http://www.scirp.org/journal/ns Natural Science, 2018, Vol. 10, (No. 1), pp: 45-58

Empirical Analysis of Decision Making of an AI Agent on IBM's 5Q Quantum Computer

Wei Hu

Department of Computer Science, Houghton College, Houghton, NY, USA

Correspondence to: Wei Hu, Wei.hu@houghton.edu

Keywords: Quantum Computation, Quantum Machine Learning, Quantum Reinforcement Learning, Quantum Circuit

Received: January 6, 2018 **Accepted:** January 27, 2018 **Published:** January 30, 2018

ABSTRACT

A recent work has shown that using an ion trap quantum processor can speed up the decision making of a reinforcement learning agent. Its quantum advantage is observed when the external environment changes, and then agent needs to relearn again. One character of this quantum hardware system discovered in this study is that it tends to overestimate the values used to determine the actions the agent will take. IBM's five qubit superconducting quantum processor is a popular quantum platform. The aims of our study are twofold. First we want to identify the hardware characteristic features of IBM's 5Q quantum computer when running this learning agent, compared with the ion trap processor. Second, through careful analysis, we observe that the quantum circuit employed in the ion trap processor for this agent could be simplified. Furthermore, when tested on IBM's 5Q quantum processor, our simplified circuit demonstrates its enhanced performance over the original circuit on one of the hard learning tasks investigated in the previous work. We also use IBM's quantum simulator when a good baseline is needed to compare the performances. As more and more quantum hardware devices are moving out of the laboratory and becoming generally available to public use, our work emphasizes the fact that the features and constraints of the quantum hardware could take a toll on the performance of quantum algorithms.

1. INTRODUCTION

Traditional computers have reached their limits of processing the ever growing data today. Quantum computing created from computer science and quantum physics emerges as a new computation paradigm, holding the promise to generate the next computing revolution. While classical computers can only store or process one of the two bits "0" or "1", quantum computers can make use of the superposition of two quantum states $|0\rangle$ and $|1\rangle$. As such they can store classically exponential size of data as linear size and can explore many qubits simultaneously. Put it in another way, a quantum computer can store all the bi-

nary numbers of the form $0,1,\cdots,i,\cdots,2^{n-1}$ simultaneously while a classical computer can only store one of them. As a result, a quantum computer can calculate multiple function f(x) values simultaneously, but a classical computer can only do one at a time. However, due to the laws of quantum mechanics, in general we cannot get all these function values by one single measurement. Rather, the key challenge in quantum algorithm design is to be creative in getting more information out through one measurement.

In certain areas such as machining learning, big data, and artificial intelligence, quantum computing has demonstrated dramatic speedups. The peculiar features of quantum computing such as super position, entanglement, and interference of quantum states are generally considered resources for this speed up. Examples of quantum machine learning algorithms can be found in [1-9].

Reinforcement learning is an area of machine learning in which a learning agent can learn from its actions taken in an environment. The nature of this kind of learning is illustrated in how the agent adjusts its actions in order to achieve its goal, say getting the maximum of reward. This is different from a supervised learning where the model learns from data directly. In this sense, we can say that reinforcement learning is learning from its actions and their feedback from the environment, so it learns from data (of interactions with the environment) indirectly (**Figure 1**). Some of the well-known reinforcement learning algorithms are Q-learning and SARSA [10-12].

A new model of reinforcement learning based on projective simulation is proposed in [13] and another variant of it, reflecting projective simulation, is shown to gain a quadratic speed up in the agent's deliberation time when it needs to adapt to rapidly changing environments [14].

As quantum machine learning is an emerging new field, it is very necessary to understand the characteristics of different quantum computing implementations and how they affect the performance of quantum algorithms. IBM recently released the Quantum Experience to make quantum computing available to the public. It allows users to connect to IBM's quantum processor via the IBM Cloud to learn the nature of quantum computing and to create and test different quantum algorithms on a real quantum processor [15]. One recent work [16] investigates the impact on a quantum classifier caused by the routinely used swap operation between two qubits on IBM's 5Q because of its star topology (**Figure 2**). Some other work using IBM's 5Q can be found in [17, 18], and a good textbook on quantum computing is [19].

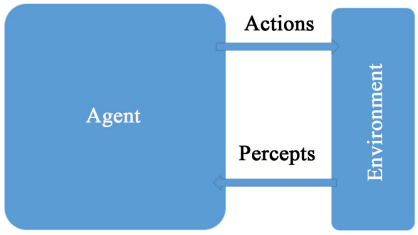

Figure 1. An illustration of the general idea of reinforcement learning: the learning of an agent is through its interactions with its environment. It takes an action and then the environment gives the feedback to the agent, and then agent learns to do better next time.

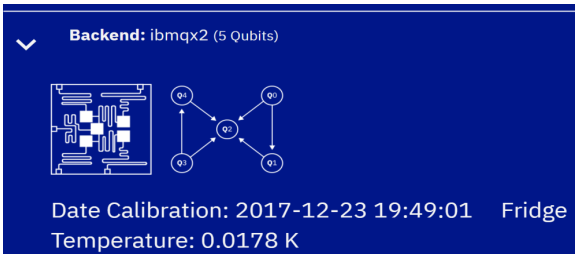

Figure 2. A schematic demonstration of the five qubits in IBM's 5Q chip, an image taken from https://quantumexperience.ng.bluemix.net/qx/editor.

2. REFLECTING PROJECTIVE SIMULATION

2.1. General Description

In projective simulation, a learning agent receives a sensory input from its environment, that is, a percept (from some set of percepts $S=\{s_1,s_2,\cdots\}$) and, based on the percept, it takes an action from the possible set of actions $A=\{a_1,a_2,\cdots,a_N\}$ This learning model uses a memory for its previous sequences of percept-action events, helping the agent to simulate future action before real action is taken. In a sense, the reflection on the percept-action history or memory helps the agent to choose a better action.

The process of the internal memory could be modeled as a Markov chain, and therefore can be conducted by discrete-time stochastic diffusion processes. They can be realized in a variety of physical systems, say in a quantum computing system. The deliberation process of the agent is based on these diffusion processes, in which the relevant Markov chain is diffused a particular number of times until a desired action is output. The choice of the action is dictated by the probability distribution from a Markov chain realized by the diffusion process.

The projective simulation learning also allows for many additional structures, which can be some percept-specific flags that are subsets of actions assigned to each percept to represent the agent's short-term memory, a feature that significantly improves the performance of the model [14].

In reflecting projective simulation (RPS), the reflection process of the agent is defined as many repetitions of the diffusion processes, where the agent takes its actions based on a specific probability distribution that can be updated during the learning process. This can helps the agent to react to the changing environment. In another word, there are two kinds of time quantities $1/\delta$ and $1/\varepsilon$, where the first measures time needed to generate the specified distribution in the agent's internal memory and the second is the time to sample a desirable action from it.

The work in [20] creates a quantum algorithm for an ion trap quantum processor that reduces the deliberation time to $1/\sqrt{\delta\varepsilon}$, compared with the $1/(\delta\varepsilon)$ time in the classical case. It studies a simplified version of RPS that uses rank-one Markov chains. In this special case, the entire Markov chain can converge in one step ($\delta=1$). So the time efficient issue of learning only has to deal with the time to sample the correct actions from the converged probability distribution which serves as like a policy for the agent in Q-learning or SARSA. The general algorithm can be modified as follows. First assume there are n flagged actions out of N actions ($n \ll N$) and use $a_1,a_2,\cdots,a_n$ to denote the initial probability distribution for these flagged actions, and $b_1,b_2,\cdots,b_n$ to represent the final stationary probability distribution for these flagged actions. In the initialization stage, the state $|\alpha\rangle=\sum_{i=1,\cdots,N}\sqrt{a_i}|i\rangle$ is prepared and the optimal number of k diffusion steps is carried out [21], with k = round $\left(\frac{\pi}{4\sqrt{\varepsilon}}-\frac{1}{2}\right)$ where $\varepsilon=\sum_{i=1,\cdots,n}b_i$ is the probability to sample a flagged action from the final stationary distribution. The reflection over these actions can be defined as:

$$ref_{\mathrm{A}}=2\sum_{i=1}^{n}|i\rangle\langle i|-I$$

After running the diffusion step a few time, a sample is taken from the distribution and if the sampled action is marked with a flag then the agent will take it as its action, otherwise the algorithm starts over again. The diffusion process consists of two reflections, over the flagged actions and over the distribution. The findings from [20] suggest that their quantum RPS (Q-RPS) agents requires an average of $O\left(1/\sqrt{\varepsilon}\right)$ samples until obtaining a flagged action while classical agents need $O(1/\varepsilon)$ samples.

The goal of Q-RPS algorithm is to increase the probability of obtaining a flagged action such that $\varepsilon_b>\varepsilon_a=\sum_{i=1,\cdots,n}a_i$ while maintaining the relative probabilities of the flagged actions according to the initial distribution, *i.e.*, $a_j/a_k=b_j/b_k$ for any $j,k\in\{1,2,\cdots,n\}$. Therefore, the main focus of the study in [20] is how to increase ε_b and maintain the same ratios of a_j/a_k and b_j/b_k given the initial distribution of a_i. One way to illustrate the application of Q-RPS is through a simple example. Next we will

explain a toy game that demonstrates the situations where this quantum learning agent can used.

2.2. Invasion Game

Let us introduce a simple game called invasion (see **Figure 3**). It has two parties, an attacker (A) and a defender (D). The goal of the attacker A is to enter the other side of the wall by going through a hole in the wall, which are placed at equal distances. The defender D will try to block a hole and thereby prevent A from invasion.

Assume that the attacker A uses a strategy that is unknown to the defender D, but, before each move, A shows some symbol that indicates its next move. In this case, it could be an arrow pointing right or left, indicating the direction of the subsequent move. The meaning of the symbols is priori unknown to D, but the symbols can be perceived and distinguished by D. In order for D to win the game, it needs to learn the meaning of the symbols during the game time. Here we also assume that the meaning of the symbols could be changed after some time, say to switch the meaning of the arrows so the right arrow means going to left. In summary, there are at least two learning tasks for D. One is to learn the meaning of the symbols and the other is to relearn the meaning if they get changed.

The main purpose of our work is to empirically study the quantum advantage for the defender D if it chooses to use some quantum algorithms for its learning as already illustrated in [20].

2.3. Quantum Circuit Design for Q-RPS

The ion trap quantum system in [20] uses two qubits to represent four action states: $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$, two of which are flagged: $|00\rangle$ and $|01\rangle$. The initial probability distribution of a_i is prepared in this state:

$$|\alpha\rangle = R_{1,y}(\theta_1)R_{2,y}(\theta_2)|0\rangle \tag{1}$$

here $R_{j,y}(\theta) = \exp\left(-i\frac{\theta}{2}Y_j\right)$, Y_j is the Pauli matrix Y for qubit j. $R_{j,z}(\theta) = \exp\left(-i\frac{\theta}{2}Z_j\right)$, Z_j is the Pauli matrix Z for qubit j. Given $\varepsilon_a = a_{00} + a_{01}$, θ_1 and θ_2 can be obtained via $\varepsilon_a = \cos^2\left(\frac{\theta_1}{2}\right)$ and $a_{00}/\varepsilon_a = \cos^2\left(\frac{\theta_2}{2}\right)$ respectively.

The reflection over the flagged actions is defined as:

$$ref_{\mathrm{A}} = R_{1,z}(-\pi) \tag{2}$$

and the reflection over the initial probability distribution of a_i is represented as:

$$ref_{\alpha} = R_{1,y}(\theta_1 - \pi)R_{2,y}\left(\theta_2 + \frac{\pi}{2}\right)U_{CNOT}R_{1,y}(-\theta_1 - \pi)R_{2,y}\left(\theta_2 - \frac{\pi}{2}\right) \tag{3}$$

where U_{CNOT} is the unitary matrix for the controlled not gate. The whole Q-RPS is encoded in the following quantum circuit (see **Figure 4**).

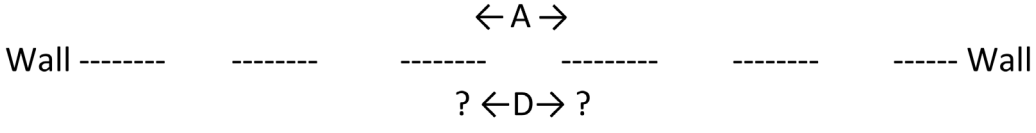

Figure 3. An illustration of the invasion game [13]: the Attacker agent A tries to go to the other side of the wall by entering one of the holes in the wall, while the Defender agent D tries to guess A's next move from learning the meaning of a symbol shown.

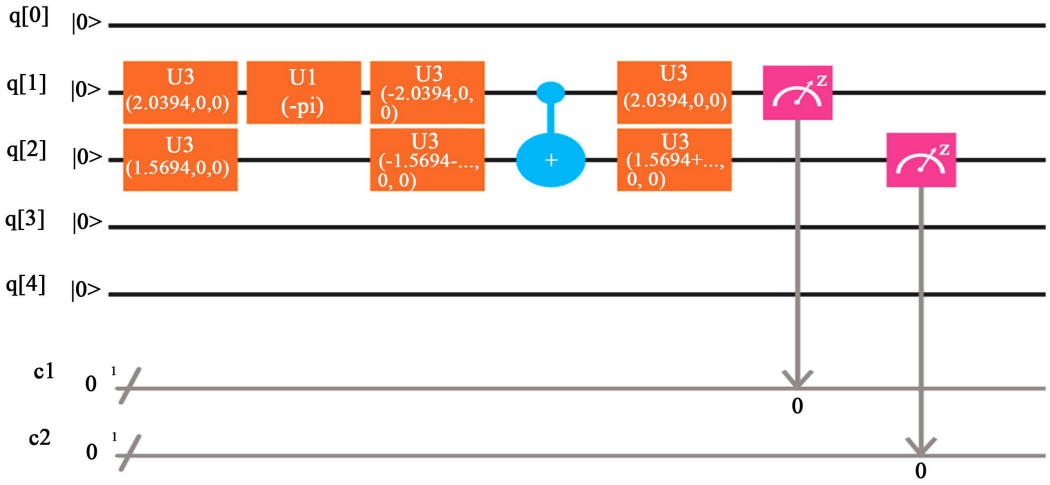

Figure 4. **An implementation of the circuit in [20] for the Q-RPS in IBM's quantum composer, where $\theta_1 = 2.0394$ and $\theta_2 = 1.5694$ as used in the experiments in [20]. The first two U3 gates prepare the input states, the U1 gate serves as ref_A and the gates after U1 make up ref_a.**

3. RESULTS

There is a randomness in quantum computing, therefore, a quantum algorithm has to repeat multiple times in order to get a stable reading of its results. IBM's quantum experience calls each such time a shot. Since the experimental results from [20] is based on 1600 shots, we decide to run the Q-RPS algorithm on IBM's 5Q using 1000 shots or multiples of 1000 to set up a fair base for comparison. Another reason is that a maximum of 1024 shots is the limit with our current credit. We use 1000 shots on IBM's simulator to match the same number of shots on IBM's 5Q, and also the maximally allowed 8192 shots on the simulator to get the best possible results. Because quantum computing involves randomness, all our experiments are conducted with various shots on IBM's 5Q or IBM's quantum simulator and the averaged results are reported here.

3.1. Running the Experiments in [20] on IBM's 5Q Processor and IBM's Quantum Simulator

The major task of the work in [20] is to elucidate the two characteristic features of rank-one Q-RPS: 1) given ε_a, how much can ε_b be increased while maintaining the same ratio value $r_b = 1$ during the whole process (**Table 1**) 2) how does the ratio of $r_b = b_j / b_k$ depend on that of $r_a = a_j / a_k$ in the process of increasing ε_b (**Table 2**). The first experiment is to fix $r_a = 1$ and let ε_a vary so the optimal k value moves from 1 to 7 and the second fix ε_a to two values so the optimal $k = 1$ in one case and $k = 3$ in the other case and let r_a change. Our purpose here is to run the same experiments as those reported in [20] on their ion trap quantum processor, but on IBM's 5Q computer. Hopefully different behaviors of these kinds of quantum hardware systems can be observed. The findings in [20] imply that their ion trap system tends to overestimate the values it computes for the Q-RPS agent. Using IBM's simulator as a baseline estimator, Table 1 shows that the ε_b values generated by IBM's 5Q (1000 shots) are closer to the true values of ε_a than those by the ion trap system (1600 shots) from [20] but the theoretical analysis indicates that the two ratios are supposed to be the same. However, due to the noise and decoherence of quantum computing, it is not observed here. To visualize the quantum hardware difference between the ion trap system and IBM's 5Q, we also display the bar charts of the two (**Figures 5-7**). Numerically, there is no clear difference between 1000 shots and 10x1000 shots for IBM's 5Q, implied by **Table 1** and **Figures 5-7**.

Table 2 and **Figure 6** and **Figure 7** highlight that IBM's 5Q is more accurate in the case for $k = 1$ and underestimate in the case for $k = 3$, remembering that the ion trap system shows overestimate in both cases [20]. More specifically, the r_b values from IBM's 5Q are closer to the r_a values than those from the

Figure 5. Bar chart to visualize the key numerical values in Table 1. The theoretical ε_b value is 1, so all other ε_b values should get close to 1.

Table 1. Experiment that changes input ε_a and see how well ion trap, 5Q, or simulator can increase ε_a. In this case γ_a remains 1.

	Theory			Theory			IBM's simulator (10 × 1000 shots)		
k	a_{00}	a_{01}	ε_a	b_{00}	b_{01}	ε_b	b_{00}	b_{01}	ε_b
1	0.1371	0.1371	0.2742	0.4966	0.49	0.99	0.49	0.5	0.99
2	0.0493	0.0493	0.0986	0.4996	0.49	0.99	0.49	0.5	0.99
3	0.252	0.252	0.504	0.4999	0.49	0.99	0.5	0.5	0.99
4	0.0152	0.0152	0.0304	0.5	0.5	1	0.49	0.5	1.0
5	0.0102	0.0102	0.0204	0.5	0.5	1	0.50	0.5	1.0
6	0.0073	0.0073	0.0146	0.5	0.5	1	0.49	0.5	1.0
7	0.0055	0.0055	0.011	0.5	0.5	1	0.5	0.5	1.0
	Ion trap processor (1600 shots)			IBM's 5Q (1000 shots)			IBM's 5Q (10 × 1000 shots)		
k	a_{00}	a_{01}	ε_a	b_{00}	b_{01}	ε_b	b_{00}	b_{01}	ε_b
1	0.449	0.44	0.889	0.449	0.505	0.954	0.4671	0.4802	0.9473
2	0.347	0.353	0.7	0.414	0.508	0.922	0.424	0.5053	0.9293
3	0.438	0.334	0.772	0.396	0.505	0.901	0.4036	0.5039	0.9075
4	0.422	0.336	0.758	0.397	0.507	0.904	0.3765	0.5147	0.8912
5	0.407	0.331	0.738	0.347	0.566	0.913	0.3637	0.529	0.8927
6	0.431	0.324	0.755	0.316	0.533	0.849	0.3289	0.5466	0.8755
7	0.365	0.299	0.664	0.297	0.561	0.858	0.3158	0.5425	0.8583

Table 2. Experiment that changes r_a and see how well r_b can keep close to r_a. The theory suggests that they should be the same.

		Theory		IBM's simulator (10 × 1000 shots)			IBM's simulator (10 × 8192) shots)		
k	a_{00}	a_{01}	r_a	b_{00}	b_{01}	r_b	b_{00}	b_{01}	r_b
1	0.0027	0.2714	0.0099	0.0098	0.9829	0.009	0.0094	0.9837	0.0095
1	0.0725	0.2015	0.3599	0.2625	0.7316	0.3588	0.2657	0.7277	0.3651
1	0.1138	0.1603	0.7100	0.4066	0.5841	0.6961	0.4104	0.5826	0.7044
1	0.1410	0.1330	1.0599	0.5108	0.4817	1.0604	0.5121	0.4807	1.0653
1	0.1604	0.1137	1.4099	0.5854	0.4077	1.435	0.5814	0.4121	1.4106
1	0.1748	0.0993	1.7599	0.6289	0.3647	1.7244	0.6336	0.3601	1.7593
1	0.1370	0.1370	1	0.4987	0.4949	1.0076	0.4955	0.4977	0.9956
3	0.0045	0.0457	0.1000	0.0913	0.9084	0.1005	0.0922	0.9075	0.1016
3	0.0163	0.0340	0.4800	0.3217	0.6781	0.4744	0.3261	0.6736	0.4841
3	0.0232	0.0270	0.8599	0.4517	0.5478	0.8245	0.4616	0.5381	0.8579
3	0.0278	0.0224	1.2402	0.5545	0.4454	1.2449	0.5526	0.4472	1.2356
3	0.0311	0.0192	1.6201	0.6196	0.3803	1.629	0.6177	0.3820	1.6168
3	0.0335	0.0167	1.9994	0.655	0.3447	1.9002	0.6652	0.3346	1.9878
	Ion trap processor (1600 shots)			IBM's 5Q (1000 shots)			IBM's 5Q (10 × 1000 shots)		
k	b_{00}	b_{01}	r_b	b_{00}	b_{01}	r_b	b_{00}	b_{01}	r_b
1	0.061	0.809	0.0754	0.047	0.904	0.05199115	0.0414	0.908	0.04564
1	0.29	0.583	0.4974	0.25	0.694	0.360230548	0.258	0.6896	0.37445
1	0.415	0.466	0.8905	0.369	0.566	0.651943463	0.3889	0.5575	0.69891
1	0.488	0.389	1.2544	0.456	0.503	0.906560636	0.4726	0.4762	0.99449
1	0.519	0.351	1.4786	0.571	0.383	1.490861619	0.5571	0.3941	1.41713
1	0.566	0.305	1.8557	0.606	0.349	1.736389685	0.5926	0.3573	1.66588
1	0.468	0.401	1.1670	0.452	0.491	0.920570265	0.4681	0.4768	0.98310
3	0.127	0.718	0.1768	0.08	0.831	0.09627	0.0875	0.8198	0.10673
3	0.301	0.518	0.5810	0.253	0.67	0.377612	0.257	0.6614	0.38857
3	0.442	0.451	0.9800	0.404	0.523	0.772467	0.3739	0.5409	0.69125
3	0.51	0.354	1.4406	0.462	0.444	1.040541	0.4546	0.4567	0.99540
3	0.551	0.305	1.8065	0.515	0.393	1.310433	0.5098	0.4006	1.27259
3	0.586	0.268	2.1865	0.569	0.355	1.602817	0.5548	0.3573	1.55275

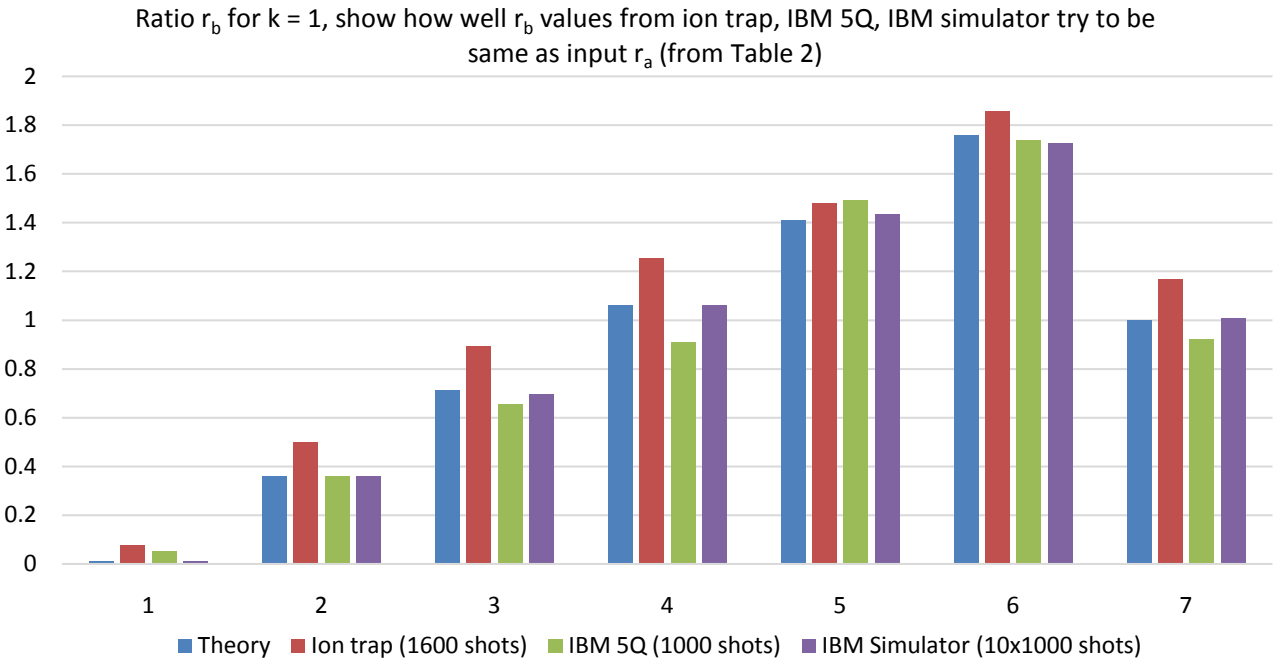

Figure 6. **Bar chart to visualize the key numerical values in Table 2 for $k = 1$. The values in the x-axis represent the seven different inputs in Table 2 for $k = 1$. This plot shows ion trap tends to overestimate and IBM's 5Q is more accurate.**

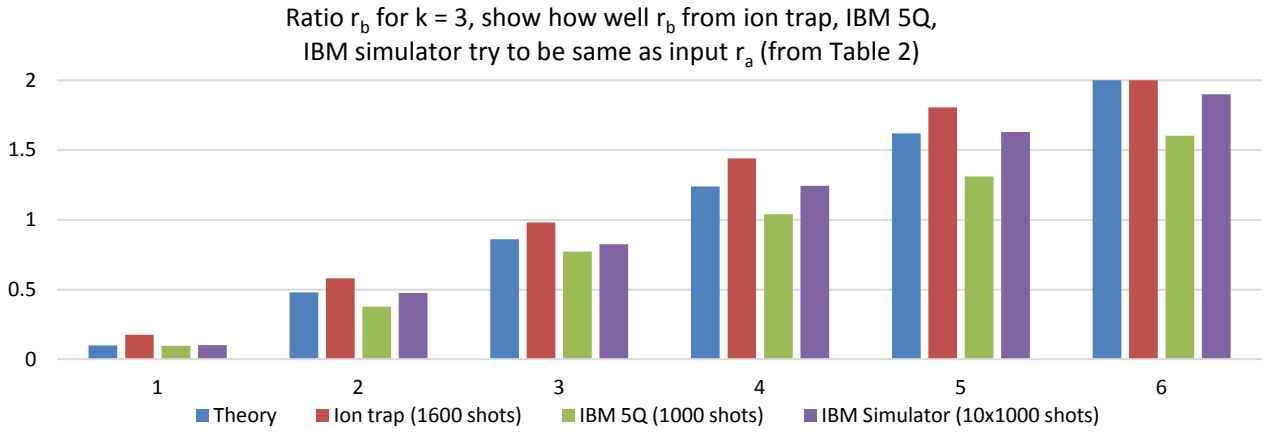

Figure 7. **Bar chart to visualize the key numerical values in Table 2 for $k = 3$. The values in the x-axis represent the six different inputs in Table 2 for $k = 3$. This plot shows ion trap tends to overestimate and IBM's 5Q tends to underestimate.**

ion trap ($k = 1$ in this case), and the r_b values from IBM's 5Q show underestimate while those from ion trap display overestimate.

Here we show the results from IBM simulator with 1000 shots to illustrate the best possible outcome for the results from IBM 5Q with 1000 shots. The case of $a_{00} = 0.0335$ and $a_{01} = 0.0167$ is a difficult learning task as IBM's simulator runs 10 times of the maximum shots of 8192 and can only get 1.98 with the target of 1.9994. Also, with 10×1000 shots, it gets 1.90002.

It would be informative if we could run the experiment for the case of $a_{00} = 0.0335$ and $a_{01} = 0.0167$ in Table 2 on IBM's simulator to see the whole process of r_b values approaching to its final

number 1.98 using various shots from 20 to 8192. The result of this run is shown in Figure 7 with r_b values along with their standard deviations. We also run the same experiment on IBM's 5Q with various shots from 20 to 1000, see Figure 8. It is interesting to see that std = 0.0981 for IBM's 5Q and std = 0.1204 for IBM's simulator at 1000 shots, and the real device's std value is even smaller than that of the simulator. In order for the simulator to reduce its std value to std = 0.0974, it needs to run up to 2000 shots. The r_b values in Figure 8 and Figure 9 are all based 10 repetitions of each run since we need to take a standard deviation of these values.

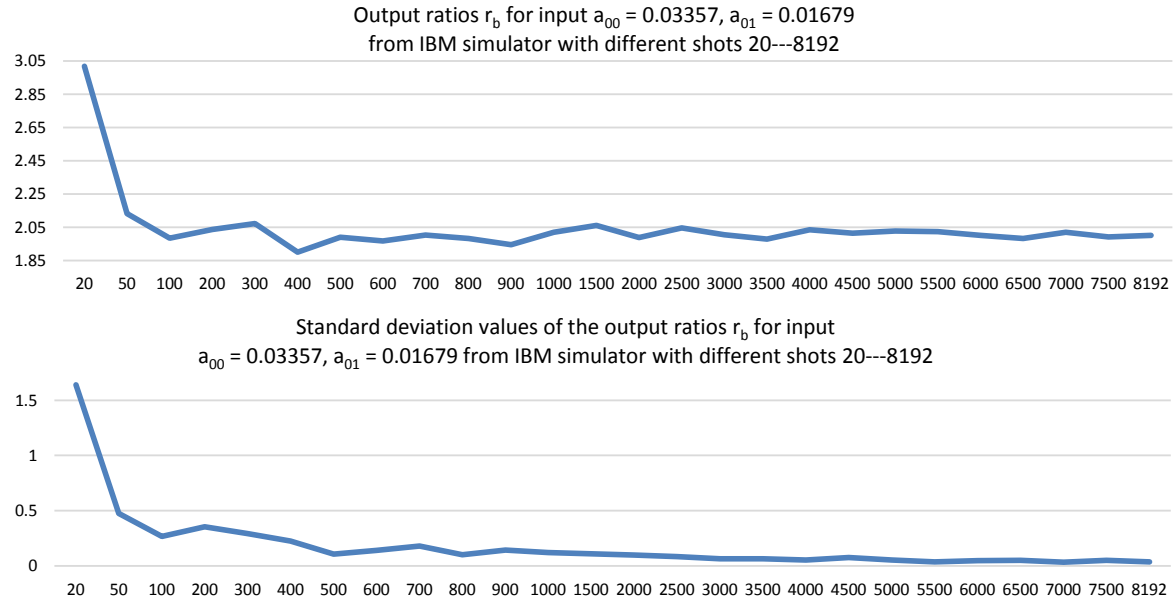

Figure 8. Curve for r_b and curve for its standard deviation for input a_{00} = 0.03357, a_{01} = 0.01679 from IBM simulator with different shots 20 - 8192.

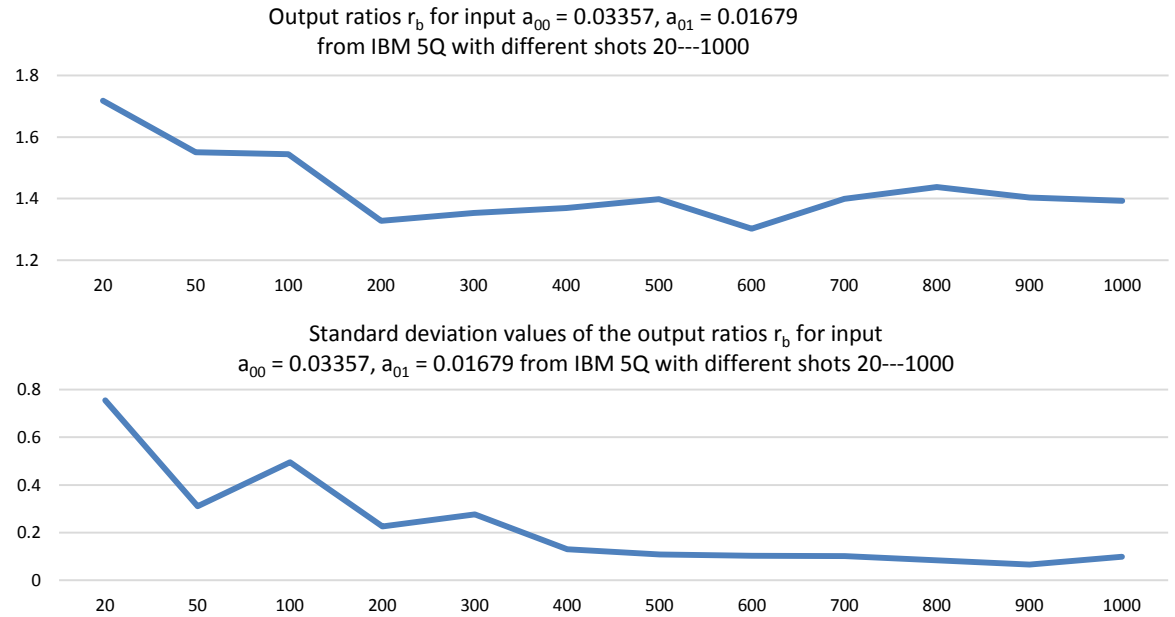

Figure 9. Curve for r_b and curve for its standard deviation for input a_{00} = 0.03357, a_{01} = 0.01679 from IBM 5Q with different shots 20 - 1000.

To summarize our results in this section, we can say that using IBM's simulator as a baseline, IBM 5Q is more accurate than the ion trap quantum system in calculating the probabilities for the flagged actions for the agent and also tends to underestimate if any.

3.2. Comparing the Quantum Circuits with and without Pi Rotation on IBM's 5Q Processor and IBM's Quantum Simulator

From the geometric meaning of $R_{1,y}$, we can simplify the reflection equation in (3) by removing the pi rotation:

$$ref_\alpha = R_{1,y}(\theta_1)R_{2,y}\left(\theta_2+\frac{\pi}{2}\right)U_{CNOT}R_{1,y}(-\theta_1)R_{2,y}\left(\theta_2-\frac{\pi}{2}\right) \tag{4}$$

We first show that these two circuits are equivalent empirically by running the same experiment using IBM's quantum simulator with maximum 8192 shots, which show that they produce the same outcome. Then we test them on IBM's 5Q. For this purpose we select one of the hard learning tasks in experiment shown in Table 1, which requires $k = 7$ to achieve the optimal result.

It also allows us to run the experiment for various k values before and after $k = 7$ to gain the whole picture of their work. Surprisingly enough, our simplified circuit outperforms the original one. Note that in this run, we use the same a_{00} and a_{01} for each k in the range of 1 to 7, showing the gradual approaching to the limit values of $b_{00} = 0.5$ and $b_{01} = 0.5$.

For each k from 1 to 7, the two circuits produce the same results on IBM's simulator, minus the randomness of quantum computing. The reason to choose maximum of 8192 shots in this run is to ensure the randomness effects of quantum computing is reduced to a possible minimum. Table 3 shows that our simplified circuit is equivalent to the original one, and the natural question is how they differ on a real quantum device? For this end, we first run them on IBM's simulator to establish a baseline line for subsequent comparison (Figure 10), then run them on IBM's 5Q, and the results are displayed in Figures 11-13, which indicate that the performance of the simplified circuit is much closer to the best possible result of the original circuit in [20]. To get a bigger picture, we intentionally run our experiment from $k = 1$ up to $k = 9$ to reveal their performance before and after the optimal value $k = 7$.

Table 3. Experiment to show that the performances of the original circuit and our simplified one on IBM's simulator are the same for input $a_{00} = 0.0055$ and $a_{01} = 0.0055$ which is used in Table 1.

	Theory			Theory			Circuit with pi rotation IBM's 5Q (10 × 8192 shots)			Circuit without pi rotation IBM's 5Q (10 × 8192 shots)		
k	a_{00}	a_{01}	e_a	b_{00}	b_{01}	e_b	b_{00}	b_{01}	e_b	b_{00}	b_{01}	e_b
1	0.0055	0.0055	0.011	0.5	0.5	1	0.049133	0.048022	1.023132	0.048352	0.048584	0.995226
2	0.0055	0.0055	0.011	0.5	0.5	1	0.123694	0.125464	0.985892	0.12771	0.12616	1.012288
3	0.0055	0.0055	0.011	0.5	0.5	1	0.22406	0.222791	1.005698	0.22533	0.227039	0.992473
4	0.0055	0.0055	0.011	0.5	0.5	1	0.326379	0.331323	0.985078	0.330566	0.32843	1.006504
5	0.0055	0.0055	0.011	0.5	0.5	1	0.419922	0.419153	1.001835	0.421594	0.418201	1.008115
6	0.0055	0.0055	0.011	0.5	0.5	1	0.479578	0.480054	0.999008	0.478076	0.480798	0.994338
7	0.0055	0.0055	0.011	0.5	0.5	1	0.500281	0.499707	1.001148	0.498096	0.501855	0.992508

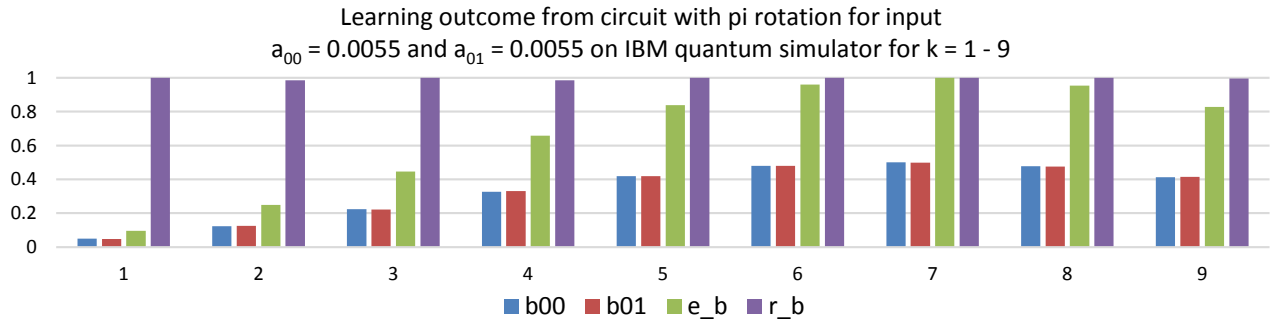

Figure 10. Bar chart to visualize the key numerical values in Table 3, which serves as a baseline for the performance comparison in Figures 11-13. In this plot, the values in the x-axis are the k values from 1 to 9. The optimal k value is 7, but here we show many k values before and after 7 to reveal the whole picture. This plot also shows that the optimal k value is 7.

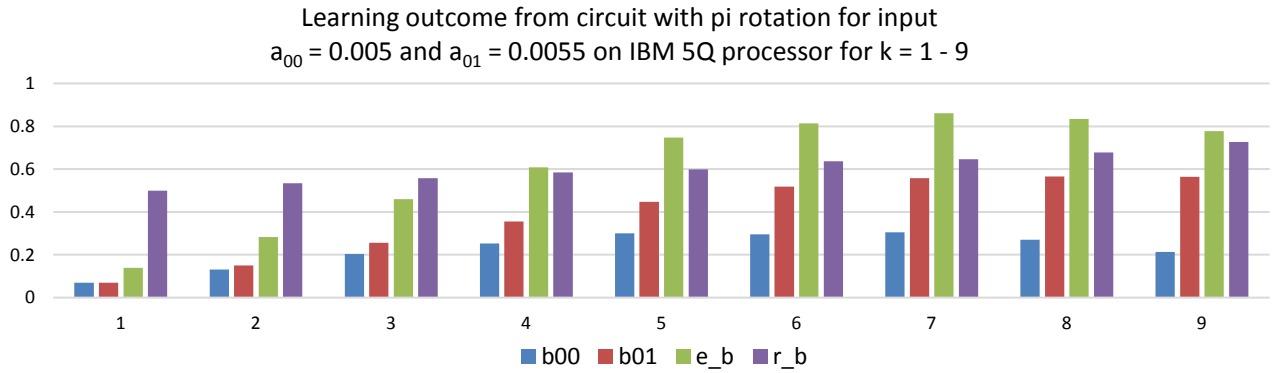

Figure 11. Run the original circuit on IBM's 5Q with 10 × 1000 shots. In this plot, the values in the x-axis are the k values from 1 to 9. The bar charts in this figure are quite different from the simulator in Figure 10.

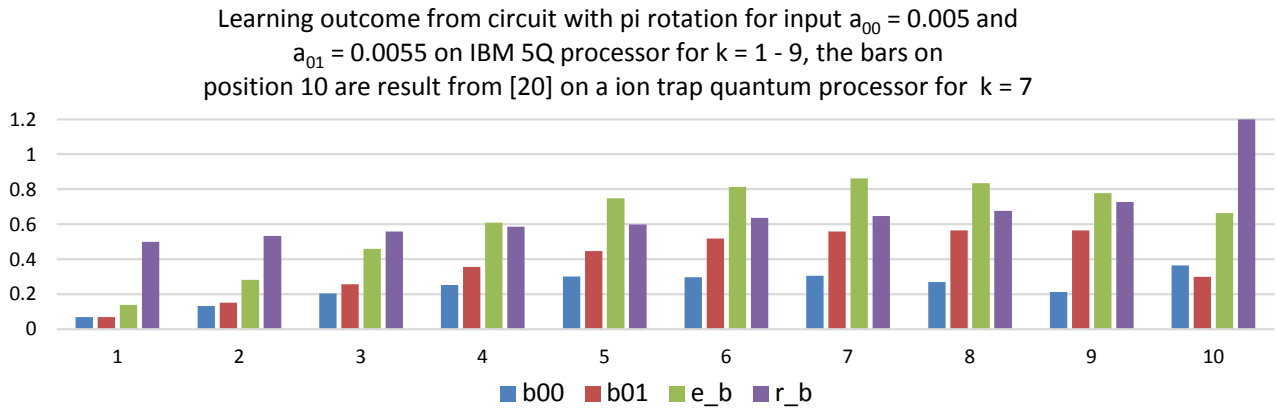

Figure 12. This figure is the same as Figure 11 with one extra: we have added the result from the ion trap in [20] for $k = 7$ here to visually show the behaviors. We place this result at position 10 and we should not think it is for $k = 10$. It is the run for $k = 7$ from [20]. The goal of this experiment is to increase ε_b value, so we can see that at $k = 7$, IBM's 5Q is better than that of the ion trap at position 10.

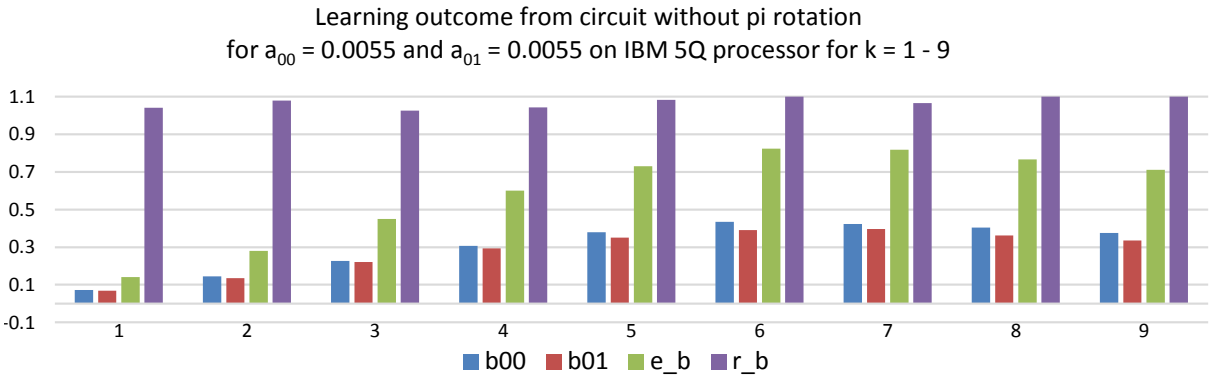

Figure 13. This is the same figure as Figure 11 but for the results from our simplified circuit. We can see that at each k, the 50 - 50 ratio of b_{00} and b_{01} is well maintained and $k = 6$ is the best result. This is expected as the round off formula for k in section 2.1 can be off by one. The patterns of the bar charts in this figure are similar to the best possible results from the simulator in Figure 10, which suggests that the results in this figure from our simplified circuit are better than those from the original one in [20].

4. CONCLUSION

As we are having more data and facing more complex problems today than yesterday, reaching the limits of the capability of classical computers, the call for advancement of quantum computation hardware and quantum algorithms is so clear and urgent. Quantum machine learning has become a matter of interest recently due to its great potential as a possible solution to some hard computing challenges. Research has shown that artificial intelligence, and in particular machine learning can benefit from quantum computing.

In a typical reinforcement learning, a learning agent needs to learn from its interactions with its environment. But the environment could change in the real world. Consequently, the agent needs to react to this change. How a quantum processor can speed up the decision making for the agent in this case is the focus of the study in [20], which also suggests that their ion trap system tends to overestimate. Our study investigates how IBM's 5Q computer enables the learning of this AI agent.

Our work uses IBM's simulator as a baseline and demonstrates that IBM's 5Q is more accurate than the ion trap system and tends to underestimate if any, during the decision making of this agent. Furthermore, our analysis reveals that the quantum circuit used in [20] could be simplified. When tested on IBM's 5Q, our simplified circuit performs better than the ion trap system on one of the hard learning tasks reported in Table 1.

Quantum computing relies on multiple repetitions of the same experiment to get a high accurate answer. In this direction, our analysis seems to demonstrate that the current quantum computing technologies do not provide highly accurate calculation when needed, compared with the accuracy that the classical computing offers today.

As more and more quantum machine learning algorithms are being developed and tested on different quantum hardware systems, it is desirable to learn the influence of quantum hardware on the performance of these quantum algorithms. Our work helps to understand the potential of a quantum computer in the field of artificial intelligence and has developed new insight into how the features and constraints of quantum hardware affect the quantum algorithms.

ACKNOWLEDGMENTS

We are grateful to Theeraphot Sriarunothai, one of the authors of [20], for valuable discussions.

REFERENCES

1. Schuld, M., Fingerhuth, M. and Petruccione, F. (2017) Implementing a Distance-Based Classifier with a Quantum Interference Circuit. *A Letters Journal of Exploring the Frontiers of Physics*, **119**, 60002.
2. Schuld, M., Sinayskiy, I. and Petruccione, F. (2016) Prediction by Linear Regression on a Quantum Computer. *Physical Review A*, **94**, Article ID: 022342. https://doi.org/10.1103/PhysRevA.94.022342
3. Schuld, M., Sinayskiy, I. and Petruccione, F. (2015) Simulating a Perceptron on a Quantum Computer. *Physics Letters A*, **379**, 660-663. https://doi.org/10.1016/j.physleta.2014.11.061
4. Schuld, M., Sinayskiy, I. and Petruccione, F. (2014) Quantum Computing for Pattern Classification. In: Palm, D.-N. and Park, S.-B., Eds., *Pacific Rim International Conference on Artificial Intelligence*, Spring International Publishing, Switzerland, 208-220. https://doi.org/10.1007/978-3-319-13560-1_17
5. Schuld, M., Sinayskiy, I. and Petruccione, F. (2014) The Quest for a Quantum Neural Network. *Quantum Information Processing*, **13**, 2567-2586. https://doi.org/10.1007/s11128-014-0809-8
6. Schuld, M., Sinayskiy, I. and Petruccione, F. (2014) Quantum Walks on Graphs Representing the Firing Patterns of a Quantum Neural Network. *Physical Review A*, **89**, Article ID: 032333. https://doi.org/10.1103/PhysRevA.89.032333
7. Cai, X.-D., Wu, D., Su, Z.-E., Chen, M.-C., Wang, X.-L., Li, L., Liu, N.-L., Lu, C.-Y. and Pan, J.-W. (2015) Entanglement-Based Machine Learning on a Quantum Computer. *Physical Review Letters*, **114**, Article ID: 110504. https://doi.org/10.1103/PhysRevLett.114.110504
8. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N. and Lloyd, S. (2017) Quantum Machine Learning. *Nature*, **549**, 195-202. https://doi.org/10.1038/nature23474
9. Dunjko, V., Taylor, J.M. and Briegel, H.J. (2016) Quantum-Enhanced Machine Learning. *Physical Review Letters*, **117**, Article ID: 130501. https://doi.org/10.1103/PhysRevLett.117.130501
10. Sutton, R.S. and Barto, A.G. (1998) Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.
11. Ganger, M., Duryea, E. and Hu, W. (2016) Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning. *Journal of Data Analysis and Information Processing*, **4**, 159-176. https://doi.org/10.4236/jdaip.2016.44014
12. Duryea, E., Ganger, M. and Hu, W. (2016) Exploring Deep Reinforcement Learning with Multi Q-Learning. *Intelligent Control and Automation*, **7**, 129-144.
13. Briegel, H.J. and De las Cuevas, G. (2012) Projective Simulation for Artificial Intelligence. *Scientific Reports*, **2**, 400.
14. Paparo, G.D., Dunjko, V., Makmal, A., Martin-Delgado, M.A. and Briegel, H.J. (2014) Quantum Speedup for Active Learning Agents. *Physical Review X*, **4**, Article ID: 031002. https://doi.org/10.1103/PhysRevX.4.031002
15. IBM. Quantum Experience. http://www.research.ibm.com/quantum
16. Hu, W. (2018) Empirical Analysis of a Quantum Classifier Implemented on IBM's 5Q Quantum Computer. *Journal of Quantum Information Science*, **8**, 1-11.
17. Alsina, D. and Latorre, J.I. (2016) Experimental Test of Mermin Inequalities on a Five-Qubit Quantum Computer. *Physical Review A*, **94**, Article ID: 012314. https://doi.org/10.1103/PhysRevA.94.012314
18. Berta, M., Wehner, S. and Wilde, M.M. (2016) Entropic Uncertainty and Measurement Reversibility. *New Journal of Physics*, **18**, Article ID: 073004. https://doi.org/10.1088/1367-2630/18/7/073004
19. Nielsen, M.A. and Chuang, I.L. (2010) Quantum Computation and Quantum Information. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511976667

20. Sriarunothai, T., Wolk, S., Giri, G.S., Friis, N., Dunjko, V., Briegel, H.J. and Wunderlich, C. (2017) Speeding-Up the Decision Making of a Learning Agent using an Ion Trap Quantum Processor. https://arxiv.org/abs/1709.01366

21. Grover, L.K. (1996) A Framework for Fast Quantum Mechanical Algorithms. In: *Proceedings of the* 28*th Annual ACM Symposium on Theory of Computing*, ACM, New York, 212-219. https://doi.org/10.1145/237814.237866