## Introduction

This report provides a brief introduction to project name ‘Navigation’. We trained an agent to navigate through a large and square environment with an objective to collect maximum yellow banana. There are **4** possible actions available to agent i.e., walk forward, walk backward, turn left, and turn right. The state space consists of **37** dimensions. There is **+1** reward of collecting a yellow banana and **-1** for a blue banana. By default, the agent interacts with an environment for maximum of **300** time steps in a single episode.

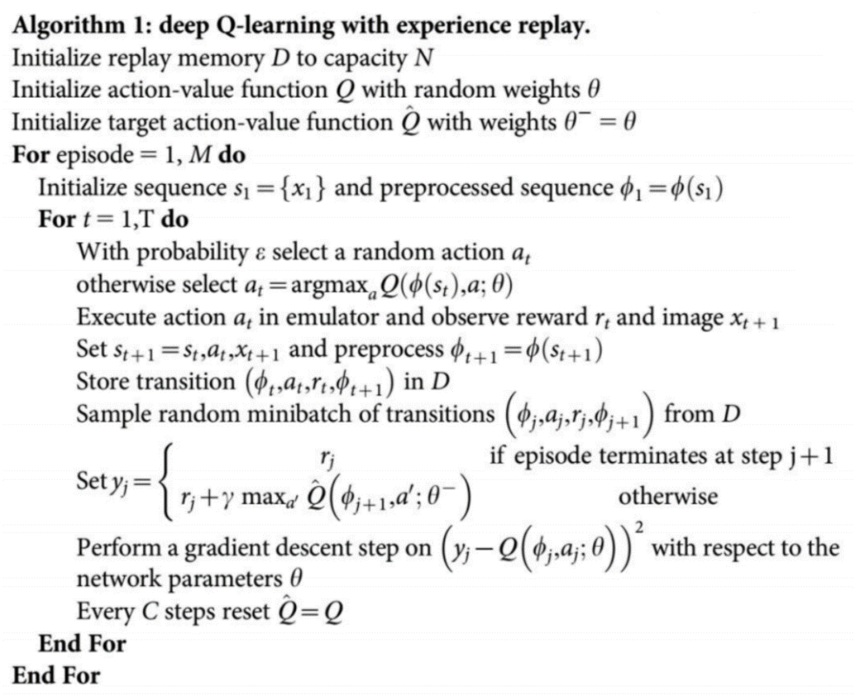
We used off-policy algorithm, i.e., deep Q network where neural network is used for function approximation to obtain an optimal navigation policy. In the following we provide a brief overview of implemented algorithms, the agents training performance and few pointers for future work.

## Implemented algorithms

### Deep Q Network(DQN)

The DQN is proposed by DeepMind which laid the foundation of deep reinforcement learning. The algorithm enhanced the classic RL algorithms called Q-learning with neural networks and techniques like experience replay and fixed-Q values. These additional techniques stabilize the use of non-linear function approximation (deep neural network) to approximate optimal action-value function.

Following is the DQN algorithm with experience replay:



```
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, $M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t$ = 1,T do
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

*Figure 1: DQN Training Algorithm (Source)*

## Extensions to DQN:

Several improvements are made to DQN. We have implemented Double-DQN and Dueling DQN in the project.

**Double DQN:** Deep Q learning show overestimation of Q-values. Vanilla DQN maintains two set of weights for the periodic update. These weights can be used to limit the overestimation of Q-values especially in the start of learning when we may not have enough information to find the best action.   The DQN is tend to perform better, if we let local network to greedy select the best value (as $\underset{a'}{\operatorname{argmax}} Q(s',\ a';\ \theta)$) and let the target network to evaluate the best action ($as\ \theta_i^-$).

*Equation 1: Double DQN loss (Source)*

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\ D}\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta_i^-) - Q(s, a; \theta_i)\right)^2$$

The rest of loss-computation remains same as for DQN. Double DQN has shown performance improvements.

**Dueling DQN:** Dueling DQN is another improvement of vanilla DQN. Typical DQN consist of sequence of convolution layers, followed by fully connected layers to map states to actions (Q values). In Dueling DQN, the idea is to use two stream networks in which one stream estimates state-value function while the other stream evaluates advantage of each action. The final Q values are than obtained by combining the state-value and advantage values. Figure 2 shows the dueling network architecture.

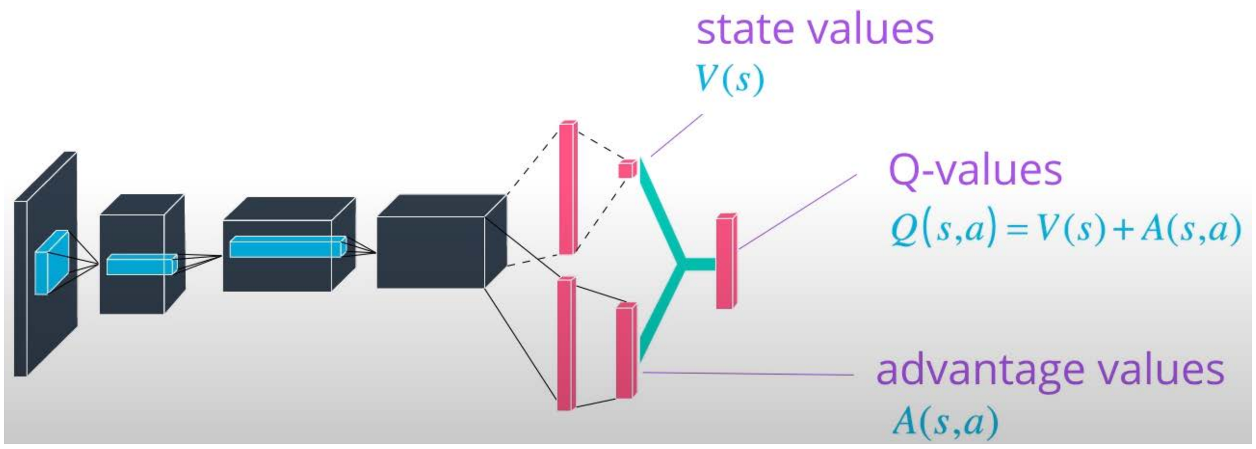


*Figure 2 : Dueling DQN architecture (Source: Udacity, Deep Reinforcement Learning Nano degree)*

## Agent in Action:

The agent is trained using Vanilla DQN, Double DQN, Dueling DQN and with their possible combination. With DQN and its improvements, the agent was able to solve the environment in around 500 episodes.  The environment is considered solved, when the agent is able to get average of 13 score for 100 consecutive episodes. Figure 3 and Figure 4 provides plots of rewards for banana environment.

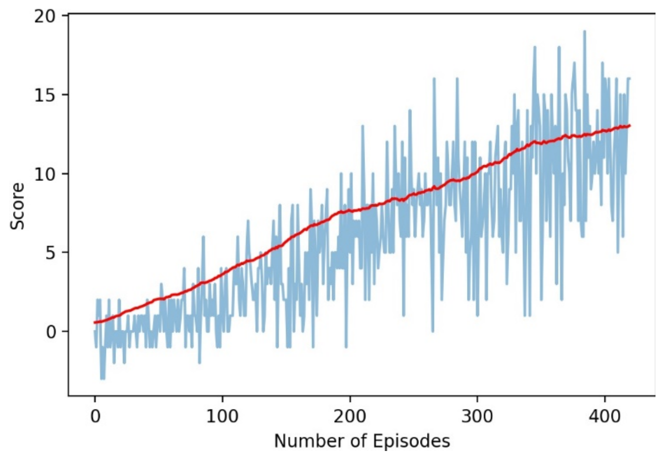


*Figure 3: Rewards using DQN during Agent's Training*

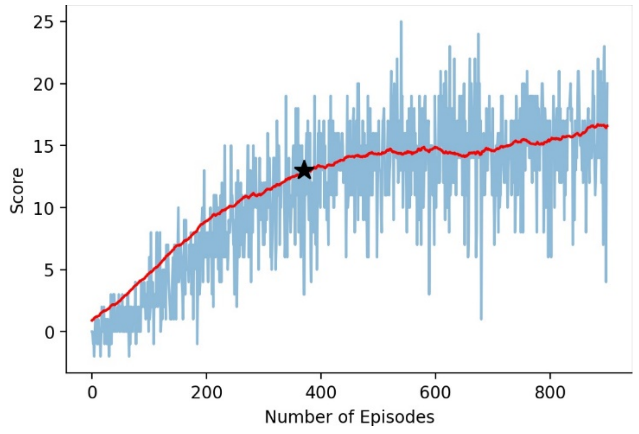


*Figure 4: Rewards using the double DQN with 1000 episodes. The environment was solved in less than 400 episodes as depicted with star.*

## Future work

For the further improvement of the agent's performance, the following future tasks will be performed:

- Implementation of prioritized experience replay
- Training agent using pixels data only
- Sometimes, the agent seems to get stuck in two set of actions (like moving forward and backward), this needs to be further investigated.
- Grid search to find optimal hyper-parameters.