Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 3511029, 11 pages
https://doi.org/10.1155/2021/3511029

Research Article

Variational Quantum Circuit-Based Reinforcement Learning for POMDP and Experimental Implementation

Tomoaki Kimura,[1] Kodai Shiba,[1,2] Chih-Chieh Chen,[2] Masaru Sogabe,[2] Katsuyoshi Sakamoto,[1,3] and Tomah Sogabe[1,2,3]

[1]*Engineering Department, The University of Electro-Communications, 182-8585 Tokyo, Japan*
[2]*Grid Inc., 107-0061 Tokyo, Japan*
[3]*i-PERC, The University of Electro-Communications, 182-8585 Tokyo, Japan*

Correspondence should be addressed to Chih-Chieh Chen; chen.chih.chieh@gridsolar.jp and Tomah Sogabe; sogabe@uec.ac.jp

Received 18 August 2021; Revised 4 October 2021; Accepted 9 November 2021; Published 23 December 2021

Academic Editor: Erivelton Geraldo Nepomuceno

Variational quantum circuit is proposed for applications in supervised learning and reinforcement learning to harness potential quantum advantage. However, many practical applications in robotics and time-series analysis are in partially observable environment. In this work, we propose an algorithm based on variational quantum circuits for reinforcement learning under partially observable environment. Simulations suggest learning advantage over several classical counterparts. The learned parameters are then tested on IBMQ systems to demonstrate the applicability of our approach for real-machine-based predictions.

1. Introduction

The combination of deep neural networks and reinforcement learning is demonstrated as an effective way to tackle computational problems that are difficult for other traditional approaches [1, 2]. In the usual reinforcement learning settings, the underlying model is the Markov decision process (MDP) [3, 4], where the information of environment can be completely obtained by the learning agent. However, in many real-world applications such as robotics [5], the observations are obtained from the sensors of mobile robots and hence are limited. In such cases, it is necessary to model the problem by the partially observable Markov decision process (POMDP) [6, 7]. POMDP is a framework for the environments where complete information cannot be obtained. One difficulty occurs in the POMDP in robotics setting is the perceptual aliasing problem, in which the learning agent cannot distinguish one state from another state due to the limitation of observation ability. To make proper decision under limited observation, the agent has to memorize its history to distinguish one state from the other.

One traditional method for POMDP is the belief value iteration [5–7], where the agent maintains a belief distribution over possible states. The value function then becomes a functional over continuous functions and hence is computationally expensive. To deal with the computational difficulties, other methods using Monte Carlo [5, 8] and recurrent neural network [9, 10] have been proposed. These methods are difficult to execute without a sufficient computing capacity and memory. Complex-valued reinforcement learning has been proposed as a POMDP algorithm that can be executed with less computational resources [11]. In complex-valued reinforcement learning, the action value function gives a complex number, and there is another complex number internal reference vector that represents time series information. Tables [11, 12] and neural networks [13, 14] have been used to express complex action value functions. However, expressing correlated complex numbers using a classical computer is not considered to be a good choice from the viewpoint of memory efficiency. On the other hand, since quantum computers perform calculations over Hilbert space, it is reasonable to think that quantum

processors can be used to represent complex-valued functions efficiently. If the abovementioned complex action value function can be expressed by a quantum computer, it may become a more memory efficient POMDP learning method.

In this work, we propose a method for performing complex-valued reinforcement learning by expressing a complex action value function using a quantum computer. Previous works apply current quantum hardware [15] for supervised machine learning using the variational quantum circuit method [16–18]. The applications of variational quantum circuit to the value function in reinforcement learning problem are also demonstrated [19–22]. Some works use classical-quantum hybrid model to solve large problems [23, 24]. Other works use variational quantum circuit to represent policy in reinforcement learning [25, 26]. Variational quantum circuit is also applied to represent both policy and value function in actor-critic method [27]. Quantum algorithms are also applied to sampling from policy in deep energy-based reinforcement learning [28]. These quantum algorithms for reinforcement learning are based on MDP. We implement the variational quantum circuit for complex-valued action value function approximation and compare the performance against other methods like complex neural networks. The learning performance shows advantage over some other classical methods. We further use the parameters learned from simulation for predictions with IBMQ systems. The agent is able to reach the goal state with the predictions made by IBM machines. This discovery suggests that the use of variational quantum circuit for POMDP provides possible advantage.

This paper is organized as follows. Background section introduces the concept of POMDP. In Methods section, variational quantum circuit and neural network are applied for complex-valued Q-learning. In Results and Discussion section, partially observable maze environment experiment results are shown. Conclusions section describes conclusion and future work.

1.1. Background. POMDP is a general framework for planning in the environments where perfect information cannot be gotten. In POMDP, the agent cannot fully observe the state but instead receives an observation from the environment, and the observation does not satisfy Markov property. A POMDP is defined as a tuple $(S, A, T, R, \Omega, O, \gamma)$. S is the set of states. A is the set of actions. T is the state transition probability. R is the reward function. Ω is the set of observations. O is the observation probability. γ is the discount rate.

When the agent executes an action a in the environment, the state transitions from a state s to a next state s' according to the state transition probability $T = \Pr(s'|s, a)$ and the agent receives an observation o from the environment according to the observation probability $O = \Pr(o|s', a)$ and a reward r according to the reward function $R(s, a)$. The history h_t is the time series of actions and observations and is expressed as $h_t = \{a_0, o_1, a_1, o_2, \ldots, a_{t-1}, o_t\}$. As the agent cannot fully observe state, it uses the belief state, which is the probability distribution that represents in which states the agent is. Belief state B is denoted by $B(s, h) = \Pr(s|h)$, and it can be updated by the following formula:

$$B(s', \text{hao}) = \frac{\sum_s \Pr(o|s', a)\Pr(s'|s, a)B(s, h)}{\sum_s \sum_{s''} \Pr(o|s'', a)\Pr(s''|s, a)B(s, h)}. \quad (1)$$

The agent's policy is denoted by $\pi(h, a) = \Pr(a|h)$ or $\pi(b, a) = \Pr(a|b)$. The purpose of the agent is to get a policy π^* that maximizes the expected total reward $\mathbb{E}_\pi[\sum_{i=t}^{T} \gamma^{i-t} r_i]$.

There are two types of methods for planning POMDP. One method is based on value iteration method [7, 29]. This type of method updates the belief state and value function using a model. Belief state is updated by equation (1), and value function is updated by updating the set of alpha vectors. Another method is the method using a black box simulator [8]. Black box simulator is a model that receives state and action and returns next state, observation, and reward. This method executes Monte Carlo tree search and Monte Carlo update of belief state using this simulator. Although these two planning methods use model or black box model, as normally model is not given to the agent, algorithms learning model or algorithms not using model are needed. The former is model-based method, and the latter is model-free method.

Model-based methods inference the model and then execute planning using the learned model. To learn the model, Bayesian method [30, 31] and nonparametric approach [32] are proposed. Model-free methods are RNN method [9, 10, 33], complex-valued reinforcement learning [11–13], etc. These methods do not use belief state but incorporate time series information directly into the value function. In this paper, we focus on the model-free method especially complex-valued reinforcement learning.

2. Methods

2.1. Complex-Valued Q-Learning for POMDP. A POMDP problem that we are interested in here can be described by a tuple $(S, A, T, R, \Omega, O, \gamma)$. S is a discrete set of states. A is a discrete set of actions. $T(s'|s, a)$ is a state transition probability matrix for $s, s' \in S$ and $a \in A$. $R(s, a)$ is a reward function for $s \in S$ and $a \in A$. Ω is a discrete set of observations. $O(o|s, a)$ is an observation probability matrix for $o \in \Omega$, $s \in S$, and $a \in A$. $\gamma \in (0, 1)$ is the discount rate. In the examples in this work, both the state transition and the observation are deterministic. We look for a policy for which the expected future cumulative reward is optimized. The policy can be derived from the action value Q-function in Q-learning.

Complex-valued Q-learning is based on the iteration:

$$\dot{Q}(o_t, a_t) \leftarrow (1 - \alpha)\dot{Q}(o_t, a_t) + \alpha\left(r_{t+1} + \gamma \dot{Q}_{\max}(t)\right)\dot{\beta}, \quad (2)$$

where $\dot{Q}$ is the complex-valued observation-action value function. The dot notation $\dot{X}$ for some quantity X is used throughout the manuscript to remind us that the quantity is complex-valued. o_t and a_t are the observation and action at time t, respectively. The learning rate $\alpha \in (0, 1)$ is a real number. The reward r_t is a real number. $\gamma \in (0, 1)$ is the

discounted rate, which is a real number. $\dot{\beta} = e^{i\omega}$ is a complex hyperparameter for some real constant ω. $\dot{Q}_{\max}(t)$ is defined as

$$\begin{aligned} \dot{Q}_{\max}(t) &= \dot{Q}(o_{t+1}, a), \\ a &= \arg\max_{a_t \in A} \Re\left[\dot{Q}(o_{t+1}, a')\bar{\dot{I}}_t\right], \\ \dot{I}_t &= \frac{\dot{Q}(o_t, a_t)}{\dot{\beta}}. \end{aligned} \tag{3}$$

Here, $\bar{\dot{I}}_t$ means the complex conjugation of $\dot{I}_t$. $\Re$ denotes taking the real part of a complex number. The learned time series is reproduced by updating each complex number in the opposite phase direction. An eligibility trace method is further implemented such that the action value function is updated according to

$$\begin{aligned} \dot{Q}(o_{t-k}, a_{t-k}) \leftarrow (1-\alpha)\dot{Q}(o_{t-k}, a_{t-k}) \\ + \alpha\left(r_{t+1} + \gamma\dot{Q}_{\max}(t)\right)\dot{u}_t(k), \end{aligned} \tag{4}$$

where $\dot{u}_t(k) = \dot{\beta}^{k+1}$ for $0 \le k < N_e$ and N_e is the trace length. This update rule is exact for the table where the complete $\dot{Q}$-table must be stored in the memory. In the variational approaches, the $\dot{Q}$ function is variationally optimized by minimizing the functional:

$$L_\theta = \frac{1}{2}\left|\dot{Q}(o_{t-k}, a_{t-k}) - \left(r_{t+1} + \gamma\dot{Q}_{\max}(t)\right)\dot{u}_t(k)\right|^2, \tag{5}$$

with gradient descent $\theta \leftarrow \theta - \alpha(\partial L/\partial\theta)$. Finally, the policy is the Boltzmann stochastic policy:

$$\pi(o_t, a) = \frac{\mathrm{Exp}\left(\Re\left[\dot{Q}(o_t, a)\bar{\dot{I}}_{t-1}\right]/T\right)}{\sum_a \mathrm{Exp}\left(\Re\left[\dot{Q}(o_t, a)\bar{\dot{I}}_{t-1}\right]/T\right)}, \tag{6}$$

where T is a temperature hyperparameter.

2.2. Variational Quantum Circuit for Q-Function. The $\dot{Q}$ function is constructed by the expectation value:

$$\begin{aligned} \dot{Q}_\theta(o, a) &= \gamma_Q \langle 0 | U_{\text{encoder}}(\theta_{\text{in}}(o, a))^\dagger W(\theta)^\dagger U_{\text{out}}(\theta_{\text{out}}, a) W(\theta) U_{\text{encoder}}(\theta_{\text{in}}(o, a)) | 0 \rangle \\ &= \gamma_Q \langle \psi_{\text{out}}(\theta, \theta_{\text{in}}(o, a)) | U_{\text{out}}(\theta_{\text{out}}, a) | \psi_{\text{out}}(\theta, \theta_{\text{in}}(o, a)) \rangle. \end{aligned} \tag{7}$$

where γ_Q is a scaling constant. θ is the set of trainable variational parameters for trainable unitary $W(\theta) = \prod_{i=1}^{l}(U_{\theta_{l-i+1}} U_{\text{ent}}) U_{\theta_0}$. The quantity $(l+1)$ will be called the circuit depth. $\theta_{\text{in}}(o, a)$ is a set of input parameters which encode the observation o and action a. $\theta_{\text{in}}(o, a)$ parameterizes the input layer local unitaries. $U_{\text{out}}(\theta_{\text{out}}, a)$ is an output unitary to be measured by Hadamard test [34] against the output wave function ψ_{out}. $U_{\text{out}}(\theta_{\text{out}}, a)$ is parameterized by some trainable θ_{out} and the action a. The circuit structure is depicted in Figure 1. We implement three types of encoding schemes, which are summarized in the following paragraphs.

In Type 1 and Type 3 quantum circuits, the input layer is $U_{\text{encoder}} = \otimes_{i=0}^{n-1} R_Z(\theta_{\text{in}}^Z) R_Y(\theta_{\text{in}}^Y)$, where the index i means the local rotation is acting on i-th qubit. The parameters θ_{in}^Z and θ_{in}^Y are uniform for all the qubits (independent of qubit index i). For Type 1 circuit, the encoding is $\theta_{\text{in}}^Y = \sin^{-1}(\overline{o})$ and $\theta_{\text{in}}^Z = \cos^{-1}(\overline{o}^2)$, where $\overline{o} = 2((o - \min o)/(\max o - \min o)) - 1$ is the rescaled observation.

For Type 3 circuit, the encoding is $\theta_{\text{in}}^Y = \sin^{-1}(\overline{o})$ and $\theta_{\text{in}}^Z = \cos^{-1}(\overline{a})$, where $\overline{a} = 2((a - \min a)/(\max a - \min a)) - 1$. In Type 2 quantum circuit, the input layer is $U_{\text{encoder}} = \otimes_{i=0}^{n-1} R_Z(\theta_{i,\text{in}}^Z) R_X(\theta_{i,\text{in}}^X)$, where the subscript i means the local rotation is acting on i-th qubit. The encoding is done by binary representation of observation $o = b_0 b_1 b_2 b_3$ for binary numbers $b_i \in \{0, 1\}$. The observation is then encoded as state $|b_0 b_1 b_2 b_3\rangle$ with some phase factors by choosing $\theta_{i,in}^Z = \pi b_i$ and $\theta_{i,in}^X = \pi b_i$.

For the output unitary, Type 3 uses $U_{\text{out}}(\theta_{\text{out}}) = \otimes_{i=0}^{n-1} R_Z(\theta_{i,\text{out}}^Z) R_Y(\theta_{i,\text{out}}^Y) R_Z(\theta_{i,\text{out}}^{Z'})$, where the subscript i means the local rotation is acting on i-th qubit. All the $\theta_{i,\text{out}}^{Z'}$, $\theta_{i,\text{out}}^Y$, and $\theta_{i,\text{out}}^Z$ are trainable. For Type 1 and Type 2, we use $U_{\text{out}}(\theta_{\text{out}}, a) = \otimes_{i \in \{0, a+1\}} R_Z(\theta_{i,a,\text{out}}^Z) R_Y(\theta_{i,a,\text{out}}^Y) R_Z(\theta_{i,a,\text{out}}^{Z'})$ for an action $a \in A$. Figure 2 depicts the quantum circuits encoding used in this work.

All circuit parameters are updated by gradient descent with loss function (5). The gradient of this loss function with respect to each parameter is calculated by the back propagation method on a classical simulator. As quantum circuit calculates the output state by the dot product of the gate matrix and the input state, this calculation is a special case of complex-valued neural networks. Therefore, back propagation method of complex-valued neural networks [35] can be used in the back propagation of quantum circuit.

The gradient of loss function (5) with respect to $\dot{Q}$ function is calculated by

$$\frac{\partial L_\theta}{\partial \dot{Q}_\theta(o_{t-k}, a_{t-k})} = \dot{Q}_\theta(o_{t-k}, a_{t-k}) - \dot{Q}_{\text{Target}}, \tag{8}$$

where $\dot{Q}_{\text{Target}} = (r_{t+1} + \gamma\dot{Q}_{\max}(t))\dot{u}_t(k)$.

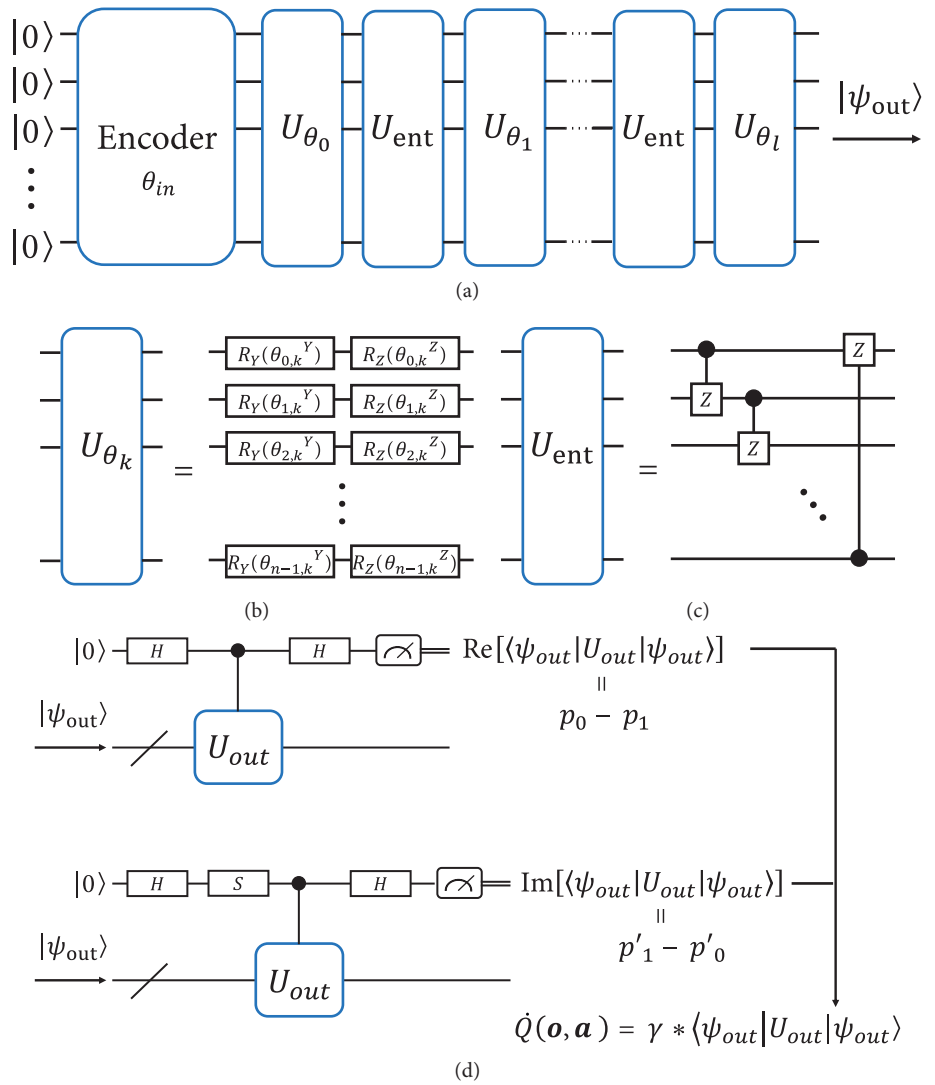


FIGURE 1: The variational quantum circuit. (a) The variational circuit. The encoder encodes action and observation with parameters θ_{in}. The output state of the circuit ψ_{out} will be measured against the output unitary to approximate the $\dot{Q}$ function. (b) Trainable local unitaries. (c) The entangler. (d) The Hadamard test for output measurement.

Using the definition $|\psi_{\text{out}}{}'(\theta_{\text{out}})\rangle = U_{\text{out}}(\theta_{\text{out}}, a_{t-k})|\psi_{\text{out}}(\theta, \theta_{\text{in}}(o_{t-k}, a_{t-k}))\rangle$, the gradient calculation of expectation value (7) is as follows:

$$\frac{\partial L_\theta}{\partial|\psi_{\text{out}}{}'(\theta_{\text{out}})\rangle} = \frac{\partial L_\theta}{\partial\dot{Q}_\theta(o_{t-k}, a_{t-k})}\overline{\gamma_Q}|\psi_{\text{out}}(\theta, \theta_{\text{in}}(o_{t-k}, a_{t-k}))\rangle,$$
$$\frac{\partial L_\theta}{\partial|\psi_{\text{out}}(\theta, \theta_{\text{in}}(o_{t-k}, a_{t-k}))\rangle} = \frac{\partial L_\theta}{\partial\dot{Q}_\theta(o_{t-k}, a_{t-k})}\overline{\gamma_Q}U_{\text{out}}(\theta_{\text{out}}, a_{t-k})^\dagger|\psi_{\text{out}}(\theta, \theta_{\text{in}}(o_{t-k}, a_{t-k}))\rangle + \overline{\frac{\partial L_\theta}{\partial\dot{Q}_\theta(o_{t-k}, a_{t-k})}}\gamma_Q|\psi_{\text{out}}{}'(\theta_{\text{out}})\rangle. \tag{9}$$

Here, gradient $(\partial L_\theta/\partial\dot{Q}_\theta(o_{t-k}, a_{t-k}))$ is gotten by equation (8). Figure 3(a) shows this gradient calculation. The gradients of output $|\psi_{\text{out}}{}'(\theta_{\text{out}})\rangle$ and $|\psi_{\text{out}}(\theta, \theta_{\text{in}}(o_{t-k}, a_{t-k}))\rangle$ are gotten. As the output $|\psi_{\text{out}}{}'(\theta_{\text{out}})\rangle$ is gotten by applying some gates with parameters θ_{out} to input $|\psi_{\text{out}}(\theta, \theta_{\text{in}}(o_{t-k}, a_{t-k}))\rangle$, the gradient of the parameters θ_{out} are gotten by back propagating $(\partial L_\theta/\partial|\psi_{\text{out}}{}'(\theta_{\text{out}})\rangle)$. Similarly, as the output $|\psi_{\text{out}}(\theta, \theta_{\text{in}}(o_{t-k}, a_{t-k}))\rangle$ is gotten by applying some gates with parameters θ to input $U_{\text{encoder}}(\theta_{\text{in}}(o_{t-k}, a_{t-k}))|0\rangle$, the gradient of the parameters θ are calculated by back

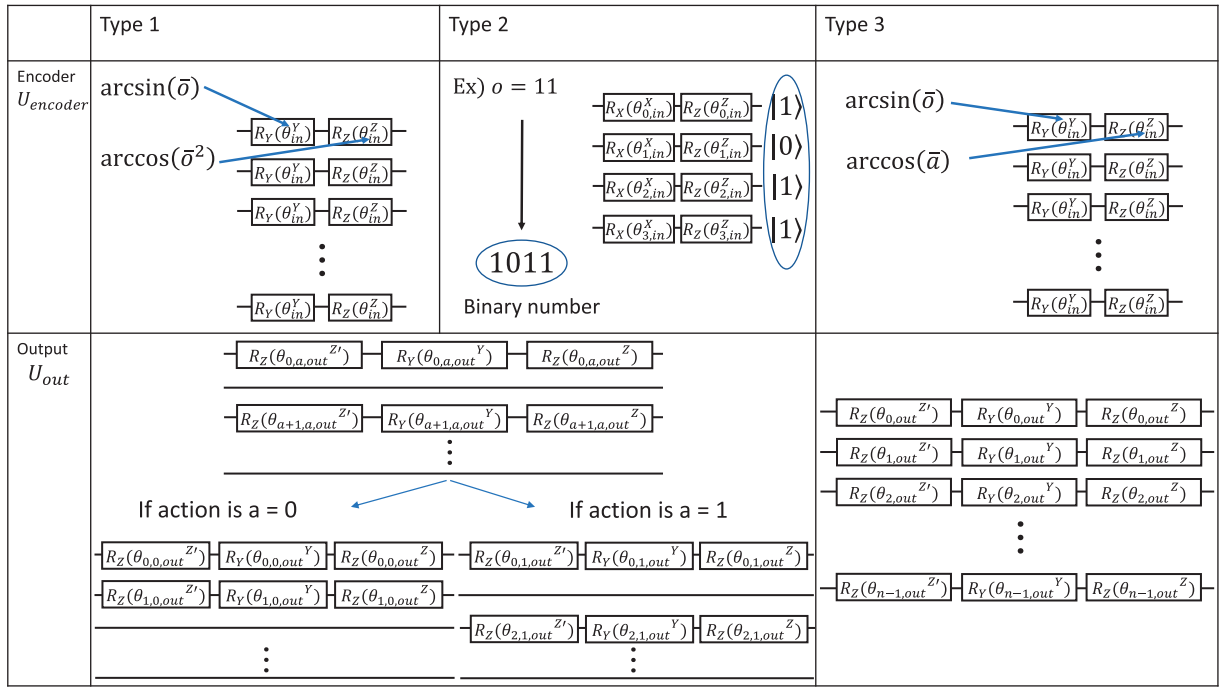
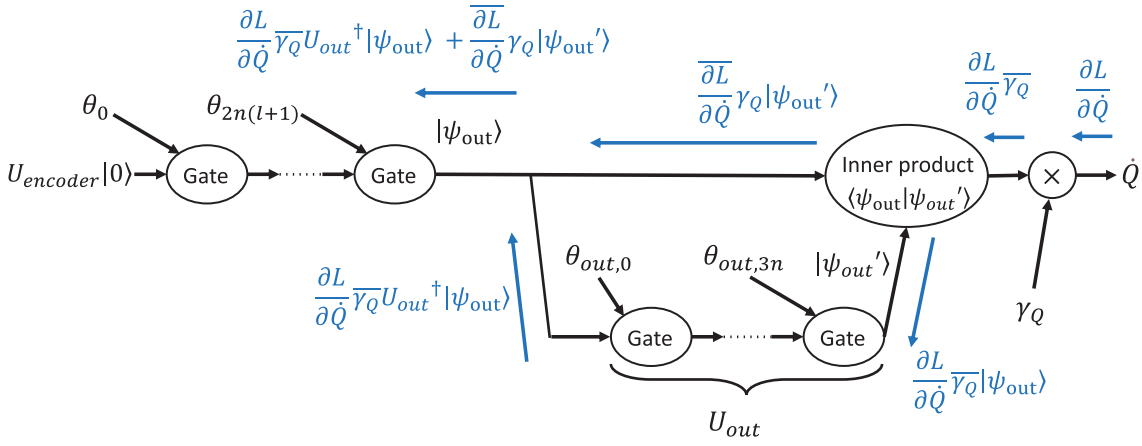


Figure 2: Three types of input encoders and output operators used in our experiments. In Type 1 circuit, the observation is encoded in the encoder, and the action is encoded by the output unitary. In Type 2 circuit, the observation is encoded in the encoder by binary state representation, and the action is encoded by the output unitary. In Type 3 circuit, both the observation and the action are encoded in the encoder.



(a)

Figure 3: Continued.

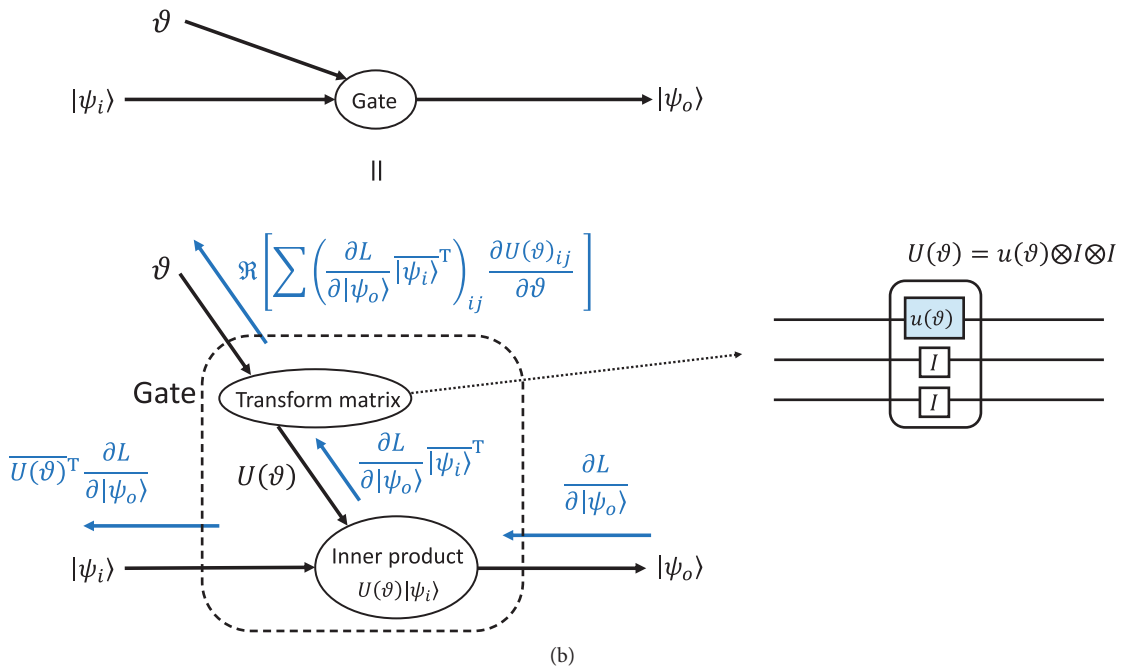


FIGURE 3: Back propagation for circuit parameters update. Black line is feed forward calculation, and blue line is back propagation. (a) Back propagation of expectation value calculation. The gradients with respect to $|\psi_{\text{out}}\rangle$ and $|\psi'_{\text{out}}\rangle$ are calculated. The former gradient is back propagated to update θ, and the latter is back propagated to update θ_{out}. (b) Back propagation of calculation applying a quantum gate. The calculation of applying a gate $u(\vartheta)$ to 0-th qubit is shown. The gradient with respect to $|\psi_i\rangle$ is then back propagated to the previous gate calculation.

propagating $(\partial L_\theta/\partial|\psi_{\text{out}}(\theta, \theta_{\text{in}}(o_{t-k}, a_{t-k}))\rangle)$. These back propagations are the almost same as the calculation of neural network back propagation, but using the quantum gates is different point. When a gate $u(\vartheta)$ with parameter ϑ is applied to input $|\psi_i\rangle$, the output $|\psi_o\rangle$ is gotten by $|\psi_o = U(\vartheta)|\psi_i\rangle$, where $U(\vartheta)$ is the transformed matrix from gate matrix $u(\vartheta)$ to calculate the inner product. The gradient is gotten by

$$\frac{\partial L}{\partial U(\vartheta)} = \frac{\partial L}{\partial|\psi_o\rangle} \cdot \overline{|\psi_i\rangle}^T,$$
$$\frac{\partial L}{\partial|\psi_i\rangle} = \overline{U(\vartheta)}^T \cdot \frac{\partial L}{\partial|\psi_o\rangle}, \tag{10}$$

where $\partial L/\partial|\psi_o\rangle$ is the gradient of loss function L with respect to the output $|\psi_o\rangle$. The gradient with respect to parameter ϑ is calculated by

$$\frac{\partial L}{\partial \vartheta} = \Re\left[\sum\left(\frac{\partial L}{\partial|\psi_o} \cdot \overline{|\psi_i\rangle}^T\right)_{ij} \frac{\partial U(\vartheta)_{ij}}{\partial \vartheta}\right], \tag{11}$$

where M_{ij} represents the (i, j) element of the matrix M. Figure 3(b) shows the back propagation of the gate calculation. The gradient of loss with respect to the last output is back propagated by equation (10), and parameters are updated by gradient of equation (11).

2.3. Neural Networks. For experimental comparison, we also implement complex-valued neural networks. The detail for the architecture of the neural networks is presented in this section. We use two types of neural networks for experiments. For both Type 1 and Type 2 neural networks, there are 3 layers: one input layer, one hidden layer, and one output layer. In Type 1 neural network, there are two input layer neurons and one output layer neuron. Both the action a and the observation o are encoded in the input layer neurons. The output neuron then gives $\dot{Q}(o, a)$. In Type 2 neural network, there is one input neuron and $|A|$ output neurons. Only the observation o is encoded in the input layer neuron. Each output neuron then gives $\dot{Q}(o, a)$ for an action a. For both Type 1 and Type 2 neural networks, there is one hidden layer, and there are 30 neurons in the hidden layer. The networks are depicted in Figure 4. We use the learning rate 0.0001 for the hidden layer and 0.001 for the output layer. We defined the architecture type 1 based on the paper [13] and the architecture type 2 for comparison. We update the parameters by gradient descent using back propagation [35]. In complex-valued neural networks, fully connected layer is calculated by

$$z = f_c(\mathbf{o}),$$
$$\mathbf{o} = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}. \tag{12}$$

where $\mathbf{x}$ is the input vector. $\mathbf{W}$ is the weight matrix. $\mathbf{b}$ is the bias vector. $\mathbf{o}$ is the weighted sum of input. $\mathbf{z}$ is the output vector. They are all complex numbers. $f_c(\mathbf{v})$ is the activation function for complex numbers and defined by

$$f_c(\mathbf{v}) = f(\Re[\mathbf{v}]) + if(\Im[\mathbf{v}]), \tag{13}$$

where $f(u)$ is activation function for real numbers.

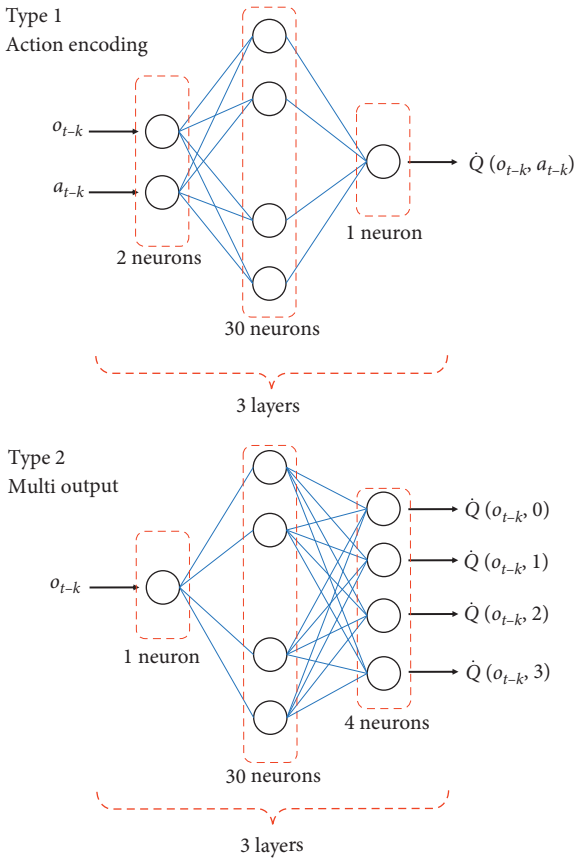


FIGURE 4: Neural networks for the experiments. In Type 1 neural network, both the action and the observation are encoded through the input layer. In Type 2 neural network, only the observation is encoded in the input layer, and the value of $\dot{Q}(o, a)$ is given by one output neuron for each action a.

Back propagation equation to calculate gradient of loss function L is

$$\frac{\partial L}{\partial \mathbf{o}} = \Re\left[\frac{\partial L}{\partial \mathbf{z}}\right]\left(\frac{\partial \Re[\mathbf{z}]}{\partial \Re[\mathbf{o}]} + i\frac{\partial \Re[\mathbf{z}]}{\partial \Im[\mathbf{o}]}\right) + \Im\left[\frac{\partial L}{\partial \mathbf{z}}\right]\left(\frac{\partial \Im[\mathbf{z}]}{\partial \Re[\mathbf{o}]} + i\frac{\partial \Im[\mathbf{z}]}{\partial \Im[\mathbf{o}]}\right),$$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{o}} \cdot \overline{\mathbf{x}}^T,$$

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{o}},$$

$$\frac{\partial L}{\partial \mathbf{x}} = \overline{\mathbf{W}}^T \cdot \frac{\partial L}{\partial \mathbf{o}}, \tag{14}$$

where $\overline{a}$ means the complex conjugation of a. $(\partial L/\partial \mathbf{z})$ is the gradient of loss L with respect to output vector $\mathbf{z}$ of this layer. The parameters are updated by gradient descent using these gradients.

2.4. The Maze Environment. The partially observable maze environment used in our experiments is depicted in Figure 5. We defined the environment with reference to the maze environments of the paper [11, 12]. In particular, the

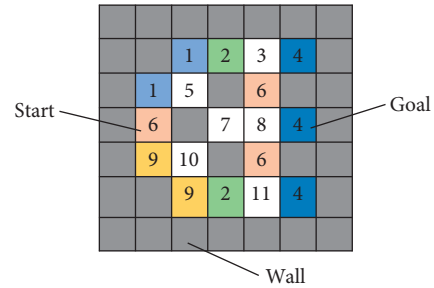


FIGURE 5: The partially observable maze environment. This is a 2D maze with 4 possible moving directions. Each cell denotes on possible state (location). The dark cells denote the wall which is not available to the agent. A number (and a color) in a cell denotes the corresponding observation received by the agent at that state. The start state and goal state are indicated.

structure is the same as in the maze of the paper [12]. Each cell in the maze represents a possible state. The agent starts from the start state and aims for the goal state. There are four possible actions $A = \{\longrightarrow, \leftarrow, \uparrow, \downarrow\} = \{0, 1, 2, 3\}$. The wall of the maze is represented by cells in dark gray color. If the agent executes an action which makes the agent hitting the wall, the agent's state is not transitioned. The numbers in cells are the observations $\Omega = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$. The cells painted in the same color are different states with the same observation (except for the white color). The reward is +100 if the agent reaches the goal state, and 0 otherwise. Each episode ends either if the agent reaches the goal state, or the number of time steps reaches a maximum value.

3. Results and Discussion

We perform numerical simulations to compare the learning curves of different approaches. Figure 6(a) shows the learning curves for Q-table (blue solid line with circle marker), complex neural networks (orange solid line and green dashed line), quantum circuit (red dotted line, purple dashed-dotted line, and brown solid line with "+" cross marker), and long short-term memory (LSTM, pink solid line with "x" cross marker). The vertical axis is the number of steps to the goal, so lower value means a better policy. The data is the average of 50 independent runs. Each episode has maximum number of steps 1000. The total number of episodes is 5000, and the data is further smoothed out by taking average over 500 sequential episodes. In all the $\dot{Q}$ learning experiments, the trace length is $N_e = 6$. For LSTM, the history sequence length is 6. All the quantum circuits are using 6 qubits with depth = 3. All the neural networks have one hidden layer consisting of 30 neurons. We observe that the $\dot{Q}$-table method has the most stable learning curve, and the learned policy is able to reach the goal with lower number of steps. This is expected since $\dot{Q}$-table does not use any approximation in the representation of $\dot{Q}$-function. Type 3 quantum circuit gives bad result, where the learned policy does not really improve the reaching time to the goal. The performance of Type 1 quantum circuits is not significantly different comparing to other classical complex

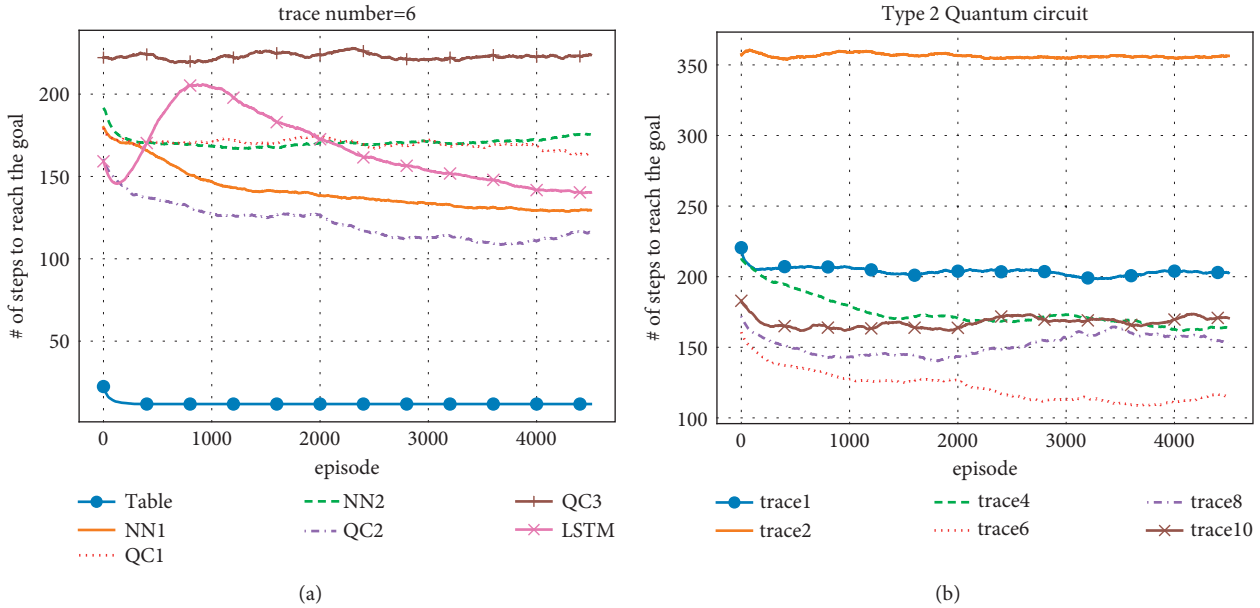


FIGURE 6: Learning curves comparison for different algorithms and different trace numbers from simulation. The data is the average of 50 independent runs, and the data is smoothed out by taking average over 500 sequential episodes. (a) Comparison for different algorithms. Blue solid line with circle marker is $\dot{Q}$-table. Orange solid lines are Type 1 neural network. Green dashed line is Type 2 neural network. Red dotted line is Type 1 quantum circuit. Purple dashed-dotted line is Type 2 quantum circuit. Brown solid line with "+" marker is Type 3 quantum circuit. Pink solid line with "x" marker is LSTM. We observe that Type 2 quantum circuit provides the second-best performance. (b) Comparison for different trace number for Type 2 quantum circuit. Blue solid line with circle marker, orange solid line, green dashed line, red dotted line, purple dashed-dotted line, and brown solid line with cross marker correspond to trace number 1, 2, 4, 6, 8, and 10, respectively. We observe better performance for trace number around $N_e = 6$.

neural network approaches. We note that Type 2 quantum circuit provides the best result among all the $\dot{Q}$ approximation schemes. It is even better than LSTM approach in our experiments.

We compare the results for various hyperparameters. Figure 6(b) shows the learning curves for Type 2 quantum circuit with different trace number. We observe that the performance could be improved by using trace number 4 (green dashed line) rather than trace number 1 (blue solid line with circle marker) or 2 (orange solid line). However, the performance of trace number 10 (brown solid line with cross marker) is not good either. We observe the best learning curves for the trace number N_e around 6 (red dotted line).

Since the best performance could be obtained by Type 2 circuit with trace number around 6, we then fix the circuit type to Type 2 and trace number to 6 and compare the learning curves for different widths and depths. Figure 7(a) shows the learning curves for various circuit depths. It is observed that the learning curve could be improved by increasing circuit depth from 1 (blue solid line), 2 (orange dashed line), 3 (green dotted line), to 4 (red solid line with circle marker) for $n = 6$. Figure 7(b) shows the learning curves for various circuit widths with fixed circuit depth = 4. We observe that higher circuit width makes the learning task more difficult. The learning curve for $n = 5$ (blue solid line) is better than that of $n = 6$ (orange dashed line) and $n = 7$ (green dotted line).

We then take an offline learning scheme to compare prediction performances of different machines. That is, the parameters are obtained from state-vector simulator-based training process, and the predictions are done using various different methods. The test results are depicted in Figure 8. We test the predictions by using a Numpy-based state-vector simulator [36], the QASM simulator provided by Qiskit [37], and the IBMQ system ibmq_guadalupe [38]. The horizontal axis "episode" indicates the number of episodes for training. Hence, episodes = 1000 means that 1000 episodes training is performed, and then the learned parameters are used for the corresponding prediction experiment. Each data point is the average of 5000 runs. The 5000 runs are conducted by 50 independently learned parameters, and for each learning, there are 100 prediction experiments. The number of shots for Hadamard-test estimation is 4096 for the QASM simulator. The quantum circuit type is Type 2. The number of qubits is 5, and circuit depth is 4. The trace number is 6. For the IBM Q system, a set of parameters trained with 5000 episodes is used for a prediction experiment. The number of shots is 1024. Five experiments are executed, and the number of steps to reach the target state is 67 steps, 1000 steps, 502 steps, 1000 steps, and 1000 steps, respectively. Since the maximum number of steps is 1000, "1000 steps" means that the agent does not reach the target in the experiment.

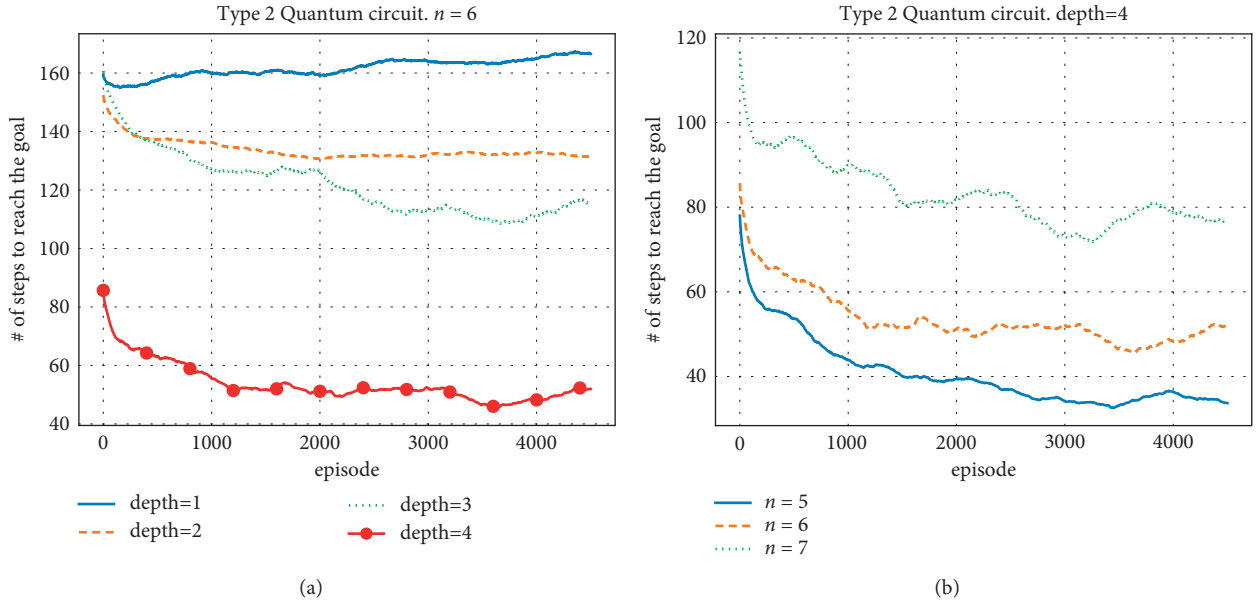


Figure 7: Learning curves comparison for different circuit widths and depths. The data is the average of 50 independent runs, and the data is smoothed out by taking average over 500 sequential episodes. (a) Comparison of different circuit depths for Type 2 quantum circuit with $n = 6$. Blue solid line, orange dashed line, green dotted line, and red solid line with circle marker are lines for depth = 1, 2, 3, 4, respectively. (b) Comparison of different circuit widths for Type 2 quantum circuit, depth = 4. Blue solid line, orange dashed line, and green dotted line are lines for $n = 5, 6, 7$, respectively.

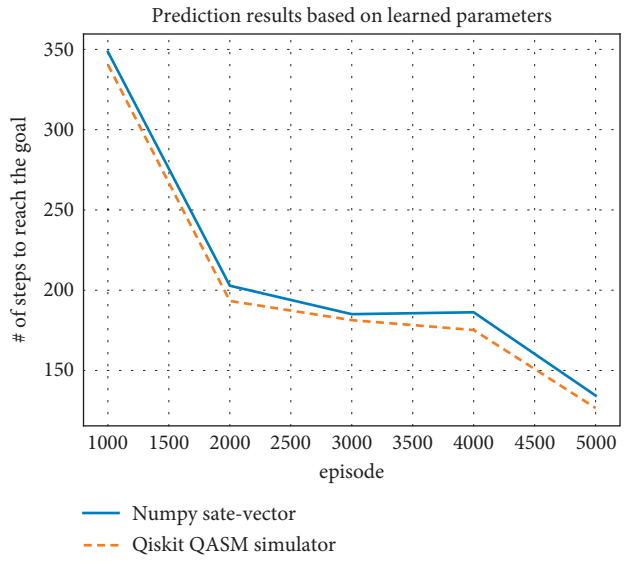


Figure 8: Prediction test results using different prediction methods with state-vector simulator-based offline training. The blue-solid line is the result predicted by the state-vector simulator, while the orange-dashed line is the prediction result given by Qiskit QASM simulator with 4096 shots.

4. Conclusions

In this work, we propose the quantum circuit algorithm for POMDP based on the complex-valued Q learning. We implemented several encoding schemes and compare them to other classical approaches by numerical simulations. The observed learning curve suggests that, with suitable encoding, the learning efficiency of quantum complex-valued Q learning can be better than other classical methods like complex-valued neural networks. The performance of quantum circuit can be further improved by suitable choice of hyper-parameters. The learned parameters from simulators are then tested with IBMQ systems. The agent is able to reach the goal state with predictions made by real quantum machines. Our results provide a new method for POMDP problems with potential quantum advantage. The partially observable maze environment experiment executed in this work is a discrete state environment. For future works, our proposed method could be applied to the other simulation problems, such as partially observable continuous space environment Mountain Car [4, 39]. Since the encoding scheme showing better result in this work is only used for discrete space, to solve continuous problem without discretization, another new encoding scheme for continuous environment will be needed.

Data Availability

The code used to support the findings of this study has been deposited in the GitHub repository https://github.com/tomo920/QC-ComplexValuedRL.

Disclosure

The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Q team.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors gratefully acknowledge the funding from New Energy and Industrial Technology Development Organization (NEDO) under grant number NEDO P20015. The authors acknowledge use of the IBM Q for this work. The authors gratefully acknowledge the funding from New Energy and Industrial Technology Development Organization (NEDO).

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] A. Geramifard, T. J. Walsh, S. Tellex, G. Chowdhary, N. Roy, and J. P. How, "A tutorial on linear function approximators for dynamic programming and reinforcement learning," *Foundations and Trends in Machine Learning*, vol. 6, no. 4, pp. 375–451, 2013.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA, 2018.

[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*, The MIT Press, Cambridge, MA, USA, 2005.

[6] S. Russell and N. Peter, *Artificial Intelligence: A Modern Approach*, Pearson Education, London, UK, 4th edition, 2021.

[7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[8] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," in *Proceedings of the 23rd International Conference on Neural Information Processing Systems ser. NIPS'10*, vol. 2, pp. 2164–2172, Curran Associates Inc., Red Hook, NY, USA, December 2010.

[9] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," 2015, https://arxiv.org/abs/1507.06527.

[10] P. Zhu, X. Li, P. Poupart, and G. Miao, "On improving deep reinforcement learning for POMDPs," 2017, https://arxiv.org/abs/1704.07978.

[11] T. Hamagami, T. Shibuya, and S. Shimada, "Complex-valued reinforcement learning," in *Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 4175–4179, Taipei, Taiwan, October 2006.

[12] T. Shibuya, S. Shimada, and T. Hamagami, "Experimental study of the eligibility traces in complex valued reinforcement learning," in *Proceedings of the 2007 IEEE International Conference on Systems, Man and Cybernetics*, pp. 1630–1635, Montréal, Canada, October 2007.

[13] M. Mochida, H. Nakano, and A. Miyauchi, "A complex-valued reinforcement learning method using complex-valued neural networks," *IEICE Technical Report*, vol. 117, no. 112, pp. 1–5, 2017.

[14] J. Bassey, L. Qian, and X. Li, "A survey of complex-valued neural networks," 2021, https://arxiv.org/abs/2101.12249.

[15] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[16] A. Peruzzo, J. McClean, P. Shadbolt et al., "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. 1, p. 4213, 2014.

[17] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, Article ID 023023, 2016.

[18] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Physical Review A*, vol. 98, no. 3, Article ID 032309, 2018.

[19] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, "Variational quantum circuits for deep reinforcement learning," *IEEE Access*, vol. 8, pp. 141007–141024, 2020.

[20] M. Moll and L. Kunczik, "Comparing quantum hybrid reinforcement learning to classical methods," *Human-Intelligent Systems Integration*, vol. 3, no. 1, pp. 15–23, 2021.

[21] O. Lockwood and M. Si, "Reinforcement learning with quantum variational circuits," 2020, https://arxiv.org/abs/2008.07524.

[22] A. Skolik, S. Jerbi, and V. Dunjko, "Quantum agents in the Gym: a variational quantum algorithm for deep Q-learning," 2021, https://arxiv.org/abs/2103.15084.

[23] O. Lockwood and M. Si, "Playing atari with hybrid quantum-classical reinforcement learning," 2021, https://arxiv.org/abs/2107.04114.

[24] S. Y.-C. Chen, C.-M. Huang, C.-W. Hsing, H.-S. Goan, and Y.-J. Kao, "Variational quantum reinforcement learning via evolutionary optimization," 2021, https://arxiv.org/abs/2109.00540.

[25] S. Jerbi, C. Gyurik, S. Marshall, H. J. Briegel, and V. Dunjko, "Variational quantum policies for reinforcement learning," 2021, https://arxiv.org/abs/2103.05577.

[26] Y. Kwak, W. J. Yun, S. Jung, J.-K. Kim, and J. Kim, "Introduction to quantum reinforcement learning: theory and PennyLane-based implementation," 2021, https://arxiv.org/abs/2108.06849.

[27] S. Wu, S. Jin, D. Wen, and X. Wang, "Quantum reinforcement learning in continuous action space," 2020, https://arxiv.org/abs/2012.10711.

[28] S. Jerbi, L. M. Trenkwalder, H. P. Nautrup, H. J. Briegel, and V. Dunjko, "Quantum enhancements for deep reinforcement learning in large spaces," *PRX Quantum*, vol. 2, Article ID 010328, 2021.

[29] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: an anytime algorithm for POMDPs," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, pp. 1025–1032, Acapulco, Mexico, August 2003.

[30] S. Ross, B. Chaib-draa, and J. Pineau, "Bayes-adaptive POMDPs," in *Proceedings of the Advances in Neural Information Processing Systems*, vol. 20, Vancouver, Canada, December 2007.

[31] P. Poupart and N. Vlassis, "Model-based bayesian reinforcement learning in partially observable domains," in *Proceedings of the 10th International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, Fort Lauderdale, FL, USA, January 2008.

[32] F. Doshi-Velez, D. Pfau, F. Wood, and N. Roy, "Bayesian nonparametric methods for partially-observable reinforcement learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 2, pp. 394–407, 2015.

[33] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," 2015, https://arxiv.org/abs/1512.04455.
[34] D. Aharonov, V. Jones, and Z. Landau, "A polynomial quantum algorithm for approximating the Jones polynomial," in *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing (STOC '06)*, pp. 427–436, Association for Computing Machinery, New York, NY, USA, 2006.
[35] T. Nitta, "An extension of the back-propagation algorithm to complex numbers," *Neural Networks*, vol. 10, no. 8, pp. 1391–1415, 1997.
[36] C. R. Harris, K. J. Millman, S. J. van der Walt et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.
[37] G. Aleksandrowicz, "Qiskit: an open-source framework for quantum computing (Version 0.7.2): Zenodo," 2019.
[38] "IBM quantum," 2021, https://quantum-computing.ibm.com/ .
[39] G. Brockman, V. Cheung, L. Pettersson et al., "OpenAI gym," 2016, https://arxiv.org/abs/1606.01540.