Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 1596385, 9 pages
https://doi.org/10.1155/2020/1596385

Research Article

Deep Q-Network with Predictive State Models in Partially Observable Domains

Danning Yu, Kun Ni, and Yunlong Liu

Department of Automation, Xiamen University, Xiamen 361005, China

Correspondence should be addressed to Yunlong Liu; ylliu@xmu.edu.cn

Received 23 September 2019; Revised 6 June 2020; Accepted 22 June 2020; Published 16 July 2020

Academic Editor: Petr Hájek

While deep reinforcement learning (DRL) has achieved great success in some large domains, most of the related algorithms assume that the state of the underlying system is fully observable. However, many real-world problems are actually partially observable. For systems with continuous observation, most of the related algorithms, e.g., the deep Q-network (DQN) and deep recurrent Q-network (DRQN), use history observations to represent states; however, they often make computation-expensive and ignore the information of actions. Predictive state representations (PSRs) can offer a powerful framework for modelling partially observable dynamical systems with discrete or continuous state space, which represents the latent state using completely observable actions and observations. In this paper, we present a PSR model-based DQN approach which combines the strengths of the PSR model and DQN planning. We use a recurrent network to establish the recurrent PSR model, which can fully learn dynamics of the partially continuous observable environment. Then, the model is used for the state representation and update of DQN, which makes DQN no longer rely on a fixed number of history observations or recurrent neural network (RNN) to represent states in the case of partially observable environments. The strong performance of the proposed approach is demonstrated on a set of robotic control tasks from OpenAI Gym by comparing with the technique with the memory-based DRQN and the state-of-the-art recurrent predictive state policy (RPSP) networks. Source code is available at https://github.com/RPSR-DQN/paper-code.git.

1. Introduction

For agents operating in stochastic domains, how to determine the (near) optimal policy is a central and challenge issue. While (deep) reinforcement learning has provided a powerful framework for decision-making and control and has achieved great success in recent years in some large-scale applications, e.g., AlphaGo [1], most of the related approaches rely on the strong assumption that the agent can completely know the environment surrounded it, i.e., the environment is fully observable. However, for many real-world applications, the problem is actually partially observable Markov decision process (POMDP) where the state of the environment may be partially observable or even unobservable [2, 3].

Much effort has been devoted to planning in partially observable environments. Some of the work aims for learning the complete model of the underlying system. Huang et al. [4–6] propose the planning methods based on the PSR model. Song et al. [7] and Somani et al. [8] propose the planning method based on the POMDP model. However, these methods are only suitable for systems with discrete observations. In this paper, we mainly focus on systems with continuous observations, and there are two main approaches for dealing with the partially observable problem in such domain. One relies on recurrent neural networks to summarize the past, and then the neural network is trained in a model-free reinforcement learning manner [2, 9, 10]. However, it will be a heavy burden for the training of networks when everything relies on it. The other approach for dealing with the partially observable problem is directly using the past histories, i.e., the past observations (frames), for the state representation, and the main problem of this approach is that the number of observations (frames)

used for the state representation can only be determined empirically. Also, too many observations for the state representation may be computation-expensive, but few observations may not be sufficient statics of the past. And neither method considers the effect of action information on state representation.

Predictive state representations (PSRs) provide a general framework for modelling partially observable systems, and unlike the latent-state based approaches, such as POMDPs, the core idea of PSRs is to work only with the observable quantities, which leads to easier learning of a more expressive model [11–13]. PSRs can also combine with the recurrent network for the modelling and planning in partially observable dynamic systems with continuous state space [14, 15].

In this paper, with the benefits of the PSR approach and the great success of deep Q-network in some real-world applications, we propose the RPSR-DQN approach; firstly, a recurrent PSR model of the underlying partially observable systems is built, then the truly state, namely, the PSR state or the belief state, can be updated and provide the sufficient information for DQN planning, and finally, the tuple of $<currentPSRstate, action, reward, nextPSRstate>$, where $currentPSRstate$ is the information of the current state and $nextPSRstate$ is the information of the next state obtained by taking action under the current state, is stored and used as the data for the training of the deep Q-network. The performance of our proposed approach is firstly demonstrated on a set of robotic control tasks from OpenAI Gym by comparing with the deep recurrent Q-network (DRQN) algorithm which uses current observation as the input and plans based on memory. Then, we compare our approach with the state-of-the-art recurrent predictive state policy (RPSP) networks [14]. Experiment results show that with the benefits of the DQN framework and the dividing of the learning of the model and the training of the policy, our approach outperforms the state-of-the-art baselines.

2. Related Work

A central problem in artificial intelligence is for agents to find optimal policies in stochastic, partially observable environments, which is an ubiquitous and challenging problem in science and engineering [16]. The commonly used technique for solving such partially observable problems is to model the dynamics of the environments by using the POMDP approach or the PSR approach firstly [3, 12] and then the problem can be solved using the obtained model. Although POMDPs provide general frameworks to solve partially observable problems, it relies heavily on a known and accurate model of the environment [17]. Therefore, in real-world applications, it is extremely difficult to build an accurate model [18]. Also, most of the POMDP-based approaches have difficulties to be extended to some larger-scale real-world applications.

As mentioned previously, PSR is an effective method for modelling partially observable environment and many related works were proposed based on the idea of running a fully observable RL method on the PSR state. In the work of Boots et al. [19], the main idea of it is firstly building accurate enough transformed PSRs with indicative and characteristic features and then the point-based value iteration technique [20] is used for finding the planning solution, where a state subset B in the state space is firstly selected under some strict conditions that B is both sufficiently small to reduce the computational difficulty and sufficiently large to obtain a good approximation function. In the work of Liu and Zheng [5, 21], the learned PSR model has been combined with Monte-Carlo tree search both online and offline, which achieves the state-of-the-art performances on some environments. However, the application of these proposed approaches is limited to domains with discrete state and action spaces.

For partially observable systems with continuous state space, most work relies on recurrent neural networks to summarize the past and then the neural network is trained in a model-free reinforcement learning manner. In order to solve the customer relationship management (CRM) problem that is considered to be partially observable, Li et al. [22] proposed a hybrid recurrent reinforcement learning approach (SL-RNN + RL-DQN) which uses the RNN to calculate the hidden states of the CRM environment. While our method was tested on some control environments as shown in the experiments and takes into account both the past observations and actions for the representation of the underlying states, for SL-RNN + RL-DQN, both the proposed approach and the related experiments focus on the CRM problem. Also, SL-RNN + RL-DQN does not consider the effect of action value when calculating the state representation, which may incur the inaccurate representations of the underlying states. Moreover, while RPSR-DQN tries to build the model of the underlying system, which makes the related approaches be easily extended to model-based reinforcement learning approaches, SL-RNN + RL-DQN can only be combined with the model-free reinforcement learning frameworks. In the work of Hausknecht and Stone [9], recurrence is added to a deep Q-network (DQN) by replacing the first fully connected layer with a recurrent LSTM by considering all historical information. Igl et al. [2] extended the RNN-based approach to explicitly support belief inference. However, while in our approach, with suitable features, the mapping between the predictive state and the prediction of the observations given the actions can be fully known and simple to be learned consistently, the main problem of these RNN-based approaches with latent states is in these recurrent models, nonconvex optimization is used, which usually leads to more difficult training than those using convex optimization [14].

Recently, some works have been proposed by using the PSR state for the replacement or quality improvement of the internal state of the RNN. In the work of Venkatraman et al. [15], recurrent neural networks are combined with predictive state decoders (PSDs), which add supervision to the network internal state representation to target predicting future observations. Hefny et al. [14] proposed recurrent predictive state policy (RPSP) networks, which consist of a recursive filter for the tracking of a belief about the state of the environment, and a reactive policy that directly maps

beliefs to actions, to maximize the cumulative reward. While RPSP networks show some promising performances on some benchmark domains, the recursive filter and the reactive policy are trained simultaneously by defining a joint loss function in an online manner. However, how to balance the loss of the recursive filter and the loss of the reactive policy is difficult, and in many cases, as also shown in the experiments, the simultaneously training of two objective functions may lead to a worse final performance.

3. Background

This section is divided into three parts. In the first part, we briefly review predictive state representations (PSRs) [12]. Then, we introduce the recurrent PSRs that can be applied to continuous observation systems. Finally, we briefly describe the DQN algorithm.

3.1. Predictive State Representations. Predictive state representations (PSRs) offer a powerful framework for modelling partially observable and stochastic systems without prior knowledge by using completely observable events to represent states [23]. For discrete systems with a finite set of observations $O = \{o^1, o^2, \ldots, o^{|O|}\}$ and actions $A = \{a^1, a^2, \ldots, a^{|A|}\}$, at time τ, the observable state representation of the system is a prediction vector composed of the probability of test occurrence conditioned on current history, where a *test* is a sequence of action-observation pairs that starts from time $\tau + 1$, a *history* at time τ is a sequence of action-observation pairs that starts from the beginning of time and ends at time τ, and the prediction of a length m and test t at history h is defined as $p(t \mid h) = p(ht)/p(h) = \prod_{i=1}^{m} \Pr(o_i \mid ha_1 o_1 \cdots a_i)$ [24].

Given the set of tests $T = \{t^1, t^2, \ldots, t^{|K|}\}$, if the prediction vector $p(t \mid h) = [p(t^1 \mid h)p(t^2 \mid h) \ldots p(t^{|K|} \mid h)]$ satisfies that for any test t, there exists a function f_t such that $p(t \mid h) = f_t(p(T \mid h))$, then T is considered to constitute a PSR. The set T is called the test core, and the prediction vector $(p(T \mid h))$ is called the PSR state. In this paper, we only consider linear PSRs, so the function f_t can be represented as the weight vector m_t. When the action a is performed from the history h and the observation o is obtained, the next PSR state $p(T \mid hao)$ can be updated from $p(T \mid h)$ as follows [12]:

$$p(T \mid hao) = \frac{p(T \mid h)^T M_{ao}}{p(T \mid h)^T m_{ao}}. \tag{1}$$

In formula (1), ^{T} is the transposing operation, the m_{ao} is a weight vector of the test ao, and the M_{ao} is a $K \times K$ matrix with the ith column corresponding to weight vector m_{aot_i}.

3.2. Recurrent Predictive State Representation. The PSR model obtained by using the substate-space method [25] or spectral learning algorithm [26] can only be applied to the modelling of the discrete observation system. More recently, Ahmed et al. [27] proposed the recurrent predictive state representation (RPSR) which treats predictive state models as the recurrent network. It is able to represent systems with continuous observations. Similar to PSR, the RPSR state p_t is the conditional distribution of future observations, so the mapping between the RPSR state p_t and the predictive observation o_t obtained for the given action can be fully known or easy to learn by selecting of features. This characteristic makes the process of learning networks become the supervised learning, which makes the modelling be simple and efficient [28, 29].

The state update process of RPSR can be divided into two steps. As can be seen from Section 3.1, if T is the test core, the $p(T \mid h_t)$ is a sufficient state representation at time t. Then, establishing an extended test core $\widehat{T}$ ensures that the $p(\widehat{T} \mid h_t)$ is the sufficient statistic of the distribution $\Pr(a_t, o_t, T \mid h_t)$ for any a_t, o_t. When the estimate of $p(\widehat{T} \mid h_t)$ is given, the $p(T \mid h_{t+1})$ can be obtained in the case of getting a_t, o_t. The $p(\widehat{T} \mid h_t)$ is called the extended state q_t. The steps of state update are as follows [14]:

(i) State extension: the state p_t transforms to the extended state q_t through the linear map W_{ext}. W_{ext} is a parameter that needs to be learned:

$$q_t = W_{\text{ext}} p_t. \tag{2}$$

(i) Conditioning: given a_t and o_t, the next state p_{t+1} can be calculated from the current extended state q_t by the conditioning function f_{cond}, where the kernel Bayes rule with the Gaussian RBF kernel is used [30]:

$$p_{t+1} = f_{\text{cond}}(q_t, o_t, a_t), \tag{3}$$

where the calculation detail is as follows: as the extended feature is a Kronecker product of the immediate feature matrix and the future feature matrix, the extended state can be further divided into two parts, which are derived from the skipped future observation and the present observation, respectively. Then, firstly, the feature vectors $\phi(a_t)$ and $\phi(o_t)$ are extracted for a given action a_t and observation o_t. Secondly, $\phi(a_t)$ and the second part of the expanded state are multiplied to calculate the observation covariance after a_t is executed, and the inverse observation covariance is multiplied by the first part of the expanded state to change "predicting observation" into "conditioning on observation", which is transformed from the joint expectation of immediate ao and T to the conditional expectation from immediate ao to T. Finally, the conditional expectation is multiplied by $\phi(a_t)$ and $\phi(o_t)$ to obtain the next state p_{t+1}.

The RPSR model can be seen as a recursive filter which is implemented by transforming formulas (2) and (3) to the recurrent network. The output of the recurrent network is a predictive observation $\widehat{o}_t = W_{\text{pred}}(p_t, a_t)$, where the W_{pred} is the predictive observation function that needs to be learned. The p_t and q_t are represented in terms of observation quantities and can be estimated by supervised regression. The W_{ext} follows from linearity formula (2). So, in the process of network training, the two-stage regression method [28] is used to initialize the state p_t, extended state q_t, and the linear map W_{ext}.

3.3. Deep Q-Network. DQN is a method combining deep learning and Q-learning, which has succeeded in handling environments with high-dimensional perception input [31]. It is a multilayered neural network which outputs a predicted future reward $Q(s, a \mid \theta)$ for each possible action, where θ are the network parameters. In other words, DQN uses a neural network as an approximation of the action value.

In DQN, the last four frames of the observations are directly input to the CNN as the first layer of DQN to compute the current state information. Then, the state information is mapped to a vector of action values for the current state through the full connection layer [32]. DQN optimizes the action value function by updating the network weights θ to minimize a differentiable loss function $L(\theta)$ [9]:

$$L(\theta) = \left(r + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1} \mid \theta\right) - Q\left(s_t, a_t \mid \theta\right) \right)^2. \quad (4)$$

4. RPSR-DQN

With the benefits of the RPSR approach and the great success of deep Q-network, we propose a model-based method, which combines the RPSR with the deep Q-learning. Firstly, we use the recurrent network to build a PSR model of the partially observable dynamic systems. Then, the truly state p_t, namely, the RPSR state, can provide sufficient information for selecting best action and be updated with the new action a_t executed and the new observation o_t received. Finally, the tuple of $< p_t, a_t, r_t, p_{t+1} >$, where r_t represents the return reward for taking action a_t in the current state p_t, is stored and used as the data for the training of the deep Q-network (DQN).

As depicted in Figure 1, the architecture of our method consists of the RPSR model part and the value-based policy part. In the RPSR model, the state p_t transforms to the extended state q_t through the extended part, i.e., a liner map. Then, extended state q_t updates to the next state p_{t+1} according to the action a_t and observation o_t. The total state update process can be represented as formula (5). For the policy part, the deep Q-network is used to select the action which can get better long-term reward according to the current state information calculated by the RPSR model:

$$p_{t+1} = f_{\mathrm{RPSR}}(p_t, o_t, a_t). \quad (5)$$

The learning process is divided into two stages: building the model and training the policy network. In the first stage, an exploration strategy is used to collect training data to build the model of the environment. We use the data-processing method proposed by Ahmed et al. [27]. We use the 1000 random Fourier features (RFFs) [33] as approximate features of observations and actions. Then, we apply principal component analysis (PCA) [34] to features to project into 25 dimensions. Here, the number of features and dimensions depends on the complexity of the environment. We denote the feature function as ϕ. The linear map W_{ext}, states p_t, and extend states q_t in the RPSR model are initialized by using a two-stage regression algorithm [28]. Use $\varphi_t^O = \phi(o_{(t:t+k-1)})$, $\varphi_t^A = \phi(a_{(t:t+k-1)})$, $\zeta_t^O = \phi(o_{(t:t+k)})$, and $\zeta_t^A = \phi(a_{(t:t+k)})$ to denote sufficient features of future observations, future actions, extended future observations, and extended future actions at time t, respectively. Because the p_t and q_t are represented in terms of observable quantities and follow from linearity of expectation, they are computed by using the kernel Bayes rule (stage-1 regression). Whereafter, the state extension function is $q_t = W_{\mathrm{ext}} p_t$, so we can linearly regress the extended state q_t from the state p_t, using a least squares approach (stage-2 regression), to compute W_{ext}. After initialization, the parameters θ_{RPSR} of the RPSR model can be optimized by using backpropagation through time [35] to minimize prediction error (see formula (6)), where $\widehat{o}_t$ is the predictive observation of the RPSR model:

$$L(\theta_{\mathrm{RPSR}}) = \sum_{t=0}^{T} \left\| \widehat{o}_t - o_t \right\|^2. \quad (6)$$

After the model is established for the dynamic environment, the current state information of the partially observable environment can be expressed by the model, and the policy part is trained on this basis. In the process of policy training, we build the evaluation network and target network, which are both composed of two fully connected layers. We use the experience replay [32] to train networks. When the agent interacts with the environment, we store transitions (p_t, r_t, a_t, p_{t+1}) in the data set D. Then, sample random transitions to train the policy network by minimizing the value difference between the target network and evaluation network. These losses are backpropagated into the weights of both the encoder and the Q-network. The value of the target network is $R = r_t + \gamma \max_{a'} Q(p_{t+1}, a'; \theta^-_{\mathrm{policy}})$, where $\theta^-_{\mathrm{policy}}$ denotes the parameters of an earlier Q-network. The details are shown in Algorithm 1.

5. Experiments

We select the following three gym environments for evaluating the RPSR-DQN performance (see Figure 2): the traditional control environment CartPole-v1, the MuJoCo robot environment Swimmer-v1, and Reacher-v1. These environments provide qualitatively different challenges. Due to the setting of experimental conditions, we make some changes to the three environments.

CartPole-v1: this task is controlled by applying left or right force to the cart to move the cart to the left or right. A reward of +1 is provided for every time step that the angle of the pole is less than 15 degrees. The episode is terminated when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. The goal is to prevent the pole which is attached by an unactuated joint to a cart from falling over. There are two action values in this environment, that is, the direction of the force applied to the cart. To make the environment partially observable, we remove the observations that represent the velocity, changing the original four observations to two observations which are the position of the cart and the angle of the pole. So, it requires the ability to calculate speed based on positions.

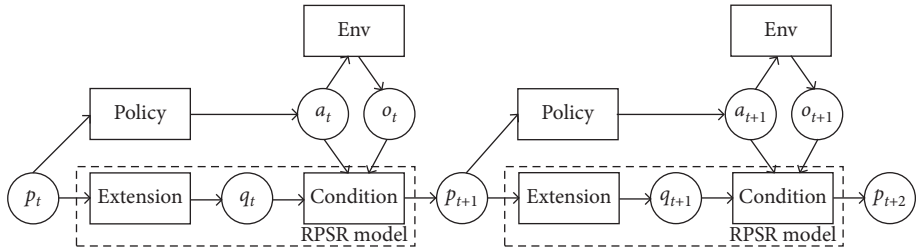


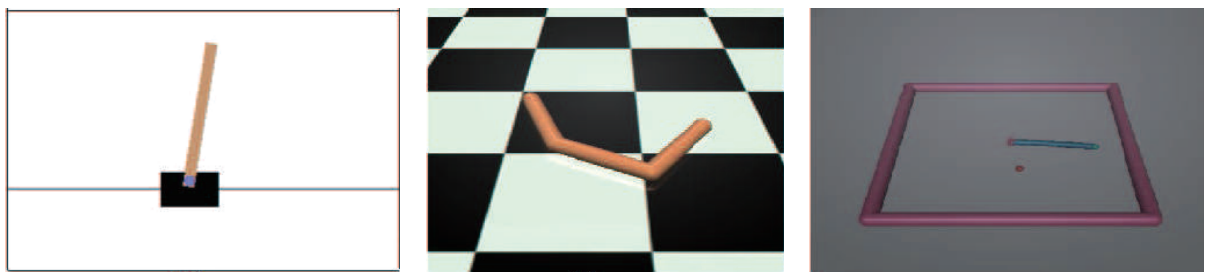
Figure 1: The architecture of the RPSR-DQN network.

Input: learning rate $\eta = \{\eta_{\text{PSR}}, \eta_{\text{policy}}\}$

Output: network parameters $\theta = \{\theta_{\text{RPSR}}, \theta_{\text{policy}}\}$

(1) Randomly select actions to generate N trajectories $\{o_0^i, a_0^i, o_1^i, a_1^i, \ldots, o_m^i, a_m^i\}_{i=1}^N$
(2) Compute the sufficient features of every trajectory $\varphi_{n,t}^h$, $\varphi_{n,t}^O$, $\varphi_{n,t}^A$, $\zeta_{n,t}^O$, $\zeta_{n,t}^A$ (n denotes the n^{th} trajectory)
(3) Establish the recurrent predictive state representation:
(4) Initialize PSR: two-stage regression
(5) Use kernel Bayes rule to estimate $p_{n,t}, q_{n,t}$
(6) Apply least squares method to formula (2) to compute W_{ext}
(7) Set p_0 to the average of $p_{n,t}$
(8) Local optimization:
(9) **for** $i = 1, N$ **do**
(10) Initialize state p_0^i
(11) **for** $t = 1, m$ **do**
(12) Predictive observation $\widehat{o_t^i} = W_{\text{pred}}(p_t, a_t^i)$
(13) Perform a gradient descent step on $(o_t^i - o_t^i)^2$
(14) Update state $p_{t+1} = f_{\text{RPSR}}(p_t, o_t^i, a_t^i, \theta_{\text{RPSR}})$
(15) **end for**
(16) **end for**
(17) Optimization policy network:
(18) Initialize reactive policy θ_{policy}^0 randomly
(19) **for** episode $= 1, M$ **do**
(20) Initialize state p_o
(21) **for** $t = 1, T$ **do**
(22) With probability $1 - \varepsilon$, select $a_t = \max_a Q(p_t, a; \theta_{\text{policy}})$, otherwise random
(23) Execute action a_t in emulator and observe reward r_t and observation o_{t+1}
(24) Set filter $p_{t+1} = f_{\text{RPSR}}(p_t, o_t, a_t)$
(25) Store transition (p_t, r_t, a_t, p_{t+1}) in D
(26) Sample random mini batch of transitions (p_i, r_i, a_i, p_{i+1}) from D
(27) Set $R_i = \begin{cases} r_i & \text{for terminal } p_{i+1} \\ r_i + \gamma \max_{a'} Q(p_{i+1}, a'; \theta_{\text{policy}}^-) & \text{for nonterminal } p_{i+1} \end{cases}$.
(28) Perform a gradient descent step on $(R_i - Q(p_i, a_i; \theta_{\text{policy}}))^2$
(29) **end for**
(30) **end for**

Algorithm 1: RPSR-DQN.



(a) (b) (c)

Figure 2: The control tasks in our experiments: (a) CartPole-v1, (b) Swimmer-v1, and (c) Reacher-v1.

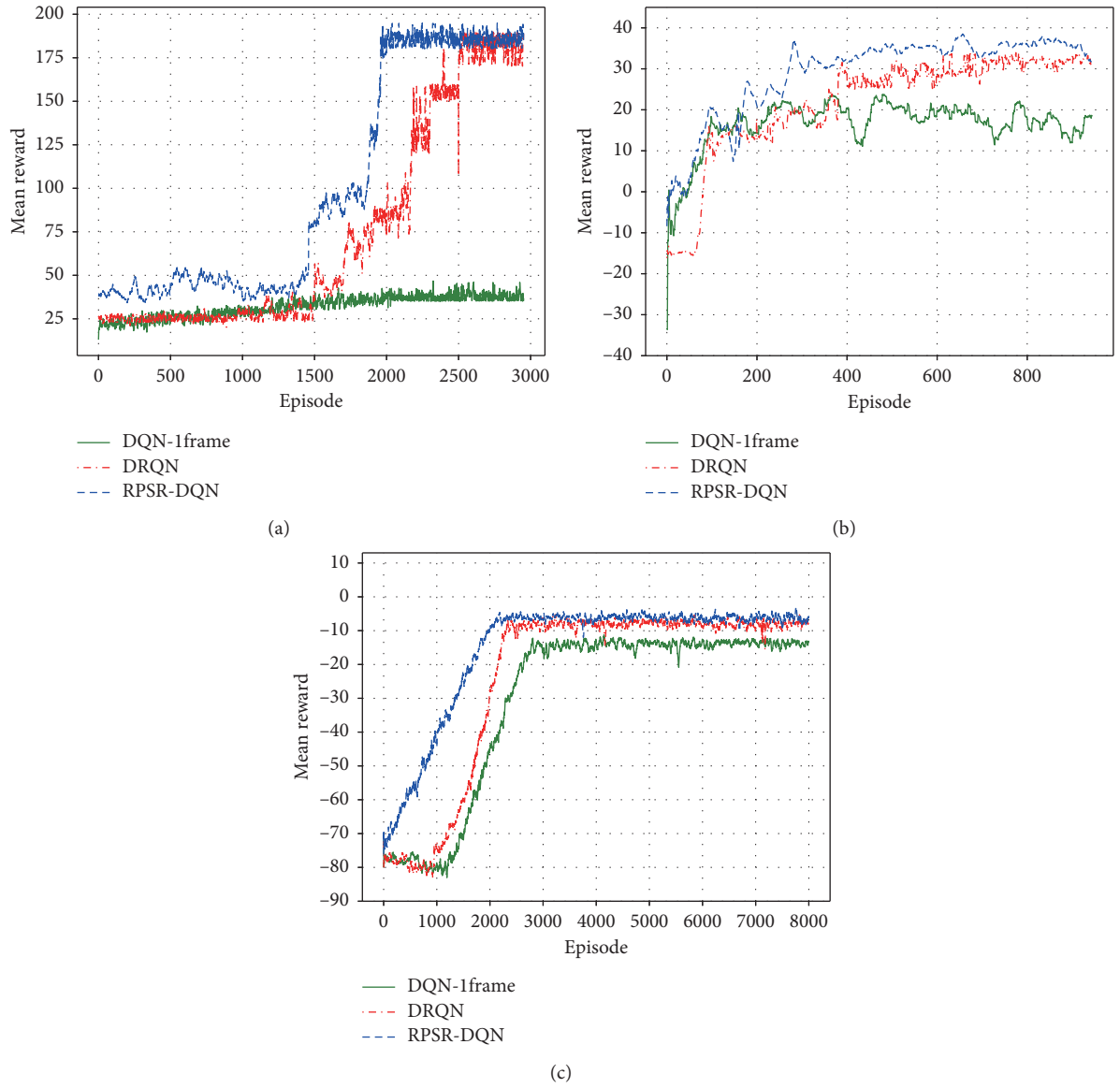


Figure 3: Comparison of RPSR-DQN to model-free methods. Plots show the performance for three methods on all tasks: (a) CartPole-v1, (b) Swimmer-v1, and (c) Reacher-v1.

Table 1: The best reward of three methods.

	CartPole-v1	Swimmer-v1	Reacher-v1
DRQN	200	56	−1.15
DQN-1frame	54	40.58	−6.43
RPSR-DQN	200	59.52	−0.02
RPSP	158	38.96	−57.78

Swimmer-v1: the environment involves a 3-link swimming robot in a viscous fluid, where the goal is to make it swim forward as fast as possible, by actuating the two joints. To make the environment partially observable, we remove the observations that represent the velocity, changing the original five observations to three observations which are the angles of three links. The action in the environment is a vector with three elements which represent the control of three links. Each element is a number in the arithmetic progression with an interval of 0.2 in the range [−1.4, 1.4].

Reacher-v1: this environment involves a 2-link robot arm which is connected to a central point. The goal of this task is to move the endpoint of the robot arm to the target location. Each step reward is the negative of the sum of the distance between the endpoint of the robot arm and the target point

Table 2: The best mean reward of three methods.

	CartPole-v1	Swimmer-v1	Reacher-v1
DRQN	189	30.23	−11.25
DQN-1frame	46.76	23.78	−17.78
RPSR-DQN	194	38.51	−9.24
RPSP	116	21.32	−70.23

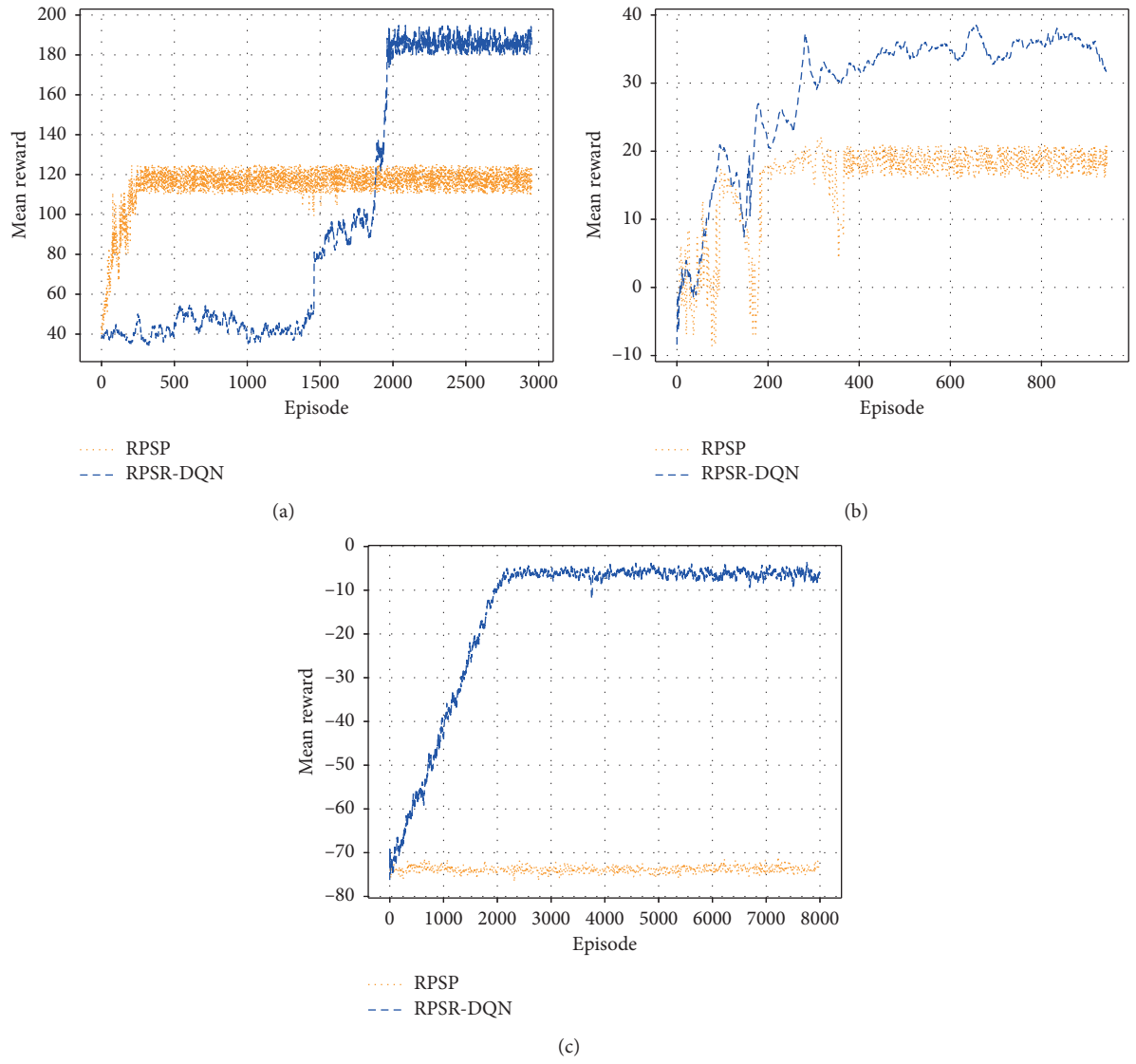


Figure 4: Comparison of RPSR-DQN to the policy-based method. Plots show the performance for three methods on all tasks: (a) CartPole-v1, (b) Swimmer-v1, and (c) Reacher-v1.

and the control cost. The action in this task is a vector with two elements which represent the force applied to the arm. Each element is a number in the arithmetic progression with an interval of 0.2 in the range [−1.4, 1.4]. To make the environment partially observable, we change the original six observable values to four, respectively, which represent the angles of two links and the relative distance between the link and the target position. In this task, it requires to find a balance between exploration and exploitation.

In this section, we compare methods using two metrics: the best reward is the best value for return rewards R_n on all iterations, where R_n is the total return reward for the n^{th} iteration, and the mean reward is the mean return reward $\widehat{R}_n = (1/25)\sum_{i=n-25}^{n} R_i$ for the last 25 iterations.

Comparison to model-free methods: we compared the performance of RPSR-DQN with the model-free methods including the DQN-1frame and DRQN. The result is shown in Figure 3. Compared with the DQN-1frame which selects

the best action only by the current observation, RPSR-DQN can be shown that the predictive state model can achieve the great effect of tracking and updating the state of the environment. Because RPSR-DQN has a model learning process, it learns faster than DRQN and can converge to a more stable state with fewer iterations. And even with sufficient iterations of the update, RPSP-DQN can still get better rewards than DRQN in the final stable situation. The first three rows of Tables 1 and 2 show the numerical result which includes the performance of three methods in all tasks.

Comparison to policy-based methods: Figure 4 shows the results of comparing the RPSR-DQN with the policy-based method RPSR [14]. Note that as a policy-based method, RPSP can be applied to both continuous and discrete environments. In the action discrete environment, our method can get better mean rewards in the final stable situation than the policy-gradient method RPSP. In the Reacher-v1 task, the reasons for the ineffective RPSP may be as follows: the initial random weight tends to output highly positive or negative value outputs, which means that most initial actions make the link have the maximum or minimum acceleration. It causes a problem, which is that this link manipulator cannot stop rotary movement as long as putting the most force in the joint. In this case, once the robot has started training, this meaningless state will cause it to deviate from its current strategy. The RPSR may make not enough exploration to select the action to stop the link manipulator from rotating. The last two rows of Tables 1 and 2 show the numerical result which includes the performance of two methods in all tasks.

6. Conclusion

In this paper, we propose RPSR-DQN, a method that can learn the model and make a decision in partially observable environment. Combining the predictive state model with a value-based approach results in good performance in a partially observable environment. We compare RPSR-DQN with DRQN in different partially observable environments and show that our method can get better performance in terms of learning speed and expected rewards. Also, we compare our approach with the state-of-the-art recurrent predictive state policy (RPSP) networks, where the PSR model and a reactive policy are simultaneously trained in an end-to-end manner. Experiment results show that with the benefits of the DQN framework and the dividing of the learning of the model and the training of the reactive policy, our approach outperforms the state-of-the-art baselines.

Data Availability

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

Conflicts of Interest

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

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 61772438 and 61375077). This work was also supported by the China Scholarship Council (201906315049).

References

[1] D. Silver, A. Huang, C. J. Maddison et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, Deep Variational Reinforcement Learning for Pomdps, 2018.

[3] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[4] C. Huang, Y. An, S. Zhou, Z. Hong, and Y. Liu, "Basis selection in spectral learning of predictive state representations," *Neurocomputing*, vol. 310, pp. 183–189, 2018.

[5] Y. Liu and J. Zheng, "Online learning and planning in partially observable domains without prior knowledge," in *Proceedings of the Generative Modeling and Model-Based Reasoning for Robotics and AI at ICML*, Long Beach, CA, USA, June 2019.

[6] Y. Liu, Z. Yang, and G. Ji, "Solving partially observable problems with inaccurate psr models," *Information Sciences*, vol. 283, pp. 142–152, 2014.

[7] W. Song, G. Xiong, and H. Chen, "Intention-aware autonomous driving decision-making in an uncontrolled intersection," *Mathematical Problems in Engineering*, vol. 2016, Article ID 1025349, 15 pages, 2016.

[8] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "DESPOT: online POMDP planning with regularization," *Advances in Neural Information Processing Systems*, vol. 58, pp. 1772–1780, 2016.

[9] M. J. Hausknecht and P. Stone, Deep recurrent q-learning for partially observable MDPs, Computing Research Repository (CoRR), 06527, 2015.

[10] A. E. L. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[11] W. Hamilton, M. M. Fard, and J. Pineau, "Efficient learning and planning with compressed predictive states," *Journal of Machine Learning Research*, vol. 15, pp. 3395–3439, 2014.

[12] M. L. Littman, R. S. Sutton, and S. Singh, "Predictive representations of state," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 1555–1561, Vancouver, Canada, January 2001.

[13] Y. Liu, H. Zhu, Y. Zeng, and Z. Dai, "Learning predictive state representations via monte-carlo tree search," in *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, New York, NY, USA, July 2016.

[14] Ahmed H., Z. Marinho, Sun W., S. Srinivasa, and G. Gordon, Recurrent predictive state policy networks, 2018.

[15] A. Venkatraman, N. Rhinehart, W. Sun et al., "Andrew Bagnell. Predictive-state decoders: encoding the future into recurrent networks," in *Proceedings of the Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA, December 2017.

[16] A. R. Cass, L. P. Kaelbling, M. L. Littman, A. R. Cassandra, and L. Littman, "Acting optimally in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1995.

[17] J. Ou, Y. Chen, F. Zhao, J. Liu, and S. Xiao, "Novel approach for the recognition and prediction of multi-function radar

behaviours based on predictive state representations," *Sensors*, vol. 17, no. 3, p. 632, 2017.

[18] S. Ragi, C. S. Tan, and E. K. P. Chong, "Guidance of autonomous amphibious vehicles for flood rescue support," *Mathematical Problems in Engineering*, vol. 2013, no. 1, 9 pages, 2013.

[19] B. Boots, S. Siddiqi, and G. Gordon, "Closing the learning-planning loop with predictive state representations," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 954–966, 2009.

[20] J. Pineau, G. J. Gordon, and S. Thrun, "Point-based value iteration: an anytime algorithm for pomdps," in *Proceedings of the IJCAI-03, Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August 2003.

[21] Y. Liu and J. Zheng, Combining Offline Models and Online Monte-Carlo Tree Search for Planning from Scratch, 2019.

[22] X. Li, L. Li, J. Gao, X. He, and J. Chen, "Recurrent reinforcement learning: a hybrid approach," 2015, https://arxiv.org/abs/1509.03044.

[23] W. L. Hamilton, M. M. Fard, and J. Pineau, "Modelling sparse dynamical systems with compressed predictive state representations," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pp. 178–186, Atlanta, GA, USA, June 2013.

[24] S. Singh, M. R. James, and M. R. Rudary, "Predictive state representations: a new theory for modeling dynamical systems," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pp. 512–519, Banff, Canada, July 2004.

[25] Y. Liu, Y. Tang, and Y. Zeng, "Predictive state representations with state space partitioning," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1259–1266, Istanbul, Turkey, May 2015.

[26] M. Rosencrantz, G. J. Gordon, and S. Thrun, "Learning low dimensional predictive representations," in *Proceedings of the Twenty-first international conference on Machine learning—ICML '04*, pp. 695–702, New York, NY, USA, 2004.

[27] H. Ahmed, C. Downey, and G. Gordon, "An efficient, expressive and local minima-free method for learning controlled dynamical systems," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, LA, USA, February 2018.

[28] A. Hefny, C. Downey, and G. J. Gordon, "Supervised learning for dynamical system learning," *Advances in Neural Information Processing Systems (NIPS)*, vol. 28, no. 9, p. 1954, 2015.

[29] W. Sun, A. Venkatraman, B. Boots, and J. Andrew Bagnell, "Learning to filter with predictive state inference machines," *Microelectronics Reliability*, vol. 61, pp. 35–42, 2015.

[30] K. Fukumizu, A. G. LeSong, and I. Steinwart, "Kernel bayes rule: bayesian inference with positive definite kernels," *Journal of Machine Learning Research*, vol. 14, pp. 3753–3783, 2014.

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

[32] V. Mnih, K. Kavukcuoglu, and D. Silver, Playing Atari with Deep Reinforcement Learnin, 2013.

[33] R. Ali and B. Recht, "Random features for large-scale kernel machines," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 1177–1184, Vancouver, Canada, December 2008.

[34] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: stochastic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[35] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.