**Task 1**

To use the Gym environment, I referred to the official documentation and learned that

1. gym.make function is used to create an environment for agent to interact with

2. env.reset method is used to reset environment especially after the current episode is over or done

3. env.render method is used to display the current state of environment

4. env.close method is used to close the display utility to reduce resource occupation

**Task 2**

I referred to the tutorial of Taxi problem and knew that the size of the observation space and action space is just 500 and 6 respectively, indicating that the problem can be solved by a navie look-up table of size 500 × 6. So, I followed the steps of tutorial and use Q-learning algorithm which will update the Q table after each iteration step. Finally, I make some optimization beyond the tutorial.

**Task 3**

I read the tutorial and replicated all the steps detailed, the code is placed in src directory. Now, I will show all the steps of Q-learning:

1. Create an environment Taxi-v3, which is available in gym package.
2. Initialize a Q table to store the Q values, which represent the long term reward according to a combination of specific state and action.
3. Use epsilon to determine whether randomly or greedily execute an action, and then update the Q table by the following equation

Qs,a ← (1 － ɑ) × Qs,a ＋ ɑ × (r ＋ γ × Qs’,a)

where ɑ is learning rate, Qs,a is Q value of state s and action a, r is the reward under the situation of the current action executed in the current state, γ is a discount rate that how extent to consider the long-term reward, and finally the s’ is the next state.

1. Set the next state as the current state, then repeat the 3 step until obtaining a good result.

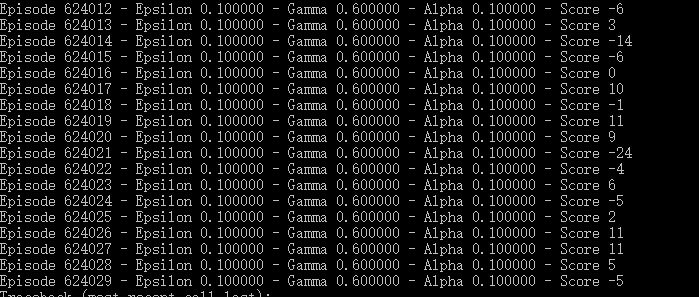


Fig 1. Q-learning

1. Evaluate the agent using the pre-trained Q table.

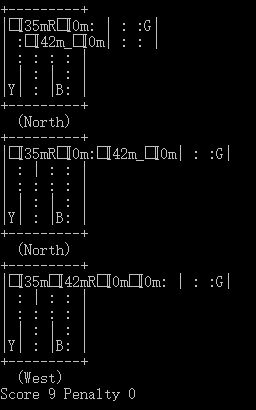


Fig 2. optimal

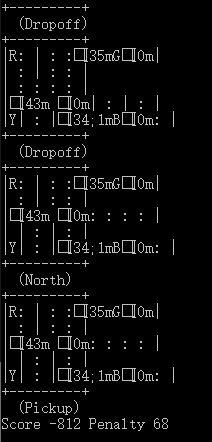


Fig 3. random

It is necessary to say that:

1. There is no need to use the unbound environment of Taxi-v3, which will never be done even after a lot of steps executed. Because sometimes too many wrong steps are executed will lead to too small score in which case the model couldn’t learn well.
2. Set epsilon to 1 at first and then decrease it after the agent have sampled a lot of experiences, which will help the agent not over-fitted.
3. Obviously, According to Fig 2 and 3, randomly sampling an action is not a good idea compared to the optimally selecting an action by Q-learning.

**Task 4**

In fact, when I took a view of the tutorial of CartPole, I didn’t find any description about Q-learning. So, I need to implement a Q-learning myself.

Besides, the Cartpole problem is unlike the aforementioned Taxi problem, it’s observation space is not discrete. In other words, the Cartpole problem couldn’t be easily solved by simple look-up table, which needs to have a large amount of row to store the Q values. Therefore, we need to find a way to represent Q values instead of just using Q table, which is impossible.

The tutorial has introduced three methods, such as random search, hill-climbing algorithm and policy gradient. All in all, I would like to implement Q-learning with policy gradient using two neural network, which is known as “Double Q Network”:

1. Construct two networks, one of which is used to predict the best action to be executed in the current state (online network), and other is used to estimate the long term reward (target network).
2. The target network will not be updated after each iteration step, but only be updated with the online network’s weight every 10 steps. It will help prevent over-fitted and stably estimate the next state’s best action.
3. The online network will be updated every iteration step, using the following equation.

Yt ← rt+1 ＋ γ × Q’s’,argmaxQs’,a

where Yt is the ground truth target value that the online network need to fit to, Q is the online network, Q’ is the target network.

1. Training from custom reward function and wait for convergence.

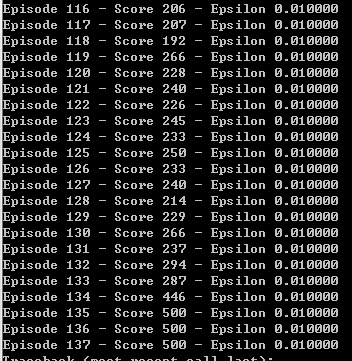


Fig 4. Double Q Network

Frankly, the target network is a stable version or a copy of online network, which is used to estimate the long term reward, while the online network is used to select the best action.

There are also some highlights needed to be pointed out:

1. The Deep Q Network uses a strategy “experience replay” to store the environment samples, and will be fitted over and over again until they are pushed out. Consequently, it help prevent over-fit.
2. I use a customized reward function that it will help converge the training faster, because reward rule do much matters.
3. Obviously, there is no need to compare the optimal policy obtained by Q-learning with random policy.

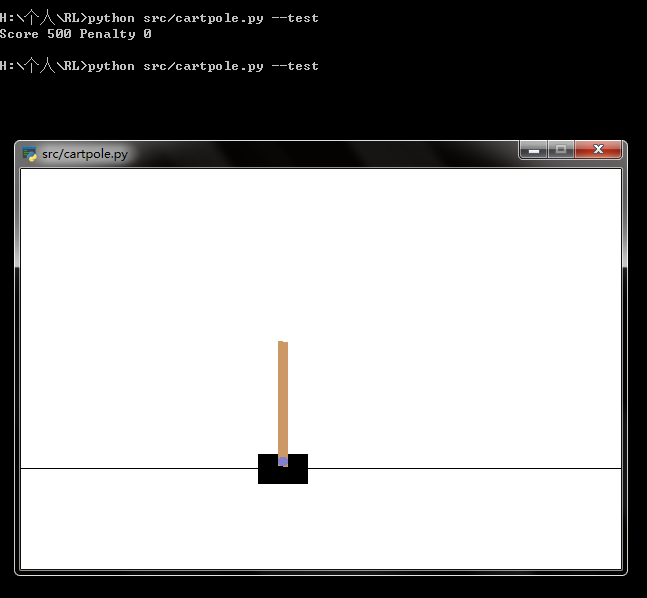


Fig 5. DDQN



Fig 6. random cartpole

**Task 5**

Comparing the Taxi and Cartpole problem, they have some differences that you can find after the aforementioned description:

1. Taxi’s observation space is discrete while Cartpole’s is continuous, which means it hard to use a enormous look-up Q table to represent all the Q values of the Cartpole. Instead, we need some technologies like bucket, boosting, neural network to handle the limit.
2. As inferred from the above evidences, we can know that the cartpole solution converges faster than taxi problem, this is because that the neural network has a powerful ability to fit data. Besides, this is also a disadvantage that the neural network has an attention to over-fit a given data, while a navie look-up table will never over-fit because it’s capability is not too small nor too large.
3. However, the two solutions still need hyper-parameter tuning, which couldn’t be ignored in ML.