Handling Large-Scale Action Space in Deep Q Network

Zhiheng Zhao, Yi Liang, Xiaoming Jin
School of Software
Tsinghua University
Beijing, China
e-mail: zhiheng3@gmail.com, liang.yi@yahoo.com, xmjin@tsinghua.edu.cn

***Abstract*—Deep reinforcement learning (DRL) is a new topic in recent years. Deep Q Network is a popular DRL implement. It is a well-studied technique and has achieved significant improvement on several challenging tasks such as Atari 2600 games. However, in some kinds of games, there are a large number of possible actions. Thus the output layer in DQN could be complicated due to the large-scale action space, which could harm the performance of DQN. In this paper, we proposed a variant structure of DQN to handle this problem. We could reduce the size of output layer in DQN. The experimental results show that our method improves significantly in some tasks with large-scale action space.**

Keywords-reinforcement learning; intelligent agents; DQN

I. INTRODUCTION

Reinforcement learning (RL) is an important technique in the area of machine learning and artificial intelligence. Most traditional RL methods relied on linear models or shallow neural networks. In recent years, deep learning methods have made significant advances in a lot of areas. Thus a natural idea is to improve RL method with a deep neural network (DNN).

Mnih et al. presented an original structure deep Q-network (DQN) [1] in 2013. They combined the popular RL algorithm Q-learning with a deep convolutional neural network (DCNN). The DQN agent received only the raw pixels of the game as inputs. And it has achieved best performance on the challenging domain of Atari 2600 games [2]. The RL methods with deep models are also known as deep reinforcement learning (DRL).

Recent works have proved the effectiveness of DQN algorithm. However, the scale of the output layer in DQN depends on the possible action space. In the Atari environment, only 18 possible actions could be played during the whole game (idle, 8-direction joystick with a button). Thus there are 18 corresponding nodes in the output layer. But in some other applications like card games, the player could make a decision among a huge set of legal combination of cards. It means there should be corresponding amount of output layer nodes in the classical DQN. And a large-scale of output layer could harm the performance of the neural network because of the sparse training labels. To avoid this problem then handle the games with a large number of possible actions, we proposed a variant structure of DQN in this paper.

States and actions are the two key elements in Q-learning. A state is a representation of game status at a certain time. And an action is the player's choice at the current moment. In a classical DQN, states are treated as inputs while actions are outputs. Straight-forward idea is to move the action nodes to the input layer. Then the network will receive actions as another part of input and output the prediction of result. The prediction could be represented by a scalar. So we could reduce the amount of output layer nodes to only one. However, there are many possible actions at a certain moment. And different choices will lead to different results. Then simply utilizing states and actions as inputs could not achieve the best performance. We will analyze this point in Section 4.

Now consider a certain state in a card game. The state stays until it receives an agents' action. In other words, the next state only depends on the previous state and the action. Thus we could represent the combination of a state-action pair by the future state. Then we take the future state as the input of neural network. The experimental results demonstrated that our method could improve significantly.

II. RELATED WORK

The objective of reinforcement learning is to find a good strategy by maximizing the cumulative future reward from the environment [3], [4]. It was widely used in game theory, control theory, information theory, etc. Sutton presented the first RL algorithm temporal-differences (TD) [5] in 1988. And the best-known success application of this method is TD-gammon [6], a backgammon program. It was learned by self-play and achieved a top level of human player.

Q-learning is another practical RL method [7]. Considering the delayed rewards is the key point of Q-learning. However, the attempts to apply RL method to other popular games were not successful in early years. Searching and knowledge-based methods were much more popular than RL methods in poker games [8]. Dahl even pointed out that value-based reinforcement learning is not applicable to imperfect information games in his work [9].

In 2013, Mnih et al. presented a method combining Q-learning with a deep neural network (DNN) [1]. And the improved version achieved human-level performance in Atari 2600 games [10]. It is a great success of RL method. The results demonstrate that RL method could be effective on real-world problems.

There were several effective improvements of classical Q-learning in the past years. Experience replay is a general optimization method, which could smooth out learning and

978-1-5386-6987-7/18/$31.00 ©2018 IEEE

Authorized licensed use limited to: University of Nantes. Downloaded on October 08,2020 at 07:54:40 UTC from IEEE Xplore. Restrictions apply.

accelerate convergence [11]. And double Q-learning is another practical optimization for Q-learning [12]. It avoids the serious overestimation of action values under certain conditions.

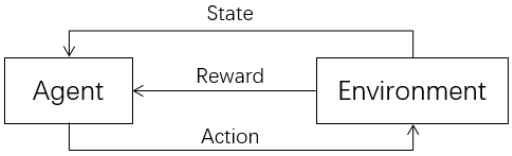

Figure 1. Reinforcement learning.

In recent years, scholars attempted to apply these optimizations to DQN. Schaul et al. proposed a prioritized experience replay method based on classic experience replay [13]. It samples the transitions which need much to learn more frequently. Van Hasselt et al. combined double Q-learning technique with DQN [14]. The method double DQN (DDQN) could reduce harmful overestimations during the training period. Hessel et al. summarized effective improvement of DQN and provided a full comparison between the methods [15]. Most of the optimizations are model-free and easy to implement.

DQN has made a great improvement in Atari 2600 games. And some researchers attempted to apply DQN to a much more difficult problem StarCraft II [16]. However, there is no significant progress at present. The results show that the agent could only learn some simple behavior by self-play. And the performance is much worse than supervised learning method.

III. BACKGROUND

Fig. 1 shows the main idea of reinforcement learning. The agent interactives with the specific environment and receives rewards as a feedback. In each state, the agent could decide to play an action and receive a corresponding reward. If the agent gets a positive reward through an action, it will tend to choose this action in the similar state next time. In this section, we will provide a detailed introduction to Q-learning and deep Q-network.

A. Q-Learning

Q-learning is a popular algorithm in the domain of reinforcement learning. The main idea of Q-learning is to choose the best action in a certain state. For a given environment, assuming that the agent follows a policy π. Then we could define a value function Q_π as follows:

$$Q_\pi(s,a) = \mathbb{E}[r_1 + \gamma r_2 + \cdots | S_0 = s, A_0 = a, \pi] \# \quad (1)$$

Here s is a state, a is an action and r_i is the potential rewards. And $\gamma \in [0,1]$ is a discount factor which makes the immediate reward more important than the future ones. The meaning of Q_π is the expectation of the cumulative reward if the agent plays action a at state s. The objective of Q-learning is to maximize the optimized value function $Q^*(s,a) = max_\pi\, Q_\pi(s,a)$. From the above two equations and Bellman-equation, we could get the following formula

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a') \,\middle|\, s,a\right] \# \quad (2)$$

In this equation s' represents the next state after playing action a at state s. The value of Q^* could be calculated by recursion.

B. Deep Q-Network

If the space of states and actions is small enough, we could simply store the value of every (s,a) pair in a table. But in the real-world problems, the space could be extremely large. Thus we should approximate the value function by some regression methods. Deep Q-network (DQN) deals with the regression problem by a deep neural network. Fig. 2 shows the architecture of a classic DQN. The network receives a representation of a state as input. And for each action a, there is an output node referring to the value of $Q^*(s,a)$. Thus all the values could be calculated by the neural network. It is clear that n output nodes are required for n possible actions. Since the method is model-free, all types of neural networks could be applied. In practice, convolutional neural network (CNN) is the most widely used one because the inputs are images in Atari 2600 games.

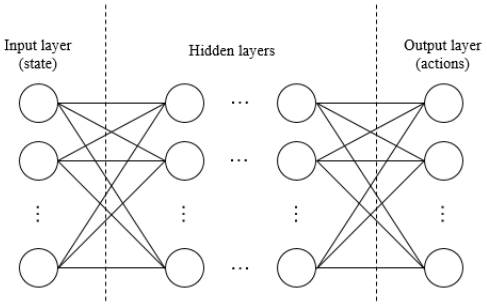

Figure 2. Deep Q-network.

IV. VARIANT DQN

A. Possible Action Space

In Atari 2600 games, there is a joystick and a button in the controller. In a moment, the agent could choose to keep the joystick in the center or push it to 8 directions. It also could press the button or not at the same time. So there are 18 possible actions during the whole process.

But in more general games like card games, there are thousands of possible actions due to the abundant combination of cards. And some actions rarely occur in the game. For example, in many shedding-type card games, a "solo" (single card) is much more common than a long "chain" (several consecutive individual cards). The lack of some classes of training labels would harm the whole performance of the network. In some extreme cases, there could be no examples for a certain action. Then the output of the corresponding node will be totally random. So it is unacceptable if there are thousands of nodes in the output layer.

B. Future State

In a game, the transition of states could be deterministic or probabilistic. It depends on the particular environment.

Authorized licensed use limited to: University of Nantes. Downloaded on October 08,2020 at 07:54:40 UTC from IEEE Xplore. Restrictions apply.

Fig. 3 shows two examples of Atari 2600 games[1]. The left one is a two-player game Football. In this game, the transition of states is probabilistic because the opponent's behavior is variable. In another word, playing action a at state s may reach different future state s'. So we have to represent the states and actions separately in the network.

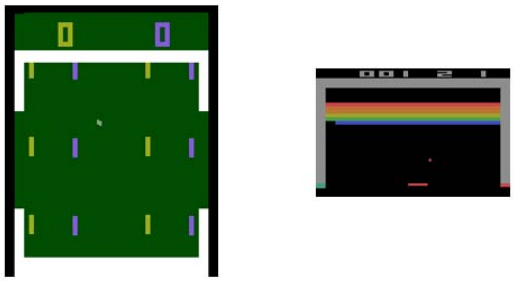

Figure 3. Football and breakout.

But the situation is totally different in another game. The right one in Fig. 3 is a popular single-player game Breakout. Every state in the game could be determined by the previous state and action. It means we could reorganize the structure of neural network without discarding any information. We will call this kind of games deterministic game in the rest of this paper.

C. Variant DQN

As we mentioned in Subsection A, the classic DQN could not handle the game with large-scale action space. So we need to modify the structure of DQN to satisfy the huge action space. Since the neural network could be treated as a regression method, a straight-forward idea is receiving both state and action information as input and simply outputting the estimation value. And we have implemented this approach as a contrast.

However, there is a serious problem with the proposed method. In the variant structure, the network receives states and actions as inputs and outputs a scalar. The scalar represents the prediction of the future rewards. As we mentioned in Section 1, same state and different actions could lead quite different results. It means there are such examples in the training dataset: the inputs are quite similar due to the same state and the outputs are very different. Obviously, the examples are ambiguous for the neural network. The network is difficult to distinguish the differences between the similar inputs. And it will tend to take the intermediate output values of all the similar examples.

To handle this problem, we consider the representations of inputs in deterministic game. Due to the determinacy of transitions, a state-action pair (s, a) corresponds to a deterministic future state s'. It means we could distinguish different state-action pair by future state. Then we could calculate the estimation of state s' for the corresponding state-action pair. Fig. 4 shows the structure of variant DQN. For every (s, a) which needs to calculate, we execute the action at state s in the environment and get the future state s'. Then we take the future state s' as the input of network. Now the network could easily learn the consequences after different actions at the same state.

In our proposed architecture, the network structure is independent of the scale of possible action space. Thus the proposed method could handle the large-scale action space in DQN.

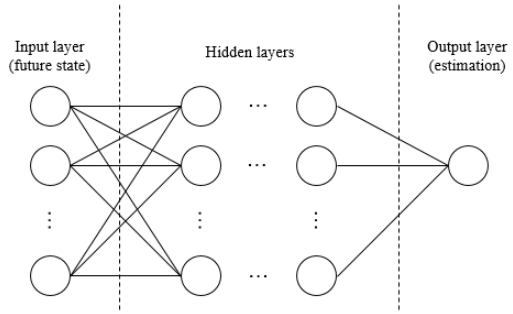

Figure 4. Variant DQN.

V. EXPERIMENTS

We have applied our method to a popular three-player Chinese card game Dou-dizhu. It is played with one pack of cards including jokers. There are three players in a game. One of them plays as the *landlord* and the other two play as the *peasants*. The objective is to be the first team to empty his hand. In the beginning, the landlord is dealt 20 cards while the peasants are dealt 17 cards each. Then they play the cards turn by turn. There are several legal move categories in the game. And the "airplane" is the most complex one. There are thousands of different "airplanes" with the different point of cards. Thus the possible action space is quite huge.

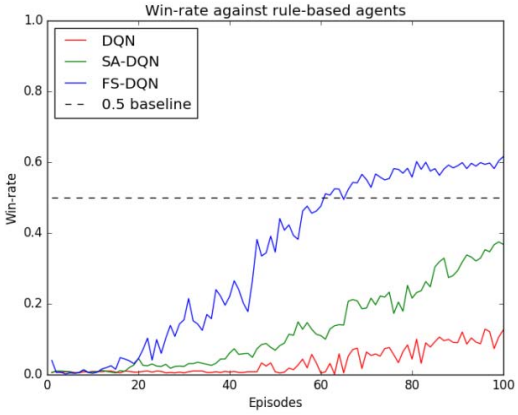

Figure 5. Results.

A. Settings

To make the game deterministic, we have produced two rule-based agents to play as the peasants. For the same state of a game, the rule-based agents will always play the same actions. So it is a deterministic game under these settings. The objective of the DQN agent is to win more games as a landlord.

[1] Picture source: https://en.wikipedia.org/wiki/Atari_2600

Authorized licensed use limited to: University of Nantes. Downloaded on October 08,2020 at 07:54:40 UTC from IEEE Xplore. Restrictions apply.

The inputs of the network are extracted features from information for the game. The information consists of remain number of cards in the hand and play records.

We have trained three different DQNs. The first one is the classic DQN with a large number of output layer nodes. Since some rare "airplane" could hardly appear in the game, we have restricted the maximum length of "airplane" to 3.

The second one is the variant DQN with state and action as input. The actions are described in a one-hot vector. Each dimension represents a corresponding action. Then we combined the two parts of representations as for the inputs.

The last one is the variant DQN with the future state as input. The representations of states are same as the other two methods.

For the convenience, we define the three methods as DQN, SA-DQN and FS-DQN. All the DQNs are trained with the optimization double DQN [14] and prioritized experience replay [13].

B. *Results*

Fig. 5 illustrates the performance of three methods. We evaluated the agents every episode with 2,000 training batches. The results demonstrated that our method FS-DQN achieved the best performance. From the figure we could find that classic DQN have learned a little after 100 episodes. And it shows that oversized output layer could harm the performance of DQN.

Another meaningful result is the comparison between two variant DQNs. The network with future state performed much better as we expected. And the scale of input layer in FS-DQN is also smaller than SA-DQN. It shows that a simpler structure achieved better performance. Thus the effectiveness of our method has been demonstrated.

VI. CONCLUSION

In this paper, we proposed a variant DQN structure to handle the large-scale action space. We explained that the game which has some specific properties could suffer the action space explosion. And in many cases, we could deal with this problem by our proposed method. The experimental results demonstrated that our approach made an improvement on the issue. We will apply our method on more real-world problems in the future work.

ACKNOWLEDGMENT

Partly supported by Key technology R&D Program of Liaoning and Shenyang (2017231005, 17-192-9-00).

REFERENCES

[1] V. Mnih et al., "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An evaluation platform for general agents," J. Artif. Intell. Res.(JAIR), vol. 47, pp. 253-279, 2013.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," Journal of artificial intelligence research, vol. 4, pp. 237-285, 1996.

[4] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction (no. 1). MIT press Cambridge, 1998.

[5] R. S. Sutton, "Learning to predict by the methods of temporal differences," Machine learning, vol. 3, no. 1, pp. 9-44, 1988.

[6] G. Tesauro, "Temporal difference learning and TD-Gammon," Communications of the ACM, vol. 38, no. 3, pp. 58-68, 1995.

[7] C. J. C. H. Watkins, "Learning from delayed rewards," King's College, Cambridge, 1989.

[8] J. Rubin and I. Watson, "Computer poker: A review," Artificial intelligence, vol. 175, no. 5-6, pp. 958-987, 2011.

[9] F. A. Dahl, "A reinforcement learning algorithm applied to simplified two-player Texas Hold'em poker," in European Conference on Machine Learning, 2001, pp. 85-96: Springer.

[10] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, p. 529, 2015.

[11] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," Machine learning, vol. 8, no. 3-4, pp. 293-321, 1992.

[12] H. V. Hasselt, "Double Q-learning," in Advances in Neural Information Processing Systems, 2010, pp. 2613-2621.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," arXiv preprint arXiv:1511.05952, 2015.

[14] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in AAAI, 2016, vol. 16, pp. 2094-2100.

[15] M. Hessel et al., "Rainbow: Combining Improvements in Deep Reinforcement Learning," arXiv preprint arXiv:1710.02298, 2017.

[16] O. Vinyals et al., "StarCraft II: a new challenge for reinforcement learning," arXiv preprint arXiv:1708.04782, 2017

Authorized licensed use limited to: University of Nantes. Downloaded on October 08,2020 at 07:54:40 UTC from IEEE Xplore. Restrictions apply.