Navigation: DRLND Project 1 Report

By Joshua Schoenfield

Learning Algorithm

> The report clearly describes the learning algorithm, along with the chosen hyper-parameters. It also describes the model architectures for any neural networks.

The learning algorithm is called **DQN**. This algorithm employs an agent which stores a representation of the Action-Value matrix in the form of **a pair of Deep-Neural-Networks**.

Each `QNetwork` takes a **37 dimensional input**, corresponding to the state and produces a **4 dimensional output**, corresponding to the expected score from taking each action. Between these is a single **64-node hidden layer**. Connecting the 3 sets of nodes are **3 fully connected layers** the first two employing **ReLU activation function**.

Within the DQN agent there is one so-called "Target Network", and another "Local Network". During each **Adams optimizer** learning iteration, **gradient descent is performed on the local network**, using the max expected payoff from the target network run on the following state as a stand-in for the true payoff (A **Max-SARSA** strategy). Then after that learning step the **target network's weights are updated** using a weighted average between the new weights from the local network and the current weights from the target network. The hyperparameter tau describes with what percentage the average is weighted to the local network. The use of two networks serves to reduce numerical instability.

To balance the tradeoff between exploration and exploitation the agent is "annealed" from a strategy of choosing a random action every time (to build experiences within the replay buffer), to a strategy of choosing a random action once out of every 100 times. The probability of choosing randomly is decayed by a factor of .995 every iteration until the minimal state of randomness is achieved. Each pair of (State, Action, Reward, Next State, Done) is stored in the replay buffer. Every 4 time steps the agent takes another learning step with the Adams optimizer using a batch of 64 random (SARSA) pairs.

Hyper-parameters

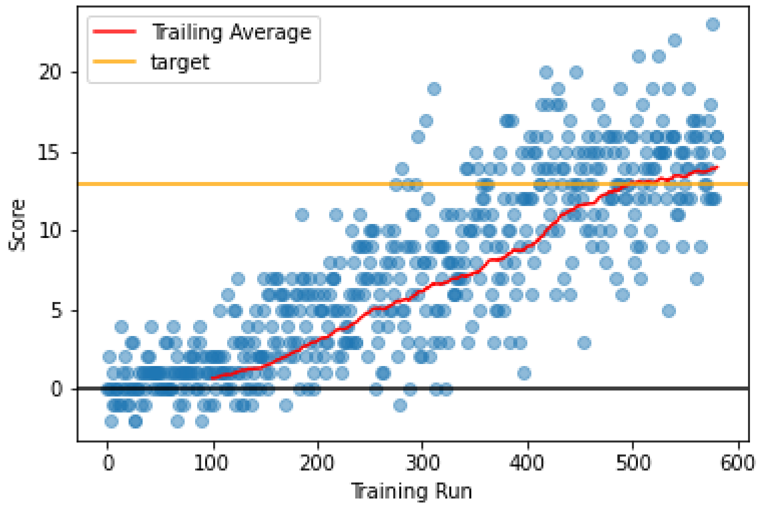
- Network Parameters:
  - Three Layers

        - Input Layer: 37x64 + ReLU
        - FC Layer 1: 64x64 + ReLU
        - Output Layer 2: 64x4
- Agent Parameters:
    - Replay Buffer:
        - Length: 10,000
        - Batch Size: 64
    - Learning Parameters:
        - Optimizer Parameters
            - Adams Optimizer
            - Adams Learning Rate (`LR`): 5e-4
            - Loss: Mean squared error
        - Reward Discount Factor (`GAMMA`): .99
        - Local to Target Momentum (`TAU`): 1e-3
        - Update Every: 4 Time-steps
        - Annealing Parameters:
            - Maximum Epsilon: 1.0
            - Minimum Epsilon: .01
            - Epsilon Decay Factor: .995

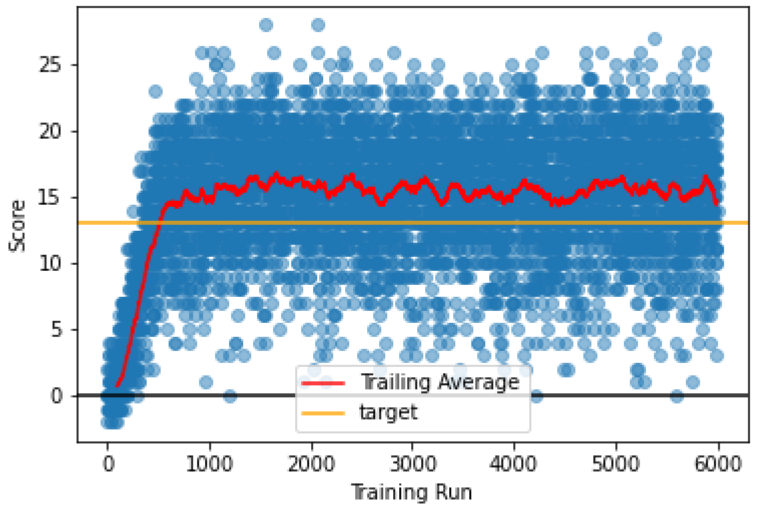
Plot of Rewards

> A plot of rewards per episode is included to illustrate that the agent is able to receive an average reward (over 100 episodes) of at least +13. The submission reports the number of episodes needed to solve the environment.

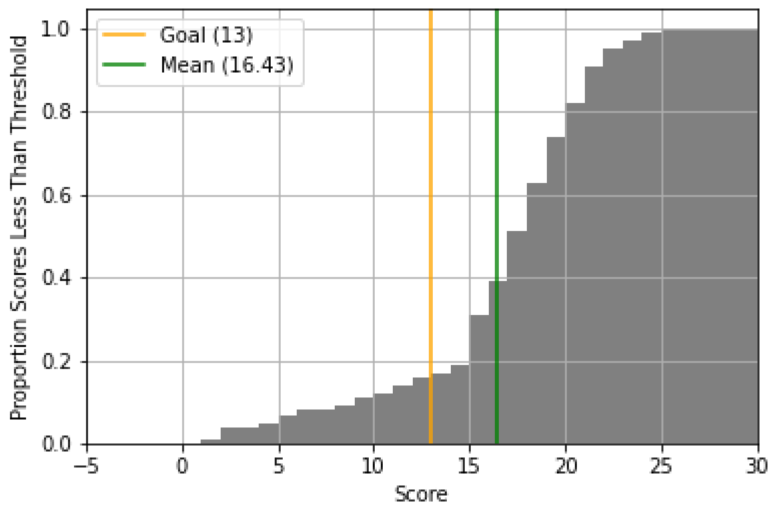
During training, there was a fair amount of variation. A plot of the training scores over time is shown here:



The environment took **482 episodes to solve**. Training for a large number of additional steps did not yield much improvement, as can be seen here:



After completion, the weights were stored. Then an agent acting with no uncertainty was evaluated using the stored weights from the long run. The CDF of the obtained score over 100 runs is shown below.



The agent achieves a **mean score of 16.43** over 100 runs, which is in excess of the threshold for solving.

Ideas for Future Work

> The submission has concrete future ideas for improving the agent's performance.

Some ideas for future improvements could be to include items from the rainbow networks such as:

- **Prioritized Experience Relay**: in which the draws from the experience relay are up-weighted by how much we expect to learn from them and the gradient descent step is scaled inversely to the increased drawing percentage.
- **Dueling DQN**: In this, the learning is decomposed into learning the value of a state and then separately learning the advantage from each action.
- **Learning on Pixels** Since the state space given to us was already parsed based on the true positions of the in-world objects a more realistic challenge could be to learn on the pixels, just as a human would have to.