximum achievable return value is 200.

Pauli/Depolarising Noise Models

First, we incorporate the gate errors specified in Section 2.5.3 in the training of the VQ-DQL agent. As for the validation on the IBM quantum device, we chose a configuration without data re-uploading. This choice was made to keep the circuit shallow and the gate count minimal, thereby reducing potential sources of errors. We selected the architecture/encoding/extraction model configuration with its corresponding hyperparameters that performed best in the ideal case, which was also utilised in previous sections (i.e. "Skolik et al./SC/GS"). Figure 5.7 shows the validation returns during the training process using the Pauli-X, -Y, -Z, and depolarizing channels with varying gate error rate. The figure illustrates that as the error rate increases, the VQ-DQL agent requires more steps to converge to the optimal performance. Nevertheless, for low error rates up to 0.5%, the agent achieves comparable results to the ideal case. This finding is particularly encouraging for near-term quantum devices, which typically exhibit error rates ranging from ca. 0.01% to 3% [44]. Moreover, certain platforms demonstrate particular resilience against specific types of noise. For instance, trapped ion systems have shown resilience against bit flip (Pauli-X) errors [88], [89], which, along with Pauli-Y errors, can be particularly troublesome at higher error rates. This resilience can be a relevant criterion when selecting a quantum platform for a software system [44]. However, it is essential to note that current NISQ devices are susceptible to various other types errors, such as coherence errors. These additional forms of noise can also be simulated and modeled using more complex noise models. For a detailed description, we refer to Ref. [26].

Real Hardware Noise Model

To evaluate the impact of additional types of noise, we employ a comprehensive noise model provided by IBM [36] in their Qiskit API [74]. This noise model specifically characterises the publicly accessible 5-qubit IBM quantum device `ibmq_belem` [36]. We train five VQ-DQL agents using this noise model and present the results in Figure 5.8. It is evident that, based on our early-stopping criterion (cf. Section 5.3), the agents are able to achieve an optimal policy for the `CartPole-v0` environment in all

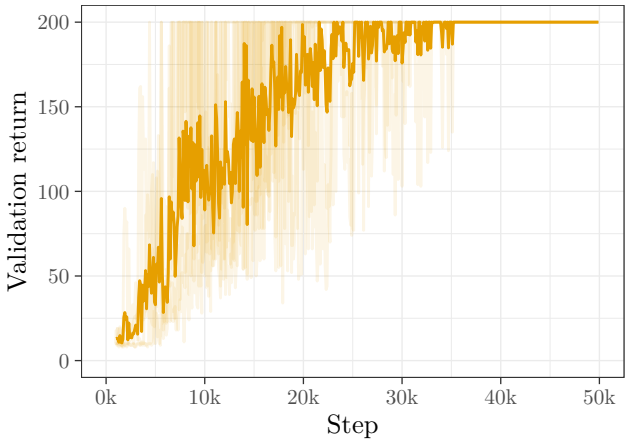


Figure 5.8.: Validation returns of VQ-DQL during the training process under the simulated `ibmq_belem` noise model. Noise was introduced during both training and validation stages. Results are averaged over five experiments. The experiments are based on the `CartPole-v0` environment, where the maximum achievable return value is 200.

five runs.

While the agents trained with noise take longer to reach optimal returns compared to the ideal case that is depicted in Figure 5.5, these results still indicate a certain degree of robustness of the VQ-DQL approach against noise. This finding aligns with the observations made in Ref. [70], where the authors also conducted noisy experiments using various noise models. Moreover, we hypothesise that further improvement in the performance of VQ-DQL can be achieved by searching for optimal hyperparameters. As demonstrated in Section 5.4.2, VQ-DQL is highly susceptible to classical hyperparameters, even in the ideal case.

Overall, this chapter provided a comprehensive experimental analysis of the VQ-DQL algorithm. We evaluated its performance under ideal and noisy conditions, compared it to classical approaches, and assessed its potential for real-world quantum computing applications. The findings and insights gained from these experiments are summarised and discussed in the following chapter.

6. Discussion and Outlook

In the previous chapter, we have systematically studied the performance of quantum-assisted reinforcement learning schemes on both simulators and physical quantum computers. Our findings revealed that, even in ideal simulation settings, variational quantum deep Q-learning (VQ-DQL) approaches exhibit instabilities leading to policy divergence. This chapter delves into the results obtained, examine the potential factors contributing to these instabilities, and propose potential strategies to mitigate them in future research.

In our approach of VQ-DQL we trained a VQC with a classical optimization loop. Such a setting is known to be prone to the *barren plateau* effect [77], which describes a problem of vanishing gradients that causes the inability to converge to an optimal return value. However, *barren plateaus* only occur in random VQCs. To counter randomness in the quantum circuits, we initialised all VQC parameters systematically to zeros. Since the VQCs in our experiments are neither very wide (four qubits), nor deep (five "layers"), randomness induced by gradient-based optimization is also limited. Therefore we rule out *barren plateaus* as the source of instabilities.

With the possibility of the barren plateau avoided, one can say that every agent with its unique architecture combinations and a reasonable encoding scheme is capable of learning the optimal policy to solve the `CartPole` environment. This can be seen from the results shown in section 5.4.2. The architecture combinations which did not learn an optimal policy during experiments conducted by different authors (Refs. [24] and [25]) showed a tendency to learn the optimal policy during our experiments. The reason why these agents show such a tendency is the selection of the right set of classical hyperparameters. The agents learned the optimal policy only for a few sets of classical hyperparameters during our hyperparameters search. This made us conclude that the VQ-DQL algorithms are highly sensitive to classical hyperparameters.

The results from sections 5.4.2 elucidate that the data re-uploading strategy does not always outperform its corresponding architecture without data re-uploading in sampling efficiency. One possible reason for this could be that the optimal hyperparameter set required for these architectures might fall outside the search space used in the experiments. One other possible reason for this underperformance can be inferred from the work of Schuld et al. [39]. Schuld et al. show that the function represented by a VQC is a Fourier sum. In particular, the variational layers determine the amplitudes and the encoding layers determine the frequency spectrum. As shown in Ref. [90], when it comes to data re-uploading strategy, the variational layers between two encoding layers might not be expressive enough which reduces the overall expressivity of the VQC. Increasing the number of variational layers can enhance expressivity, but this would require further investigation and architectural modifications in future studies.

Furthermore, we observe that in the current early stage of technological development, quantum computers, even in the absence of noise, do not exhibit clear advantages over classical approaches, as

indicated by the findings presented in Section 5.5.

Nonetheless, a number of constructive insights can be drawn from our experiments. Following previous work, we have trained models on classical simulators and only performed the execution step on quantum hardware. This approach, albeit practically necessitated by current-day hardware, creates a mis-match in terms of handling noise. Most importantly, our results do *not* corroborate observations made when reinforcement learning on quantum computers was first introduced into the literature in Ref. [23]: While the authors in this approach upload weights determined by classical training onto a quantum machine as we do in this paper, they find that executing the model does not vary much between simulation and NISQ machine. We, on the contrary, observe a total mismatch in performance. We expect the most probable explanation for this discrepancy to lie in (a) the size of the machine (five versus 27 qbits) and the problem of choice ("cognitive radio" [69] versus `CartPole` [53]; a random policy as would be caused by growing amounts of noise from NISQ devices is obviously better suited to the former than the latter).

To address the challenges posed by noisy systems, one potential strategy is to incorporate noise in the training process. Our results for simulated noise models from Section 5.6.2, along with previous studies [25], [71], [91] on small-scale systems, indicate that existing noise models can effectively bridge the gap between simulation and hardware, enabling a more accurate comparison of algorithmic performance. A logical next step would be to validate this statement by executing the VQ-DQL agent, trained with a simulated noise model on real hardware. Unfortunately, due to limited access to hardware, we were unable to pursue this validation in the scope of this thesis.

We encounter hindrances towards the practical application of quantum computers: Waiting time on queues in a shared, cloud-like environment is a major practical issue, which will however be alleviated with the broader availability of quantum chips. Nonetheless, the temporal contributions of sequential elements of algorithms to the overall computation time would also occur in a non-shared setting and do substantially increase wall-time run-times, which is an obvious impediment to practical utility.

As long as noise and imperfections are unavoidable, we propose a *hardware-software co-design* approach as a pathway to practical utility. This approach, as highlighted by recent studies [92]–[94], involves leveraging simulated quantum processing unit (QPU) designs that exhibit tunable and physically realistic noise behavior. By exploring optimal noise models and parameters within these inherent constraints, we can identify an "ideal" noise model that aligns with the characteristics of real-world quantum systems. Moreover, future QPUs can be designed to closely mimic the identified noise and imperfection behavior by carefully considering trade-off decisions during hardware design. In essence, we put forth the hypothesis that by recognizing the varying impact of hardware imperfections on different computations, there exists a degree of freedom within the realm of hardware design decisions. This freedom can be harnessed to develop custom algorithmic-specific hardware solutions that optimise performance in the presence of noise and imperfections, leading to more practical and effective quantum computing systems.

7. Conclusion

This thesis presented an empirical analysis of a recent class of variational hybrid approaches to quantum reinforcement learning, namely VQ-DQL. Our findings indicate that current VQ-DQL methods do not offer clear advantages over classical methods, particularly on current NISQ devices. However, these results highlight potential directions for future research.

One area of focus for future investigations is the adaptation of algorithms to address the challenges posed by noise and imperfections in quantum systems. By incorporating techniques that mitigate the impact of these factors, it may be possible to improve the practical utility of quantum reinforcement learning approaches.

Furthermore, the design of algorithmic-specific quantum hardware represents another promising approach. By tailoring hardware architectures to suit the requirements of quantum algorithms, it may be possible to leverage the unique properties of quantum computing and realise quantum advantage in specific problem domains.

Overall, this work emphasises the need for further research and development to unlock the full potential of quantum reinforcement learning. By addressing the challenges posed by noise and imperfections, and by considering the design of algorithmic-specific quantum hardware, we can pave the way for more effective and practical quantum-assisted learning algorithms.

Acknowledgements

The author acknowledges the use of IBM Quantum services for this work. The views expressed are those of the author, and do not reflect the official policy or position of IBM or the IBM Quantum team.

Funding: This work was supported by the German Federal Ministry of Education and Research (BMBF), funding program “Quantum Technologies – from Basic Research to Market”, grant numbers 13N15645 and 13N16092.

A. Validation Results

Table A.1.: Average validation results after training with early stopping criterion, collected over 100 validation episodes. The VQ-DQL agents were trained using the best-performing hyperparameter sets listed in Table 5.3 for each configuration of VQC architecture and encoding/extraction method.

Architecture	Baseline	Baseline + data re-uploading
[24]/C/GSP	198	197
[24]/C/GS	191	196
[24]/SC/GSP	189	200
[24]/SC/GS	192	198
[25]/C/GSP	-	195
[25]/C/GS	-	193
[25]/SC/GSP	200	200
[25]/SC/GS	200	198

B. Cross-Validation – Full Results

This appendix depicts the full result set of our cross-validation experiments introduced in Section 5.4.2. The VQ-DQL agents were trained for the `CartPole` environment using the early stopping criterion (cf. Section 5.3).

B.1. Baseline without data re-uploading

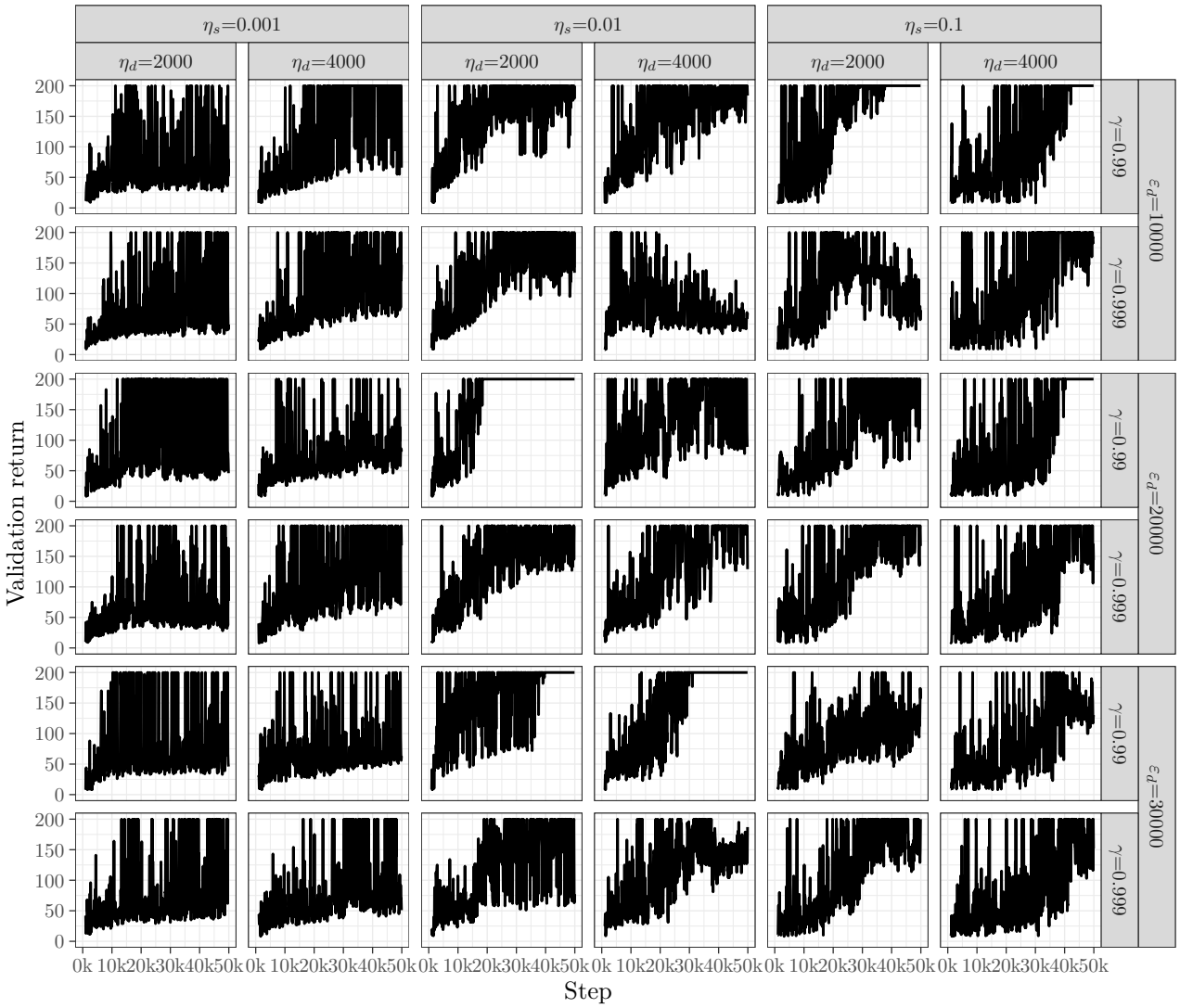


Figure B.1.: Cross-validation results for using the VQC architecture from Ref. [24], Continuous (C) input encoding and Global Scaling with quantum Pooling (GSP) for Q-value extraction (Baseline).

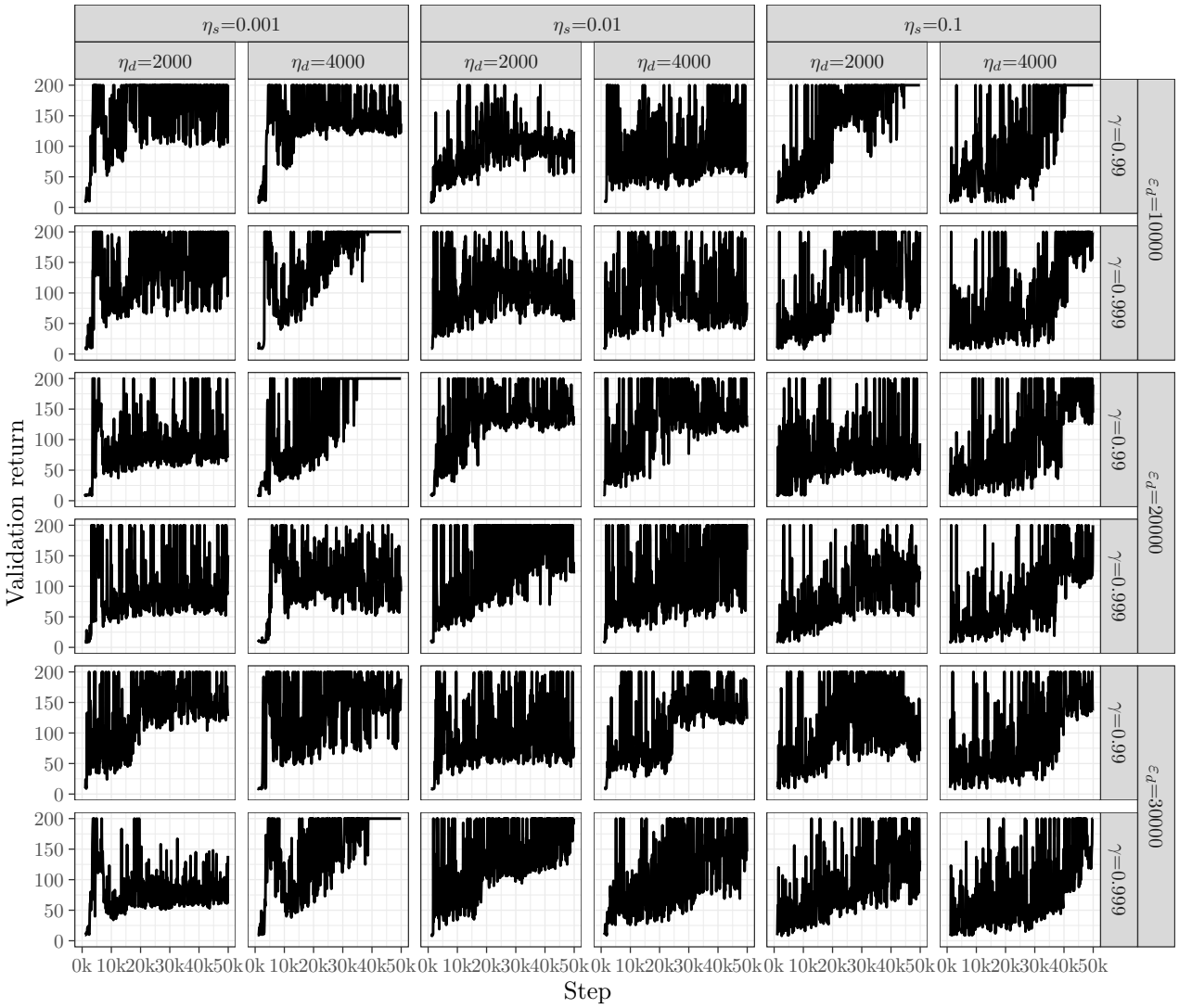


Figure B.2.: Cross-validation results for using the VQC architecture from Ref. [24], Continuous (C) input encoding and Global Scaling (GS) for Q-value extraction (Baseline).

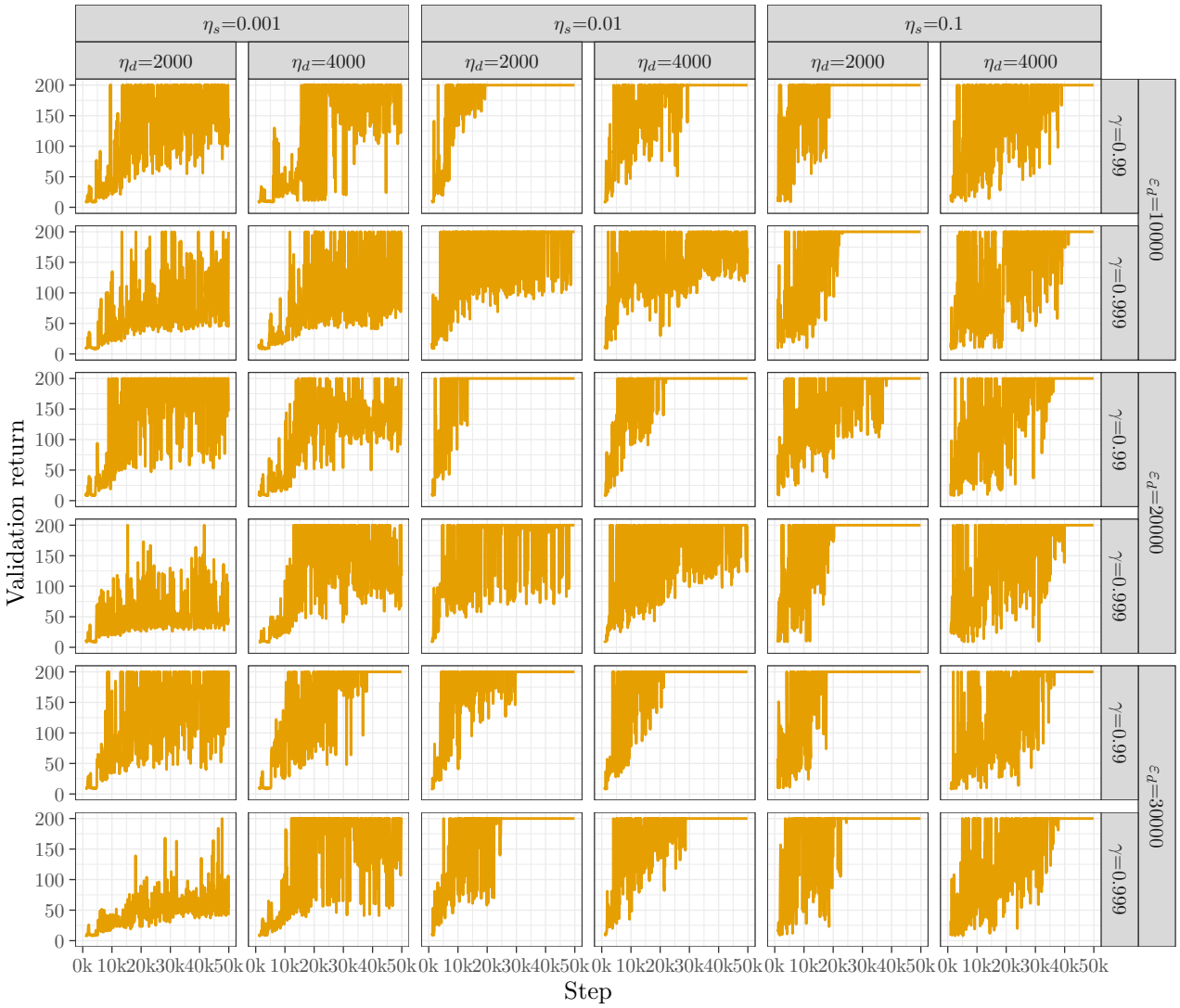


Figure B.3.: Cross-validation results for using the VQC architecture from Ref. [24], Scaled & Continuous (SC) input encoding and Global Scaling with quantum Pooling (GSP) for Q-value extraction (Baseline).

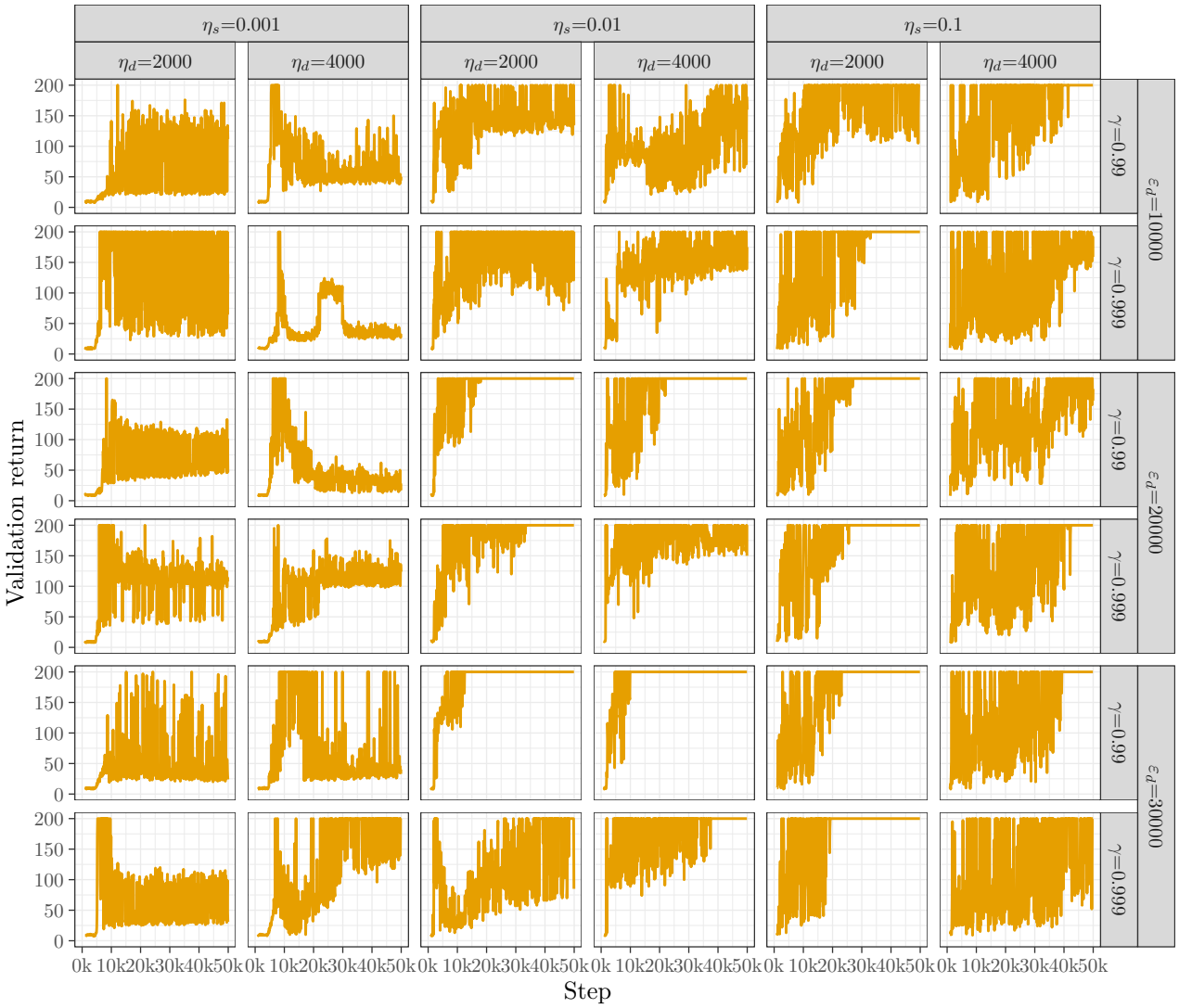


Figure B.4.: Cross-validation results for using the VQC architecture from Ref. [24], Scaled & Continuous (SC) input encoding and Global Scaling (GS) for Q-value extraction (Baseline).

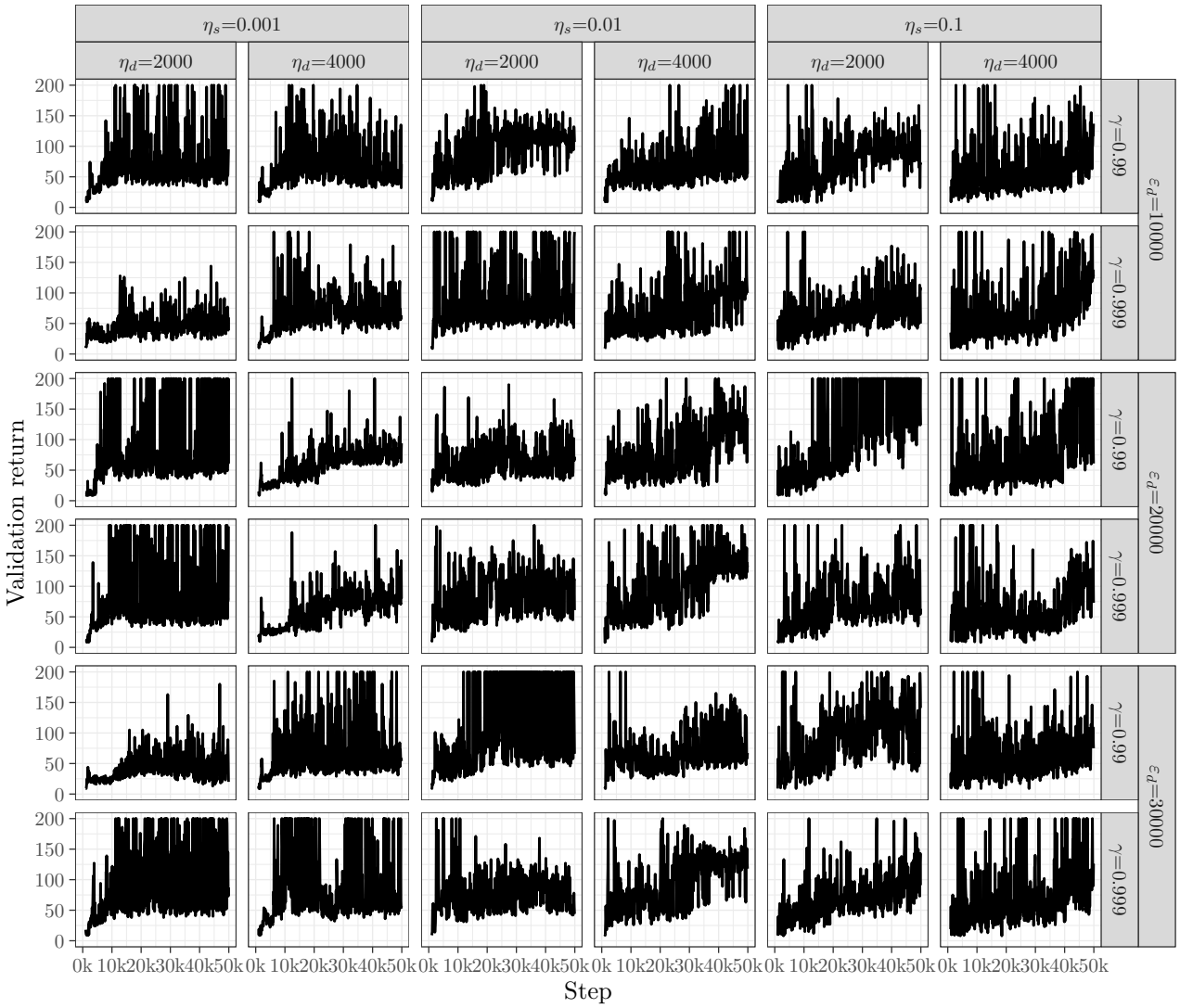


Figure B.5.: Cross-validation results for using the VQC architecture from Ref. [25], Continuous (C) input encoding and Global Scaling with quantum Pooling (GSP) for Q-value extraction (Baseline).

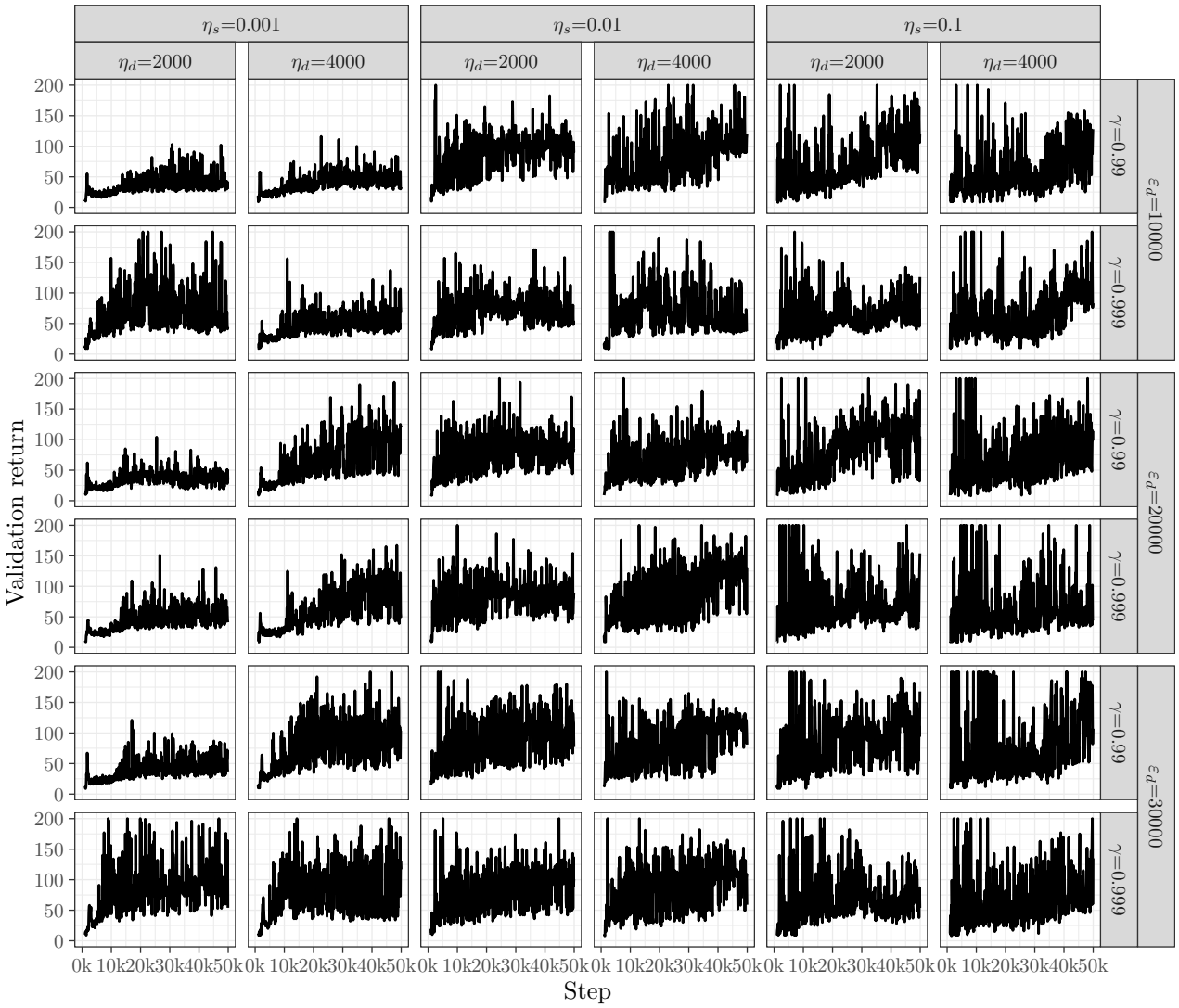


Figure B.6.: Cross-validation results for using the VQC architecture from Ref. [25], Continuous (C) input encoding and Global Scaling (GS) for Q-value extraction (Baseline).

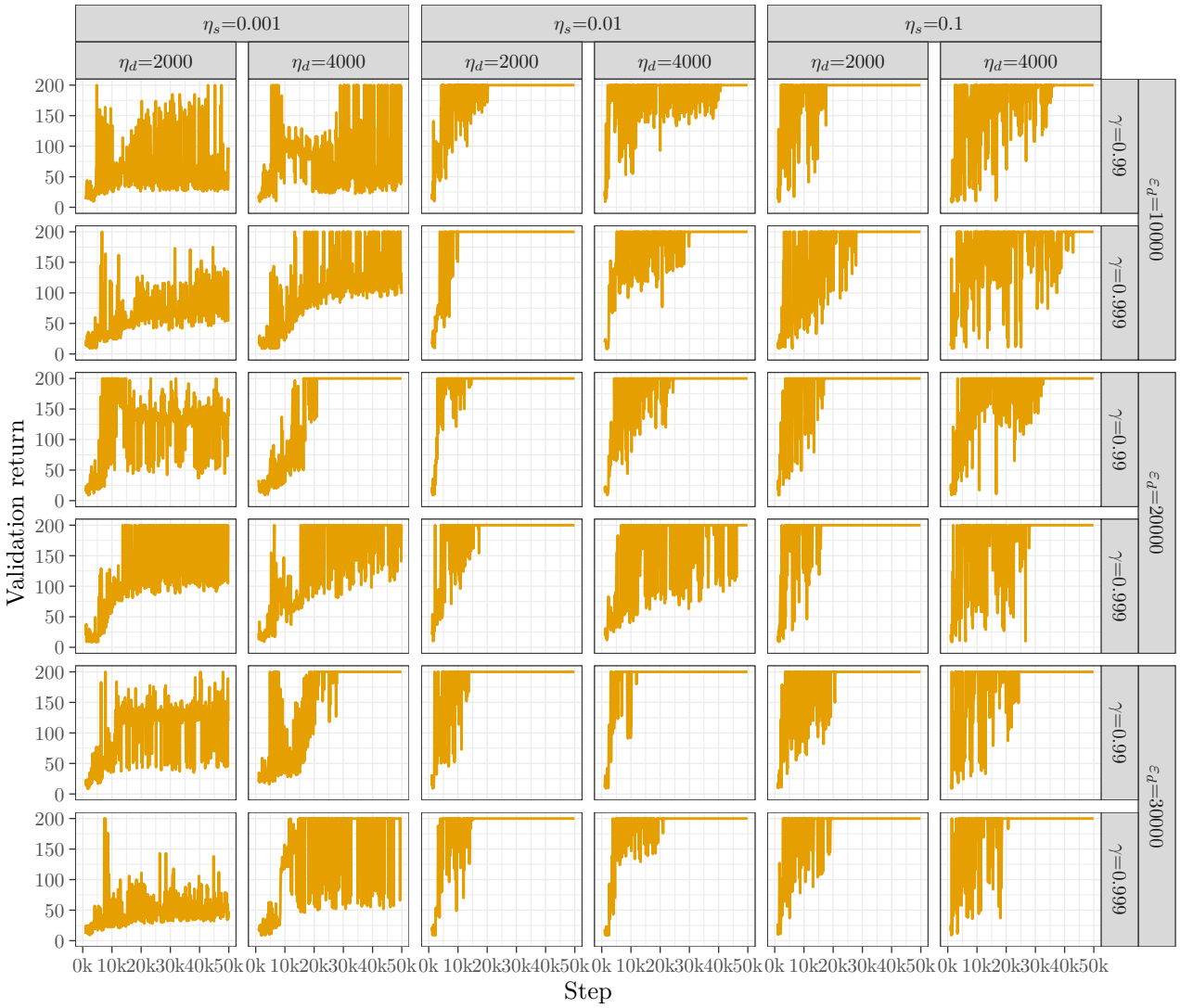


Figure B.7.: Cross-validation results for using the VQC architecture from Ref. [25], Scaled & Continuous (SC) input encoding and Global Scaling with quantum Pooling (GSP) for Q-value extraction (Baseline).

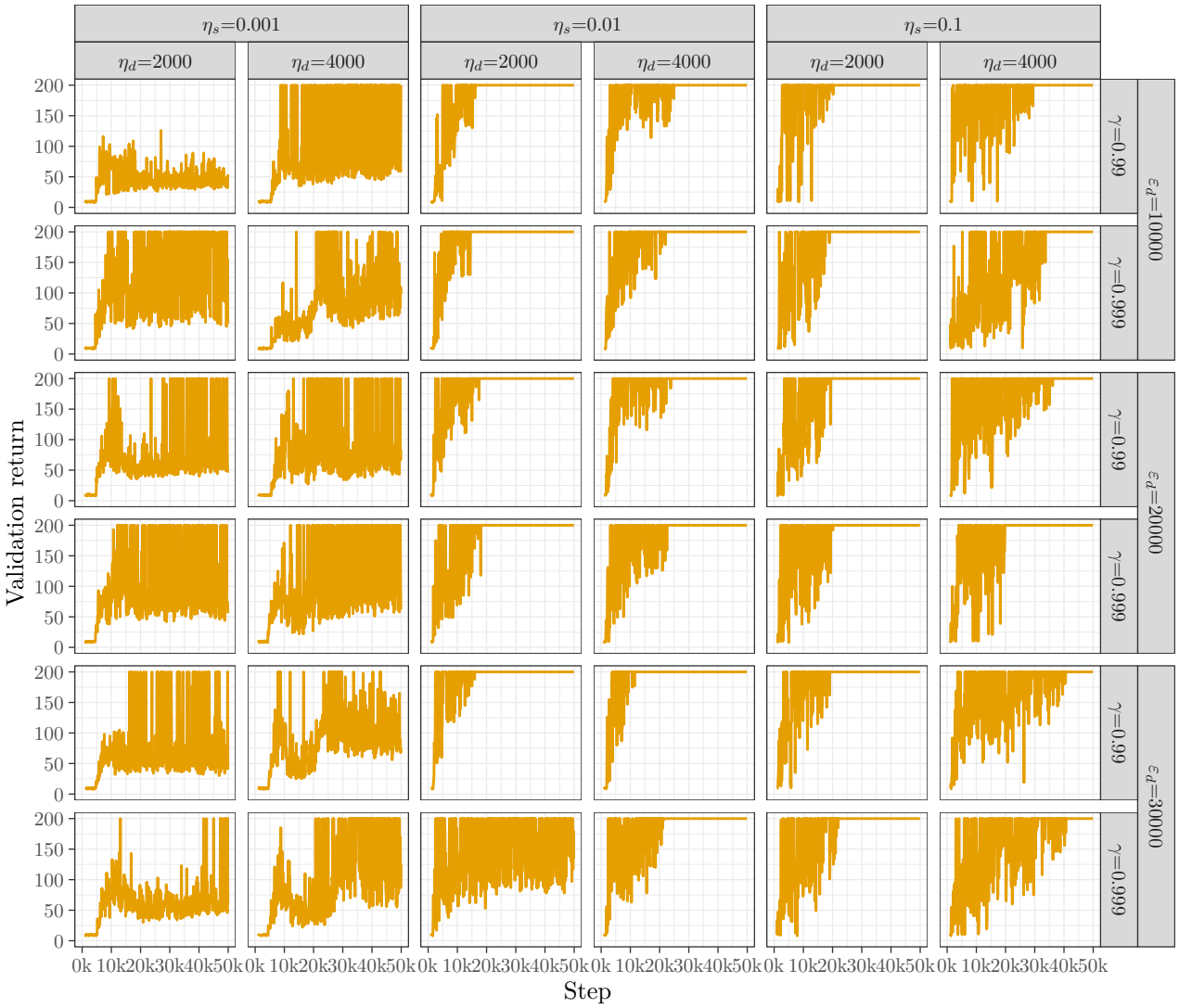


Figure B.8.: Cross-validation results for using the VQC architecture from Ref. [25], Scaled & Continuous (SC) input encoding and Global Scaling (GS) for Q-value extraction (Baseline).

B.2. Baseline with data re-uploading

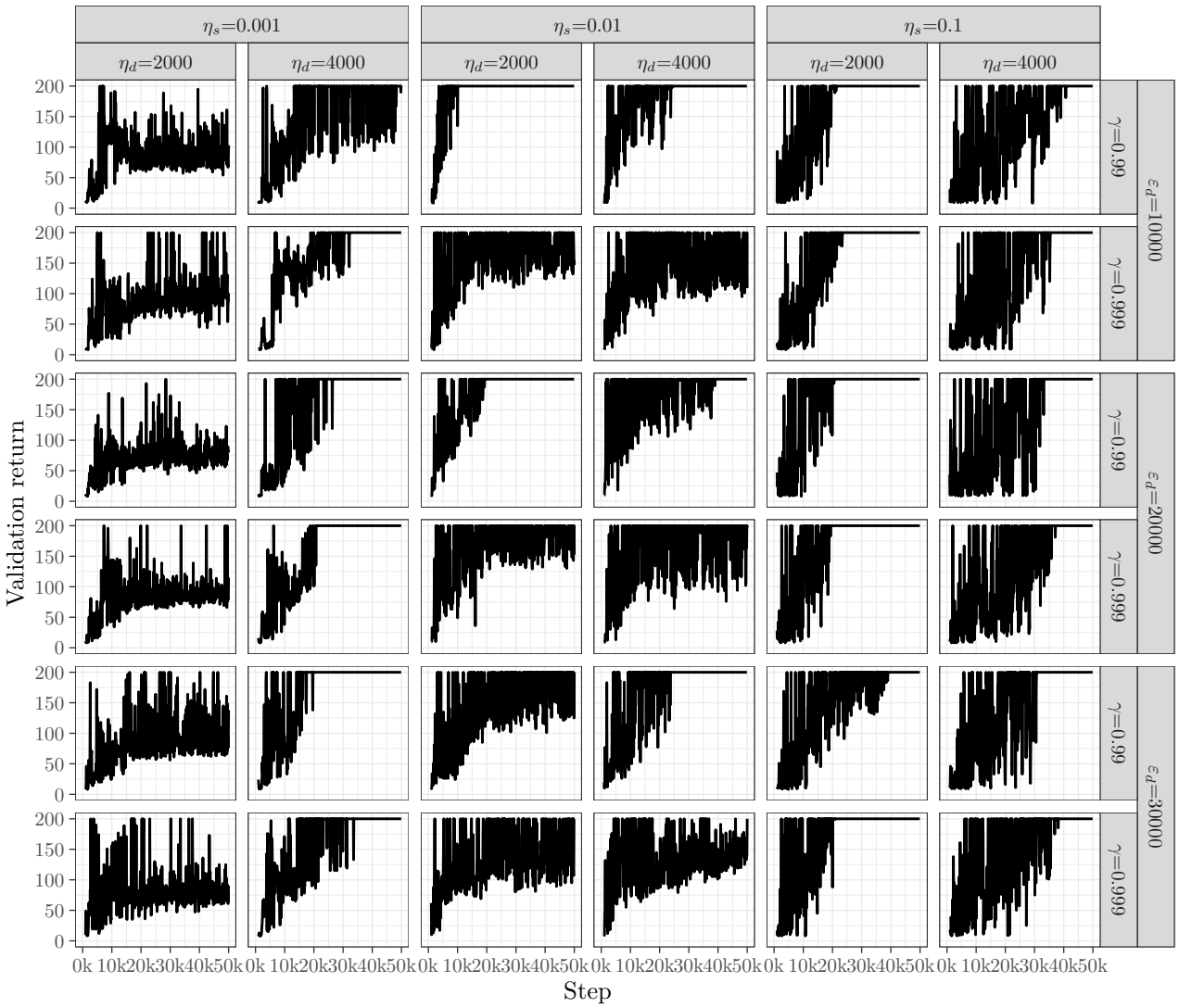


Figure B.9.: Cross-validation results for using the VQC architecture from Ref. [24], Continuous (C) input encoding and Global Scaling with quantum Pooling (GSP) for Q-value extraction (Baseline + data re-uploading).

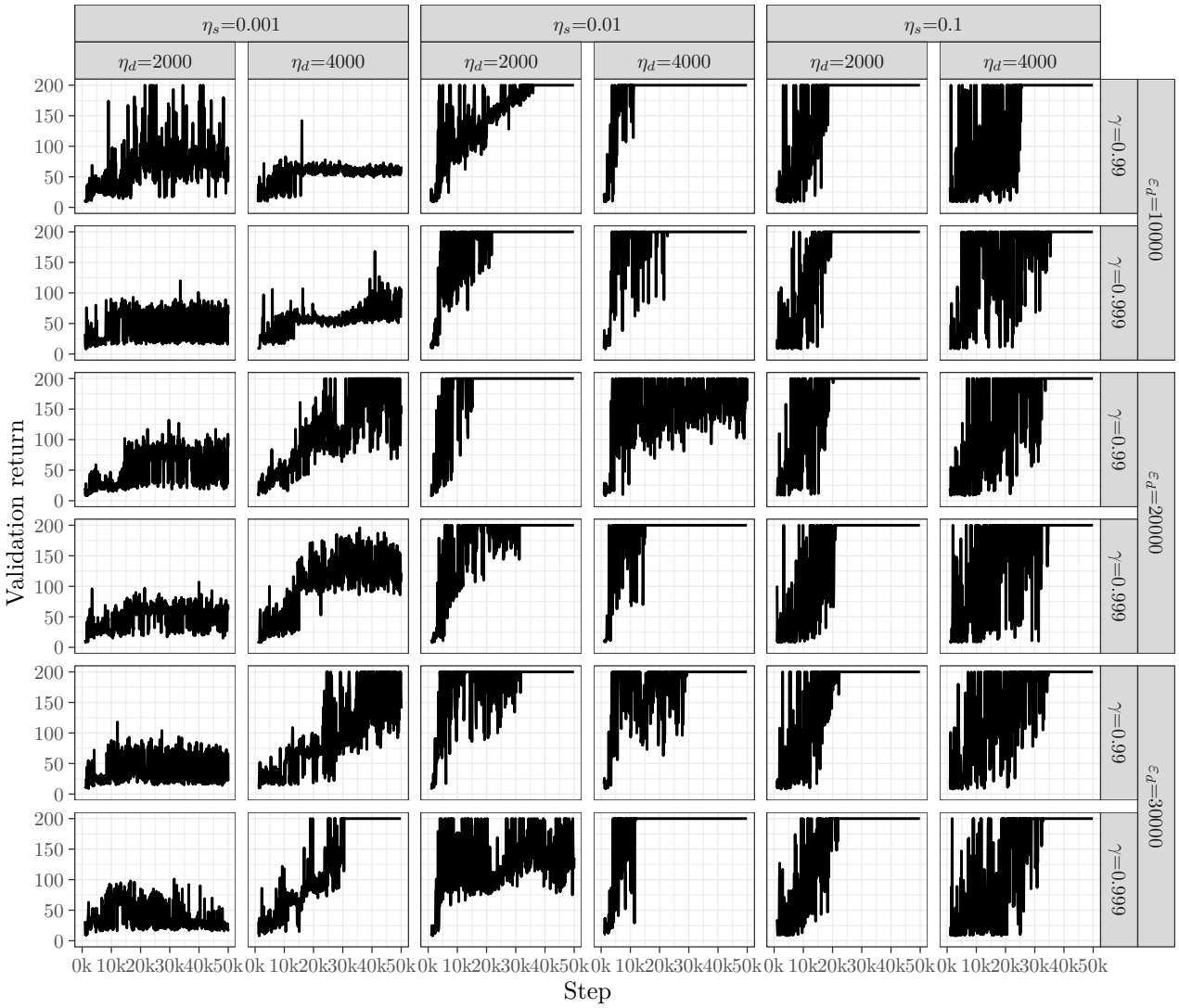


Figure B.10.: Cross-validation results for using the VQC architecture from Ref. [24], Continuous (C) input encoding and Global Scaling (GS) for Q-value extraction (Baseline + data re-uploading).

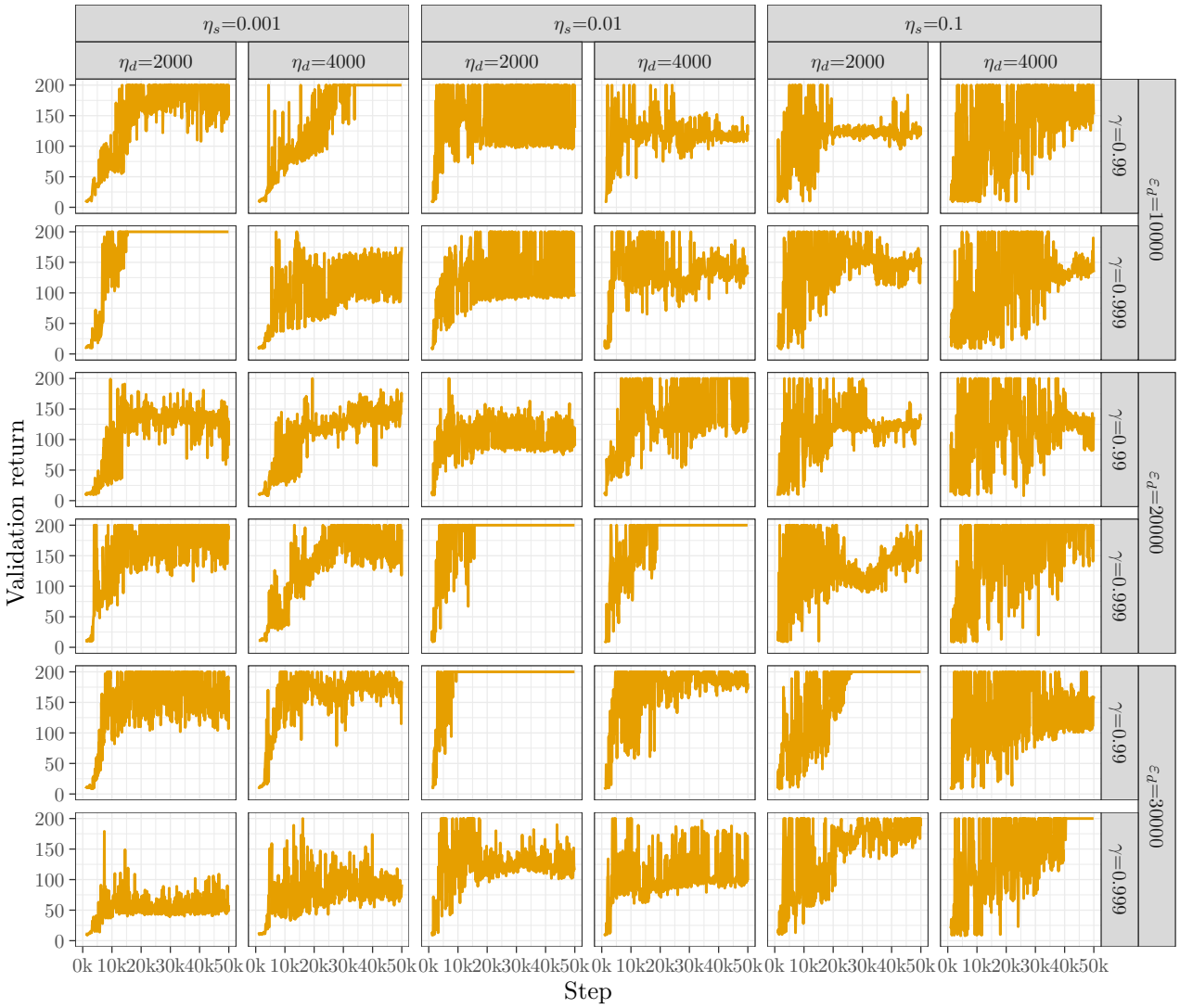


Figure B.11.: Cross-validation results for using the VQC architecture from Ref. [24], Scaled & Continuous (SC) input encoding and Global Scaling with quantum Pooling (GSP) for Q-value extraction (Baseline + data re-uploading).

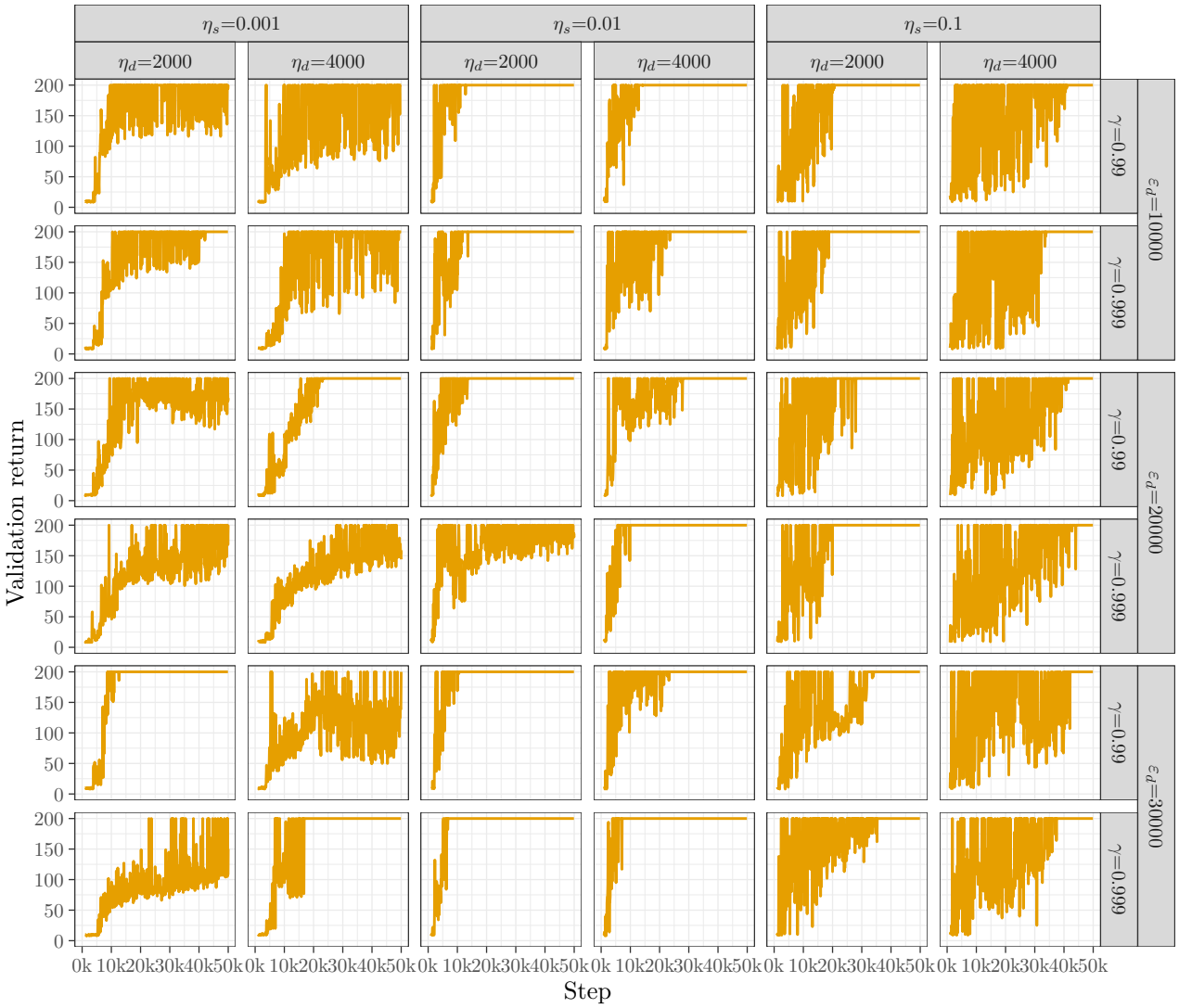


Figure B.12.: Cross-validation results for using the VQC architecture from Ref. [24], Scaled & Continuous (SC) input encoding and Global Scaling (GS) for Q-value extraction (Baseline + data re-uploading).

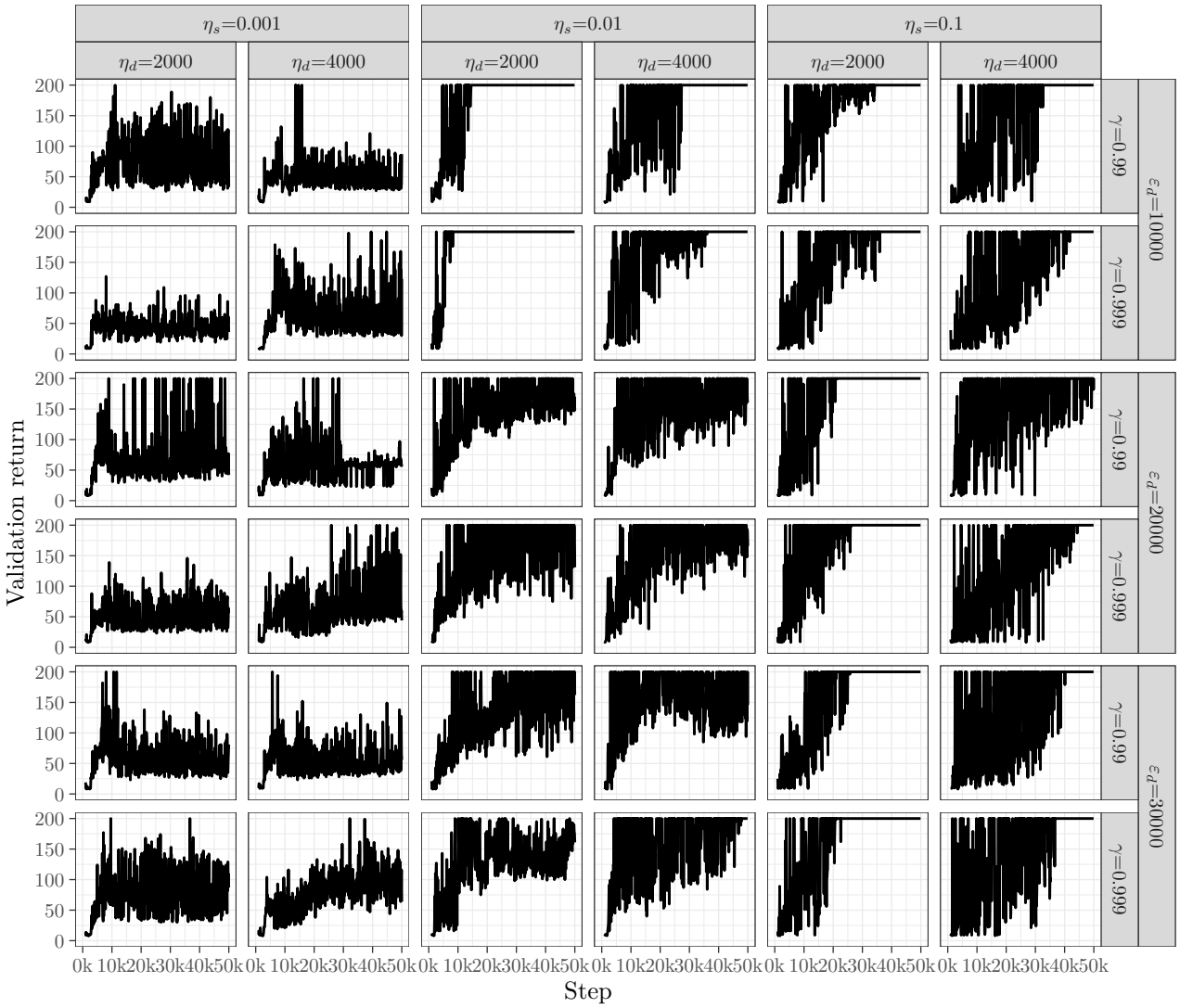


Figure B.13.: Cross-validation results for using the VQC architecture from Ref. [25], Continuous (C) input encoding and Global Scaling with quantum Pooling (GSP) for Q-value extraction (Baseline + data re-uploading).

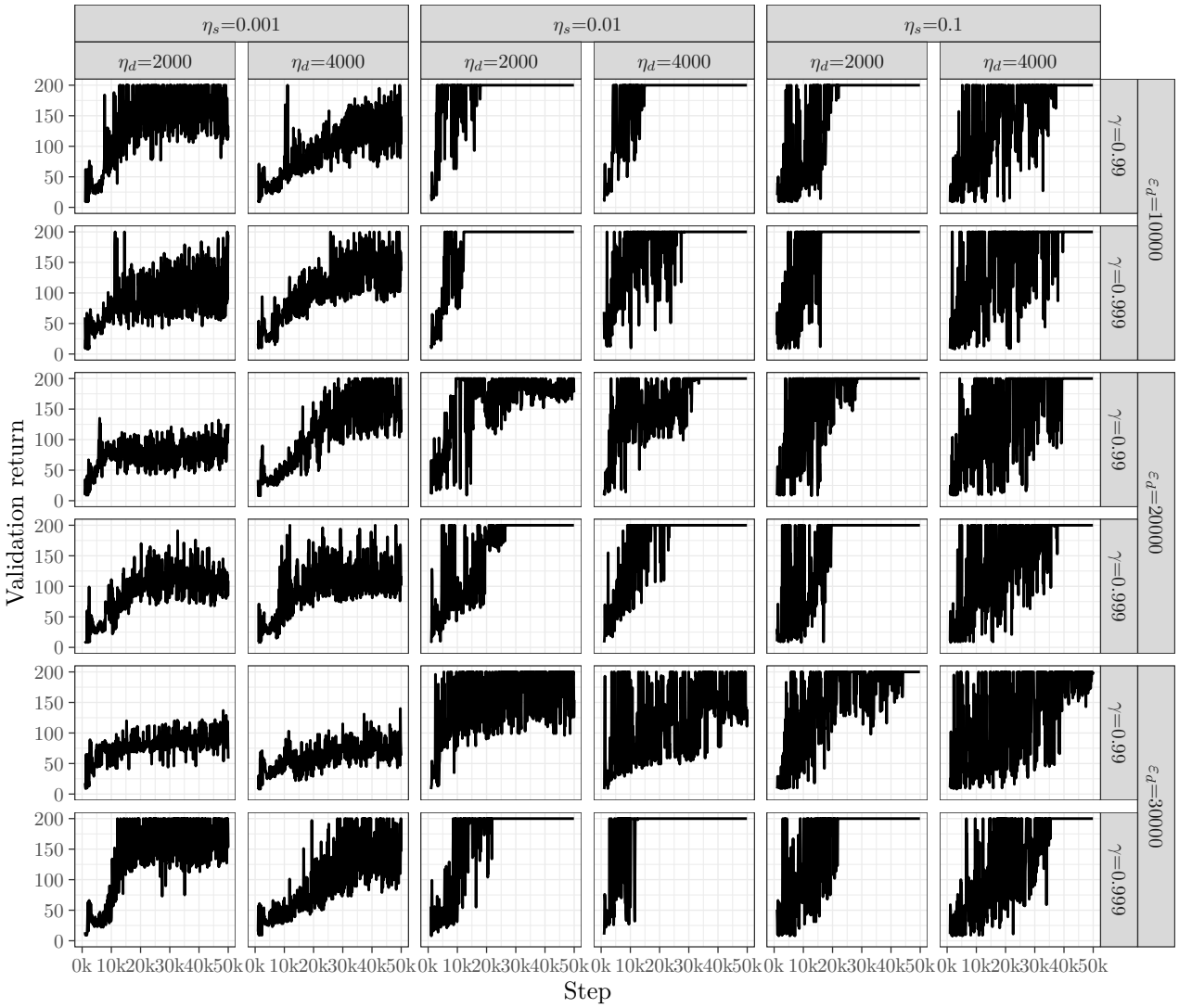


Figure B.14.: Cross-validation results for using the VQC architecture from Ref. [25], Continuous (C) input encoding and Global Scaling (GS) for Q-value extraction (Baseline + data re-uploading).

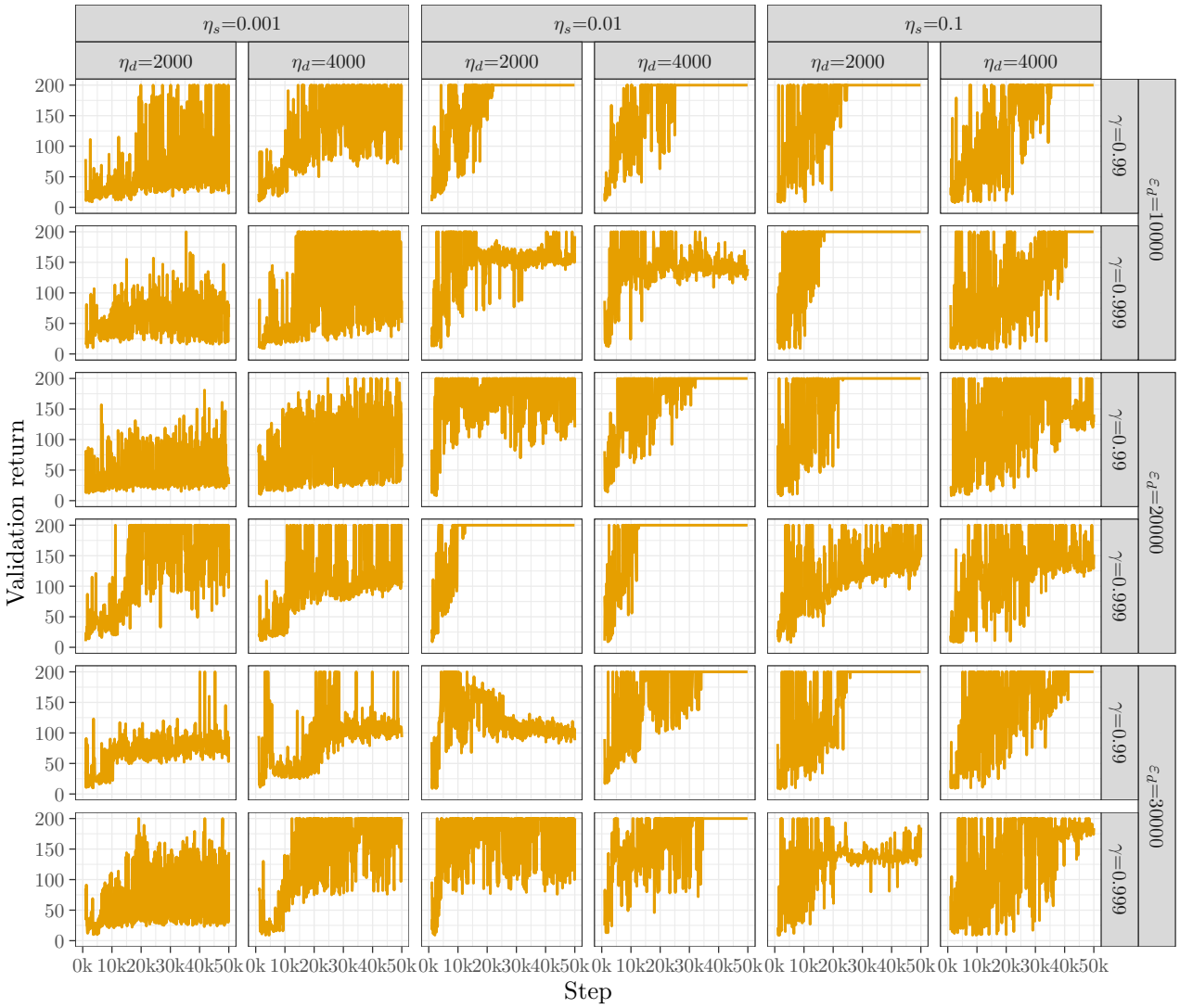


Figure B.15.: Cross-validation results for using the VQC architecture from Ref. [25], Scaled & Continuous (SC) input encoding and Global Scaling with quantum Pooling (GSP) for Q-value extraction (Baseline + data re-uploading).

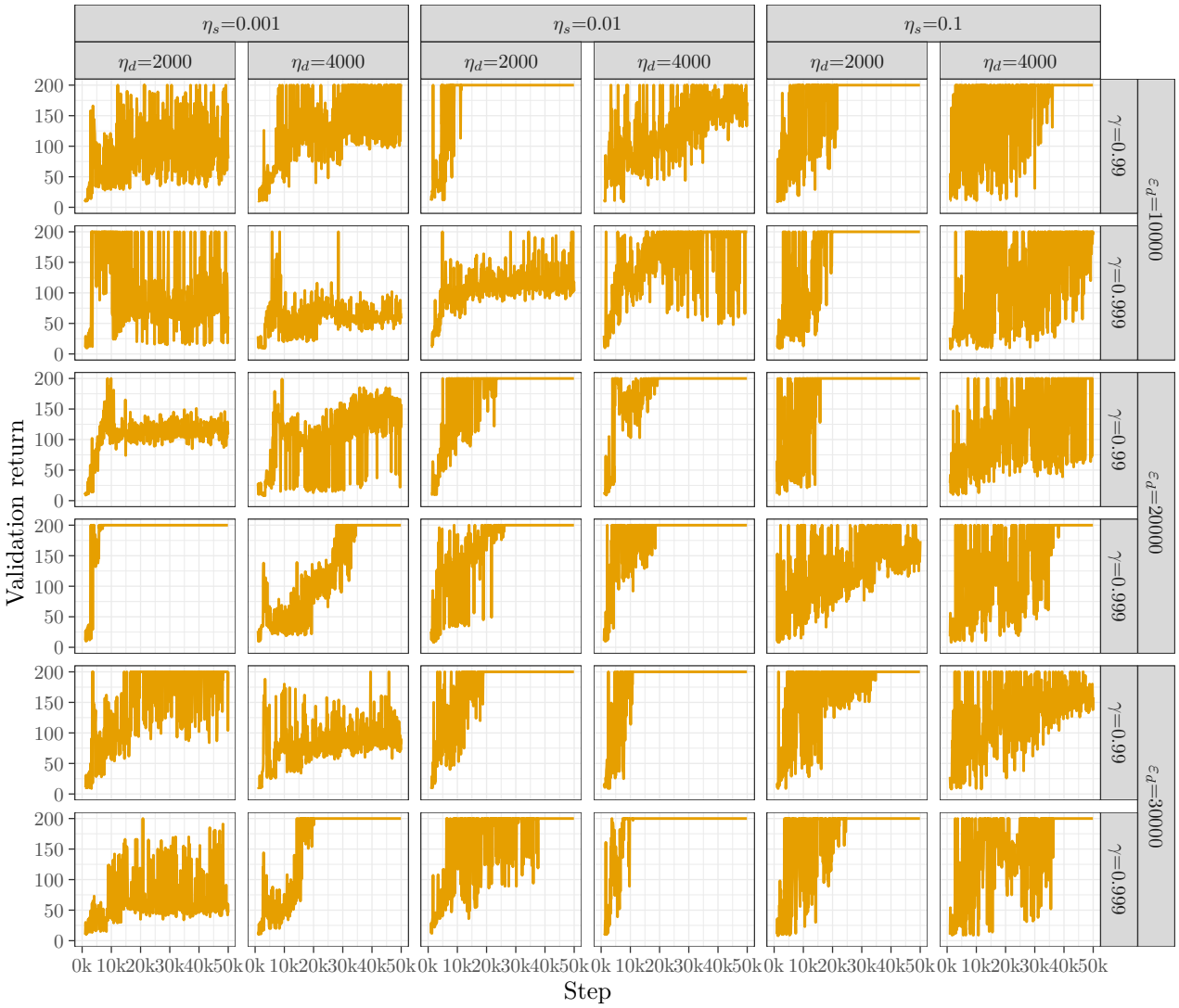


Figure B.16.: Cross-validation results for using the VQC architecture from Ref. [25], Scaled & Continuous (SC) input encoding and Global Scaling (GS) for Q-value extraction (Baseline + data re-uploading).

List of Figures

2.1. Bloch sphere representation of a single qubit. 4
2.2. Circuit symbols for the $CNOT$ gate and the CZ gate. 7
2.3. The structure of a variational hybrid quantum-classical algorithm using a VQC. 8
2.4. VQC architectures with and without data re-uploading. 10

3.1. Transition $(S_t, A_t, R_{t+1}, S_{t+1})$ in a MDP [8]. 15

5.1. VQC architectures from Refs. [24] and [25]. 24
5.2. Reproduced results from Refs. [24] and [25] 25
5.3. Validation returns for using the VQC-layer structure as in Ref. [24] and Ref. [25] with different input encoding strategies. 29
5.4. Validation returns for our best-performing hyperparameter constellations in the baseline configurations. 31
5.5. Comparison between a VQC and classical neural network 32
5.6. Results of our validation run on `ibmq_ehningen` [36]. 34
5.7. Validation returns of VQ-DQL during the training process under different noise variants and gate error rates. 35
5.8. Validation returns of VQ-DQL during the training process under the simulated `ibmq_belem` noise model. 36

List of Tables

5.1. State observations in `CartPole` 23
5.2. Hyperparameter settings for cross-validation. 30
5.3. Best-performing hyperparameters identified in cross-validation. 31

A.1. Validation results after training with early stopping criterion. 43

Bibliography

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search", in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219, ISBN: 0897917855. DOI: 10.1145/237814.237866.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999. DOI: 10.1137/S0036144598347011.

[3] J. Preskill, "Quantum Computing in the NISQ era and beyond", *Quantum*, vol. 2, p. 79, Aug. 2018, ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79.

[4] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, *et al.*, "Noisy intermediate-scale quantum algorithms", *Rev. Mod. Phys.*, vol. 94, p. 015 004, 1 Feb. 2022. DOI: 10.1103/RevModPhys.94.015004.

[5] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning", *Phys. Rev. A*, vol. 98, p. 032 309, 3 Sep. 2018. DOI: 10.1103/PhysRevA.98.032309.

[6] M. Franz, L. Wolf, M. Periyasamy, C. Ufrecht, D. D. Scherer, A. Plinge, C. Mutschler, and W. Mauerer, "Uncovering instabilities in variational-quantum deep q-networks", *Journal of the Franklin Institute*, 2022, ISSN: 0016-0032. DOI: 10.1016/j.jfranklin.2022.08.021.

[7] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum random access memory", *Phys. Rev. Lett.*, vol. 100, p. 160 501, 16 Apr. 2008. DOI: 10.1103/PhysRevLett.100.160501.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press Ltd, 2018, ISBN: 978-0-262-03924-6.

[9] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies", *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[10] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters, "Stable reinforcement learning with autoencoders for tactile and visual data", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 3928–3934. DOI: 10.1109/IROS.2016.7759578.

[11] D. Kalashnikov, A. Irpan, P. Pastor, *et al.*, *Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation*, 2018. arXiv: 1806.10293 [cs.LG].

[12] S. Bhalla, S. Ganapathi Subramanian, and M. Crowley, "Deep multi agent reinforcement learning for autonomous driving", in *Advances in Artificial Intelligence*, C. Goutte and X. Zhu, Eds., Cham: Springer International Publishing, 2020, pp. 67–78, ISBN: 978-3-030-47358-7.

[13] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge", *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[14] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model", *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[15] I. Trummer, J. Wang, Z. Wei, D. Maram, S. Moseley, S. Jo, J. Antonakakis, and A. Rayabhari, "SkinnerDB: Regret-bounded query evaluation via reinforcement learning", *ACM Transactions on Database Systems*, vol. 46, no. 3, 9:1–9:45, 2021, ISSN: 0362-5915. DOI: 10.1145/3464389.

[16] R. Marcus and O. Papaemmanouil, "Deep reinforcement learning for join order enumeration", in *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, Houston TX USA: ACM, 2018, pp. 1–4, ISBN: 978-1-4503-5851-4. DOI: 10.1145/3211954.3211957.

[17] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the Atari human benchmark", in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, Jul. 2020, pp. 507–517.

[18] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents", *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, Jun. 2013. DOI: 10.1613/jair.3912.

[19] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, 2015. arXiv: 1509.06461 [cs.LG].

[20] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, *Deep reinforcement learning and the deadly triad*, 2018. arXiv: 1812.02648 [cs.AI].

[21] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, *A closer look at deep policy gradients*, 2020. arXiv: 1811.02553 [cs.LG].

[22] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan, *On the theory of policy gradient methods: Optimality, approximation, and distribution shift*, 2020. arXiv: 1908.00261 [cs.LG].

[23] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, "Variational quantum circuits for deep reinforcement learning", *IEEE Access*, vol. 8, pp. 141 007–141 024, 2020. DOI: 10.1109/ACCESS.2020.3010470.

[24] O. Lockwood and M. Si, "Reinforcement learning with quantum variational circuit", in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, 2020, pp. 245–251.

[25] A. Skolik, S. Jerbi, and V. Dunjko, "Quantum agents in the Gym: A variational quantum algorithm for deep Q-learning", *Quantum*, vol. 6, p. 720, May 2022, ISSN: 2521-327X. DOI: 10.22331/q-2022-05-24-720.

[26] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information - 10th Anniversary Edition*. Cambridge: Cambridge University Press, 2010, ISBN: 978-1-107-00217-3.

[27] P. A. M. Dirac, *The principles of quantum mechanics*. Oxford university press, 1981.

[28] E. G. Rieffel and W. H. Polak, *Quantum Computing - A Gentle Introduction*. Cambridge: MIT Press, 2014, ISBN: 978-0-262-52667-8.

[29] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?", *Physical review*, vol. 47, no. 10, p. 777, 1935.

[30] J. S. Bell, "On the einstein podolsky rosen paradox", *Physics Physique Fizika*, vol. 1, no. 3, p. 195, 1964.

[31] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, *et al.*, "Variational quantum algorithms", *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.

[32] B. Polyak, *Introduction to Optimization*. Optimization Software, Publications Division, 1987, ISBN: 978-0-911-57514-9.

[33] T. Winker, S. Groppe, V. J. E. Uotila, Z. Yan, J. Lu, M. Franz, and W. Mauerer, "Quantum machine learning: Foundation, new techniques, and opportunities for database research", in *Proceedings of ACM SIGMOD/PODS International Conference on Management of Data*, (Seattle, USA), Jun. 2023.

[34] M. Weigold, J. Barzen, F. Leymann, and M. Salm, "Data encoding patterns for quantum computing", in *Proceedings of the 27th Conference on Pattern Languages of Programs*, ser. PLoP '20, Virtual Event: The Hillside Group, 2022, ISBN: 9781941652169.

[35] M. Weigold, J. Barzen, F. Leymann, and M. Salm, "Encoding patterns for quantum algorithms", *IET Quantum Communication*, vol. 2, no. 4, pp. 141–152, 2021.

[36] IBM. "IBM Quantum". (2023), [Online]. Available: https://quantum-computing.ibm.com.

[37] M. Mottonen and J. J. Vartiainen, *Decompositions of general quantum gates*, 2005. arXiv: quant-ph/0504100 [quant-ph].

[38] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, "Data re-uploading for a universal quantum classifier", *Quantum*, vol. 4, p. 226, Feb. 2020, ISSN: 2521-327X. DOI: 10.22331/q-2020-02-06-226.

[39] M. Schuld, R. Sweke, and J. J. Meyer, "Effect of data encoding on the expressive power of variational quantum-machine-learning models", *Phys. Rev. A*, vol. 103, p. 032 430, 3 Mar. 2021. DOI: 10.1103/PhysRevA.103.032430.

[40] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms", *New Journal of Physics*, vol. 18, no. 2, p. 023 023, Feb. 2016. DOI: 10.1088/1367-2630/18/2/023023.

[41] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware", *Phys. Rev. A*, vol. 99, p. 032 331, 3 Mar. 2019. DOI: 10.1103/PhysRevA.99.032331.

[42] A. Mari, T. R. Bromley, and N. Killoran, "Estimating the gradient and higher-order derivatives on quantum hardware", *Phys. Rev. A*, vol. 103, p. 012 405, 1 Jan. 2021. DOI: 10.1103/PhysRevA.103.012405.

[43] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

[44] F. Greiwe, T. Krueger, and W. Mauerer, "Effects of imperfections on quantum algorithms: A software engineering perspective", en, in *Proceedings of the IEEE Quantum Software Week*, 2023.

[45] A. Markov and N. Nagorny, *The Theory of Algorithms* -. Dortrecht Heidelberg London New York: Springer Netherlands, 1988, ISBN: 978-9-027-72773-2.

[46] C. J. Watkins and P. Dayan, "Q-learning", *Machine learning*, vol. 8, pp. 279–292, 1992.

[47] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning", *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[48] F. S. Melo, "Convergence of q-learning: A simple proof", *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2001.

[49] Y. Dodge, *Mean Squared Error*. New York, NY: Springer New York, 2008, pp. 337–339, ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_251.

[50] S.-i. Amari, "Backpropagation and stochastic gradient descent method", *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[51] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning", in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937.

[52] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching", *Machine learning*, vol. 8, pp. 293–321, 1992. DOI: 10.1007/BF00992699.

[53] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016. arXiv: 1606.01540 [cs.LG].

[54] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning", in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, Jun. 2016, pp. 1995–2003.

[55] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, *Prioritized experience replay*, 2016. arXiv: 1511.05952 [cs.LG].

[56] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, *Distributed prioritized experience replay*, 2018. arXiv: 1803.00933 [cs.LG].

[57] A. Nair, P. Srinivasan, S. Blackwell, *et al.*, *Massively parallel methods for deep reinforcement learning*, 2015. arXiv: 1507.04296 [cs.LG].

[58] A. P. Badia, P. Sprechmann, A. Vitvitskyi, *et al.*, *Never give up: Learning directed exploration strategies*, 2020. arXiv: 2002.06038 [cs.LG].

[59] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning", in *International conference on learning representations*, 2019.

[60] J. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation", *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997. DOI: 10.1109/9.580874.

[61] H. Hasselt, "Double q-learning", in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23, Curran Associates, Inc., 2010.

[62] R. S. Sutton, A. R. Mahmood, and M. White, "An emphatic approach to the problem of off-policy temporal-difference learning", *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2603–2631, Jan. 2016, ISSN: 1532-4435.

[63] D. Dong, C. Chen, H. Li, and T.-J. Tarn, "Quantum reinforcement learning", *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 5, pp. 1207–1220, 2008. DOI: 10.1109/TSMCB.2008.925743.

[64] V. Dunjko, J. M. Taylor, and H. J. Briegel, *Framework for learning agents in quantum environments*, 2015. arXiv: 1507.08482 [quant-ph].

[65] F. Flamini, A. Hamann, S. Jerbi, L. M. Trenkwalder, H. P. Nautrup, and H. J. Briegel, "Photonic architecture for reinforcement learning", *New Journal of Physics*, vol. 22, no. 4, p. 045 002, Apr. 2020. DOI: 10.1088/1367-2630/ab783c.

[66] F. Neukart, D. Von Dollen, C. Seidel, and G. Compostella, "Quantum-enhanced reinforcement learning for finite-episode games with discrete state spaces", *Frontiers in physics*, vol. 5, p. 71, 2018. DOI: 10.48550/arXiv.1708.09354.

[67] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms", in *Proceedings of the 31st International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., ser. Proceedings of Machine Learning Research, vol. 32, Bejing, China: PMLR, 22–24 Jun 2014, pp. 387–395.

[68] S. Jerbi, C. Gyurik, S. Marshall, H. Briegel, and V. Dunjko, "Parametrized quantum policies for reinforcement learning", *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 362–28 375, 2021.

[69] P. Gawłowicz and A. Zubow, "Ns-3 meets openai gym: The playground for machine learning in networking research", in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM '19, Miami Beach, FL, USA: Association for Computing Machinery, 2019, pp. 113–120, ISBN: 9781450369046. DOI: 10.1145/3345768.3355908.

[70] A. Skolik, S. Mangini, T. Bäck, C. Macchiavello, and V. Dunjko, *Robustness of quantum reinforcement learning under hardware errors*, 2022. arXiv: 2212.09431 [quant-ph].

[71] K. Borras, S. Y. Chang, L. Funcke, *et al.*, "Impact of quantum noise on the training of quantum generative adversarial networks", *Journal of Physics: Conference Series*, vol. 2438, no. 1, p. 012 093, Feb. 2023. DOI: 10.1088/1742-6596/2438/1/012093.

[72] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG].

[73] M. Broughton, G. Verdon, T. McCourt, *et al.*, *Tensorflow quantum: A software framework for quantum machine learning*, 2021. arXiv: 2003.02989 [quant-ph].

[74] Qiskit contributors, *Qiskit: An open-source framework for quantum computing*, 2023. DOI: 10.5281/zenodo.2573505.

[75] M. Schönberger, M. Franz, S. Scherzinger, and W. Mauerer, "Peel | pile? cross-framework portability of quantum software", in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, 2022, pp. 164–169. DOI: 10.1109/ICSA-C54293.2022.00039.

[76] B. Baker, O. Gupta, N. Naik, and R. Raskar, *Designing neural network architectures using reinforcement learning*, 2017. arXiv: 1611.02167 [cs.LG].

[77] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes", *Nature Communications*, vol. 9, no. 1, p. 4812, 2018, ISSN: 2041-1723. DOI: 10.1038/s41467-018-07090-4.

[78] A. Skolik, J. R. McClean, M. Mohseni, P. van der Smagt, and M. Leib, "Layerwise learning for quantum neural networks", *Quantum Machine Intelligence*, vol. 3, pp. 1–11, 2021. DOI: 10.1007/s42484-020-00036-4.

[79] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[80] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners", in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[81] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need", in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017.

[82] Y. Bengio, *Practical recommendations for gradient-based training of deep architectures*, 2012. arXiv: 1206.5533 [cs.LG].

[83] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[84] S. M. Kakade, *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

[85] Student, "The probable error of a mean", *Biometrika*, vol. 6, no. 1, pp. 1–25, 1908.

[86] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters", in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018. DOI: https://doi.org/10.1609/aaai.v32i1.11694.

[87] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, and B. R. Johnson, *Quality, speed, and scale: Three key attributes to measure the performance of near-term quantum computers*, 2021. arXiv: 2110.14108 [quant-ph].

[88] K.-N. Schymik, V. Lienhard, D. Barredo, P. Scholl, H. Williams, A. Browaeys, and T. Lahaye, "Enhanced atom-by-atom assembly of arbitrary tweezer arrays", *Phys. Rev. A*, vol. 102, p. 063 107, 6 Dec. 2020. DOI: 10.1103/PhysRevA.102.063107.

[89] S. Ebadi, T. T. Wang, H. Levine, A. Keesling, G. Semeghini, A. Omran, D. Bluvstein, R. Samajdar, H. Pichler, W. W. Ho, *et al.*, "Quantum phases of matter on a 256-atom programmable quantum simulator", *Nature*, vol. 595, no. 7866, pp. 227–232, 2021. DOI: PhysRevA.102.063107.

[90] M. Periyasamy, N. Meyer, C. Ufrecht, D. D. Scherer, A. Plinge, and C. Mutschler, "Incremental data-uploading for full-quantum classification", in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2022, pp. 31–37. DOI: 10.1109/QCE53715.2022.00021.

[91] N. Meyer, "Variational Quantum Circuits for Policy Approximation", M.S. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Nuremberg, Germany, 2021.

[92] M. Schönberger, S. Scherzinger, and W. Mauerer, "Ready to leap (by co-design)? Join order optimisation on quantum hardware", in *Proceedings of ACM SIGMOD/PODS International Conference on Management of Data*, 2023.

[93] K. Wintersperger, H. Safi, and W. Mauerer, "QPU-system co-design for quantum HPC accelerators", in *Architecture of Computing Systems*, M. Schulz, C. Trinitis, N. Papadopoulou, and T. Pionteck, Eds., Cham: Springer International Publishing, 2022, pp. 100–114, ISBN: 978-3-031-21867-5.

[94] H. Safi, K. Wintersperger, and W. Mauerer, "Influence of HW-SW-co-design on quantum computing scalability", en, in *Proceedings of the IEEE Quantum Software Week*, 2023.

Erklärung zur Masterarbeit

1.

Mir ist bekannt, dass dieses Exemplar der Abschlussarbeit als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.

2.

Ich erkläre hiermit, dass ich diese Abschlussarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Regensburg, den 23. Mai 2023

Maja Franz