Improved Deep Q Network based on Weighted Dueling Architecture

Yuning Wu

FF8B73

ywu190@jh.edu

December 14, 2021

Abstract

In reinforcement learning,agent plays the role of an actual decision-maker of action, to decide which kind of behavior to show at the next state. In most real-life circumstances, an agent is expected to explore the environment without using predictions of the environment response. Based on model-based learning, model free learning has become the more universal method that agent wants. Deep Q-learning is a commonly used model-free algorithm extended by Q-learning which stores the data obtained by exploring the environment in a buffer, and then randomly samples and updates the parameters of the deep neural network to obtain independent training samples.[1] Inspired by this, this project introduces an improved algorithm based on Q-learning, and further extends it to the improvement on Deep Q-learning.

I. Introduction

Current reinforcement learning can be classified into model-based reinforcement learning and model-free reinforcement learning. In model-free reinforcement learning, Q-learning is the most common algorithm. It uses Q-table to construct action-state space and represent the reward function, which can make action selection directly according to the Q value. However, since Q-learning algorithm involves a maximization process, it leads to the overestimation of the Q value[2]. In addition, due to the huge dimension of the state space and action space in practical problems, it is impossible to build a Q-table in computer memory. To solve this problem, Q-table can be turned into function approximation problem and uses function value to represent Q value instead of Q-table. In recent years, a broad trend in deep learning yields the concept of using combination of reinforcement learning and deep learning(Deep Reinforcement Learning, DRL) to train the agent. The Deep Q-Network algorithm proposed by Deepmind has greatly surpassed human level in different fields such as Go and video games. This innovation significantly solves the curse of dimensionality in Q-learning. However, due to estimation bias and the effect of noise, overestimation exists in this algorithm. To resolve this problem, Double Deeq Q-Network is introduced [3], which adapts double estimator to seperate action selection and evaluation; Dueling Network Architecture [4]is proposed to divide the Q-Network into value function and advantage function and improve the algorithm through optimizing network structure. However, introducing a double estimator may lead to underestimation. In this project, we propose a Weighted Dueling Double Deep Q-Network based on weight to adjust Q value, which combines the Dueling Network Architecture and Double Deep Q-network to address the overestimation and underestimation problem.

The report is organized as follows. In the second section, we go through the principle of DQN and its improvements. In the third section, we demonstrate the improvement of introducing weight in Dueling Double Deep

Q-Network (WD3QN). We will also provide mathematical proofs for the proposed weight value. In the fourth section, we empirically analyze the performance of Dueling Double Deep Q-Network(D3QN) and Weighted Dueling Double Deep Q-Network(WD3QN).

II. Background

i. Deep Q-Network

Comparing to Q-learning, the main improvement made by Deep Q-Network is in the following three aspects:

- DQN uses deep convolutional networks (Convolutional Neural Networks, CNN) to approximate the value function
- DQN uses experience replay to train the learning process of reinforcement learning
- DQN independently sets the target network to deal with the deviation in the timing difference separately

In the training process, under the current state s, action a is chosen and we get reward r, the dataset (s, a, r, s') is stored in the replay memory and gradient decent step is performed with respect to the network parameter θ, the network target value should be:

$$y^{DQN} = r + \gamma max Q(s', a', \theta^-) \tag{1}$$

The loss function is:

$$l = (y^{DQN} - Q(s', a', \theta^-))^2 \tag{2}$$

To avoid the impact of the correlation between samples, each time m samples are drawn randomly.

ii. Double Deep Q-Network

In nature DQN, the target Q value is observed by taking the maximum value. It can be seen that if we averaging the Q value after taking the maximum values, the result will be larger than that if we first average the Q values, then to take maximum value, which is called overestimation. Mathmetically, it can be denoted as:

$$E(max(X_1, X_2, ...)) \leq max(E(X_1, E(X_2), ...) \tag{3}$$

If the overestimated Q value of a suboptimal action exceeds the Q value of the optimal action, the optimal action will never be found, which is why it is essential to overestimation So an effective improvement of DQN called Double Deep Q-Network can be introduced, which uses double estimator that separate action selection and evaluation. Using the primary network to select action and evaluate the action with target network, thereby mitigate overestimation. The target Q value is:

$$y^{DDQN} = r + \gamma Q(s', argmax Q(s', a', \theta); \theta^-) \tag{4}$$

iii. Dueling Network Architecture

Dueling Network Architecture effectively divide the Q-network into value function $V(s; \theta, \alpha)$ and advantage function $A(s, a; \theta, \beta)$. The value function part is only related to the current state s and has nothing to do with the selected action a. The advantage function represents the additional value brought by the selection of an action over other actions. The target Q value is:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \alpha) + A(s, a; \theta, \beta) \tag{5}$$

α and β are the unique network parameter of the value function and advantage function respectively, and θ is the public parameter.
Equation(4) the respective impact of the value function and advantage function cannot be identified directly. To enhance identifiability, the method used in practical problems is as follows:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \alpha) + [A(s, a; \theta, \beta) - \frac{1}{A} \sum_{a' \in A} A(s, a'; \theta, \beta)] \tag{6}$$

III. Contribution

A combination of Double Deep Q-Network algorithm and Dueling Architecture can be derived to further improve DQN(D3QN). However, underestimation still exists. Based on this idea, this project extends the Dueling Double Deep Q-Network to applying weight value to adjust Q value and propose a new algorithm named Weighted Dueling Double Deep Q-Network(WD3QN).
As well separate the action selection and evaluation, based on dueling architecture, using the linear combination of $Q(s', a'; \theta, \alpha, \beta)$ and $Q(s', a'; \theta', \alpha, \beta)$ to obtain the target Q value and formulate a trade-off between the overestimation caused by the single estimator and underestimation caused by the double estimator, thereby reduce the bias of the target value estimation.

i. Weighted double estimator

Mathematically, the estimator can be written as follows:

$$\mu(S) = \eta\mu^A_{a^*}(S) + (1-\eta)\mu^B_{a^*}(S) \tag{7}$$

Here η is the weighted value, where $\eta \in [0, 1]$. We need to define the function of η. Based on the definition of the unbiased estimator[7], for variables in a set $X = \{X_1, X_1, ..., X_M\}$, assume $X_i$follows distribution H_i. Since we are investigating the factor with two different distribution, we can use Kullback-Leibler divergence $KL(H_i||H_j)$to denote the similarity of two estimators X_i and X_j. Here, we need to make sure that $maxKL(H_i||H_j)$ is small enough to make $\mu(S)$ the unbiased estimator. We can get that $KL(H_{a_*}||H_{a_L})$ denotes the approximation to $maxKL(H_i||H_j)$, where $a_L \in argmin\mu^A_i(S)$. Since we have no access to the specific distribution of variables in X, we can use the corresponding cumulative expectation of reward $|E\{X_{a_*}\} - E\{X_{a_L}\}|$ to approximate $KL(H_{a_*}||H_{a_L})$. Since $|\mu^A_{a_*}(S) - \mu^B_{a_L}(S)|$ is the unbiased estimator of $|E\{X_{a_*}\} - E\{X_{a_L}\}|$, the weighted value η can be defined as follows:

$$\eta = \frac{|\mu^A_{a_*}(S) - \mu^A_{a_L}(S)|}{c + |\mu^A_{a_*}(S) - \mu^A_{a_L}(S)|} \tag{8}$$

Equation (8) can make η smaller when the distributions of the two variables are getting more similar to each other, which is why the c is added into the equation, where $c \leq 0$.

ii. WD3QN algorithm

The δ is denoted as follows:

$$\delta = |Q(s', a'; \theta, \beta) - Q(s', a''; \theta, \beta)| \tag{9}$$

a' and a'' denotes the actions that evaluate the maximum and minimum Q value of the initial network respectively:

$$\begin{aligned} a' &= argmaxQ(s', a; \theta, \alpha, \beta) \\ a'' &= argminQ(s', a; \theta, \alpha, \beta) \end{aligned} \tag{10}$$

Under the above MDP setting, we can define the final function for η based on equation (8):

$$\eta = \delta/(c + \delta) \tag{11}$$

Here c is a hyperparameter, which is used to calculate the weight η. In problems with different feature, the optimal value of c is not identical. Therefore, the value of c is usually set adaptively according to the feature of the specific problem. The value of c used in the experiments in this project is set to be 10 on the basis of the parameter control experiment.

$$\begin{aligned} y^{WD3QN} = r + \gamma[\eta * Q(s, a'; \theta, \alpha, \beta) + \\ (1-\eta) * Q(s, a'; \theta', \alpha, \beta)] \end{aligned} \tag{12}$$

The algorithm is basically similar with the combination of the Double Deep Q-Network algorithm and the dueling network architecture. The core contribution is made in the part of calculating the target value, which we have added weight value to improve the overestimation and underestimation. The flow of the WD3QN algorithm is shown as follows:

Algorithm 1 Weighted Dueling Double Deep Q-learning(WD3QN)

Initializing
for $episode = 1, max_{episode}$ **do**
  Observe initial state s_0
  for $i = 1, T$ **do**
    Input state $s_i$to the Q network
    Select action a_i according to $\epsilon - greedy$ algorithm
    Observe the next state s_{i+1} and reward r_i
    Store (s_i, a_i, r_i, s_{i+1}) in replay buffer $\mathcal{D}$
    Sample n from the replay buffer $\mathcal{D}$ (s_k, a_k, r_k, s_{k+1})
    $a' = argmax_a Q(s_{k+1}, a; \theta, \alpha, \beta)$
    $a'' = argmin_a Q(s_{k+1}, a'; \theta, \alpha, \beta)$
    $\delta = |Q(s_{k+1}, a'; \theta', \alpha, \beta) - Q(s_{k+1}, a''; \theta', \alpha, \beta), \eta = \delta/(c + \delta)|$
    Compute target Q value:
      $y_k = r_k + \gamma[\eta * Q(s_{k+1}, a'; \theta, \alpha, \beta) + (1 - \eta) * Q(s_{k+1}, a'; \theta', \alpha, \beta)]$
    Perform gradient descent step on $y_k - Q(s_k, a_k; \theta, \alpha, \beta))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

IV. Empirical Analysis

i. Platform and Parameter settings

This paper takes the gym experimental platform developed by openai as the experimental environment, adopting the control problem Cartpole in Gym as the experimental object. The experiment takes the D3QN algorithm as the baseline to compare them with the newly proposed WD3QN algorithm. The hyperparameter settings are as follows:

Hyperparameter	Value
Minibatch Size	32
Replay Memory Size	200000
γ	0.95
Learning rate	0.0005
ϵ	1
ϵ decay	0.995
Training Epoch	600
c	1,10,100

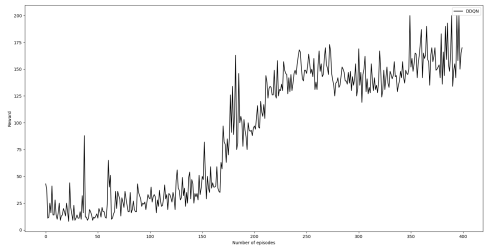
Table 1: *Setting of Hyperparameters*

In the WD3QN algorithm, taking the system state tuple as the input, the first fully connected layer

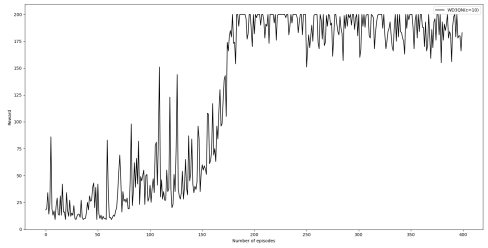
ii. Result analysis

We first investigate the impact of different hyperparameter c on the WD3QN algorithm. First taking 1,10,100 and conduct training. Figure 2 shows the performance of the WD3QN algorithm with different value of c. X-axis represents the training stage and y-axis represents the reward of each training episode.

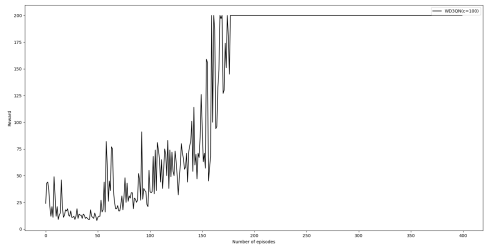
Figure 1 indicates that the performance of



(a) c=1



(b) c=10



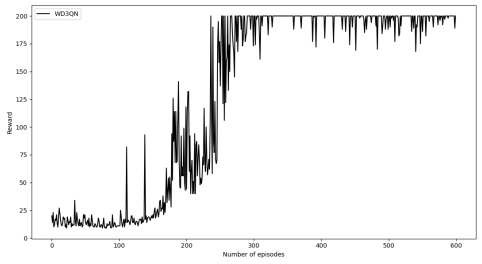
(c) c=100

Figure 1: *Experimental results of the algorithm with hyperparameter $c = 1, 10, 100$*

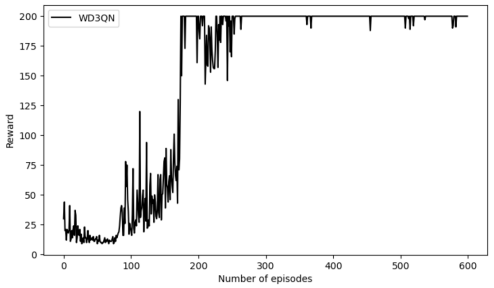
WD3QN algorithm is better when $c = 10$ and $c = 100$ compared to that when $c = 1$, which means that the WD3QN algorithm can reduce the estimation bias better and learn the key policy more effectively with parameter $c = 10$, $c = 100$ in this experiment. The performance is close for the algorithm when $c = 10$ and $c = 100$. However, such setting is not accurate

enough, a better value of c may still exist. Probably c can be denoted as a function of the Q value and make c a dynamically self-learned parameter. In experiments of this project, the default value of parameter $c = 10$. Figure 2 compares the total reward of three algorithms in 600 episodes directly.

Figure 2 indicates that during the training pro-



(a) D3QN



(b) WD3QN

Figure 2: *Comparison between the total reward of D3QN and WD3QN*

cess, as the training phase increases, the agent reaches the highest total reward faster with WD3QN algorithm than the D3QN algorithm. In addition, the reward curve of the WD3QN algorithm tends to be flatter compared to the other algorithms, which proves that it has better convergence and stability. The overestimation over Q value at the start of the training is significantly reduced and the underestimation in the late stage of training is also mitigated compared to the D3QN algorithm, especially when $episode > 300$. The experimental results show that the WD3QN algorithm using weight-based double Q learning can reduce the bias of target value estimation to a greater extent, and produce more accurate Q value estimation. Therefore, some key strategies can be learned more quickly and the performance can be improved. The performance of the WD3QN algorithm is better than that of DDQN, which shows that the use of weight-based double Q learning and training agent is more effective than using only double Q learning and training. Furthermore, the training time of different algorithms is compared in this paper. The corresponding time is listed in table 2:

From the table, it can be seen that the training

Algorithm	Training time/min
D3QN	40.1
WD3QN	86.6

Table 2: *Training time comparison between three algorithms*

time of the WD3QN algorithm is over twice the D3QN algorithm cost and, simultaneously, provides better performance. In practical application with more complex environment, a trade-off should be made between the increment of training time and the performance to enhance the comprehensive efficiency of the algorithm.

V. Conclusion

In conclusion, with the combination of dueling architecture and double deep Q network and introducing the double estimator with weight to balance the overestimation of single estimator and underestimation of the double estimator, can estimate the target value much more accurately, thus providing better policy selection. From the empirical result based on Cartpole problem, it can be observed that WD3QN algorithm can dramatically mitigate the underestimation problem which Double Deep Q-Network and Dueling architecture do not resolve. The result is inconsistent and the parameter c is not optimal in this project. Future investigation should include how to turn the hyperparameter c in the WD3QN algorithm into

an adaptive parameter instead of a constant. In addition, apply Recurrent Neural Network in the model to further improve the performance of the DQN algorithm. With the current hardware environment, it is not allowed to test the algorithm on bigger and more complex data sets due to time costs.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S. 2015. Human-level control through deep reinforcement learning. *nature, 518(7540)*, pp.529-533.

[2] Watkins, C.J. and Dayan, P. 1992. Q-learning. Machine learning. *Machine learning, 8(3-4)*, pp.279-292.

[3] H. Van Hasselt, A. Guez, and D. Silver. 2015. Deep reinforcement learning with double q-learning. *arXiv preprint*, arXiv:1509.06461.

[4] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M. and Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. *In International conference on machine learning*, pp.1995-2003.