Hindawi
Computational Intelligence and Neuroscience
Volume 2021, Article ID 5370281, 9 pages
https://doi.org/10.1155/2021/5370281

Research Article

Minibatch Recursive Least Squares Q-Learning

Chunyuan Zhang, Qi Song, and Zeng Meng

School of Computer Science and Technology, Hainan University, Haikou, Hainan 570228, China

Correspondence should be addressed to Chunyuan Zhang; zcy7566@126.com

Received 12 August 2021; Revised 21 September 2021; Accepted 23 September 2021; Published 8 October 2021

Academic Editor: Yugen Yi

The deep Q-network (DQN) is one of the most successful reinforcement learning algorithms, but it has some drawbacks such as slow convergence and instability. In contrast, the traditional reinforcement learning algorithms with linear function approximation usually have faster convergence and better stability, although they easily suffer from the curse of dimensionality. In recent years, many improvements to DQN have been made, but they seldom make use of the advantage of traditional algorithms to improve DQN. In this paper, we propose a novel Q-learning algorithm with linear function approximation, called the minibatch recursive least squares Q-learning (MRLS-Q). Different from the traditional Q-learning algorithm with linear function approximation, the learning mechanism and model structure of MRLS-Q are more similar to those of DQNs with only one input layer and one linear output layer. It uses the experience replay and the minibatch training mode and uses the agent's states rather than the agent's state-action pairs as the inputs. As a result, it can be used alone for low-dimensional problems and can be seamlessly integrated into DQN as the last layer for high-dimensional problems as well. In addition, MRLS-Q uses our proposed average RLS optimization technique, so that it can achieve better convergence performance whether it is used alone or integrated with DQN. At the end of this paper, we demonstrate the effectiveness of MRLS-Q on the CartPole problem and four Atari games and investigate the influences of its hyperparameters experimentally.

1. Introduction

Reinforcement learning (RL) is an important machine learning methodology for solving sequential decision-making problems. In theory, by interacting with an initially unknown environment, the RL agent can learn the optimal action policies at different states to maximize the cumulative expected return [1]. Unfortunately, in the past several decades, due to the so-called "curse of dimensionality," RL can only be used to solve some real-world problems with the small-scale discrete or low-dimensional continuous state space. It is not until 2013 that this dilemma was partially solved by Mnih et al. [2]. By combining the Q-learning algorithm with deep learning, they proposed the preliminary version of the deep Q-network (DQN) algorithm. Two years later, Mnih et al. [3] presented the normal version of DQN, which achieves the human-level performance on 49 classical Atari games. Since then, DQN has attracted more and more research attention, and many other novel deep RL algorithms [4, 5] and new applications [6, 7] have been proposed, and thus deep RL has become a thriving research branch in artificial intelligence. However, although DQN has succeeded in some more complicated problems [8–10], it still has many drawbacks, such as slow convergence, instability, and low sample efficiency. Therefore, we will focus on how to improve the DQN's performance in this paper.

Currently, there are three main categories of research work on improving DQN. The first category mainly focuses on how to estimate action values accurately. For example, Hasselt et al. [11] proposed the double DQN, which can reduce the observed overestimation by exploiting the idea of double Q-learning. Wang et al. [12] introduced a dueling network architecture, which separately estimates state values and advantage values to improve the policy evaluation. Hausknecht and Stone [13] presented the deep recurrent Q-network, which is more suitable for solving partial observation problems, by adding recurrent LSTM layers to convolutional networks. Kim et al. [14] combined the

mellowmax method with DQN to calculate the target action values, preventing overestimation effectively. Anschel et al. [15] proposed the averaged DQN, which uses some previously learned action-value estimates to produce the current action value. This algorithm can reduce the approximation error variance in the target values. The second category mainly focuses on how to explore or exploit samples efficiently. Schaul et al. [16] presented a prioritized experience replay, which can make the effective use of historical samples to improve the DQN's convergence performance. Fortunato et al. [17] proposed the noisynet DQN, which adds noise to the deep network parameters for aiding efficient exploration. Lee et al. [18] introduced an episodic backward update to improve the sample efficiency. The third category mainly focuses on how to reduce memory and computation. Mnih et al. [19] proposed asynchronous variants of four standard reinforcement learning algorithms, such as the asynchronous one-step Q-learning algorithm and the asynchronous n-step Q-learning algorithm. Interestingly, this work also opens the door to research the asynchronous advantage actor-critic (A3C) algorithm.

In traditional RL, Q-learning algorithms often use linear functions to approximate action values, which have better stability and fewer hyperparameters to be trained than DQNs [20]. In particular, the least squares (LS) type RL algorithms, such as the least squares policy iteration (LSPI) algorithm [21], the fitted-Q iteration (FQI) algorithm [22], and the recursive least squares temporal difference with forgetting factor (RLS-TD-f) algorithm [23], not only have better stability but also have faster convergence. In the research community of adaptive filtering, the LS and the recursive least squares (RLS) algorithms are famous for their fast convergence rate. Obviously, the success of LS-type RL algorithms mainly benefits from this merit. In recent years, many new machine learning algorithms, such as the extreme learning machine (ELM) [24] and the broad learning system [25, 26], have been proposed by combining LS or RLS algorithms. In practice, the last layer of the neural network used for DQN is usually a linear layer, which means that we probably can improve the DQN's performance by integrating DQN with the LS-type RL algorithms. In fact, Levine et al. [20] proposed a hybrid approach—the least squares deep Q-network (LS-DQN), which combines DQN with LSPI or FQI. By retraining the last layer of the policy network with a batch least squares update periodically, LS-DQN can obtain better convergence performance than DQN, whereas LS-DQN is not easy to use. At each update by using LSPI or FQI, LS-DQN needs to use the current network parameters to generate a training dataset, which requires running a forward pass of the deep network for each sample in the experience replay buffer. In addition, LS-DQN needs to generate new state-action features and compute the matrix inverse. From the DQN's learning mechanism, a perfect integrated LS-type algorithm should be able to use the inputs of the DQN's last layer for approximating action values and should have the same learning mode as DQN.

In our previous work [27], we propose two policy control algorithms called ESNRLS-Q and ESNRLS-Sarsa. They seem to meet the above requirements to some extent, although they are also difficult to integrate with DQNs. They use the same experience replay and minibatch learning mode as DQN. In addition, they can avoid computing the matrix inverse and are more suitable for online learning by using recursive least squares (RLS). Based on this work and inspired by the work of Levine et al., we propose a novel minibatch RLS Q-learning algorithm with linear function approximation, called the MRLS-Q. Our main contributions are as follows. (1) By borrowing the experience replay to remove the temporal correlation between the observed transitions, we first combine the traditional Q-learning algorithm with the RLS optimization technique. (2) By using state features rather than state-action features for linear function approximation, we make MRLS-Q able to be used alone and also be integrated into DQN seamlessly. (3) In order to reduce the computational complexity and make the RLS method suitable for training parameters in the minibatch mode, we present an average approximation method for updating the RLS autocorrelation matrix. (4) In order to alleviate the feature change of the same state and integrate MRLS-Q into DQN, we present a new method to define the feature function of MRLS-Q. (5) We demonstrate the effectiveness of MRLS-Q, alone and as the last layer of DQN, by using the CartPole problem and four Atari games, respectively. We also test the influences of its hyperparameters experimentally.

The remainder of this paper is organized as follows. Section 2 describes the related theories and algorithms of MRLS-Q. Section 3 represents the detailed derivation and the practical implementation of MRLS-Q. Then, in Section 4, comparison experiments on the CartPole problem and four Atari games are conducted to separately verify the effectiveness of MRLS-Q used alone and as the last layer of DQN. Finally, Section 5 summarizes the whole paper.

2. Background

In this section, we briefly review the related theories and algorithms of our MRLS-Q, including the Markov decision process (MDP), DQN, and LS-DQN. In addition, we also describe some notations that will be used throughout this paper.

2.1. Markov Decision Process. In RL, a sequential decision problem is generally formulated as an MDP with a five-tuple $\langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(\mathbf{s}_t'|\mathbf{s}_t, a_t) \in [0,1]$ and $r(\mathbf{s}_t'|\mathbf{s}_t, a_t) \in \mathcal{R}$ are the state-transition probability and the immediate reward from the state $\mathbf{s}_t$ to the next state $\mathbf{s}_t'$ by taking the action a_t, and $\gamma \in (0,1]$ is the discount factor. At the state $\mathbf{s}_t$, the agent's action a_t is determined by the control policy π.

For a given MDP, the goal of RL is to learn the optimal policy π^* for maximizing the cumulative expected return $J(\pi)$, i.e.,

$$\pi^* = \arg\max_{\pi} J(\pi), \tag{1}$$

where $J(\pi)$ is usually defined in the form of discount return [1] as

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | \mathbf{s}_0, \pi\right], \tag{2}$$

where $\mathbf{s}_0$ is the initial state, whereas $J(\pi)$ is hardly calculated by using the above equation directly, since $P(\mathbf{s}_t'|\mathbf{s}_t, a_t)$ is unknown in RL, and $\mathbf{s}_t'$ and r_t can only be obtained by the agent's interaction with the environment.

To tackle this problem, RL usually resorts to estimating the action value $Q^\pi(\mathbf{s}_t, a_t)$ to measure the performance of π when the initial state and action are $\mathbf{s}_t$ and a_t. In this paper, we assume that $\mathcal{S}$ is continuous and $\mathcal{A}$ is discrete. For this kind of MDP problems, to overcome the curse of dimensionality, $Q^\pi(\mathbf{s}_t, a_t)$ is often approximated by linear function approximators or deep neural networks.

2.2. The DQN Algorithm. DQN is probably the most important algorithm in deep RL. It combines the Q-Learning algorithm with deep neural networks and uses the experience relay for breaking the correlation among samples and training network parameters.

The DQN algorithm can be summarized as follows. At the current step t, the agent firstly uses the ϵ-greedy policy to select the action a_t as

$$a_t = \begin{cases} \arg\max_{a \in A} Q(\mathbf{s}_t, a; \Theta_{t-1}), & \text{w.p. } 1 - \epsilon, \\ \text{a random action in } A, & \text{w.p. } \epsilon, \end{cases} \tag{3}$$

where ϵ is the exploration factor, Θ_{t-1} is the policy network parameter, and $Q(\mathbf{s}_t, a; \Theta_{t-1})$ is approximated by this network. Then, after taking a_t, the agent moves to the next state $\mathbf{s}_t'$, obtains the reward r_t, and stores $(\mathbf{s}_t, a_t, \mathbf{s}_t', r_t, d_t)$ into the experience replay buffer $\mathcal{D}$, where $d_t \in \{0, 1\}$ denotes that $\mathbf{s}_t'$ is the terminal state or not. Next, by using the minibatch $\mathcal{M}_t = \{(\widehat{\mathbf{s}}_{t,i}, \widehat{a}_{t,i}, \widehat{\mathbf{s}}_{t,i}', \widehat{r}_{t,i}, \widehat{d}_{t,i})\}_{i=1,\ldots,M}$ sampled from $\mathcal{D}$ randomly, the algorithm calculates the loss of the policy network as

$$L(\Theta_{t-1}) = \frac{1}{2M}\left\| Q^\pi\left(\widehat{S}_t, \widehat{\mathbf{a}}_t\right) - Q\left(\widehat{S}_t, \widehat{\mathbf{a}}_t; \Theta_{t-1}\right)\right\|_2^2, \tag{4}$$

where $\widehat{S}_t = [\widehat{\mathbf{s}}_{t,1}, \ldots, \widehat{\mathbf{s}}_{t,M}]^T$, $\widehat{\mathbf{a}}_t = [\widehat{a}_{t,1}, \ldots, \widehat{a}_{t,M}]^T$, and $Q^\pi(\widehat{S}_t, \widehat{\mathbf{a}}_t)$ is the target value of $Q(\widehat{S}_t, \widehat{\mathbf{a}}_t; \Theta_{t-1})$, which is estimated by the target network as

$$Q^\pi\left(\widehat{S}_t, \widehat{\mathbf{a}}_t\right) = \widehat{\mathbf{r}}_t + \gamma\left(1 - \widehat{\mathbf{d}}_t\right) \circ \max_{a \in \mathcal{A}} Q\left(\widehat{S}_t', a; \widetilde{\Theta}\right), \tag{5}$$

where $\widehat{\mathbf{r}}_t = [\widehat{r}_{t,1}, \ldots, \widehat{r}_{t,M}]^T$, $\widehat{S}_t' = [\widehat{\mathbf{s}}_{t,1}', \ldots, \widehat{\mathbf{s}}_{t,M}']^T$, $\widehat{\mathbf{d}}_t = [\widehat{d}_{t,1}, \ldots, \widehat{d}_{t,M}]^T$, $\circ$ denotes the Hadamard product, and $\widetilde{\Theta}$ is the target network parameter which is copied from the policy network every some fixed steps or episodes. Finally, by using some gradient descent optimization method, the algorithm updates Θ_{t-1} to Θ_t. For example, by using the SGD method [28], Θ_{t-1} is updated as

$$\Theta_t = \Theta_{t-1} - \alpha \nabla_{\Theta_{t-1}}, \tag{6}$$

where α is the learning rate and $\nabla_{\Theta_{t-1}}$ denotes $\partial L(\Theta_{t-1})/\partial \Theta_{t-1}$.

2.3. The LS-DQN Algorithm. LS-DQN is a hybrid approach, which combines the traditional LSPI or FQI algorithm with the DQN algorithm. By enjoying the stability and efficiency of LSPI or FQI, it can obtain better performance than DQN.

The LS-DQN algorithm can be briefly summarized as follows. Whenever the agent runs DQN some steps, it uses LSPI or FQI to retrain the last layer of the policy network once. The retraining consists of the following three substeps. Firstly, by recalculating all samples in the experience replay buffer with the current network parameters, the policy network generates a new dataset $\widetilde{\mathcal{D}}$. Secondly, by using the current network parameters and the dataset $\widetilde{\mathcal{D}}$, the algorithm generates state-action features. Finally, the algorithm uses LSPI to retrain the current last-layer parameter Θ_t^L in the policy network as

$$\left[\left(\Theta_{t(:,1)}^L\right)^T, \ldots, \left(\Theta_{t(:,|\mathcal{A}|)}^L\right)^T\right]^T = A^{-1}b, \tag{7}$$

where $\Theta_{t(:,i)}^L$ is the i^{th} column vector of Θ_t^L. Besides, A and $\mathbf{b}$ are defined as follows:

$$A = \frac{1}{|\widetilde{\mathcal{D}}|}\sum_{j=1}^{|\widetilde{\mathcal{D}}|}\left[\boldsymbol{\phi}\left(\widehat{\mathbf{s}}_j, \widehat{a}_j\right)\left(\boldsymbol{\phi}\left(\widehat{\mathbf{s}}_j, \widehat{a}_j\right) - \gamma\left(1 - d_j\right)\boldsymbol{\phi}\left(\widehat{\mathbf{s}}_j', \pi\left(\widehat{\mathbf{s}}_j'\right)\right)\right)^T\right],$$

$$\mathbf{b} = \frac{1}{|\widetilde{\mathcal{D}}|}\sum_{j=1}^{|\widetilde{\mathcal{D}}|}\left[\boldsymbol{\phi}\left(\widehat{\mathbf{s}}_j, \widehat{a}_j\right) r_j\right], \tag{8}$$

where $\phi(\widehat{\mathbf{s}}_j, \widehat{a}_j)$ is the state-action feature of the state-action pair $(\widehat{\mathbf{s}}_j, \widehat{a}_j)$. As Levine et al. stated in their work [20], the algorithm can also retrain Θ_t^L by using FQI, since it is a batch shallow RL algorithm that computes iterative approximations of the Q-function using regression. For brevity, we will not discuss the FQI algorithm in this paper.

3. The Proposed Algorithm

In this section, we will introduce the detailed derivation and the practical implementation of our proposed algorithm, respectively. Our algorithm, the MRLS-Q algorithm, can be used not only alone but also as the last layer of DQN.

3.1. Algorithm Derivation. MRLS-Q is a new Q-learning algorithm with linear function approximation, but it is more similar to the DQN algorithm rather than the traditional Q-learning algorithm. It uses the experience replay and the minibatch training mode, separates the linear function approximator into a policy approximator and a target approximator, and uses the state features rather than the state-action features. Besides, it uses an average RLS method for updating parameters.

First, we introduce the agent's interaction with the environment. At the current step t, the agent also uses the

ϵ-greedy policy to select the action a_t as equation (3). Note here that $Q(\mathbf{s}_t, a; \Theta_{t-1})$ is approximated by the policy approximator as

$$Q(\mathbf{s}_t, a; \Theta_{t-1}) = \boldsymbol{\phi}(\mathbf{s}_t)^T \Theta_{t-1(:,\mathrm{in}(a))}, \tag{9}$$

where $\phi(\mathbf{s}_t) \in \mathscr{R}^N$ is the feature vector of $\mathbf{s}_t$, $\Theta_{t-1} \in \mathscr{R}^{N \times |\mathscr{A}|}$ is the policy approximator parameter, $\mathrm{in}(a)$ denotes the index of a in $\mathscr{A}$, and $\Theta_{t-1(:,\mathrm{in}(a))}$ is the $\mathrm{in}(a)^{\mathrm{th}}$ column vector of Θ_{t-1}. Then, the agent takes a_t, moves to $\mathbf{s}'_t$, obtains r_t, and stores $(\mathbf{s}_t, a_t, \mathbf{s}'_t, r_t, d_t)$ into the experience replay buffer $\mathscr{D}$.

Second, we introduce the RLS update of the policy approximator parameter in the minibatch training mode. Let $\mathscr{M}_n = \{(\hat{\mathbf{s}}_{n,i}, \hat{a}_{n,i}, \hat{\mathbf{s}}'_{n,i}, \hat{r}_{n,i}, \hat{d}_{n,i})\}_{i=1,\ldots,M}$ denote the minibatch sampled from $\mathscr{D}$ at the n^{th} step, and let $\Phi(\hat{S}_n) = [\phi(\hat{\mathbf{s}}_{n,1}), \ldots, \phi(\hat{\mathbf{s}}_{n,M})]^T$ denote the feature matrix of $\hat{S}_n$. Define the least squares loss function as

$$\hat{L}(\Theta) = \frac{1}{2M} \sum_{n=1}^{t} \lambda^{t-n} \left\| Q^{\pi}(\hat{S}_n, \hat{\mathbf{a}}_n) - Q(\hat{S}_n, \hat{\mathbf{a}}_n; \Theta) \right\|_2^2, \tag{10}$$

where $\lambda \in (0, 1]$ is the forgetting factor and $Q(\hat{S}_n, \hat{\mathbf{a}}_n; \Theta)$ is approximated by the policy approximator as

$$Q(\hat{S}_n, \hat{\mathbf{a}}_n; \Theta) = \Phi(\hat{S}_n) \Theta_{(:,\mathrm{in}(\hat{\mathbf{a}}_n))}, \tag{11}$$

and $Q^{\pi}(\hat{S}_n, \hat{\mathbf{a}}_n)$ is estimated by the target approximator as

$$Q^{\pi}(\hat{S}_n, \hat{\mathbf{a}}_n) = \hat{\mathbf{r}}_n + \gamma(1 - \hat{\mathbf{d}}_n) \circ \max_{a \in \mathscr{A}} \Phi(\hat{S}'_n) \tilde{\Theta}_{(:,\mathrm{in}(a))}, \tag{12}$$

where $\tilde{\Theta}$ is the target approximator parameter which is copied from the policy approximator every some fixed steps or episodes. Then, the parameter learning problem of the policy approximator can be transformed into

$$\Theta_t = \arg\min_{\Theta} \hat{L}(\Theta). \tag{13}$$

By using the chain rule, we can get

$$\hat{\nabla}_{\Theta} = -\frac{1}{M} \sum_{n=1}^{t} \lambda^{t-n} (\Phi(\hat{S}_n))^T (\hat{Q}^{\pi}(\hat{S}_n, \mathscr{A}) - \hat{Q}(\hat{S}_n, \mathscr{A}; \Theta)), \tag{14}$$

where $\hat{\nabla}_{\Theta}$ denotes $\partial \hat{L}(\Theta)/\partial \Theta$, and an element in $\hat{Q}(\hat{S}_n, \mathscr{A}; \Theta) \in \mathscr{R}^{M \times |\mathscr{A}|}$ is defined as

$$\hat{Q}(\hat{\mathbf{s}}_{n,i}, a; \Theta) = \begin{cases} Q(\hat{\mathbf{s}}_{n,i}, \hat{a}_{n,i}; \Theta), & a = \hat{a}_{n,i}, \\ 0, & a \neq \hat{a}_{n,i}, \end{cases} \tag{15}$$

and an element in $\hat{Q}^{\pi}(\hat{S}_n, \mathscr{A}) \in \mathscr{R}^{M \times |\mathscr{A}|}$ is defined as

$$\hat{Q}^{\pi}(\hat{\mathbf{s}}_{n,i}, a) = \begin{cases} Q^{\pi}(\hat{\mathbf{s}}_{n,i}, \hat{a}_{n,i}), & a = \hat{a}_{n,i}, \\ 0, & a \neq \hat{a}_{n,i}. \end{cases} \tag{16}$$

Let $\hat{\nabla}_{\Theta} = 0$. Then, we can get

$$\Theta_t = A_t^{-1} B_t, \tag{17}$$

where

$$A_t = \frac{1}{M} \sum_{n=1}^{t} \lambda^{t-n} (\Phi(\hat{S}_n))^T \Phi(\hat{S}_n),$$
$$B_t = \frac{1}{M} \sum_{n=1}^{t} \lambda^{t-n} (\Phi(\hat{S}_n))^T \hat{Q}^{\pi}(\hat{S}_n, \mathscr{A}). \tag{18}$$

Rewrite the above two equations as the following recursive forms:

$$A_t = \lambda A_{t-1} + \frac{1}{M} (\Phi(\hat{S}_t))^T \Phi(\hat{S}_t), \tag{19}$$

$$B_t = \lambda B_{t-1} + \frac{1}{M} (\Phi(\hat{S}_t))^T \hat{Q}^{\pi}(\hat{S}_t, \mathscr{A}). \tag{20}$$

Further, rewrite the above two equations as the following vector forms:

$$A_t = \lambda A_{t-1} + \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{\phi}(\hat{\mathbf{s}}_{t,i}) (\boldsymbol{\phi}(\hat{\mathbf{s}}_{t,i}))^T, \tag{21}$$

$$B_t = \lambda B_{t-1} + \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{\phi}(\hat{\mathbf{s}}_{t,i}) \hat{Q}^{\pi}(\hat{\mathbf{s}}_{t,i}, \mathscr{A}), \tag{22}$$

where $\hat{Q}^{\pi}(\hat{\mathbf{s}}_{t,i}, \mathscr{A})$ is the i^{th} row vector of $\hat{Q}^{\pi}(\hat{S}_t, \mathscr{A})$. Unfortunately, we cannot directly use the Sherman–Morrison formula [29] to compute A_t^{-1}, since the last term in the right-hand side of equation (21) is a sum of vector products.

Next, we present an average approximation method to deal with the above problem. Considering that all training samples are from the same environment and thus their features have some similarity, we rewrite equations (21) and (22) as follows:

$$A_t \approx \lambda A_{t-1} + k \overline{\boldsymbol{\phi}}_t \overline{\boldsymbol{\phi}}_t^T, \tag{23}$$

$$B_t \approx \lambda B_{t-1} + k \overline{\boldsymbol{\phi}}_t \overline{\mathbf{q}}_t^{\pi}, \tag{24}$$

where k is the approximation factor, and $\overline{\phi}_t$ and $\overline{\mathbf{q}}_t^{\pi}$ are defined as

$$\overline{\boldsymbol{\phi}}_t = \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{\phi}(\hat{\mathbf{s}}_{t,i}), \tag{25}$$

$$\overline{\mathbf{q}}_t^{\pi} = \frac{1}{M} \sum_{i=1}^{M} \hat{Q}^{\pi}(\hat{\mathbf{s}}_{t,i}, \mathscr{A}). \tag{26}$$

Let $P_t = A_t^{-1}$. By using the Sherman–Morrison formula for (23), we can get

$$P_t = \frac{1}{\lambda} (P_{t-1} - \mathbf{g}_t \mathbf{v}_t^T), \tag{27}$$

where

$$\mathbf{v}_t = P_{t-1} \overline{\boldsymbol{\phi}}_t, \tag{28}$$

$$\mathbf{g}_t = \frac{k \mathbf{v}_t}{\lambda + k \mathbf{v}_t^T \overline{\boldsymbol{\phi}}_t}. \tag{29}$$

Plugging equations (24) and (27) into (17), we finally get

$$\Theta_t \approx \Theta_{t-1} + \frac{kP_{t-1}\overline{\boldsymbol{\phi}}_t\left(\overline{\mathbf{q}}_t^{\pi} - \overline{\mathbf{q}}_t\right)}{\lambda + k\mathbf{v}_t^T\overline{\boldsymbol{\phi}}_t}, \tag{30}$$

where

$$\overline{\mathbf{q}}_t = \frac{1}{M}\sum_{i=1}^{M}\widehat{Q}\left(\widehat{\mathbf{s}}_{t,i}, \mathscr{A}; \Theta_{t-1}\right), \tag{31}$$

where $\widehat{Q}(\widehat{\mathbf{s}}_{t,i}, \mathscr{A}; \Theta_{t-1})$ denotes the i^{th} row vector of $\widehat{Q}(\widehat{S}_t, \mathscr{A}; \Theta_{t-1})$.

3.2. Practical Implementation. As reviewed in Section 2.2, DQN generally uses gradient descent methods to update network parameters. To make MRLS-Q easier to be integrated into DQN, we next rewrite equation (30) as the "gradient descent" form of $\nabla_{\Theta_{t-1}}$.

If the loss function of MRLS-Q is defined by equation (4), by using the chain rule for equation (4), we can get

$$\nabla_{\Theta_{t-1}} = -\frac{1}{M}\left(\Phi\left(\widehat{S}_t\right)\right)^T\left(\widehat{Q}^{\pi}\left(\widehat{S}_t, \mathscr{A}\right) - \widehat{Q}\left(\widehat{S}_t, \mathscr{A}; \Theta_{t-1}\right)\right). \tag{32}$$

Recall the fact that we once used $k\overline{\boldsymbol{\phi}}_t\overline{\boldsymbol{\phi}}_t^T$ and $k\overline{\boldsymbol{\phi}}_t\overline{\mathbf{q}}_t^{\pi}$ in equations (23) and (24) to replace $(1/M)(\Phi(\widehat{S}_t))^T\Phi(\widehat{S}_t)$ and $(1/M)(\Phi(\widehat{S}_t))^T\widehat{Q}^{\pi}(\widehat{S}_t, \mathscr{A})$ in equations (19) and (20), respectively, which means

$$k\overline{\boldsymbol{\phi}}_t\overline{\boldsymbol{\phi}}_t^T = \frac{1}{M}\left(\Phi\left(\widehat{S}_t\right)\right)^T\Phi\left(\widehat{S}_t\right), \tag{33}$$

$$k\overline{\boldsymbol{\phi}}_t\overline{\mathbf{q}}_t^{\pi} = \frac{1}{M}\left(\Phi\left(\widehat{S}_t\right)\right)^T\widehat{Q}^{\pi}\left(\widehat{S}_t, \mathscr{A}\right). \tag{34}$$

In addition, from equation (31), we can obtain

$$k\overline{\boldsymbol{\phi}}_t\overline{\mathbf{q}}_t = \frac{k}{M}\overline{\boldsymbol{\phi}}_t\sum_{i=1}^{M}\widehat{Q}\left(\widehat{\mathbf{s}}_{t,i}, \mathscr{A}; \Theta_{t-1}\right). \tag{35}$$

Using equation (9) yields

$$\widehat{Q}\left(\widehat{\mathbf{s}}_{t,i}, \mathscr{A}; \Theta_{t-1}\right) = \boldsymbol{\phi}\left(\widehat{s}_{t,i}\right)^T\Theta_{t-1}. \tag{36}$$

Then, equation (35) can be written as

$$k\overline{\boldsymbol{\phi}}_t\overline{\mathbf{q}}_t = \frac{k}{M}\overline{\boldsymbol{\phi}}_t\sum_{i=1}^{M}\boldsymbol{\phi}\left(\widehat{\mathbf{s}}_{t,i}\right)^T\Theta_{t-1}. \tag{37}$$

Further, from equation (25), the above equation can be written as

$$k\overline{\boldsymbol{\phi}}_t\overline{\mathbf{q}}_t = k\overline{\boldsymbol{\phi}}_t\overline{\boldsymbol{\phi}}_t^T\Theta_{t-1}. \tag{38}$$

Next, plugging equation (33) into equation (38), we have

$$k\overline{\boldsymbol{\phi}}_t\overline{\mathbf{q}}_t = \frac{1}{M}\left(\Phi\left(\widehat{S}_t\right)\right)^T\Phi\left(\widehat{S}_t\right)\Theta_{t-1}. \tag{39}$$

From equations (9) and (11), the above equation can be rewritten as

$$k\overline{\boldsymbol{\phi}}_t\overline{\mathbf{q}}_t = \frac{1}{M}\left(\Phi\left(\widehat{S}_t\right)\right)^T Q\left(\widehat{\mathbf{s}}_{t,i}, \mathscr{A}; \Theta_{t-1}\right). \tag{40}$$

Using equations (34) and (40), we can get

$$k\overline{\boldsymbol{\phi}}_t\left(\overline{\mathbf{q}}_t^{\pi} - \overline{\mathbf{q}}_t\right) = -\nabla_{\Theta_{t-1}}. \tag{41}$$

Therefore, we can rewrite equation (30) as

$$\Theta_t \approx \Theta_{t-1} - \frac{P_{t-1}}{\lambda + k\mathbf{v}_t^T\overline{\boldsymbol{\phi}}_t}\nabla_{\Theta_{t-1}}. \tag{42}$$

It shows that $(P_{t-1}/(\lambda + k\mathbf{v}_t^T\overline{\boldsymbol{\phi}}_t))$ is the learning rate of Θ_t in MRLS-Q.

However, although RLS has a fast convergence rate, it often suffers from overfitting. In recent years, there has been extensive research on this problem. Based on Ekşioğlu's work [30], we add an L_1 regularization term into the above equation, i.e.,

$$\Theta_t \approx \Theta_{t-1} - \frac{P_{t-1}}{\lambda + k\mathbf{v}_t^T\overline{\boldsymbol{\phi}}_t}\nabla_{\Theta_{t-1}} - \eta P_t\operatorname{sgn}\left(\Theta_{t-1}\right), \tag{43}$$

where η is the regularization factor and $\operatorname{sgn}(\cdot)$ is the sign function.

Based on the above derivation, the pseudocode of MRLS-Q is summarized in Algorithm 1, and the flow diagram of MRLS-Q is summarized in Figure 1. In the practical implementation, here $\nabla_{\Theta_{t-1}}$ can be calculated by the automatic differentiation package of PyTorch or TensorFlow directly. Besides being used alone, MRLS-Q can also be used as the last layer of DQN, since it uses the same loss function and experience replay as DQN. However, there is still an obstacle to the combination of MRLS-Q and DQN. As the training goes on, the parameters of the DQN network are continuously changing, and the outputs of the same inputs are changing as well. Thus, we cannot use the inputs of the DQN's last layer as the features of MRLS-Q directly. In order to alleviate this kind of change and integrate MRLS-Q into DQN, we present a new method to define the feature function of MRLS-Q as

$$\Phi\left(\widehat{S}_n\right) = \frac{X_t^L}{\left(1/MN_{L-1}\right)\sum_{i=1}^{M}\sum_{j=1}^{N_{L-1}}X_{t,i,j}^L + \nu}, \tag{44}$$

where $X_t^L \in \mathscr{R}^{M\times N_{L-1}}$ is the output matrix of the DQN's penultimate layer and ν is a small hyperparameter to prevent the denominator becoming zero.

4. Experiments

In this section, we use two sets of experiments to demonstrate the effectiveness of MRLS-Q. Our experiments are divided into two sections. In Section 4.1, we test MRLS-Q on the CartPole problem as an independent algorithm. In Section 4.2, we test MRLS-Q on four Atari games as the last layer of DQN.

4.1. The CartPole Problem. In this set of experiments, we firstly verify the performance of MRLS-Q on the CartPole-v0 problem, which is from the OpenAI Gym. For comparison

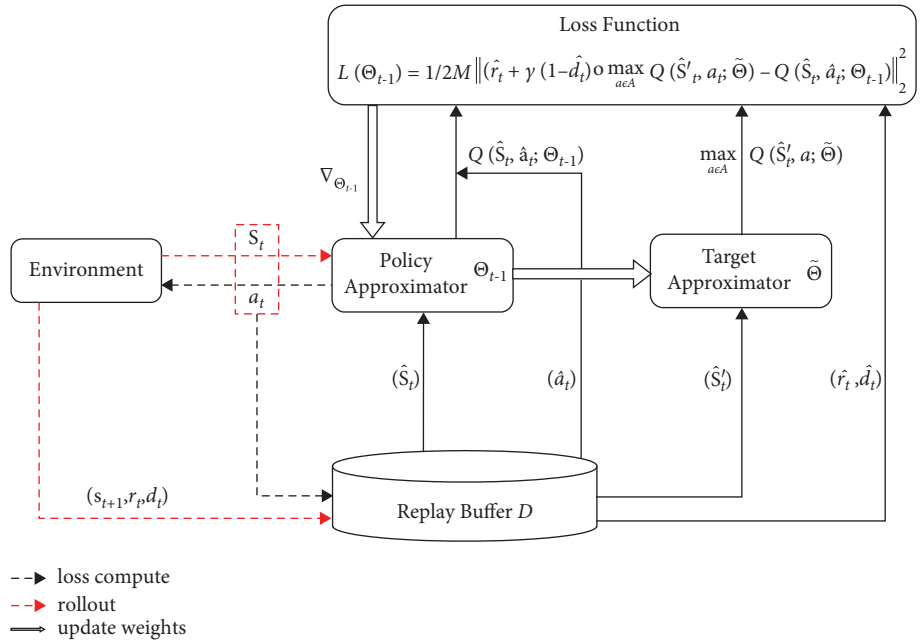

Figure 1: Flow diagram of MRLS-Q.

(1) **input** $\{\phi_l(\cdot)\}_{l=1,\ldots,N}$, ϵ, γ, λ, k, η, M and **initialize** $\mathscr{D} =$, Θ_0, $\tilde{\Theta}$, $P_0 = \alpha I$
(2) **for** episode = 1, . . ., *MaxEpsidoes* **do**
(3) initialize state $\mathbf{s}_1$
(4) **for** t = 1, . . ., *MaxSteps* **do**
(5) select action a_t with ϵ- greedy policy, and take action a_t
(6) measure next state $\mathbf{s}_{t+1}$, reward r_t and terminal state d_t
(7) store $(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}, r_t, d_t)$ into $\mathscr{D}$
(8) sample minibatch $\mathscr{M}_t$ from $\mathscr{D}$
(9) compute loss function $L(\Theta_{t-1})$ according to (11), (12) and (4)
(10) compute gradient $\nabla_{\Theta_{t-1}}$ according to (32)
(11) update Θ_t and P_t according to (43) and (27)
(12) update $\tilde{\Theta}$ by Θ_t every fixed steps or episodes
(13) **if** d_t == 1 **or** t == MaxSteps **do**
(14) set $P_0 = P_t$ and $\Theta_0 = \Theta_t$, and **break** the inner loop
(15) **end if**
(16) **end for**
(17) **end for**

Algorithm 1: MRLS-Q.

purposes, we build a new algorithm called Adam-Q, by replacing $P_{t-1}/(\lambda + k\mathbf{v}_t^T \overline{\phi}_t)$ in equation (42) with the Adam optimizer, since the traditional Q-learning algorithm with linear function approximation is hardly convergent in 100 episodes. Then, we verify the influences of hyperparameters on MRLS-Q, experimentally.

To compare the performance between MRLS-Q and Adam-Q, the experimental settings are summarized as follows. (1) Both algorithms use 400 radial basis functions (RBFs) for action-value approximation. These RBFs are generated from 10^4 random samples in the CartPole's state space, by using eight scikit-learn RBFSamplers [31] with kernel parameters {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0}. (2) The exploration rate ϵ is initialized to 0.95 and is gradually decreased to 0.01 over 1000 steps. (3) The discount factor γ is 0.99. (4) The capacity of the experience replay buffer $\mathscr{D}$ is 10^4, and the minibatch size is 32. The learning starts when the number in $\mathscr{D}$ reaches the minibatch size. (5) The policy approximator parameter Θ_0 and the target approximator parameter $\tilde{\Theta}$ are initialized randomly. (6) $\tilde{\Theta}$ is updated by Θ_t each step. Note that the performances of both algorithms will get worse if we increase the update steps, since the CartPole problem is very simple and both algorithms converge fast. (7) The max norm of $\nabla_{\Theta_{t-1}}$ is clipped to 1 by the L_2 norm. (8) The two algorithms run five times and 100 episodes for each time. In each episode, each algorithm runs

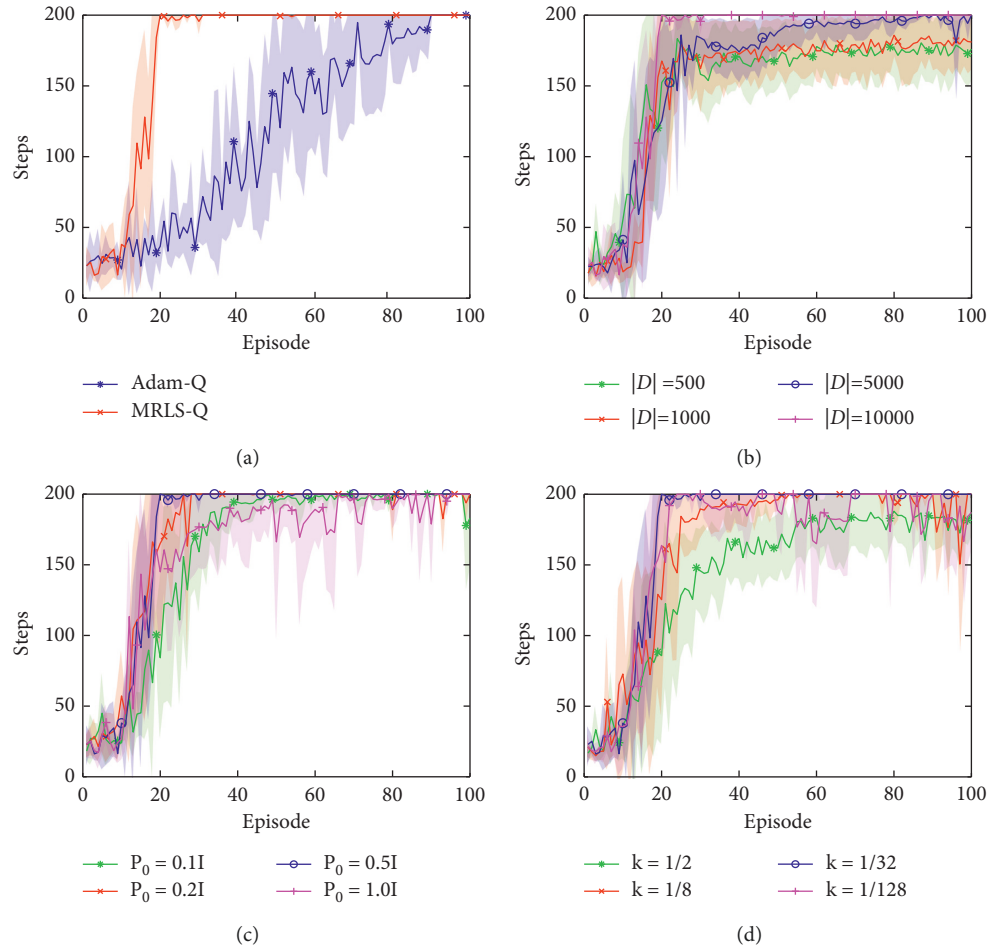

FIGURE 2: Performance comparison and hyperparameter influences on MRLS-Q. (a) Adam-Q vs. MRLS-Q. (b) Influence of $\mathscr{D}$'s capacity. (c) Influence of P's initialization. (d) Influence of k value.

200 steps at most. (9) Besides, in Adam-Q, the learning rate, β_1, and β_2 of Adam are 0.001, 0.9, and 0.999; in MRLS-Q, the initialization P_0, the forgetting factor λ, the regularization factor η, and the approximation factor k are $0.5I$, 1, 10^{-5}, and 1/32, respectively. The average result of this experiment is shown in Figure 2(a). It can be seen that our MRLS-Q has better convergence than Adam-Q.

To investigate hyperparameter influences on MRLS-Q, we test $|\mathscr{D}| \in \{500, 1000, 5000, 10000\}$, $P_0 \in \{0.1I, 0.2I, 0.5I, I\}$, and $k \in \{(1/2), (1/8), (1/32), (1/128)\}$, respectively. The other settings of these experiments are the same as what we did for MRLS-Q in the previous experiment. The average results of these experiments are presented in Figures 2(b)–2(d). From Figure 2(b), it shows that the capacity of $\mathscr{D}$ has a significant influence on the performance of MRLS-Q. The larger capacity will result in the better performance, since big $\mathscr{D}$ is helpful to remove the correlation between the observed transitions. From Figure 2(c), it can be seen that MRLS-Q is robust to the initialization P_0, whereas too big P_0 will make MRLS-Q become unstable and too small P_0 will make MRLS-Q converge slowly. From Figure 2(d), it can be seen that k also has a significant influence on MRLS-Q. From equation (29), bigger k will make P_t update with higher strength. If state feature values change greatly, k should be set to a big value.

4.2. Four Atari Games. In this set of experiments, we verify MRLS-Q as the last layer of DQN on four Atari games: Pong-v0, Breakout-v0, SpaceInvaders-v0, and RiverRaid-v0, which are from the OpenAI Gym. Here we choose the traditional DQN algorithm with the Adam optimizer for comparison. For Adam-DQN and in the second to fifth layers of Hybrid-DQN, the learning rate, β_1, and β_2 of Adam are 0.0000625, 0.9, and 0.999; in the last layer of Hybrid-DQN, the initialization P_0, the forgetting factor λ, the regularization factor η, the approximation factor k, and ν are 0.1I, 1, 10^{-8}, 1/2, and 10^{-12}, respectively. Note that here we use a big k to update P_t for adapting to the feature change.

The average evaluation results are presented in Figure 3. It shows that Hybrid-DQN can speed up the convergence of all tested games. Figure 3(a) is much clearer to demonstrate this advantage, since the Pong game is much simpler than other three games. In addition, Figures 3(a), 3(c), and 3(d) show that Hybrid-DQN can improve the convergence quality of Pong, SpaceInvaders, and RiverRaid, and

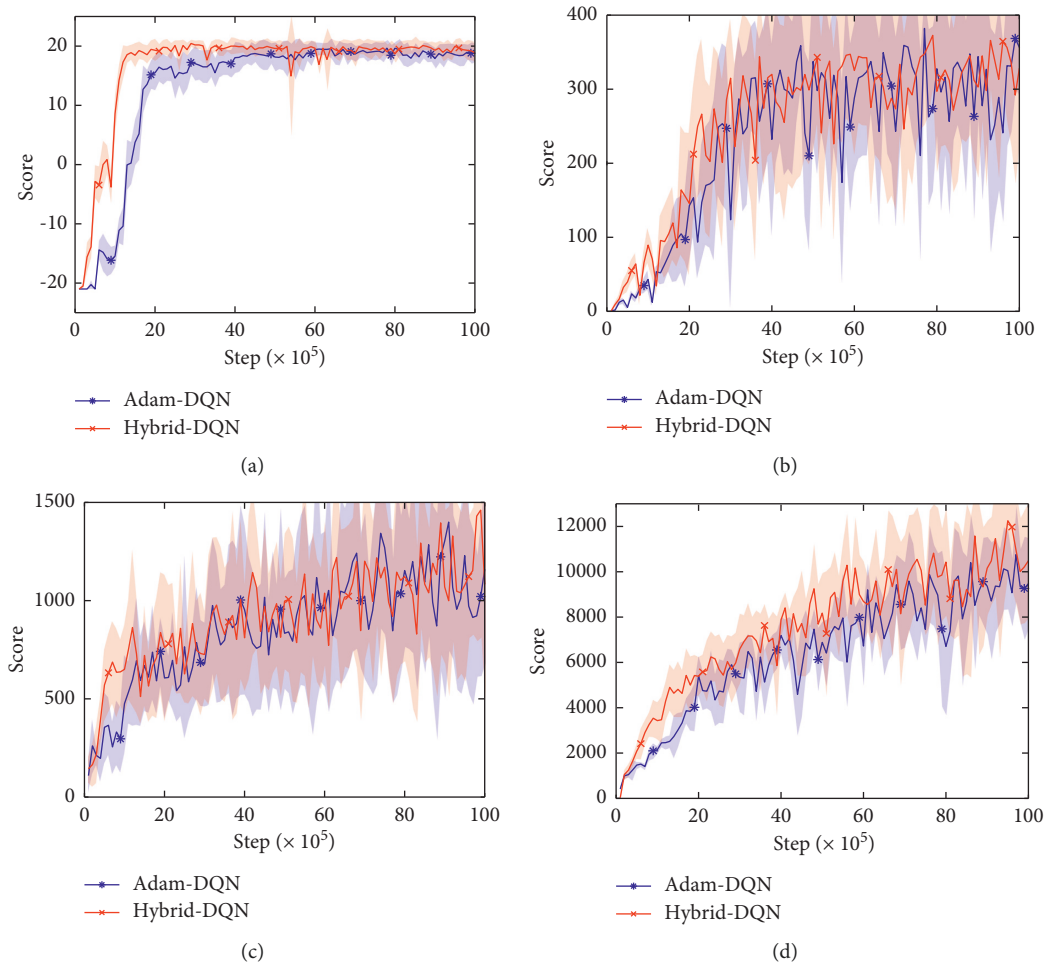

Figure 3: Performance comparison between Adam-DQN and Hybrid-DQN. (a) Pong. (b) Breakout. (c) SpaceInvaders. (d) RiverRaid.

Figure 3(b) shows that Hybrid-DQN can improve the learning stability of Breakout. In summary, by integrating our MRLS-Q, Hybrid-DQN can improve the stability and performance. Compared with the LS-DQN algorithm, MRLS-Q can be used as the last layer of DQN directly, and thus Hybrid-DQN is easier to use.

5. Conclusion

How to improve convergence and stability of the DQN algorithm is one of the key issues in deep RL. In this paper, we propose MRLS-Q, a linear RLS function approximation algorithm with the similar learning mechanism to DQN. MRLS-Q can be used not only alone but also as the last layer of DQN. Similar to LS-DQN, the Hybrid-DQN with MRLS-Q can enjoy rich representations from deep RL networks as well as stability and data efficiency of the RLS method, but it can seamlessly integrate MRLS-Q and thus is easier to use. In MRLS-Q, we use the experience replay to break the correlation between training samples, present an average RLS optimization method to improve the convergence performance and reduce the computational complexity, employ an L_1 regularization technique to prevent overfitting, and propose a new method to define the feature function for alleviating the feature change of the same state and integrating MRLS-Q into DQN. Experiment results on the CartPole problem demonstrate that MRLS-Q has better convergence than Adam-Q and reveal the hyperparameter influences on MRLS-Q. In addition, experiment results on four Atari games demonstrate that DQN can improve convergence and stability by integrating with MRLS-Q.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no known conflicts of interest or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was supported by the National Natural Science Foundation of China (grant nos. 61762032 and 11961018).

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, USA, 2nd edition, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," in *Proceedings of the 27th Conference on Neural Information Processing Systems Deep Learning Workshop*, Lake Tahoe, USA, 2013.

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

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, pp. 1856–1865, Stockholm, Sweden, July 2018.

[5] D. Kim, M. Liu, M. Riemer et al., "A policy gradient algorithm for learning to learn in multiagent reinforcement learning," in *Proceedings of the 38th International Conference on Machine Learning*, pp. 5541–5550, 2021.

[6] C. Zheng, S. Liu, Y. Huang, and L. Yang, "Hybrid policy learning for energy-latency tradeoff in MEC-assisted VR video service," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, p. 1, 2021.

[7] B. Gu, X. Yang, Z. Lin, W. Hu, M. Alazab, and R. Kharel, "Multiagent actor-critic network-based incentive mechanism for mobile crowdsensing in industrial systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 9, pp. 6182–6191, 2021.

[8] P. Wang and C. Chan, "Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge," in *Proceedings of the 20th International Conference on Intelligent Transportation Systems*, 2017.

[9] B. Yang and M. Liu, "Keeping in touch with collaborative UAVs: a deep reinforcement learning approach," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 562–568, 2018.

[10] L. Liu, B. Tian, X. Zhao, and Q. Zong, "UAV autonomous trajectory planning in target tracking tasks via a DQN approach," in *Proceedings of the 2019 International Conference on Real-time Computing and Robotics*, Irkutsk, Russia, August 2019.

[11] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 2094–2100, Phoenix, AZ, USA, February 2016.

[12] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning*, pp. 1995–2003, New York, NY, USA, June 2016.

[13] M. J. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proceedings of the 2015 AAAI Fall Symposia*, pp. 29–37, Arlington County, VA, USA, November 2015.

[14] S. Kim, K. Asadi, M. Littman, and G. Konidaris, "DeepMellow: removing the need for a target network in deep Q-learning," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 2733–2739, Macao, China, August 2019.

[15] O. Anschel, N. Baram, and N. Shimkin, "Averaged-DQN: variance reduction and stabilization for deep reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 176–185, Sydney, Australia, August 2017.

[16] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proceedings of the 4th International Conference on Learning Representations*, Juan, Puerto Rico, May 2016.

[17] M. Fortunato, M. G. Azar, B. Piot et al., "Noisy networks for exploration," in *Proceedings of the 6th International Conference on Learning Representations*, Vancouver, Canada, 2018.

[18] S. Y. Lee, S. Choi, and S. Chung, "Sample-efficient deep reinforcement learning via episodic backward update," in *Proceedings of the 33rd Conference on Neural Information Processing Systems*, pp. 2110–2119, Vancouver, Canada, 2019.

[19] V. Mnih, A. P. Badia, M. Mirza et al., "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33nd International Conference on Machine Learning*, pp. 1928–1937, New York, NY, USA, June 2016.

[20] N. Levine, T. Zahavy, D. J. Mankowitz, A. Tamar, and S. Mannor, "Shallow updates for deep reinforcement learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 3135–3145, Long Beach, CA, USA, December 2017.

[21] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[22] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[23] S. Baldi, Z. Zhang, and D. Liu, "Eligibility traces and forgetting factor in recursive least-squares-based temporal difference," *International Journal of Adaptive Control and Signal Processing*, 2021.

[24] G. Huang, Q. Zhu, and C. K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1–2, pp. 489–501, 2006.

[25] C. L. P. Chen and Z. Liu, "Broad learning system: an effective and efficient incremental learning system without the need for deep architecture," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 1, pp. 10–24, 2018.

[26] D. Liu, S. Baldi, W. Yu, and C. L. P. Chen, "A hybrid recursive implementation of broad learning with incremental features," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2020.

[27] C. Zhang, C. Liu, Q. Song, and J. Zhao, "Recursive least squares policy control with echo state network," in *Proceedings of the 4th International Conference on Artificial Intelligence and Big Data*, pp. 104–108, Zibo, China, September 2021.

[28] L. Bottou, *Online Learning and Stochastic Approximations*, Cambridge University Press, Cambridge, UK, 2nd edition, 1998.

[29] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.

[30] E. M. Ekşioğlu, "RLS adaptive filtering with sparsity regularization," in *Proceedings of the 10th International Conference on Information Science, Signal Processing and their Applications*, pp. 550–553, Kuala Lumpur, Malaysia, May 2010.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit-learn: machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.