EE551000 System Theory

Homework 4 Report: Solving cartpole using RL

周柏宇 105061110

Implementation

✓ Briefly describe your implementation.

In our task, the environment outputs 4 types of continuous values as states. In order to solve it with the tabular methods, we need to quantize the continuous value into finite number of codes. After some research and try-and-error, for the cart pole position, since the round terminates as soon as the cart deviates from the center for 2.4, we only need to concentrate on the value from -2.4 to 2.4. Same argument can apply to the pole angular position, the round terminates when the angle is outside of -24~24 degree. Furthermore, with some prior knowledge, the angle should have more precision for better control feedback, we can divide the interval finer than the other values. The other 2 quantities are unbounded as document says, but after some trials, they appear to be inside certain intervals with high probability, thus we can confine the domain to smaller one. After quantizing the state, the rest of procedure is the very same as Q learning from homework 2.

If the agent is policy gradient, we can take continuous value as inputs and it will produce the distribution. I use a deep neural network as our policy and train it every 50 episodes for efficiency.
Network structure:
  Input layer: units=4
  Hidden layer: units=128, activation=ReLU, initialize with all ones
  Output layer: units=2, activation=softmax, initialize with all ones

Training parameters:
  Optimizer: Adam
  Learning rate: 0.01
  Discount factor: 0.99

In my actor-critic implementation, the agent consists of two deep neural networks, one actor network and the other is the critic network. The hyperparameters are as follows.

Actor network structure:

Input layer: units=4

Hidden layer: units=30, activation=ReLU, initialize with ones.

Dropout layer: dropout rate=0.6

Output layer: units=1, activation=sigmoid, initialize with ones.

Critic network structure:

Input layer: unit=4

Hidden layer 1: unit=16, activation=ReLU

Hidden layer 2: unit=16, activation=ReLU

Output layer: unit=1, activation=linear

(the default initializer is glorot_uniform())

Training parameters:

Optimizer: Adam

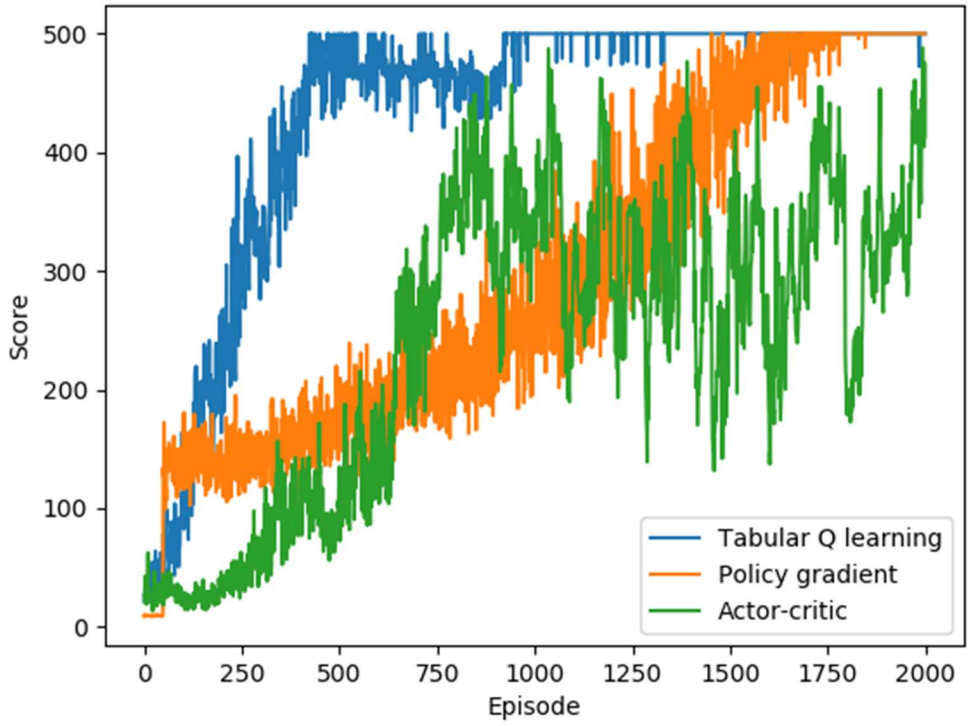
Actor learning rate: 0.001

Critic learning rate: 0.001

Discount factor: 0.99

Experiments and Analysis

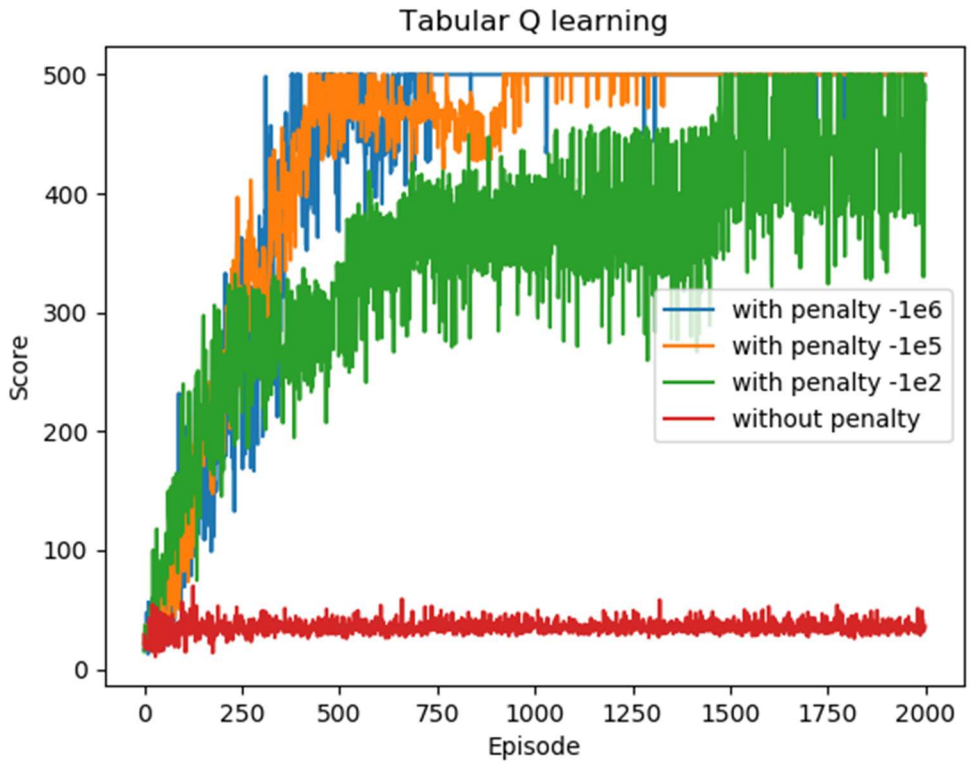
✓ Compare all methods regarding to performance.



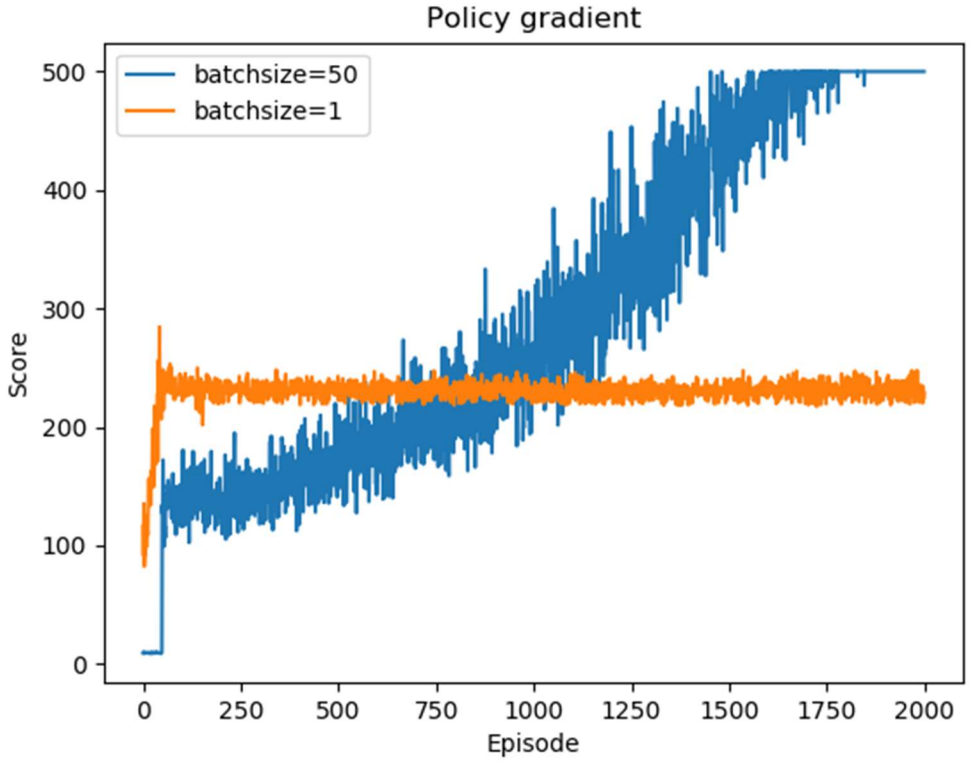
The figure above is the average score of 5 independently trained agents of three methods. I consider both the tabular Q learning and the policy gradient agent to be successful in solving the cartpole problem, since they can consistently achieve max time step after learning. For the tabular Q learning agent, it takes about 900 episodes; for the policy gradient agent, it takes about 1800 episodes. For the actor-critic case, I did not able to have it perform well. As the figure suggests, we can somewhat see the improvement but the performance variates largely. One thing to mention is that it takes forever to train the actor-critic agent, because it needs to update two networks for every step. Besides that, the agent is hard to finetune because the combination of parameters is massive compared with the former two agents. My comment to this result is that for the simple control problem like this, the policy gradient and the actor-critic agent maybe an overkill. Especially after state quantization, the resulting number of states is not a lot. In this case, tabular learning is very efficient. On the other hand, policy gradient can be used to dealt with non-discrete states and developing a stochastic policy. The result suggests it might not be too helpful. Finally, the actor-critic updates in a temporal manner and have the advantage of policy gradient and value leaning methods. In my opinion, there is a large gap between the theory and practice in the field of deep learning, because, unfortunately, finetuning affects the performance tremendously, and most of the tuning is nothing but trial-and-error. Therefore, I think the cartpole task cannot really give a convincing conclusion of the superiority of one method over the others, because I am not confident that they are all fine-tuned to their best.

- ✓ Analyze anything you'd like to.

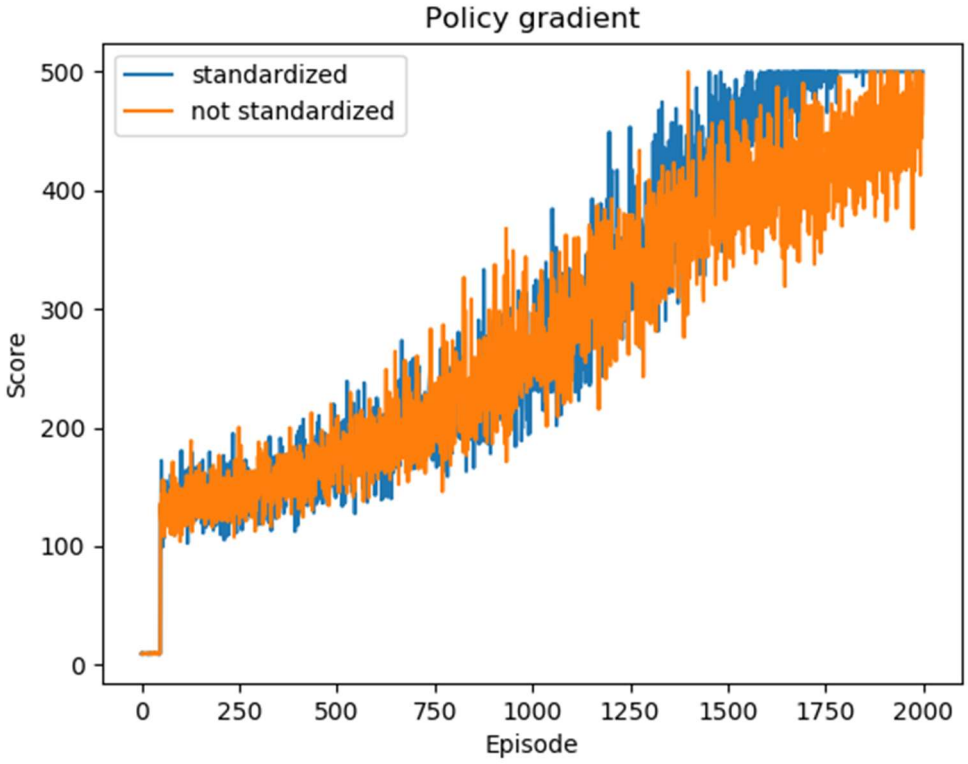
  I this section, I would like to share some techniques that significantly improve the performance of the agents.



I called this technique the “penalty”. Normally, we get reward +1 when the episode is not terminated, even for the step causing termination. Intuitively, we would like to punish those actions which make the episode terminate before 500 steps. Thus, I assign the reward to be a negative value for those actions. The figure above shows the effect of penalty. The finally choice of penalty is –1e5.



For the policy gradient agent, we actually don't need to update the agent for every episode! Updating in batch not only speeds up the training but also gives a significantly better result.



We can further improve the performance if we standardize the accumulated reward within a batch. This is a popular trick in training a DNN.

As for the last thing I share, I cannot come up with any theoretical explanation to this. I change the initialization of some weights in the neural network, and the performance difference is shown in the figure below.