Navigation

The purpose of the navigation project was to train a Deep Q-learning Network (DQN) agent to navigate an environment collecting yellow bananas while avoiding blue bananas. A standard DQN approach was used to train the agent which could successfully reach the goal of achieving 13 points on the environment after training for ~500-600 episodes (1500-1800 time steps total). Agents were trained using seed 0 for both agent & environment and evaluated for 500 episodes using seed 15 for agent and environment respectively. An example of agent performance during training and evaluation are provided below (95% confidence interval shown in light red).

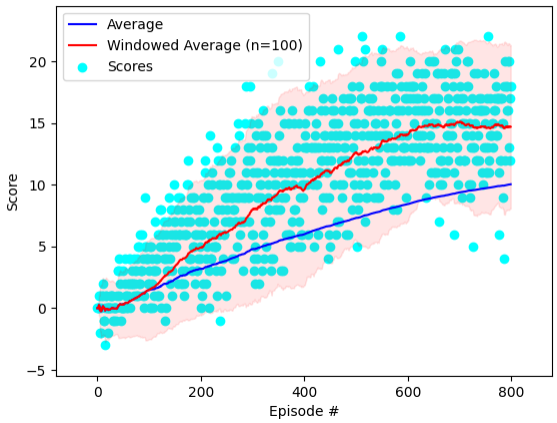

Figure 1 – DQN Training

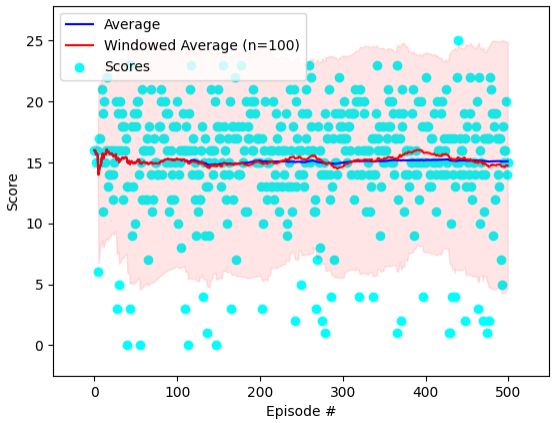

Figure 2 – DQN Evaluation

Deep Q-learning Network (DQN) Agent

The DQN agent used was based off the DQN solution provided earlier in Lesson 2 Deep Q-Networks, Module 6: Coding Exercise. Additional code was added to provide an early stop condition if windowed average performance performed worse than its previous run as well as code to save the best performing model during training. Further details about the agent and associated Q-Network are provided in the sections below.

Q-Network Model Architecture

A straightforward multilayer perceptron approach was used for both DQN agent local and target Q-networks. Surprisingly, a single hidden layer comprised of 64 neurons, was sufficient to learn the navigation task. Rectified Linear Unit (ReLu) activation functions were used for all layers except for final output. A summary of the Q-Network architecture is provided below.

```
QNetwork(
        (input_layer): Linear(in_features=37, out_features=64, bias=True)
        (hidden_layers): ModuleList((0): Linear(in_features=64, out_features=64, bias=True))
        (output_layer): Linear(in_features=64, out_features=4, bias=True)
)
```

Agent Hyperparameters

Default values provided from Lesson 2 Deep Q-Networks, Module 6: Coding Exercise were used for the DQN agent. A summary of agent parameters is provided below and a full listing of recording DQN agent parameters can also be found in the provided model logs (example_*models*/1622254542__best_model/1622254542__*dqn.json*).

Table 1 - DQN Agent Parameters

Batch Size: 64	*Buffer Size: 100000*	*Gamma: 0.99*	*Learn Rate: 0.0005*	*Seed: 0*
Tau: 0.001	*Update Every: 4*	*Q-Network Local*	*Q-Network Target*	
Optimizer: Adam (Parameter Group 0, amsgrad: False, betas: (0.9, 0.999), eps: 1e-08, lr: 0.0005, weight_decay: 0)				

Future Work

There are many improvements that could potentially improve DQN agent performance. Goals would be not only to increase averaged windowed score of DQN agent but also to reduce standard deviation to demonstrate more stable agent performance between episodes (example below).

Episode 100 Average Score: 1.46 ± 2.00
Episode 200 Average Score: 4.92 ± 2.71
Episode 300 Average Score: 7.94 ± 3.80
Episode 400 Average Score: 9.55 ± 3.92
Episode 500 Average Score: 11.79 ± 3.73
Episode 600 Average Score: 14.14 ± 3.84
Episode 700 Average Score: 14.26 ± 3.38
Episode 800 Average Score: 15.31 ± 2.84
Environment solved in 851/2000 episodes! Average Score: 16.16±3.09

A list of potential improvements, grouped by category, are provided below.

Agent Algorithm Modifications

- Implement DQN improvements such as double DQN, dueling DQN, prioritized experience replay, etc., approaching a more Rainbow-like strategy.
- Prioritized experience replay would be first approach as it may be the easiest to implement by sampling with respect to memory error as distribution probability

Model Modifications

- Use a CNN to analyze game pixels as observations instead of using ray trace observations
- Add dropout layers between fully connected MLP layers to prevent overfitting and improve generalizability
- Investigate various architectures with respect to both number of layers and neurons

Hyperparameter Optimization

- Optimize hyperparameters by empirically testing various parameter combinations or using some Bayesian approach
- Use a learning rate scheduler to decrease learning rate throughout training to promote the discovery of better performing policies

Reward Modifications

- Implement additional rewards not provided by Unity Banana environment to penalize actions that don't result in closer distance to a yellow banana
- Reward scheme above may also help to avoid agent shaking in some observed states (agent cycling between same actions)