Variability and Overestimation of Deep Q Network

Ziteng Weng
School of Data Science
Chinese University of Hong Kong, Shenzhen
Shenzhen 518172, China
117010283@link.cuhk.edu.cn

Abstract

It is well known that Tabular Q Leaning and Vanilla Deep Q Network (DQN) both overestimate the state-action value functions. Moreover, the large variability of DQN tends to affect its performance adversely. To deal with these two issues, some scientists have proposed some extensions of DQN, including Double DQN, Ensemble DQN and Averaged-DQN. In this paper, some experiments were conducted to compare the performances of these proposed algorithms and Averaged-DDQN (a combination of Double DQN and Averaged-DQN) in Gridworld environments.

1 Introduction

Reinforcement Learning seeks an optimal policy for a sequential decision-making problem by maximizing a numerical reward signal (Sutton and Barto, 1998). Q Learning is one of the famous Reinforcement Learning algorithms, but it is not impractical in large-scale problems, which means that with the increased number of state-action pairs, Q Learning becomes inefficient. Besides, in Q Learning, the action-value function is estimated separately for each sequence, without any generalization (Mnih et al., 2013). Most importantly, real-world problems usually involve high-dimensional inputs forcing linear function approximation methods to rely upon hand-engineered features. These problem-specific features diminish the agent flexibility, so the need for an expressive and flexible non-linear function approximation emerges (Anschel et al., 2017).

One of the most popular algorithms that are used to deal with the preceding issues is Deep Q Network (DQN). DQN combines the non-linear function approximation technique known as Neural Network with Q Learning to perform better in large-scale problems and the non-linear function approximation. DQN also introduces the concept of the target network and uses an Experience Replay buffer (ER buffer) to increase the training stability by breaking the Reinforcement Learning problem into sequential supervised learning tasks (Anschel et al., 2017).

However, DQN sometimes learns unrealistically high action values because it includes a maximization step over estimated action values, which tends to prefer overestimated to underestimated values (Van Hasselt et al., 2015). Besides, the approximation error variance in the target values of DQN is generally large.

In recent years, many algorithms based on DQN have been proposed to deal with the preceding issues. For example, Double DQN (DDQN) is proposed to reduce the overestimation (Van Hasselt et al., 2015). Besides, Averaged-DQN is used to reduce the instability and the variability of DQN (Anschel et al., 2017).

In this paper, some experiments were conducted to compare the variability, overestimation and computational efforts of DQN, DDQN and Averaged-DQN in Gridworld environments. Moreover, the combination of DDQN and Averaged-DQN was proposed, and it will also be included in the experiments.

Preprint. Under review.

2 Background

This project is based on the following Reinforcement Learning algorithms: Deep Q Network, Double DQN, Ensemble DQN and Averaged-DQN.

2.1 Reinforcement Learning

In this paper, a usual Reinforcement Learning framework is considered (Sutton and Barto, 1998). An agent is faced with a sequential decision-making problem, where interaction with the environment occurs at discrete time steps ($t = 0, 1, ...$). At time t the agent observes state $s_t \in S$ selects an action $a_t \in A$, which results in a scalar reward $r_t \in R$, and a transition to a next state $s_{t+1} \in S$ (Anschel et al., 2017). We could use the expected sum of future rewards to estimate the state-action value of each action, i.e., given a state s and a policy π, the state-action value is

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_1 + \gamma r_2 + \ldots \mid s_0 = s, a_0 = a, \pi\right]$$

where γ is a discount factor. The optimal value is denoted as $Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$ and the optimal policy is denoted as $\pi^*(s) \in \operatorname{argmax}_a Q^*(s, a)$.

2.2 Q Learning

Q Learning is one of the most important Reinforcement Learning algorithms based on the value iteration update. Tabular Q learning contains a table that is used to store and update the current state-action values. The details of Q Learning (Tabular) were shown in Algorithm 1.

With the increase of state-action pairs, it is impractical to maintain the table. A common solution to this issue is to use the function approximation parametrized by θ, such that $Q(s, a) \approx Q(s, a, \theta)$ (Anschel et al., 2017). Moreover, in Q Learning, the action-value function is estimated separately for each sequence, without any generalization. One of the solutions to this issue is to use the experience replay buffer that will be introduced in Section 2.3.

Algorithm 1: Q Learning (Tabular)

Input : initial state s_0, step size sequence μ_t, $Q^0(s, a) \equiv 0$

for $t \leftarrow 1$ **to** n **do**

Choose action a_t^* of s_t, observe r_t and s_{t+1}

$$Q^{t+1}\left(s_t, a_t^*\right) = \left(1 - \mu_t\right) Q^t\left(s_t, a_t^*\right) + \mu_t \left(r_t + \gamma \max_{a'} Q^t\left(s_{t+1}, a'\right)\right)$$

$$= Q^t\left(s_t, a_t^*\right) + \mu_t \left(r_t + \gamma \max_{a'} Q^t\left(s_{t+1}, a'\right) - Q^t\left(s_t, a_t^*\right)\right)$$

end

Output : $\hat{Q} = Q^{t+1}$ and $\hat{\pi}(s) = \operatorname{argmax}_a \hat{Q}(s, a)$

2.3 Deep Q Network

In 2013, Deepmind proposed a famous Deep Reinforcement Algorithm called Deep Q Network (DQN). DQN will interact with the environment in each step to get a new transition and store it into the experience replay buffer. For every predefined period c, The DQN loss is minimized using a Stochastic Gradient Descent (SGD) variant, sampling mini-batches from the ER buffer (Anschel et al., 2017). Also, for every predefined period d, the weights of behavior DQN will be copied to the weights of target DQN. The details of this algorithm are covered in Algorithm 2.

It is well known that DQN leads to the overestimation issue because of the use of the max operator. To deal with this issue, researchers proposed Double DQN and Averaged-DQN which will be introduced in Section 2.4 and Section 2.5.

2.4 Double DQN

To deal with the overestimation issue of DQN, Deepmind proposed a famous algorithm called Double DQN in 2015. Double DQN uses two Neural Networks: one is used to select the action while another

Algorithm 2: Deep Q Network

Input : update frequency of behavior network c and update frequency of target network d
Initialize replay memory D to capacity N and behavior network $\tilde{Q}(s, a, \tilde{\theta})$ with random weights
Initialize target network $Q(s, a, \theta)$ with same weights of $\tilde{Q}$
for $episode \leftarrow 1$ **to** M **do**
  Initial sequence $s_1 = \{x_1\}$
  for $t \leftarrow 1$ **to** T **do**
    Choose action a_t^* using $\tilde{Q}$, observe r_t and s_{t+1}
    Store transition $\{s_t, a_t, r_t, s_{t+1}\}$
    if *t mod c == 0* **then**
      Sample random minibatch of transitions $\{s_t, a_t, r_t, s_{t+1}\}$ from D Set

$$\mathrm{y}_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q\left(s_{j+1}, a', \theta\right) & \text{for non-terminal } s_{j+1} \end{cases}$$

      Perform gradient descent step on $(y_j - \tilde{Q}(s_j, a_j, \tilde{\theta}))^2$
    end
    if *t mod d == 0* **then**
      Copy the weights of $\tilde{Q}$ to Q
    end
  end
end
Output : $Q(s, a, \theta)$

is used for the estimation. More specifically, Double DQN uses the behavior network to choose the optimal action for s_{t+1} and uses the target network to calculate the estimated Q-values. This decoupled idea in Double DQN separates these two operations to eliminate the effect from the max operator in standard DQN. The details of this algorithm are covered in Algorithm 3.

Algorithm 3: Double DQN

Input : update frequency of behavior network c and update frequency of target network d
Initialize replay memory D to capacity N and behavior network $\tilde{Q}(s, a, \tilde{\theta})$ with random weights
Initialize target network $Q(s, a, \theta)$ with same weights of $\tilde{Q}$
for $episode \leftarrow 1$ **to** M **do**
  Initial sequence $s_1 = \{x_1\}$
  for $t \leftarrow 1$ **to** T **do**
    Choose action a_t^* using $\tilde{Q}$, observe r_t and s_{t+1}
    Store transition $\{s_t, a_t, r_t, s_{t+1}\}$
    if *t mod c == 0* **then**
      Sample random minibatch of transitions $\{s_t, a_t, r_t, s_{t+1}\}$ from D Set

$$\mathrm{y}_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma Q\left(s_{j+1}, \underset{a\prime}{\operatorname{argmax}} \tilde{Q}\left(s_{j+1}, a, \tilde{\theta}\right), \theta\right) & \text{for non-terminal } s_{j+1} \end{cases}$$

      Perform gradient descent step on $(y_j - \tilde{Q}(s_j, a_j, \tilde{\theta}))^2$
    end
    if *t mod d == 0* **then**
      Copy the weights of $\tilde{Q}$ to Q
    end
  end
end
Output : $Q(s, a, \theta)$

2.5 Ensemble DQN

The first algorithm used to reduce approximation error variance proposed by Anschel et al., 2017 is Ensemble DQN (EDQN). This algorithm is based on Ensemble Learning, i.e., trains K DQNs in parallel and uses the average of these K Q-values estimates to be the output. This algorithm could reach $\frac{1}{K}$ variance reduction, i.e., $\mathrm{Var}[Q_{EDQN}(s,a)] = \frac{1}{K}\mathrm{Var}[Q_{DQN}(s,a)]$. Moreover, compared to DQN, this algorithm requires K-fold computation effort because this algorithm trains K models. The details of this algorithm are covered by Algorithm 4.

Algorithm 4: Ensemble DQN

Input : update frequency of behavior network c and update frequency of target network d
Initialize replay memory D and K behavior networks $\tilde{Q}(s,a,\tilde{\theta}^k)$ with random weights
Initialize K target networks $Q(s,a,\theta^k)$ with same weights of corresponding $\tilde{Q}(s,a,\tilde{\theta}^k)$
for $episode \leftarrow 1$ **to** M **do**
  Initial sequence $s_1 = \{x_1\}$
  for $t \leftarrow 1$ **to** T **do**
    Choose action a_t^* using $\tilde{Q}$, observe r_t and s_{t+1}
    Store transition $\{s_t, a_t, r_t, s_{t+1}\}$
    if *t mod c == 0* **then**
      Sample random minibatch of transitions $\{s_t, a_t, r_t, s_{t+1}\}$ from D
      for $k \leftarrow 1$ **to** K **do**
        Set $\mathrm{y}_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q\left(s_{j+1}, a', \theta^k\right) & \text{for non-terminal } s_{j+1} \end{cases}$
        Perform gradient descent step on $(y_j - \tilde{Q}(s_j, a_j, \tilde{\theta}^k))^2$
      end
    end
    if *t mod d == 0* **then**
      Copy the weights of K behavior networks to corresponding target networks
    end
  end
end
Output : $Q(s,a) = \frac{1}{K}\sum_{k=1}^{K} Q\left(s,a;\theta_i^k\right)$

2.6 Averaged-DQN

Averaged-DQN (ADQN) can be used to deal with both the overestimation issue and the large variability issue. Averaged-DQN is a simple extension of DQN, which uses the K previously learned Q-values estimates to produce the current action-value estimate. Moreover, the output of Averaged-DQN is the average over the last K previously learned Q-networks (Anschel et al., 2017). Compared to Ensemble DQN, Averaged-DQN can achieve more variance reduction, i.e., $\mathrm{Var}[Q_{ADQN}(s,a)] < \frac{1}{K}\mathrm{Var}[Q_{DQN}(s,a)]$. Besides, the computation effort of Averaged-DQN is smaller than it of Ensemble DQN. More specifically, Averaged-DQN requires K-fold computation effort in the forward propagation and the same computation effort in the backpropagation. The details of this algorithm are covered by Algorithm 5.

3 Approach

In this project, all the algorithms covered in Section 2 are developed. Moreover, The combination of DDQN and Averaged-DQN is also concluded in this project.

Based on the discussion in Section 2.4 and Section 2.6, Double DQN and Averaged-DQN use different ideas to reduce overestimation. Therefore, we could combine the ideas of these two algorithms and develop a new algorithm called Averaged-DDQN. In this project, some experiments on this algorithm are also conducted to determine whether it could reduce overestimation further or not.

Algorithm 5: Averaged-DQN

Input : update frequency of behavior network c and update frequency of target network d
Initialize replay memory D to capacity N and behavior network $\tilde{Q}(s, a, \tilde{\theta})$ with random weights
Initialize K target networks $Q(s, a, \theta^k)$ with same weights of corresponding $\tilde{Q}$
$i = 1$
for $episode \leftarrow 1$ **to** M **do**
  Initial sequence $s_1 = \{x_1\}$
  for $t \leftarrow 1$ **to** T **do**
    Choose action a_t^* using $\tilde{Q}$, observe r_t and s_{t+1}
    Store transition $\{s_t, a_t, r_t, s_{t+1}\}$
    if *t mod c == 0* **then**
      Sample random minibatch of transitions $\{s_t, a_t, r_t, s_{t+1}\}$ from D
      Set $\mathrm{y}_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_a \left(\frac{1}{K} \sum_{k=1}^{K} Q\left(s_{j+1}, a', \theta^k\right) \right) & \text{for non-terminal } s_{j+1} \end{cases}$
      Perform gradient descent step on $(y_j - \tilde{Q}(s_j, a_j, \tilde{\theta}))^2$
    end
    if *t mod d == 0* **then**
      Copy the weights of $\tilde{Q}$ to $(i \bmod K)$-th target network
      $i = i + 1$
    end
  end
end
Output : $Q(s, a) = \frac{1}{K} \sum_{k=1}^{K} Q\left(s, a; \theta_i^k\right)$

4 Experiment results

4.1 Environments

The environments used in this project are Gridworld which is a common Reinforcement Learning benchmark and the optimal value function Q^* can be accurately computed using Bellman Equation.

The first environment is a $20 * 20$ Gridworld (Figure 1 (a)), i.e., the state space contains pairs of points from a 2D discrete grid $\{(x, y)_{x,y \in 1,2,\ldots,20}\}$. The algorithm interacts with the environment through raw pixel features with a one-hot feature map $\phi(s_t) := (\mathbf{1}\{s_t = (x, y)\})_{x,y \in 1,\ldots,20}$. In each state, the agent has four actions, i.e., moving one step for each direction. As for the reward, The agent will receive a reward of $+1$ if it is in state $(20, 20)$. If the agent is in state $(20, 20)$ or goes out of the boundary, it will be sent back to the initial state $(0, 0)$.

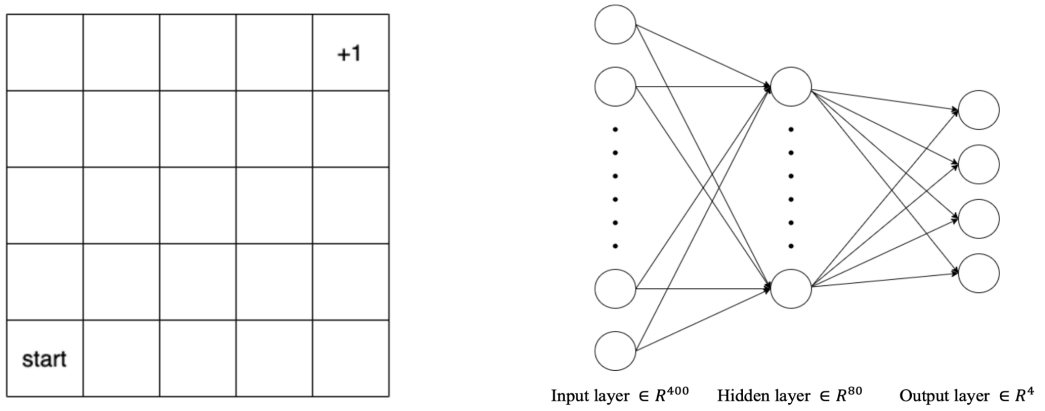
The second environment is based on the first environment. Some traps with a reward of -1 are included in this environment. The locations of these traps are: $(0, 19)$, $(19, 0)$, $(4, 4)$, $(15, 15)$, $(2, 8)$, $(17, 11)$, $(6, 10)$, $(13, 9)$, $(8, 6)$, $(11, 13)$. If the agent is in traps, it will also be sent to the initial state.

4.2 Network architecture

The network architecture (Figure 1 (b)) composes of a small fully connected neural network with one hidden layer of 80 neurons. For minimization of the DQN loss, the ADAM optimizer was used on 100 mini-batches of 32 samples per target network parameters (Anschel et al., 2017).

4.3 Expectation of the optimal state values

The average of optimal state values is formulated as $E_s[\max_a Q^*(s, a)]$ and it is used to evaluate the overestimation of the proposed algorithms. If the average predicted values of an algorithm are more closed to the average of optimal state values, we could conclude that this algorithm has a better estimation. We could use the value iteration method with specific stopping criteria to calculate the optimal state-action values of Gridworld and then take the expectation of optimal state values. The expectation of optimal state values for the above two environments are both 0.217.



(a) $5 * 5$ Gridworld (b) Network Architecture

Figure 1: The $5 * 5$ Gridworld and the network architecture

4.4 Hyperparameters

In this project, hyperparameter tuning was conducted on the learning rate and the update frequency of the behavior network. Candidates for the learning rate are 0.0002, 0.002, 0.01, 0.02 and 0.1, and candidates for the update frequency of the behavior network are 1, 2, 4 and 8. The results are in Figure 2.

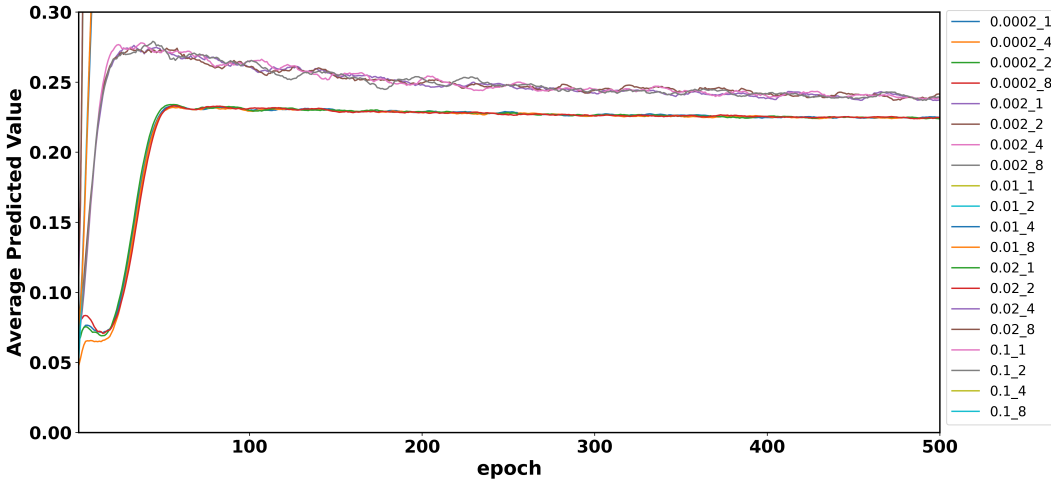


Figure 2: The result of hyperparameter tuning. Each line represents the average of 10 independent learning trials using the corresponding combination of the learning rate and the update frequency of the behavior network.

I chose 0.002 as the learning rate because the lines using this learning rate are similar to the results in the paper (Anschel et al., 2017). The differences between the results with different update frequencies are small, so the update frequency of the behavior network in the paper was used.

Other hyperparameters were copied from the paper (Anschel et al., 2017), and all hyperparameters I chose are in Table 1.

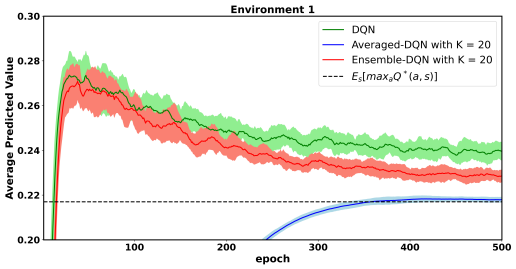
4.5 Variability

The size of the shaded areas was used as the indicator of the variability.

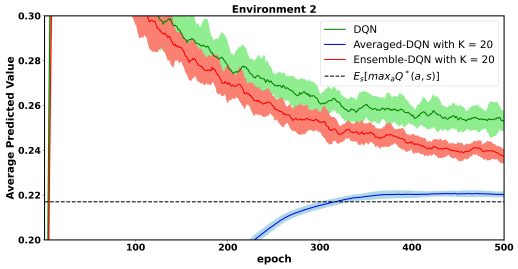
According to Figure 3 (a) and Figure 3 (b), it can be seen that Ensemble DQN and Averaged-DQN both can be used to reduce the approximation error variance. However, they could not reach the

Iterations in each epoch	400	batch size	32
Learning rate	0.002	Epsilon-Greedy rate ϵ	1 to 0.1 in 1000 steps
Update frequency of behavior net	4	Discount factor γ	0.9
Update frequency of target net	400	Number of epochs	500

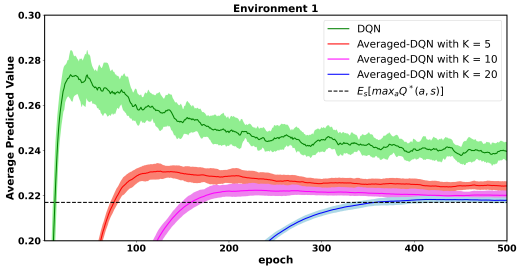
Table 1: The hyperparameter table



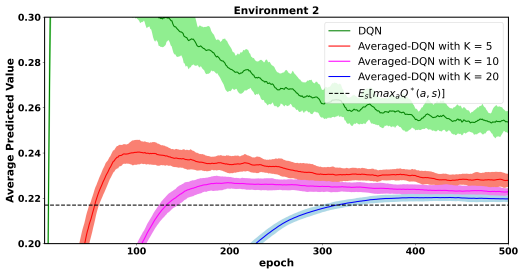
(a) DQN, Ensemble DQN with $K = 20$ and Averaged-DQN with $K = 20$ in Environment 1



(b) DQN, Ensemble DQN with $K = 20$ and Averaged-DQN with $K = 20$ in Environment 2



(c) DQN and Averaged-DQN with with $K = 5, 10$ and 20 in Environment 1



(d) DQN and Averaged-DQN with with $K = 5, 10$ and 20 in Environment 1

Figure 3: This figure shows the average predicted values of DQN, Ensemble DQN and Averaged-DQN in the environments. For Figure (a) and Figure (b), The bold lines are averaged over 20 independent learning trials. For Figure (c) and Figure (d), The bold lines are averaged over 40 independent learning trials. For each trial, there are 500 epochs, and each epoch contains 400 iterations. The shaded areas represent one standard deviation.

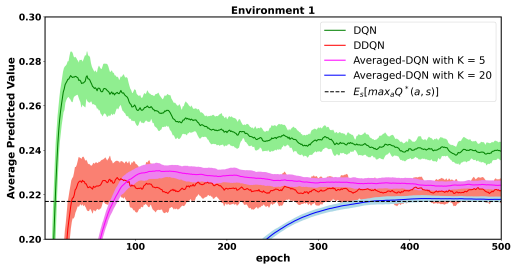
theoretical results in Section 2.5 and Section 2.6. Moreover, It is clear that Averaged-DQN is superior to Ensemble DQN in this issue, i.e., Averaged-DQN reaches a better variance reduction compared to Ensemble DQN with the same K.

According to Figure 3 (c) and Figure 3 (d), we can conclude that Averaged-DQN with larger K reaches a better variance reduction because it uses more previously learned Q-values estimates to stabilize the training process.

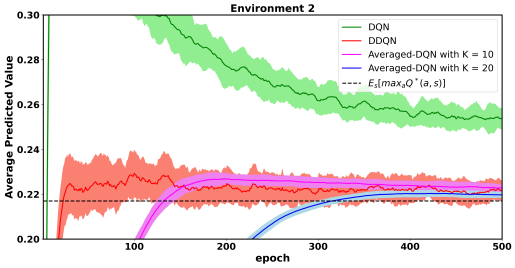
4.6 Overestimation

As stated in Section 4.3, the average of optimal state values was used to evaluate the overestimation of the proposed algorithms, and it is represented using the horizontal black lines in the figures. According to Figure 4 (a) and Figure 4 (b), it can be seen that Double DQN could reduce overestimation. However, the approximation error variance of this algorithm is still large. Moreover, Averaged-DQN can also reduce the overestimation, and the Averaged-DQN with larger K has a less overestimation.

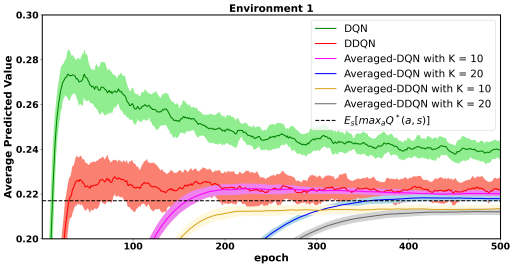
According to Figure 4 (c) and Figure 4 (d), the combination of the ideas of DDQN and Averaged-DQN leads to a better overestimation reduction. Nevertheless, Averaged-DDQN has an opposite issue: underestimation, i.e., the convergence values are smaller than the optimal one.



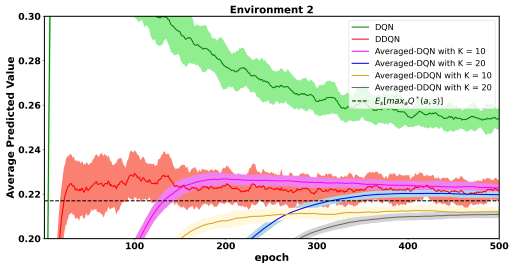
(a) DQN, DDQN and Averaged-DQN with $K = 5$ and 20 in Environment 1



(b) DQN, DDQN and Averaged-DQN with $K = 5$ and 20 in Environment 2



(c) DQN, DDQN and Averaged-DQN with $K = 5$ and 20 and Averaged-DDQN with $K = 5$ and 20 in Environment 1



(d) DQN, DDQN and Averaged-DQN with $K = 5$ and 20 and Averaged-DDQN with $K = 5$ and 20 in Environment 2

Figure 4: This figure shows the average predicted values of DQN, DDQN, Averaged-DQN and Averaged-DDQN in two environments. The bold lines are averaged over 40 independent learning trials. For each trial, there are 500 epochs, and each epoch contains 400 iterations. The shaded areas presents one standard deviation.

4.7 Comparisons between proposed algorithms

In the above experiments, DQN was used to be the benchmark.

In the variability issue, DDQN could not have a significant effect. Averaged-DQN and Averaged-DDQN both have good performances compared to Ensemble DQN.

As for the overestimation issue, Ensemble DQN has a limited effect. Among other proposed algorithms, Averaged-DDQN achieves the largest reduction. However, the underestimation issue appears. Compared with Double DQN, Averaged-DQN is better in dealing with this issue.

The last criterion used for the performance of the algorithms is the computation effort. To evaluate that, the averaged running times of one epoch using each algorithm in these two environments were measured, and they are summarized in Table 2.

Ensemble DQN is very time-consuming because it trains K DQN and averages the estimation. As for the idea of Double DQN, it does not take significant extra time, but it could achieve an overestimation reduction. Moreover, with the increase of K, the running time of averaged-DQN increases. However, this increase is not significant because the computation effort in the time-consuming backpropagation is the same as DQN.

4.8 Comparisons between two environments

As discussed in Section 4.1, the only difference between these two environments is that the second environment contains some traps with a reward of -1. Because the expectations of the optimal state values is defined as $E_s\left[\max_a Q^*(s,a)\right]$, the expectations of the optimal state values of these two environments are identical. Therefore, we could find that the average predicted values converge to the same values no matter which environment I used. According to Figure 3 and Figure 4, it can be seen that the overestimation issue is more severe in Environment 2, and more iterations are needed

Algorithm	Environment 1	Environment 2
DQN	161.15s	185.63s
Double DQN	163.31s	191.63s
Ensemble DQN with $K = 20$	2477.43s	191.63s
Averaged-DQN with $K = 5$	235.92s	292.81s
Averaged-DQN with $K = 10$	318.49s	400.87s
Averaged-DQN with $K = 20$	475.77s	600.09s
Averaged-DDQN with $K = 10$	320.85s	404.44s
Averaged-DDQN with $K = 20$	480.49s	620.83s

Table 2: The running times of one epoch using each algorithm in these two environments. They were measured using NVIDIA Tesla P100 on the Google Colab platform.

Usage	Hyperparameter tuning	Environment 1	Environment 2
Ensemble DQN		GW_Ensemble.ipynb	Trap_Ensemble.ipynb
Other models	HT.ipynb	GW_DQN.ipynb	Trap_DQN.ipynb
Plot	HT_plot.ipynb	GW_plot.ipynb	Trap_plot.ipynb

Table 3: The information of submitted code files.

for convergence. For DQN and Double DQN, the average predicted values peak at around 0.44 in Environment 2, and these are much larger than those in Environment 1. Moreover, the approximation error variance of the algorithm in Environment 2 is larger than the approximation error variance in Environment 1 using the same algorithm. These two phenomena are due to the traps. More steps are required for the agent to learn this new feature, i.e., learn how to not fall into these traps.

Besides, shown in Table 2, Environment 2 takes more computational efforts to train the algorithms compared to Environment 1.

5 Submitted code files

The submitted code files are based on the following Github repository: `https://github.com/elephanting/Averaged-DQN-Pytorch`.

There are 8 code files, and their usages are summarized in Table 3.

6 Conclusion

In this project, I have learned the ideas of DQN, DDQN, Ensemble DQN, Averaged-DQN and Averaged-DDQN. After that, I implemented these algorithms and Averaged-DDQN to compare their performances, especially the variability, overestimation and computation efforts. Based on the discussion in Section 4.7, Averaged-DQN is the most powerful algorithm among the proposed algorithms because it can simultaneously reduce variability and overestimation using a reasonable computation effort.

As for the future work, I will investigate why Ensemble DQN and Averaged-DQN could not achieve the theoretical variance reductions. Moreover, more sophisticated environments and more advanced algorithms will be taken into consideration.

References

[1] Anschel, Oron, Baram, Nir and Shimkin, Nahum. Averaged-DQN: variance reduction and stabilization for deep reinforcement learning, *Proceedings of the 34th International Conference on Machine Learning*, pp. 176-185, 2017.

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

[3] Sutton, Richard S and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press Cambridge, 1998.

[4] Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double Q-learning. *arXiv preprint arXiv: 1509.06461*, 2015.