Project 1: Navigation

Timothy Bernard

May 2022

1 Problem and Algorithm Overview

The goal of this project was to use a deep reinforcement learning strategy to train an agent to maximize its cumulative reward by collecting yellow bananas and avoiding blue ones. The large state space and discrete action space made this a good choice for a value-based method like learning with Deep Q-Networks (DQN). The learning strategy is based off of that of Q-learning, an Off-policy TD Control algorithm. With DQN, the idea is to use a deep neural network to approximate the Q (action-value) function, of taking actions in certain states. Similar to Q-learning, it attempts to approximate the optimal action-value function. This network is essentially trained using stochastic gradient descent. In this case, experience replay is used to randomly sample past (s, a, r, s') tuples and perform stochastic gradient descent to minimize the loss between the predicted Q-values and the target (the greedy strategy). Additionally, the targets are produced from a separate network (with an identical architecture) that uses frozen parameters for a set number of steps. These two details prevent learning issues caused by correlated data (less smoothly distributed) and targets coupled with network parameters (update step would be like chasing a "moving target").

2 Implementation

This algorithm was implemented using Python and PyTorch, to train a Deep Q-Network to achieve the goal of +13.0 score across 100 consecutive episodes. It is heavily based off of the DQN Lunar Lander example, and even uses the same hyper-parameters:

Table 1: **Hyper parameters**

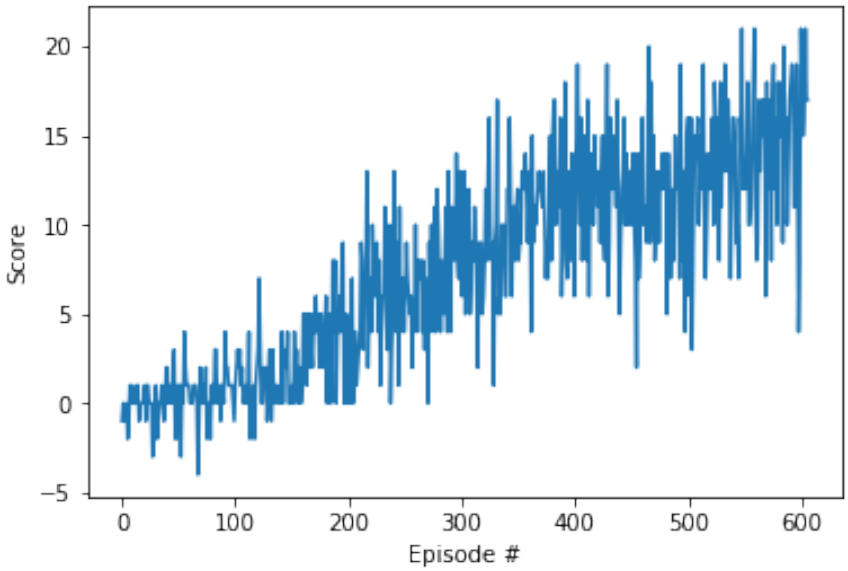
Replay Memory Buffer	100,000
Mini-batch Size	65
Update Rate	4
Tau (for soft update of parameters)	1e-3
Discount Factor	0.99
Learning Rate	5e-4
Max Number of Episodes	2000

The ϵ value (for $\epsilon - greedy$ policy) decayed from 1.0 to 0.01 to make the policy Greedy in the Limit with Infinite Exploration which encourages better convergence to the optimal Q-value.

The network structure is an mlp with two hidden layers, each with 100 nodes. Relu activations are used after each hidden layer, and the input layer and output layer match the state and action space dimensions, respectively.

3 Results

Figure 1: Training Results



The algorithm was able to propel the agent to an average score of 14 over the past 100 episodes in a total of 606 episodes.

4 Future Ideas

One way to improve this algorithm is through prioritized experience replay, which randomly selects more important (s, a, r, s') tuples with greater probability. This probability can be, for example, in proportion to the Temporal Difference error (the error between the action-value prediction and the greedy target). This is intended to quantify how "unexpected" an experience transition is (1). In our case, this could help the agent prioritize learning on unexpected transitions concerning its velocity, and perception of surrounding objects with regards to its action choices. This could be used to speed up training!

References

[1] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015. [Online]. Available: https://arxiv.org/abs/1511.05952