Solving the Stochastic Multi-Armed Bandit Problem on a Photonic Quantum Computer

by

Nathaniel Timothy Bunch

Submitted in partial fulfillment of the requirement for Computer Science Honors in Computer Science

Houghton College, Houghton, New York

May 2019

Honors Committee:

Chair Name: Signature: _____

Professor Name: Signature: _____

Professor Name: Signature: _____

Solving the Multi-Armed Bandit Problem on a Photonic Quantum Computer

Nathaniel T. Bunch

Abstract

Machine learning is the study of how to teach computers (learning agents) to learn and gain insight from either a dataset or a given environment. There are three common types of machine learning: supervised, unsupervised, and reinforcement learning. In this paper we focus primarily on reinforcement learning. Reinforcement learning is where a computer is tasked with maximizing reward by interacting with an environment. The computer, otherwise known as the agent, learns to map situations to actions it should take. In this paper, we implement an agent through the use of a quantum neural network (QNN) to solve the multi-armed bandit problem. The quantum neural network is based on the continuous variable (CV) model of quantum computing. In the stochastic multi-armed bandit problem, the environment consists of a k-armed bandit, in which each arm of the bandit gives a reward for pulling it, described by a reward distribution. The agent's goal is to learn which arm provides the most reward, through interacting with all the various arms on the bandit. We made the problem more difficult, by shuffling the rewards for each of the arms part-way through the learning process, so that we might observe the effectiveness of the QNN. Our results show that, despite adding noise to the reward distribution and shuffling the rewards for each arm, our QNN successfully distinguishes the most rewarding arm on the bandit from the rest of the arms.

Keywords:

Machine Learning, Neural Network, Quantum Computing, Continuous Variable Model, Quantum Neural Network, Stochastic Multi-armed Bandit Problem

1. Introduction

Three common machine learning types are: supervised, unsupervised, and reinforcement learning. Supervised learning is where a computer learns from a set of example data, paired with a label for each example. The example is a description of a specific situation, and the label is a specification that indicates the correct action to take, given the situation [1]. Unsupervised learning is learning from a set of example data. The task in this type of learning is to extrapolate the labels for the data, by identifying hidden structures within the dataset. Reinforcement learning, unlike both supervised and unsupervised machine learning, is learning through interactions with a given environment. Algorithms that interact with their environment are called *agents*. The task of this learning is to maximize the reward signal, so that the maximum reward is achieved from an environment. The agent is complete, interactive, and goal-seeking. These characteristics enable it to map specific actions to situations [1].

1.1 Reinforcement Learning

In reinforcement learning, there are four main sub-elements other than the agent and environment: a *policy*, *reward signal*, *value function*, and the optional *model* of the environment. A policy is a mapping from perceived states in the environment to specific actions that should be taken when the agent is placed in those states. Over the course of the learning process, the policy is updated, attempting to maximize the reward from the environment [1]. The policy is the heart of the reinforcement learning agent, since it determines the agent's behavior.

The reward signal defines the overall goal of the given reinforcement problem. As the agent learns from the environment, every time the agent takes an action the agent receives a number known as the reward. The goal of the agent is to maximize this reward it receives in the long run. The reward is what determines an action is either a good or a bad action for the agent to take [1]. The rewards serve as the primary basis for the changes to the policy. If an action, given some situation, yields a low reward, then the policy can change to choose another action in the future. A reward can be stochastic functions of the state in the environment and the history of actions already taken.

The reward indicates the benefit or decrement of a given action in the short-term, whereas the value function pertains to what is beneficial in the long-term. Values of the states in an environment is the total amount of reward an agent can expect to receive over a long period of time, starting from any one of those states [1]. The value of a given state takes into account states that are likely to follow and the rewards available from those states. Some states might give a low reward, but have a high value, due to states that regularly follow afterwards having a high reward, or vice versa.

Reward and values relate by using values to determine which actions to take in order to determine how to achieve the highest reward by selecting other states with the highest value. Rewards are directly provided from the environment and do not require estimation and re-estimation calculations like values. This calculation of estimating and re-estimating values efficiently is arguably the most important detail in reinforcement learning [1].

The final sub-element of reinforcement learning is the model of the environment. This model mimics the behavior of the environment that the agent is in. Given a state and an action, the model predicts the next state and reward. This technique can be used to plan a strategy to maximize the reward more quickly. This is planning by anticipating future events before the agent takes an action. There are two types of agents: *model-based* which uses a model of the environment and *trial and error* agents, which solely rely on interactions with the environment for learning.

1.2 Artificial Neural Networks

One of the ways an agent can be implemented is through the use of an artificial neural network. An artificial neural network (ANN) is a versatile and effective machine learning technique. ANNs typically consist of a collection of interconnected neurons, similar to that of a vast network of a biological brain's neurons or an ANN can be constructed from just a single neuron [3]. Each neuron has three distinct parts: the input, the summer, and the activation function. Each input value *p* has a weight *w* associated with it and a bias *b*, which are summed together to produce the net input *y* [4] where *n* is the number of inputs:

$$y = \sum_{i=0}^{n} w_i p_i + b_i \tag{1}$$

This net input in then passed to some activation function *f*, which produces the output *a* for that neuron [3]:

$$a = f\left(\sum_{i=0}^{n} w_i p_i + b_i\right) \tag{2}$$

Most neural networks consist of three layers of artificial neurons: the input, hidden, and output layers. In the case that there is only one neuron, then the ANN is called a perceptron. The example of a basic neural network, as described by equations 1 and 2, is sufficient for approximating any continuous variable function [3,4].

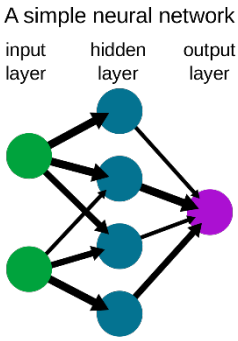

Figure 1: A basic, three-layered artificial neural network.

Many algorithms for machine learning and consequentially reinforcement learning fundamentally require the use of continuous variables, such as continuous vectors and tensors. Currently, these variables are estimated, due to the discrete nature of the conventional digital computing technology. Quantum computing makes use of quantum states as superposition, interference, and entanglement, which ultimately provides a unique advantage over classical computational methods [5]. The more familiar qubit-based quantum computing model (as defined by Josza et al [6]) does provide a drawback when approaching continuous variables: its nature is not wholly continuous, since such measurements of that system are discrete. Because of this fact, qubit-based hardware can be noted as digital quantum hardware and is only partially suited for continuous-variable related problems. The most naturally continuous model of quantum computing is the continuous variable (CV) model (as described by Killoran et. al. [5]). In this paper, we focus on the use of a CV model-based quantum neural network to tackle the multiarmed bandit problem in a stochastic setting.

1.3 The Stochastic Multi-armed Bandit Problem

The multi-armed bandit, also known as the *k*-armed bandit, problem is an environment consisting of a *k*-armed bandit, where each arm has its own respective reward associated with it, according to a pre-defined reward distribution, *r*. Each time the agent pulls an arm, the agent receives a reward. The agent's is goal is to determine, over a sequence of trials, which arm provides the most payout. This classical problem is a simple model that has a trade-off between both exploration and exploitation of the environment [7]. With each trial, the agent chooses one of the *k* arms to pull and receives that arms reward, which can be either positive, negative, or zero. As the agent learns which of the arms pays out the most,

it also must attempt to maximize the total reward received. The agent may conclude that a particular arm is the highest paying arm, and be incorrect [7,8]. Because of the simplicity of this problem, we chose to take the problem a step further by providing a more realistic stochastic reward. We introduced stochastic-ness to the reward distribution by adding noise with a random normal distribution, with various σ values such that $\sigma \in \{0.3, 0.6, 0.9\}$. We default $\mu = 0$. We create this generated distribution according to equation 3:

$$N(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3}$$

The stochastic *k*-armed bandit problem is initialized in an environment very similar to the non-stochastic *k*-armed bandit problem. When a particular arm is pulled noise is added to the reward the agent receives. The noise fluctuates the reward either above or below the initial distribution. The higher the fluctuation, controlled by the σ parameter of the random normal distribution (see equation 3) is, the harder it is for the agent to find the high reward arm on the bandit. In this paper we focus on the application of a CV-model based quantum neural network to solve the stochastic *k*-armed bandit problem. We test the overall performance while increasing the stochastic nature of the problem on each set of trials. We also explore how the neural network responds to shuffling the reward distribution entirely during the learning process.

2. Related Work

Quantum information can be encoded with continuous degrees of freedom using the techniques described by Killoran et. al. [5]. In their work they describe a series of gates that work on continuous variables (CVs) that collectively form the fundamental building blocks for new quantum machine learning algorithms. These series of gates are grouped into two types: *Gaussian* and *non-Gaussian* [9,10]. A few examples of basic *Gaussian* gates are the *rotation* $R(\phi)$, *displacement* $D(\alpha)$, and *squeezing* $S(r)$ [11]. There is an additional basic two-mode *Gaussian* gate, the *beamsplitter* $BS(\theta)$. The basic *Gaussian* gates are defined below, according to Killoran et. al. [5]:

$$R(\phi): \quad \begin{bmatrix} x \\ p \end{bmatrix} \mapsto \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ p \end{bmatrix}, \tag{6}$$

$$D(\alpha): \quad \begin{bmatrix} x \\ p \end{bmatrix} \mapsto \begin{bmatrix} x + \mathrm{Re}(\alpha) \\ p + \mathrm{Im}(\alpha) \end{bmatrix}, \tag{7}$$

$$S(r): \quad \begin{bmatrix} x \\ p \end{bmatrix} \mapsto \begin{bmatrix} e^{-r} & 0 \\ 0 & e^{r} \end{bmatrix} \begin{bmatrix} x \\ p \end{bmatrix}, \tag{8}$$

$$BS(\theta): \quad \begin{bmatrix} x_1 \\ x_2 \\ p_1 \\ p_2 \end{bmatrix} \mapsto \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & -\sin\theta \\ 0 & 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ p_1 \\ p_2 \end{bmatrix}, \tag{9}$$

The ranges for the parameter values are $\phi, \theta \in [0, 2\pi]$, $\alpha \in \mathbb{C} \cong \mathbb{R}^2$, and $r \in \mathbb{R}$ [5]. Besides there being *Gaussian* gates, there are also *non-Gaussian* gates, with one of those gates being the cubic phase gate, which is used in our quantum neural network circuit (see fig. 2). The cubic phase is described as such:

$$V(\gamma) = e^{i\frac{\gamma}{3}\hat{x}^3} \tag{10}$$

where the cubic phase gate operates on the *x* axis in the *x, p* phase space (see [5] for definition of phase space), by use of the $\hat{x}$ operator, defined in the previous section in equation 4.

Most of the names of each of the Gaussian gates suggest that the gates have a linear character (see [5, 11] for definition of Gaussian). There is a relationship between Gaussian operations and affine transformations on phase space. For a system of *N* modes, the most general Gaussian transformation has the following effect:

$$\begin{bmatrix} \mathrm{x} \\ \mathrm{p} \end{bmatrix} \mapsto M \begin{bmatrix} \mathrm{x} \\ \mathrm{p} \end{bmatrix} + \begin{bmatrix} \alpha_r \\ \alpha_i \end{bmatrix}, \tag{11}$$

where *M* is a real-valued *symplectic matrix* and $\alpha \in \mathbb{C}^N \cong \mathbb{R}^{2N}$ is a complex vector with real and imaginary parts α_r and α_i. This affine structure will be the key for building quantum neural networks [5, 12].

A matrix *M* is said to be *symplectic* when it is said to satisfy the relation $M^T \Omega M = \Omega$ where

$$\Omega = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{12}$$

is the $2N \times 2N$ *symplectic form*. We can define Gaussian transformations as circuits which contain only the Gaussian gates.

3. Methods

In this section we shall discuss the quantum neural network. We developed this network based on the QNN by Killoran et. al. [5]. The purpose behind designing and constructing this neural network was to observe the effectiveness and accuracy on the stochastic *k*-armed bandit problem. The network consists of a single layer, which is all that is necessary for solving the *k*-armed bandit problem. This single layer is composed of displacement, beamsplitter, squeeze, and cubic phase gates as seen in figure 2. We chose to initialize our quantum neural network as a single-layered neural network to improve simulation time. If we so had chosen to add more layers, it certainly would have come at the cost of time for better accuracy [14]. Each gate in the QNN layer serves a specific purpose: the displacement gates are utilized for inputting the input values onto the qumodes for quantum computations, then we utilize beamsplitter gates to entangle all the qumodes. Once entanglement is achieved between each of the qumodes, we have squeeze gates that are used to adjust the learning of the QNN, by adjusting the squeezing of the qumode states in either the *x* or *p* axis in the quantum phase space. We follow the squeeze gates with cubic phase gates. All the gates are utilized in sequence to satisfy the role of a 'fully connected' matrix transformation [17, 18]. We add the cubic phase gate so that nonlinearity is added to the sequence of gates, let these cubic phase gates be represented by: ϕ. Using $\mathbf{z} = (\mathbf{x}, \mathbf{p})$, we can write the combined transformations into this form:

$$\mathcal{L}(\mathbf{z}) = \phi(M\mathbf{z} + \alpha) \tag{13}$$

where $\mathcal{L}$ signifies an individual neuron's output and is similar to equation 2. We chose the cubic phase gate to serve as the activation function due to other gates, such as the Kerr gate, causing the output to result in a non-number value (nan). Finally, after the cubic phase gates, we measure each of the qumodes in the Fock basis, resulting in real number measurements. Each of these numbers are then normalized, and then used as the output weights to determine which arm from the multiarmed bandit is the most profitable. Predictions about which arm of the *k*-armed bandit are made by taking the argmax of the output weights and the resulting value is the bandit prediction, see algorithm 1.

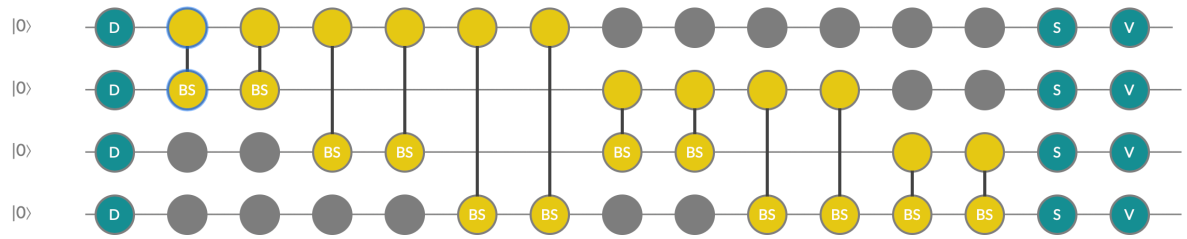

Figure 2. The QNN quantum circuit used to solve the stochastic k-armed bandit problem. All gates are fully parameterized. The displacement gates (D), are used to first apply the values to be learned by the neural network on the qumodes. The beamsplitter (BS) gates are then applied to ensure each qumode is entangled. This is followed by both squeeze (S) and cubic phase (V) gates. Each qumode is them measured in the Fock basis (F), which results in a respective measurement (m_n). These measurements are used to make a prediction of which bandit is the most rewarding.

Algorithm 1: Training of a quantum neural network to solve the stochastic *k*-armed bandit problem

---

QNN Parameters: BS, S, V Gate Parameters

QNN Input: Action taken (one index from 1 to k, where the index is the chosen arm to pull, and k is the number of arms of the multiarmed bandit)

Choosing an action: In order to choose an action, with the maximum reward, we take the index of the probability with the highest value.

Total episodes: 50,000

1. While episode < total episodes:
2. Choose an action from QNN
3. Apply chosen action to respective arm of the bandit and receive a reward.
4. Based on reward, update parameters of the QNN.
5. Record rewards for each bandit arm.

QNN Output: Output: Probability associated with each arm (probability distribution of size k, where each value is the probability of receiving a positive reward)

---

4. Results

In this section, we analyze the performance of the QNN on the stochastic *k*-armed bandit problem. The metric we primarily focus on is the average rewards received by the QNN.

Our results show that, despite both the increase in stochastic-ness of the environment and shuffling the reward distribution during the learning process, our QNN is robust and accurate enough to learn the most rewarding arm of the multiarmed bandit.

A k=4 sized array is used to represent the reward distribution of the arms on the k-armed bandit. The distribution is initialized with the values: $r = [0.0 \quad 0.5 \quad 0.3 \quad 0.2]$. After the shuffling of the reward distribution at 5000th episode, we resulted in the following: $r = [0.3 \quad 0.2 \quad 0.0 \quad 0.5]$. The same shuffled reward distribution is used through all the trial cases, in order to preserve consistency between trial cases. In order to confirm the capability of our QNN, the shuffling was carried out only once at the 5000th episode, so that we can observe when the network would pick which arm was the new most rewarding arm.

The QNN can take one of k actions, where each action is an index associated with one of values in the k-sized reward distribution and a probability of 10% that it would choose a random action. With the pulling of each arm, the QNN would be rewarded, according to the reward associated with each action by the reward distribution plus the stochastic learning rate. The objective of the QNN was simply to discover the most rewarding arm.

In order to compare the effectiveness of our QNN, we trained it on four different trials, with each trial consisting of 10 individual tests. One trial where there was no stochastic reward, and three others with increasing σ values of 0.3, 0.6, and 0.9 (see equation 3). Each trial was run in batches of 10 tests, with each test being 50,000 episodes, where the rewards for each bandit were recorded every episode, and the 10 different test runs were averaged. This was done to smooth-out the graphs, so that observations could be made easier from the results.

When running the QNN on the multiarmed bandit problem, we use the $\sigma = 0.0$ as a basis for analyzing each of the other results. The respective results can be found in figure 3.

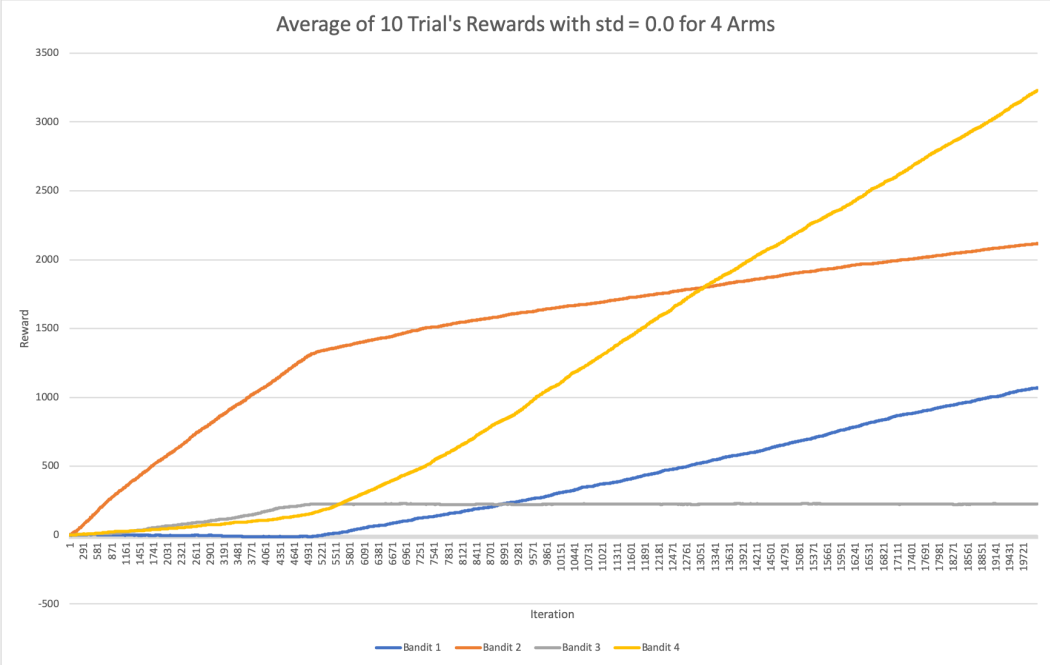

Figure 3. In this graph, it can be observed that, before iteration 5000, the second bandit had the highest reward, which is correct according to the starting distribution: [0.0, 0.5, 0.3, 0.2]. After the shuffle of the distribution at episode 5000, we can observe that the QNN makes a correction and realizes the fourth bandit is the most rewarding. This realization is true, according to the resulting shuffled distribution we used: [0.3, 0.2, 0.0, 0.5].

As we increased the stochastic-ness of the *k*-armed bandit environment between trials, we noticed that the number of iterations for the QNN to realize the new most rewarding arm on the *k*-armed bandit had also increased. The results of the σ values for 0.3, 0.6, and 0.9 can be found in figure 4, figure 5, and figure 6 respectively.

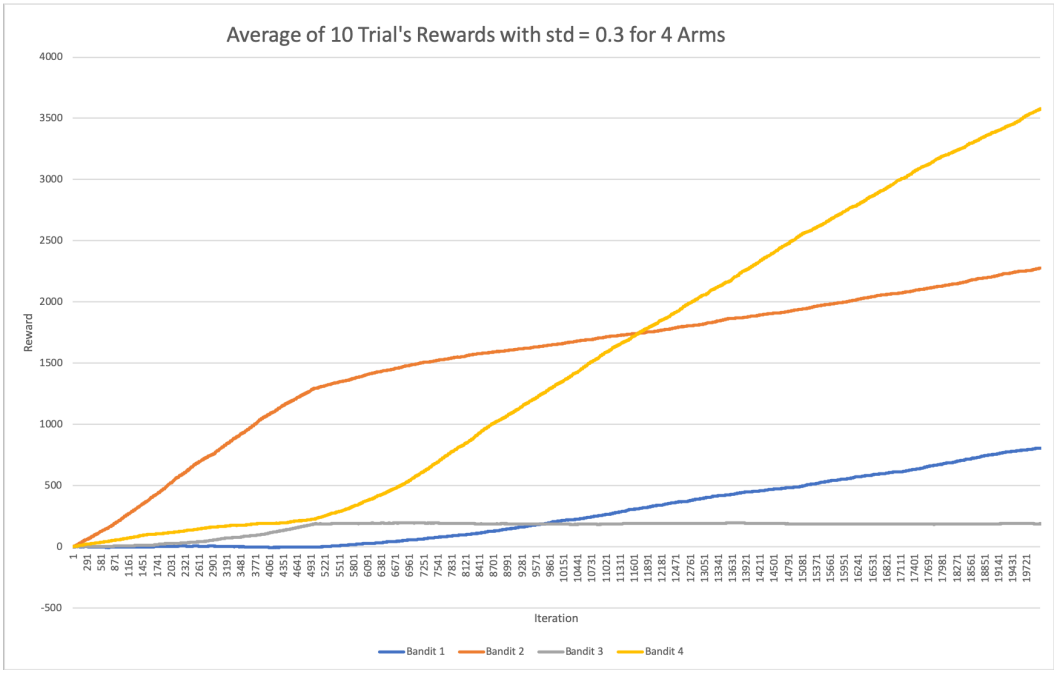

Figure 4. This graph resembles the $\sigma = 0.0$ graph. The slope of the most rewarding bandit is lower after the shuffling of the reward distribution. This decrease in the slope is more apparent with the increase of the stochastic-ness.

We made another observation, when the σ of the environment was increased to 0.6 and 0.9, the network had a hard time distinguishing the rewards for two of the bandit's arms. The two it struggled with was the second and third most rewarding, as the values for those arms respectively were 0.3 and 0.2. This can be observed in both figure 5 and figure 6.

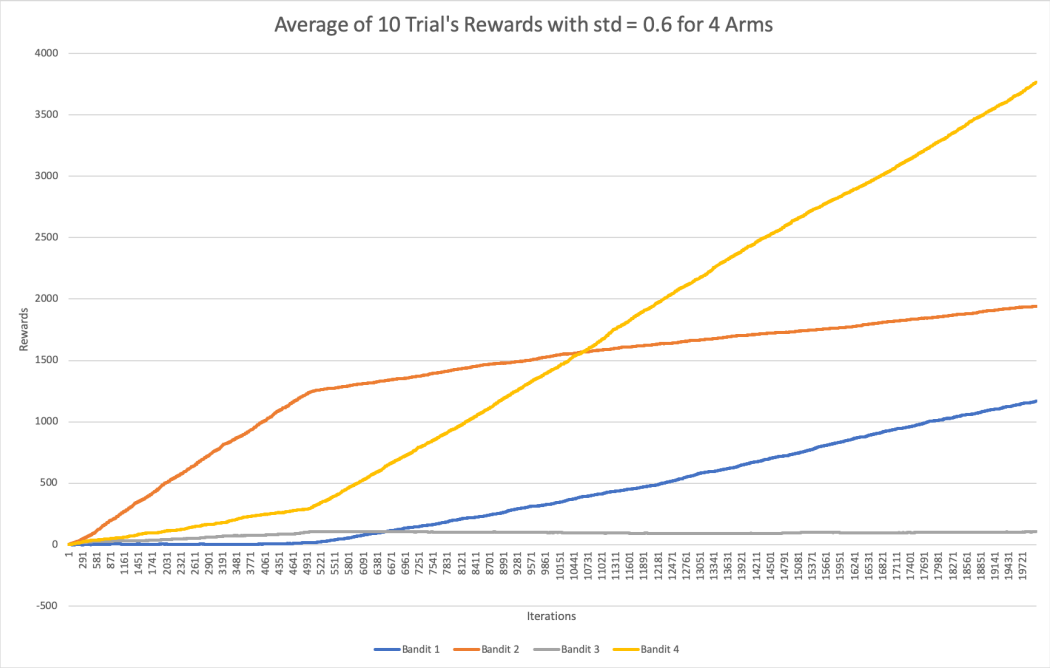

Figure 5. In this graph, we note that bandit arm 1 (which is the third most rewarding, in comparison to bandit arm 2), will eventually overtake bandit arm 2. It can also be noticed that the lines are more unstable as they increase, due to the reward being more randomized.

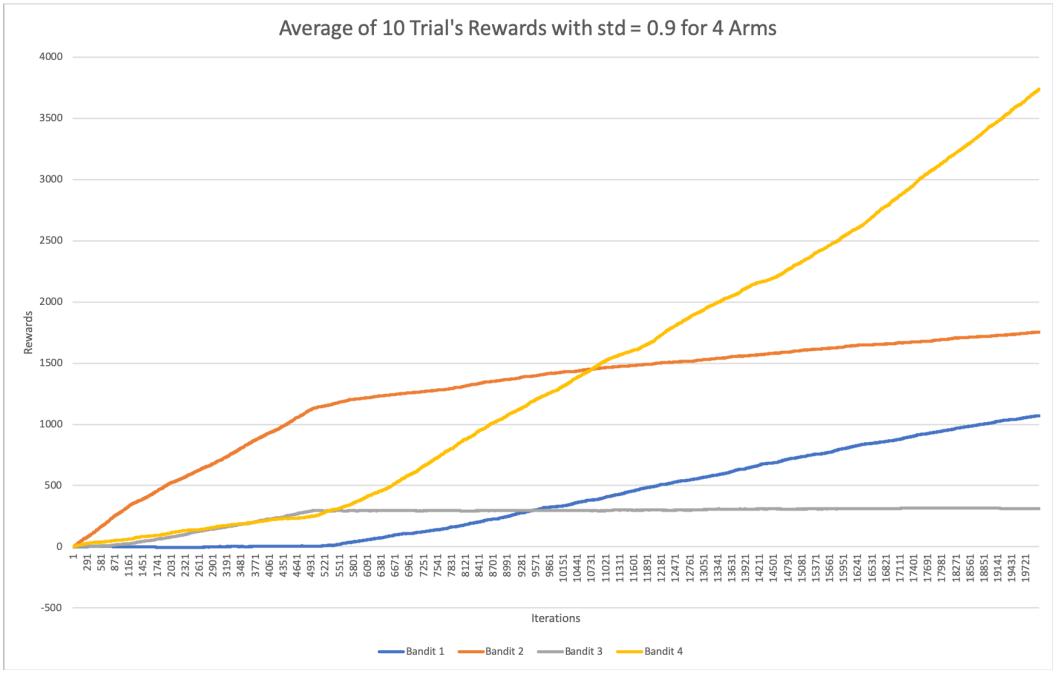

Figure 6. In this graph, it can be observed that there is much more instability in each reward for each bandit arm. This is due to the reward being mostly random (at $\sigma = 0.9$). The QNN, will observe, incorrectly, that bandit arm 1 is more rewarding than bandit arm 2. We can note too that, despite the

randomness, the most rewarding arm, bandit arm 4, is the most rewarding and that this graph resembles the base case of $\sigma = 0.0$.

5. Conclusion

This paper presents a quantum neural network to solve the stochastic *k*-armed bandit problem. Our findings have proven the quantum neural network to be both capable and robust enough to choose the most rewarding arm in this problem environment. In each of the trials with increasing σ values such that $\sigma \in \{0.3, 0.6, 0.9\}$ (see equation 3). The QNN was able to observe the most rewarding arm and exploit that arm to obtain the most reward. After the 5000^{th} episode for each trial, the reward distribution for the bandit arms was shuffled. Our quantum neural network was still able to detect the most rewarding arm afterwards.

References

[1]. Sutton, R.S. and Barto, A.G. (2017) Reinforcement Learning: An Introduction, *Vol. 1. MIT Press, Cambridge*.

[2]. Duryea, E., Ganger, M. and Hu, W. (2016) Exploring Deep Reinforcement Learning with Multi Q-Learning. *Intelligent Control and Automation*, 7, 129-144.

[3]. Dey. A. (2016). Machine Learning Algorithms: A Review. *Vol 7. International Journal of Computer Science and Information Technologies.* 7. 1174-1179.

[4]. Nazzal. J., El-Emary. I., Najim. S. (2008). Multilayer Perceptron Neural Network (MLPs) For Analyzing the Properties of Jordan Oil Shale. *World Applied Sciences Journal*. 5 (5): 546-552.

[5]. Killoran. N., Bromley. T., Arrazola. J., Schuld. M., Quesada. Nicolas., Lloyd. S. (2018) Continuous-variable quantum neural networks. *Xanadu. Massachusetts Institute of Technology*. arXiv:1806.06871v1.

[6]. Jozsa. R. (2005). An Introduction to Measurement Based Quantum Computation. *University of Bristol.* arXiv:quant-ph/0508124

[7]. Auer. P., Cesa-Bianchi. N., Freund. Y., Schapire. R. (2001). The Nonstochastic Multiarmed Bandit Problem. *Vol. 32, SIAM Journal on Computing*. Doi: 10.1137/S0097539701398375.

[8]. Bergemann. D., Valimaki. J. (2006). Bandit Problems. *Helsinki Center of Economic Research.*

[9]. Yanofsky, N. (2007). An Introduction to Quantum Computing. arXiv:0708.0261

[10]. Killoran. N., Izaac. J., Quesada. N., Bergholm. V., Amy. M., Weedbrook. C. (2018). Strawberry Fields: A Software Platform for Photonic Quantum Computing. *Xanadu*. arXiv:1804.03159v2

[11]. Ferraro, A., Olivares, S., Paris, M., (2005). Gaussian states in continuous variable quantum information. arXiv preprint quant-ph/0503237

[12]. Wittek, P. (2014). Quantum machine learning: what quantum computing means to data mining. *Academic Press*.

[13]. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S. (2017). Quantum machine learning. *Nature*. 549(7671):195.

[14]. Perdomo-Ortiz, A., Benedetti, M., Realpe-G´omez, A., Biswas, R. (2017). Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers. arXiv:1708.09757

[15]. Hornik, K., Stinchcombe, M., White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[16]. Maass, W., Schnitger, G., Sontag, E. (1994). A comparison of the computational power of sigmoid and boolean threshold circuits. In Theoretical Advances in Neural Computation and Learning, *Springer*. pages 127–151.

[17]. Cao, Y., Guerreschi, G., Aspuru-Guzik. A. (2017). Quantum neuron: an elementary building block for machine learning on quantum computers. arXiv preprint arXiv:1711.11240

[18]. Wan, K., Dahlsten, O., Kristjánsson, H., Gardner, R., Kim, MS. (2017). Quantum generalisation of feedforward neural networks. npj *Quantum Information*, 3(1):36