Article I. Solution Design

Three algorithms were experimented to train an AI Agent to maximize the reward by collecting more of yellow bananas. Three algorithms used are Deep Q Networks, Double Deep Q Networks and Duel Deep Q Networks.

Article II. Algorithms

Section 2.01 Deep Q Networks

(a) Description

Q Value approach in reinforcement learning used to tabulate q value for each state-action pair. Deep Q Network is an approach to replace the table with a deep neural network. At each state we approximate the neural network parameter values, to find the q-value against each action.

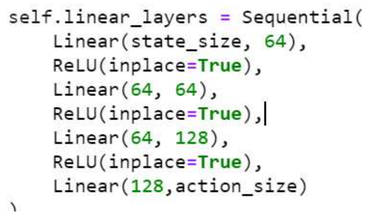
Now when we deal with supervised problem, training a neural network means training the parameters to minimize the loss calculated over the difference between target and predicted end values. In the case of Reinforcement, our target Q value is also moving, that is our target is not fixed. Therefore, we use two neural networks, first one updates the parameters/weights to minimize the difference between target and predicted value, and other network that used parameters calculated by first network in previous time frame to know current targets. These targets are updated after some time.

Learning Algorithm to train agent using DQN algorithm is following:

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights W
Initialize target action-value function Q' with weights W'=W
For episode=1, M do
  Initialize sequence s1= x1
        For t=1, T do
              With probability e select a random action at
              otherwise select at= $\text{argmax}_a$ Q(s1,a | W)    { Q with W parameter represents value returned from first NN }
              Execute action at in emulator and observe reward $r_t$ and image $x_{t+1}$
              Set $s_{t+1} = s_t, a_t, x_{t+1}$
              Store transition in D
              Sample random minibatch of transitions $(s_j,a_j,r_j,s_{j+1})$
              Set target
              if episode terminates at next step
                     $y_j = r_j$ {reward}
              else:
                     $y_j = r_j +$ gamma* argmaxa Q(s1,a | W')  {Q with W' parameter represents value returned from 2nd NN }

              Perform a gradient descent step on $[yj- \{Q\ wj,aj;\ W)]^2$ with respect to the network
       parameters W
              Every C steps reset Q'= Q
       End For
End For
```

(b) Model Architecture



```
self.linear_layers = Sequential(
    Linear(state_size, 64),
    ReLU(inplace=True),
    Linear(64, 64),
    ReLU(inplace=True),
    Linear(64, 128),
    ReLU(inplace=True),
    Linear(128,action_size)
)
```

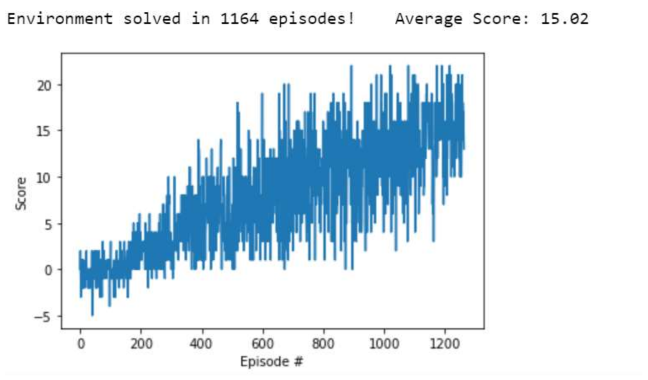
(c) Hyper Parameters

- Replay Buffer Size = 1e6
- Minibatch Size = 64
- Discount Factor (Gamma) = 0.99
- For soft update of target parameters TAU = 1e-4
- Learning Rate= 5e-5
- How often update the network (UPDATE_EVERY) = 2

(d) Result

(i) Training

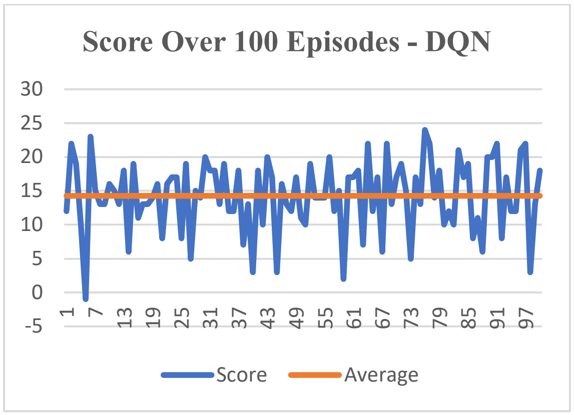
The environment was first solved in 1164 episodes to achieve average score of 15.02



```
Environment solved in 1164 episodes!	Average Score: 15.02
```

(ii) Test

Average Reward over 100 episodes = 14.26



The values can be found in last cell of Navigation-DQN.ipynb. Computation Details of Graph can be found in graph.xlsx

Section 2.02 Duel Deep Q Networks

(a) Description

Deep Q Networks calculate Q values for each possible Action. Where as Duel Deep Q Networks Calculate two different values, on input state, the first value calculated is A (s, a) action value corresponding to each action for given state, and the other value is V(s) is the value of each state.

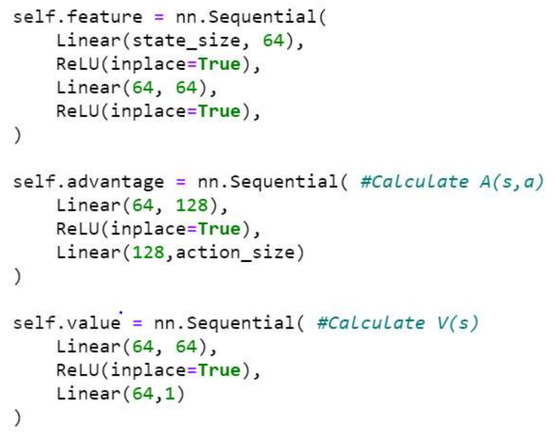
Q (s, a) is then calculated by summing the corresponding A (s, a) value and V(s) and tuning it with mean of A (s, a).

$$Q(s, a) = A(s, a) + V(s) - \text{mean}_a(A(s, a))$$

A (s, a) and V(s) share common neural net architecture at first and then have different NN architecture in the end.

After calculating these values the basic DQN algorithm described above is used , to train the agent.

(b) Model Architecture



```
self.feature = nn.Sequential(
    Linear(state_size, 64),
    ReLU(inplace=True),
    Linear(64, 64),
    ReLU(inplace=True),
)

self.advantage = nn.Sequential( #Calculate A(s,a)
    Linear(64, 128),
    ReLU(inplace=True),
    Linear(128,action_size)
)

self.value = nn.Sequential( #Calculate V(s)
    Linear(64, 64),
    ReLU(inplace=True),
    Linear(64,1)
)
```

(c) Hyper Parameters

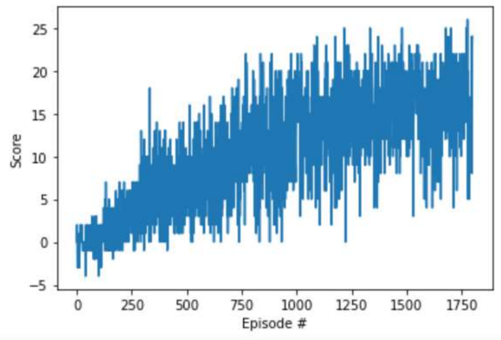
- Replay Buffer Size = 1e6
- Minibatch Size = 64
- Discount Factor (Gamma) = 0.99
- For soft update of target parameters TAU = 1e-4
- Learning Rate= 5e-5
- How often update the network (UPDATE_EVERY) = 2

(d) Result

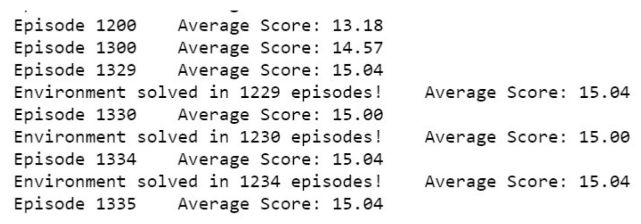
(i) Training

Overall Average score in 1800 episodes was 16.6

```
Environment solved in 1800 episodes!    Average Score: 16.60
```



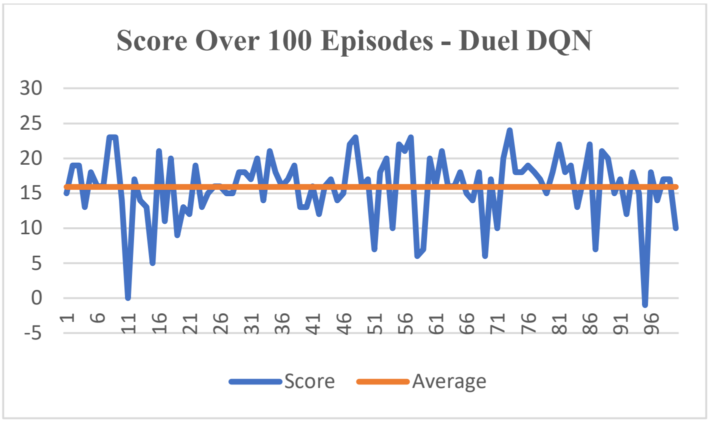
The environment was first solved in 1329 episodes to achieve average score of 15.04.



```
Episode 1200	Average Score: 13.18
Episode 1300	Average Score: 14.57
Episode 1329	Average Score: 15.04
Environment solved in 1229 episodes!	Average Score: 15.04
Episode 1330	Average Score: 15.00
Environment solved in 1230 episodes!	Average Score: 15.00
Episode 1334	Average Score: 15.04
Environment solved in 1234 episodes!	Average Score: 15.04
Episode 1335	Average Score: 15.04
```

(ii) Test

Average Reward over 100 episodes = 15.91



The values can be found in last cell of Navigation-Duel DQN.ipynb.

Computation Details of Graph can be found in graph.xlsx

Section 2.03 Double Deep Q Networks

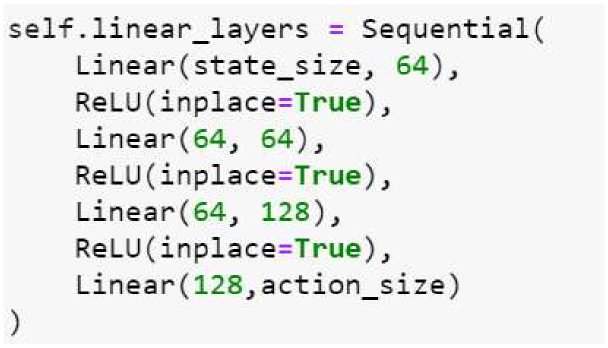
(a) Description

Double Deep Q Networks use different technique to get target Q values , than from DQN, as expressed in following pseudocode (else block).

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights W
Initialize target action-value function Q’ with weights W’=W
For episode=1, M do
  Initialize sequence s1= x1
      For t=1, T do
            With probability e select a random action at
            otherwise select at= $\text{argmax}_a$ Q(s1,a | W)    { Q with W parameter represents value returned from first NN }
            Execute action at in emulator and observe reward $r_t$ and image $x_{t+1}$
            Set $s_{t+1}$ = $s_t$, $a_t$, $x_{t+1}$
            Store transition in D
            Sample random minibatch of transitions ($s_j$,$a_j$,$r_j$,$s_{j+1}$)
            Set target
            if episode terminates at next step
                   $y_j$ = $r_j$ {reward}
            else:
                   max_index= $\text{argmaxIndex}_a$ Q(s1,a|W) {Gets the action depending on Q values returned from NN with W }
                   $y_j$= $r_j$+ gamma* Q(s1,a | W’)[max_index]  { Gets the Q value of max_index value from 2nd with W’ }

            Perform a gradient descent step on $[yj- \{Q\ wj,aj;\ W)]^2$ with respect to the network
       parameters W
            Every C steps reset Q’= Q
       End For
End For
```

(b) Model Architecture



```
self.linear_layers = Sequential(
    Linear(state_size, 64),
    ReLU(inplace=True),
    Linear(64, 64),
    ReLU(inplace=True),
    Linear(64, 128),
    ReLU(inplace=True),
    Linear(128,action_size)
)
```

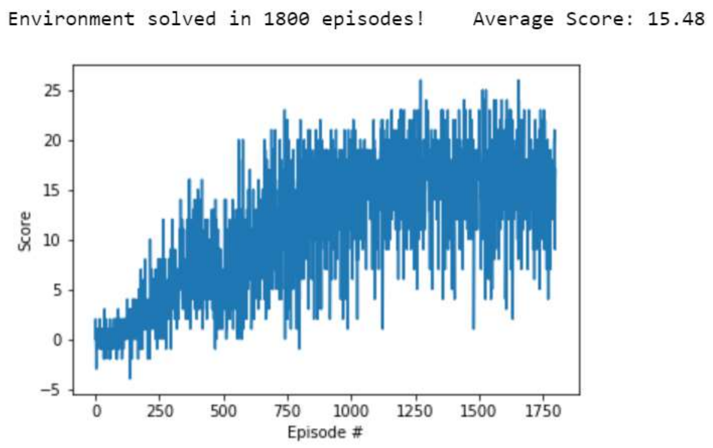
(c) Hyper Parameters

- Replay Buffer Size = 1e6
- Minibatch Size = 64
- Discount Factor (Gamma) = 0.99
- For soft update of target parameters TAU = 1e-4
- Learning Rate= 5e-5
- How often update the network (UPDATE_EVERY) = 2

(d) Result

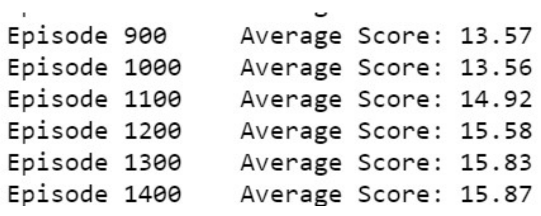
(i) Training

Overall Average score in 1800 episodes was 15.48.



```
Environment solved in 1800 episodes!	Average Score: 15.48
```

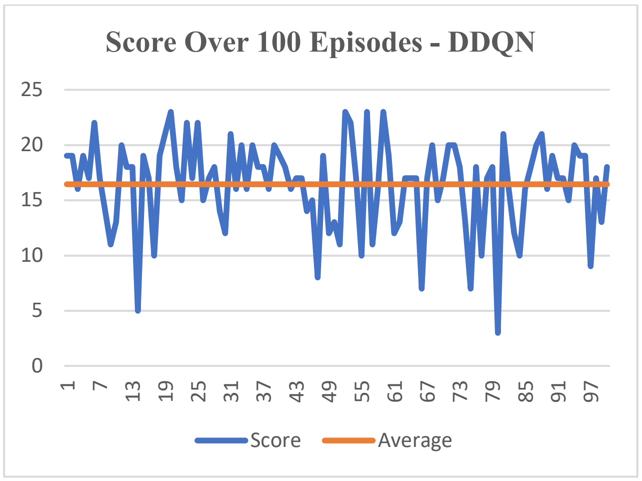
The environment was first solved in 1200 episodes to achieve average score of 15.58.



```
Episode 900	Average Score: 13.57
Episode 1000	Average Score: 13.56
Episode 1100	Average Score: 14.92
Episode 1200	Average Score: 15.58
Episode 1300	Average Score: 15.83
Episode 1400	Average Score: 15.87
```

(i) Test

Average Reward over 100 episodes = 16.44



The values can be found in last cell of Navigation-DDQN-Test.ipynb.

Computation Details of Graph can be found in graph.xlsx

Article III. Conclusion

The best average reward over 100 episodes when training mode is off , was achieved by DDQN Algorithm, it was trained to solved the environment i.e to get average >15 (threshold set) in 1200 episodes .

Article IV. Future Ideas

1. In order to improve agents performance further , prioritized experience replay can be used instead of random sampling of experiences. The priority can be defined either on basis of loss or using some other priority technique like to replace the new experience with the oldest experience that came in. Definitely priority a/c to loss occurred would be more meaningful.
2. In order to improve agent's performance an attention mechanism can be introduced that calculates the Q values of given state, keeping in mind the affect of previous and next states on the current state to maximize the reward. These attention values can be calculated by using different and independent NN in the system.