

# Exploring the Applications of Deep Reinforcement Learning And Quantum Variational Circuit In Quantum Machine Learning

by

Prashanta Kumar Saha  
17301103

Vishal Saha  
19101671

A thesis submitted to the Department of Computer Science and Engineering  
in partial fulfillment of the requirements for the degree of  
B.Sc. in Computer Science

Department of Computer Science and Engineering  
BRAC University  
September 2021

© 2021. Brac University  
All rights reserved.

# Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

## Student's Full Name & Signature:

*Prashanta*  
02.10.21

---

Prashanta Kumar Saha  
17301103

*Vishal Saha*  
02.10.21

---

Vishal Saha  
19101671

# Approval

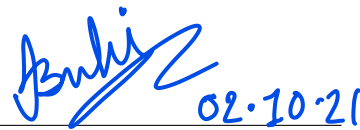
The thesis/project titled “Exploring the Applications of Deep Reinforcement Learning And Quantum Variational Circuit In Quantum Machine Learning” submitted by

1. Prashanta Kumar Saha (17301103)
2. Vishal Saha (19101671)

Of Summer, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on October 2, 2021.

## Examining Committee:

Supervisor:  
(Member)

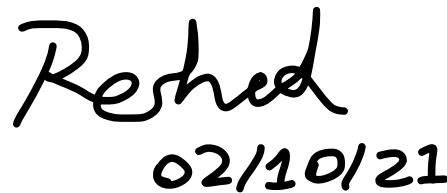


---

Ipshita Bonhi Upoma  
Lecturer

Department of Computer Science and Engineering  
Brac University

Co-Supervisor:  
(Member)



---

Mr. Md Reshad Ur Rahman  
Lecturer

Department of Mathematics and Natural Sciences  
Brac University

Program Coordinator:  
(Member)

---

Md. Golam Rabiul Alam, PhD  
Associate Professor

Department of Computer Science and Engineering  
Brac University

Head of Department:  
(Chair)

---

Sadia Hamid Kazi, PhD  
Chairperson and Associate Professor  
Department of Computer Science and Engineering  
Brac University

# Abstract

In recent years, quantum computing has outperformed classical computing in many aspects, including the advancement of approaches in Reinforcement Learning problems. Particularly, it has the power to utilize the quantum phenomena of superposition and entanglement, that can fastened the calculation of a vast amount of data which is very challenging for classical computers. Unfortunately, the current Quantum Computing platforms are very complex to initiate classical reinforcement learning problems for uncontrollability and intricacy of quantum circuits. In our work, we explore the application of Quantum Variational Circuit (QVC) in Deep Q-Network (DQN) instead of classical Reinforcement Learning approaches to enhance the performance of Reinforcement Learning. To achieve that, we use Quantum Variational Circuit (QVC) based reinforcement learning approaches to solve the classical problems and we also solve the classical problems using classical DQN and Double Deep Q-Network (DDQN) Reinforcement Learning to compare between classical and quantum approaches. We solve Atari and Lunar Lander in OpenAI Gym environments using QVC based DQN Reinforcement learning. We study encoding techniques such as amplitude encoding, scaled encoding and directional encoding which were previously used in this paper[1]. We exercise IBM's open-source SDK (QISKit) and IBM-Q for quantum circuit implementation which can produce improved applications like Quantum error Correction codes etc. We also use TensorFlow Quantum to implement the hybrid classical-quantum computation and experimentally analyze our work.

**Keywords:** Quantum Computing, Reinforcement Learning, Quantum Machine Learning (QML), Quantum Variational Circuit (QVC), Deep Q-Network (DQN), Double Deep Q-Network (DDQN), OpenAI Gym, IBM-Q, TensorFlow Quantum.

## Acknowledgement

We are grateful to our advisors, Ms. Ipshita Bonhi Upoma of BRAC University's Department of Computer Science and Engineering and Mr. Md. Reshad Ur Rahman of BRAC University's Department of Mathematics and Natural Sciences. They were extremely helpful whenever we needed it. This research would not have been possible without their knowledge, guidance, and dedication. We would also like to thank Mr. Wasif Ahmed and Mr. Sowmitra Das of BRAC University's Department of Computer Science and Engineering. They taught us the fundamentals of quantum computing and inspired us. Their advice was critical to our research.

# Table of Contents

Declaration	i
Approval	ii
Abstract	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
List of Tables	ix
Nomenclature	x
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	1
1.2 Research Objectives . . . . .	1
<b>2 Reinforcement Learning</b>	<b>3</b>
2.1 Reinforcement Learning . . . . .	3
2.2 Q-Learning . . . . .	3
2.3 Deep Q-Learning . . . . .	4
2.4 Double Q-Learning . . . . .	4
<b>3 Quantum Computing</b>	<b>6</b>
3.1 Quantum Computing . . . . .	6
3.2 Quantum Gates . . . . .	9
<b>4 Quantum Variational Circuit</b>	<b>11</b>
<b>5 Function and Preparation of QVC(or PQC)</b>	<b>12</b>
<b>6 Quantum Data Encoding</b>	<b>15</b>
<b>7 Quantum Reinforcement Learning</b>	<b>16</b>
7.1 Introduction . . . . .	16
7.2 Representation . . . . .	16
7.3 Action selection policy . . . . .	17
7.4 Paralleling state value updating . . . . .	18

7.5	Probability amplitude updating . . . . .	18
<b>8</b>	<b>Experiments &amp; Results</b>	<b>20</b>
8.1	Experiments . . . . .	20
8.2	Results . . . . .	23
<b>9</b>	<b>Conclusion</b>	<b>29</b>
	<b>Bibliography</b>	<b>31</b>



# List of Figures

2.1	DQN Architectue . . . . .	4
3.1	Controlled-NOT gate . . . . .	9
4.1	Function of QVC . . . . .	11
5.1	An example of Code for classical post-processing calculation . . . . .	13
5.2	A typical structure of Layered and Alternating Ansatz . . . . .	14
5.3	A typical architecture of Tensor network Ansatz . . . . .	14
7.1	Grover algorithm in QRL . . . . .	19
8.1	OpenAI Gym Environment . . . . .	21
8.2	Equation . . . . .	21
8.3	QVC based on 5 qubits . . . . .	21
8.4	QVC based on 3 qubits . . . . .	22
8.5	Figure representing 1000 episodes . . . . .	22
8.6	QVC based on 4 qubits . . . . .	22
8.7	Atari Environment form OpenAI Gym . . . . .	23
8.8	Atari game's module . . . . .	23
8.9	Result of DQN algorithm in classical RL. . . . .	23
8.10	Result of DQN algorithm in PQC RL. . . . .	24
8.11	Graph of DQN Algorithm using classical RL . . . . .	24
8.12	Graph of DQN Algorithm with PQC RL . . . . .	24
8.13	Graph of DQN algorithm in QVC RL [1] . . . . .	26
8.14	Graph of DQN Algorithm with 3 qubits QVC RL . . . . .	26
8.15	Result of DQN Algorithm with 3 qubits QVC RL . . . . .	26
8.16	Graph of DQN Algorithm with 4 Qubits QVC RL . . . . .	27
8.17	Result of DQN Algorithm with 4 qubits QVC RL . . . . .	27
8.18	Result of DQN Algorithm with 4 qubits QVC RL . . . . .	28
8.19	Result of DQN Algorithm with 4 qubits QVC RL . . . . .	28

# List of Tables

8.1	Comparison between Classical RL and QVC base RL with DQN Algorithm . . . . .	25
8.2	Comparison among Owen Lockwood's QVC based RL for CartPole [1], 3 Qubit based QVC on CartPole using the DQN Algorithm and 4 Qubit based QVC on CartPole using the DQN Algorithm . . . . .	27
8.3	Comparison between the Classical DQN based Atari and QVC based Atari using DQN . . . . .	28

# Nomenclature

The next list describes several symbols and abbreviation that will be later used within the body of the document

<i>DDQN</i>	Double Deep Q-Network
<i>DQN</i>	Deep Q-Network
<i>DRL</i>	Deep Reinforcement Learning
<i>MDP</i>	Markov Decision-making Process
<i>PQC</i>	Parameterized Quantum Circuit
<i>QML</i>	Quantum Machine Learning
<i>QVC</i>	Quantum Variational Circuit
<i>RL</i>	Reinforcement Learning

# Chapter 1

## Introduction

### 1.1 Problem Statement

Deep Reinforcement Learning (DRL) has been advancing at an incredible rate for some years. We observe many Supervised and Unsupervised approaches that have been used over the years to solve problems like Atari, Lunar Lander but nowadays Deep Reinforcement Learning approaches are doing remarkably better to solve similar sorts of problems. Further, DRL is boosting lots of games to improve the games agents and minimize the time needed for training. Though there are some limitations in performance using classical computers in RL as their calculation is based on binary bits. Whereas Quantum Computers use ‘Qubits’ which will generate more possibilities and outcomes that create better performance than classical computers. The problem is that the quantum system becomes extremely crucial and unreliable when dealing with qubits because of noises, quantum decoherence faults, and fails before any program is completed. In this work, we aspire to implement Quantum Variational Circuits (QVC) based reinforcement learning. In this work, we explore a hybrid approach by implementing QVC based reinforcement learning to solve Atari, Lunar Lander and QVC based cartpole using TensorFlow Quantum. QVC is a combination of qubits and circuits on which we execute rotations and other quantum operations before measuring our system or model till it is optimized. Previously experiments were carried out to determine if QVC-based DQN outperforms conventional DQN techniques, and researchers discovered that QVC-based DQN takes  $O(N)$  space to solve a 4x4 Frozen Lake problem from the OpenAI Gym environment, whereas classical DQN approaches use  $O(N^2)$  space [1]. As a result of our proposed implementation, we hope to achieve a greater precise error correction with the assistance of Quantum variational circuits, which may be a significant advance in reinforcement learning.

### 1.2 Research Objectives

Quantum computing has been able to demonstrate excellent efficiency in numerical precision in recent years than any classical computers[2]. In addition, Quantum computers have a wide variety of significant applications in the future[3]. Because of their potential to conduct even more complicated calculations at much faster speeds, including solving problems that are literally beyond the capabilities of today’s com-

puters[4]. Working with quantum computing, however, is not only an easy task, but also with appropriate algorithms, we need to regulate the qubit rotation and the gates. This is why we want to explore Atari, Lunar Lander based on QVC in this paper to demonstrate that we can achieve better results in machine learning problems by using QVC in RL.

QVC is a combination of qubits and circuits on which we execute rotations, we improve the value of  $\theta$ , and other quantum operations before measuring our system or model till it is optimized. The parameter shift differentiator is used in this work. Though TensorFlow-Quantum implements the process, having an appropriate mathematical overview is important. Mainly, we are focused on comparing the performance of classical DQN and classical Double DQN with QVC DQN and QVC Double DQN. Moreover, we want to explore different encoding, decoding schemes which can be used in QVC based deep reinforcement learning.

We will apply these algorithms on OpenAI Gym Atari and Lunar Lander environments and produce the result. After applying the procedures on the games we will find the efficient strength of quantum computing in promoting reinforcement learning.

# Chapter 2

## Reinforcement Learning

### 2.1 Reinforcement Learning

Reinforcement Learning (RL) is a subset of Machine Learning (ML), although it is distinct from supervised and unsupervised learning. In RL, an agent learns about its environment and tries to attain a certain objective. An agent is someone that interacts with the environment and makes decisions based on the probability distribution of all potential states. Some environmental restrictions are referred to as policies. If our agent is in state  $S_t$ , it will pick its actions, which means it will choose its next state  $S_{t+1}$  based on the reward it receives for being in state  $S_t$ . The Markov Decision-making Process (MDP) is used by RL, which includes five components:  $\{S, A, P, r, V\}$  [5] [2]

$$\begin{aligned} V_{(s)}^n &= E\{r_{t+1} + \gamma r_{t+1} + \dots \mid s_t = s, \pi\} \\ &= E[r_{t+1} + \gamma V_{(s)t+1}^n \mid s_t = s, \pi] \\ &= \sum_{a \in A_s} \pi(s, a) [r_s^a + \gamma \sum_{s'} p_{ss'}^a V_{s'}^\pi] \end{aligned} \tag{2.1}$$

$$V_{(s)}^* = \max_{a \in A_s} [r_s^a + \gamma \sum_{s'} p_{ss'}^a V_{s'}^*] \tag{2.2}$$

The agent is continually intending to optimize the goal function and selecting states depending on this objective function.

#### Action-Value Function, $Q(s, a)$ :

Here we can apply different kinds of processes of approximating the action-value function  $Q$ . We have to use values as a state representative to the DQN architecture as an input. Also, we have to measure the output layer as a separate output for every varied action. Here is given DQN architecture:

### 2.2 Q-Learning

Q-learning does not necessitate the use of a reinforcement learning algorithm. [5]. Before we begin the process of learning, we set a random number to  $Q$ . Q-learning selects its next state in order to update  $Q$ . [6]:

$$Q(s_t, a_t) \leftarrow a(s_t, a_t) + a[r_t + \gamma \max_a Q(S_{t+1}, a) - Q(s_t, a_t)] \tag{2.3}$$

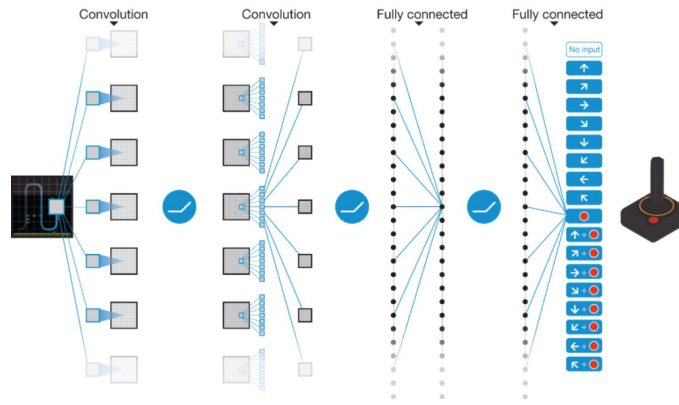


Figure 2.1: DQN Architecture

## 2.3 Deep Q-Learning

A 2-D list could be used to represent the action-value function  $Q(s, a)$ . [5] [7]. The concept of 2-D interpretation does not perform efficiently, when either state or action is enormous. Within this circumstance, we employ features such as neural networks and this is known as deep reinforcement learning (DRL). DRL has now been extensively employed throughout several sectors of problem solving. The action-value function is  $Q(s, a; \theta)$ .  $\theta$  is a parameter that we can optimize via an ongoing problem-solving approach. The Q-learning approach seems to be the simplest. The goal of this technique is to optimize the action-value function  $Q$  by reducing the loss function: [8]

$$L(\theta) = E[(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta))^2] \quad (2.4)$$

Here,  $Q(s_t, a_t)$  is prediction;  $\theta$ ,  $\theta$  expresses the policy network parameter, and  $r_t$  is target  $+ \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$  where  $\theta^-$  is the target network parameter and  $s_{t+1}$  is the state encountered at state  $s_t$  after playing action  $a_t$ . In DRL, the loss function hardly converges. Although it diverges when we use a nonlinear approximator. [8].

## 2.4 Double Q-Learning

In traditional Q-learning and DQN, the max operator selects and evaluates actions using the same values. This makes overestimated values more likely to be selected, resulting in overoptimistic estimates of value. We can decouple the selection from the assessment to avoid this. Two value functions are learned in Double Q-learning by randomly assigning experience to update weight sets,  $\theta$  and  $\theta'$  [9]. One set is used for each update to identify the greedy policy and the other to determine its value. In Q-learning, we can untangle the selection and evaluation and rewrite the target for clear comparison as

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_a Q(S_{t+1}, a; \theta_y); \theta_t) \quad (2.5)$$

The error in Double Q-learning:

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_a Q(S_{t+1}, a; \theta_y); \theta_{t'}) \quad (2.6)$$

Note that the action selection in argmax is still due to  $\theta_t$  online weights. This implies that, as in Q-learning, according to current values, we are estimating the cost of the greedy policy, as defined by  $\theta_t$ . However, we use the 2nd weights  $\theta_{t'}$  to assess the value fairly. Users can symmetrically update this second set of weights by changing the roles of  $\theta$  and  $\theta'$  [9].



# Chapter 3

## Quantum Computing

### 3.1 Quantum Computing

Quantum mechanics is a set of principles or mathematical foundation for constructing physical theories. In the 1970s, the idea of integrating quantum physics and information theory arose. Quantum information and computing present a valuable set of problems with varying degrees of complexity. People design strategies to better manage single quantum systems, encourage the development of novel experimental procedures, and give recommendations as to the most intriguing paths in which to conduct experiments as a result of these productive challenges. We know about bits. Bits are the fundamental units of information which are used by classical computers.[10] In classical computers, we use two possible states either 1 or 0. However, a classical computer using classical algorithms struggles to tackle a crucial issue or factor a huge number, among other things. Also, it takes an excessive amount of time and memory. The building elements of quantum information are quantum bits, often known as qubits. They are denoted by  $|0\rangle$  and  $|1\rangle$ . The states of the bits can be superimposed on a qubit. Superposition means bits can coincide. Quantum Computing uses the concept of superposition and entanglement to perform a computation. Now we will explain Superposition, Entanglement, measurement and its application in brief.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (3.1)$$

Similarly, we represent 1 as -

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (3.2)$$

As seen here, a qubit depicts a 2-D quantum system.

$$\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \quad (3.3)$$

A classical bit is either 1 or 0, while a qubit could be a pair of  $|0\rangle$  or  $|1\rangle$ , which is referred to as a superposition:[11]

$$|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle \quad (3.4)$$

$\|c_0\|^2 + \|c_1\|^2 = 1$ ; Where  $\|c_0\|^2$  is the likelihood of finding in state  $|0\rangle$ . Again,  $\|c_1\|^2$  the likelihood of finding in state  $|1\rangle$ . Quantum devices must cope with many

qubits. Check out the following eight bits: 01101011.

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (3.5)$$

The following tensor product is said to represent coupled quantum systems:

$$|0\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle \quad (3.6)$$

And it belongs to the following vector space:

$$C^2 \otimes C^2 \otimes C^2 \otimes C^2 \otimes C^2 \otimes C^2 \otimes C^2 \otimes C^2 \quad (3.7)$$

Another way to express 8 qubits:

$$\begin{bmatrix} \mathbf{00000000} \\ \mathbf{00000001} \\ \vdots \\ \mathbf{01101010} \\ \mathbf{01101011} \\ \mathbf{01101100} \\ \vdots \\ \mathbf{11111110} \\ \mathbf{11111111} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} .$$

(3.8)

Eight qubits together is called a qubyte. Thus a state of eight qubits can be written by 256 complex numbers.

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad (3.9)$$

$$\frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix} \quad (3.10)$$

written as  $\frac{1}{\sqrt{3}}|00\rangle - \frac{1}{\sqrt{3}}|10\rangle + \frac{1}{\sqrt{3}}|11\rangle$  Unlike pure-state models, it does not draw its strength from the generation of a large amount of entanglement. Entanglement is

a state that cannot be described or stated in terms of a tensor product of qubits or other states. Vidal, G. and Latorre, J. I. and Rico, E. and Kitaev, A.

One of the most fascinating aspects of quantum theory is state entanglement. If the system is in the state

$$\frac{|11\rangle + |00\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} |11\rangle + \frac{1}{\sqrt{2}} |00\rangle \quad (3.11)$$

then that means the two qubits are entangled. That indicates if we check any one qubit and find it in state  $|1\rangle$ , we know the other qubit must be in  $|0\rangle$ . and vice versa. (M. A. Nielsen, 2011) [11]

## 3.2 Quantum Gates

A quantum gate is basically a qubit-acting operator. Unitary matrices will be used to represent such operators.

### NOT Gate

$$NOT = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (3.12)$$

$$NOT |0\rangle = |1\rangle$$

$$NOT |1\rangle = |0\rangle$$

### Pauli gates

The following three matrices are called Pauli matrices:

$$BitFlip, X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (3.13)$$

The pauli gate is analogous to our traditional NOT gate.

$$PhaseFlip, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad (3.14)$$

$$Z |0\rangle = |0\rangle$$

$$Z |1\rangle = -|1\rangle$$

$$Bit - PhaseFlip, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad (3.15)$$

### Hadamard gate

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (3.16)$$

$$H |0\rangle = |+\rangle$$

$$H |1\rangle = |-\rangle$$

### Controlled-NOT gate

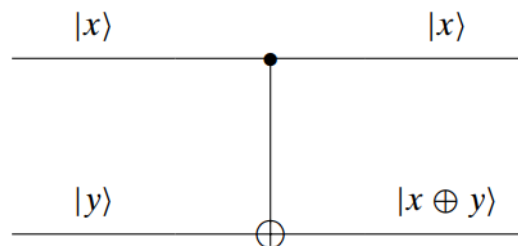


Figure 3.1: Controlled-NOT gate

To function, a controlled-NOT gate requires two qubits. Let  $|\psi_1\rangle$  be a control bit; if  $|\psi_1\rangle$  is  $|0\rangle$ , the associated qubit remains unchanged; if  $|\psi_1\rangle$  is  $|1\rangle$ , the associated qubit changes to its inverse value.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (3.17)$$

$$\begin{aligned} CX |0\rangle \otimes |0\rangle &= |0\rangle \otimes |0\rangle \\ CX |0\rangle \otimes |1\rangle &= |0\rangle \otimes |1\rangle \\ CX |1\rangle \otimes |0\rangle &= |1\rangle \otimes |1\rangle \\ CX |1\rangle \otimes |1\rangle &= |1\rangle \otimes |0\rangle \end{aligned}$$

### Rotational gate

Rotational gates are qubit-operated gates that rotate along the x, y, and z axes. The matrices for a qubit rotating gates  $R_x$ ,  $R_y$ , and  $R_z$  are as follows:

$$\begin{aligned} R_x(\theta) &= \begin{bmatrix} \cos \theta & -i \sin \theta \\ -i \sin \theta & \cos \theta \end{bmatrix} \\ R_y(\theta) &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \\ R_z(\theta) &= \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix} \end{aligned}$$

# Chapter 4

## Quantum Variational Circuit

Quantum variation circuit is also known as parameterised quantum circuit(PQC). Quantum variational circuit, like other quantum circuits, operates on qubits. Quantum variational circuits could be built utilizing unitary and rotational gates, for example. The special feature of quantum variational circuits is that they are parameter dependent. We might simply customize the circuit to address quantum machine learning problems through altering the values of parameters. The application of quantum machine learning for the use of quantum variational circuits is growing enormously. To create quantum variational circuits, we must first take qubits and set them to zero. Then, depending on the problem for which we are designing our quantum variational circuits, we add various sorts of gates such as rotation gates, controlled-NOT gates, and so on. It is important to highlight that we are addressing conventional machine learning issues with quantum variational circuits that employ quantum computing. Given that quantum computers are still in the early stages of development, one would wonder how we might do computations utilizing quantum variational circuits. The explanation is that we employ a platform that is neither pure classical nor pure quantum, but rather a classical-quantum platform. As a result, the quantum variational circuits are created utilizing the laws of quantum computations, which outperform conventional computations. However, the quantum variational circuits are measured using classical components, and the results are used to update the quantum variational circuits parameters. In this case, measurement may be seen as a function in which qubits are mapped to classical bits. The classical component that aids in the updating of the parameters of quantum variational circuits following measurements is commonly referred to as the optimizer.[12][3][13][4][14][15]

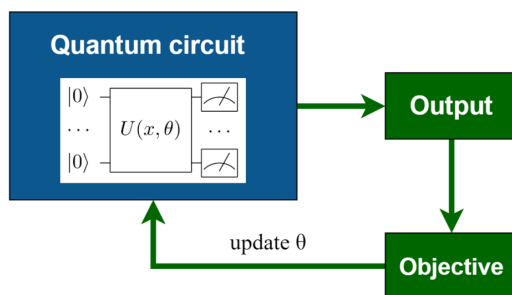


Figure 4.1: Function of QVC

# Chapter 5

## Function and Preparation of QVC(or PQC)

Quantum Computing's supremacy and brilliance in several disciplines, such as Quantum Machine Learning, Optimization, and Cryptography, has led to the solution of a variety of issues that classical computers were unable to solve. QVC is one of these algorithms for tackling quantum machine-learning challenges. In Quantum Computing, they serve the same purpose that Neural Networks do in Classical Computing. A quantum variational circuit(QVC) is usually divided into three parts, **Preparation of initial state**: In this phase, we set zero to all of the involved qubit states.

**Parameterized Quantum circuit**: As illustrated in figure 7, it will be generated by  $U(x, \theta)$ , which consists of input parameters( $x$ ) and variational parameters( $\theta$ ). We will use a conventional optimization technique that makes queries to the quantum device to train variational circuits. Quantum Encodings are performed using the data input to these circuits. In the Hilbert feature space, Quantum Encoding expresses classical bits as qubits. In Hilbert space, there are several encoding techniques ranging from conventional to quantum data. Here, we will discuss two approaches.[16][17][18]

The most basic approach of converting data to quantum data is **basis encoding**. This approach connects the computational base of n-qubit input with n-bit classical input. In general, we utilize the following equation to encode data sets using the basis approach.

$$|D\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^M X^m \quad (5.1)$$

where  $D$  equals  $\{X^1, X^2, \dots, X^M\}$  is classical data in binary,  $X^m$  equals  $\{b_1, b_2, \dots, b_N\}$ ,  $b_i \in \{0, 1\}$ ,  $i \in \{0, 1, 2, \dots, N\}$  where  $N$  represents features.

In Quantum Machine Learning, **Amplitude encoding** is a common and widely utilized way of encoding. The coupling of classical data with quantum state amplitudes is the foundation of amplitude encoding. We must transform classical number vec-

tors to normalized classical vectors in order to encode quantum amplitudes.[19]b27

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_{2^n} \end{bmatrix} \quad (5.2)$$

Here X is normalized,  $x \in C^{2^n}$ , and C stands for complex number. State amplitudes is being represented in the following ways::

$$|\psi_x\rangle = \sum_{i=1}^{2^n-1} x_i |i\rangle \quad (5.3)$$

where  $|\psi\rangle \in \text{Hilbert space}(\mathcal{H})$  and  $\sum_i |x_i|^2 = 1$

We will also employ some more data strategies that were described in the literature review section in addition to these two data encoding approaches.

The state parameters(x) will be encoded into a feature map by the first few gates in the circuit, while variational parameters( $\theta$ ) will be used by the succeeding gates. The circuit architecture is random, with the exception of the basic rule that the input and variational parameters are utilized as arguments for the gates. Furthermore, similar to the weights in a neural network, the variational parameters are set to a random value. Selecting an appropriate circuit that produces the best outcomes is a time-consuming process. We will use several well-documented feature maps to improve accuracy. If we have N-dimensional input parameters, we may use the displacement operator with N qubits to encode them.

**Measurement:** Measurement of the QVC at the end is also known as the expected value. With some traditional post-processing, we will estimate the cost of the circuit from these anticipated values. In Fig. 7. Here is an example of how to calculate the circuit's traditional post-processing cost.

```
def cost(weights, x_train, labels):
    predictions = [circuit(weights, f) for f in x_train]
    loss=0
    for i in range(len(predictions)):
        min=predictions[i][0]
        max=predictions[i][0]
        for k in range(3):
            if predictions[i][k]>max:
                max=predictions[i][k]
            if predictions[i][k]<min:
                min=predictions[i][k]
        x=(predictions[i][labels[i]+1]-min)/(max-min)
    loss+= (1-x)**2 # [0.4,0.3,0.3]
    return loss/len(predictions)
```

Figure 5.1: An example of Code for classical post-processing calculation

**The architecture of the Variational Circuit:** There are several Ansatzes for Variational Circuits that have been presented. The application determines the circuit's strength and precision. A circuit that provides great precision for one set of images may not provide the same level of precision for other sets. As a result, the Variational Circuit's architecture is critical in deciding the accuracy of the classifier model. Variational Circuits are often divided into three designs.

**Layered gate Architecture:** A layer is formed by a series of gates in layer architecture. This layer is repeated based on the demand, forming a hyperparameter. The



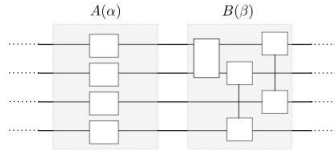


Figure 5.2: A typical structure of Layered and Alternating Ansatz

circuit is frequently divided into two units, A and B. These are further categorized into subclasses based on the parameterization of blocks A and B.

**Alternating operator Ansatz:** As these Ansatz are represented by two blocks A and B, they are comparable to the Layered Gate design. The main difference is that Hamiltonian, which changes over time  $t$ , defines the unitaries of expressing the blocks A and B. This approach is similar to Adiabatic Quantum Computing in concept.

**Tensor network Ansatz:** This architecture is made up of a single permanent structure rather than layers like Layered or Alternate operators. Figure 9 illustrates an example of this circuit.

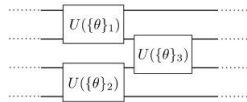


Figure 5.3: A typical architecture of Tensor network Ansatz

# Chapter 6

## Quantum Data Encoding

When working with the classical data on a quantum device, we must verify that the classical data is well converted into quantum data. We need data encoding, which will convert traditional binary data to quantum data and vice versa.

There are a variety of data encoding techniques that may be used to accomplish this. One option is to use qubits to transform all classical binary data into quantum data, but this could take more time and occupy more space because to convert a single-precision floating-point from a classical system to a quantum system, could take more than thirty qubits.[1] So, we aim to use the following two encoding techniques in our thesis study, although we have various theoretical techniques.[1]

**Scaled Encoding:** This method is intended for setups input data that falls within some defined constraints. The procedure is straightforward; it scales every parameter between 0 and  $2\pi$  then rotates across  $R_x$  and  $R_z$ . Here,  $R_x$  and  $R_z$  represent parameterized rotation gates which will be used along with some other quantum gates to build the proposed quantum variational circuit or QVC (Literature review, section G). The binary input values that will be transformed into quantum data are denoted by parameter.

However, the approach described above has certain drawbacks. We may encounter a circumstance in which the binary input data is not between 0 and  $2\pi$ . If the constraints exceed the bounds i.e.,  $-\infty$  to  $\infty$ , this method may not work effectively. In such a situation we could use another data encoding technique which is called **Directional Encoding**, which involves qubit rotation respect to  $R_x$  and  $R_z$  either by 180 degree or zero, as defined by the following constraint: if the input is greater than 0 or positive then the rotation will be  $\pi$  otherwise there will be no rotation.

The above-mentioned techniques are efficient and would require two gates for each qubit. Some data encoding techniques are theoretical and not suitable for the quantum variational circuit (QVC) based deep reinforcement learning.[20][21] Because some techniques are impossible to simulate on TensorFlow quantum, TensorFlow Quantum is a library for quantum machine learning, and some require more and more quantum gates to build.

We have studied various types of quantum data encoding techniques but have yet to apply them in our work; however, we intend to do so in future works such as optimization problems based on Quantum Variational Circuits (QVC).

# Chapter 7

## Quantum Reinforcement Learning

### 7.1 Introduction

Like traditional reinforcement learning, three main sub-elements can also be identified by a quantum reinforcement learning system: an environment, a function that gives reward and a policy or rule. However, quantum reinforcement learning algorithms vary considerably from standard RL algorithms mostly in four ways.

### 7.2 Representation

We now have specifications and theorems for quantum reinforcement learning since we describe a QRL process applying quantum concepts. Definition 1: Choosing a measurable quantum system as well as its eigenvectors in a Hilbert space yields a collection of full orthogonal bases. The quantum state of a generic enclosed quantum system described as a state  $|\psi\rangle$  in a Hilbert space. We write the inner product of  $|\psi_1\rangle$  and  $|\psi_2\rangle$  into  $\langle\psi_1|\psi_2\rangle$  and the condition for being normalization is  $\langle\psi|\psi\rangle = 1$ . In a basic quantum mechanical system, it is possible to describe the state of the qubit as  $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$  and the condition of its normalization is equivalent to  $|\alpha|^2 + |\beta|^2 = 1$ . [2]

In quantum Information, according to the superposition principle, Because a quantum reinforcement learning system may exist in orthogonal quantum states , it could exist in a random superposition. We can write the superposition as follows:

$$|\psi\rangle = \sum_n \beta_n |\psi_n\rangle \quad (7.1)$$

We can obtain Proposition 1 in a quantum reinforcement learning system for any random state. An invariance state, s in QRL can be expressed as follows in respect of an orthogonal collection of eigenvalue states s (or eigen actions a):[2]

$$|S\rangle = \sum_n \alpha_n |s_n\rangle \quad (7.2)$$

$$|A\rangle = \sum_n \beta_n |a_n\rangle \quad (7.3)$$

Where  $\alpha_n$  and  $\beta_n$  are amplitudes of probability, and  $\sum_n |\alpha_n|^2 = 1$  and  $\sum_n |\beta_n|^2 = 1$ . are satisfied.

QRL's states and actions vary somewhat from conventional RL: I. In traditional RL, the sum of multiple states has no definite meaning, however, the sum of QRL states remains a potential state of the same quantum system. If  $|S\rangle$  assumes its own state  $|S_n\rangle$ , it is exclusive. Nevertheless, the likelihood of being in the eigenstate of  $|S_n\rangle$  is  $|\alpha_n|^2$ . It is also applicable for the  $|A\rangle$  (action).[2]

Quantum computation is based on the qubit. Multiple qubit systems express states and actions . Let the number of states and actions be  $N_s$  and  $N_a$ , then select numbers m and n. The following inequalities are identified by  $N_s$  and  $N_a$ ,

$$N_s \leq 2^m \leq 2N_s, N_a \leq 2^n \leq 2N_a \quad (7.4)$$

Here, m and n qubit to respectively represent the state,  $S = \{|S_i\rangle\}$  and action,  $A = \{|A_j\rangle\}$ . We get the following relation:

$$|S^{(N_s)}\rangle = \sum_{i=1}^{N_s} C_i |S_i\rangle \rightarrow |S^{(m)}\rangle = \sum_{S=00\dots 0}^{\overbrace{11\dots 1}^m} C_s |S\rangle \quad (7.5)$$

The same relation also can be obtained for Action  $|A\rangle$ . The amplitude of probability  $C_s$  and  $C_a$  are complex figures and fulfill **b5**

$$\sum_{s=00\dots 0}^{\overbrace{11\dots 1}^m} |C_s|^2 = 1 \quad (7.6)$$

$$\sum_{a=00\dots 0}^{\overbrace{11\dots 1}^n} |C_a|^2 = 1 \quad (7.7)$$

### 7.3 Action selection policy

The agent will learn a policy  $\pi : S \times U_{i \in S} A_{(i)} \rightarrow [0,1]$ , that will optimize the projected value of each state's award in QRL. The mapping is  $\pi : S \rightarrow A$ , and we get

$$f(s) = |a_s^n\rangle = \sum_{a=00\dots 0}^{\overbrace{11\dots 1}^n} c_a |a\rangle \quad (7.8)$$

In this case, the action selection system is founded on the collapse postulate: If an action  $|A\rangle = \sum_n \beta_n |a_n\rangle$  is measured, it is altered and randomly collapses into one of its actions  $|a_n\rangle$  with this probability  $|\langle a_n | A \rangle|^2$ :

$$|\langle a_n | A \rangle|^2 = |(\langle a_n |) |A\rangle|^2 = \left| (\langle a_n |) \sum_n \beta_n |a_n\rangle \right|^2 = |\beta_n|^2 \quad (7.9)$$

We increase the likelihood of a "good" action in the QRL algorithm based on the appropriate incentives. When we measure the quantum state, it is a real number value. Without measurement this results in between exploration and exploitation and a natural "action choice". [2]

## 7.4 Paralleling state value updating

In proposition 1, We mentioned that in terms of an orthogonal complete set of eigen state  $|S\rangle$ , every possible state of QRL can be extended as follow:

$$|S_n\rangle : |S\rangle = \sum_n \alpha_n |S_n\rangle \quad (7.10)$$

A unitary operation of U on the qubits can indeed be enforced as per quantum parallelism. Presume we now have a function that can analyze these  $2^m$  states mostly with TD(0) value update rule at the very same time:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s)) \quad (7.11)$$

It is like updating the traditional RL's parallel value over all states.

## 7.5 Probability amplitude updating

Action method is determined in QRL by measuring action  $|a_s^n\rangle$  related to certain state  $|S\rangle$ , which collapses to  $|a\rangle$  with the probability of  $|C_a|^2$  occurring. There is therefore no doubt that updating the probability amplitude is the key to measuring the experience of "trial-and-error" and learning to be more skilled. The update of the amplitude of probability is based on the Grover iteration [22]. Initially, the evenly balanced superposition must be prepared.

$$|a_o^{(a)}\rangle = \frac{1}{\sqrt{2^n}} \left( \sum_{a=00\dots0}^{\overbrace{11\dots1}^n} |a\rangle \right) \quad (7.12)$$

This method may be readily accomplished by applying n separate qubits with starting states of 0 each.[22].Which may be stated in the following way:

$$H^{\oplus n} \left| \overbrace{11\dots1}^n \right\rangle = \frac{1}{\sqrt{2^n}} \left( \sum_{a=00\dots0}^{\overbrace{00\dots0}^n} |a\rangle \right) \quad (7.13)$$

We take  $U_a$  and  $U_{a_0(n)}$  to construct the Grover iteration.

$$U_a = I - 2|a\rangle \langle a| \quad (7.14)$$

$$U_{a_0(n)} = H^{\oplus n} (2|0\rangle \langle 0| - I) H^{\oplus n} = 2 \left| a_0^{(n)} \right\rangle \left\langle a_0^{(n)} \right| - I \quad (7.15)$$

In the Grover algorithm[11], where I is the unitary matrix and  $U_a$  is oracle O.  $|a\rangle \langle a| = |a\rangle \langle a|$  ( $|a\rangle \langle a|$ )\* is defined as the external product  $|a\rangle \langle a|$ . [2]

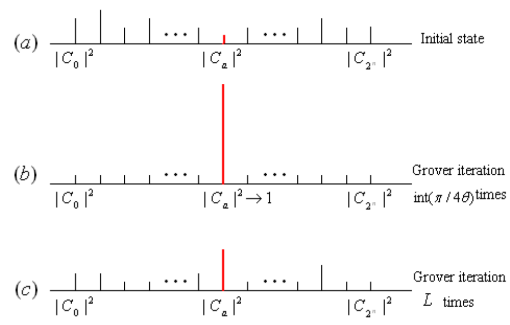


Figure 7.1: Grover algorithm in QRL

# Chapter 8

## Experiments & Results

### 8.1 Experiments

We are updating the conventional DQN on the Atari, as well as the LunarLander-v2, in preparation for the Quantum Variational Circuit (QVC) based Deep Reinforcement Learning (DRL) implementation in this study. To evaluate and validate the performance of DQN, we use traditional methods on Atari and Lunar Lander-v2. The goal is to compare the performance of conventional algorithms versus QVC-based solutions. So, we discuss about Q-Learning and its components. It is the most fundamental approach for deep reinforcement learning, and it has some terms associated with it. They are as follows:

**Agent:** It is an entity that will exist in an environment to execute activities that will change the state of the environment in order to receive rewards.

**Environment:** The Environment is the universe in which the agent exists, and it has a distinct state that can alter the agent's behavior.

**Step:** It occurs several times when the agent makes an activity that changes the condition of the environment.

**Episode:** A series of events that ends in the environment entering a terminal state.

**Epoch:** A training iteration of the agent with the specified number of episodes.

**Terminal State:** When the agent has won or lost, or when the environment has taken more than the maximum number of steps.

#### **Classical DQN based LunarLander VS QVC based LunarLander:**

The Lunar Lander game's landing platform placement is at x equals 0 and y equals 0. At zero speed, moving from the top of the screen to the landing pad awards the agent some points. If the lander departs from the landing space, it forfeits the prize. In this work, we state points which means the integer value. While the lander cracks, then users will receive an extra -100 or +100 points. Moreover, every leg's bottom contact is worth ten points. Each frame costs -0.3 points to fire the main engine. The total score was 197. There is a possibility that it could land outside the platform. Because fuel is limitless, an agent may learn to fly and land on the first try. Now we implement the DQN to train agents to solve it with the help of the OpenAI gym. Here there have been four alternatives for users to choose from:

1. Get nothing
2. Fire the left-oriented motor
3. Fire the main motor

#### 4. Fire the right-oriented motor

```

env = gym.make('LunarLander-v2')
env.seed(0)
print('State shape: ', env.observation_space.shape)
print('Number of actions: ', env.action_space.n)

```

Figure 8.1: OpenAI Gym Environment

To install the OpenAI Gym dependencies into the environment, we can use the following code

Again we developed an Agent class to train the model. Then, using the DQN algorithm, we construct a DQN Agent, which determines the outcome after training the Agent.

**For DQN:** To estimate the optimal action-value function, we utilize a deep neural network as follows:

$$Q^*(s, a) = \max E \left[ \sum_{s=0}^{\infty} \gamma^s r_{t+s} \mid s_t = s, a_t = a, \pi \right] \quad (42)$$

Figure 8.2: Equation

Here, Q represents the action-value function, s represents states, a represents actions, rt represents the maximum amount of rewards discounted by at each time step t, and  $\pi$  represents the behavior policy.

#### PQC(or QVC) based DRL LunarLander:

TensorFlow-quantum must be installed before we can run the LunarLander with the DQN algorithm in the PQC or QVC. To perform this operation, we use 5 qubits. Because LunarLander is such a complicated environment. A complicated environment necessitates a large number of measurements. Here's an example of employing 5 qubits:

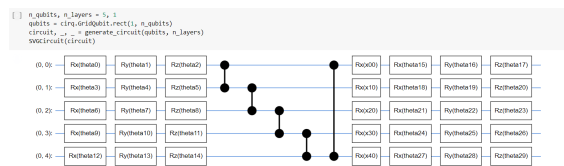


Figure 8.3: QVC based on 5 qubits

In this case, we employ Rx, Ry, and Rz gates, each of which is parametrized by a distinct Q. Then, in the rotation gates, we utilize a set of CNOT gates to act on the qubits. We receive the result after training the Agent with the PQC(or QVC) DQN method, which is displayed in the result section.

#### Owen Lockwood's QVC based RL for CartPole , 3 Qubit based QVC on CartPole using the DQN Algorithm VS 4 Qubit based QVC on CartPole using the DQN Algorithm:

CartPole is a game included with OpenAI Gym. It seems to be a pendulum in this setting, taking gravity in the middle to its pivot point. It may be unstable at first, which means it will shift to the right or left to steady itself. As a result, the major



goal of this game is to maintain the CartPole steady by adjusting the pivot point. [1] A pole is attached to a cart that moves over a frictionless track with the help of an unactuated joint. The device is controlled by applying a +1 or -1 pressure on the cart. The pendulum starts out upright, and the goal is to keep it from falling over. For each timestep that the pole remains erect, a +1 is awarded. The episode finishes when the pole travels more than 15 degrees from vertical or the cart moves more than 2.4 inches from the center. The author of the PosterLockwood article utilized Directional Encoding to construct QVC RL using the DQN algorithm. They utilized Directional Encoding because it has an endless number of input values and can tackle complex settings such as CartPole. This encoding has simplified the technique by allowing the input to be arrays of floating point and integer data. In the result section, a graph depicts the outcome of this Encoding CartPole. To finish the job, we used three qubits in the 3 Qubit CartPole. Here's an example:

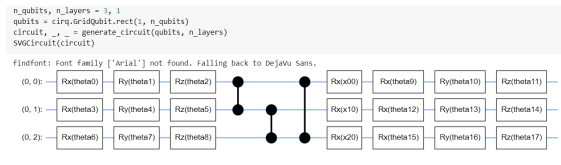


Figure 8.4: QVC based on 3 qubits

We utilized 1000 episodes to see which system produced the best outcomes.

```
[ ] state_bounds = np.array([2.4, 2.5, 0.21, 2.5])
    gamma = 1
    batch_size = 10
    n_episodes = 1000
```

Figure 8.5: Figure representing 1000 episodes

Another picture of using 4 qubits to train the agent:

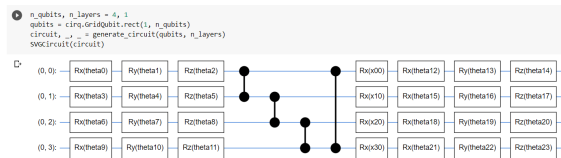


Figure 8.6: QVC based on 4 qubits

The result and graph obtained after training the agent are shown in the Result section.

### Classical DQN based Atari VS QVC DQN based Atari:

In this work, we use DQN on OpenAI Gym Atari games. OpenAI Gym is a simple to use general intelligence benchmark with a unique setting. The environment, which determines the game in which our reinforcement algorithm will compete, is the most important aspect of OpenAI-Gym. The Gym library environment may provide us with a variety of features. To build up the gym library, we're utilizing Breakout-v0. Here's an example of how to use Breakoutv0:

Atari(Pong-v0), First and foremost, we chose Pong-V0 to practice the Atari game. This version is observed on an RGB (Red, Green, Blue) picture of the screen. It is an array with the size [210, 160, 3], with each action repeating for k intervals.

```

query_environment("Breakout-v0")
Action Space: Discrete(4)
Observation Space: Box(210, 160, 3)
Max Episode Steps: 10000
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: None

```

Figure 8.7: Atari Environment form OpenAI Gym

Following that, we select the maximum number of iterations since increasing the number of collecting stages will help the algorithm train well. The next step is to configure Atari’s Environment to carry out the following process. The environments of several Atari games change depending on whether they are shown in 2D or 3D. In our study, the Atari games are represented as a 3D using Open-AI Gym. For effective computing To speed up the computation, we skipped certain frames, lowering the resolution.

We created the Pong-v0 environment after utilizing Breakout-V4. Here is an image of the game’s module:

```

env.reset()
PIL.Image.fromarray(env.render())

```

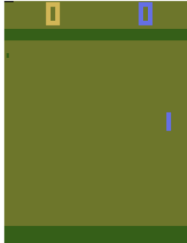


Figure 8.8: Atari game’s module

## 8.2 Results

### Classical DQN based LunarLander VS QVC DQN based LunarLander:

We achieve 200.02 in 987 episodes after training the agent with the DQN method in conventional RL. Furthermore, after training the agent in PQC with the DQN algorithm: 291.3 within 220 episodes. Here is the picture:

Episode 100	Average Score: -185.86
Episode 200	Average Score: -120.30
Episode 300	Average Score: -41.34
Episode 400	Average Score: -27.98
Episode 500	Average Score: -7.89
Episode 600	Average Score: 22.11
Episode 700	Average Score: -36.13
Episode 800	Average Score: -19.06
Episode 900	Average Score: 76.79
Episode 987	Average Score: 200.02
Environment solved in 887 episodes! Average Score: 200.02	

Figure 8.9: Result of DQN algorithm in classical RL.

Here, We set the average score for both trials at 200, and after computing the job, we see that the traditional RL requires 987 episodes to achieve 200.02 and the PQC

```
Episode 150, average last 10 rewards 157.8
Episode 160, average last 10 rewards 149.0
Episode 170, average last 10 rewards 157.1
Episode 180, average last 10 rewards 120.5
Episode 190, average last 10 rewards 122.3
Episode 200, average last 10 rewards 141.1
Episode 210, average last 10 rewards 135.6
Episode 220, average last 10 rewards 291.3
```

Figure 8.10: Result of DQN algorithm in PQC RL.

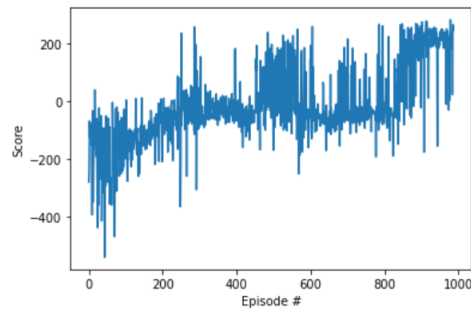


Figure 8.11: Graph of DQN Algorithm using classical RL

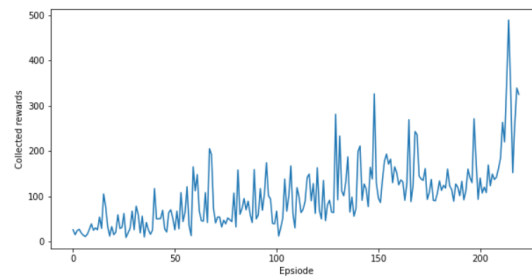


Figure 8.12: Graph of DQN Algorithm with PQC RL

RL requires 220 episodes to reach 291.3. Based on this finding, we may infer that PQC RL outperforms traditional RL.

Here is a table that clearly shows the distinction between them:

<b>Name</b>	<b>Episodes</b>	<b>Score</b>	<b>Time</b>
Classical RL	987	200.02	4 hours
QVC based RL	220	291.3	6 Hours

Table 8.1: Comparison between Classical RL and QVC base RL with DQN Algorithm

Owen Lockwood's QVC based RL for CartPole , 3 Qubit based QVC on CartPole using the DQN Algorithm VS 4 Qubit based QVC on CartPole using the DQN Algorithm: From the PosterLockwood paper we get the graph:[1]

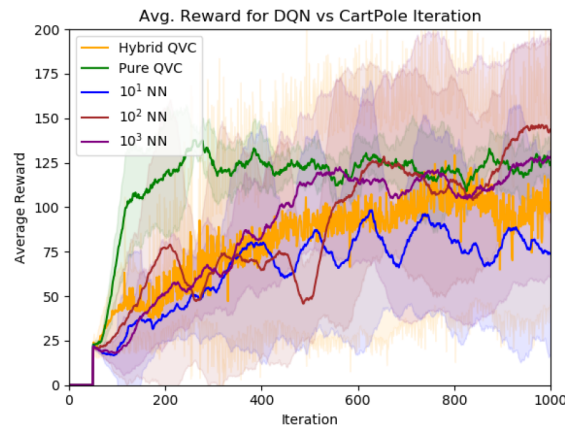


Figure 8.13: Graph of DQN algorithm in QVC RL [1]

They utilized 1000 episodes and received an average reward of approximately 125.5. On the other hand, we get the following graph as a consequence of participating three qubits based QVC with the DQN Algorithm:

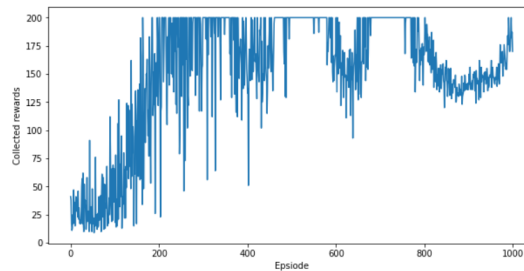


Figure 8.14: Graph of DQN Algorithm with 3 qubits QVC RL

The result is shown here:

```

Finished episode 900 Average rewards: 139.8
Finished episode 910 Average rewards: 142.4
Finished episode 920 Average rewards: 144.7
Finished episode 930 Average rewards: 144.1
Finished episode 940 Average rewards: 146.6
Finished episode 950 Average rewards: 146.0
Finished episode 960 Average rewards: 147.9
Finished episode 970 Average rewards: 147.6
Finished episode 980 Average rewards: 160.0
Finished episode 990 Average rewards: 167.6
Finished episode 1000 Average rewards: 182.9

```

Figure 8.15: Result of DQN Algorithm with 3 qubits QVC RL

Here is the another graph of 4 Qubits based QVC DQN:

In QVC DQN, we present the entire result of 4 Qubits based QVC of the DQN Algorithm:

From this diagram, we can see that after 500 episodes, all of the average prizes are more than 200, and it took a long time to finish till 1000 episodes. That is why we are only displaying the most recent 540 episodes for 4 qubits based QVC.

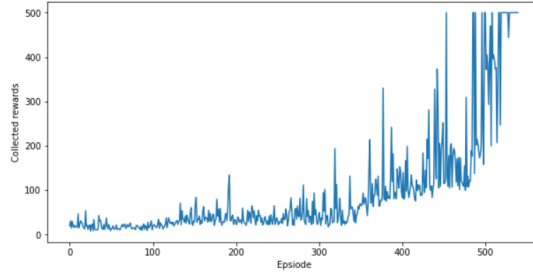


Figure 8.16: Graph of DQN Algorithm with 4 Qubits QVC RL

```

Episode 370/2000, average last 10 rewards 109.6
Episode 380/2000, average last 10 rewards 108.2
Episode 390/2000, average last 10 rewards 120.2
Episode 400/2000, average last 10 rewards 112.2
Episode 410/2000, average last 10 rewards 113.2
Episode 420/2000, average last 10 rewards 104.9
Episode 430/2000, average last 10 rewards 113.3
Episode 440/2000, average last 10 rewards 161.7
Episode 450/2000, average last 10 rewards 205.2
Episode 460/2000, average last 10 rewards 178.1
Episode 470/2000, average last 10 rewards 150.2
Episode 480/2000, average last 10 rewards 143.6
Episode 490/2000, average last 10 rewards 255.7
Episode 500/2000, average last 10 rewards 254.9
Episode 510/2000, average last 10 rewards 390.6
Episode 520/2000, average last 10 rewards 382.3
Episode 530/2000, average last 10 rewards 494.4
Episode 540/2000, average last 10 rewards 500.0

```

Figure 8.17: Result of DQN Algorithm with 4 qubits QVC RL

We can use a table to compare among the three:

Name	Episodes	Rewards
Owen Lockwood's QVC based RL for CartPole [1]	1000	125.5
3 Qubit based QVC on CartPole using the DQN Algorithm	1000	182.9
4 Qubit based QVC on CartPole using the DQN Algorithm	540	500

Table 8.2: Comparison among Owen Lockwood's QVC based RL for CartPole [1], 3 Qubit based QVC on CartPole using the DQN Algorithm and 4 Qubit based QVC on CartPole using the DQN Algorithm

Based on this analysis, we can conclude that the 4 Qubit CartPole outperforms the others. Because increasing the qubit in a system or environment allows it to operate with more qubits, which can do many more measurements than fewer qubits. For them, the measurement will be speedier and the task will be completed in less time.

**Atari DQN with Classical RL VS Atari DQN with QVC RL:** We obtained the following result after training the agent using traditional DQN (Figure 8.18): Furthermore, after training the agent using QVC based DQN, we obtained the following result (Figure 8.19):

A table can be used to compare the Classical DQN based Atari and the QVC based Atari using DQN algorithm (Table 8.3):

We can see from the data that the QVC based Atari using DQN performed better than the Classical DQN based Atari.

```

step = 205000: loss = 0.007518010213971138
step = 210000: loss = 0.028996415436267853
step = 215000: loss = 0.01371004804968834
step = 220000: loss = 0.007023532874882221
step = 225000: loss = 0.004790903069078922
step = 225000: Average Return = -4.400000095367432
step = 230000: loss = 0.006244136951863766
step = 235000: loss = 0.025019707158207893
step = 240000: loss = 0.02555653266608715
step = 245000: loss = 0.012253865599632263
step = 250000: loss = 0.004736536182463169
step = 250000: Average Return = 2.4000000953674316

```

Figure 8.18: Result of DQN Algorithm with 4 qubits QVC RL

```

step = 130000; loss = 0.0083117931201081574
step = 135000; loss = 0.0021548856969878444
step = 140000; loss = 0.0102145236548765546
step = 145000; loss = 0.0008974566324233254
step = 150000; loss = 0.0100124369875632142
step = 150000; Avg. Return = -3.656254562147895320
step = 155000; loss = 0.0245987563214785960
step = 160000; loss = 0.0132458976541236589
step = 165000; loss = 0.0001457896541236547
step = 170000; loss = 0.0059874236987456321
step = 175000; loss = 0.0187456321478956354
step = 175000; Avg. Return = 2.671458975623852964

```

Figure 8.19: Result of DQN Algorithm with 4 qubits QVC RL

Name	Steps	Avg. Return	Time
Classical DQN based Atari QVC RL	250000	2.40	14 hours
QVC based Atari using DQN	175000	2.67	11 hours

Table 8.3: Comparison between the Classical DQN based Atari and QVC based Atari using DQN

# Chapter 9

## Conclusion

We built the QVC-based RL in our study to see if it outperformed the conventional RL. In addition, we demonstrated that our QVC-based RL approach outperformed their QVC RL system. Furthermore, we discovered that training an agent with enough iterations or episodes can result in better results. Taking more Qubits may result in higher scores or better incentives after the agent has been trained. To solve a complex environment, we need more computations and time. In this case, increasing the number of Qubits will allow us to measure the result more quickly and precisely. Although we used the QVC-based approach in the CartPole, Atari, and Lunar Lander settings, there should be opportunities to use it in other situations in the future. Adopting the DDQN algorithm may yield better results than the DQN Algorithm in the future. Furthermore, using various Encoding techniques to solve the QVC-based RL may yield better results in the future. We will be able to improve performance if we can apply Quantum Computing to other fields such as quantum finance optimization problems and quantum web security.



# Bibliography

- [1] 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.
- [2] 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.
- [3] E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” *arXiv preprint arXiv:1411.4028*, 2014.
- [4] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, “Quantum circuit learning,” *Physical Review A*, vol. 98, no. 3, p. 032309, 2018.
- [5] R. Sutton, *Anda. g. barto, “reinforcement learningan introduction”*, 1998.
- [6] 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. 141007–141024, 2020.
- [7] 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 *International conference on machine learning*, PMLR, 2016, pp. 1928–1937.
- [8] 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.
- [9] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.
- [10] S. GAMBLE, “Quantum computing: What it is, why we want it, and how we’re trying to get it,” in *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2018 Symposium*, National Academies Press, 2019.
- [11] M. A. Nielsen and I. L. Chuang, *Quantum computing and quantum information*, 2000.
- [12] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, “Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets,” *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

- [13] 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, 2016.
- [14] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe, “Circuit-centric quantum classifiers,” *Physical Review A*, vol. 101, no. 3, p. 032 308, 2020.
- [15] E. Farhi and H. Neven, “Classification with quantum neural networks on near term processors,” *arXiv preprint arXiv:1802.06002*, 2018.
- [16] M. Schuld and N. Killoran, “Quantum machine learning in feature hilbert spaces,” *Physical review letters*, vol. 122, no. 4, p. 040 504, 2019.
- [17] M. Schuld and F. Petruccione, *Supervised learning with quantum computers*. Springer, 2018, vol. 17.
- [18] A. Mott, J. Job, J.-R. Vlimant, D. Lidar, and M. Spiropulu, “Solving a higgs optimization problem with quantum annealing for machine learning,” *Nature*, vol. 550, no. 7676, pp. 375–379, 2017.
- [19] G. Sergioli, R. Giuntini, and H. Freytes, “A new quantum approach to binary classification,” *PloS one*, vol. 14, no. 5, e0216224, 2019.
- [20] G.-L. Long and Y. Sun, “Efficient scheme for initializing a quantum register with an arbitrary superposed state,” *Physical Review A*, vol. 64, no. 1, p. 014 303, 2001.
- [21] P. Q. Le, F. Dong, and K. Hirota, “A flexible representation of quantum images for polynomial preparation, image compression, and processing operations,” *Quantum Information Processing*, vol. 10, no. 1, pp. 63–84, 2011.
- [22] L. K. Grover, “Quantum mechanics helps in searching for a needle in a haystack,” *Physical review letters*, vol. 79, no. 2, p. 325, 1997.