Quantum Multiple Q-Learning

by
Michael Ganger

Submitted in partial fulfillment of the requirement for Applied Mathematics and Computer Science Honors in Computer Science

Houghton College, Houghton, NY
May 2017

Honors Committee

Chair Name:	Wei Hu	Signature:_____
Professor Name:	Kurt Aikens	Signature:_____
Professor Name:	Kenneth Bates	Signature:_____

Quantum Multiple Q-Learning

by
Michael Ganger

Submitted in partial fulfillment
of the requirement for
Applied Mathematics and Computer Science Honors in
Computer Science

May 2017

Abstract

In this paper, a collection of value-based quantum reinforcement learning algorithms are introduced which use Grover's algorithm to update the policy, stored as a superposition of qubits associated with each possible action, and their parameters are explored. These algorithms may be grouped in two classes, one class which uses value functions ($V(s)$) and another class which uses action value functions ($Q(s,a)$). The $Q(s,a)$-based quantum algorithms are found to converge faster than $V(s)$-based algorithms, and in general the quantum algorithms are found to converge than their classical counterparts, netting larger returns during training. It is also found that the $Q(s,a)$ algorithms are more robust to higher learning rates. Furthermore, an increase in stability with respect to the learning rate is observed by introducing multiple $V(s)$ or $Q(s,a)$ functions, allowing higher learning rates to be used. These additional functions tend to improve the convergence properties in environments with stochastic rewards.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Reinforcement Learning

In recent years, the field of reinforcement learning has seen an increase in popularity. Reinforcement learning algorithms are a subset of machine learning algorithms which find an optimal sequence of actions to achieve a goal; unlike supervised learning algorithms, reinforcement learning algorithms solve an implicit problem. Because of this, these algorithms may be applied to a wide range of problem domains, from robotics [1] to buying and selling stocks [2].

The main goal of reinforcement learning is to maximize a signal, known as the *reward*, over a sequence of time steps, known as an *episode*, by finding a *policy* which describes what action to take from each state. The combination of the reward signal with the environment with which the agent interacts implicitly defines an optimal policy; the goal of all reinforcement learning algorithms is to find this policy or a good approximation of it.

While many algorithms for reinforcement learning work well in environments with a reasonable number of states, they become ineffective in large state spaces (such as a continuous state space). To address this, algorithms must use function approximation and train with a relatively small number of experiences. One recent success was the application of reinforcement learning to classic Atari games [3], which used a neural network approximation to Q-learning, a well known algorithm. This success was repeated a few years later using double Q-networks [4], demonstrating even greater success. Another recent demonstration of the power of reinforcement learning was its application to the game of Go. AlphaGo, a reinforcement learning algorithm which combines deep neural networks and tree search to learn the game of Go [5], was able to defeat the European champion of Go in multiple rounds. This represents a significant milestone in the field of reinforcement learning because there are about 2×10^{170} unique, legal board configurations [6], meaning exact reinforcement learning methods are intractable.

1.1.1 Markov Decision Process

Direct policy search algorithms search the policy space for the optimal policy; these algorithms, such as Williams' REINFORCE [7], do not make assumptions about the underlying environment. However, this can be sensitive to a noisy reward signal. A different approach is to assume that the process of reinforcement learning is a *Markov Decision Process* (MDP) [8]. A Markov Decision Process consists of an *agent*, which may take action a, interacting with an *environment*, which is in state s. At each time t, the state s_t expresses all relevant information about the previous states $\{s_{t-1}, s_{t-2}, \ldots\}$. When the agent takes the action a_t, the environment transitions to state s_{t+1} with probability $P\left(s_{t+1} \mid s = s_t, a = a_t\right)$ and receives reward r_t according to the expected reward function $r_t = r(s_t, a_{t-1}, s_{t-1})$.

The goal of the agent is to maximize a discounted *return*, defined as

$$G_t = \sum_{k=t}^{T} \gamma^k r_{t+k+1},$$

where T is the number of time steps until the end of the episode and γ is a discount parameter which decrease the importance of future rewards. The return is maximized by varying the policy $\pi\left(a_t \mid s = s_t\right) = P\left(a_t \mid s = s_t\right)$, which gives the probability that the agent will take action a_t when the environment is in state s_t. The return may also be express recursively as

$$G_t = r_t + \gamma G_{t+1}.$$

Using the policy and the transition probabilities, we may express the probability of the environment moving from s_t to s_{t+1} as

$$P\left(s_{t+1} \mid s = s_t\right) = \sum_b P\left(s_{t+1} \mid s = s_t, a = b\right) \pi\left(b \mid s = s_t\right).$$

Noting that the $P\left(r_{t+1} \mid s = s_t\right) = P\left(s_{t+1} \mid s = s_t\right)$, we may express the expected return given policy π as

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left\{V(s) \mid s_t = s\right\} \\ &= \mathbb{E}\left\{r_t + \gamma G_{t+1} \mid s_t = s\right\} \\ &= \sum_a \pi\left(a \mid s\right) \sum_{s'} P\left(s' \mid s, a\right) \left[r(s', a, s) + \gamma V(s')\right], \end{aligned}$$

where $r(s', a, s)$ is the reward for taking action a from state s and transitioning into state s'. In general, $V(s)$ is known as the value function, and the equality is known as the *Bellman Equation*.

The Bellman Equation uniquely defines a value function for a given policy, which may be determined iteratively from an initial estimate until it reaches convergence. This is known as policy evaluation. However, this does not allow us to update the policy. In order to update the policy, given a value function, we choose the action with the highest expected reward:

$$\pi\left(a \mid s\right) = \begin{cases} 1, & \text{if } a = \arg\max_a \sum_{s'} P\left(s' \mid s_t, a\right) \left[r(s', a, s_t) + \gamma V(s')\right] \\ 0, & \text{otherwise.} \end{cases}$$

This is known as policy improvement, and is guaranteed to update the policy. By alternating between policy improvement and policy evaluation, the optimal policy may be quickly achieved.

1.1.2 Q-Learning

We may introduce another function, the action value function, which gives the expected return given an action:

$$Q(s, a) = \sum_{s'} P\left(s' \mid s, a\right) \left[r(s', a, s) + \gamma V(s')\right].$$

Conversely, we may write the value function in terms of the action value function,

$$V(s) = \sum_a \pi\left(a \mid s\right) Q(s, a).$$

Combining these two relationships, we obtain a restatement of the Bellman Equation in terms of the action value function,

$$Q(s, a) = \sum_{s'} P\left(s' \mid s, a\right) \left[r(s', a, s) + \gamma \sum_{a'} \pi\left(a' \mid s'\right) Q(s', a')\right].$$

Observe that this gives a relationship between the action values corresponding to states separated by a single time step, s and s'. If we choose a policy of the form

$$\pi\left(a \mid s\right) = \begin{cases} 1, & \text{if } a = \arg\max_a Q(s, a) \\ 0, & \text{otherwise,} \end{cases}$$

we may write $\sum_{a'} \pi\left(a' \mid s'\right) Q(s', a') = \max_{a'} Q(s', a')$. If we substitute this in, we obtain the *Bellman Optimality Equation* for the action value function,

$$Q^*(s, a) = \sum_{s'} P\left(s' \mid s, a\right) \left[r(s', a, s) + \gamma \max_{a'} Q^*(s', a')\right],$$

where $Q^*(s,a)$ is the maximum action value function. Note that although this may in general be represented by a function, the simplest representation is often as a table; in the rest of this work, we only consider the specific tabular case of the algorithms, although it is possible that they may be generalized to any functional form.

Finally, we observe that $P(s' \mid s,a)$ is the transition probability of the Markov Decision Process, and may be expressed as

$$P(s' \mid s,a) = \frac{n(s,a,s')}{n(s,a)},$$

where $n(s',a,s)$ is the number of times the action a taken from state s resulted in a transition to s', and $n(s,a)$ is the number of times action a was taken from state s. It can be shown that by updating the estimate of $Q(s,a)$ a small amount at every time step using the observed transition from the previous time step, $Q(s,a)$ will converge to the expected value. Thus, we replace the transition probability with a small number α, yielding the update equation for Q-learning described by Watkins [9],

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\right].$$

Because Q-learning only depends on two states, s_t and s_{t+1}, to update the estimate of the action value, it is considered a *temporal difference* algorithm.

1.1.3 Double Q-Learning

Q-learning works well for environments that have deterministic rewards; however, if the agent receives rewards for some action that are drawn out of some distribution, it becomes significantly less effective. A variation on Q-learning described by Hasselt [10] known as Double Q-learning uses two action-value functions $Q_1(s,a)$ and $Q_2(s,a)$ in order improve the rate of learning in a stochastic environment. The key difference between Q-learning and Double Q-learning is the update rule

$$Q_A(s_t,a_t) = Q_A(s_t,a_t) + \alpha\left[r_{t+1} + Q_B\left(s_{t+1}, \arg\max_{a'} Q_A(s_{t+1},a')\right) - Q_A(s_t,a_t)\right],$$

where $P(A=1,B=2) = P(A=2,B=1) = \frac{1}{2}$. This update rule effectively decouples Q_1 and Q_2, updating each one using only half of the overall experiences that the agent encounters. The final target policy that the agent follows may be described by:

$$\pi(a \mid s) = \begin{cases} 1, & \text{if } a = \arg\max_{a'} \frac{1}{2}\left(Q_1(s,a') + Q_2(s,a')\right) \\ 0, & \text{otherwise.} \end{cases}$$

Although the idea of doubled learning was originally applied to Q-learning, it may also be applied to other algorithms. A few examples of this are Double Sarsa and Double Expected Sarsa [11]; both of these algorithms use two action value functions to improve learning in environments with stochastic rewards. In general, the result of doubling is to stabilize the estimated action values, which is difficult in such environments.

1.1.4 Multiple Q-Learning

Although Double Q-learning increases the ability of the agent to learn with a stochastic reward, even greater learning and stability may be achieved using more action-value functions. This is known as Multiple Q-learning and is described by Duryea [12]; like Double Q-learning the key difference from Q-learning is in the update rule for each action-value function. For a Multiple Q-learning algorithm with $Q_1(s,a), Q_2(s,a), \ldots, Q_N(s,a)$, the update rule for each function becomes

$$Q_A(s_t,a_t) = Q_A(s_t,a_t) + \alpha\left[r_{t+1} + \frac{1}{N-1}\sum_{i=1,i\neq A}^{N} Q_i\left(s_{t+1}, \arg\max_{a'} Q_A(s_{t+1},a')\right) - Q_A(s_t,a_t)\right],$$

where $P(A = i) = \frac{1}{N} \; \forall \, i \in [1..N]$ and α is a parameter called the learning rate that increases or decreases the rate at which the value function is updated. Like Double Q-learning, this update rule decouples the functions by only using $\frac{1}{N}$ of the experiences the agent encounters to update each function. The target policy of Multiple Q-learning is given by

$$\pi\left(a \mid s\right) = \begin{cases} 1, & \text{if } a = \arg\max_{a'} \frac{1}{N} \left(\sum_{i=1}^{N} Q_i(s, a') \right) \\ 0, & \text{otherwise.} \end{cases}$$

1.2 Quantum Computing

1.2.1 Grover's Algorithm

Recently, there has been increased interest in developing quantum computing algorithms. This is partially due to the parallel nature of quantum computing, which may be exploited to reduce the computational complexity of certain algorithms and increase the overall speed and efficiency. Two important algorithms which are the building blocks of more complicated algorithms are known as Shor's algorithm [13] and Grover's algorithm [14]. Shor's algorithm finds the prime factors of large integers in $\mathcal{O}(\log \log N)$ time, and is an important algorithm in cryptography. However, in this thesis we only consider the application of Grover's algorithm to reinforcement learning.

Grover's algorithm is a well-known search algorithm in quantum computing that can find an item in $\mathcal{O}(\sqrt{N})$ instead of $\mathcal{O}(N)$ (which is the runtime of classical algorithms). The basic concept of Grover's algorithm is to increase the probability that a given quantum-mechanical system, when measured, will yield the correct answer, which is determined by an oracle function $f(x) = 0$. While originally proposed as a search algorithm, Grover's algorithm may be used more generally as a method to increase the probability of measuring any state, making it a useful process that may be incorporated in other algorithms.

1.2.2 Quantum Reinforcement Learning (VQRL)

With the increase in interest in quantum computing has come interest in applying it to reinforcement learning. As the application of reinforcement learning to real-world problems generally requires a very large state-space, the hope is that the application of quantum computing would significantly reduce the amount of time for the algorithm to reach convergence.

Quantum Reinforcement Learning [15] is one such algorithm that applies quantum computing to reinforcement learning. The basic idea is to store the policy as a superposition of actions so that quantum computing algorithms may be applied to gain the quantum speedup. The policy that the agent follows is given by

$$\pi\left(a \mid s\right) = \left|\langle a | a_s \rangle\right|^2,$$

where a is an action taken from state s, $|a\rangle$ is its associated eigenstate, and $|a_s\rangle$ is the state of a quantum system that represents the policy for state s. The state of these systems are updated according to the rule

$$|a_s(t+1)\rangle = \hat{U}_g \, |a_s(t)\rangle \, ,$$

where $\hat{U}_g$ is a unitary operator that represents one Grover iteration. This may be expressed as a combination of a reflection and diffusion, $\hat{U}_a$ and $\hat{U}_{a_s}$ [15], respectively, given by

$$\hat{U}_g = \hat{U}_{a_s} \hat{U}_a.$$

The effect of $\hat{U}_a$ is to invert the amplitude of $\hat{a}$, and the effect of $\hat{U}_{a_s}$ is to invert all the amplitudes of the state about their mean. When applied an equiprobable state, applying $\hat{U}_g$ increases the probability of obtaining a from a measurement of the state. However, $\hat{U}_g$ may not be applied indefinitely; after a

certain number of iterations, Grover's algorithm tends to *decrease* the probability of measuring a. The maximum number of iterations depends on the number of possible actions, and is given by

$$L_{max} = \text{int}\left(\frac{\pi}{4\theta} - \frac{1}{2}\right),$$

where L_{max} is the maximum number of iterations that may be applied until the probability decreases and $\tan\theta = |\langle a_s|a\rangle|$.

1.3 Quantum Q-learning and Multiple $V(s)$ or $Q(s,a)$ Functions

In this research, a collection of new quantum reinforcement learning algorithms are introduced which are based on Quantum Reinforcement Learning (VQRL), which was first described by Dong, et al. [15]. These quantum algorithms store the policy as a superposition of qubits, and use Grover's algorithm to update the probability amplitudes corresponding to different actions in a given state. The first modification made to the VQRL is to replace $V(S)$ with $Q(s,a)$, introducing a new algorithm known as Quantum Q-learning (QQRL). It was found in test experiments that introducing QQRL converged faster on average than VQRL, which is likely due to the extra precision in action selection which $Q(s,a)$ provides over $V(s)$. Both VQRL and QQRL exhibit much faster convergence than their classical counterpart, Q-learning, which is likely due to the balance of exploration and exploitation provided by the quantum nature of the policy.

Another alteration done to VQRL, and also done to QQRL, is to increase the number of $V(s)$ and $Q(s,a)$ functions, similar to to the Multiple Q-learning algorithm described by Duryea, et al. [12]. These algorithms are known as Multiple Quantum Reinforcement Learning (MVQRL) and Multiple Quantum Q-learning (MQQRL). In general, it was found that increasing the number of $V(s)$ or $Q(s,a)$ functions increased the performance of the algorithms in a stochastic environment, where the reward is sampled from some distribution of values instead of a single value.

2 Algorithms

2.1 Double Quantum Reinforcement Learning (DVQRL)

Double Quantum Reinforcement Learning (DVQRL) combines the idea of doubled learning (such as used in Double Q-learning) with Quantum Reinforcement Learning. The main idea of DVQRL is to use two separate value functions, $V_1(s)$ and $V_2(s)$, and with probability $P(A = 1, B = 2) = P(A = 2, B = 1) = \frac{1}{2}$ update either $V_1(s)$ or $V_2(s)$ according to

$$V_A(s) \leftarrow V_A(s) + \alpha \left(r + V_B(s') - V_A(s)\right).$$

The expected value of each state is then the average of the two value functions $V(s) = \frac{1}{2}\left(V_1(s) + V_2(s)\right)$. This is used to compute L, which is the number of times to apply Grover's operator to the policy, $|a_s\rangle$. First, θ is computed by solving

$$\tan\theta = |\langle a_s|a\rangle|.$$

Then, after solving for θ, L may be computed using

$$L = \min\left\{k\left(r + V(s')\right), \text{int}\left(\frac{\pi}{4\theta} - \frac{1}{2}\right)\right\},$$

where L is the number of times to apply Grover's operator and k is a parameter which controls the rate at which the policy is updated. The Grover operator $\hat{U}_g$ may then be applied L times to the current policy, given by

$$|a_s(t+1)\rangle = \hat{U}_g^L |a_s(t)\rangle.$$

The algorithm for DVQRL can be seen in Table 1.

Table 1: Double Quantum Reinforcement Learning (DVQRL) .

	Procedure
1.	Initialize $V_1(s), V_2(s), \|a_s\rangle$.
2.	**loop** for each episode {
3.	**for each** s in S {
4.	Observe a from $\|a_s\rangle$.
5.	Take action a , observe next state s' and reward r .
6.	**with** probability $P(A = 1, B = 2) = P(A = 2, B = 1) = \frac{1}{2}$ {
7.	Update $V_A(s)$ according to $V_A(s) \leftarrow V_A(s) + \alpha\left(r + \gamma V_B(s') - V_A(s)\right)$.
	}
8.	Compute $V(s') = \frac{1}{2}\left(V_1(s') + V_2(s')\right)$.
9.	Compute $L = \min\left\{k(r + V(s')), \text{int}(\frac{\pi}{4\theta} - \frac{1}{2})\right\}$.
10.	Update $\|a_s\rangle$ according to $\|a_s\rangle = \hat{U}_g^L \|a_s\rangle$.
	}
	}

2.2 Multiple Quantum Reinforcement Learning (MVQRL)

Multiple Quantum Reinforcement Learning (MVQRL) is similar to Double Quantum Reinforcement Learning, but allows for N action value functions instead of only 2. Similar to Multiple Q-learning, the effect of increasing the number of functions is an improvement in learning in stochastic environments. The estimated value of each state is stored in the functions $V_1(s), V_2(s), \ldots, V_N(s)$. The value

functions are updated at each step according to

$$V_A(s) \leftarrow V_A(s) + \alpha \left(r + \frac{\gamma}{N-1} \sum_{i=1, i \neq A}^{N} V_i(s') - V_A(s) \right),$$

where $P(A = i) = \frac{1}{N} \; \forall \, i \in [1..N]$. The algorithm for Multiple VQRL may be seen in Table 2.

Table 2: Multiple Quantum Reinforcement Learning (MVQRL)

	Procedure
1.	Initialize $V_1(s), V_2(s), \ldots, V_N(s), \lvert a_s \rangle$.
2.	**loop** for each episode {
3.	**for each** s in S {
4.	Observe a from $\lvert a_s \rangle$.
5.	Take action a , observe next state s' and reward r .
6.	**with** probability $P(A = i) = \frac{1}{N}$ {
7.	Update $V_A(s) \leftarrow V_A(s) + \alpha \left(r + \frac{\gamma}{N} \sum_{i=1, i \neq A}^{N} V_i(s') - V_A(s) \right)$.
	}
8.	Compute $V_{s'} = \frac{1}{N} \sum_{i=1}^{N} V_i(s')$.
9.	Compute $L = \min \left\{ k(r + V_{s'}), \text{int} \left(\frac{\pi}{4\theta} - \frac{1}{2} \right) \right\}$.
10.	Update $\lvert a_s \rangle$ according to $\lvert a_s \rangle = \hat{U}_g^L \lvert a_s \rangle$.
	}
	}

2.3 Quantum Q-Learning (QQRL)

The idea of Quantum Reinforcement Learning may also be adapted to utilize an action value function instead of a value function. The advantage of this is that the action value function holds more specific information than the value function, potentially leading to faster convergence. Quantum Q Reinforcement Learning (QQRL), which adapts VQRL, has an action value function that replaces the value function in VQRL. This action value function is used to compute the expected value of the next state according to

$$V_{s'} = \max_{a'} Q(s', a'),$$

where $Q(s, a)$ is the expected value of taking action a from state s, and $V_{s'}$ is the expected value of the agent being in state s' under the current policy. Like VQRL, in QQRL the policy $\lvert a_s \rangle$ is updated according to

$$\lvert a_s(t+1) \rangle = \hat{U}_g^L \lvert a_s(t) \rangle .$$

The full algorithm describing QQRL is shown in Table 3.

Table 3: Quantum Q-learning (QQRL)

	Procedure
1.	Initialize $Q(s, a), \lvert a_s \rangle$.
2.	**loop** for each episode {
3.	**for each** s in S {
4.	Observe a from $\lvert a_s \rangle$.
5.	Take action a , observe next state s' and reward r .
6.	Compute $V_{s'} = \max_{a'} Q(s', a')$.
7.	Compute $L = \min \left\{ k(r + V_{s'}), \text{int}(\frac{\pi}{4\theta} - \frac{1}{2}) \right\}$.

	Procedure
8.	Update $\|a_s\rangle \leftarrow \hat{U}_g^L \|a_s\rangle$.
9.	Update $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma V_{s'} - Q(s,a))$.
	}
	}

2.4 Double Quantum Q-Learning (DQQRL)

Similar to VQRL and Double Q-learning, QQRL may be doubled to use two different action value functions, $Q_1(s,a)$ and $Q_2(s,a)$. This algorithm is described in Table 4. At each time step, only one function in the algorithm is updated at a time; this is done by choosing indices A and B with probability $P(A=1,B=2) = P(A=2,B=1) = \frac{1}{2}$. The update equation then becomes

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha\left(r + \gamma Q_B(s', \arg\max_{a'} Q_A(s',a')) - Q(s,a)\right).$$

When computing L to determine the number of Grover iterations to be performed on the policy, $V(s)$ is computed according to

$$V(s) = \max_a \frac{1}{2}\left(Q_1(s,a) + Q_2(s,a)\right).$$

The policy is then updated in the same way as in QQRL.

Table 4: Double Quantum Q-learning (DQQRL) algorithm.

	Procedure
1.	Initialize $Q_1(s,a), Q_2(s,a), \|a_s\rangle$.
2.	**loop** for each episode {
3.	**for each** s in S {
4.	Observe a from $\|a_s\rangle$.
5.	Take action a , observe next state s' and reward r .
6.	Compute $V_{s'} = \max_{a'} \frac{1}{2}\left(Q_1(s',a') + Q_2(s',a')\right)$.
7.	Compute $L = \min\left\{k(r+V_{s'}), \text{int}(\frac{\pi}{4\theta} - \frac{1}{2})\right\}$.
8.	Update $\|a_s\rangle \leftarrow \hat{U}_g^L \|a_s\rangle$.
9.	**with** probability $P(A=1,B=2) = P(A=2,B=1) = \frac{1}{2}$ {
10.	Update $Q_A(s,a) \leftarrow Q_A(s,a) + \alpha\left(r + \gamma Q_B(s', \arg\max_{a'} Q_A(s',a')) - Q(s,a)\right)$.
	}
	}
	}

2.5 Multiple Quantum Q-Learning (MQQRL)

QQRL may also be extended to have any number of action value functions; this is done in a similar way to Multiple Q-learning and MVQRL. The algorithm for Multiple Quantum Q-learning (MQQRL) may be seen in Table 5. At each time step, a single function is chosen to be updated; this is done by choosing index A with probability $P(A=i)\,\forall\, i \in [1..N]$. Then, $Q_A(s,a)$ is updated according to

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha\left(r + \frac{\gamma}{N-1}\sum_{i=1, i\neq A}^{N} Q_i(s',b) - Q(s,a)\right),$$

where $b = \arg\max_{a'} Q_i(s', a')$. In order to compute the number of Grover iterations, L, $V(s)$ is computed as the maximum average of $Q_i(s, a)$:

$$V(s) = \max_a \frac{1}{N} \sum_{i=1}^{N} Q_i(s, a).$$

The probability amplitudes of the policy are then updated in the same way as in QQRL.

Table 5: Multiple Quantum Q-learning (MQQRL).

	Multiple Quantum Q Reinforcement Learning (MQQRL)
1.	Initialize $Q_1(s, a), Q_2(s, a), \ldots Q_N(s, a), \|a_s\rangle$.
2.	**loop** for each episode {
3.	**for each** s in S {
4.	Observe a from $\|a_s\rangle$.
5.	Take action a , observe next state s' and reward r .
9.	**with** probability $P(A = i) = \frac{1}{N} \forall\, i \in [1..N]$ {
6.	Compute $V_{s'} = \max_{a'} \frac{1}{N} \sum_{i=1}^{N} Q_i(s', a')$.
7.	Compute $L = \min\left\{k(r + V_{s'}), \text{int}(\frac{\pi}{4\theta} - \frac{1}{2})\right\}$.
8.	Update $\|a_s\rangle \leftarrow \hat{U}_g^L \|a_s\rangle$.
9.	Choose $b = \arg\max_{a'} Q_i(s', a')$.
10.	Update $Q_A(s, a) \leftarrow Q_A(s, a) + \alpha \left(r + \frac{\gamma}{N-1} \sum_{i=1, i\neq A}^{N} Q_i(s', b) - Q(s, a)\right)$.
	}
	}
	}

3 Results

3.1 Test Environment and Optimal Paths

The environment used to test the algorithms can be seen in Figure 1. At each time step, the agent may move between adjacent states through the actions *up*, *down*, *left*, and *right*. The two optimal paths through the environment are shown in Figure 2. This is the environment used by Brandon [16].

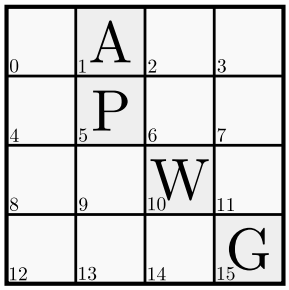

Figure 1: Environment used to test each algorithm. A represents the starting location of the agent in the environment, P represents the pit , where the agent receives a large negative reward, W represents the wall , which is a disallowed state, and G represents the goal , where the agent receives a large positive reward. At each time step, the agent receives a small negative reward so that the optimal policy is the shortest path through the environment from the initial state to the goal.

In order to denote paths through the environment, an action sequence is used. This is a string of action numbers; the mapping from number to action may be seen in Table 6. For example, a path might be denoted by "31210;" this represents the movements right, down, left, down, and up, in sequence. Note that an action may not change the state; if the agent is in its initial position and follows the path "000," it will remain in the same position.

Table 6: Numerical labels of each action. A string of these actions such as 32132 indicates a path through the environment.

Number	Action
0	Up
1	Down
2	Left
3	Right

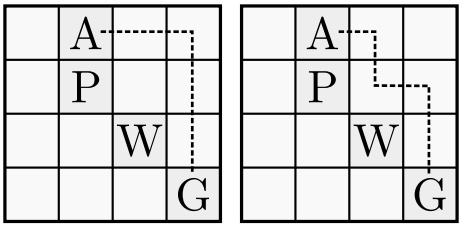

Figure 2: Optimal and sub-optimal paths through the environment. The optimal path (left) corresponds to an action sequence of 33111, while the sub-optimal path (right) corresponds to an action sequence of 31311. Although both paths have the same number of steps, the sub-optimal path is closer to the pit ; for stochastic policies, this means that there is an increased probability that the agent will take an action that moves into this state.

The "pit" and the "goal" are terminal states; that is, when the agent enters these states, the episode is finished. When the agent enters the "pit," it receives a reward of $r = -10$, and when it enters the "goal," it receives a reward of $r = +10$. When it enters any of the other states, it receives a reward with mean $\bar{r} = -1$ and standard deviation σ. In a deterministic environment, $\sigma = 0$; in a stochastic environment, $\sigma \neq 0$, and the agent receives the rewards $-1 + \sigma$ and $-1 - \sigma$ with equal probability.

Due to the stochastic nature of the quantum algorithms, the results for convergence were averaged over multiple runs for each experiment. The number of repeated runs was different for each experiment; these are given in the text and figure captions. In each repeated run, the only difference between the algorithms was the seed value for random number generation. In other words, the position of each feature was the same as in Figure 1 in every run.

3.2 Learning Curves for Single, Double, and Multiple Value Function Algorithms

One of the most important comparisons between reinforcement learning algorithms is the relative number of episodes until each algorithm converges. It is often the case that an algorithm with faster convergence will be preferred to one with slow convergence. Figure 3 shows the learning curves for Q-learning, VQRL, and QQRL, averaged over 100 different runs. In these graphs, it is clear that the quantum algorithms perform significantly better the classical Q-learning. Furthermore, it may be observed that QQRL converges about 25% faster than VQRL, which is a significant increase in speed.

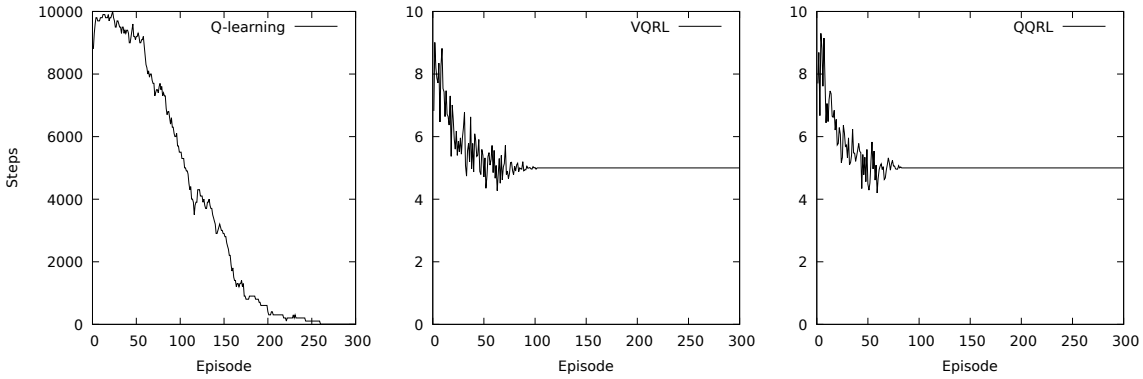

Figure 3: Learning curves for Q-learning, VQRL, and QQRL. These compare the number of steps per episode for each of the three algorithms as they are trained. Note that the Steps scale for Q-learning is different than for VQRL and QQRL. For this experiment, $\alpha = 0.05$. The results were averaged over 100 experiments.

While it is important that the algorithms show good convergence properties in a deterministic environment, it is also import that the algorithms show good convergence properties when rewards are stochastic. Figure 4 shows the learning curves for the single, double, and multiple value function algorithms in a stochastic environment with $\sigma = 1$. These figures show that, even in a stochastic environment, the quantum algorithms still converge in about the same number of iterations as in a deterministic environment. This shows that the quantum algorithms are robust even in a stochastic environment.

3.3 Convergence of Q-learning, VQRL, and QQRL vs. Learning Rate

However, the convergence time is highly dependent on the learning rate, α. Figure 5 shows the number of episodes to convergence for Q-learning, VQRL, and QQRL. As may be seen, Q-learning is highly dependent on the learning rate; until it begins to fail, the number of episodes for Q-learning

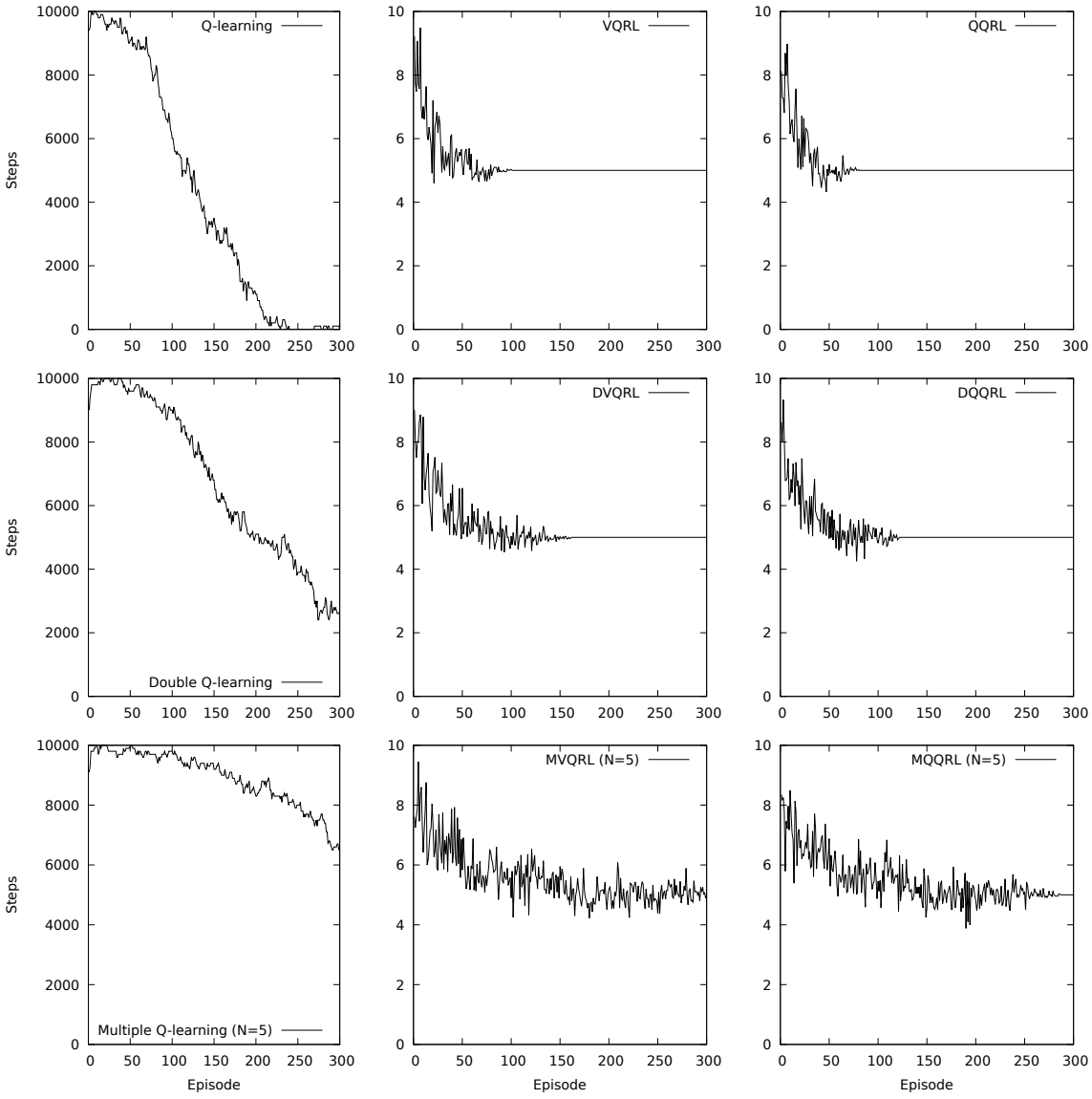

Figure 4: Comparison of the learning curves for single, double, and multiple value function versions of Q-learning, VQRL, and QQRL in a stochastic environment, where $\sigma = 1$. For this experiment, $\alpha = 0.05$. The results were averaged over 100 runs.

steadily decreases; in VQRL and QQRL, the number of episodes is not highly related to the learning rate. As the learning rate approaches a threshold, however, the all three algorithms begin to fail. In Q-learning, this has a relatively sudden effect, but in VQRL and QQRL it is more gradual, increasing the likelihood of failure. Q-learning was found to fail at $\alpha = 1.1$, VQRL at $\alpha = 1.2$, and QQRL at $\alpha = 1.8$. The failure point was found by determining the value of α past which $N > 1000$ for the algorithm.

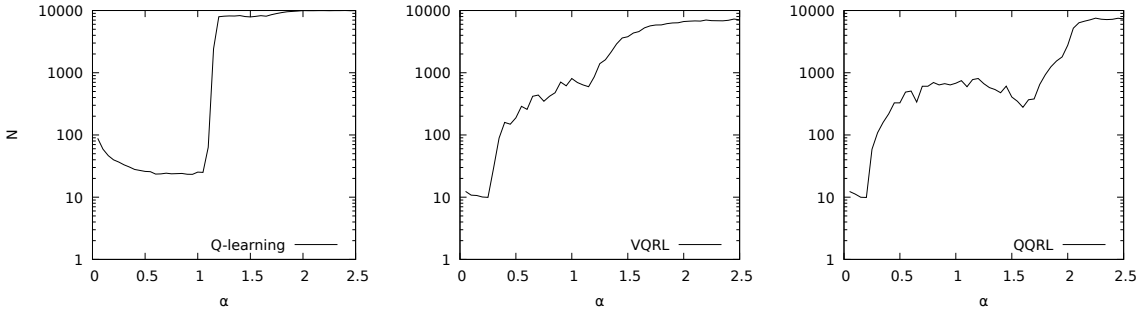

Figure 5: Number of iterations until Q-learning, Quantum Reinforcement Learning (VQRL), and Quantum Q Reinforcement Learning (QQRL) reached convergence, as a function of the learning rate. The failure point of Q-learning was found to be $\alpha = 1.1$, for VQRL $\alpha = 1.2$, and for QQRL $\alpha = 1.8$. The results were averaged over 1000 runs, and were plotted using a log scale for N .

One important difference to note in Figure 5 is the difference between VQRL and QQRL as the learning rate is increased. QQRL begins to fail much sooner than VQRL, but it appears to maintain this failure over a wider range of learning rates. In other words, while QQRL begins to fail for lower values of α, it maintains a lower level of failure for a longer range, on the interval $0.5 < \alpha < 1.5$, before beginning to fail. Thus, while VQRL may fail at a larger learning rate than QQRL, QQRL may be considered more robust with respect to the learning rate as it maintains partial failure over a larger range of learning rates than VQRL.

3.4 Convergence of Multiple Q-learning, MVQRL, and MQQRL vs. Learning Rate

One of the most important differences between the single value function algorithms (Q-learning, VQRL, and QQRL) with the corresponding multiple value function algorithms (Multiple Q-learning, MVQRL, and MQQRL) is an increase in stability with the addition of value functions. This is most clearly seen in Figure 6, which plots the breakdown learning rate against the number of value functions for algorithm. As the number of functions increases, all three algorithms become more robust, allowing higher learning rates to be used. One reason for this is that each function is constructed using only a fraction of the overall experiences. Combining these functions then results in an estimate of the value or the action value which is less sensitive than when only a single function is used.

The breakdown learning rate was defined in the same manner as above for the single value function algorithms; in other words, the value of α past which $N > 1000$. This was computed for Multiple Q-learning, MVQRL, and MQQRL for each value of $N \in [1..10]$ and is shown in Figure 6. From this, it can be seen that for each of the three algorithms, increasing the number of value functions (either $Q(s, a)$ or $V(s)$) increases the learning rate at which the algorithms break down; in other words, additional value functions increase the stability of the policy.

Another feature of Figure 6 to note is that MQQRL consistently has a higher breakdown learning rate than Multiple Q-learning and MVQRL. This indicates that, for the same N, MQQRL is more robust to a higher learning rate. One advantage of this is that a higher learning rate may be set

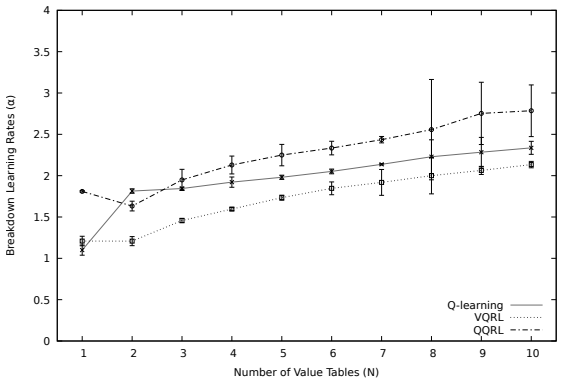

Figure 6: Breakdown learning rate as function of the number of $Q(s, a)$ functions. The breakdown learning rate is the minimum value of α where the algorithm fails to find the goal. Note that $N = 1$ corresponds to Q-learning, QQRL, and VQRL, and that $N = 2$ corresponds to Double Q-learning, DQQRL, and DVQRL.

with MQQRL than MVQRL, which generally increases the speed of learning. Furthermore, both Multiple Q-learning and MQQRL have higher breakdown learning rates for $N > 1$, which indicates that $Q(s, a)$ is a more robust against higher learning rates than $V(s)$.

3.5 Convergence of VQRL and QQRL vs. k

Similar to the learning rate, the parameter k also has a significant effect on the convergence of the quantum algorithms; effectively, it determines how quickly the policy changes. A larger k results in a faster change in the policy, which generally means that the learning rate of the algorithm increases. This may be seen in Figure 7.

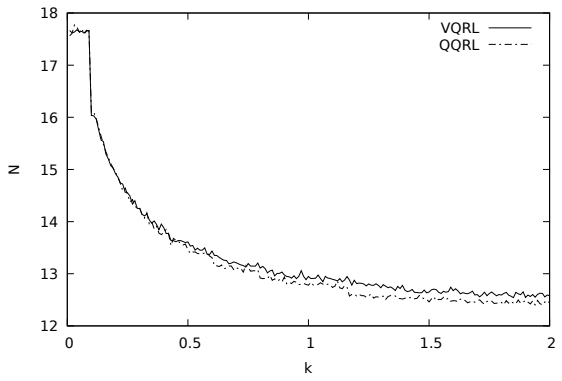

Figure 7: Iterations to convergence as a function of k in a deterministic environment. The results were averaged over 100000 runs. For this experiment, $\alpha = 0.05$.

The relationship between k and the speed of convergence of VQRL and QQRL is interesting because, above a certain threshold, the number of episodes until convergence asymptotes to about 13. This indicates that the algorithms are not sensitive to the particular value of k, given that it is sufficiently high, which suggests that a reasonable strategy for the selection of a particular k value may be to increase the value until the number of iterations to convergence does not change anymore.

3.6 Branching Ratios of Optimal and Sub-Optimal Paths

While the speed of convergence is an important characteristic when comparing reinforcement learning algorithms, it is also important to consider the quality of the policies the algorithms generate and the paths through the environment that they produce. An interesting comparison between Q-learning, VQRL and QQRL is the relative path distribution between each of the algorithms. Ideally, each of the algorithms will converge over time to the optimal path, which is often the shortest path. The branching ratio, which is the fraction of times that the algorithm converged to a certain path, can be seen as a function of episode number in Figure 8.

While Q-learning converges slower than VQRL and QQRL, over time it converges to the optimal path instead of the sub-optimal path. However, when VQRL and QQRL converge to the sub-optimal path, their policies have become essentially deterministic; consequently, there is little difference between the optimal path and the sub-optimal path as both have are the same length. The only reason why the sub-optimal path is less ideal than the optimal one is because it is closer to the "pit," so an agent with a stochastic policy has a higher probability of entering the "pit."

The reason for why the policies of VQRL and QQRL become deterministic over time may be attributed to repeated applications of Grover's algorithm to the policies. In these algorithms, Grover iterations have the tendency to increase the probability of actions with higher expected returns and decrease actions with lower ones. As the value functions of these algorithms converge, repeated Grover iterations tend to increase the probability of the most favorable action—the action with the highest expected return—and decrease the probability of all of the other actions. In the limit, the probability of the action with the maximum expected return approaches 1, and the probabilities of all other actions approach 0. In effect, the probability becomes deterministic in the limit.

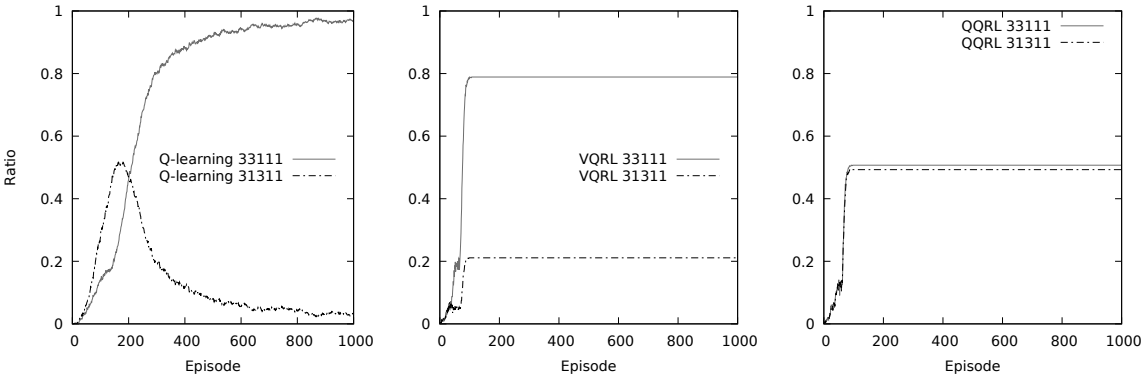

Figure 8: Branching ratio of Q-learning, VQRL, and QQRL. The optimal path 33111 and the suboptimal path 31311 are shown in Figure 2 . For this experiment, $\alpha = 0.05$. The results were averaged over 1000 runs.

While it is clear in Figure 8 that VQRL and QQRL converge much faster than Q-learning, it is also apparent that Q-learning tends to converge to the optimal path over time, while the quantum algorithms converge to a distribution between the optimal and sub-optimal paths. This highlights one trade off between the quantum algorithms and their classical counterparts: while the quantum algorithms converge much faster to a less optimal path distribution, Q-learning eventually converges to the optimal path.

3.7 Comparison of Value Functions for VQRL, QQRL, and Q-learning

An important difference between Q-learning, VQRL, and QQRL is the estimate of the value of each of the states; these are shown in Figure 9 for the initial state of the agent as a function of the number of episodes. In the experiment shown, VQRL and QQRL converge much faster to the expected value

of the state. As the stability of the value function is highly related to the stability of the policy, this indicates that the policies of VQRL and QQRL converge much quicker than Q-learning. An interesting distinction between VQRL and QQRL is that VQRL converges to a much higher value than QQRL. While the cause of this is uncertain, it is likely related to the path distribution that the algorithm converges to (see Figure 8).

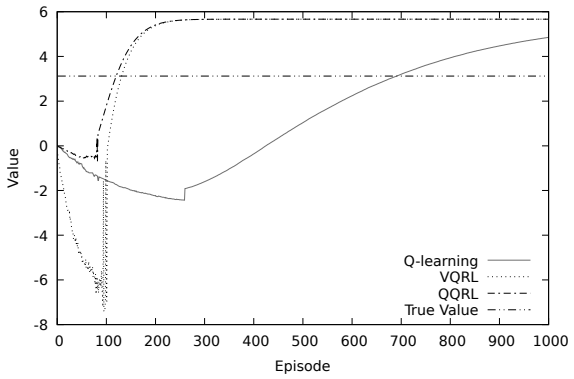

Figure 9: Expected value of initial state for Q-learning, VQRL, and QQRL. For VQRL, $V(s)$ is shown, but for QQRL and Q-learning $V_{max}(s) = \arg\max_a Q(s,a)$ is shown for comparison. The true value (V(s) = 3.122) is also shown for comparison. For this experiment, $\alpha = 0.05$; the results were averaged over 1000 runs.

In addition to the maximum value of the state, Q-learning and QQRL exhibit further differences in the value of each action for the state. This is shown in Figure 10 for the initial state. An interesting observation is that for the first 100 episodes, $Q(s,a)$ for Q-learning and QQRL follow the same downward trend for each a. After episode 100, however, the values diverge; QQRL converges much faster than Q-learning, which does not converge in the first 1000 episodes.

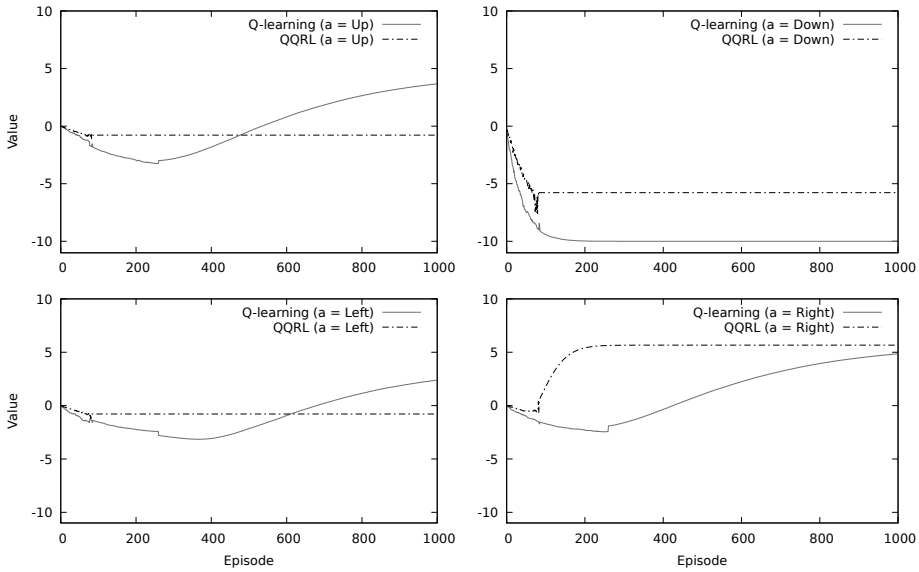

Figure 10: Comparison of $Q(s,a)$ for Q-learning and QQRL for each action. Clockwise from top left, the actions shown are Up , Down , Left , and Right . For this experiment, $\alpha = 0.05$; the results were averaged over 1000 runs.

3.8 Comparison of Value Functions for Double Q-learning, DVQRL, and DQQRL

One of the effects of doubling the value functions of reinforcement learning algorithms is that the expected value of each state is different for each state. Generally, this difference is an underestimate in comparison with the estimate of the value in the single function algorithm; this may be observed in Figure 11. However, despite this initial underestimate at any given episode, the shape of the double function algorithms generally follows the shape of the single function algorithms, but at a slower rate.

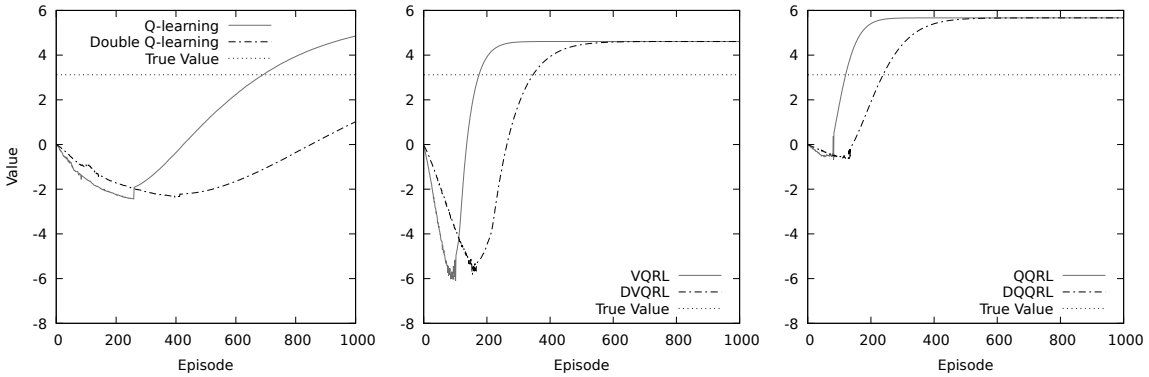

Figure 11: Comparison of $V(s)$ between single function algorithms (Q-learning, VQRL, QQRL) with corresponding double function algorithms (Double Q-learning, DVQRL, DQQRL) for the initial state. The true value (V(s) = 3.122) is shown for comparison. For Q-learning, Double Q-learning, QQRL, and DQQRL, $V(s)$ was computed according to $V_{max}(s) = \max_a Q(s, a)$.

3.9 Comparison of Value Functions for Multiple Q-learning, MVQRL, and MQQRL

Like Double Q-learning, DVQRL, and DQQRL, Multiple Q-learning, MQVRL, and MQQRL also exhibit underestimation of $Q(s, a)$ and $V(s, a)$ for the single versions of the algorithms (Q-learning, VQRL, and QQRL). This is shown for $N = 3, 6, 10$ in Figure 12. An interesting feature of MQQRL graph is that, as the number of $Q(s, a)$ functions increases, the initial amount of underestimate of the value exhibited by MQQRL increases. Furthermore, as may be seen for $N = 3$ and implied for $N = 6$, the value which MQQRL converges to is greater than QQRL. Neither of these behaviors are exhibited by MVQRL, which essentially has the same behavior as VQRL but at a slower rate.

3.10 Summary of Results

The results in Section 3.3 demonstrate that QQRL reaches convergence in fewer episodes than VQRL. QQRL and VQRL only differ in how the value of the current state is stored; that is, QQRL uses $Q(s, a)$ while VQRL only uses $V(s)$. Likely, the reason for such improvement in convergence speed is due to the extra precision provided by $Q(s, a)$ when computing the value of a certain action and then choosing the best action; in contrast, $V(s)$ is the expectation of the value of all possible actions, and as a consequence is much less precise. The extra precision is especially important when updating the policy in QQRL as it uses $\max_a Q(s, a)$ to compute the number of Grover iterations to perform.

Another important observation is that the quantum algorithms, VQRL, QQRL, MVQRL, and MQQRL, have much faster convergence than their classical counterparts, Q-learning and Multiple Q-learning. As discussed by Dong, et al. [15], this is likely the quantum algorithms strike a better balance between exploration and exploitation in the way the policies are updated, at least as compared

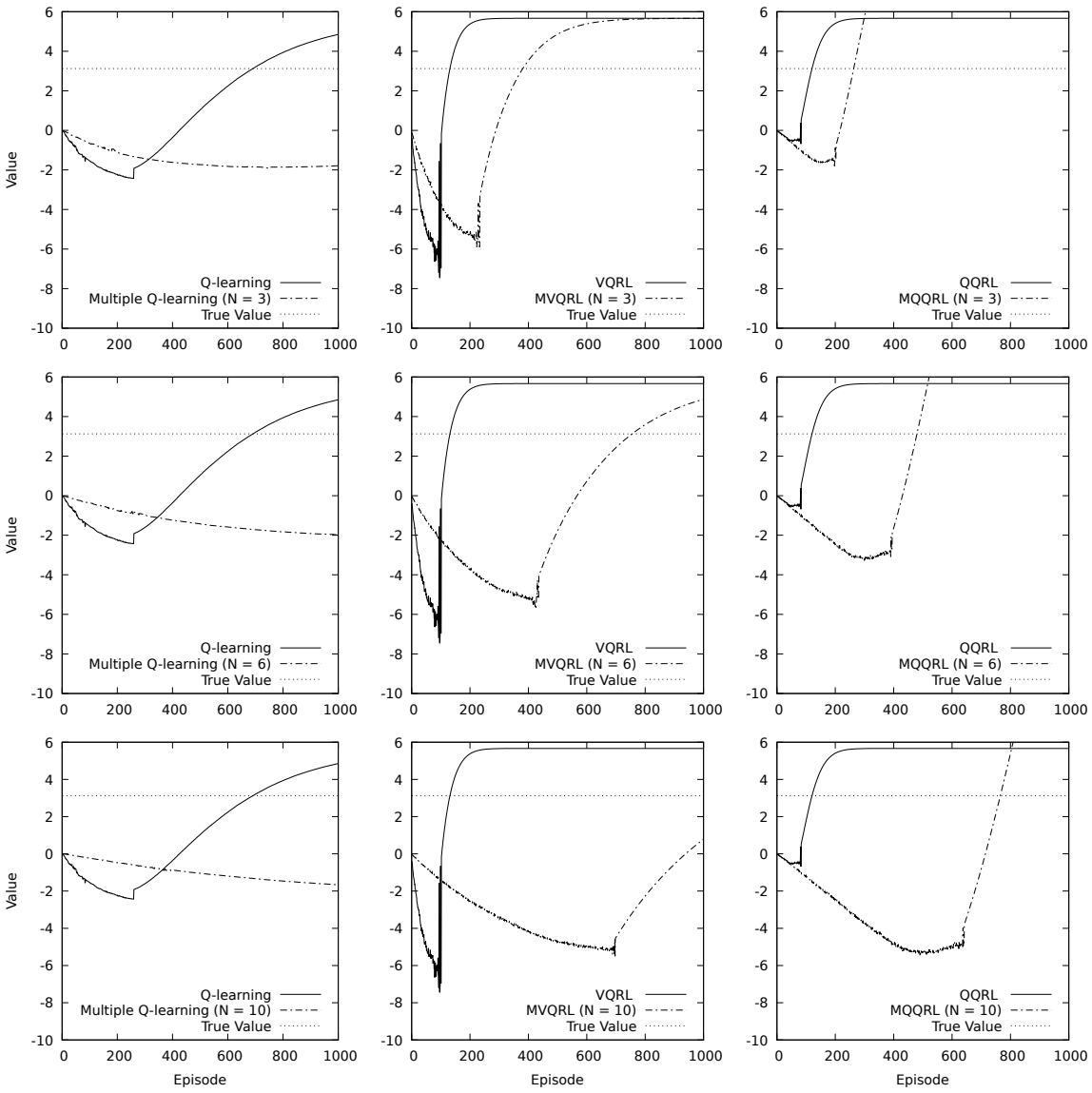

Figure 12: Comparison of $V(s)$ between single function algorithms (Q-learning, VQRL, QQRL) with corresponding multiple function algorithms (Multiple Q-learning, MVQRL, MQQRL) for the initial state. The values of $N = 3$, $N = 6$, and $N = 10$ were chosen to sample the effect of increasing N . For Q-learning, Multiple Q-learning, QQRL, and MQQRL, $V(s)$ was computed according to $V_{max}(s) = \max_a Q(s, a)$. The top row shows the case where $N = 3$, the middle row $N = 6$, and the bottom row $N = 10$. In each row, the first graph compares Q-learning with Multiple Q-learning, the second graph compares VQRL with MVQRL, and the third graph compares QQRL with MQQRL. The true value (V(s) = 3.122) is shown for comparison.

to the classical algorithms. This is because the quantum algorithms store the policy as a superposition of all possible algorithms, and use Grover's algorithm to increase the probability of taking an action.

Finally, it was found that adding extra $V(s)$ and $Q(s,a)$ functions improved the learning in a stochastic environment. In other words, when the reward was drawn from some distribution, MVQRL and MQQRL exhibited better performance than VQRL and QQRL, respectively, in the same way that Multiple Q-learning exhibited better performance than Q-learning [12]. This is because each $V_i(s)$ or $Q_i(s,a)$ function is constructed using only $\frac{1}{N}$ of the total number of experiences, and the variations among the function are reduced when computing the average $V(s)$ or $Q(s,a)$.

4 Discussion

4.1 Convergence of Quantum and Classical Algorithms

One comparison which may be drawn between the quantum and classical algorithms is the number of episodes to convergence. Generally, it was observed that the value functions $V(s)$ and $Q(s,a)$ in VQRL and QQRL, respectively, converge much faster than the $Q(s,a)$ in Q-learning. This result leads to significantly faster convergence times for the same learning rate, as may be seen in Figure 5; for smaller learning rates, the quantum algorithms converge about 10 times faster.

A particular feature of QQRL is that it has a higher breakdown learning rate than either Q-learning or VQRL; this generalizes to Multiple Q-learning, MQVRL, and MQQRL. QQRL not only converges faster than Q-learning for a *certain* learning rate, but also allows for higher learning rates than Q-learning does. This is an interesting phenomenon which indicates that QQRL gives faster convergence in general than Q-learning.

4.2 Comparison of QQRL with VQRL

One of the main differences between VQRL and QQRL is the convergent path distribution for each algorithm (see Figure 8). While QQRL converges in 25% fewer episodes than VQRL, it converges to a path distribution that equally weights the optimal and sub-optimal paths. However, both paths are the same length; the sub-optimal path is only considered such because an agent with a stochastic policy is more likely to move into the "pit" (and receive a large negative reward). Thus, the fact that QQRL converges to an equal path distribution indicates that the policy is highly deterministic, in which case there is no advantage of one path over the other.

4.3 Final Remarks

In this paper, we introduced multiple new quantum reinforcement learning algorithms–QQRL, DVQRL, DQQRL, MVQRL, and MQQRL–compared them with the corresponding classical algorithms–Q-learning, Double Q-learning, and Multiple Q-learning–as well as compared them to each other. It was found that QQRL converged faster than VQRL, due to the increase in precision of $Q(s,a)$, and that both quantum algorithm converged much faster than Q-learning. Furthermore, QQRL was found to be more robust to higher learning rates than VQRL. This was found to generalize to MQQRL and MVQRL, as adding extra value functions to both algorithms increased the breakdown learning rate for the algorithms.

In addition, increasing the number of $V(s)$ or $Q(s,a)$ functions in MVQRL and MQQRL tended to improve the stability of the algorithms in stochastic environments. This is similar to the fashion in which additional functions in Multiple Q-learning tended to improve the stability. Each additional function decreases the number of experiences used to train each individual function. Averaging over these then produces a better estimate of the value or action value than each individual function.

References

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] J. W. Lee, "Stock price prediction using reinforcement learning," in *Industrial electronics, 2001. proceedings. isie 2001. IEEE international symposium on*, 2001, vol. 1, pp. 690–695.

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

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning." in *AAAI*, 2016, pp. 2094–2100.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and others, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] J. Tromp, "Number of legal Go positions." 2016 [Online]. Available: http://tromp.github.io/go/legal.html

[7] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, nos. 3-4, pp. 229–256, 1992.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT Press Cambridge, 1998.

[9] C. J. C. H. Watkins, "Learning from delayed rewards," PhD thesis, University of Cambridge England, 1989.

[10] H. V. Hasselt, "Double q-learning," in *Advances in neural information processing systems*, 2010, pp. 2613–2621.

[11] M. Ganger, E. Duryea, and W. Hu, "Double sarsa and double expected sarsa with shallow and deep learning," *Journal of Data Analysis and Information Processing*, vol. 4, no. 04, p. 159, 2016.

[12] E. Duryea, M. Ganger, and W. Hu, "Exploring deep reinforcement learning with multi q-learning," *Intelligent Control and Automation*, vol. 7, no. 04, p. 129, 2016.

[13] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Foundations of computer science, 1994 proceedings., 35th annual symposium on*, 1994, pp. 124–134.

[14] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on theory of computing*, 1996, pp. 212–219.

[15] 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.

[16] Brandon, "Q-learning with neural networks." 2015 [Online]. Available: http://outlace.com/Reinforcement-Learning-Part-3/