REINFORCEMENT LEARNING WITH AI - PHASE 2
GROUP PROJECT
CPSC 4371 FALL 2019

Group Members: Sarah Baker, Caleb Lipko, Kiarra Jackson, Eli Taylor
Instructor: Doerschuk

PROBLEM:

The pendulum is an inverted swing-up problem. In this environment, a pendulum starts in a random position swinging back and forth. The goal of the environment is to get the pendulum to swing into an upright position and stay put. Similar to mountain-car the pendulum needs to build up enough strength to reach its goal position. Therefore, we are required to figure out how to use our knowledge in Q-learning in reinforcement learning to get the pendulum to reach goal position.

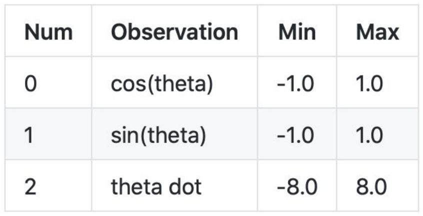
METHODOLOGY:

To solve the pendulum problem we implemented Q-Learning, Q-Learning is an algorithm used in machine learning. Q-Learning searches a predetermined Q-table using a greedy-epsilon algorithm that allows the machine to discover the best action with some variation. Within this problem, Q-Learning is used to learn how much will need to apply to a pendulum to get it to stand upright and to remain in the upright position. As it discovers more effective values it will update the Q-table to reflect better results.

KNOWN DATA:

The Observation State:

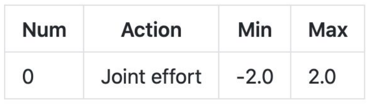
The observation state of the pendulum is an array of 3 values:



Num	Observation	Min	Max
0	cos(theta)	-1.0	1.0
1	sin(theta)	-1.0	1.0
2	theta dot	-8.0	8.0

The Action State:

The action state of the pendulum is one value, which consists of a joint effort between -2.0 and 2.0



Num	Action	Min	Max
0	Joint effort	-2.0	2.0

Reward:

The reward is given when the pendulum remains upright with the least rotational velocity with the least amount of effort. No reward is given when the pendulum fails to stay upright and the program times out after a given amount of episodes. In our case that is 200 episodes.

The Starting State:

The starting state is between -π and π.

<u>SOLUTION:</u>

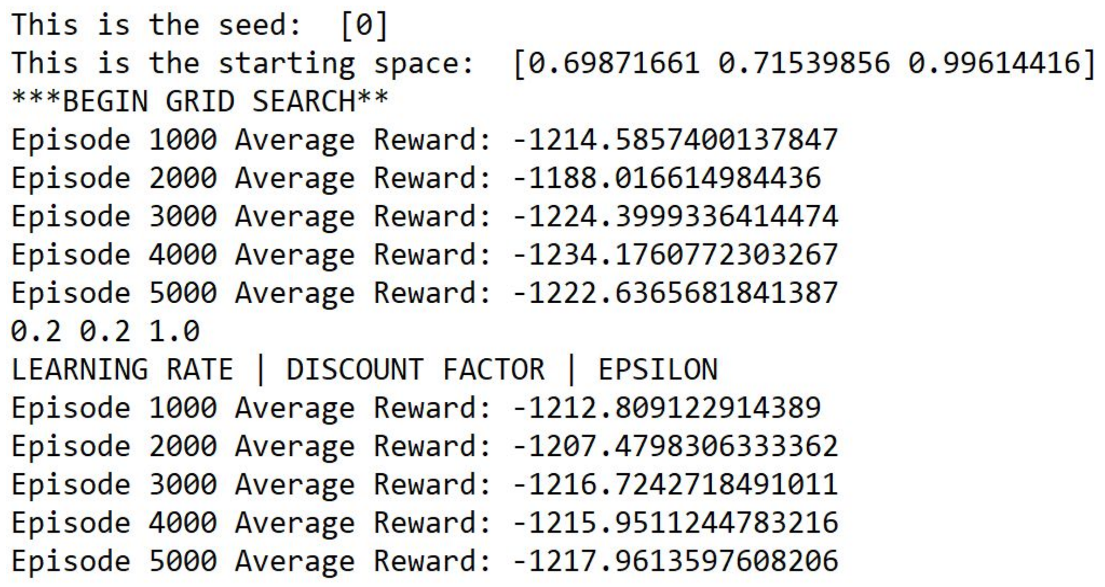
In order to solve the problem of Pendulum using Reinforcement Learning, we used code from the site of Nicolas Mine “Q-Learning”. The program runs 5,000 episodes of the Pendulum maintaining a vertical position with zero angles, with the least rotational velocity, and the least effort. The program returns the average reward per seed along with different hyperparameter combinations. Pendulum-v0 also requires a change in state size by discretizing the state space. The state-space has a different amount of spaces that need to be discretized than with MountainCar. As for the mentioned hyperparameters, we also added onto the code to find which hyperparameters would give us the best outcome of rewards for 3 separate seeds (0, 1, 2) of the environment.

The Code:

The code to the agent of Pendulum-vO is very similar to the code that was used for MountainCar-v0. The differences are that the code for the Pendulum-v0 agent has a different discretization method for discretizing the states, and there is no terminal state to test against. The code follows relatively the same process as the MountainCar agent. The parameters for the Pendulum agent are exactly the same as the MountainCar agent with the exception of env.

<u>DATA RETURNED:</u>

The data returned from the program gives the seed number, the associated starting space, and then the average reward at the end of each 1000th episode up to the 5000th episode. After the return of the average reward, the data includes which hyperparameter was used for that particular set of episodes.



```
This is the seed:  [0]
This is the starting space:  [0.69871661 0.71539856 0.99614416]
***BEGIN GRID SEARCH**
Episode 1000 Average Reward: -1214.5857400137847
Episode 2000 Average Reward: -1188.016614984436
Episode 3000 Average Reward: -1224.3999336414474
Episode 4000 Average Reward: -1234.1760772303267
Episode 5000 Average Reward: -1222.6365681841387
0.2 0.2 1.0
LEARNING RATE | DISCOUNT FACTOR | EPSILON
Episode 1000 Average Reward: -1212.809122914389
Episode 2000 Average Reward: -1207.4798306333362
Episode 3000 Average Reward: -1216.7242718491011
Episode 4000 Average Reward: -1215.9511244783216
Episode 5000 Average Reward: -1217.9613597608206
```

Example of returned data for the first seed (seed: [0])

<u>CONCLUSION:</u>

In conclusion, we had to demonstrate how to use reinforcement learning in a more complex environment than in part 1 of our project. To do that we used the OpenAI Gym Pendulum problem. We had to follow a lot of the same process as we did in part 1 involving Q-Learning and its implementation through discretization. As you can see the rewards are much higher than that of the much simpler Mountain Car problem in part 1. Reinforcement learning is very general, it compasses all problems that involve making a sequence of decisions, the reinforcement learning algorithms have started to achieve good results in many different environments as we have proved through 2 different examples in this project and part 1 with its Mountain Car example. Reinforcement learning has only be slowed by a couple of factors, one being the need for better benchmarking, the other is the lack of standardization of environments used in publications. However, reinforcement learning did its job with the help of Q-Learning and grid searches in our case. From looking at the data and the variance of rewards over the 3 seeds we can conclude that between the 1000-2000th episodes is when it performed the best on average consistently overall the 3 seeds. There was never a dip below -1100 in rewards and never broke -1300 in our testing, so it remained very consistent. We saw that through those things previously mentioned we were able to solve the Pendulum problem by getting the platform to hold the pendulum upright. Again we were able to show that we have completed the first step by learning the basics of OpenAI and how to manipulate Q-learning, Q-tables, random seeds, and grid search.

References:

Code was influenced by Phase 1 and used discretization help from the following website:

http://nicolasmine.com/scripts/solution/Q-Learning.html

Information about state space and rewards:

https://github.com/openai/gym/wiki/Pendulum-v0

Pendulum environment source code:

https://github.com/openai/gym/blob/master/gym/envs/classic_control/pendulum.py