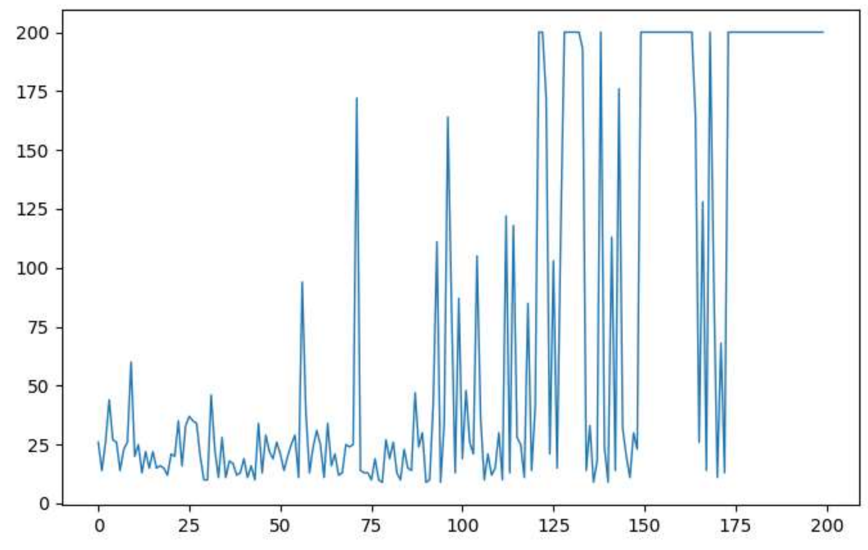
莊晴雯
108062281

Basic-report

In this assignment, what we have to do is to balance the pole on the cart. I tried several number of episodes from 100, 200, up to 300, and found that 200 produces the best result with total rewards of 200. When the number of episodes and its length is set to 100, the total rewards are only 27, and for 300, the total rewards are only 13, and both did not pass the test. Next is to save the Q-table into a .json file. Then, for the choose_action function, we need to choose a random action if the random sample is less than the parameter epsilon, else, we return the argmax of the given state from the Q-table. I tried various values for the n_buckets tuple, such as 9 for the angle of pole, etc., but the best result was produces when the tuple consists of values (1, 1, 6, 3). Since we are initializing the Q-table for each state-action pair, we need to have an array filled with zeros. Then, we need to store the rewards in a list named rewards_record, then for each episodes, we need to get the epsilon value and also the learning rate value, then reset the environment each time and also the rewards to 0, also turn the observation into a discrete state. Then for each episode length, we need to choose an action based on the choose_action() function that we implemented earlier, then do the action and accumulate the reward, and find the maximum next state action value from the Q-table. Next, to find the maximum next state action value from the Q-table, we can use amax and pass the array q_table[next_state]) since we need to find the next state's maximum action value. Then we can update the Q-table by the equation q_table[state + (action,)] += lr * (reward+gamma*q_next_max-q_table[state+(action,)]). Next, we can update to the next state.

I did encounter some difficulties in this assignment. At first, I was confused on why the .json file was not automatically generated, because as far as I remember, Python works fine with generating a .txt file automatically. Hence, I solved this problem by first creating the empty .json file and my program runs perfectly fine afterwards. Second, it took me some time to get the ideal episode number and length. After several tries, I eventually found that the ideal value is 200. I also did some trial and errors in deciding for the values of the n_buckets tuple. For the other parts, I tried to read through the descriptions thoroughly and follow the instructions to complete the code.



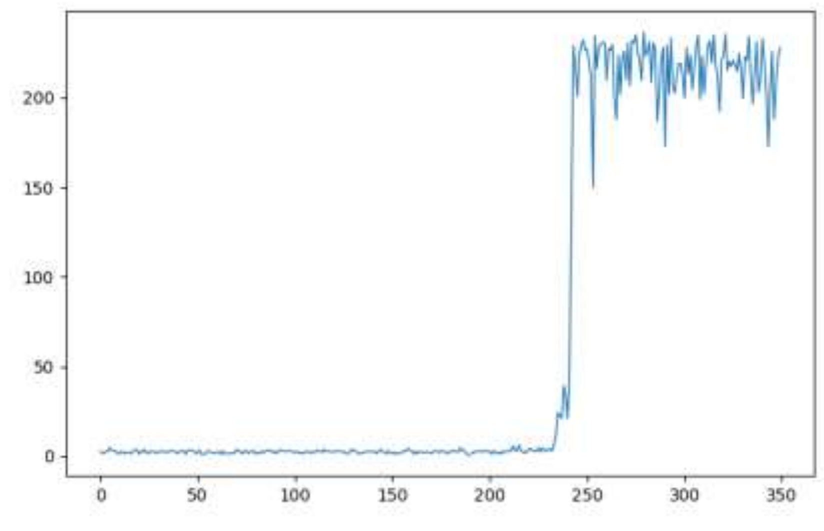
Reward record for Q-table

莊晴雯
108062281

Advanced-report

For the advanced part, I experimented with the episode length and number, and ended up with 350. I did try 200, 400, 500, but they did not produce a better result compared to 350. In the class Net, I just passed in n_states, n_hidden, and n_actions for self.fc1 and self.out. Then for the DQN class, I ended up using n_states*2+2 as the memory slot size, and if I used other values, it yields an error. For the rest of the initialization, I just passed in the input parameters for the properties' values. Similar with the Q-table, we will to choose a random action if the random sample is less than the parameter epsilon, else, we return the argmax of the action_values.data. Next in the store_transition() function, we will use np.hstack function to stack state, [action, reward], and next_state column-wise. Then we get the index of the memory array that we want to place the value transition into, and increment the counter by one. In the elarn() function, the q_target value can be computed by adding b_reward to gamma * q_next.max(1).values.unsqueeze(-1). The lost function can be found from self.loss_func(q_eval, q_target). Determining the hyperparameters values took the most time in finishing this part. I tried different value combinations by changing the number of hidden layers, trying out 32, 64, 100, 128, also changing the batch_size, learning rate, etc. After a lot of trial and errors, the best reward that I could get was when n_hidden = 64, batch_size = 128, lr = 0.0075, epsilon = 0.05, gamma = 0.9, target_replace_iter = 100, and memory_capacity = 2000. I also tried expanding the memory capacity to 3000 but it does not work since I did not pass the dqn test at the end. After creating the DQN, we need to store the rewards in a list named all_rewards. Then, for each episodes, we need to set the rewards to 0, also reset the environment to initial state in each iteration. Then for each episode length, we need to choose an action using the choose_action() function in the DQN class, then do the action and accumulate the reward. We can also choose to use the cheat which is to change the reward value to speed up the training process. We then need to store the experience in memory and accumulate the reward value, then let the agent learn if there is enough memory. Lastly, we just need to change to the next state.

The difficulties that I encountered in this part was to choose the hyper parameters. As I said, I experimented with a lot of values until I ended up with what I have now.



Cartpole DQN action reward