Udacity Reinforcement Learning Nanodegree Project 1 - Navigation

Jayanth Nair, August 2020

Project Details:

Objective: Navigate around the Unity Banana Collector Environment, collecting yellow bananas while avoiding blue bananas using Deep Reinforcement Learning.

Reward System:

The reward system is as follows

- +1 for collecting a yellow banana
- -1 for collecting a yellow banana

State Space:

The state space has 37 dimensions which contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction.

Action Space:

The action space has 4 discrete actions which are:

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

Solution Criterion:

The task is considered solved when the agent scores on average +13 over 100 consecutive episodes.

Learning Algorithm

The algorithm chosen to solve this problem is the Double DQN (Deep Q Network) learning algorithm. The Double DQN is an improvement on the basic DQN learning algorithm which overestimates action values under certain conditions. It was published by Hasselt et al. in 2016.

Algorithm steps:

1. Initialize parameters for local and target Q networks with random weights and an empty replay buffer of a specified size
2. Feed the initial state to the local network and use epsilon-greedy action selection to choose an action
3. Execute chosen action in the environment and observe the reward, next state and check if the episode is done
4. Store this transition in a replay buffer
5. Once replay buffer has reached a threshold size, sample a random mini-batch of transitions from the replay buffer
6. For every transition to a new state, use the local network to select an action.
7. Use the selected action and calculate Q value using the target network
8. Calculate the loss function which is the mean squared error between the Q values predicted by local network and the target network
9. Do gradient descent using the selected optimizer to minimize the loss in respect to the local network parameters
10. Every N steps, modify the parameters of the target network by annealing the parameters of the local network
11. Repeat from step 2 until converged

Q Network Architecture:

The selected Q Network architecture consists of 1 input layer (37 neurons corresponding to state size), 4 hidden layers (512, 256, 128 and 64 neurons respectively) and 1 output layer (4 neurons corresponding to the 4 actions). ReLU activation is applied to each hidden layer.

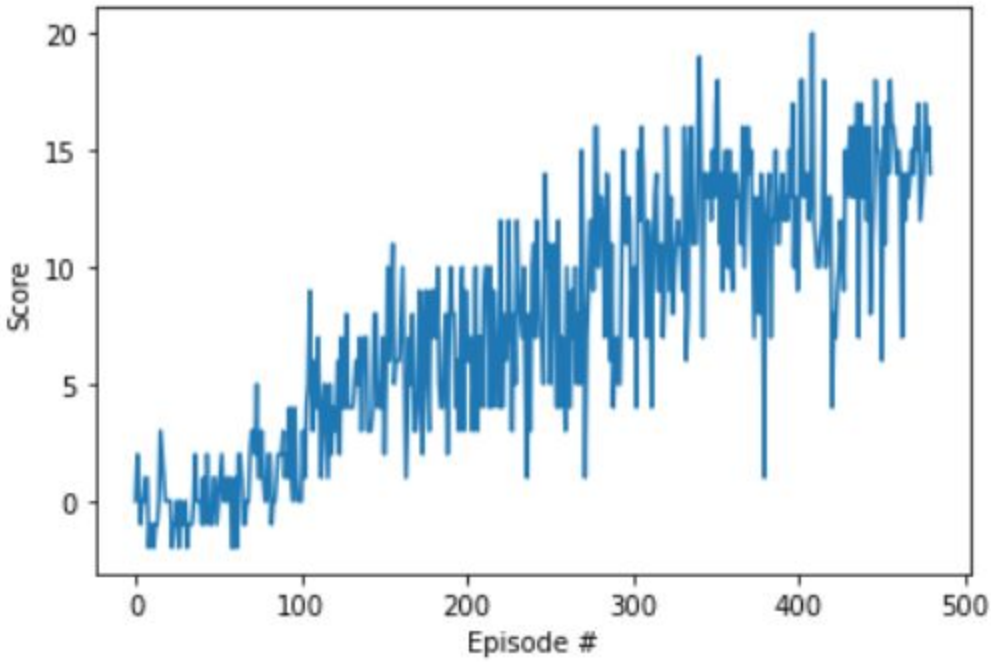
Hyper-parameters:

The hyper-parameters used were as follows:

- Batch-size - 32
- Replay-size - 10,000
- Gamma - 0.99 (Discount factor)
- Tau - 0.005 (Annealing factor for updating local and target network parameters)
- Learning Rate - 0.0005
- Number of stacked frames - 6

Results

The results obtained are shown below (plot of scores vs episodes). The agent obtained an average score per 100 episodes of 13 in 480 episodes.



Score vs Episodes - Training Performance

Next Steps and Room for Improvement

As the score plot shows, in later episodes the agent reached scores as high as 20 but as low as 2. Even though the environment is technically solved, sometimes the agent will fail because of this unstable performance. To improve this some of the following can be implemented:

1. Fine-tuning hyper-parameters. The current set of hyper-parameters were chosen after some rough experimentation. Running experiments to do a grid search of hyper-parameters can result in more stable performance
2. Trying other DQN extensions. Dueling DQN and Prioritized Replay Buffer should improve performance and add stability to the agent.

References:

1. Udacity Course Material, Deep Reinforcement Learning Nanodegree
2. Deep Reinforcement Learning Hands-on by Maxim Lapan