Project 1: Navigation

---

1 Introduction

For this project, I have trained an Agent using the Deep Q Network algorithm with Experience Replay. I have trained an agent to navigate (and collect bananas!) in a large, square world. A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of the agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and the agent can take four actions

- 0 - move forward.
- 1 - move backward.
- 2 - turn left.
- 3 - turn right.

This game ends when the mean score across the last 100 episodes is greater than 13.

2 Dependencies

1. Download the environment from one of the links below. You need only select the environment that matches your operating system:
   - Linux: click here
   - Mac OSX: click here
   - Windows (32-bit): click here
   - Windows (64-bit): click here
2. (*For Windows users*) Check out this link if you need help with determining if your computer is running a 32-bit version or 64-bit version of the Windows operating system.
   (*For AWS*) If you'd like to train the agent on AWS (and have not enabled a virtual screen), then please use this link to obtain the environment.
3. Place the file “Navigation Project Solution v2.ipynb” in the folder, and unzip (or decompress) the file.
   env = UnityEnvironment(file_name="/data/Banana_Linux_NoVis/Banana.x86_64")

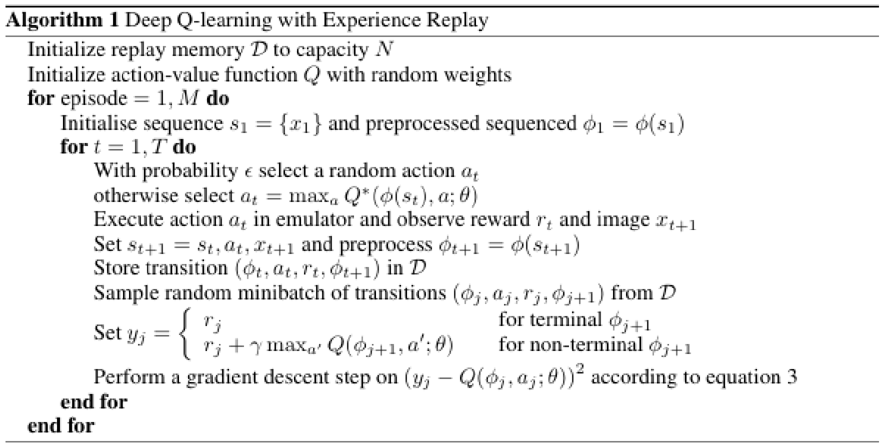
3 How to Execute this solution

Follow the solution in the file **Navigation Project Solution v2.ipynb** to train the agent to pick up the Yellow bananas.

4 Implementation Approach

The Deep Q Learning algorithm with Experience Replay is implemented. The details of the algorithm are given in this research paper. The goal is to find the optimal action-value function that maximises the cumulative reward. In this case I used a Neural Network in place of Q table Q-table. The agent uses two Q-Networks to solve the environment, a target and a local network.

https://arxiv.org/abs/1312.5602



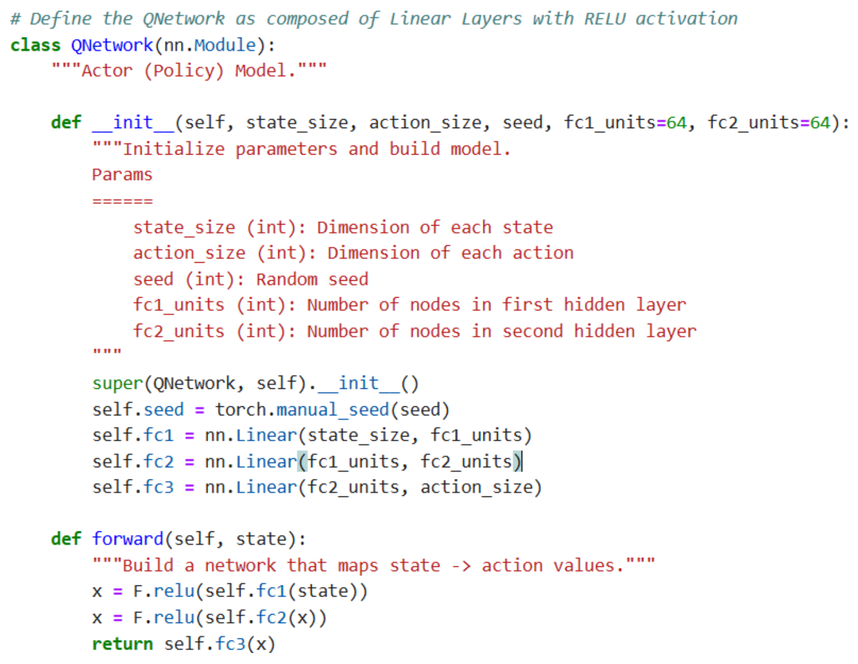
Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

The following steps were taken to implement the algorithm

4.1 Define the Q Network

Define a Neural Network with 37 neurons in the input layer and 4 in the output layer corresponding to the state space and the action space of the environment. The neural network consists of a hidden layer comprising of 64 neurons.



```python
# Define the QNetwork as composed of Linear Layers with RELU activation
class QNetwork(nn.Module):
    """Actor (Policy) Model."""

    def __init__(self, state_size, action_size, seed, fc1_units=64, fc2_units=64):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            seed (int): Random seed
            fc1_units (int): Number of nodes in first hidden layer
            fc2_units (int): Number of nodes in second hidden layer
        """
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, action_size)

    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

4.2 Define the class for Experience Replay

I used the example given in the class to create Experience Replay which is the Replay Buffer for state transitions along with actions, rewards.

4.3 Define the Agent

I defined the agent to implement the Deep Q Network algorithm with Experience Replay. The agent is trained for approximately 377 episodes. By this time, the mean score across last hundred episodes is more than 13 and the game terminates.

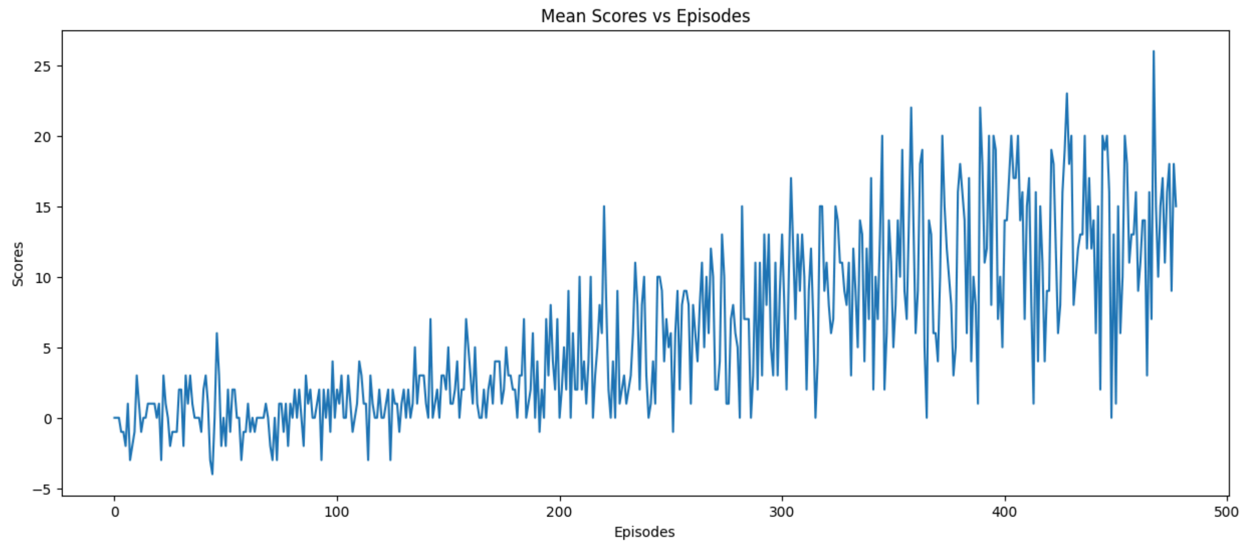
The following hyperparameters were considered for the DQN Agent training.

BUFFER_SIZE = int(1e5) # replay buffer size
BATCH_SIZE = 64   # minibatch size
GAMMA = 0.99   # discount factor
TAU = 1e-3   # for soft update of target parameters
LR = 5e-4   # learning rate
UPDATE_EVERY = 4   # how often to update the network

For the values for Epsilon I adopted an Epsilon Decay approach where the Epsilon starts with value 1.0 and then gets reduced by a decay factor of 0.995. The minimum value for Epsilon is 0.01.
eps_min = 0.01
eps_start = 1.0
eps_decay = 0.995

4.4 Plot the Scores

Using the Matplotlib library, I plotted the scores. The score curve looks as below



4.5 Evaluate the agent for 1 episode

I loaded the weights for the agent from the saved file ‘dqn.pth’ and played one episode of collecting Bananas. The agent was able to collect a score of 14 after one episode.

5 Solution Details

1. Navigation Project Solution v2.ipynb : The main file that contains the Q Network definition, Experience Replay, Agent and training functions.
2. dqn.pth - Trained weights saved after the agent was able to get an average score of 100 in the last 100 episodes.
3. Readme.md
4. Report.pdf

6 Ideas for Future Work

I would like to continue working on this project by implementing the Double DQN, Duelling DQN and the Prioritized Experience Replay algorithms to improve the performance.