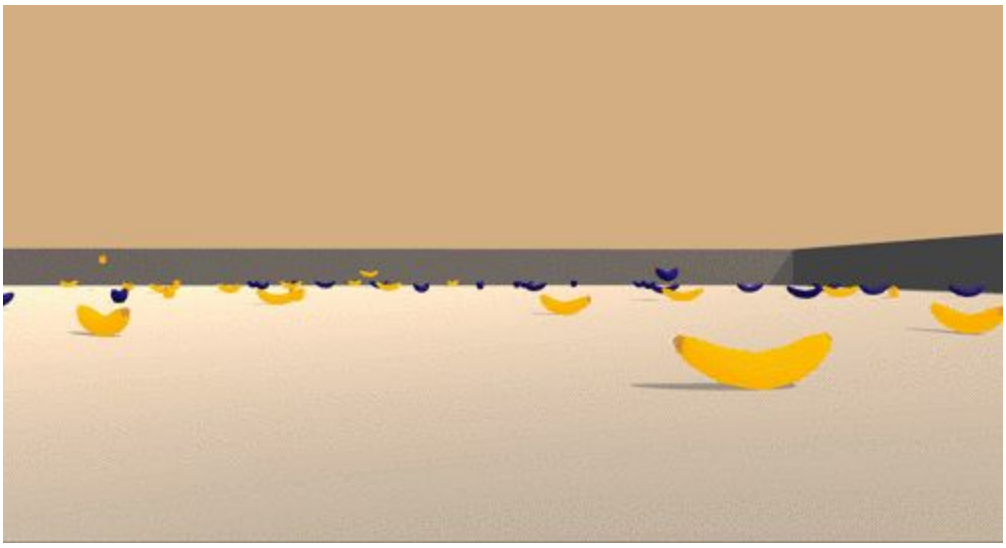
Deep Reinforce Learning - Navigation Project Report

Objective

The Objective of this project is for an agent to navigate in an environment full of bananas. The environment contains blue bananas and yellow bananas, and the agent's goal is to collect as many yellow bananas as possible while avoiding the blue bananas. Whenever the agent collects a yellow banana, the environment will reward the agent with 1 point and when the agent collects a blue banana, the environment will give the agent a -1 point. This type of objective is perfect for deep reinforcement learning, and in this project the agent will be trained by a Deep Q-Networks and Dueling DQN, where the network will take in the environment state as an input and output the best action for the agent to take.



Environment and Agent Exploration

In order to fully understand how the agent interacts with the environment, we should first examine the environment state and the available actions that the agent can take.

In this particular environment, the environment state is a vector with 37 features which represent the agent's velocity and a ray-based perception of objects around the agent's forward direction.

As for the agent, there are 4 discrete actions that are available, which are:
0 - move forward

1 - move backward
2 - turn left
3 - turn right

Models

2 Models will be used for this project and they are DQN and Dueling DQN (which can be found in the jupyter notebook)

Deep Q-Network

A Deep Q-Network is a reinforcement learning technique where given an input state, the network will output Q-values Q(s,a) to tell the agent on when action to take next based on the current environment state. The basic idea here is that given a state of the environment, the state is then fed into a neural network and the neural net will output a Q value of all possible actions of that state.

Dueling DQNetwork

Just like a Deep Q-Network, a Dueling Deep Q-Network also outputs Q-values Q(s,a) for the state action pair. However, the Dueling Deep Q-Network decomposes the Q-values into a Value function V(s) and Advantage function A(a), where V(s) is the value of being in state s and A(a) is the advantage of taking action a in state s. The inspiration of this approach is that the state value doesn't change much across actions (ie: changing lanes when no car is ahead of the agent). However, in some situations, a certain action could give a higher advantage (ie: changing land when there is a car ahead of the agent) and therefore the advantage function is needed to identify the advantages.

Experience Replay

One of the improved methods in training a Deep Q network is called experience replay. When the agent is interacting with the environment, the sequence of experience will be highly correlated. In other words, it is likely that the agent will learn an action value pair that is based on the sequential experience, which in turn can cause the model to diverge or oscillate. Therefore a technique of experience replay will be used in training in order to break the sequence correlation.

Experience replay makes use of a replay buffer, which is the collection of all the experience tuples (S(state), A(action), R(reward), S'(next state)). During training, a small batch of experience tuples will be sampled randomly from the replay buffer and this random sampling will break up any correlation arising from the sequence of experience.

Agent with Fixed Q-Targets

To train this model, we will be using an DQN agent that utilized a technique called Fixed Q-Targets. Normally in Q-Learning, we update a guess based on a guess, and this is troublesome as it can lead to harmful correlation. Therefore, we will utilize 2 Q models in the agents, where we will keep one Q model frozen and update the other Q model. After a predefined step of training, the frozen Q model will be updated with the latest parameters from the updated Q model. And we continue this process until the desired training is completed. The parameter w in the Q model will be updated based on the following equation.

$$\Delta w = \alpha \cdot \overbrace{(\underbrace{R + \gamma \max_a \hat{q}(S', a, w^-)}_{\text{TD target}} - \underbrace{\hat{q}(S, A, w)}_{\text{old value}})}^{\text{TD error}} \nabla_w \hat{q}(S, A, w)$$

Where w- is from the frozen Q model and w is from the local Q model that is constantly updating during the learning phase.

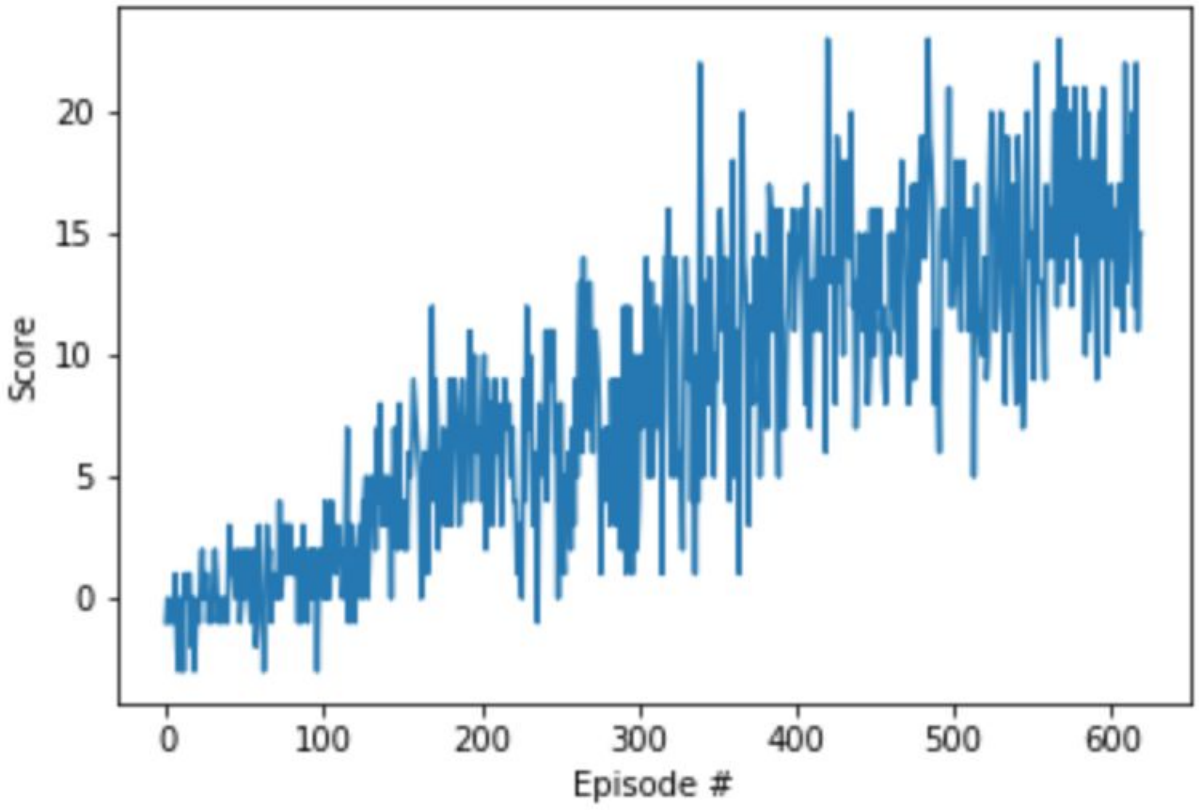
Agent

The agent class in this project served several purposes and they are step, act, learn and soft update.

1. step
    a. Adding the experience tuple into the memory buffer
    b. Keeping track on how many steps the agent has experienced (for update purpose)
2. act
    a. Based on the state input, get the action based on the local Q network
    b. Based on the epsilon greedy algorithm, either return the action or a random action
3. learn
    a. Based of the random batch of experiences, find the next q target
    b. Based on the reward, we find the q target that the model should go toward
    c. Find the current q value as predicted in the local network
    d. Find the difference between the current q value and the q target value
    e. Update q value using the optimizer to minimize the loss
4. Update
    a. Update the weight of the target q model with the weight of the local q model

Result

Using the dueling DQN for training, the model was able to solve the environment in 520 episodes. When comparing the training between dueling DQN and DQN, I found that dueling DQN was able to solve the environment much faster than the DQN. Below is the training graph of the dueling DQN.



Conclusion

This project was a great experience in learning deep Q-network. Traditional reinforcement learning works great when the state space is small but when the state space and action space is large, deep Q-network is a better choice. In this project, there are many parameters that can allow the model to train better (ie: learning rate, update rate, gamma).

To further improve this project, I would want to next try the concept of prioritized experience replay. This concept will allow us to sample experience tuples that contain a higher error rate or tuples that are more rare to come by. In addition, I would like to set up a way in which I can

systematically try out different parameters and see which sets of parameters will allow faster training.

Also, it will be interesting to experiment with different optimizers and different neural network architectures. The opportunity of improvements are endless and that's why deep reinforcement learning is such an interesting topic to me.