Udacity Navigation Project

(Reinforcement Learning Nanodegree)

For this project, an agent is trained 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 agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:
0 - move forward.
1 - move backward.
2 - turn left.
3 - turn right.

The task is episodic.
Target: Agent have to get an average score of +13 over 100 consecutive episodes.

Agent Implementation

This project is solved using Deep Reinforcement Learning using Deep Q-Network.

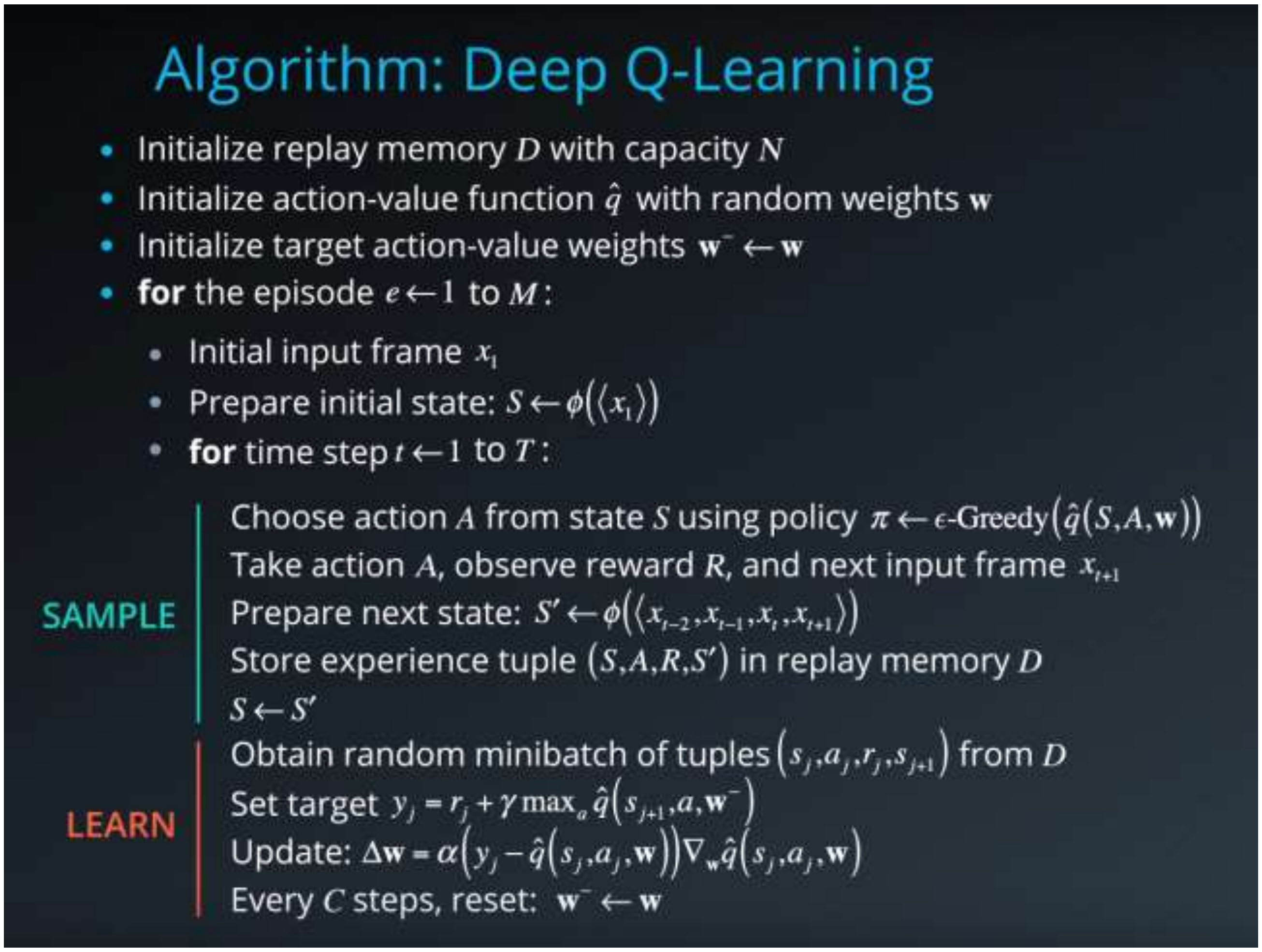
1. The notebook Navigation.ipynb contains the implementation for training and visualising the untrained agent initially. Then the training code is implemented and at the end we can visualize the Trained agent working on the provided environment.
2. dqn_agent.py contains the code to understand and determine how the agent interacts with the environment and learns to optimize the reward.
3. model.py contains the architecture of the deep learning model used in this implementation.

Algorithm

- Used DQN - Deep Q Network Algorithm

Q-learning is a model-free reinforcement learning algorithm. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. Q-Learning is an Off-Policy algorithm for Temporal Difference learning.

<u>Algorithm: Deep Q-Learning</u>



Algorithm: Deep Q-Learning

```
• Initialize replay memory $D$ with capacity $N$
• Initialize action-value function $\hat{q}$ with random weights $\mathbf{w}$
• Initialize target action-value weights $\mathbf{w}^- \leftarrow \mathbf{w}$
• for the episode $e \leftarrow 1$ to $M$:
    • Initial input frame $x_1$
    • Prepare initial state: $S \leftarrow \phi(\langle x_1 \rangle)$
    • for time step $t \leftarrow 1$ to $T$:
SAMPLE  | Choose action $A$ from state $S$ using policy $\pi \leftarrow \epsilon\text{-Greedy}(\hat{q}(S,A,\mathbf{w}))$
        | Take action $A$, observe reward $R$, and next input frame $x_{t+1}$
        | Prepare next state: $S' \leftarrow \phi(\langle x_{t-2}, x_{t-1}, x_t, x_{t+1} \rangle)$
        | Store experience tuple $(S,A,R,S')$ in replay memory $D$
        | $S \leftarrow S'$
LEARN   | Obtain random minibatch of tuples $(s_j, a_j, r_j, s_{j+1})$ from $D$
        | Set target $y_j = r_j + \gamma \max_a \hat{q}(s_{j+1}, a, \mathbf{w}^-)$
        | Update: $\Delta\mathbf{w} = \alpha(y_j - \hat{q}(s_j, a_j, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(s_j, a_j, \mathbf{w})$
        | Every $C$ steps, reset: $\mathbf{w}^- \leftarrow \mathbf{w}$
```

(above taken from udacity tutorials)

Code implementation

The code used here is derived from the "Lunar Lander" tutorial from the Deep Reinforcement Learning Nanodegree, and has been slightly adjusted for being used with the banana environment.

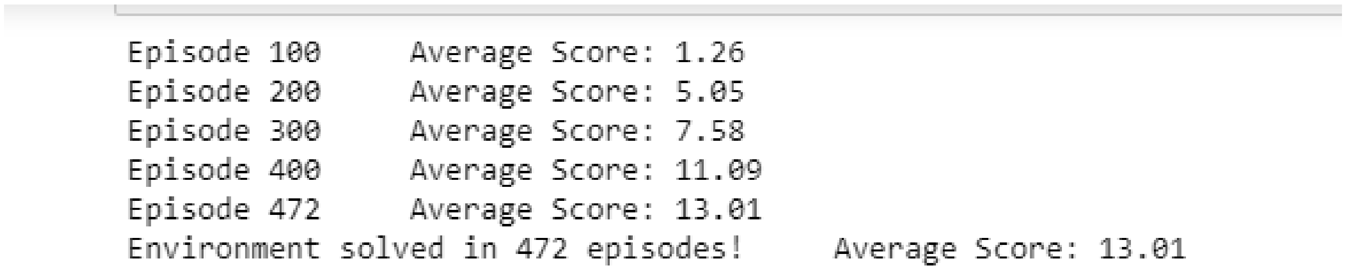
The code consist of:

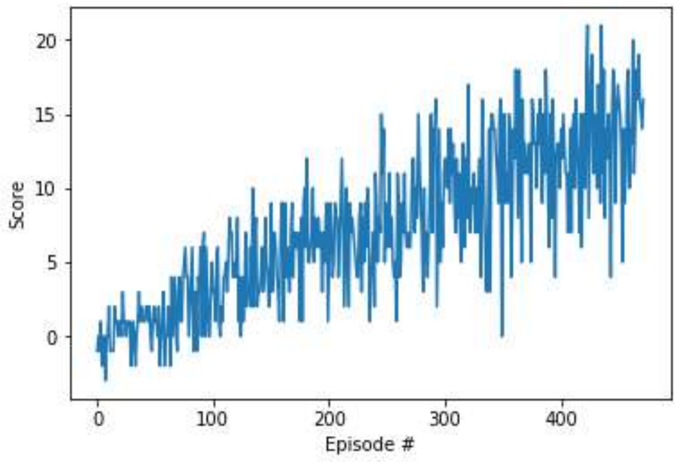
- model.py
    - The model has 2 fully connected layers with 64 nodes.
    - Relu activation function is used betweeen the two layers
    - Adam optimizer
    - Input size is equal to state size = 37
- dqn_agent.py
    - In this python file, a DQN agent and a Replay Buffer memory used by the DQN agent are defined.
    - Methods implemented
        - Step( ) – Storing step taken by agent
        - Act( ) - To return actions for the given state as per current policy
        - Learn( ) – To update the Neural Network value parameters using given batch of experiences
- Navigation.ipynb
    - Start the environment
    - Examine the State and Action Spaces
    - Take Random Actions in the Environment (No display)
    - Train an agent using DQN
    - Plot the scores

Hyperparameters

```
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
```

Result


```
Episode 100     Average Score: 1.26
Episode 200     Average Score: 5.05
Episode 300     Average Score: 7.58
Episode 400     Average Score: 11.09
Episode 472     Average Score: 13.01
Environment solved in 472 episodes!     Average Score: 13.01
```



Ideas for Future Work

The Reinforcement Learning agent was trained using Deep Q network using Experience Replay and Fixed Q targets

- Using existing method, the hyperparameters can be tuned more to get a faster response and higher rewards.
- We can use Double DQN, Prioritized Experience Replay or Duelling DQN