Report Project 1: Navigation

Learning Algorithm

I have implemented Double Deep Q-Learning to get faster convergence than Vanilla Deep Q-Learning with experience replay.

For implementing Double Deep Q-Learning I used Von Dollen's article "Investigating Reinforcement Learning Agents for Continuous State Space Environments" (https://arxiv.org/ftp/arxiv/papers/1708/1708.02378.pdf) as a starting point.

I used Von Dollen's recommended neural network architecture of a three layer fully connected network with first hidden layer of 128 nodes and second hidden layer of 256 nodes. Also I used the tanh activation function for connecting the first to the second hidden layer (refer to model.py for code).

To implement Double learning the network for determining the next state's max actions (qnetwork_local) is separated from the network that determines the next state's action values (qnetwork_target) (code snippet from dqn_agent.py):

```
next_actions = self.qnetwork_local(next_states).detach().argmax(1).unsqueeze(1)
Q_targets_next = self.qnetwork_target(next_states).gather(1, next_actions)
```

After every UPDATE_TARGET_NETWORK_EVERY steps (set to 1200 steps, following Von Dollen’s recommendation) both networks are synchronized (code snippet from dqn_agent.py):

```
if (self.t_step % UPDATE_TARGET_NETWORK_EVERY) == 0:
  self.update(self.qnetwork_local, self.qnetwork_target)
```

When training the agent uses an epsilon greedy policy with epsilon decreasing as (code snippet from dqn_agent.py):

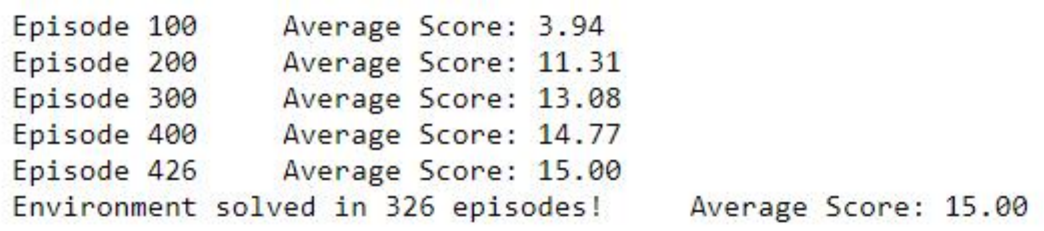
```
eps = LAMBDA/math.sqrt(self.t_step + 1)
```

I have tested the implementation first on OpenAI Gym’s LunarLander-v2 to see if I could replicate Von Dollen's results. After the results on OpenAI Gym’s LunarLander-v2 were satisfactory then I applied the same solution to the Banana Navigation project. Then I had to decrease the optimizer's learning rate from 2e-3 to 2e-4 to get stable convergence.

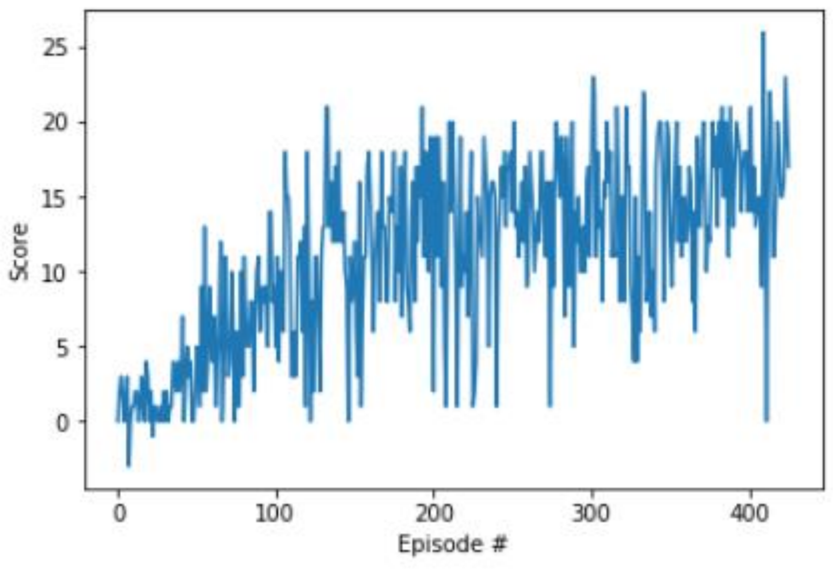
The following table lists all the hyper parameter values.

Hyper Parameter	**Value**	**Description**
BUFFER_SIZE	120000	Replay buffer size
BATCH_SIZE	64	Minibatch size
GAMMA	0.99	Discount factor
UPDATE_TARGET_NETWORK_EVERY	1200	How often the target network will be updated with local network
LR	2e-4	Learning rate for Optimizer
UPDATE_EVERY	3	How often to update the local network
LAMBDA	0.3	For calculating eps = LAMBDA/sqrt(t)

Plot of Rewards



```
Episode 100     Average Score: 3.94
Episode 200     Average Score: 11.31
Episode 300     Average Score: 13.08
Episode 400     Average Score: 14.77
Episode 426     Average Score: 15.00
Environment solved in 326 episodes!     Average Score: 15.00
```



The plot shows fast convergence to the target score of 15.0 in 326 episodes.

Ideas for Future Work

To further improve the algorithm the following DQN improvement could be implemented as well:

- DQN-Prioritized Replay: to use prioritization on the experience replay to learn more effectively