EE526X Final Project

Xingche Guo

December 20, 2019

1 Introduction

In this project, we use the environment, 'Acrobot-v1', from the OpenAI gym package. As described in: `https://gym.openai.com/envs/Acrobot-v1/`, Acrobot is a 2-link pendulum with only the second joint actuated. Initially, both links point downwards. The goal is to swing the end-effector at a height at least the length of one link above the base. Both links can swing freely and can pass by each other.

For each observation, the state is a 6-dimension vector that consists of the cos and sin values of the two rotational joint angles, and the joint angular velocities, i.e. $(\cos\theta_1, \sin\theta_1, \cos\theta_2, \sin\theta_2, \omega_1, \omega_2)$. The angle of the second link is relative to the angle of the first link. An angle of 0 corresponds to having the same angle between the two links. A state of $(1, 0, 1, 0, \dots)$ means that both links point downwards. The action, labeled as $\{0, 1, 2\}$, is either applying -1, 0 or $+1$ torque on the joint between the two pendulum links. Figure 1 provides the visualization of Acrobot-v1. In this project, we are going to design a deep reinforcement learning algorithm (double deep Q-learning network) to accomplish that goal.

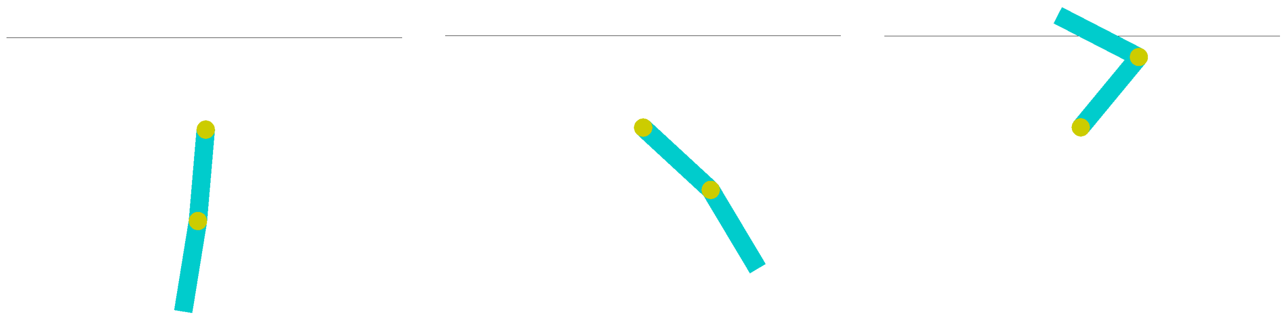

(a) initial status (b) middle status (c) successful status

Figure 1: Three status of Acrobot

2 Algorithm

In this part, I will introduce the details about using Double Deep Q-Network to approximate the Q function. I design the double DQN algorithm following the algorithm in the deep Q-network paper (Mnih et al., 2015), where two neural networks (current-net & target-net) are introduced to approximate and update function Q. According to Van Hasselt et al. (2016), the only difference between DQN and double

DQN is that in DQN, we compute the target Y_t^{DQN}:

$$Y_t^{DQN} = R_{t+1} + \gamma max_a Q_{\text{target}}(S_{t+1}, a);$$

while in double DQN, we compute the target Y_t^{DoubleQ}:

$$Y_t^{\text{DoubleQ}} = R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_a Q_{\text{target}}(S_{t+1}, a));$$

The algorithm in (Mnih et al., 2015) also use experience replay (i.e. remember M states and update the neural network only using data in a size-m batch, where $m \ll M$) to avoid overfitting and make the algorithm more robust. Besides, I made several adjustments to the original algorithm: (a). In order to accelerate the algorithm, we only update the current-net every 100 iterations; (b). We use soft-update to reset the target-net to further reduce the possibility of overfitting; (c). We stop training before going through all episodes if the model perform good enough. All the details of the double DQN algorithm can be found in Algorithm 1.

Algorithm 1 Pseudo Code for Double DQN

```
1: Initialize replay memory D to capacity 10000, batch size = 50;
2: Create two NN (current-net & target-net) as the initial function approximation of $Q(s, a)$ and $Q_{\text{target}}(s, a)$.
3: for Episode = $1, \cdots, 2000$ do
4:   Initialize state $S$;
5:   Set TotalStep = 0;
6:   for t = $1, \cdots, 2000$ do
7:     Choose action A based on $\epsilon$-greedy algorithm with $\epsilon$;
8:     Execute $A$ in environment, observe $R$, $S^{'}$, and terminated (i.e. termination status);
9:     Store and fresh the memory with $[S, A, R, S^{'}, \text{terminated}]$;
10:    TotalStep = TotalStep + 1;
       ****************** update current-net *******************
11:    if TotalStep % 100 == 0 then
12:      Randomly select a batch of state-action pairs from memory;
13:      for $[S, A, R, S^{'}, \text{Done}]$ in batch do
14:        Compute $A_{\max} = \text{argmax}_a Q(S^{'}, a)$;
15:        Compute $\hat{Q}_{\text{target}}(S, A) = R + \gamma Q_{\text{target}}(S^{'}, A_{\max})$;
16:        Train current-net by minimizing the square error between $Q_{\text{target}}(S, A)$ and $\hat{Q}_{\text{target}}(S, A)$;
17:      end for
18:    end if
       ***************** soft-update target-net *****************
19:    if TotalStep % 2000 == 0 then
20:      Get NN parameters $\Theta_{\text{current}}$ and $\Theta_{\text{target}}$;
21:      $\Theta_{\text{target}} \leftarrow \tau\Theta_{\text{current}} + (1 - \tau)\Theta_{\text{target}}$;
22:    end if
23:    Update state $S = S^{'}$
24:    Break if the episode is terminated.
25:  end for
26:  Break if the averaged rewards of the past few episodes is good enough.
27: end for
```

By experiments, I eventually choose $\epsilon = 0.1$, $\gamma = 0.9$, $\tau = 0.5$. We model the current-net and target net using:

(1). Single layer neural network with input size 6, output size 3, and linear function as the activation function.

(2). Two layers neural network with input size 6, hidden layer size 10, output size 3, RELU and linear functions as the two activation functions.

Finally, mean squared error is used as evaluation criteria and learning rate is chosen to be 0.001 under 'Adam' optimizer. Keras is used to build the double DQN under Python.

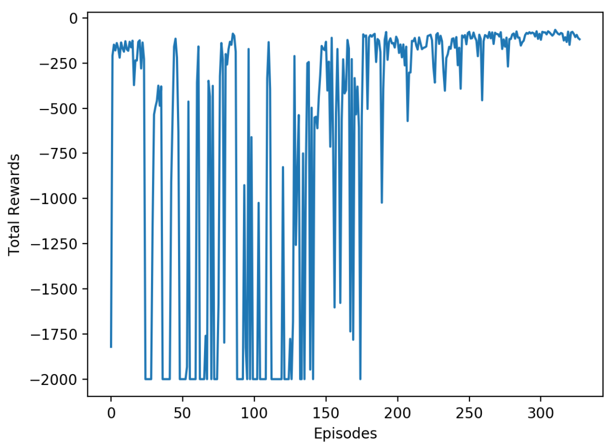
3 Results

3.1 Training Results

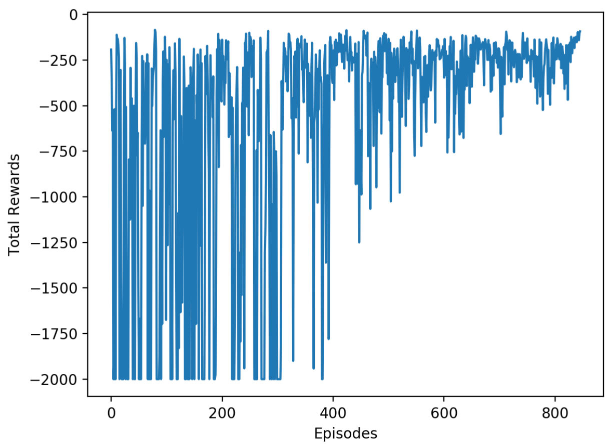
We compare the results between the two neural network structures in this section. From Figure 2, we find that the 2-layer model is not as stable as the single-layer model since it has larger variance of reward than the single-layer model. Meanwhile, the training procedure of the 2-layer model stops after 800 episodes, while the single-layer model only need around 300 episodes in order to perform well. A more detailed training summary can be found in Table 1.

structure	total episodes	comp. time [min]	avg. rewards	avg. rewards (last 100 episodes)
single-layer	328	6.1	-635.64	-122.58
2-layer	846	14.53	-554.56	-241.30

Table 1: Summary of training for the two models



(a) NN with single layer



(b) NN with two layers

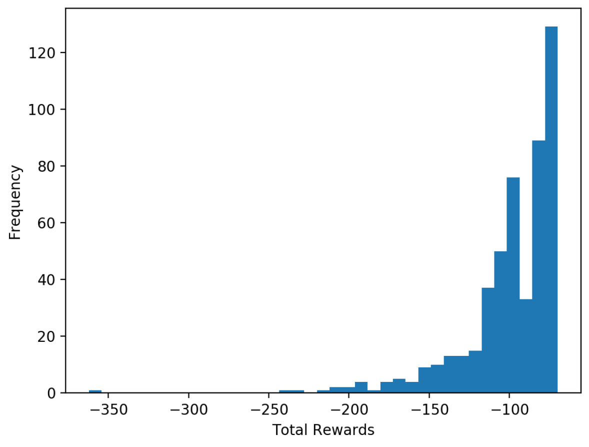
Figure 2: Total rewards (training) for different network structure

3.2 Testing Results

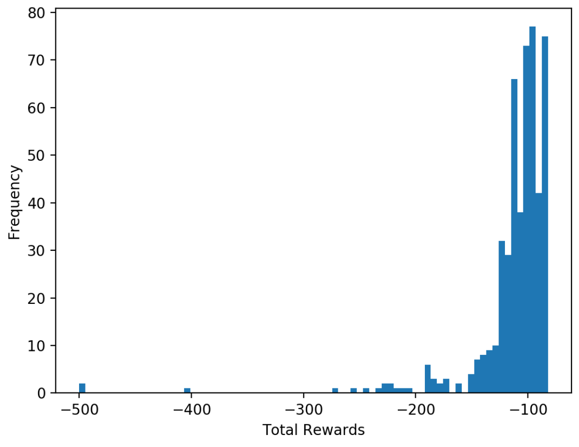
The histogram of testing results for the two trained models for 500 testing episodes can be found in Figure 3. It's easy to see that both models perform pretty well in terms of the total rewards. We can also see from Table 2 that, single-layer model is relatively better than the 2-layer model in terms of both mean and variance. Also, the single-layer model achieve the goal in 500 steps for every episodes, which is really impressive.

structure	mean	median	std. dev.	failed ratio (in 500 steps)
single-layer	-99.4	-93.0	30.9	0.0%
2-layer	-115.0	-105.0	73.9	0.4%

Table 2: Summary of testing for the two models



(a) NN with single layer



(b) NN with two layers

Figure 3: Histogram of the total rewards (500 testing episodes) for different network structure

4 Further Improvements

The results that I presented in this report is the best attempts. In fact, I faced several problems when I train the models:

- The parameter settings is really difficult to tune and the models are easily overfitted.
- Using more complex neural networks (more than 2 layers) is more likely to overfit in the end, the testing results for a 3-layer neural networks is really unstable.
- It is likely that the performance of the model becomes better at first and then get worse when there are too many episodes, that why I need to use the early stop trick.

The possible improvements are:

- Make more efforts on tuning the hyper-parameters since the parameter setting is important for the training.
- Compare the double DQN algorithms with other deep reinforcement learning algorithms (e.g. DQN, Actor-Critic, etc.).

- Use more reasonable reward function (I notice that the reward function is ?1 for all states and actions).

References

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In Thirtieth AAAI conference on artificial intelligence.