Navigation Project

We trained an Agent in an adapted Unity environment to collect yellow bananas while avoiding blue bananas. The agent was trained used Deep Q-learning.

Learning Algorithm

Specifically, a three layer fully connected neural network was used as the Q-network. The input size was a 37-dimensional observation vector. Both the first and the second hidden layers had 64 units while the final output layer had 4 units each corresponding to a different action. Each of the two hidden layers was followed with a ReLU activation function. Given a state as input, the network learned to output the state-action value associated with each action taken from that state. This enabled the agent to rapidly (with a single forward pass) evaluate *Q(s,a)* for all actions that could be taken from the current state *s*.

The size of the hidden layers (64 units) was chosen because it offered good performance while keeping the number of parameters low. A larger network could have been used; however, this may have required longer training times (since more training data would be needed to ensure generalizability). Additionally, smaller networks could be used however performance could be impacted if the network is made to small and we found that a network with two hidden layers each containing 64 units trained quickly enough (about 20 minutes).

The agent is trained with the Deep Q-Learning algorithm. A local DQN is used to estimate the state-action values while a second DQN is used to act as an oracle and provide a target for the loss function. Since the target is dependent on what is being optimized (the Q-network) it is necessary to maintain to Q-networks (one doesn't require gradients).

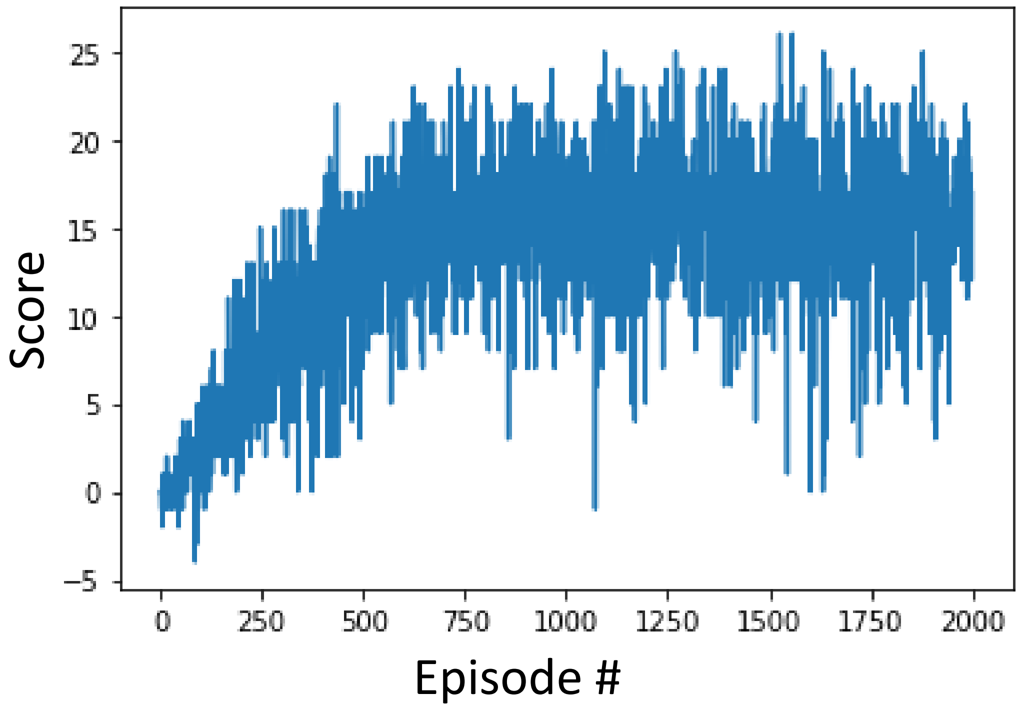
At every timestep the agent plays epsilon-greedy actions. Initially epsilon starts off as one (the agent plays completely random actions) and at the end of each episode epsilon decays by a factor of 0.995. This allows the agent to exhibit highly exploratory behavior for a lot of episodes in the beginning but once the agent has amassed a large amount of experience epsilon will get smaller to allow the agent to exploit its existing knowledge in favor of exploring new actions. Epsilon is never allowed to decay to less than 0.01

Performance

Using this training paradigm, the agent receives an average score of around 15-16 after around 1,000 training episodes (see figure below). For completeness we train for 2,000 episodes but don't see significant improvements after the first 1,000.

Future Ideas

While we see the agent effectively solves this environment and attains good performance, we are excited in future improvements and see three possible ways to improve the agent's learning: incorporating prioritized replay, double Q-learning, and a dueling Deep Q-Network.



Because some experience tuples will be more important for learning while others may not contain much useful information (e.g. collecting a blue banana should be a salient experience that greatly effects learning while walking forward through empty space should not be very salient), sampling tuples in a prioritized way such that more important experiences are revisited more often could result in better training performance. We could achieve this by sampling tuples non-uniformly. The probability of sampling a tuple could be based on the TD-error such that tuples that result in a larger TD-error (and hence are more surprising to the agent may get revisited more often).

Since the local Q-network is used both to pick the action with the highest value and uses this same network to estimate the action's value, there may be an overestimation bias. For instance, consider the case were all actions have zero value, but our Q-network is a noisy estimator. Then if we pick the action with the maximum value, we have biased ourselves to overestimating its value since due to noise the maximum Q-value could be larger than 0. To alleviate this, we could keep two local Q networks. Then, at each training step we could use one to estimate the best action and the second to estimate the value of this best action. Thus if due to noise one Q-network predicted an action's value was greater than 0, then the second Q-network may not if this is just due to estimation noise. At each training step we can alternate between which Q-network is used for determining the best action and which Q-network gets used for determining the value of that action.

Another potential future improvement would be to use a dueling DQN which learns a value for each state and an advantage function that learns how much each action effects the value of the current state. Since many states do not vary significantly across different actions, directly estimating the value of a state may improve the agent’s performance