Report – Navigation

Introduction

The project uses a Deep Q-Network to train an agent to navigate Udacity's version of the Unity ML Banana environment. In the environment, an agent navigates a square world to collect yellow bananas while avoiding blue bananas. Each yellow banana leads to a reward of +1.0, while each blue banana gives a reward of -1.0. The goal is to achieve an average score of at least 13 over 100 consecutive episodes.

The implementation is based on the dqn example in Udacity's Deep Reinforcement Learning Nanodegree program.

Environment

An agent navigates a square world to collect yellow bananas while avoiding blue bananas. Each yellow banana leads to a reward of +1.0, while each blue banana gives a reward of -1.0.

States

The agent observes 37 variables corresponding to its velocity and a ray-based perception of the environment in its forward direction.

Actions

There are four discrete actions: moving forward, moving backward, turning left, and turning right.

Helper functions

The implementation uses a helper function `open_brain_surgery` to reshape outputs from the environment. The function unpacks the next state, rewards, and episode termination information.

Learning Algorithm

The implementation uses different versions of Deep Q-Networks. In addition to the standard DQN agent, it supports agents with double DQN, agents with dueling DQN, and agents that combine double and dueling DQN.

Agent Class

The DDPG agent is implemented in the `Agent` class. In addition to the usual inputs (the size of the state and action spaces and a random seed), the number of hidden layers (at least one) and their neurons for the neural network can be passed to the

`hidden_sizes` parameter. Moreover, the `Agent` class currently supports four types of agents that can be passed using the `flavor` parameter: `plain`, `double`, `dueling`, and `double-dueling`.

Hyperparameters

The hyperparameters for training in all settings were set to the following values:

```
BUFFER_SIZE = int(1e5)

BATCH_SIZE = 64

GAMMA = 0.99

TAU = 1e-2

LR = 5e-4

UPDATE_EVERY = 4
```

The parameters for epsilon exploration are defined in the training function and were set to the following values:

```
eps_start = 1.0

eps_end = 0.01

eps_decay = 0.95
```

Model Architectures

The local and the target networks use the Adam optimizer for the learning rates, and soft-updates are used to update the values of the target networks.

The plain and double DQN (depicted below) consist of two hidden layers with 64 neurons each and a final layer that maps outputs from the hidden layers to action values, all using the RELU activation function.

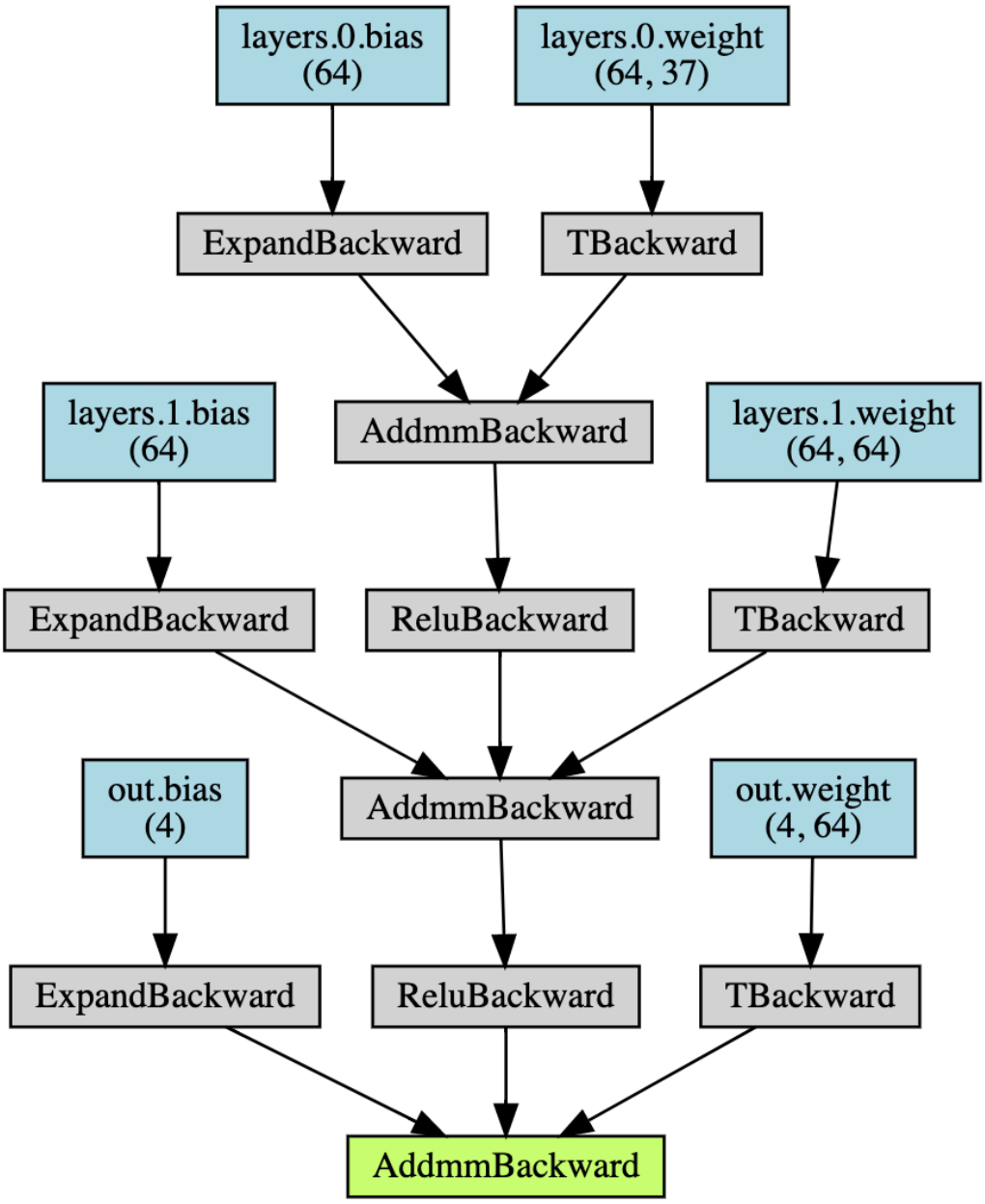

Figure 1: Plain network

The dueling and double-dueling DQN (depicted below) consist of two hidden layers with 64 neurons each, both using the RELU activation function. The network then branches in two final layers for the estimated advantages of each action and the estimated value of the state, again using the RELU activation function.

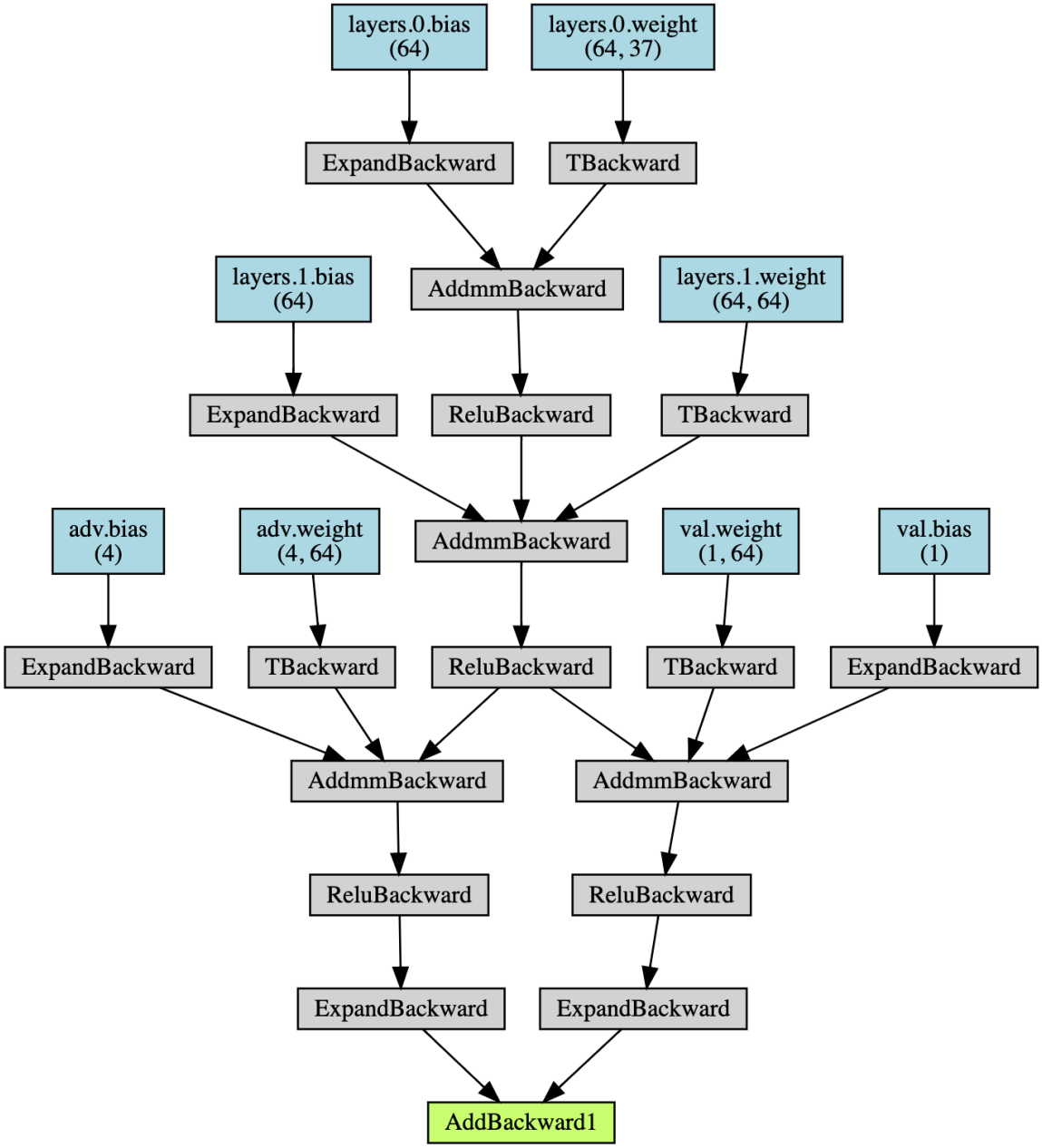

Figure 2: Dueling network

Replay Buffer

The implementation uses a Replay Buffer to store past experiences (tuples of the state, action, reward, next state, and episode termination signal) and use random batches of past experiences to learn.

Noise

Epsilon noise is used to encourage exploration during the initial phase of the training. Epsilon starts at 1.0 (completely random) and then decreases geometrically with decay 0.95 until it reaches a final value of 0.01.

Training

Four types of agents were tested:

1. Standard DQN: `flavor = 'plain'`
2. Double DQN: `flavor = 'double'`
3. Dueling DQN: `flavor = 'dueling'`
4. Double dueling DQN: `flavor = 'double-dueling'`

1. Standard DQN

The algorithm is learning relatively quickly, solving the environment in under 400 episodes. Learning is also relatively reliable (previous runs took 300 to 600 episodes).

```
Environment solved in 364 episodes!     Average Score: 13.06
```

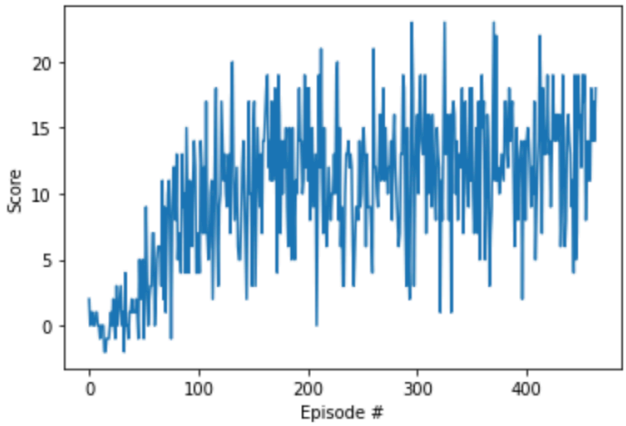

Figure 3: Plot of Rewards

2. Double DQN

The algorithm is slightly faster than the standard DQN, solving the environment in less than 300 episodes. Learning is again relatively reliable.

```
Environment solved in 258 episodes!     Average Score: 13.04
```

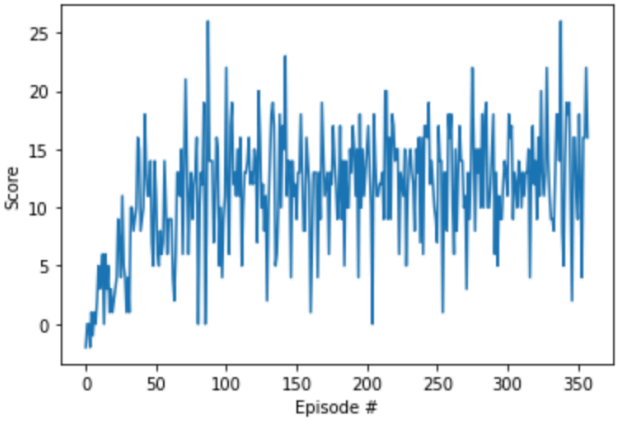

Figure 4: Plot of Rewards

3. Dueling DQN

The algorithm is again slightly faster than the standard DQN, solving the environment in less than 300 episodes. Learning is again relatively reliable.

```
Environment solved in 261 episodes!     Average Score: 13.01
```

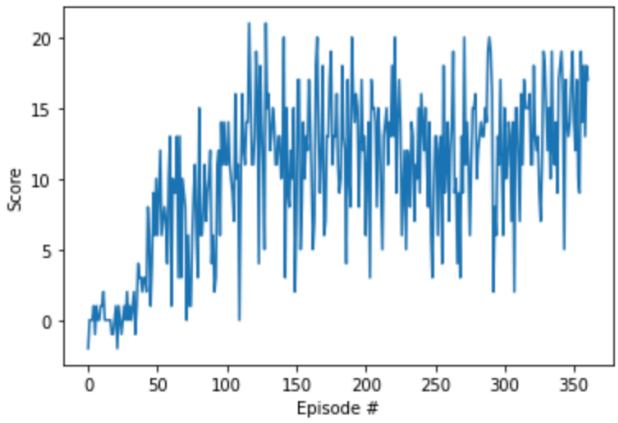

Figure 5: Plot of Rewards

4. Double Dueling DQN

Combing the two improvements yields the fastest training. The environment is solved the environment in under 200 episodes. Learning is again relatively reliable.

```
Environment solved in 180 episodes!     Average Score: 13.00
```

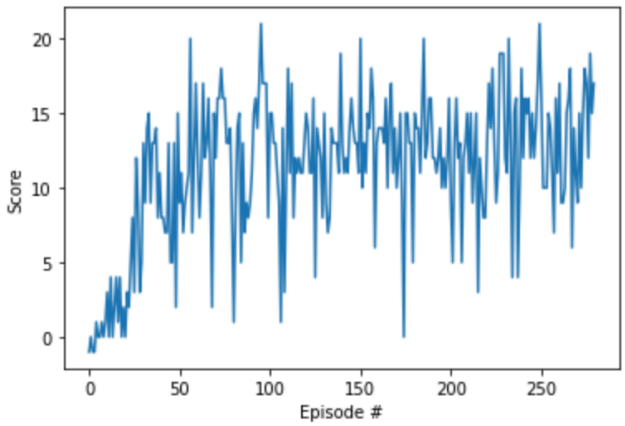

Figure 6: Plot of Rewards

Conclusion and Possible Extensions

Performance and training speed could potentially be increased further:

1. The hyperparameters used are likely not optimal. Additional tuning or grid search could be used to make the training more efficient.

2. The current network architecture is surprisingly effective, given its simplicity. However, more elaborate architectures with additional layers of neurons might yield considerable performance gains.

3. There have been many improvements to the DQN algorithm, some of which have been consolidated in the Rainbow Agent. In addition to Dueling and Double DQN implemented here, the following additional improvements:

    - The ReplayBuffer class could be appended to support prioritized experience replay.

    - Noisy layers in the network could lead to more efficient exploration.

    - Distributional DQN would aim to learn the distribution of rewards instead of the mean. This is presumably less important for this application, as there is relatively little uncertainty within an episode of the environment.

    - The loss function could be changed to support multi-step learning.