Analyzing Reinforcement Learning: Q-Learning with Cart-Pole

Jacob Berman
CSCI 3202 – Spring 2019
University of Colorado Boulder

Table of Contents

Introduction

The problem that I have decided to solve and analyze is the Cart-Pole balancing problem. This problem that is very similar to the classic inverted pendulum problem. The inverted pendulum problem goal is to essentially adjust the position of some base in order to keep the object upright. This is an unstable system that is usually solved applying control theories however; we will be using reinforcement learning also known as Q-Learning.

Cart-Pole uses the cart as the base and the pole as the object that needs to be kept upright that can move along a frictionless track. The actions that a player can take to control the system are +1 right and -1 left. A reward of +1 is provided for every time step that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.[1]

Methods

To implement and solve this problem I used OpenAI Gym. This is a game simulator environment. It is a resource used for developing, comparing, and analyzing reinforcement learning algorithms. I used the OpenAI Gym environment so simulate the Cart-Pole game. First I will describe starting the game. In order to start the implementation there is a call to the environment function. After that there is a for loop that will run the instance of the game for 1000 time steps, which will render the environment at each step[2]. Next, calling the step function with an action will return observation, reward, and done.

I implemented Q learning in order to solve this problem. In the implementation, we have our reward table from where the agent will learn. Using the reward table it chooses the next action if it's beneficial or not and then they update a new value called Q-Value. The reward table in Q-Learning has columns, which are actions and rows that are states[3]. Using the Q-function that takes two inputs, it will return the expected future rewards.

$$Q(state, action) \leftarrow (1-\alpha)Q(state, action) + \alpha\left(reward + \gamma \max_{a} Q(next\ state, all\ actions)\right)$$

Figure 1

In my implementation, at every time step, I observe the states of the system. These are position (*x*), velocity (*x_dot*), angle (*theta*), and angular velocity (*theta_dot*). In other words, the state-space of the Cart-Pole has four dimensions of continuous values and the action-space has one dimension of two discrete values.[4]

---

[1] https://gym.openai.com/envs/CartPole-v0/

[2] https://gym.openai.com/docs/

[3] https://towardsdatascience.com/reinforcement-learning-with-python-8ef0242a2fa2

[4] https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947

In the OpenAI Gym environment, you will get rewarded as long as the cart is in bounds and the pole is somewhat upright. If either of these two constraints fails then the episode will end. In order to find an optimal policy I used the process of discretizing the input space. I did this because since the world is static the Q-values would converge. This would cause the optimal policy of a given state would result with the largest Q-value[5]. The problem with this is discretizing the input space may lead to convergence not being possible. I have to do some trial and error with the values in order to keep the pole up while also not hurting the agent when it failed.

Results

To get my results I ran the OpenAI Gym Cart-Pole game for 1000 episodes. It takes a few minutes to run and display the results. In order to get the results I did, like I stated above, I had to do trial and error with some parameters. The parameters that I had to modify to find the best results were learning rate, exploration rate, and discount. To get my results, with both exploration and learning rate, I initially set them as 1 and gradually reduced them over time after testing. Although for the reward parameter I kept it at 1. I chose this approach for adjusting my parameters because the goal is to keep the pole upright as much as possible.

I ran the Q-Learning algorithm 10 times so I was able to see how the results varied. Below is the following graph that came from these tests.

	Episodes Ran	# Of Episodes to Solve	Episode 100 Checkpoint (Mean reward)	Episode 200 Checkpoint (Mean reward)
Test 1	244	144	32.87	162.24
Test 2	268	168	38.92	119.38
Test 3	255	155	28.62	143.48
Test 4	223	123	40.83	176.91
Test 5	284	184	37.85	120.57
Test 6	274	174	34.21	177.04
Test 7	230	130	38.05	165.52
Test 8	228	128	41.48	177.52
Test 9	291	191	46.97	146.57
Test 10	246	146	35.5	153.8
Means	254.3	154.3	37.53	154.3

Figure 2

By analyzing the results from this table, after running 10 tests the mean of how many episodes is 254.3. Respectively, the mean of how many episodes it took for the Q-Learning implementation to beat the game 100 times in a row is 154.3. This is why Episodes Ran is

[5] https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947

always 100 more episodes than # Of Episodes to Solve. Based off of these results the implementation is successful and is stable. There may be some edge cases, but it seems for the most part that we can almost always solve the problem.

I also recorded the mean reward for these first 100 episodes and the next 100 after that. The mean for the first 100 means is 37.53 and the next 100 is 154.3.
Below are graphs for the checkpoints to see if there is a trend or not.

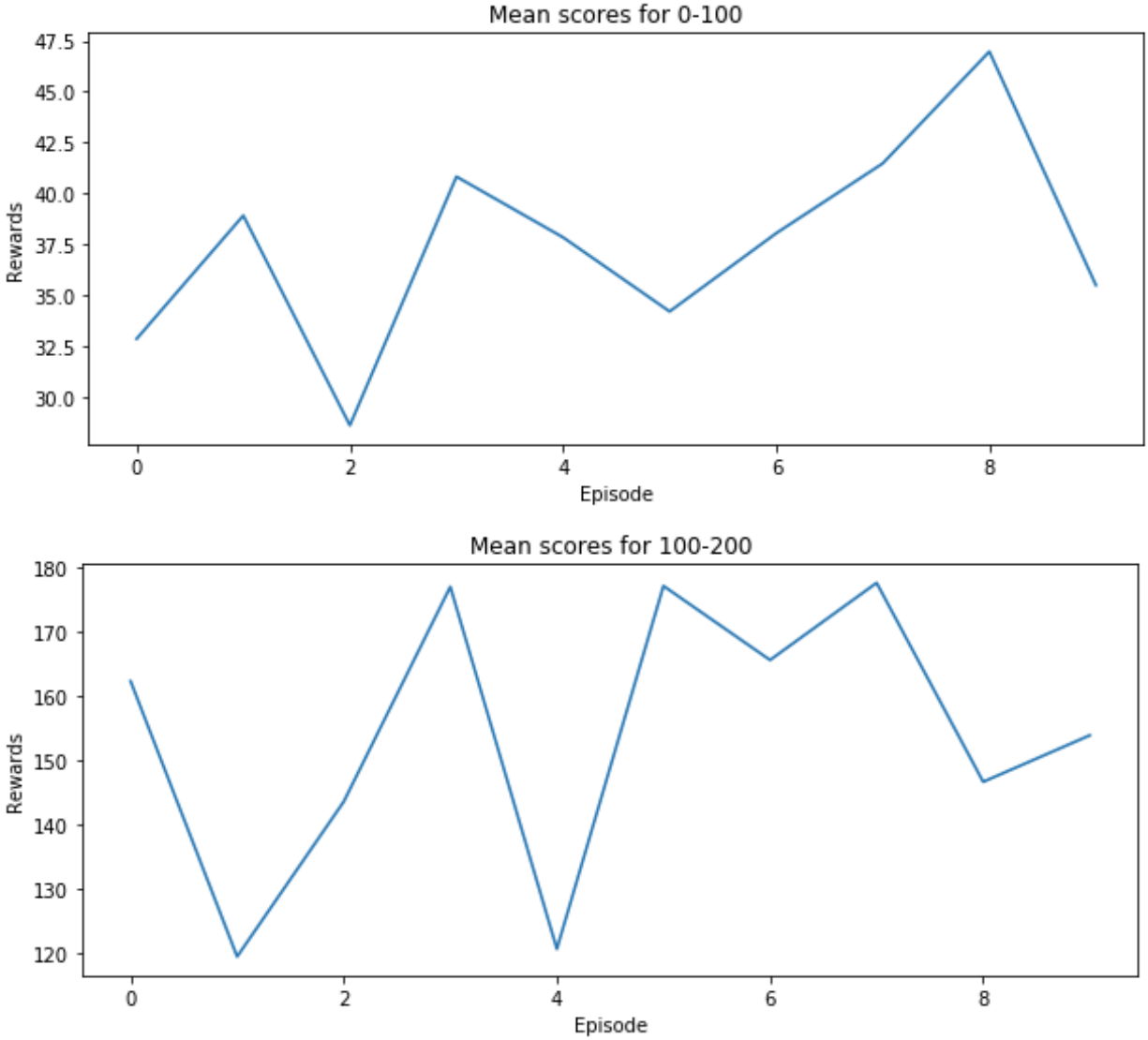

Figure 3

The following graph is showing Rewards vs. Episodes. It is a good model of how the system behaves under Q-Learning.

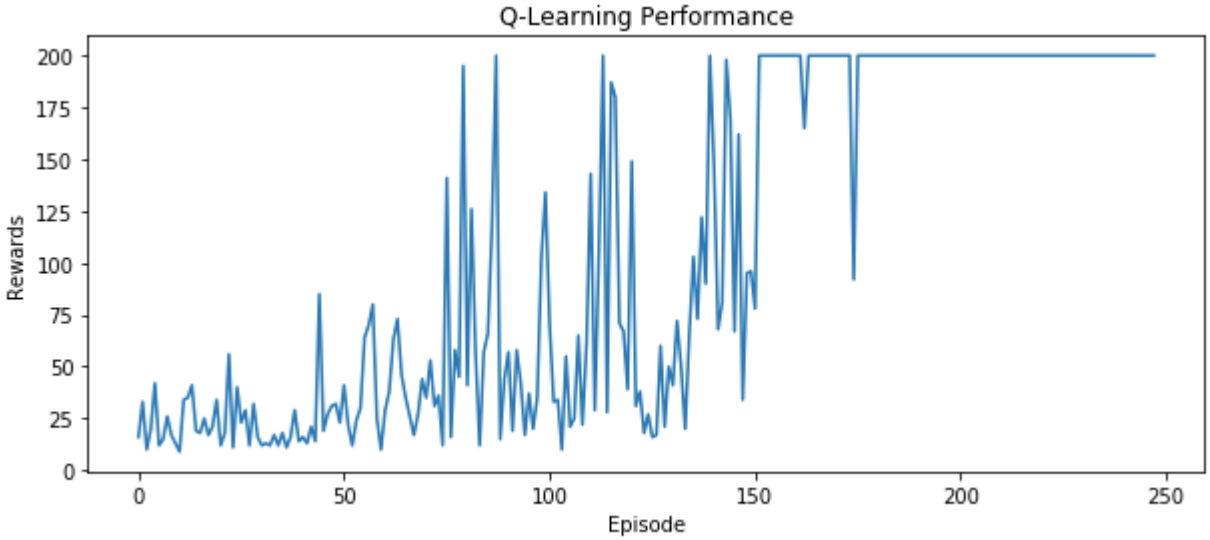

Figure 4

By analyzing figure 4, we can observe that in the beginning the Q-Learning agent does not have many rewards but gradually approves over episodes. It does not make any sudden jumps, which is important to note. It does seem that sometimes the reward can go very high and drop back down however, after ~150 episodes the reward stays high and fairly stable. From viewing the data from the 10 tests it is normal for there to be at least one drop in the reward even after it hits 200. However, from Figure 2 we know that it should always become stable after 200 episodes.

Conclusion

For this final practicum I was able to successfully implement a successful and stable Q-Learning agent for the Cart-Pole game. By analyzing the results we can tell that the implementation preformed well to increase rewards after a consistent number of episodes in each test. There was some variation however the minimum episodes ran was 223 and the max was 291 with a mean of 254.3. These numbers make sense and there are no outliers that help prove that this implementation was successful. It would be interesting to run this model for over 100 or 1000 tests and see what the results would be and to see if the game ever failed.

When researching this topic it seems many others have implanted Deep Q-Learning with the Cart-Pole game. It would be interesting to test these two against each other and see how neural networks would affect the results. It would also be interesting to see how Q-Learning did in other simple games like this and if the results were as successful as mine were.

References

Matthew Chan and Matthew Chan. 2016. Cart-Pole Balancing with Q-Learning. (November 2016). https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947

Vihar Kurama and Vihar Kurama. 2018. Reinforcement Learning with Python. (November 2018). https://towardsdatascience.com/reinforcement-learning-with-python-8ef0242a2fa2

OpenAI. A toolkit for developing and comparing reinforcement learning algorithms. https://gym.openai.com/docs/

Anon. 2019. Q-learning. (April 2019). https://en.wikipedia.org/wiki/Q-learning

Anon. the CartPole-v0 environment. https://gym.openai.com/envs/CartPole-v0/