3202 Final Project - OpenAI Gym

By Nicole Lincoln

Github Repository: https://github.com/nikkilinx/3202_Final
Agent Demo: https://youtu.be/A9djaCPoihg

ABOUT:

In this project, I built a Deep Q-Network in order to better understand how the things we have learned in this class are tied together to create an AI agent. I chose to build a DQN because one of the things I struggled to understand while we were doing the AlphaGo paper was how the algorithms we were learning worked with a neural network. After having learned about neural networks, and having more understanding of reinforcement learning, I really wanted to see how the two worked together in a real agent. Building a DQN to solve one of the OpenAI Gym environments seemed like a good opportunity to experiment and solidify my understanding.

APPROACH:

I used the OpenAI Gym LunarLandar-v2 discrete environment for this project. This is a Box2d control environment. The goal is to have a lander land on a landing pad at the coordinates (0,0). There are positive and negative rewards for landing, crashing, coming to a rest, leg contact with the ground, and firing the main engine. Solved is a score of 200 points.

I chose to use a control environment because I wanted to spend more time focusing on building the DQN and tuning the parameters, and I was concerned that a more complicated environment would require more time spent figuring the environment out. The DQN did end up being more complicated than I thought, even though I built a very simple one, so I'm glad I went with the less complex environment.

My Keras model is a Sequential model that has 3 Dense layers -- an input layer, a hidden layer and an output layer. It uses an Adam optimizer with a learning rate of 0.0005. Since this is a DQN, there is also a Replay Buffer for storing and retrieving episodes for training purposes. It also has a target network to stabilize the learning. The target network copies the neural network and uses it to backpropagate through and train the main Q-network.

The most difficult part of the implementation that took the most troubleshooting was actually OpenAI gym to work. I mainly work out of Windows and Jupyter Notebooks, so not being able to render the environment was frustrating. I tried using Google Colab to take advantage of the GPU, however, I could not get the environment to render. I ended up switching over to my Linux system for this project. After getting the programming running successfully, I again tried Google Colab, but could not get the Box2d environment to work. Since I was not able to take advantage of the GPU, testing my program was very slow, and the amount of parameter tuning I was able to do was limited.

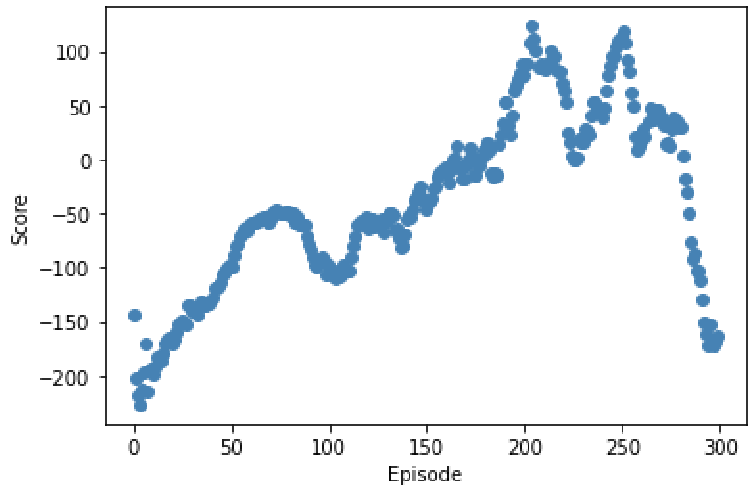
RESULT:

Overall, I was able to run the agent almost 4 full times. The initial run took about 18 hours to do 281 episodes, at which point I stopped the training because it looked like it was over training. The scores went from mostly positive in the 200 range to really low negative numbers (-1266, -975, -873). That run was with the following parameters:

ReplayBuffer length = 10000
Batch Size = 64
Number of Episodes= 500
Learning Rate = 0.001
Gamma = 0.99
Epsilon = 1.0
Size of Dense Layer 1 = 256
Size of Dense Layer 2 = 256

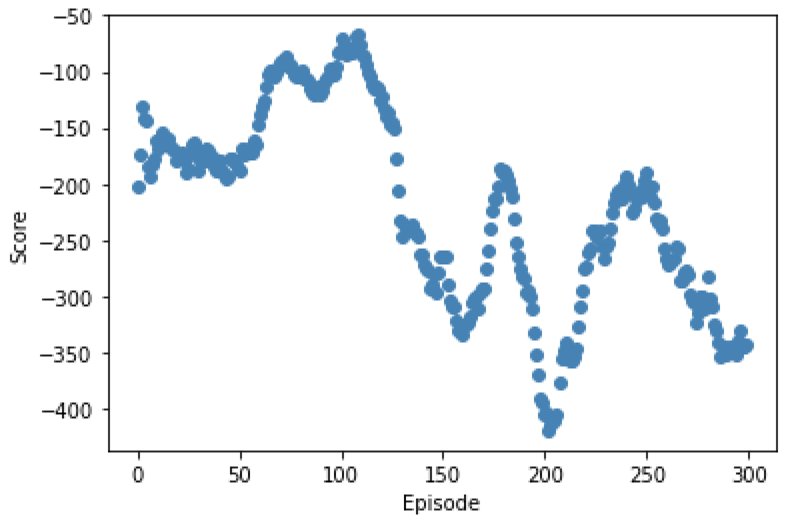
I did not have the coding for graphing the score per episode when I did this run, so I do not have a graph to show progress. Given the time it took (approximately 18 hours), I could not run a failed agent to get a graph.

For the second attempt, I reduced the size of the Dense layers in my model. The lunar landing problem is not extremely complicated with a lot of input information, so I felt like I could reduce the layer size and that would help with overfitting. I also tried reducing the Replay Buffer size and the number of episodes. Since my first try started overfitting after 300 episodes, I tried stopping there. The reduction in ReplayBuffer size was an attempt to get the program to run faster. I wasn't actually timing it, but it still took about 18 hours. This was the result:

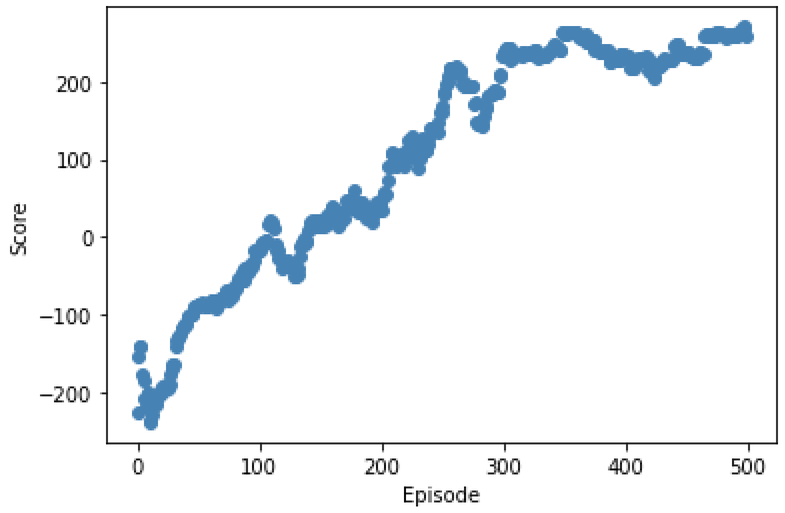


The running after score never reached over 200 and performance dropped significantly after 250 episodes.

The third attempt was a complete failure. I returned the ReplayBuffer size back to 10000 so my agent would have enough episodes to train from. Then I tried adjusting the model by adding dropout layers to reduce overfitting, and I also tried using a softmax activation on the final layer, since this is a multiclass classification function. The result was an agent that completed 3 runs in only 5 hours, but that never had a positive score:



My final, successful test of my algorithm took 19.4 hours to complete. I was able to successfully train the agent to get a score of over 200 consistently after around 300 episodes.



As you can see from the graph above, the running average score is about 200 after 250 episodes, however, it dips back down before leveling out at a score above 200 at episode 300 and beyond. I used a ReplayBuffer max size of 10000, a minibatch size of 64, and Dense layers of size 64. I did not use any dropout layer or activation on the output layer of my model. I reduced the learning rate from 0.001 to 0.0005. My epsilon greedy probability started at 1.0 and decayed at a rate of 0.99, with the minimum epsilon set to 0.01. This allows the agent to start with more exploration and finish with more exploitation. Ultimately, the only difference between this final attempt and my second attempt was the learning rate. I also increased the episodes to 500, but given that the agent was successful after 300 episodes, I do not think that change was significant. The small change in the learning rate prevented the model from overfitting and was successful in consistently getting a score over 200.

CONCLUSION AND REFLECTION:

I thought this project was a really efficient way to provide us with the necessary tools to put together everything we've learned in this class. It was helpful to get the hands on experience of building an actual AI agent, and it was definitely satisfying when it finally succeeded. It was interesting to see how much of a difference the learning rate made in the success of my project. I'm also much more of a hands on learner, so I struggled with understanding the DQN, replay buffer, and target network just from the lectures. Putting it all together in this project was critical in my ability to really learn how the Bellman Equation, Q-learning, and a neural network all work together.

The strongest conclusion I came to from this project was machine learning is very challenging to do without a fast processor. This was a very basic DQN, and not really a complicated project, yet most of my trials took 18-20 hours. It took over 4 days just to make 4 small changes to the hyperparameters. I was hoping to be able to expand my project by generalizing it so that it could be used in multiple environments, but just the testing time for each attempt was prohibitive.

Ultimately, I thought this was a great final project for this class, with enough flexibility to allow me to focus on topics that I was very interested in.

REFERENCES:

Tutorials on using OpenAI Gym and TensorFlow:
https://www.youtube.com/channel/UCUbeqkIVP808fEP0ae-_akw
https://www.youtube.com/watch?v=SMZfgeHFFcA

Additional References:
https://rubikscode.net/2019/07/08/deep-q-learning-with-python-and-tensorflow-2-0/
https://stackoverflow.com/questions/58441514/why-is-tensorflow-2-much-slower-than-tensorflow-1
https://www.katnoria.com/nb_dqn_lunar/
https://towardsdatascience.com/handling-overfitting-in-deep-learning-models-c760ee047c6e
https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c