# Quantum reinforcement learning in continuous action space

Shaojun Wu, Shan Jin, Dingding Wen, Donghong Han, and Xiaoting Wang†

Institute of Fundamental and Frontier Sciences, University of Electronic Science and Technology of China, Chengdu, 610051, China



**Quantum reinforcement learning (QRL) is a promising paradigm for near-term quantum devices. While existing QRL methods have shown success in discrete action spaces, extending these techniques to continuous domains is challenging due to the curse of dimensionality introduced by discretization. To overcome this limitation, we introduce a quantum Deep Deterministic Policy Gradient (DDPG) algorithm that efficiently addresses both classical and quantum sequential decision problems in continuous action spaces. Moreover, our approach facilitates single-shot quantum state generation: a one-time optimization produces a model that outputs the control sequence required to drive a fixed initial state to any desired target state. In contrast, conventional quantum control methods demand separate optimization for each target state. We demonstrate the effectiveness of our method through simulations and discuss its potential applications in quantum control.**

## 1 Introduction

Reinforcement learning (RL) [1] plays a vital role in machine learning. Unlike supervised and unsupervised learning to find data patterns, the idea of RL is to reduce the original problem into finding a good sequence of decisions leading to an optimized long-term reward, through the interaction between an agent and an environment. This feature makes RL advantageous for solving a wide range of sequential decision problems, including game-playing [2, 3], e.g., AlphaGo [4], robotic control [5, 6], quantum error correction [7, 8] and quantum control [9–15]. Typical RL algorithms include Q-learning [16, 17], Deep Q-Network(DQN) [3, 18], and Deep Deterministic Policy Gradient(DDPG) [19]. Despite its broad applications, the implementation of RL on classical computers becomes intractable as the problem size grows exponentially, such as the cases from quantum physics. Analogous to quantum computation for conventional computational problems [20], quantum machine learning has been proposed to solve machine learning problems on quantum computers to potentially gain an exponential or quadratic speedup [21–28]. To date, a variety of quantum reinforcement learning (QRL) algorithms have been developed, leveraging different quantum paradigms to explore potential computational advantages [29]. Early works on implementing RL on quantum circuits demonstrated a quadratic speedup by leveraging Grover's algorithm [30–35]. Inspired by deep neural networks, several VQC-based approaches have been proposed, following either value-based [36–40] or policy-based [41–44] reinforcement learning frameworks. These methods are designed to be compatible with near-term quantum devices, leveraging parameterized quantum circuits for efficient function approximation. In addition to VQC-based approaches, researchers have also explored QRL methods for fault-tolerant quantum computers, some of which utilize oracle-based access to improve computational efficiency [31, 45–48]. Another direction of QRL research is projective simulation, which extends classical reinforcement learning by employing a memory network and quantum random walks, potentially accelerating action selection and offering quantum advantages [49–53]. One interesting open question is whether a quantum reinforcement learning(QRL) algorithm can be constructed to guarantee an exponential speedup over its classical counterpart in terms of gate complexity. Besides,

Xiaoting Wang†: xiaoting@uestc.edu.cn

another interesting question is how to design the QRL algorithm so that it can efficiently and effectively solve RL problems in *continuous action space*(CAS) without the curse of dimensionality due to discretization, especially the decision problems on quantum systems. In this work, inspired by the classical DDPG algorithm, we propose a quantum DDPG method that can be applied to the quantum state generation in the continuous action space. Specifically, for a fixed state $|s_d\rangle$, we first train the agent to design a parametrized unitary sequence $\{U_a(\boldsymbol{\theta}_t)\}$ that will sequentially drive any state $|s_0\rangle$ into $|s_d\rangle$, where $|s_0\rangle$ can be any state and the action parameters $\boldsymbol{\theta}_t$ take values from a continuous domain and $t = 0, \cdots, T$. In our method, the agent's policy and the value function for QRL are constructed from variational quantum neural networks(QNN). The optimal policy function is obtained by continuously optimizing the policy QNN and the value QNN. Once the training is completed, the optimal policy QNN generates the desired sequence $\{U_a(\boldsymbol{\theta}_t)\}$ that transforms any $|s_0\rangle$ to $|s_d\rangle$. Due to the reversibility of unitary operations, the inverse sequence $\{U_a^\dagger(\boldsymbol{\theta}_t)\}$ can be applied to drive $|s_d\rangle$ back to $|s_0\rangle$, where $|s_0\rangle$ can be any state. In particular, if the target state $|s_0\rangle = |s_{\text{unknown}}\rangle$ is unknown but multiple copies are available, one may use the reversed control sequence to efficiently generate $|s_{\text{unknown}}\rangle$ from the fixed initial state $|s_d\rangle$. Hence, our method enables single-shot quantum state generation: a one-time optimization produces a model that outputs the control sequence required to drive a fixed initial state to any desired target state. In contrast, conventional state generation methods typically require separate optimization for each different target state. Moreover, when the target state is unknown, traditional approaches require state tomography to first identify the unknown target state before optimization. In the following, we will first provide a brief introduction to RL and then propose our own QRL algorithm.

## 2 Classical reinforcement learning

In artificial intelligence, an agent is a mathematical abstraction representing an object with learning and decision-making abilities. It interacts with its environment, which includes everything except the agent. The core idea of RL is, through

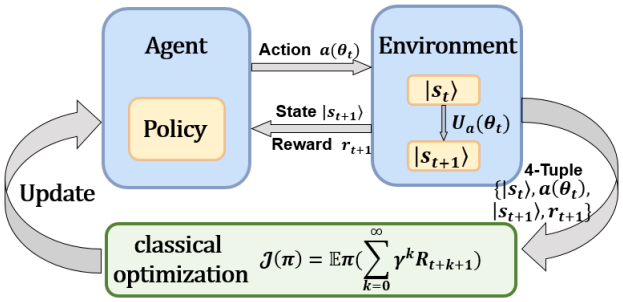


Figure 1: The QRL model. Each iterative step can be described by the following loop: (1) at step $t$, the agent receives $|s_t\rangle$ and generates the action parameter $\boldsymbol{\theta}_t$ according to the current policy; (2) the agent generates $|s_{t+1}\rangle \equiv U_a(\boldsymbol{\theta}_t)|s_t\rangle$; (3) based on $|s_t\rangle$ and $|s_{t+1}\rangle$, a reward $r_{t+1}$ is calculated and fed back to the agent, together with $|s_{t+1}\rangle$; (4) based on $|s_{t+1}\rangle$ and $r_{t+1}$, the policy is updated and then used to generate $\boldsymbol{\theta}_{t+1}$.

the iterative interactions, the agent learns and selects actions, and the environment responds to these actions, by updating its state and feeding it back to the agent. In the meanwhile, the environment also generates rewards, which are some value-functions the agent aims to maximize over its choice of actions along the sequential interactions [1].

Most reinforcement learning problems can be described by a Markov Decision Process (MDP) [1, 54] with basic elements including a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$ and the reward $\mathcal{R}$. The agent interacts with its environment at each of a sequence of discrete time steps, $t = 0, 1, \cdots, T$. Each sequence like this generated in RL is called an *episode*. At each time step $t$, the agent receives a representation of the environment's state, denoted by an $N$-dimensional vector $s_t \in \mathcal{S}$, based on which it then chooses an action $a_t \in \mathcal{A}$, resulting the change of the environment's state from $s_t$ to $s_{t+1}$. At the next step, the agent receives the reward $r_{t+1}$ determined by the 3-tuple $(s_t, a_t, s_{t+1})$. The aim of the agent is to find a policy $\pi$ that maximizes the cumulative reward $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$, where $\gamma$ is a discount factor, $0 \leq \gamma \leq 1$. A large discount factor $\gamma$ means that the agent cares more about future rewards. The policy can be considered as a mapping from $\mathcal{S}$ to $\mathcal{A}$. The update of the policy $\pi$ is achieved by optimizing the value function $Q(s_t, a_t) \equiv E[R_t|s_t, a_t]$, i.e., the expectation of $R_t$ under the policy $\pi$. Depending on whether the action space is discrete or continuous, the RL problems can be classified into two

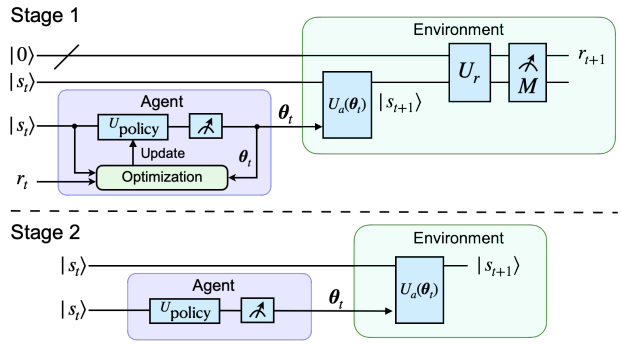


Figure 2: The quantum circuit for our QRL framework at each iteration. The entire QRL process includes two stages, so we give the circuit separately. In stage 1, the circuit includes two registers: the reward register, initialized $|0\rangle$, and the environment register $|s_t\rangle$. $U_{\text{policy}}$ is generated by the quantum neural network, and determines the action unitary $U_a(\boldsymbol{\theta}_t)$. $U_r$ and $M$ are designed to generate the reward $r_{t+1}$. In stage 2, the circuit has only environment register and does not need to feedback the reward value and update the policy.

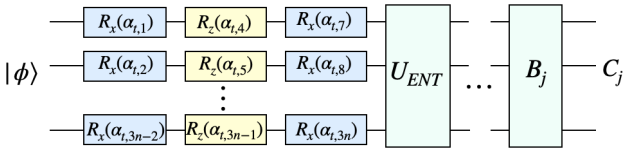

Figure 3: Circuit architecture for the VQC. $R_\beta(\alpha) \equiv \exp(-i\sigma_\beta\alpha/2)$ with $\beta = x, z$. $U_{\text{ENT}} \equiv \prod_{k=1}^{n-1} \text{CNOT}_{(k,k+1)}$, where $\text{CNOT}_{(k,k+1)}$ denotes the CNOT gate using the $k$-th qubit to control the $(k+1)$-th qubit. $C_j$ is the outcome of the measurement on the observable $B_j$, $j = 1, 2, \cdots$.

categories: DAS(*discrete action space*) and CAS, with different algorithmic design to update the agent's policy. For DAS problems, popular RL algorithms includes Q-learning [16], Sarsa [55], DQN [18], etc.; for CAS problems, popular algorithms include Policy Gradient [56], DDPG [19], etc.

## 3 The framework of quantum reinforcement learning

In order to construct a quantum framework that works for both CAS and DAS cases, we present the following QRL model, as shown in Fig. 1. The essential idea is to map the elements of classical RL into the quantum counterparts. We introduce a quantum 'environment' register to represent the environment in RL, and its quantum state $|s_t\rangle$ to represent the classical state $s_t$ at time step $t$. Then the action $a(\boldsymbol{\theta}_t)$ can be represented as a parameterized action unitary $U_a(\boldsymbol{\theta}_t)$ on $|s_t\rangle$, where $\boldsymbol{\theta}_t$ is the action parameter, which is continuous for CAS case, and takes values from a finite set for DAS case. In order to generate the quantum reward function, by introducing a reward register $|r_t\rangle$, we design the reward unitary $U_r$ and the measurement observable $M$ such that $r_{t+1} \equiv f(\langle s_t|\langle 0|U_a^\dagger(\boldsymbol{\theta}_t)U_r^\dagger M U_r U_a(\boldsymbol{\theta}_t)|0\rangle|s_t\rangle)$ will match the actual reward defined by the RL problem. Here, $f$ is a function determined by the problem and $|0\rangle$ is the initial state of the reward register. It will be clear in the context of a concrete problem how to design $M$, $U_r$, and $f$ in the correct way, which will be discussed in detail based on the quantum state generate problem and the eigenvalue problem in the following.

With all RL elements represented as the components of a quantum circuit shown in Fig. 2, it remains to show how to find the optimal policy $\boldsymbol{\theta}_t = \boldsymbol{\pi}(|s_t\rangle)$ at each time step $t$, such that the iterative sequence $U_{\text{tot}} = U_a(\boldsymbol{\theta}_T)\cdots U_a(\boldsymbol{\theta}_1)U_a(\boldsymbol{\theta}_0)$ will drive the initial state $|s_0\rangle$ converging to the state $|s_d\rangle$. The entire QRL process can be divided into two stages. In stage 1, we construct the optimal policy $U_{\text{policy}}$ through the agent training, including the policy update and the value-function estimation, which can be realized through the function fitting using QNNs. In stage 2, under the established optimal policy, we can iteratively generate the desired sequence $\{U_a(\boldsymbol{\theta}_t)\}$ that will drive the initial state to the target state, and complete the RL task.

In our method, in order to solve RL problems in CAS, we utilize QNNs to construct the policy function and the value function. One popular way of implementing a QNN is to use the variational quantum circuit (VQC) [36, 57–59], whose parameters can be iteratively optimized for the given objective function on classical computers. The VQC circuit of our quantum DDPG algorithm consists of a sequence of unitary $\{D^{(k)}(\boldsymbol{\alpha})\}$, and is ended by measurements of observables $\{B_j\}$ with $\text{Tr}(B_iB_j) = 0$ (Fig.3), where $B_j = b_{j,1} \otimes b_{j,2} \otimes \cdots \otimes b_{j,n}$ and $b_{j,i} \in \{\sigma_x, \sigma_y, \sigma_z, I\}$. The choice of $\{B_j\}$ are not unique, usually we will choose a set of appropriate $\{B_j\}$ in order to improve the trainability of the quantum neural network. Each $D^{(k)}(\boldsymbol{\alpha})$ can be chosen to have an identical structure, $D^{(k)}(\boldsymbol{\alpha}) \equiv U_{\text{ENT}}V^{(k)}(\boldsymbol{\alpha})$, where $V^{(k)}(\boldsymbol{\alpha}) \equiv$

$\otimes_{l=1}^{n}(R_x(\alpha_{k,3l-2})R_z(\alpha_{k,3l-1})R_x(\alpha_{k,3l}))$, $U_{\rm ENT} \equiv \prod_{k=1}^{n-1} {\rm CNOT}_{(k,k+1)}$ and ${\rm CNOT}_{(k,k+1)}$ uses the $k$-th qubit to control the $(k+1)$-th qubit. Here, $R_{x,z}$ are rotations, with $R_\beta(\alpha) \equiv \exp(-i\sigma_\beta\alpha/2)$, $\beta = x, z$. For the input $|\phi\rangle$, the output of the VQC can be expressed as the expected measurement outcome $C_j \equiv \langle\phi|D^\dagger(\boldsymbol{\alpha})B_jD(\boldsymbol{\alpha})|\phi\rangle$, based on which the parameter $\boldsymbol{\alpha}$ can then be optimized for the given optimization problem, on a classical computer.

## 4 Quantum DDPG algorithm

For RL problems in CAS, we aim to design QNNs to iteratively construct a sequence of unitary gates that will drive the environment register from the initial state eventually to the target state. This is the essential idea of the quantum DDPG algorithm. Analogous to the classical DDPG algorithm, we make use of the QNNs to construct the desired policy function $\pi_{\boldsymbol{\eta}}$ and the value function $Q_{\boldsymbol{\omega}}$. Specifically, the policy-QNN is used to approximate the policy function $\theta_{t,j} \equiv \langle s_t|D^\dagger(\boldsymbol{\eta})B_jD(\boldsymbol{\eta})|s_t\rangle$ with $\boldsymbol{\theta}_t = (\theta_{t,1}, \theta_{t,2}, \dots)$, and the Q-QNN is used to approximate the value function $Q(|s_t\rangle, \boldsymbol{\theta}_t) \equiv \langle\boldsymbol{\theta}_t|\langle s_t|D^\dagger(\boldsymbol{\omega})B_QD(\boldsymbol{\omega})|s_t\rangle|\boldsymbol{\theta}_t\rangle$. Here, it should be noted that the input of Q-QNN is $|s_t\rangle|\boldsymbol{\theta}_t\rangle$, so we first need encode $\boldsymbol{\theta}_t$ into quantum state $|\boldsymbol{\theta}_t\rangle$ by the amplitude encoding method. For example, we can encode the action parameters as $|\boldsymbol{\theta}_t\rangle = \sum_{j=1}^{n}\theta_{t,j}|j\rangle$. In order to make the training more stable, two more target networks are included with the same structure as the two main networks [3, 18, 19]. Therefore, the quantum DDPG uses four QNNs in total: (1) the policy-QNN $\pi_{\boldsymbol{\eta}}(|s_t\rangle)$, (2) the Q-QNN $Q_{\boldsymbol{\omega}}(|s_t\rangle, \boldsymbol{\theta}_t)$, (3) the target-policy $\pi'_{\boldsymbol{\eta'}}(|s_t\rangle)$ and (4) the target-Q $Q'_{\boldsymbol{\omega'}}(|s_t\rangle, \boldsymbol{\theta}_t)$.

The quantum DDPG method contains two stages. In stage 1, the agent training is divided into three parts: (1) experience replay [60], (2) updates of the Q-QNN and the policy-QNN, and (3) updates of the target networks. The aim of the experience replay is to prevent the neural network from overfitting. We store the agent's experiences $(|s_t\rangle, \boldsymbol{\theta}_t, r_t, |s_{t+1}\rangle)$ in a finite-sized replay buffer $\boldsymbol{D}$ at each time step. During the training, we randomly sample a batch of experiences from the replay buffer to update the Q-QNN and the policy-QNN. First, we update the policy-QNN parameters by minimizing the expected return $J = E[Q_{\boldsymbol{\omega}}(|s\rangle, \boldsymbol{\theta})|_{|s\rangle=|s_i\rangle, \boldsymbol{\theta}=\pi(|s_i\rangle)}]$. Then we update the Q-QNN parameters by minimizing the mean-squared loss $L = \frac{1}{G}\sum_i(y_i - Q_{\boldsymbol{\omega}}(|s_i\rangle, \boldsymbol{\theta}_i))^2$ between the predicted Q-value and the original Q-value, where $y_i = r_i + \boldsymbol{\gamma}Q'_{\boldsymbol{\omega'}}(|s_{i+1}\rangle, \pi'_{\boldsymbol{\eta'}}(|s_{i+1}\rangle))$ is the predicted Q-value and calculated by the target-Q networks, $G$ is the size of the batch. Here, we use the gradient descent algorithm AdamOptimizer [61] to minimize the loss function of these two quantum neural networks. Finally, we update the two target networks using a soft update strategy [19]: $\boldsymbol{\omega'} \leftarrow \tau\boldsymbol{\omega} + (1-\tau)\boldsymbol{\omega'}$, $\boldsymbol{\eta'} \leftarrow \tau\boldsymbol{\eta} + (1-\tau)\boldsymbol{\eta'}$, where $\tau$ is a parameter with $0 < \tau < 1$. The entire training process is summarized in Algorithm 1. In stage 2, with the optimal policy-QNN and $T$ iterations, we can construct a sequence of $\{U_a(\boldsymbol{\theta}_t)\}$ and $|s_t\rangle$ for each given initial $|s_0\rangle$, satisfying $|s_T\rangle$ is sufficiently close to the target $|s_d\rangle$.

**Algorithm 1** Quantum DDPG algorithm

Randomly initialize $Q_{\boldsymbol{\omega}}(|s\rangle, \boldsymbol{\theta})$ and $\pi_{\boldsymbol{\eta}}(|s\rangle)$;
Initialize target $Q'$ and $\pi'$;
Initialize replay buffer $\boldsymbol{D}$;
**for** episode=1, M **do**
  Prepare the initial state $|0, s_0\rangle$;
  **for** t=1:$T$ **do**
    Select the actions: $\boldsymbol{\theta}_t = \pi_{\boldsymbol{\eta}}(|s_t\rangle)$;
    Apply $U_a(\boldsymbol{\theta}_t)$: $|s_{t+1}\rangle = U_a(\boldsymbol{\theta}_t)|s_t\rangle$;
    Apply $U_r$ and $M$ to obtain $r_{t+1}$;
    Store tuple $(|s_t\rangle, \boldsymbol{\theta}_t, r_t, |s_{t+1}\rangle)$ in $\boldsymbol{D}$;
    Sample a batch of tuples $(|s_i\rangle, \boldsymbol{\theta}_i, r_i, |s_{i+1}\rangle)$ from $\boldsymbol{D}$;
    Set $y_i = r_i + \boldsymbol{\gamma}Q'_{\boldsymbol{\omega'}}(|s_{i+1}\rangle, \pi'_{\boldsymbol{\eta'}}(|s_{i+1}\rangle))$;
    Update Q-QNN by minimizing the loss: $L = \frac{1}{G}\sum_i(y_i - Q_{\boldsymbol{\omega}}(|s_i\rangle, \boldsymbol{\theta}_i))^2$;
    Update the policy-QNN: $\nabla_{\boldsymbol{\eta}}J \approx \frac{1}{G}\sum_i \nabla_{\boldsymbol{\theta}}Q_{\boldsymbol{\omega}}(|s_i\rangle, \boldsymbol{\theta_i})\nabla_{\boldsymbol{\eta}}\pi_{\boldsymbol{\eta}}(|s_i\rangle)$;
    Update the target QNNs: $\boldsymbol{\omega'} \leftarrow \tau\boldsymbol{\omega}+(1-\tau)\boldsymbol{\omega'}$; $\boldsymbol{\eta'} \leftarrow \tau\boldsymbol{\eta}+(1-\tau)\boldsymbol{\eta'}$.
  **end for**
**end for**

For DAS problems, the above QRL proposal still works if the quantum DDPG design in Fig. 2 is replaced by the quantum DQN design, analogous to the classical DQN algorithm [18]. Compared with the quantum DDPG, the quantum DQN maps states of the environment into the

computational basis, rather than into the amplitudes of a quantum register. Moreover, for quantum DQN, only the value function needs to be approximated by the QNN, while the policy function can be described by the greedy algorithm [1]. Detailed proposals to solve DAS problems using QNNs are presented in [36, 37]. It is worthwhile to note that the quantum DQN cannot efficiently solve CAS problems, since the dimensionality problem is inevitable when solving the CAS problems through discretization.

## 5 Application of Quantum DDPG in quantum state generation problems

The quantum state generation problem is a very important ingredient in quantum computation and quantum control. The standard definition of a state generation problem is as follows: given a desired target state that is challenging to prepare, the goal is to design a unitary evolution that drives an easily prepared initial state into the target state. In general, solving this problem requires a case-by-case approach, as any change in the target state necessitates a redesign of the corresponding unitary transformation. An natural question arises: is it possible to develop a unified framework that, for any given target state, automatically generates the appropriate unitary transform to achieve this evolution? The answer is yes. Our proposed method accomplishes precisely this. Our approach begins by designing a process that ensures any given initial state evolves into an easily prepared target state. Then, by swapping the roles of the initial and target states and reversing the process, we construct a general framework that drives the easily prepared initial state into the given target state—regardless of what that target state is.

Specifically, our algorithm has different goals during the training phase (Stage 1) and the execution phase (Stage 2). In stage 1, the purpose of quantum neural network training is to find an optimal policy network that can generate a sequence of $\{U_a(\boldsymbol{\theta}_t)\}$ to drive any given initial state $|s_0\rangle$ to a fixed target state $|s_d\rangle$. Here, the choice of $|s_d\rangle$ is not unique, in fact, in order to realize the measurement of the reward, we will choose some quantum states that are easy to prepare, such as $|0\rangle^{\otimes n}$. In stage 2, leveraging the reversibility of the unitary operation, the inverse sequence $\{U_a^\dagger(\boldsymbol{\theta}_t)\}$ drives the easily prepared initial state $|s_d\rangle$ to the target state $|s_0\rangle$, for any given $|s_0\rangle$. Thus, we have completed the task of state generation. In particular, if the target state is unknown but multiple copies are available, the reversed control sequence can still be applied to efficiently generate it. This highlights the 'single-shot' feature of our approach: a one-time optimization yields a model that outputs the desired control sequence to drive a fixed initial state into any desired target state.

The QRL circuit for the state generation is shown in Fig. 2. First, we define the environment $U_r$, and at each step $t$, we define the reward function $r_{t+1} = f(p_{t+1})$. Given the training target $|s_d\rangle$, we choose an observable $M_d \equiv |s_d\rangle\langle s_d|$. Then we can obtain an estimate of $p_{t+1} \equiv \langle M_d\rangle = \langle s_{t+1}|M_d|s_{t+1}\rangle$ through the measurement statistics of $M_d$, with estimation error $\epsilon$. Notice that the number of measurements required to obtain $p_{t+1}$ and $r_{t+1} = f(p_{t+1})$ is independent of the system size $N = 2^n$ of the $n$-qubit environment register, and the proof will be shown later in the following. Let $|s_0\rangle = \sum_{k=1}^{N} \alpha_{0,k}|v_k\rangle$ be the initial state of the environment register, where $\alpha_{0,k} = \langle v_k|s_0\rangle$. At time step $t$, applying $U_{\text{policy}}$ and quantum measurements on $|s_t\rangle$, as shown in Fig. 2, we obtain the action parameter $\boldsymbol{\theta}_t$, and generate the corresponding action unitary $U_a(\boldsymbol{\theta}_t) \equiv U_{\text{ENT}}V(\boldsymbol{\theta}_t)$, where $V(\boldsymbol{\theta}_t)$ and $U_{\text{ENT}}$ are defined as in Fig. 3. Then we apply $U_a(\boldsymbol{\theta}_t)$ and get

$$|s_{t+1}\rangle = U_a(\boldsymbol{\theta}_t)|s_t\rangle = U_{\text{ENT}}V(\boldsymbol{\theta}_t)|s_t\rangle. \quad (1)$$

Next, by measuring $M_d$, we obtain an estimation of

$$p_{t+1} \equiv \langle s_{t+1}|M_d|s_{t+1}\rangle = |\langle s_{t+1}|s_d\rangle|^2 \quad (2)$$

through $K$ number of measurements. For this state generation problem, we choose the reward $r_{t+1} = 10p_{t+1}$. If $p_{t+1} \to 1$ as $t \to \infty$, then $|s_{t+1}\rangle$ will converge to $|s_d\rangle$ to complete the RL goal. Notice that the evaluation of $r_{t+1}$ is only needed for the QRL training stage; in the application stage, since $U_{\text{policy}}$ is optimized, measurements are no longer required to estimate $r_{t+1}$. In addition, we will use the same quantum state $|s_t\rangle$ more than once in the algorithm, as shown in Fig. 2. Since the action parameters $\boldsymbol{\theta}_t$ can be recorded, we can

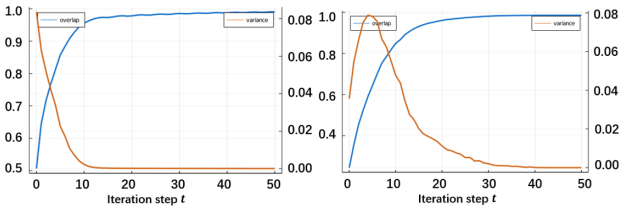


Figure 4: Simulation results for quantum state generation problem of the one-qubit and the two-qubit Hamiltonian by quantum DDPG. For 1000 different initial $|s_0\rangle$, we plot how the average $\bar{p}_t$ and the variance $\Delta(p_t)$ change with the iteration step $t$. For the one-qubit case, at $t = 50$, $\bar{p}_{50} \geq 0.99$ and $\Delta(p_{50}) \leq 4.47 \times 10^{-5}$. For the two-qubit case, at $t = 50$, $\bar{p}_{50} \geq 0.98$ and $\Delta(p_{50}) \leq 4.04 \times 10^{-7}$.

obtain multiple copies of $|s_t\rangle$ by repeating preparation.

To verify the effectiveness of our method, we study the state generation problem for one-qubit and two-qubit cases. In stage 1, we apply the quantum DDPG algorithm to update the policy until we obtain the optimal $U_{\text{policy}}$. Since the interaction between agent and environment is influenced by noise, we randomly choose the error $\epsilon \sim \mathcal{N}(p_t, \epsilon^2)$ on the reward value in the simulation. In stage 2, based on $U_{\text{policy}}$ and $|s_0\rangle$, we generate a sequence of $\{U_a(\boldsymbol{\theta}_t)\}$ and the corresponding $|s_{50}\rangle = U_a(\boldsymbol{\theta}_{49}) \cdots U_a(\boldsymbol{\theta}_0)|s_0\rangle$, and record the overlap $p_t$ at each $t$.

Specifically, for the one-qubit case, the training target state is chosen to be $|1\rangle$. In stage 1, the size of the replay buffer is set as 1000, the size of the batch is set as 16, and the other parameters are set as $\gamma = 0.9$, $\tau = 0.001$. In addition, the quantum registers for the policy-QNN and the value-QNN are both comprised of three qubits, and the depth of the QNN circuits is one. For the policy-QNN, in order to increase the nonlinearity of the QNN, we use auxiliary qubits. Specifically, we add two ancilla qubits initialized in $|00\rangle$ together with $|s_t\rangle$ as the input of the policy network. Usually, the size of the auxiliary qubits is one to two times the number of qubits in the environment state $|s_t\rangle$. For the value-QNN, we encode $\boldsymbol{\theta}_t$ into $|\boldsymbol{\theta}_t\rangle$ and use it together with $|s_t\rangle$ as the inputs of the value-QNN. Then we perform the training process. After the training stage, we randomly select 1000 different initial states $|s_0\rangle$ to test the performance of the policy $U_{\text{policy}}$. In Fig. 4, we can see that almost all final states $|s_{50}\rangle$ are sufficiently close to the target $|s_d\rangle$, with $p_{50} \geq 0.99$ and $\Delta(p_{50}) \leq 4.67 \times 10^{-5}$ at $t = 50$. For the two-qubit case, the target state $|s_d\rangle$ is chosen to be the maximum entangled state $(|00\rangle + |11\rangle)/\sqrt{2}$. As the number of qubits of the environment state increases, the training of the neural network will become more difficult. In our simulation, using the original quantum neural network training can not get satisfactory results. In order to address this problem, we apply the universal quantum neural network proposed in Ref. [62], which introduced a classical activation function and a weight matrix on the basis of VQC. The QNNs with this structure has the ability to approximate arbitrary functions. Also, we set the size of the replay buffer as 10000, the size of the batch as 128, and other parameters as $\gamma = 0.9$, $\tau = 0.005$. Here, the quantum registers to implement these two QNNs both contain six qubits, and the depth of both QNN circuits is three. Simulation results in Fig. 4 show that $\bar{p}_{50} \geq 0.98$ and $\Delta(p_{50}) \leq 4.04 \times 10^{-7}$ at $t = 50$. Notice that for both one-qubit and two-qubit cases, a one-shot optimization is sufficient to find the optimal policy $U_{\text{policy}}$ through QNN learning. Once the QRL model is constructed, for each $|s_0\rangle$, it can efficiently generate the desired control sequence $\{U_a(\boldsymbol{\theta}_t)\}_{t=0}^{49}$ to drive $|s_0\rangle$ to $|s_d\rangle$. If we take $|s_d\rangle$ as the initial state, then the reversed sequence $\{U_a^\dagger(\boldsymbol{\theta}_t)\}_{t=49}^{0}$ will drive $|s_d\rangle$ to different $|s_0\rangle$. Thus, we have completed the task of any given state generation. It is worthwhile to note that it is not necessary for $|s_0\rangle$ to be known to make our model work: even if $|s_0\rangle$ is unknown, as long as we have sufficient number of identical copies of $|s_0\rangle$, our model is still able to output the desired sequence that will drive $|s_d\rangle$ to $|s_0\rangle$.

## 6 Quantum DDPG in solving the eigenvalue problem

In quantum complexity theory, the Quantum Merlin Arthur(QMA) is the set of all decision problems that can be verified by quantum computers in polynomial time [63–65]. One typical QMA-complete problem is the $k$-local Hamiltonian problem, which is to find the ground energy of a $k$-local Hamiltonian $H$ with $k \geq 2$ [66]. Essentially, this problem is an eigenvalue problem for a given physical Hamiltonian, and one way of solving it on a quantum computer is to use the variational quantum eigensolver (VQE)

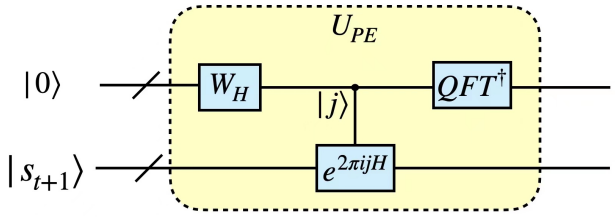


Figure 5: The quantum phase estimation circuit $U_{\rm PE}$.

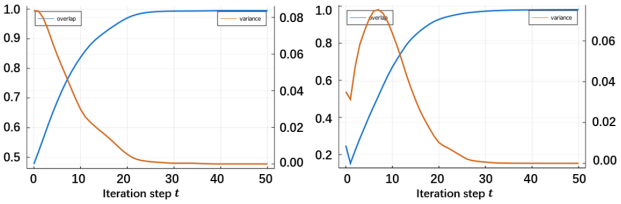


Figure 6: Simulation results for solving the eigenvalue problem of the one-qubit and the two-qubit Hamiltonian by quantum DDPG. For 1000 different initial $|s_0\rangle$, we plot how the average $\bar{p}_t$ and the variance $\Delta(p_t)$ change with the iteration step $t$. For the one-qubit case, at $t = 50$, $\bar{p}_{50} \geq 0.99$ and $\Delta(p_{50}) \leq 1.98 \times 10^{-7}$. For the two-qubit case, at $t = 50$, $\bar{p}_{50} \geq 0.98$ and $\Delta(p_{50}) \leq 3.01 \times 10^{-6}$.

algorithm [57]. Here, we present an alternative method, formulating the $k$-local Hamiltonian problem as a reinforcement learning problem in CAS and applying our quantum DDPG algorithm to it. Specifically, let $H$ be the Hamiltonian defined on $N$ dimensional quantum system $E$, and $H$ is a sparse matrix that can be efficiently constructed. Assuming an unknown eigenvalue $\lambda_0$ of $H$ is located in a neighborhood of $\bar{\lambda}$, i.e. $\lambda_0 \in \delta(\bar{\lambda}) \equiv [\bar{\lambda} - \delta, \bar{\lambda} + \delta]$, we aim to find out $\lambda_0$ and its corresponding eigenvector $|u_0\rangle$. To implement the QRL circuit in Fig. 2 for the eigenvalue problem, we choose $U_r$ as the quantum phase estimation $U_{\rm PE}$ shown in Fig. 5. The role of $U_{\rm PE}$ together with the subsequent measurement is to map the input state $|s_{t+1}\rangle$ into the desired eigenstate with certain probability. Specifically, the reward function $r_{t+1}$ can be defined as the overlap between the $(t+1)$-th states with $|u_0\rangle$: $r_{t+1} \equiv 10|\langle s_{t+1}|u_0\rangle|^2$. Let $|0\rangle$ and $|s_0\rangle = \sum_{k=1}^{N} \alpha_{0,k}|u_k\rangle$ be the initial states of the reward register and the $n$-qubit environment register, where $n = \log N$ and $\alpha_{0,k} = \langle u_k|s_0\rangle$. At the time step $t$, applying $U_{\rm policy}$ and quantum measurements to $|s_t\rangle$, we obtain the action parameter $\boldsymbol{\theta}_t$, and the next state is $|s_{t+1}\rangle = U_a(\boldsymbol{\theta}_t)|s_t\rangle$.

Then, by applying $U_{\rm PE}$, we obtain

$$U_{\rm PE}|0\rangle|s_{t+1}\rangle = \sum_{k=1}^{N} \alpha_{t+1,k}|\lambda_k\rangle|u_k\rangle, \qquad (3)$$

where $|u_k\rangle$ is the eigenvector corresponding to the eigenvalue $\lambda_k$. Here, the input state of the eigenstate register is in a superposition of different eigenstates, so the output state becomes an entangled state between the eigenvalue phase register and the the eigenstate register. Therefore, in the next step, we only need to measure the eigenvalue phase register on the computational basis to obtain $|\langle s_{t+1}|u_0\rangle|^2$. Specifically, by measuring the eigenvalue phase register, we obtain the outcome $\lambda_0$ with probability

$$p_{t+1} \equiv |\langle s_{t+1}|u_0\rangle|^2 = |\alpha_{t+1,0}|^2 \qquad (4)$$

which can be estimated by the frequency of obtaining $\lambda_0$ in $K$ number of measurements. The reward can be written as $r_{t+1} = 10p_{t+1}$.

Next, as numerical demonstration, we apply this method to the Heisenberg model with $n = 1, 2$. In stage 1, the parameter settings of the quantum neural networks are the same as in the cases of $n = 1, 2$ in the state generation problem. After the training in stage 1, we randomly select 1000 different initial states $|s_0\rangle$ to test the policy $U_{\rm policy}$. For the one-qubit case, we choose the Hamiltonian as $H \equiv \frac{1}{4}(0.13\sigma_x + 0.28\sigma_y + 0.95\sigma_z + I)$, where the three coupling constants are randomly generated. One can see from Fig. 6 that as $t$ increases, trajectories $\{|s_t\rangle\}$ with different initial $|s_0\rangle$ all converge to the ground state $|u_0\rangle$, with $\bar{p}_{50} \geq 0.99$ and $\Delta(p_{50}) \leq 1.98 \times 10^{-7}$ at $t = 50$. Next, we consider a two-qubit Hamiltonian model $H = \frac{1}{2}(\sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y + \sigma_z \otimes \sigma_z) + 0.25\sigma_x \otimes I$, and use the quantum DDPG algorithm to find an energy state of it. The simulation result is shown in Fig. 6 that as $t$ increases, trajectories with different initial $|s_0\rangle$ all converge to the ground state of $H$, with $\bar{p}_{50} \geq 0.98$ and $\Delta(p_{50}) \leq 3.01 \times 10^{-6}$ at $t = 50$. However, we are not satisfied with the accuracy rate of 0.98, so we can use the final state $|s_{50}\rangle$ as the input of the method of Ref. [67] to obtain an exact state.

Again, analogous to the state generation problem, our QRL method demonstrates an advantage over the conventional optimal control [68]: the optimal $U_{\rm policy}$ generated through a one-shot optimization is useful for any given initial state, while a different optimization is required for each different initial state in optimal control or VQE.

## 7 Complexity analysis

Analogous to the VQE algorithm, our quantum DDPG algorithm is essentially a quantum-classical hybrid optimization. Similar to other discussion, we will mainly focus on the quantum circuit complexity of our algorithm, including the circuit gate complexity and the measurement complexity. In our method, quantum measurements are required to obtain both the reward value and the action parameters.

Let $B$ be an observable with $B|u_j\rangle = \lambda_j|u_j\rangle$, for an $n$-qubit system, $N = 2^n$. Assuming the system is in the state $|\psi\rangle = \sum_{j=1}^{N} \alpha_j|u_j\rangle$, the measurement of $B$ in $|\psi\rangle$ will generate the measurement outcome $m$, with probability distribution $P(m = \lambda_j) = |\langle u_j|\psi\rangle|^2 = |\alpha_j|^2$. Notice that $m$ and $B$ have the same expectation value and the variance: $\mathbb{E}(m) = \langle B\rangle$ and $\mathrm{Var}(m) = \langle B^2\rangle - \langle B\rangle^2$. Then we have the following well-known result in probability theory:

**Lemma 1** (Chebyshev's inequality)**.** *Let $X$ is a random variable with expected value $\mu$ and variance $\sigma^2$. For any real number $k > 0$, $P(|X-\mu| \geq k\sigma) \leq \frac{1}{k^2}$.*

Based on Lemma 1, we further derive the following relationship between the measurement precision error $\epsilon$ and the number of measurements $K$ to reach that precision.

**Theorem 1.** *Let $B$ be an observable with $B = b_1 \otimes b_2 \otimes \cdots \otimes b_n$, where $b_i \in \{\sigma_x, \sigma_y, \sigma_z, I\}$. We implemented measurements for $K$ times to obtain the sample average $\overline{m}_K$ to estimate the expected value $\langle B\rangle$. Then, the probability of the difference between $\overline{m}_K$ and $\langle B\rangle$ lager than $\epsilon$ is given by*

$$P(|\overline{m}_K - \langle B\rangle| \geq \epsilon) \leq \frac{1}{K\epsilon^2}. \tag{5}$$

*Proof.* After $K$ number of measurements on $B$, we obtain a sample of measurement outcomes, $m_1, m_2, \cdots, m_K$, with sample mean $\overline{m}_K \equiv \frac{1}{K}\sum_{i=1}^{K} m_i$ and $\{m_i\}$ to be independent and identically distributed. Let the expectation and the variance of $m_i$ be $\mu$ and $\sigma^2$. Due to the weak law of large numbers, we have $\overline{m}_K \to \mu$ for $n \to \infty$. Then the expectation of the sample mean is $\mathbb{E}(\overline{m}_K) = \mu = \langle B\rangle$ and the variance is $\mathrm{Var}(\overline{m}_K) = \frac{\sigma^2}{K}$. Choosing $k = \frac{\epsilon\sqrt{K}}{\sigma}$ in Lemma 1 and using Chebyshev's inequality result in $P(|\overline{m}_K - \langle B\rangle| \geq \epsilon) \leq \frac{\sigma^2}{K\epsilon^2}$.

For the observable $B = b_1 \otimes b_2 \otimes \cdots \otimes b_n$ and $b_i \in \{\sigma_x, \sigma_y, \sigma_z, I\}$, we have $\lambda_i = \pm 1$ and $\langle B^2\rangle$ is 1. Further, we have $0 \leq \langle B\rangle^2 \leq 1$ and $0 \leq \langle B^2\rangle - \langle B\rangle^2 \leq 1$. Hence, $0 \leq \sigma \leq 1$, and we find $P(|\overline{m}_K - \langle B\rangle| \geq \epsilon) \leq \frac{1}{K\epsilon^2}$. □

Now we apply Theorem 1 to analyze the measurement complexity of our algorithm. For a given estimation error $\epsilon$, according to Theorem 1, choosing $K = \mathcal{O}(\frac{1}{\epsilon^2})$ will make $\overline{m}_K$ converge to $\langle B\rangle$ with high probability. Next, we analyze the gate complexity of our algorithm. During a single iteration at $t$ of stage 1, denoting the gate complexities of $U_a(\boldsymbol{\theta_t})$, $U_{\text{policy}}$(policy-QNN) and $U_Q$(Q-QNN) as $C_a$, $C_p$ and $C_q$, we can see that the number of parameters $\boldsymbol{\theta_t}$ in $U_a(\boldsymbol{\theta_t})$ is at most equal to $C_a$. Hence, in a single iteration at $t$ of stage 1, the gate complexity is $\mathcal{O}(\frac{C_aC_p+C_a^2t+C_q}{\epsilon^2})$. Analogously, in a single iteration at $t$ of stage 2, the gate complexity is $\mathcal{O}(C_at) + \mathcal{O}(\frac{C_p+C_at}{\epsilon^2}C_a)$; hence, the total gate complexity of stage 2 is $\mathcal{O}(\frac{T^2C_a^2+C_aC_p}{\epsilon^2})$.

## 8 Concluding discussion

In this work, inspired by the classical DDPG algorithm, we have proposed a quantum DDPG algorithm that can solve both CAS and DAS reinforcement learning problems. For CAS tasks, our quantum DDPG algorithm encodes the environment state into the quantum state amplitude to avoid the dimensionality disaster due to discretization. As a useful application, our method can be used to solve the state generation problem. A distinguishing feature of our method is that, for each target state, it only requires a one-shot optimization to construct the QRL model that is able to efficiently output the desired control sequence driving the initial state to the target state. In comparison, in conventional quantum control methods, different optimizations are required for each different target state. Simulation results for one-qubit and two-qubit quantum systems demonstrate that, our QRL method can be used for any given state generation and eigenstate preparation for a given Hamiltonian. We have also analyzed the complexity of our proposal in terms of the QNN circuit complexity and the measurement complexity.

## Acknowledgments

This research was supported by the National Natural Science Foundation of China (Grant No.92265208) and the National Key R&D Program of China (Grant No.2018YFA0306703). We also thank Xiaokai Hou, Yuhan Huang, and Qingyu Li for helpful and inspiring discussions.

## References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[2] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature(London)*, 550(7676):354–359, 2017. DOI: 10.1038/nature24270. URL https://doi.org/10.1038/nature24270.

[3] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Graves Alex, Antonoglou Ioannis, Wierstra Daan, and Riedmiller Martin. Playing atari with deep reinforcement learning. 2013. DOI: 10.48550/ARXIV.1312.5602. URL http://arxiv.org/abs/1312.5602.

[4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature(London)*, 529(7587):484–489, 2016. DOI: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

[5] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003. DOI: 10.1109/LARS/SBR/WRE51543.2020.9307084. URL https://ieeexplore.ieee.org/document/9307084.

[6] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1329–1338, 2016. DOI: 10.5555/3045390.3045531. URL https://dl.acm.org/doi/10.5555/3045390.3045531.

[7] Hendrik Poulsen Nautrup, Nicolas Delfosse, Vedran Dunjko, Hans J. Briegel, and Nicolai Friis. Optimizing Quantum Error Correction Codes with Reinforcement Learning. *Quantum*, 3:215, December 2019. ISSN 2521-327X. DOI: 10.22331/q-2019-12-16-215. URL https://doi.org/10.22331/q-2019-12-16-215.

[8] Philip Andreasson, Joel Johansson, Simon Liljestrand, and Mats Granath. Quantum error correction for the toric code using deep reinforcement learning. *Quantum*, 3:183, September 2019. ISSN 2521-327X. DOI: 10.22331/q-2019-09-02-183. URL https://doi.org/10.22331/q-2019-09-02-183.

[9] Pantita Palittapongarnpim, Peter Wittek, Ehsan Zahedinejad, Shakib Vedaie, and Barry C. Sanders. Learning in quantum control: High-dimensional global optimization for noisy quantum dynamics. *Neurocomputing*, 268: 116 – 126, 2017. ISSN 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.12.087. URL http://www.sciencedirect.com/science/article/pii/S0925231217307531.

[10] Zheng An and D. L. Zhou. Deep reinforcement learning for quantum gate control. *EPL (Europhysics Letters)*, 126 (6):60002, jul 2019. DOI: 10.1209/0295-5075/126/60002. URL https://doi.org/10.1209/0295-5075/126/60002.

[11] Marin Bukov, Alexandre G. R. Day, Dries Sels, Phillip Weinberg, Anatoli Polkovnikov, and Pankaj Mehta. Reinforcement learning in different phases of quantum control. *Phys. Rev. X*, 8: 031086, Sep 2018. DOI: 10.1103/Phys-

RevX.8.031086. URL https://link.aps.org/doi/10.1103/PhysRevX.8.031086.

[12] Murphy Yuezhen Niu, Sergio Boixo, Vadim N Smelyanskiy, and Hartmut Neven. Universal quantum control through deep reinforcement learning. *npj Quantum Information*, 5(1):1–8, 2019. DOI: 10.1038/s41534-019-0141-3. URL https://doi.org/10.1038/s41534-019-0141-3.

[13] Han Xu, Junning Li, Liqiang Liu, Yu Wang, Haidong Yuan, and Xin Wang. Generalizable control for quantum parameter estimation through reinforcement learning. *npj Quantum Information*, 5(82): 1–8, 2019. DOI: 10.1038/s41534-019-0198-z. URL https://doi.org/10.1038/s41534-019-0198-z.

[14] Xiao-Ming Zhang, Zezhu Wei, Raza Asad, Xu-Chen Yang, and Xin Wang. When does reinforcement learning stand out in quantum control? a comparative study on state preparation. *npj Quantum Information*, 5 (85):1–7, 2019. DOI: 10.1038/s41534-019-0201-8. URL https://doi.org/10.1038/s41534-019-0201-8.

[15] Matteo M. Wauters, Emanuele Panizon, Glen B. Mbeng, and Giuseppe E. Santoro. Reinforcement-learning-assisted quantum optimization. *Phys. Rev. Research*, 2:033446, Sep 2020. DOI: 10.1103/PhysRevResearch.2.033446. URL https://link.aps.org/doi/10.1103/PhysRevResearch.2.033446.

[16] Christopher J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989. URL https://doi.org/10.1016/0921-8890(95)00026-C.

[17] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. DOI: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature(London)*, 518(7540):529–533, 2015. DOI: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

[19] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. 2015. DOI: 10.48550/ARXIV.1509.02971. URL https://arxiv.org/abs/1509.02971.

[20] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, USA, 10th edition, 2011. ISBN 1107002176. DOI: https://doi.org/10.1017/CBO9780511976667.

[21] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature(London)*, 549(7671):195–202, 2017. DOI: 10.1038/nature23474. URL https://doi.org/10.1038/nature23474.

[22] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. DOI: 10.1109/SFCS.1994.365700. URL https://ieeexplore.ieee.org/document/365700.

[23] Lov Kumar Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325–328, Jul 1997. DOI: 10.1103/PhysRevLett.79.325. URL https://link.aps.org/doi/10.1103/PhysRevLett.79.325.

[24] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Phys. Rev. Lett.*, 109:050505, Aug 2012. DOI: 10.1103/PhysRevLett.109.050505. URL https://link.aps.org/doi/10.1103/PhysRevLett.109.050505.

[25] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Phys. Rev. Lett.*, 113:130503, Sep 2014. DOI: 10.1103/PhysRevLett.113.130503. URL https://link.aps.org/doi/10.1103/PhysRevLett.113.130503.

[26] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Phys. Rev. Lett.*, 121:040502, Jul 2018. DOI: 10.1103/PhysRevLett.121.040502.

URL https://link.aps.org/doi/10.1103/PhysRevLett.121.040502.

[27] Sankar Das Sarma, Dong-Ling Deng, and Lu-Ming Duan. Machine learning meets quantum physics. *Physics Today*, 72(3): 48–54, Mar 2019. ISSN 1945-0699. DOI: 10.1063/pt.3.4164. URL http://dx.doi.org/10.1063/PT.3.4164.

[28] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014. DOI: 10.1038/nphys3029. URL https://doi.org/10.1038/nphys3029.

[29] Nico Meyer, Christian Ufrecht, Maniraman Periyasamy, Daniel D. Scherer, Axel Plinge, and Christopher Mutschler. A survey on quantum reinforcement learning, 2024. URL https://arxiv.org/abs/2211.03464.

[30] Daoyi Dong, Chunlin Chen, Hanxiong Li, and Tzyh-Jong Tarn. Quantum reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(5):1207–1220, 2008. DOI: 10.1109/TSMCB.2008.925743. URL https://ieeexplore.ieee.org/document/4579244.

[31] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Phys. Rev. Lett.*, 117:130501, Sep 2016. DOI: 10.1103/PhysRevLett.117.130501. URL https://link.aps.org/doi/10.1103/PhysRevLett.117.130501.

[32] Giuseppe Davide Paparo, Vedran Dunjko, Adi Makmal, Miguel Angel Martin-Delgado, and Hans J. Briegel. Quantum speedup for active learning agents. *Phys. Rev. X*, 4:031002, Jul 2014. DOI: 10.1103/PhysRevX.4.031002. URL https://link.aps.org/doi/10.1103/PhysRevX.4.031002.

[33] Vedran Dunjko, Jacob M Taylor, and Hans J Briegel. Advances in quantum reinforcement learning. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 282–287, 2017. DOI: 10.1109/SMC.2017.8122616. URL https://ieeexplore.ieee.org/document/8122616.

[34] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81 (7):074001, jun 2018. DOI: 10.1088/1361-6633/aab406. URL https://doi.org/10.1088/1361-6633/aab406.

[35] Sofiene Jerbi, Lea M. Trenkwalder, Hendrik Poulsen Nautrup, Hans J. Briegel, and Vedran Dunjko. Quantum enhancements for deep reinforcement learning in large spaces. *PRX Quantum*, 2: 010328, Feb 2021. DOI: 10.1103/PRXQuantum.2.010328. URL https://link.aps.org/doi/10.1103/PRXQuantum.2.010328.

[36] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. *IEEE Access*, 8:141007–141024, 2020. DOI: 10.1109/ACCESS.2020.3010470. URL https://ieeexplore.ieee.org/abstract/document/9144562.

[37] Owen Lockwood and Mei Si. Reinforcement learning with quantum variational circuits. In *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'20. AAAI Press, 2020. ISBN 978-1-57735-849-7. URL https://dl.acm.org/doi/abs/10.5555/3505464.3505499.

[38] Andrea Skolik, Sofiene Jerbi, and Vedran Dunjko. Quantum agents in the Gym: a variational quantum algorithm for deep Q-learning. *Quantum*, 6:720, May 2022. ISSN 2521-327X. DOI: 10.22331/q-2022-05-24-720. URL https://doi.org/10.22331/q-2022-05-24-720.

[39] Owen Lockwood and Mei Si. Playing atari with hybrid quantum-classical reinforcement learning. In Luca Bertinetto, João F. Henriques, Samuel Albanie, Michela Paganini, and Gül Varol, editors, *NeurIPS 2020 Workshop on Pre-registration in Machine Learning*, volume 148 of *Proceedings of Machine Learning Research*, pages 285–301. PMLR, 11 Dec 2021. URL https://proceedings.mlr.press/v148/lockwood21a.html.

[40] Samuel Yen-Chi Chen. Quantum Deep Q-Learning with Distributed Prioritized Experience Replay . In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 31–35, Los Alamitos, CA, USA, Septem-

ber 2023. IEEE Computer Society. DOI: 10.1109/QCE57702.2023.10180. URL https://doi.ieeecomputersociety.org/10.1109/QCE57702.2023.10180.

[41] Sofiene Jerbi, Casper Gyurik, Simon Marshall, Hans Briegel, and Vedran Dunjko. Parametrized quantum policies for reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 28362–28375. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/eec96a7f788e88184c0e713456026f3f-Paper.pdf.

[42] Nico Meyer, Daniel Scherer, Axel Plinge, Christopher Mutschler, and Michael Hartmann. Quantum policy gradient algorithm with optimized action decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 24592–24613. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/meyer23a.html.

[43] Nico Meyer, Daniel D. Scherer, Axel Plinge, Christopher Mutschler, and Michael J. Hartmann. Quantum Natural Policy Gradients: Towards Sample-Efficient Reinforcement Learning . In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 36–41, Los Alamitos, CA, USA, September 2023. IEEE Computer Society. DOI: 10.1109/QCE57702.2023.10181. URL https://doi.ieeecomputersociety.org/10.1109/QCE57702.2023.10181.

[44] André Sequeira, Luis Paulo Santos, and Luis Soares Barbosa. Policy gradients using variational quantum circuits. *Quantum Machine Intelligence*, 5(1):18, 2023. ISSN 2524-4914. DOI: 10.1007/s42484-023-00101-8. URL https://doi.org/10.1007/s42484-023-00101-8.

[45] Valeria Saggio, Beate E Asenbeck, Arne Hamann, Teodor Strömberg, Peter Schiansky, Vedran Dunjko, Nicolai Friis, Nicholas C Harris, Michael Hochberg, Dirk Englund, et al. Experimental quantum speed-up in reinforcement learning agents. *Nature*, 591 (7849):229–233, 2021. DOI: 10.1038/s41586-021-03242-7. URL https://doi.org/10.1038/s41586-021-03242-7.

[46] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Advances in quantum reinforcement learning. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, page 282–287. IEEE Press, 2017. DOI: 10.1109/SMC.2017.8122616. URL https://doi.org/10.1109/SMC.2017.8122616.

[47] El Amine Cherrat, Iordanis Kerenidis, and Anupam Prakash. Quantum reinforcement learning via policy iteration. *Quantum Machine Intelligence*, 5(2):30, 2023. ISSN 2524-4914. DOI: 10.1007/s42484-023-00116-1. URL https://doi.org/10.1007/s42484-023-00116-1.

[48] Daochen Wang, Aarthi Sundaram, Robin Kothari, Ashish Kapoor, and Martin Roetteler. Quantum algorithms for reinforcement learning with a generative model. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10916–10926. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/wang21w.html.

[49] Hans J. Briegel and Gemma De las Cuevas. Projective simulation for artificial intelligence. *Scientific Reports*, 2 (1):400, 2012. ISSN 2045-2322. DOI: 10.1038/srep00400. URL https://doi.org/10.1038/srep00400.

[50] Alexey A. Melnikov, Adi Makmal, Vedran Dunjko, and Hans J. Briegel. Projective simulation with generalization. *Scientific Reports*, 7(1):14430, 2017. ISSN 2045-2322. DOI: 10.1038/s41598-017-14740-y. URL https://doi.org/10.1038/s41598-017-14740-y.

[51] Giuseppe Davide Paparo, Vedran Dunjko, Adi Makmal, Miguel Angel Martin-Delgado, and Hans J. Briegel. Quantum speedup for active learning agents. *Phys. Rev. X*, 4:031002, Jul 2014. DOI: 10.1103/Phys-

RevX.4.031002. URL https://link.aps.org/doi/10.1103/PhysRevX.4.031002.

[52] V Dunjko, N Friis, and H J Briegel. Quantum-enhanced deliberation of learning agents using trapped ions. *New Journal of Physics*, 17(2):023006, jan 2015. DOI: 10.1088/1367-2630/17/2/023006. URL https://dx.doi.org/10.1088/1367-2630/17/2/023006.

[53] Th Sriarunothai, S Wölk, G S Giri, N Friis, V Dunjko, H J Briegel, and Ch Wunderlich. Speeding-up the decision making of a learning agent using an ion trap quantum processor. *Quantum Science and Technology*, 4 (1):015014, dec 2018. DOI: 10.1088/2058-9565/aaef5e. URL https://dx.doi.org/10.1088/2058-9565/aaef5e.

[54] Martijn Van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement Learning*, pages 3–42. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-27645-3_1. URL https://doi.org/10.1007/978-3-642-27645-3_1.

[55] Gavin A Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. Technical report, 1994. URL http://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/rummery_tr166.pdf.

[56] Sham M Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems*, volume 14, pages 1531–1538. MIT Press, 2002. URL https://proceedings.neurips.cc/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

[57] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014. DOI: 10.1038/ncomms5213. URL https://doi.org/10.1038/ncomms5213.

[58] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. DOI: 10.48550/ARXIV.1411.4028. URL https://arxiv.org/abs/1411.4028.

[59] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, nov 2019. DOI: 10.1088/2058-9565/ab4eb5. URL https://doi.org/10.1088/2058-9565/ab4eb5.

[60] Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, USA, 1992. URL https://dl.acm.org/doi/10.5555/168871.

[61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6980.

[62] Xiaokai Hou, Guanyu Zhou, Qingyu Li, Shan Jin, and Xiaoting Wang. A universal duplication-free quantum neural network. URL https://arxiv.org/abs/2106.13211.

[63] Dorit Aharonov and Tomer Naveh. Quantum np-a survey. *arXiv preprint quant-ph/0210077*, 2002. DOI: 10.48550/ARXIV.QUANT-PH/0210077. URL https://arxiv.org/abs/quant-ph/0210077.

[64] John Watrous. Quantum computational complexity. DOI: 10.48550/ARXIV.0804.3401. URL https://arxiv.org/abs/0804.3401.

[65] Sevag Gharibian, Yichen Huang, Zeph Landau, and Seung Woo Shin. Quantum hamiltonian complexity. pages 7174–7201, 2009. DOI: 10.1007/978-0-387-30440-3_428. URL https://doi.org/10.1007/978-0-387-30440-3_428.

[66] Julia Kempe, Alexei Kitaev, and Oded Regev. The complexity of the local hamiltonian problem. In *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*, volume 35, pages 372–383, 2006. URL https://doi.org/10.1007/978-3-540-30538-5_31.

[67] Daniel S. Abrams and Seth Lloyd. Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. *Phys. Rev. Lett.*, 83:5162–5165, Dec 1999. DOI: 10.1103/PhysRevLett.83.5162. URL https://link.aps.org/doi/10.1103/PhysRevLett.83.5162.

[68] Navin Khaneja, Timo Reiss, Cindie Kehlet,

Thomas Schulte-Herbrüggen, and Steffen J. Glaser. Optimal control of coupled spin dynamics: design of nmr pulse sequences by gradient ascent algorithms. *Journal of Magnetic Resonance*, 172(2): 296–305, 2005. ISSN 1090-7807. DOI: https://doi.org/10.1016/j.jmr.2004.11.004. URL https://www.sciencedirect.com/science/article/pii/S1090780704003696.

[69] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 1934–1940. AAAI Press, 2016. URL https://ojs.aaai.org/index.php/AAAI/article/view/10226.

[70] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000. DOI: 10.1162/089976600300015961. URL https://ieeexplore.ieee.org/document/6789455.

[71] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. DOI: 10.48550/ARXIV.1606.01540. URL https://arxiv.org/abs/1606.01540.

# A Classical Reinforcement Learning

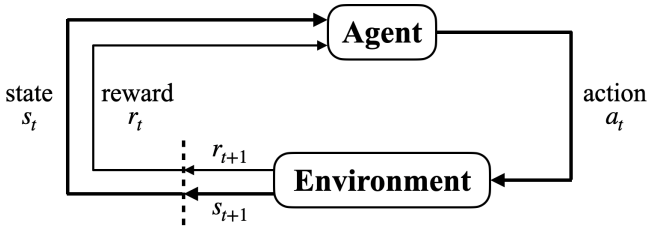


Figure 7: An illustration of the standard RL method.

Here, we will give a detailed introduction to the basic concepts of reinforcement learning for beginners to learn. In RL, the basic elements include a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, the reward $\mathcal{R}$. The model of reinforcement learning is shown in Fig. 7 [1], the agent and the environment interact continually. In a step, the agent receives an observation $s_t$, then chooses an action $a_t$. Next, the agent performs an action $a_t$, and the environment move to next state $s_{t+1}$ and emits a reward $r_{t+1}$. At the next step, the agent receives the reward $r_{t+1}$ determined by the 3-tuple $(s_t, a_t, s_{t+1})$. Given an initial state $s_0$, the agent-environment interactions will generate the following sequence: $s_0, a_0, r_1, s_1, a_1, r_2, \cdots$. Such a sequence is called an *episode* in RL. Next, we define the following three key elements of RL:

(1) Policy

The policy can be considered as a mapping from $\mathcal{S}$ to $\mathcal{A}$, which sets the rules on how to choose the action based on the environment's state. Such policy is determined by certain optimization objective, such as maximizing the cumulative reward. A policy can be either deterministic or stochastic. A deterministic policy is characterized by a function $a = \pi(s)$, meaning that under the same policy, at time step $t$, the action $a_t$ is uniquely determined by the current environment's state $s_t$. Given the state $s$, we define the stochastic policy, $\pi_\theta(a|s) \equiv P[a|s, \theta]$ as the probability of choosing the random action $a$, where $\theta$ is parameter charactering the distribution of $P[a|s, \theta]$.

(2) Cumulative reward

As mentioned above, at time step $t$, the policy goal of the agent is to maximize the cumulative reward it receives in the long run. At each step $t$, if we define the accumulative reward as $R_t = \sum_{k=0}^{\infty} r_{t+k+1}$, it may not be convergent and becomes ill-defined; alternatively, we can introduce a discount factor $\gamma(0 \leq \boldsymbol{\gamma} \leq 1)$ and define the cumulative reward as $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. The larger the discount factor, the longer time span of future rewards we will consider to determine the current-state policy. At time step $t$, the reward $r_t$ determines the immediate return, and the cumulative reward $R_t$ determines the long-term return.

(3) Value function

Notice that when $a_t$ or $s_t$ is stochastic, $r_t$ and $R_t$ are also stochastic. Hence, we further define the value function $Q$ to be the expectation of the cumulative reward, $Q(s, a) \equiv E[R_t|s, a]$, under the policy $\pi$. The goal of RL is to find the optimal policy that maximizes the value function $Q$.

RL problems can be classified into two categories: discrete-action-space(DAS) problems and continuous-action-space (CAS) problems. In a DAS problem, the agent chooses the action from a finite set $\{a_k\}$, $k = 1, \cdots, l$. For example, in the Pong game [18], the action set for moving the paddle is {up, down}. In a CAS problem, the action can be parametrized as a real-valued vector[69]. In the CartPole environment [70], the action is the thrust and can be parametrized as a continuous variable $\theta \in [-1, 1]$. For DAS problems, popular RL algorithms include Q-learning [16], Sarsa [55], Deep Q-learning Network(DQN) [18], etc.; for CAS problems, popular algorithms include Policy Gradient [56], Deep Deterministic Policy Gradient(DDPG) [19], etc.

Notice that the DQN algorithm is only efficient when solving problems with small DAS. It quickly becomes inefficient and intractable when the size of the DAS becomes large. Hence, although a CAS problem can be converted into a DAS problem through discretization, the DQN algorithm will not work in solving the converted DAS problem, if we require high discretization accuracy. For CAS problems, it is better to use a CAS algorithm, such as DDPG.

## B Discrete Action Space Algorithms

Q-learning is a milestone in reinforcement learning algorithms. The main idea of the algorithm is to construct a Q-table of state-action pairs to store the Q-values, and then the agent chooses the action that can obtain the largest cumulative reward based on the Q-values.

In the Q-learning algorithm, an immediate reward matrix $R$ can be constructed to represent the reward value from state $s_t$ to the next state $s_{t+1}$. The Q-value is calculated based on the matrix $R$, and it is updated by the following formula [1]:

$$\begin{aligned} Q(s_t, a_t) \longleftarrow\ & Q(s_t, a_t) \\ & + \alpha[r_t + \boldsymbol{\gamma} \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \end{aligned} \tag{6}$$

where $\boldsymbol{\gamma}$ is the discount factor, $\alpha$ is the learning rate that determines how much the newly learned value of $Q$ will override the old value. By training the agent, the Q-value will gradually converge to the optimal Q-value.

The Q-learning is only suitable for storing action-state pairs which are low-dimensional and discrete. In large-space tasks, the corresponding Q-table could become extremely large, causing the RL problem intractable to solve, which is known as the curse of dimensionality. The DQN algorithm takes the advantage of deep learning technique to solve the RL problems. Specifically, it introduces a neural network defined as Q-network $Q(s_t, a_t; \boldsymbol{\omega})$, whose function is similar to the Q-table in approximating the value function. The input of the Q-network is the current state $s_t$, and the output is the Q-value. The $\epsilon$-greedy strategy is used to choose the value of $a$ according to the following probability distribution [1]:

$$a = \begin{cases} \arg\max_a Q(a), & \text{with probability } 1-\epsilon; \\ \text{a random action}, & \text{with probability } \epsilon. \end{cases} \tag{7}$$

In order to stabilize the training, the DQN algorithm uses two tricks: experience replay and a target network. The method of experience replay is to use a replay buffer to store the experienced data and to sample some data from the replay buffer at each step $t$ to update the parameters in the neural network. The DQN algorithm introduces a target-Q network $Q(s_{t+1}, a_{t+1}; \boldsymbol{\omega}')$ which is a copy of the Q-network. The input of $Q(s_{t+1}, a_{t+1}; \boldsymbol{\omega}')$ is $s_{t+1}$ and the output is $Q(s_{t+1}, a_{t+1})$. However, the target-Q network parameters are updated using Q-network parameters at every $m$ steps, where $m$ is a constant. The DQN algorithm updates the Q-network by reducing the value of the loss function $L(\boldsymbol{\omega}) = E[((r_t + \boldsymbol{\gamma} \max_{a_{t+1}} Q'(s_{t+1}, a_{t+1}; \boldsymbol{\omega}')) - Q(s_t, a_t; \boldsymbol{\omega}))^2]$.

## C Continuous Action Space Algorithms

For tasks in continuous action space, we usually use the DDPG algorithm. The DDPG algorithm make use of the neural network to construct the desired policy function $\pi : s_t \to a_t$ such that the value function is maximized. As shown in Fig. 8, the quantum DDPG includes four neural networks: the policy-network, the Q-network, the target-policy-network and the target-Q-network. Specifically, the policy NN accesses $s_t$, and the Q-NN accesses $s_t$ and $\boldsymbol{\theta}_t$, where $\boldsymbol{\theta}_t$ is the output of policy QNN. Therefore, these two QNNs cannot be executed in parallel. The target policy NN accesses $s_{t+1}$, and the target Q-NN accesses $s_{t+1}$ and $\boldsymbol{\theta}_{t+1}$, so these two target networks and the two original networks can be executed in parallel. In addition, the Q-network is used to approximate the value function, while the policy-network is used to approximate the policy function.

The DDPG algorithm uses the same tricks as DQN to stabilize the training. The update of the policy-network is achieved by reducing the loss function $L(\boldsymbol{\eta}) = -Q(s_t, a_t; \boldsymbol{\eta})$. Similar to DQN, the update of the Q-network in DDPG is achieved through reducing the value of the loss function $L(\boldsymbol{\omega}) = E[((r_t + \boldsymbol{\gamma} \max_{a_{t+1}} Q'(s_{t+1}, a_{t+1}; \boldsymbol{\omega}')) - Q(s_t, a_t; \boldsymbol{\omega}))^2]$. Through training, the estimated value output of the Q-network will be more accurate, and the action given by the policy-network will make the Q-value higher.

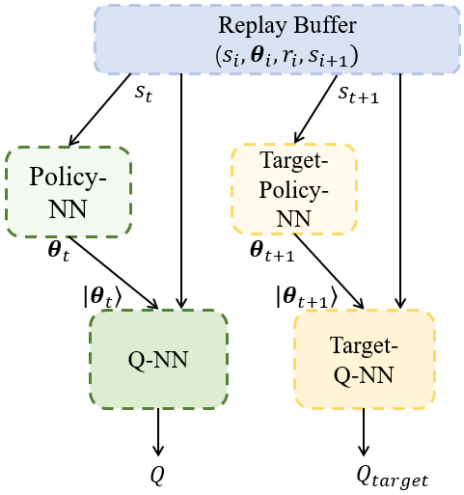


Figure 8: The neural network structure of DDPG Algorithm.

## D Quantum Reinforcement Learning in Discrete Action Space

Besides DAS problems, our QRL framework can solve DAS problems as well. For DAS tasks, we can use quantum Q-learning or quantum DQN algorithm. Specifically, we consider a Frozen Lake environment model [71] in which both the action space and the state space are finite dimensional, as shown in Fig. 9. In this environment, the agent can move from a grid to its neighboring grids and the goal is to move from the starting position to the target position. Some positions of the grids are walkable, while the others will make the agent fall into the water, resulting in a large negative reward, and a termination of the episode.

In order to solve the Frozen Lake problem using our QRL framework, we number the $N$ grids from 0 to $N-1$, and encode them into the set of basis states $S = \{|j\rangle\}$, $j = 0, \cdots, j-1$ of an $N$-dimensional quantum environment, composed of $n = \log N$ qubits. At each step $t$, the state of the environment $|s_t\rangle$ equals to one of the basis states in $S$. The action $a(\boldsymbol{\theta}_t)$ can be represented as a parameterized action unitary $U_a(\boldsymbol{\theta}_t)$ on $|s_t\rangle$, where $\boldsymbol{\theta}_t$ is the action parameter. Assuming that at the position $|j\rangle$, there are four actions(up, down, left, and right) the agent can choose from, corresponding to the discrete set $\{U_a(\boldsymbol{\theta}_{j,k})\} = \{U_a(\boldsymbol{\theta}_{j,1}), U_a(\boldsymbol{\theta}_{j,2}), U_a(\boldsymbol{\theta}_{j,3}), U_a(\boldsymbol{\theta}_{j,4})\}$ where $\boldsymbol{\theta}_{j,k} = (\theta_{j,k}^{(1)}, \cdots, \theta_{j,k}^{(n)})^T$ is a real vector. Specifically, we construct $U_a(\boldsymbol{\theta}_{j,k})$ as $U_a(\boldsymbol{\theta}_{j,k}) \equiv R_y(\theta_{j,k}^{(1)}) \otimes \cdots \otimes R_y(\theta_{j,k}^{(n)})$, where $R_y(\theta_{j,k}^{(n)}) = \exp(-i\theta_{j,k}^{(n)}\sigma_y/2)$, and $\theta_{k,n}^{(j)} \in \{0, \pi\}$. For example, in Fig. 9, if the agent is at the position $|5\rangle$ and wants to move right to the position $|6\rangle$, then the required action parameter is $\boldsymbol{\theta}_{5,4} = (0, 0, \pi, \pi)$, corresponding to $U(\boldsymbol{\theta}_{5,4}) \equiv R_y(0) \otimes R_y(0) \otimes R_y(\pi) \otimes R_y(\pi)$, satisfying $U(\boldsymbol{\theta}_{5,4})|0101\rangle = |0110\rangle$.

We generate the reward function $f$ by introducing a reward register $|r_t\rangle$ and design the reward unitary $U_r = \sum_{j\in F} |j\rangle\langle j| \otimes I \otimes I + \sum_{j\in H} |j\rangle\langle j| \otimes I \otimes \sigma_x + \sum_{j\in G} |j\rangle\langle j| \otimes \sigma_x \otimes I$ and the measurement observable $M = \sum_{j=0}^{N} j|j\rangle\langle j|$. Then the reward $r_{t+1}$ is

$$r_{t+1} = f(p_{t+1}) = \begin{cases} -1, & p_{t+1} = 0 \\ -10, & p_{t+1} = 1 \\ 10, & p_{t+1} = 2 \end{cases} \tag{8}$$

where $p_{t+1} \equiv \langle s_t|\langle 0|U_a^\dagger(\boldsymbol{\theta}_{t,k})U_r^\dagger M U_r U_a(\boldsymbol{\theta}_{t,k})|0\rangle|s_t\rangle$ and $r_{t+1} = f(p_{t+1})$. Here, $r_{t+1}$ is the reward for the action $U_a(\boldsymbol{\theta}_{t,k})$ at the state $|s_t\rangle$ and $|0\rangle$ is the initial state of the reward register. With all RL elements represented as the components of a quantum circuit, we can use the quantum DQN algorithm to solve the Frozen Lake problem. In stage 1, we train the agent to find a state-action sequence to maximize the cumulative reward. In the training, the data $(|s_t\rangle, a(\boldsymbol{\theta}), r_{t+1})$ obtained from each interaction between

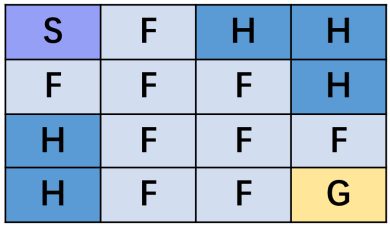


Figure 9: Frozen Lake environment model. S is the starting position; F is the walkable position; H is the hole position, and G is the goal position. The actions that the agent can choose at each position are up, down, left, and right.

the agent and the environment is recorded and these data are used to update the Q-value. In stage 2, we use the optimal policy to generate $\{U_a(\boldsymbol{\theta}_{0,k}), \cdots, U_a(\boldsymbol{\theta}_{T,k})\}$ to complete the task. In simulation, we set the size of the replay buffer is set as 2000, the size of the batch is set as 32, and the other parameters are set as $\gamma = 0.9$, $\tau = 0.001$. The quantum registers to implement the QNNs contain four qubits, and the depth of the QNN circuits is one. After 500 episodes of training, the agent can reach the target position by moving 6 steps. The sequence of actions is right, down, right, down, right, down. We can see that the agent has obtained one of the shortest paths through training.