Quantum Reinforcement Learning to Solve Cart Pole Environment

Maria Gabriela Jordão Oliveira (PG 50599) [1] and Miguel Caçador Peixoto[1] (PG 50657)

1 Mestrado em Engenharia Física, Campus de Gualtar, Universidade do Minho, 4701-057 Braga, Portugal

June 14, 2023

Abstract

A systematic study on solving the Cart Pole environment using quantum reinforcement learning is conducted and benchmarked against a deep neural network classical model. The quantum models studied differ in the type and number of entanglement layers and the use of data re-uploading. Varying the number of layers of the deep neural network, too, it is found that for less than four layers the classical model seems to be at least compatible with the quantum one, and for more that the quantum one is superior. The best performance found both in training and testing belongs to a quantum model.

1 Introduction

In recent years, as science and technology evolve, a growing interest in exploring quantum mechanics has emerged. Quantum computing (QC) appeared, finding applications in various fields, videlicet physics simulation [1], quantum machine learning (QML) [2], cryptography [3], medicine [4], etc. Concurrently, machine learning (ML), particularly reinforcement learning (RL), has emerged as a powerful paradigm. The crux of RL algorithms is to optimize an agent's behavior through trial and error, but they face challenges in efficiency and scalability for complex tasks [5]. As a result, there is a rising interest in combining RL with QC to leverage the unique properties of quantum systems to improve learning algorithms.

Concerning quantum computation, variational quantum circuits (VQCs) - quantum circuits parameterized by optimizable classical variables - promise the potential to solve RL tasks, and so, they are explored in this article. In the context of RL, the policy-gradient method is commonly used to optimize the behavior of the agent, i.e. to maximize the reward. This method has been shown effective in RL tasks [6, 7], including complex control problems and game playing. Hence, merging this method with VQCs aims to take advantage of the quantum systems' computational power to improve the learning process. In this merged approach, the quantum circuit stands for the agent's policy and the goal is to find the optimal set of parameters that maximize the expected reward.

Even though QC, and QML, promise to solve complex tasks and overcome the actual barriers of classical computation, the field is still in its infancy, and so, the devices and their access are far from perfect. In other words, although it's expected to achieve a fault-tolerant era [8], currently the devices belong to the Noise Intermediate Scale Quantum era, so the number of qubits is limited, the access to devices is prostated and they severally suffer from noise problems.

After presenting the state of the art, the goal is to solve the Cart Pole environment using quantum RL. Furthermore, a systemic study on this problem is made and the best model is bench-marked against a classical deep neural network model that solves this exact problem. This article is then divided into 5 more sections. In section 2 and section 3 the theoretical fundamentals of QML and RL are exposed. Following, in subsection 4.4, the methodology followed in the systematic study is presented, and, in section 5, the results are exposed and discussed. Finally, in section 6 the conclusions are taken.

2 Quantum Machine Learning

QML is an interdisciplinary field that combines the principles of QC and ML to develop algorithms and models and tries to enhance the capabilities of artificial intelligence systems to solve complex problems.

In a fault-tolerant world, QML aims to surpass classical ML through the use of intrinsic quantum phenomena, namely superposition and entanglement, and the exploitation of quantum parallelism to perform computations on multiple possible states simultaneously.

Some of the most studied QML algorithms and techniques include quantum support vector machines, quantum neural networks, quantum variational algorithms, and quantum generative models. In the studied model, and in many others, the circuit used is known as a variational quantum circuit (VQC).

2.1 Variational Quantum Circuit

As already mentioned, VQCs are parameterized quantum circuits where the parameters are optimized to solve certain problems, or most commonly to minimize an objective function. A generic quantum circuit is nothing more than a collection of gates chosen to solve or compute certain problems. VQCs follow a specific circuit structure, i.e. data embedding through rotations, parameterized entanglement layer(s) - essentially controlled gates (e.g. cx) and parameterized rotations -, and a measurement layer on the end.

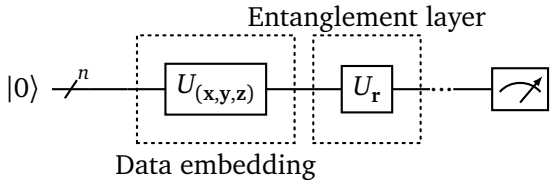

Figure 1: Generic VQC for n qubits. Data embedding layer ($U_{(\mathbf{x},\mathbf{y},\mathbf{z})}$), followed by parameterized entanglement layer(s) and, finally, a measurement layer.

3 Reinforcement Learning

RL is a sub-field of ML where an agent interacts with an environment by taking actions and receiving feedback in the form of rewards or penalties. The agent's goal is to learn a policy, which is the mapping from states to actions that maximize the expected cumulative reward over time.

To effectively implement an RL model, certain choices are essential and inevitable. Specifically, the learning model, typically either Markov Decision Process or Q learning, and the parameter update method, such as policy-gradient methods, play crucial roles. This work primarily focuses on the Q Learning model and policy-gradient methods, delving into their significance and implications.

3.1 Q-Learning

Q-learning is a RL algorithm that doesn't require a predefined model of the environment, making it a model-free approach. It is also an off-policy algorithm, i.e. the agent learns by observing and interacting with the environment, but the policy it learns from - the target policy - can be different from the policy it follows during exploration - the behavior policy. In this algorithm, Q-values are updated based on the maximum expected future reward.

To understand this algorithm and how the model of the environment is learned, it is important to have some concepts in mind. Videlicet,

- **State and Action Spaces** - The agent interacts with an environment defined by a set of states and actions and it takes actions based on its current state;
- **Q-values** - The expected cumulative reward for taking an action over a specific state is represented by Q-values. These values are stored in a table, known as Q-table, that maps the state-action pairs into their associated Q-values.

  To update Q-values, the agent selects an action based on the current state, receives the reward and next state from the environment, and updates the Q-values using the rule:

$$Q(s,a) = Q(s,a) + \alpha * [R + \gamma * max(Q(s',a')) - Q(s,a)]$$

where $Q(s,a)$ is the Q-value for the state s and action a, α is a learning rate that determines the weight given to new information, R is the reward obtained after taking action a over state s and γ is a factor that weights the importance of immediate and future rewards.

- **Exploration and exploitation** - The exploration phase is when the agent explores actions and states to gather information about the environment. On the other hand, in the exploitation phase, it utilizes the learned information to select actions with higher Q-values. Balancing these phases, in this study with an epsilon greedy strategy, optimizes learning and decision-making.

- **Episodes** - Interactions with the environment occur across multiple episodes. Each episode starts with an initial state, the agent takes actions that update the Q-values until it reaches a terminal state or a specified step limit. This allows the agent to learn different states and actions. The learning process continues until the Q-values converge, indicating that the agent's policy stabilized and further exploration won't significantly enhance the policy. Once convergence is achieved, the agent can extract the learned policy by selecting the action with the highest Q-value for each state.

3.2 Policy Gradient

Policy gradient methods are algorithms that directly learn a policy without explicitly estimating value functions or Q-values. Instead of updating Q-values based on reward differences, policy gradient methods optimize the policy parameters to maximize the expected cumulative reward.

The policy is a parameterized function that takes the state as input and outputs a probability distribution over actions. Parameters are updated using gradient ascent on the expected cumulative reward. The update rule is

$$\theta \leftarrow \theta + \alpha \nabla J(\theta),$$

here, θ represents the policy parameters, α is the learning rate, and $\nabla J(\theta)$ is the gradient of the expected cumulative reward with respect to the parameters. The gradient is estimated using the policy gradient theorem [9], which involves sampling trajectories from the environment and computing the gradient based on those samples. According to the policy gradient theorem

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_\pi(a|s,\theta)$$

where $\mu(s)$ is the distribution over states, $q_\pi(s,a)$ is the state-action value function, and $\nabla_\pi(a|s,\theta)$ is the gradient of the policy with respect to the parameters for a specific action in a given state.

The gradient determines the direction in which the parameters should be updated to improve the policy's performance. By following the gradient ascent update rule, the policy parameters are adjusted iteratively to maximize the expected cumulative reward. With this in mind, the policy-gradient algorithm presented in [7] will be used throughout this work.

4 Implementation

This section details the implementation of the quantum and classical reinforcement learning models. The quantum model was implemented using TensorFlow Quantum and the classical model using Tensorflow [10]. The quantum circuits are executed in a simulation environment provided by the same Python library.

4.1 Environment

The chosen environment for the RL agent was Cart Pole (More specifically CartPole-v1 [11]). It consists of a pole attached to a cart, which can move along a frictionless track. The pole starts upright and the goal is to prevent it from falling over by controlling the cart.

The observation from the environment that is fed to the agent is a $4D$ vector representing the position and velocity of the cart, and the angle and angular velocity of the pole. The agent should in turn select one of the two possible actions: push the cart right $(+1)$ or left (-1). A summary is provided in Table 1.

A reward of 1 is provided for every timestep that the pole remains upright. The episode ends when the pole tips over some angle limit, the card is outside world edges, or the hard-limit 500-time steps is exceeded.

Action Space	Discrete(2)
Observation Shape	(4,)
Observation High	[4.8 inf 0.42 inf]
Observation Low	[-4.8 -inf -0.42 -inf]

Table 1: Information regarding the shape and numerical limits of the action and observation vectors.

4.2 Quantum Circuit Architecture

The QML models are implemented using VQCs, subsection 2.1. The VQC pipeline is the following:

1. **State Preparation**: The initial quantum state is prepared using a parameterized quantum circuit (PQC). This circuit is composed by rotation gates $R_X(w_1)$, $R_Y(w_2)$, and $R_Z(w_2)$, where w_1, w_2 and w_3 are the learnable parameters.

2. **Entanglement**: After the initial state preparation, entangled quantum states are created using controlled gates.

3. **Data Embedding**: The classical information (numerical vector X) is mapped into the quantum space, resulting in the state $|\psi_X\rangle$, which corresponds to an encoded version of the classical data within the quantum system.

4. **Quantum Circuit**: The same PQC architecture used for state preparation, but with different parameters, is applied to the encoded state $|\psi_X\rangle$, resulting in the final state $|\psi'_X\rangle = U(w)|\psi_X\rangle$.

5. **Final Prediction**: In the final, all qubits are measured. This measurement guides the decision-making process of the RL agent, indicating the recommended action for maximum reward.

4.2.1 Data Embedding

As far as data embedding is concerned, in this study, Angle Embedding was chosen. More precisely, for an N-dimensional vector of classical information, $X = (x_1, x_2, \ldots, x_N)$, the state $|\psi_X\rangle$ is constructed by employing rotations around the x-axis on the Bloch sphere. Mathematically, the quantum state embedded resulting from the embedding of classical information is

$$|\psi_X\rangle = \bigotimes_{i=1}^{N} R_X(x_i) = \bigotimes_{i=1}^{N} \left[\cos\left(\frac{x_i}{2}\right)|0\rangle - i \sin\left(\frac{x_i}{2}\right)|1\rangle \right], \tag{1}$$

where $R_X(x) = e^{-i\frac{x}{2}\hat{\sigma}_x}$ and $\hat{\sigma}_x$ represents a Pauli operator.

An additional normalization technique is applied, involving the scaling of the observation vector using the hyperbolic tangent function. This scaling procedure effectively maps the ranges specified in Table 1 to values within the range of $[-1, 1]$. After normalization, the input values are further scaled by learnable parameters denoted as input scaling. This mapping greatly enhances the stability and manageability of the training dynamics. Moreover, if enabled, data re-uploading will be conducted, which involves performing the embedding step in each layer of the circuit.

4.2.2 Entanglement

In this study, four different entanglement layers are investigated. These are:

- **Controlled-X with circular format** - The controlled gates used are CX (CNOT) and they are circularly applied (Figure 2).
- **Controlled-Z with circular format** - The controlled gate used are CZ and they are circularly applied.
- **Controlled-X with ladder format** - The controlled gate used are CX (CNOT) and they are applied in a ladder format (Figure 3).
- **Controlled-Z with ladder format** - The controlled gate used are CZ and they are applied in a ladder format.

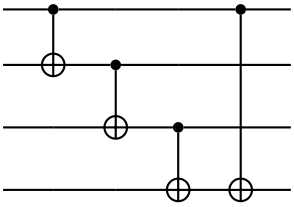

Figure 2: Circular format entanglement layer for 4 qubits.

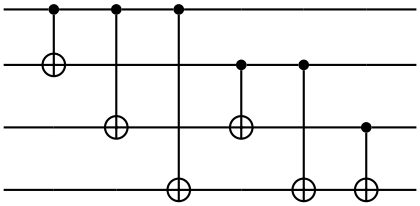

Figure 3: Ladder format entanglement layer for 4 qubits.

4.2.3 Parametrized Quantum Circuit

The PQC plays a vital role in the VQC pipeline housing the learnable parameters and serving as a critical component in both the initial state preparation and the quantum circuit section. It consists of a rotation gates collection (R_x, R_y and R_z) applied to all qubits in the system. The rotation angles are the learnable parameters, and so each rotation R has 3 learnable parameters ϕ, θ, ω, i.e.

$$R(\phi,\theta,\omega) = RZ(\omega)RY(\theta)RZ(\phi) = \begin{bmatrix} e^{-i(\phi+\omega)/2}\cos(\theta/2) & -e^{i(\phi-\omega)/2}\sin(\theta/2) \\ e^{-i(\phi-\omega)/2}\sin(\theta/2) & e^{i(\phi+\omega)/2}\cos(\theta/2) \end{bmatrix} \tag{2}$$

Since all the learnable parameters of the VQC are contained inside the rotation gates, and each gate has 3 parameters, the shape of the weight vector is $w \in \mathbb{R}^{n\times l\times 3}$, where n is the number of qubits of the current system and l is the number of layers in the network.

4.2.4 Final prediction

The model outputs two values, one for each possible action. These values are obtained by measuring the expectation value of $\hat{\sigma}_z$ operator of all of the qubits and then multiplying the value of the first two qubits for one of the actions and the value of the other two for the other action. These values are then scaled by learnable parameters in a process denoted as output scaling.

4.3 Deep Neural Network (DNN)

A DNN emulates the brain's functioning using interconnected neurons, connected through weights and biases. These parameters are essential for processing input data and producing the desired output.

Each neuron multiplies an initial value by a weight, w, sums the results with inputs from other neurons, adds the neuron's bias, b, and applies an activation function, in this case the relu function, for non-linearity (Equation 3).

$$y = f\left(\sum_{i=0}^{N}(\boldsymbol{w_i}\cdot\boldsymbol{x_i}) + b\right) \tag{3}$$

The output vector $\boldsymbol{X^{(n)}}$ of the n-th layer in a DNN is described mathematically as:

$$X^{(n)} = f(W^{(n)}X^{(n-1)} + B^{(n)}). \tag{4}$$

This process repeats for each layer until the last layer (output layer), which provides predictions for the classification task.

Analogous to the quantum algorithm, the DNN architecture employs four input neurons, a variable hidden layer size (each with ten neurons), and two outputs (one for each agent action).

4.4 Training details

During the training phase of the quantum algorithm, a grid search was conducted. In addition to the Q-learning and policy gradient methods, an epsilon greedy strategy was employed.

The epsilon-greedy strategy is a method that effectively balances the exploration and exploitation phases. In simple terms, this approach allows an agent to make decisions by randomly selecting an action for exploration, with a probability of ϵ. Conversely, with a probability of (1-ϵ), the agent chooses the action with the highest estimated reward for exploitation. Over time, the value of ϵ gradually decreases, enabling a shift in focus from exploration towards exploitation.

Two different architectures were used to tackle the Card Pole environment: The VQC and the DNN. The corresponding training was done for the set of HP summarized in Table 3 and Table 2. For each set of HP, 10 architectures are initialized with different seeds for their learnable parameters. For the VQC the parameters are initialized with a normal distribution[1] and for the DNN the default initialization is used.

Variable HP	Values
Data Re-Uploading	{0, 1}
Entanglement Type	{CX, CZ}
Entanglement Format	{Ladder, Circular}
Number of Layers	[1, 8]

Table 2: List of scanned hyperparameters. Note that the Data Re-Uploading, the Entanglement Type and the Entanglement Format are exclusive to the VQC architecture.

Fixed HP	Values
Batch Size	16
Learning Rate (LR)	0.001
Learning Rate (IO Scaling)	0.01
ϵ_0	1
ϵ_{decay}	0.99
ϵ_{min}	0.01
Buffer Size	0.01
Target Update Frequency (Steps)	5
Online Train Frequency (Steps)	1
Win Threshold (Episodes)	100

Table 3: List of fixed hyperparameters.

The VQC use, as a training strategy, the framework described so far. However, initial tests using the same regime for training a DNN showed a lack of convergence and failure of the architecture. Consequently, a more conventional training regime based on Deep Q-Learning was employed instead. For optimization, the VQC utilizes the Huber loss function and the mean squared error for the DNN.

4.5 Testing details

After completing the training phase, the models proceed to the testing phase where their performance is evaluated through benchmarking. In this phase, 100 game simulations were conducted using the trained models and meticulously recorded the rewards achieved. Subsequently, we carefully analyzed these results to effectively compare and evaluate their performance during this testing phase.

4.6 Code Availability

The code used in this work is publicly available in https://github.com/mcpeixoto/ReinforcementQML

[1]Via the "tf.random_normal_initializer" from the Tensorflow [10] package.

5 Results

Quantum models are benchmarked against DNNs, both in the training and testing phases. The presented quantum models vary in terms of the entanglement layers employed (subsubsection 4.2.2) and the utilization of re-uploading.

To evaluate the training performance, it's utilized the Area Under the Curve (AUC) of the rewards over the episodes, accompanied by its corresponding standard deviation. This metric is computed for each run, considering the number of layers, by dividing the sum of rewards obtained by the product of the number of episodes and the maximum achieved reward. To ensure statistical significance, there are obtained 10 different values for each run by varying the architecture initialization using a seed, and the standard deviation is calculated accordingly. The figure depicting this metric for the classical and quantum models under investigation is shown in Figure 4.

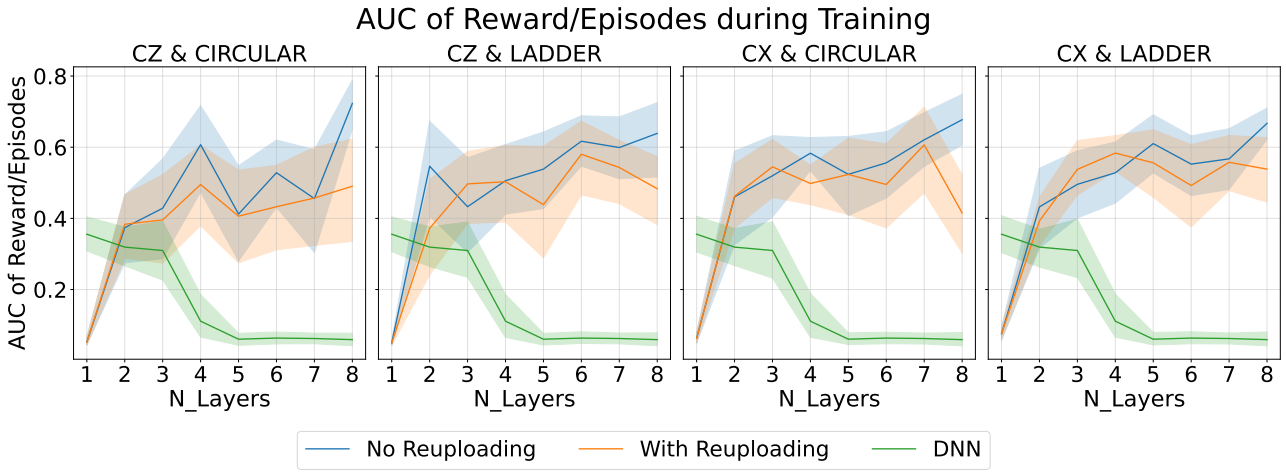

Figure 4: Plot of the AUC metric. Quantum models (VQCs, denoted with 'No re-uploading' and 'With re-uploading' labels) are benchmarked against the classical model (DNN, denoted with the 'DNN' label). For each data point, 10 different seeds are used.

Upon examining the figure above, it becomes apparent that the training performance of quantum models improves as the number of layers increases. Interestingly, the majority of the analyzed quantum models demonstrate comparable performances across most layers, except for the case of 8 layers. In this scenario, not using re-uploading proves to be superior to using re-uploading when employing a circular configuration of entanglement. Conversely, the classical model's performance appears to decrease as the number of layers increases. To put it simply, the classical model outperforms the quantum models when only one layer is utilized. As the number of layers increases to two, their performances become comparable. However, beyond that point, the quantum models exhibit superior behavior.

Refocusing our attention on a specific metric (Figure 5), a noteworthy pattern can be observed. Classical models consistently achieve performance that aligns with the maximum attainable reward for up to four layers. However, the situation differs for quantum models, as this trend holds only when employing more than four layers - when using a smaller number of layers, most of the quantum model fails to secure victory in the game.

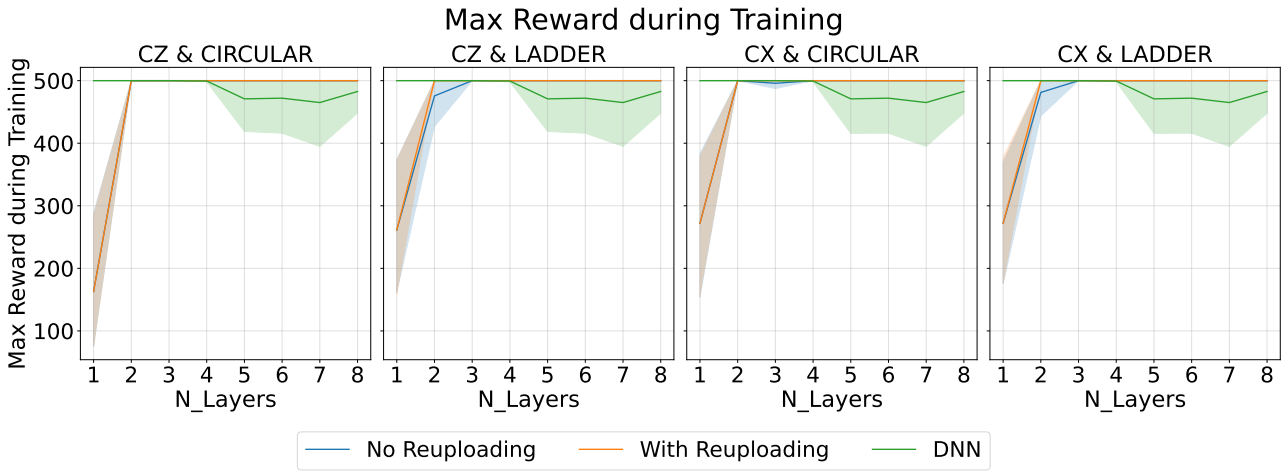

Figure 5: Plot of the maximum reward achieved during the training. Quantum models (VQCs, denoted with 'No re-uploading' and 'With re-uploading' labels) are benchmarked against the classical model (DNN, denoted with the 'DNN' label), 10 different seeds are used.

The performance of the testing phase is evaluated based on the percentage of winning games out of a universe of 100 games, once the model training is completed and the best model is determined. To ensure reliability, testing was done using 10 different seeds for each run. Additionally, the standard deviation was computed to account for any potential variations. The corresponding plots depicting the performance of the trained models are presented in Figure 6.

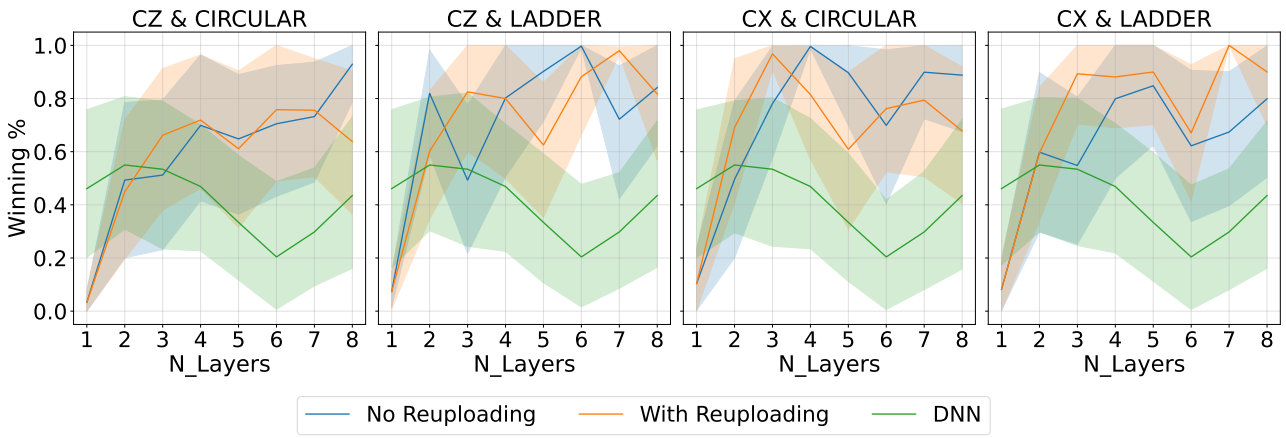

Figure 6: Plot of the percentage of winning games during the testing phase. Quantum models are benchmarked against the classical one. For each point, 10 different seeds are used.

Upon analyzing the figure, there appears to be no noticeable correlation between the number of layers and the percentage of winning games for the quantum models. However, it is worth mentioning that most of the quantum models demonstrate similar levels of compatibility in terms of game wins. Interestingly, the CZ & ladder and CX & circular layer types show slightly better performance in this regard. Comparatively, the classical model outperforms the quantum model with a single layer but then aligns with the quantum model's performance up to six layers. Notably, the classical models exhibit a performance dip at this point, followed by an increase until reaching the maximum number of layers under study.

6 Conclusion

With the aim of doing a systematic analysis of solving the cart pole game using quantum reinforcement learning, a comprehensive grid search study was conducted. Furthermore, as a classical comparison, a deep neural network was also implemented. The systematic study involved benchmarking against the classical model and examining different quantum models, exploring different types and numbers of entanglement

layers, as well as the use of data re-uploading. Performance comparisons were made during both the training and testing stages.

Overall, the results of comparing the quantum and classical models revealed that the quantum models consistently exhibited more favorable behavior than their classical counterparts. Furthermore, when considering the peak performance achieved, the quantum model outperformed the classical model both in training and testing. It is worth noting, however, that these quantum results were obtained in a simulated environment, and therefore, the performance of these models may not be as optimal on current devices.

Nevertheless, it is important to acknowledge that in order to have a stronger classical model and make a fairer comparison, an in-depth study of the classical model would be necessary, which would require significantly more time than was available for this particular research endeavor. Hence, it is neither possible nor fair to definitively assert that the quantum model is inherently superior to the classical model. While the performance of the quantum models can be compared with each other with a certain degree of confidence, for a fair comparison with classical models, the latter's study would necessitate a level of depth similar to that of the quantum one.

To summarize, based on the models examined in this study, quantum RL shows promise in solving the cart pole environment, and in the training and testing regimes investigated, quantum RL appears to be more advantageous. However, it is important to acknowledge the limitations of this conclusion, as it cannot be guaranteed that there isn't a superior classical model yet to be discovered.

References

[1] J. Preskill, *Quantum computing in the nisq era and beyond*, Quantum **2**, 79 (2018).

[2] M. C. Peixoto, N. F. Castro, M. C. Romão, M. G. J. Oliveira and I. Ochoa, *Fitting a collider in a quantum computer: Tackling the challenges of quantum machine learning for big datasets*, arXiv preprint arXiv:2211.03233 (2022).

[3] V. Chamola, A. Jolfaei, V. Chanana, P. Parashari and V. Hassija, *Information security in the post quantum era for 5g and beyond networks: Threats to existing cryptography, and post-quantum cryptography*, Computer Communications **176**, 99 (2021).

[4] M. Zinner, F. Dahlhausen, P. Boehme, J. Ehlers, L. Bieske and L. Fehring, *Quantum computing's potential for drug discovery: Early stage industry dynamics*, Drug Discovery Today **26**(7), 1680 (2021).

[5] A. Pramod, H. S. Naicker and A. K. Tyagi, *Machine learning and deep learning: Open issues and future research directions for the next 10 years*, Computational analysis and deep learning for medical care: Principles, methods, and applications pp. 463–490 (2021).

[6] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*, Packt Publishing Ltd (2018).

[7] S. Jerbi, C. Gyurik, S. Marshall, H. Briegel and V. Dunjko, *Parametrized quantum policies for reinforcement learning*, Advances in Neural Information Processing Systems **34**, 28362 (2021).

[8] P. W. Shor, *Fault-tolerant quantum computation* (1997), quant-ph/9605011.

[9] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, MIT Press, Cambridge, MA, USA, 2st edn. (1998).

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org (2015).

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, *Openai gym*, arXiv preprint arXiv:1606.01540 (2016).