Multiple Target Prediction for Deep Reinforcement Learning

Jen-Tzung Chien and Po-Yen Hung
Department of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan

***Abstract*—This paper presents the multiple target prediction for model-free deep reinforcement learning. Traditionally, the learning algorithm based on deep Q network (DQN) suffers from slow convergence in learning process which usually constrains the system performance. To speed up the learning process, this study incorporates the *prediction network* and the *auxiliary replay memory* in DQN which allows us to predict multiple target values over different actions rather than relying on a single target value from an action. The prediction network is trained in a sequential manner by using the samples from replay memory while the auxiliary replay memory is introduced to store the states and the predicted targets which are related to individual possible actions that are taken. With these two additional components functioned in DQN algorithm, the resulting approach can efficiently predict Q values and rewards to train an agent with comprehensive target values. The experiments on different tasks demonstrate the merit of multiple target prediction in reinforcement learning.**

***Index Terms*—machine learning, reinforcement learning, deep learning, deep Q network**

I. Introduction

Deep reinforcement learning (DRL) [1] has been rapidly developing in the era of artificial intelligence. DRL deals with complex sequential decision problems such as computer games [2], [3], [4], automatic control [5], [6], [7] and natural language processing [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] as well as many other tasks including hyperparameter tuning and architecture selection for optimization of deep neural network (DNN) [19], [20], [21], [22]. The standard DRL based on deep Q network (DQN) [1], [23] is trained as a regression-based DNN which uses the states as inputs to implement a specialized Q learning. This learning predicts the Q values of different actions from which the agent can choose as the output. To learn a desirable policy, the agent needs to interact with the environment thousands to millions of times which result in a barrier for system development with very limited computation power. The issue of slow convergence in reinforcement learning is basically because of inefficient exploration as well as sparse reward in learning procedure. For model-free DRL, the regression-based DNN in implementation of DQN is only updated by the actual actions the agent had taken. The agent can only update the Q value depends on the action it had taken because it doesn't know the target values of other actions. The weaknesses in training are accordingly caused by its limited ability to predict target values over different actions. If the agent can accurately predict what may happen next as well as human learners, it can learn in a much more efficient way. This paper presents multiple target prediction to tackle the weaknesses in training procedure of DQN. Our idea is to develop a general component which brings prediction to DQN-based models. In addition to traditional components, we introduce the complementary processing units in DQN which consists of a prediction network and an auxiliary replay memory. Prediction network is used to predict all Q values of a state after taking each individual action. With the auxiliary replay memory, we perform multiple target updating for DQN at each state, which improves the training efficiency. This method is evaluated by the experiments on computer games and language understanding.

II. Deep Reinforcement Learning

The issues of inefficient exploration, sparse reward and weak prediction in deep reinforcement learning have been investigated in the literature. Regarding the issue of exploration, a bootstrapped method [24] was proposed to rapid the ϵ-greedy exploration in traditional DQN via a deep exploration using DNN where the randomized value functions were implemented. In [25], a noise term was added in parameter space to allow the agent with fast exploration. To deal with the issue of sparse reward, the idea of intrinsic reward [26], [27], [28], [29], [30], automatically generated by the agent, was proposed to perform self learning. In [31], the agent performed the sample-efficient learning based on hindsight experience replay. How the actions were right or wrong could be learned from sparse reward. On the other hand, the issue of insufficient prediction was handled by learning a model with domain knowledge of control system and simulating the fictional experience from such a model [32]. Both fictional and real experiences were learned accordingly. In [33], a neural network in a form of autoencoder was constructed in Atari games to predict the states in next image frames to compensate the lack of prediction. This paper presents a new framework which deals with insufficient prediction in deep reinforcement learning by predicting the values rather than the states. This framework facilitates rapid computation and learning in training procedure, particularly in early training steps, through multiple target prediction based on prediction network and auxiliary replay memory.

III. Multiple Target Prediction

Figure 1 displays the whole procedure of an extended DQN which carries out multiple target prediction for reinforcement learning. The standard DQN is performed according to the lower half part with blue shading where the Q network receives

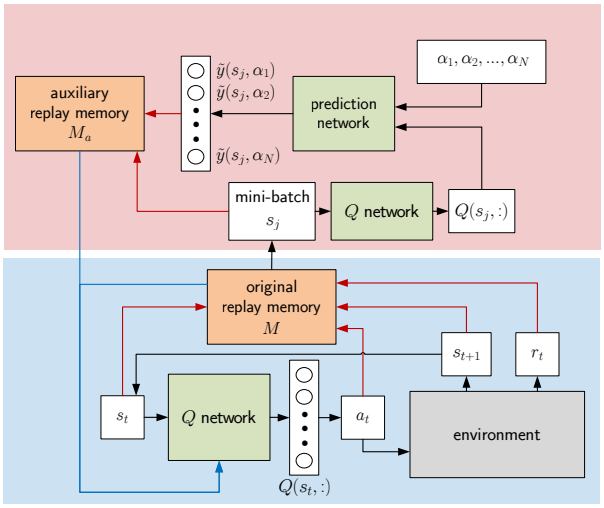


Fig. 1. An overview for implementation procedure of multiple target prediction in deep Q network.

a state s_t from environment at time t as input and produces an action a_t based on a set of value functions $Q(s_t,:)$ of s_t over different actions $a_t \in \{\alpha_n\}_{n=1}^N$. Replay memory M stores the information of consecutive states and current action and reward $\{s_t, s_{t+1}, a_t, r_t\}$. This primary information is merged with an auxiliary information from the upper half of the system with red shading. The fusion of information is then used to conduct multiple target prediction and calculate the value functions $Q(s_t,:)$ via Q network in RL agent when choosing an action a_t. The auxiliary information is obtained by a prediction network and an auxiliary replay memory. Prediction network is introduced to address the prediction issue while auxiliary replay memory M_a is used to update the value network in RL agent with a whole vector of target values instead of a single target value in traditional DQN.

A. Prediction network

To evaluate what may happen next in an environment, a kind of autoencoder was successfully developed to predict next image frames after taking an action for Atari games [33]. However, predicting the next frames is computationally difficult. And sometimes the information in a frame is too little to make a decision by the agent. Therefore, instead of using the frame data, this study mitigate the computation barrier by presenting a prediction network to predict Q values from state s_j to new state s_{j+1}. Here, j means the mini-batch index when sampling from the original replay memory M which is created by basic RL agent. We use $Q(s_j,:)$ of Q network of state s_j and action as the inputs and $\widehat{Q}(s_{j+1},:)$ of target network of state s_{j+1} and the reward r_j as the supervised outputs for training the prediction network. After the training, this prediction network can predict the values $\widetilde{Q}(s_{j+1},:)$ corresponding to different actions for state s_{j+1} in prediction phase. We iteratively feed N different actions $\{\alpha_n\}_{n=1}^N$ with the Q values of state s_j sampled from M into the prediction network to obtain the outputs $\{\widetilde{Q}(s_{j+1},:), \widetilde{r}_j\}$. Target network and prediction network have the target values

$$y(s_j) = r_j + \gamma \max_a \widehat{Q}(s_j, a) \tag{1}$$

$$\widetilde{y}(s_j, \alpha_n) = \widetilde{r}_j + \gamma \widetilde{Q}(s_{j+1}, \alpha_n) \tag{2}$$

respectively, where $\forall n = 1, \ldots, N$ and the discount factor is given by $\gamma = 0.99$. Basically, prediction network is seen as a kind of generator which generates future Q values to allow RL agent to improve the training efficiency.

B. Auxiliary replay memory

Auxiliary replay memory M_a is used to store the consecutive states $\{s_j, s_{j+1}\}$ and the predicted multiple target values $\widetilde{y}(s_j,:)$ via prediction network in a mini-batch j. This memory M_a is separate from the original memory M. With the auxiliary replay memory M_a, RL agent samples a set of data pairs $\{s_j, s_{j+1}, \widetilde{y}(s_j,:)\}$ in different mini-batches s_j from M_a to learn the parameters of Q network where multiple target values $\widetilde{y}(s_j,:)$ are used rather than using a single target value $\widehat{y}(s_j)$ in traditional DQN. This paper integrates the prediction network and the auxiliary replay memory to implement multiple target prediction in DQN (also named as the MTP-DQN). The samples from auxiliary memory M_a are used to learn Q network while those from original memory M are used to train prediction network as detailed below.

IV. Training Procedure

A. Training of deep Q network

First of all, the agent interacts with the environment to acquire data. At each time step t, the agent receives a state s_t from the environment. This state s_t is fed into Q network to obtain Q value vector $Q(s_t,:)$. Given this Q value vector, an action is selected by $a_t = \arg\max_a Q(s_t, a)$. After executing an action, the next state s_{t+1} is received with a reward r_t from the environment. RL agent then saves the transition information $\{s_t, s_{t+1}, a_t, r_t\}$ in replay memory M. At each training step, the agent randomly samples the transitions $\{s_j, s_{j+1}, a_j, r_j\}$ from replay memory M. These samples are time independent, so the samples are indexed by j instead of t. The agent feeds the target network with the state s_{j+1} to obtain a Q value vector $\widehat{Q}(s_{j+1},:)$. The target value $y(s_{j+1})$ is then calculated according to Eq. (1). Traditional Q network is trained by minimizing the regression loss $\mathcal{L}_q$ given by

$$\mathcal{L}_q = \sum_{s_j} \left(y(s_j) - Q(s_j, a_j)\right)^2. \tag{3}$$

B. Training of prediction network

In [33], an autoencoder was proposed to predict the states or image frames for deep reinforcement learning. Basically, predicting the states accurately is not as useful as predicting the Q values and reward accurately. Accordingly, we use Q value vector and action as inputs to the prediction network, and the Q value of next state and the reward as outputs. In

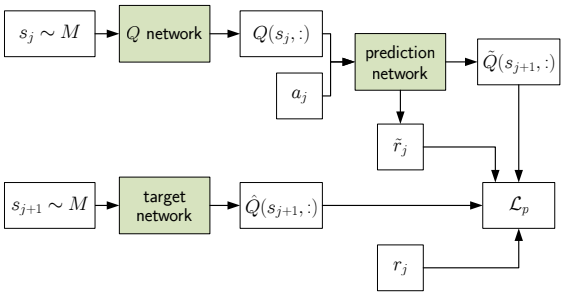


Fig. 2. Calculation of learning objective $\mathcal{L}_p$ for prediction network.

order to have an accurate predicting result, we need to train the prediction network separately. Figure 2 depicts the calculation of learning objective for prediction network $\mathcal{L}_p$.

The agent randomly samples a mini-batch of transitions $\{s_j, s_{j+1}, a_j, r_j\}$ from replay memory M. State s_j is then fed into Q network to receive Q value vector $Q(s_j,:)$ while the next state s_{j+1} is fed into target network to obtain the target Q value vector $\widehat{Q}(s_{j+1},:)$. Prediction network uses the Q values of s_j and the action a_j as the inputs, and the corresponding Q value vector $\widehat{Q}(s_{j+1},:)$ and reward r_j as the supervised targets to train the prediction network according to loss function

$$\mathcal{L}_p = \sum_{s_j} \Bigg[\sum_{n=1}^{N} \Big(\widehat{Q}(s_{j+1}, \alpha_n) - \widetilde{Q}(s_{j+1}, \alpha_n) \Big)^2 + (r_j - \widetilde{r}_j)^2 \Bigg] \tag{4}$$

where $\widetilde{Q}(s_j, \alpha_n)$ is the predicted Q value from prediction network and $\widetilde{r}_j$ is the predicted reward.

C. Construction of auxiliary replay memory

In order to perform mini-batch training for Q network using the predicted data, we need to construct an auxiliary replay memory M_a to store these predicted samples. Figure 3 shows the procedure of constructing auxiliary replay memory consisting of data pairs of states s_j and the corresponding target values $\widetilde{Q}(s_j,:)$. First, the agent randomly samples s_j from replay memory M and obtains Q value vector $Q(s_j,:)$ from Q network. Then, this agent iteratively introduces N different actions $a \in \{\alpha_n\}_{n=1}^N$ and s_j sampled from M into the prediction network to obtain the predicted Q value $\widetilde{Q}(s_{j+1}, \alpha_n)$ and calculate the target values $\widetilde{y}(s_j, \alpha_n)$ according to Eq. (2). Then, the agent combines all predicted target values into a predicted target vector $\widetilde{y}(s_j,:)$. Data pairs $\{s_j, \widetilde{Q}(s_{j+1},:)\}$ or equivalently $\{s_j, \widetilde{y}(s_j,:)\}$ are stored in auxiliary replay memory M_a. Importantly, these data pairs figure out the mapping between a state s_j and its corresponding Q values or target values under different actions.

D. Training with auxiliary replay memory

Once the auxiliary replay memory M_a is constructed, Q network is trained by using this memory. To do so, we randomly sample a mini-batch of $\{s_j, \widetilde{y}(s_j,:)\}$ from M_a

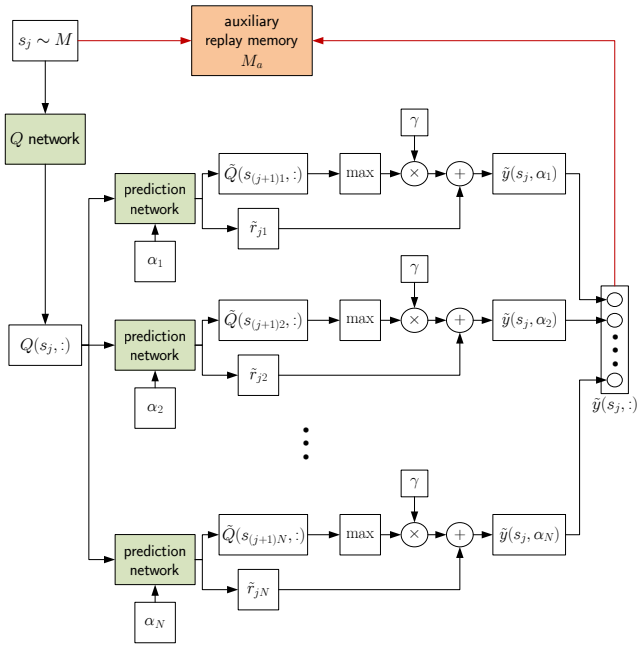


Fig. 3. Construction of auxiliary replay memory M_a.

where s_j is fed into Q network to obtain $Q(s_j,:)$. Training Q network is performed by minimizing the regression loss $\mathcal{L}_{q_m}$ in a form of

$$\mathcal{L}_{q_m} = \sum_{s_j} \sum_{n=1}^{N} \left(\widetilde{y}(s_j, \alpha_n) - Q(s_j, \alpha_n) \right)^2 \tag{5}$$

where the subscript q_m in $\mathcal{L}_{q_m}$ reflects the loss of Q network due to multiple target prediction which is different from that in Eq. (3) due to single target prediction. The parameters of MTP-DQN for Q network $Q(\cdot)$, target network $\widehat{Q}(\cdot)$, prediction network $\widetilde{Q}(\cdot)$ are jointly trained with replay memory M and auxiliary replay memory M_a by minimizing three loss functions $\mathcal{L}_q$, $\mathcal{L}_p$ and $\mathcal{L}_{q_m}$.

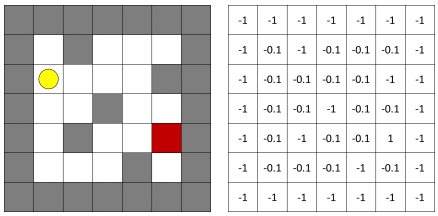
E. Single target versus multiple targets

The main difference between single target prediction in DQN and multiple target prediction in the proposed MTP-DQN is the loss function for updating Q network. DQN uses Eq. (1) as loss function where only the chosen action is employed in updating the Q network. MTP-DQN adopts the loss function Eq. (5) where all possible actions are considered in updating Q network in soft manner. Such a difference plays a key role to improve learning efficiency. Importantly, the original replay memory M with $\{s_t, s_{t+1}, a_t, r_t\}$ is used to train prediction network while the auxiliary replay memory M_a with $\{s_j, \widetilde{y}(s_j,:)\}$ is used to train the new Q network. Since the prediction network is trained in a supervised way with $\{Q(s_j,:), a_j\}$ as inputs and $\{\widehat{Q}(s_{j+1},:), r_j\}$ as targets, after a while, the distribution of the predicted values $\{\widetilde{Q}(s_{j+1},:), \widetilde{r}_j\}$ will be close to that of $\{\widehat{Q}(s_{j+1},:), r_j\}$. Owing to the storage

of predicted values in auxiliary memory M_a, the distribution of data sampled from M_a at training steps is supposed to be close to that of the data sampled from M. Training with M_a is comparable with that with M. In case of single target prediction, the single target updating is regarded as a vector updating with only the chosen action a_j set to the calculation of target value y. Every updating only changes a single entry of the vector. In case of multiple target prediction, updating a whole vector of target values is regarded as updating a mini-batch of single target values.

V. Experiments

This paper presented a new DQN for deep reinforcement learning which was evaluated by three tasks. The first task was a simple grid world with known Q values of every states and actions. The second task was the Atari 2600 games. The third task was a language understanding task with the states and actions all described in words and sentences. In the evaluation, the episode-reward plots were shown to see the performance of training efficiency by using different methods. In the implementation, at each episode, we acquired data samples $\{s_t, s_{t+1}, a_t, r_t\}$ from environment in different time steps and stored them in M. ϵ-greedy algorithm was applied. In this sequential process, the prediction network was run every 10 time steps. Then, the standard DQN was executed every 8 steps and the construction of auxiliary replay memory was performed every 20000 steps. Next, Q network was trained every 8 steps and the target network was updated every 10000 steps. The DNN parameters in different networks based on loss functions in Eqs. (3)-(5) were updated by stochastic gradient descent (SGD) algorithm. In practice, we performed single target updating and multiple target updating alternatively to tradeoff between stability and capability using MTP-DQN.



-1	-1	-1	-1	-1	-1	-1
-1	-0.1	-1	-0.1	-0.1	-0.1	-1
-1	-0.1	-0.1	-0.1	-0.1	1	-1
-1	-0.1	-0.1	-1	-0.1	-0.1	-1
-1	-0.1	-1	-0.1	-0.1	1	-1
-1	-0.1	-0.1	-0.1	-1	-0.1	-1
-1	-1	-1	-1	-1	-1	-1

Fig. 4. The environment (left) and the reward map (right) of a grid world.

A. Grid world

Grid world was a simple toy world built in a matrix. Each element in the matrix was a position. The agent received its current position, given by two one-hot vectors $\mathbf{x}$ and $\mathbf{y}$ for two axes, as its state. There were four possible actions: up, down, left and right to move to the nearby positions. The goal of the environment was to find a shortest path to the goal. The environment is illustrated in Figure 4 where yellow circle denotes the current position of an agent, gray blocks denote the obstacles and red block denotes the goal. The reward map of this environment is also shown. We construct the environment of this toy task where the ground-truth Q values of different states can be calculated to eliminate any uncertainty caused by Q values which are calculated by the target network. The DQNs with single target prediction (DQN) and multiple target prediction (MTP-DQN) are compared. The Q network architecture used in this task was a simple architecture with two fully-connected hidden layers consisting of 256 and 128 hidden units. Both layers adopted the ReLU activation function. The optimizer was based on back propagation.

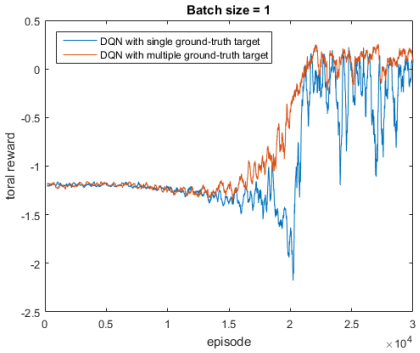


Fig. 5. Learning curves of total rewards in grid world.

The comparison of learning curves using single target updating and multiple target updating is shown in Figure 5. This is an environment with action space of 4. Even if single target updating is less efficient, with the greedy algorithm, the agent can easily choose the right path. Therefore, we simply set the batch size to 1. The experiment result shows that multiple target updating works much better than single target updating. Updating with multiple target is beneficial for efficient training in reinforcement learning.

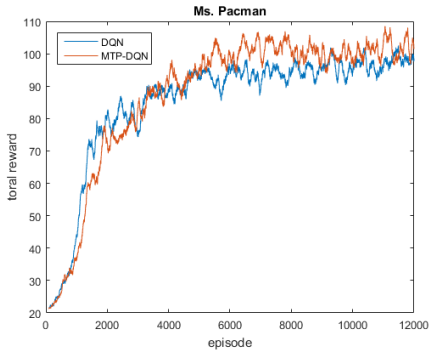


Fig. 6. Learning curves of total rewards in Atari 2600 games.

B. Atari 2600 games

This study further used the OpenAI Gym [34] to implement the Atari 2600 Games which is a benchmark task in evaluation of reinforcement learning. The network architecture was the same as that in original DQN provided in [1], [35]. The prediction network architecture is same as the one used for grid world. In particular, the game on Ms. Pacman was examined.

The learning curves comparing DQN and MTP-DQN are illustrated in Figure 6. The experimental result shows that MTP-DQN performs better than DQN in this comparison. Using MTP-DQN, the possibility that inaccurate prediction can cause a failure is low because the ghosts are moving slower and they are often blocked behind a wall.

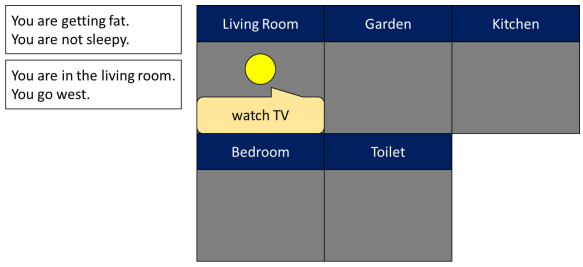


Fig. 7. Illustration of natural language-based home world

C. Natural language-based home world

This language understanding task is a kind of text-based game. Using deep reinforcement learning on such an environment first appears in [8]. In this text-based game, all interactions in the virtual world were through text. The underlying state could not be observed. We constructed the home world with five different rooms. The agent was given a goal which described what it should do in this episode. The goal of the agent was to find the right room and do the right thing. Basically, the agent received 4 sentences as its state shown on the left side of Figure 7. In an initial state, the goal and the starting place in the sentences were randomly chosen. The action of the agent contained a verb chosen from {go, eat, sleep, watch, do, take} and a noun chosen from {north, south, east, west, something, bath, TV, exercise, here}. An illustration of the environment is shown in the figure, notice that the agent can only see the sentences. The agent could only receive positive reward by choosing the right pair of action in the right place with the right goal (ex: action:{take, bath}, place: toilet, goal: dirty). It would not receive any positive reward in other conditions even if the agent was heading for the right room. The reward in such an environment was so sparse that the agent could hardly learn what to do. In the implementation, the input sentences were encoded into a matrix. The agent received 4 sentences with a maximum length of 5 words. After adding the beginning and the ending characters, the maximum length was 7. The input state had a vocabulary size of a 7×4 matrix. We used a long short-term memory (LSTM) [36], [37] layer to represent natural language. The output units of LSTM layers was 256. After the LSTM layer, we used an average-pooling layer and two dense layers both with 256 hidden units. Because the environment received two words as an action, the network outputs should be separate. After the dense layers, one hidden layer with 128 units at both heads was used.

In general, this is a multi-goal task where the goals in individual episode are different. This condition motivates us to incorporate the hindsight experience replay (HER) [31] in DQNs where the first and the fourth sentences were used as the current goal, and the second and the third sentences were treated as the current state. HER could provide additional information to the agent. The environment was built by providing agent information about if the agent's current action would complete some other goals or not. Accordingly, the network architecture was constructed with two inputs: state and goal. These two inputs went through their individual LSTM and average-pooling layer. After these layers, two input paths were concatenated. Basically, the goal remained the same during a whole episode. Because of this multi-output environment, we used two individual networks to predict two Q values of two words with two shared layers. All of the hidden layers had 128 hidden units and hyperbolic tangent as the activation function.

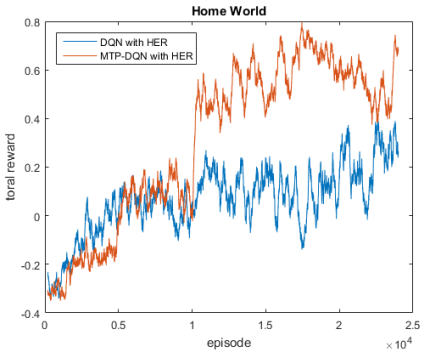


Fig. 8. Learning curves of total reward in home world

The experimental result on natural language-based home world is shown in Figure 8 where HER is applied in both DQN and MTP-DQN. The learning curves of total reward clearly show that DQN is significantly improved by applying the proposed multiple target prediction. The learning curve climbs steeply with big jumps at about episodes 5000 and 10000. These two jumps might happen when the agent creates a new auxiliary replay memory. The Q values generated by prediction network in this natural language environment work well. This result once again demonstrates the benefit of multiple target prediction in reinforcement learning.

VI. Conclusions

This paper presented a new multiple target updating approach to deep reinforcement learning where the learning efficiency using deep Q network was improved. The prediction network and the auxiliary replay memory were proposed to fulfill multiple target prediction based on DQN. The experiments on three tasks showed that the incorporation of these two additional components in DQN consistently improved the conventional method based on single target predicting. The proposed MTP-DQN will be further investigated in different perspectives. First, combining the prediction network with methods such as Monte Carlo tree search [38] will allow the agent to deal with the sparse reward problem. Second, the deep deterministic policy gradient [39] algorithm will be

introduced to implement the proposed method for continuous control tasks, e.g. robot moving and autonomous driving. Third, the replay memory in DQN and the auxiliary replay memory in MTP-DQN will be related to the memory networks where attention mechanism could be applied. Fourth, other RL domains and tasks will be examined.

REFERENCES

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] Guillaume Lample and Devendra Singh Chaplot, "Playing FPS games with deep reinforcement learning," in *Proc. of AAAI Conference on Artificial Intelligence*, 2017, pp. 2140–2146.

[4] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee, "Control of memory, active perception, and action in minecraft," in *Proc. of International Conference on International Conference on Machine Learning*, 2016, pp. 2790–2799.

[5] Shakir Mohamed and Danilo Jimenez Rezende, "Variational information maximisation for intrinsically motivated reinforcement learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2125–2133.

[6] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *arXiv preprint arXiv:1511.03791*, 2015.

[7] Huan-Hsin Tseng, Yi Luo, Sunan Cui, Jen-Tzung Chien, Randall K. Ten Haken, and Issam El Naqa, "Deep reinforcement learning for automated radiation adaptation in lung cancer," *Medical Physics*, vol. 44, no. 12, pp. 6690–6705, 2017.

[8] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay, "Language understanding for text-based games using deep reinforcement learning," in *Proc. of Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1–11.

[9] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao, "Deep reinforcement learning for dialogue generation," in *Proc. of Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1192–1202.

[10] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio, "An actor-critic algorithm for sequence prediction," in *Proc. of International Conference on Learning Representation*, 2017.

[11] Karthik Narasimhan, Adam Yala, and Regina Barzilay, "Improving information extraction by acquiring external evidence with reinforcement learning," in *Proc. of Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 2355–2365.

[12] Seonggi Ryang and Takeshi Abekawa, "Framework of automatic text summarization using reinforcement learning," in *Proc. of Conference on Empirical Methods in Natural Language Processing*, 2012, pp. 256–265.

[13] Cody Rioux, Sadid A Hasan, and Yllias Chali, "Fear the REAPER: A system for automatic multi-document summarization with reinforcement learning.," in *Proc. of Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 681–690.

[14] Ji He, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng, "Deep reinforcement learning with a combinatorial action space for predicting popular reddit threads," in *Proc. of Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1838–1848.

[15] Jen-Tzung Chien and Wei Xiang Lieow, "Meta learning for hyperparameter optimization in dialogue system," in *Proc. of Annual Conference of International Speech Communication Association*, 2019, pp. 839–843.

[16] Jen-Tzung Chien, "Deep Bayesian natural language processing," in *Proc. of Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, 2019, pp. 25–30.

[17] Shinji Watanabe and Jen-Tzung Chien, *Bayesian Speech and Language Processing*, Cambridge University Press, 2015.

[18] Dong Yu, Geoffrey Hinton, Nelson Morgan, Jen-Tzung Chien, and Shigeki Sagayama, "Introduction to the special section on deep learning for speech and language processing," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 4–6, 2011.

[19] Samantha Hansen, "Using deep Q-learning to control optimization hyperparameters," *arXiv preprint arXiv:1602.04062*, 2016.

[20] Natasha Jaques, Shixiang Gu, Richard E Turner, and Douglas Eck, "Tuning recurrent neural networks with reinforcement learning," in *Proc. of ICLR Workshop*, 2017.

[21] Barret Zoph and Quoc V. Le, "Neural architecture search with reinforcement learning," in *Prof. of International Conference on Learning Representation*, 2017.

[22] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. of International Conference on Learning Representation*, 2017.

[23] Zachary Lipton, Xiujun Li, Jianfeng Gao, Lihong Li, Faisal Ahmed, and Li Deng, "BBQ-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems," in *Proc. of AAAI Conference on Artificial Intelligence*, 2018.

[24] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy, "Deep exploration via bootstrapped DQN," in *Advances in Neural Information Processing Systems*, 2016, pp. 4026–4034.

[25] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz, "Parameter space noise for exploration," *arXiv preprint arXiv:1706.01905*, 2017.

[26] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 16–17.

[27] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly, "Episodic curiosity through reachability," in *Proc. of International Conference on Learning Representations*, 2018.

[28] Jen-Tzung Chien and Po-Chien Hsu, "Stochastic curiosity maximizing exploration," in *Proc. of International Joint Conference on Neural Networks*, 2020, pp. 1–8.

[29] Jen-Tzung Chien, Wei-Lin Liao, and Issam El Naqa, "Exploring state transition uncertainty in variational reinforcement learning," in *Proc. of European Signal Processing Conference*, 2020, pp. 1527–1531.

[30] Jen-Tzung Chien and Po-Chien Hsu, "Stochastic curiosity exploration for dialogue systems," in *Proc. of Annual Conference of International Speech Communication Association*, 2020, pp. 3885–3889.

[31] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5055–5065.

[32] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine, "Continuous deep Q-learning with model-based acceleration," in *Proc. of International Conference on Machine Learning*, 2016, pp. 2829–2838.

[33] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh, "Action-conditional video prediction using deep networks in Atari games," in *Advances in Neural Information Processing Systems*, 2015, pp. 2863–2871.

[34] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, "Openai gym," 2016.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[36] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[37] Jen-Tzung Chien and Yuan-Chu Ku, "Bayesian recurrent neural network for language modeling," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 2, pp. 361–374, 2016.

[38] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck, "Monte-Carlo tree search: A new framework for game AI," in *Proc. of Aritificial Intelligence and Interactive Digital Entertainment Conference*, 2008.

[39] 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," *arXiv preprint arXiv:1509.02971*, 2015.