Course Project of Reinforcement Learning (2023), **xx**:xx 1–6

RESEARCH ARTICLE

Deep Q-learning and its variants

Author: Yao Tang

Keywords: Deep Q-learning, Double DQN, Dueling DQN

Abstract
In this experiment, we implemented three variants of the Deep Q-Network (DQN) algorithm: DQN, double DQN, and dueling DQN. We trained and evaluated these models on the MountainCar-v0 environment from OpenAI Gym. We found that all three variants of DQN were able to achieve good performance on this task, but that the dueling DQN model outperformed the others on several metrics, including stability and speed of learning. These results suggest that dueling DQN may be a promising direction for further research in the field of reinforcement learning.

Contents

1. Introduction

Q-learning is an important part of reinforcement learning, but tabular Q-learning that we implemented before is only suitable for environments with small state-action spaces. For environments with complex state-action spaces, it may consume a large amount of memory space and be inefficient in terms of computation runtime. Deep Q-Network (DQN) overcomes these drawbacks by using neural networks to approximate the Q-function.

In this experiment, we focused on three variants of the DQN algorithm: DQN, double DQN, and dueling DQN. These variants aim to address some of the limitations of the original DQN algorithm, such as overestimation of Q-values and poor performance on tasks with sparse rewards.

We evaluated these models on the MountainCar-v0 environment from OpenAI Gym, which is a classic RL task that requires the agent to learn to navigate a car up a steep hill. We compared the performance of the three DQN variants on several metrics, including reward, stability, and speed of learning.

The results of this experiment demonstrate that all three variants of DQN are capable of achieving good performance on the MountainCar-v0 task. However, the dueling DQN model outperformed the other variants on several metrics, suggesting that it may be a promising direction for further research in the field of reinforcement learning.

Overall, this experiment highlights the potential of DQN and its variants for solving challenging RL tasks, and underscores the importance of continuing research into these algorithms.

2. Preliminaries

2.1. Deep Q-learning

DQN, or Deep Q-Network, is a reinforcement learning algorithm that uses a neural network to approximate the Q-function. It can handle environments with large or continuous state spaces. DQN has been applied to a variety of tasks, such as Atari games and robotics, and has achieved state-of-the-art performance on many benchmarks. The algorithm works by training a neural network to predict the Q-values of each action for a given state. It then uses an epsilon-greedy policy to choose actions based on these Q-values. DQN also employs a technique called experience replay, which stores transitions in a replay buffer and samples batches of them to train the network. This makes the learning process more stable and efficient. The detailed algorithm is shown below.

Algorithm 1 Deep Q-Learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for $episode = 1, M$ **do**
  Initialize sequence $s_1 = x_1$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a \, Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image $xt + 1$
    Set $st + 1 = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    if $t \bmod C = 0$ **then**
      Reset $\hat{Q} = Q$
    end if
  end for
end for=0

2.2. Double DQN

A Double Deep Q-Network, or Double DQN utilises Double Q-learning to reduce overestimation by decomposing the max operation in the target into action selection and action evaluation. We evaluate the greedy policy according to the online network, but we use the target network to estimate its value. The update is the same as for DQN, but replacing the target with:

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \hat{Q}(\phi_{j+1}, argmax_{a'} Q(s_{j+1}, a'; \theta); \theta^-) & \text{otherwise} \end{cases}$$

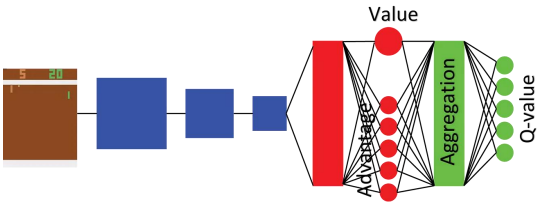

Figure 1. *Network structure of Dueling DQN.*

Compared to the original formulation of Double Q-Learning, in Double DQN the weights of the second network are replaced with the weights of the target network for the evaluation of the current greedy policy.

2.3. Dueling DQN

The difference in Dueling DQN is in the structure of the model. The model is created in a way to output the formula below:

$$Q(s,a) = V(s) + A(s,a)$$

Here, the V(s) stands for the Value of state s and A is the Advantage of doing action a while in state s. The Value of a state is independent of action. It means how good is it to be in a particular state.

How is it different from Q-value? Let's explain it with an example. The agent might be in a state where each of the actions would give the same Q-value. So there is no good action in this state. What would happen if we divide the Q-value to Value of a state and the Advantage that each action has. If every action has the same result, then the Advantage of each action will have the same value. Now, if we subtract the mean of all the Advantages from each advantage, we get zero (or close to zero) and Q-value would actually be the value that the state has.

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{N}\sum_{a} A(s,a)$$

So overtime the Q-value would not overshoot. The states that are independent of action would not have a high Q-value to train on.

3. Experiment Analysis

3.1. Setting

We implemented four variations of DQN, including Vanilla DQN, Double DQN, Dueling DQN, and D2 DQN (Double DQN + Dueling DQN), and conducted experiments in the MountainCar-v0 environment of the OpenAI Gym.

Both DQN and Double DQN used single hidden-layer neural networks with ReLu activation function, mapping from the state dimension inputs to the action dimension outputs. Dueling DQN and D2 DQN used shared neural networks with single hidden-layers and activation function of ReLu, but projected to 2 heads,a single neural output representing the state-value function and an advantage function output with action dimension, respectively.

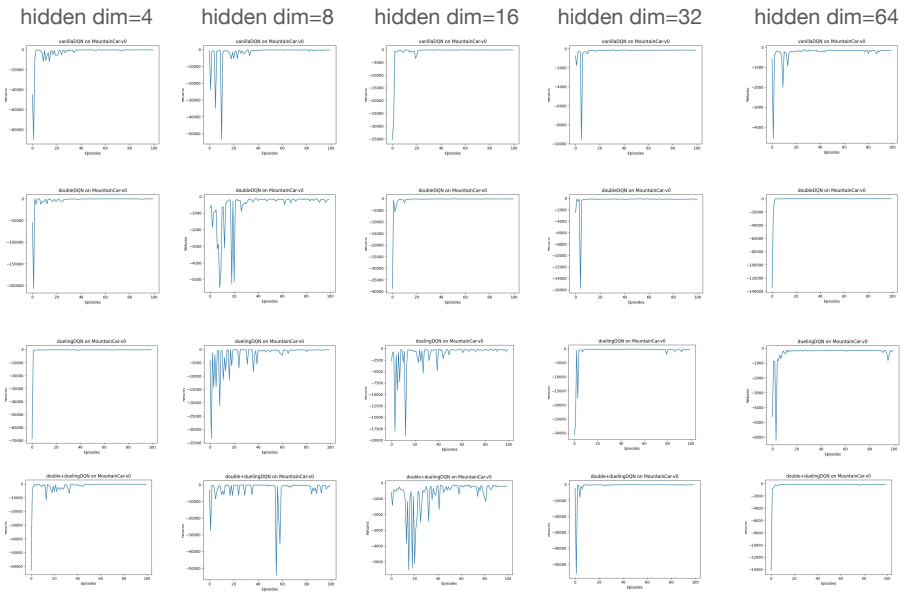

Figure 2. *The learning curves of DQN under different hidden dimensions..*

In the experiment, we compared the convergence speed, training performance, and degree of convergence for the four DQN methods with different hidden layer dimensions and convergence speed, convergence effect, and network convergence levels among the different DQN methods.

3.2. The influence of hidden dimension

In DQN, the dimension of the hidden layer can affect the algorithm's performance, especially in complex environments. A smaller hidden layer may not be sufficient to capture the complex features of the environment, leading to a decrease in performance. On the other hand, a larger hidden layer may result in overfitting and longer training times. Therefore, selecting an appropriate hidden layer dimension can improve the training efficiency of the algorithm without sacrificing performance. However, determining the optimal hidden layer dimension usually requires experimentation and comparison of algorithm performance under different dimensions. Therefore, in the MountainCar-v0 environment, we compared the effects of hidden dimensions of 4/8/16/32/64 on four types of DQN to investigate this issue, and the experimental results are shown in the following figure2.

The figure shows the training curves of four DQN algorithms under different hidden dimensions (4/8/16/32/64) in the MountainCar-v0 environment. The five columns represent the hidden dimensions, while the four rows correspond to Vanilla DQN, Double DQN, Dueling DQN, and D2 DQN, respectively. The y-axis represents the return (the sum of total rewards) of completing one episode during training, and the x-axis represents the number of episodes trained (proportional to the training time).

The training environment of MountainCar has a relatively simple action and state space, and it can be observed that the four DQN algorithms already perform well when hidden dim is 4. As hidden dimension gradually increases (up to 8/16), some oscillations and slower convergence can be observed in the graphs. Intuitively, this is likely due to the increase in model complexity leading to more parameters to train and slower convergence. However, when hidden dimension increases to 64/32, the convergence speed becomes faster and more stable. This may be due to the increased hidden dimension enhancing the model's representational power and better capturing the complex features of the environment, thereby improving the algorithm's performance and training efficiency. Additionally, as the hidden dimension

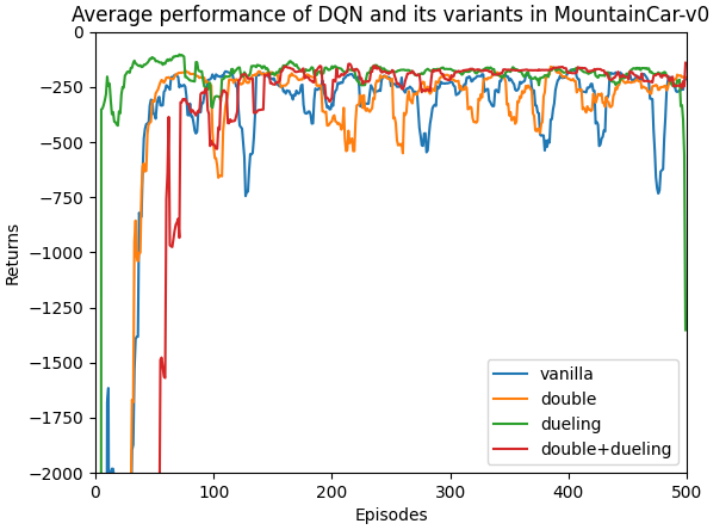

Figure 3. *The learning curves of 4 DQN algorithms in 500 episodes with hidden dim=4..*

reaches a certain point, the model's representational power will saturate, preventing overfitting and ensuring stable training of the algorithm.

3.3. Comparison between Vanilla DQN, Double DQN, Dueling DQN and D2 DQN(Double+Dueling DQN)

We plotted the training curves of Vanilla DQN, Double DQN, Dueling DQN, and D2 DQN (Double+Dueling DQN) during the training of 500 episodes on the same figure, as shown in Figure 3.

From the experimental results in Figure 3, we can see that in terms of convergence speed, Dueling DQN has the fastest convergence speed, while Vanilla DQN and Double DQN have similar convergence speeds, and Double+Dueling DQN has the slowest convergence speed. In terms of convergence stability, Double+Dueling DQN > Dueling DQN > Double DQN > Vanilla DQN, which is also intuitively understandable.

We also conducted evaluation experiments on the four trained agents by evaluating 20 episodes in the same environment, and the average returns obtained are shown in the Table1 below.

Algorithm	**Average Evaluation Return**
Vanilla DQN	-211.0
Double DQN	-170.0
Dueling DQN	-150.0
D2 DQN	-143.0

Table 1. *Average evaluation Returns in 4 DQN algorithms.*

It can be seen that Dueling DQN can effectively help the agent accurately obtain the most important information in the current environment and choose the optimal action; Double DQN can effectively alleviate the overestimation problem of DQN through a simple modification on Vanilla DQN, which helps the agent learn better. From the evaluation results, it can be seen that combining different improvements to DQN can help the agent learn better to varying degrees.

4. Conclusion

In this project, we implemented and compared four variations of DQN algorithms: Vanilla DQN, Double DQN, Dueling DQN, and D2 DQN (Double DQN + Dueling DQN) in the MountainCar-v0 environment of OpenAI's gym. Our experiments showed that Dueling DQN had the fastest convergence rate, while Vanilla DQN and Double DQN had similar and slower convergence rates. D2 DQN (Double DQN + Dueling DQN) had the slowest convergence rate among the four algorithms. As for convergence stability, Double + Dueling DQN > Dueling DQN > Double DQN > Vanilla DQN.

In the evaluation experiments, the average returns of the four agents were compared on 20 episodes. The results showed that Dueling DQN was able to effectively help the agent accurately obtain the most important information in the current environment and select the optimal action. Double DQN was able to effectively alleviate the overestimation problem of DQN with a simple modification and help the agent learn better. From the evaluation results, it can be concluded that combining different improved DQN methods can help the agent learn better to varying degrees.

In conclusion, our experiments showed that Dueling DQN had the fastest convergence rate and combining different improved DQN methods can help the agent learn better to varying degrees. The choice of appropriate hidden layer dimensions in the DQN algorithm can improve training efficiency without sacrificing learning performance. These findings can provide guidance for the selection and tuning of DQN algorithms in other environments.