Project 1: Navigation using Deep Q-Networks

In this project, an agent was trained to solve the Unity Environments navigation task to collect as many yellow bananas as possible. The overall agent value Q-network was a simple linear neural network with rectified linear unit activation functions between layers except for the last layer. The input to the neural network was the number of states plus the number of states multiplied by the number of lags to the states. For example, the proposed solution used 4 lags meaning that the input state to the network included the 4 previous states in addition to the current state for 5 sets of states in total (37 x 5). This was flattened to an array with size 185 as input to the linear model. Adding lags to the states helps the neural network consider time information to help guide action selection. Two hidden layers with size of 128 and 64 were used to create a lightweight neural network. Lastly, the output layer mapped the final value approximation to the 4 actions (move forward, backward, left, or right). Two separate neural networks with the same structure were used to train the agent, namely a primary network that updated at every step and a target network that updated after every 4 steps. This provided a fixed Q-target to avoid correlation between rewards and weight updates. Additionally, a memory replay buffer of 100,000 state, action, reward, next state, and done tuples were saved. During training, tuples were sampled from the memory buffer at random for an uncorrelated experience learning.

Over the course of 10 seed iterations, an average of 366 ± 51 episodes were required to achieve the reward goal of 13 during an episode (Figure 1).

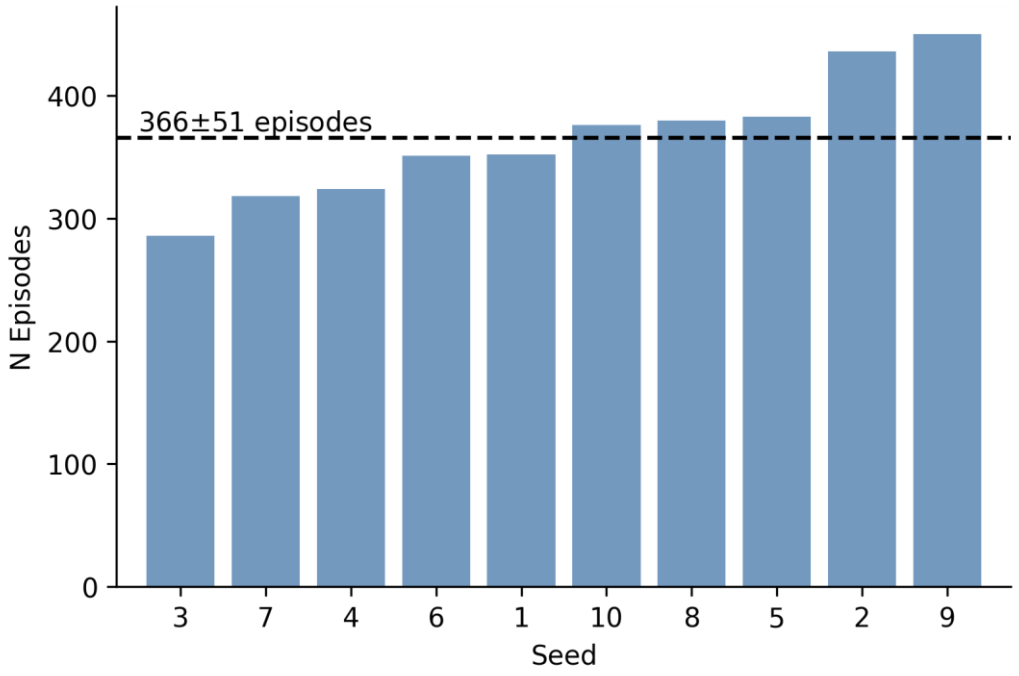

Figure 1: Number of episodes to reach reward goal over 10 seeds

An example of a single agent's training rewards (seed=1) over time is shown in Figure 2. While the score oscillated quite a bit as the agent was learning, there is a noticeable upward trend with a relatively quick convergence to an optimal policy.

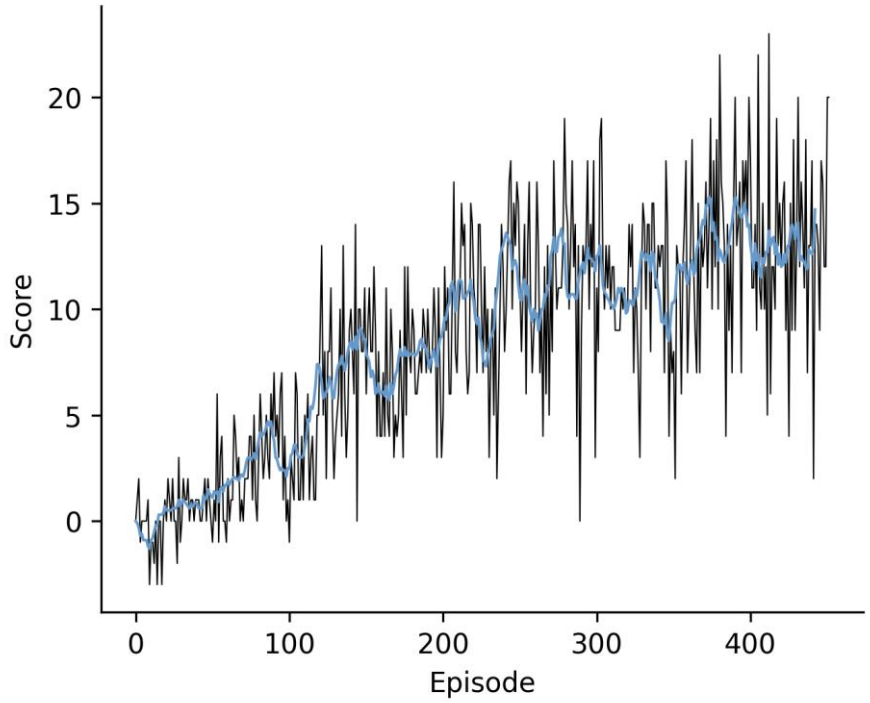

Figure 2: Seed 1 Agent's score throughout training (achieved goal at 352 episodes)

Future ideas would be to consider using prioritized experience replay. Rather than sampling randomly from the buffer, examples which yielded the highest temporal difference (TD) error, indicating greater information could be prioritized. A sampling priority based on TD error magnitude may help the agent's value function approximation to converge faster. Additionally, a neural network that favors time-series information would be preferable, such as a LSTM or transformer. Rather than flattening the state lags to feed into a linear layer, a time-series model may be able to handle this time information better. Despite not using a more complex time-series model to approximate the value function, the linear model does well in this simple task. As the environment becomes more complex, however, a time-series model should be considered. Lastly, a dueling network may help the agent generalize learning across actions. A separate estimator of the value function and advantage function has shown to outperform vanilla DQNs and should be considered for more complex tasks.