NAVIGATION PROJECT REPORT

project overview:

This project was about training an agent to pick bananas in a large square space. This space has blue and yellow bananas,if the agent picked a yellow banana it will get a reward of +1 .On the other hand if it picked a blue banana it will get a reward of -1.When the agent reaches a total average reward of +13.00 the environment is solved.

The state space has a dimension of size 37 which includes the the agent's velocity, along with ray-based perception of objects around the agent's forward direction. The action space has a dimension of size 4;it consists of the following actions;(0)move forward,(1)move backward, (2)turn left and (3)turn right.

Learning Algorithim implementation

1.*dqn agent class:*

This class defines how the agent is going to learn in the environment.

a)***act()method:***

Here the agent chooses the suitable action given current state s,by using an epsilon greedy policy. Whereby it chooses the action with the highest Q value with probability (1-epsilon) and chooses an action randomly with a probability(epsilon value).The lower the epsilon the agent is most likely to choose the greedy action.

*b)****step()method:***

This method first saves the resent agent's experience replay (SARS)tuple into the memory. If the memory is larger than the batch size ;an experience is randomly selected from the memory,the, the agent performs some learning.

c)**learn() method:**

This method defines how the agent will learn.

First,we pass an experience tuple into the method .

Get the Q_target values:

$Q\_target = R + \gamma max_a Q(s',a')$

(i)R is the reward.

(ii)$\gamma max_a Q(s',a')$ is the discounted maximum Q value in the next_state s' .

To get $\gamma max_a Q(s',a')$,pass the next_state to the target model then choose the highest action value and multiply by gamma (which is the discount value).

Get the expected Q value:pass the current state into the local model and choose the action value of the action A of the experience replay tuple.

Use the Mean Squared Error function to get the error between the Q_target value and Q_expected value.

Perform backpropagation on the local model .

Perform a step on the optimizing function.

Then update the weights of the target model .

d)**update_weights():**

This updates the weights of the target model with the local current weights of the local model.

$\theta\_target = \tau*\theta\_local + (1 - \tau)*\theta\_target$

2.**replay_buffer class:**

a)**sample() method:**

This randomly samples a SARS tuple from the memory.

b)**add() method:**

this adds a SARS(experience) tuple into the memory.

c)**len() method:**
This returns the current length of the memory.

Model architecture:

My DQN model architecture was mainly composed of three fully connected layers. I tried the model with high dense layers and lower dense layers. I realized that the the model worked better on a bit lower dense layers.
Here are the results:

HIDDEN *LAYER1 NODES*	**HIDDEN LAYER2 NODES**	**NUMBER OF EPISODES THAT THE ENVIRONMENT WAS SOLVED.**
128	64	388
64	32	272

Choosing hyper-parameters:
1.Buffer_size:le5
2.Batch_size:128
3.gamma: 0.95
This is the discount of the accumulated returns. Choosing a higher gamma the agent will care more about the future reward ,on the other hand choosing a lower gamma the agent will care more about the immediate reward. The agent worked better on a relatively high gamma.

4.update_every: 4
This updates the weights of the dqn model after a number of episodes.
5.TAU:le-3
6.LEARNING RATE:0.0005
The model converged well on lower learning rates.
7.EPSILON:
a)epsilon start=1.0
b)epsilon_decay=0.99
c)epsilon_min=0.01
I chose an epsilon of a range of 1.0 to 0.01 probability.
I realized that choosing a very small epsilon caused a very inconsistent average score. This is because a very low epsilon will lead to choosing the actions with too much randomness.
I applied an epsilon decay at the end of every episode to gradually reduce the epsilon up to 0.01 probability. This will help in slightly reducing the chance of always choosing the greedy policy. The agent solved the environment faster in a high epsilon decay rate.

Training the agent:

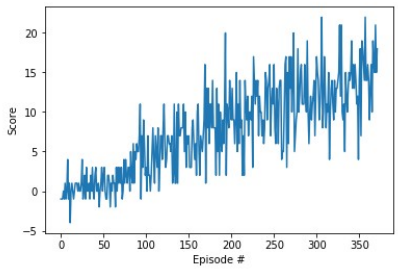
I used two models ;a local model and a target model .The local model was used for training and the target model was used to keep the local model's weights of the previous iterations($\bar{\theta}=\theta_{i-1}$) and to get the Q target values.

I trained the agent in 1000 episodes and each episode had 2000time steps. After every episode I added the total reward to the score_window and also performed epsilon decay .

BEST RESULTS: I managed to solve the environment in 272 episodes.

```
Episode 100     Average Score: 1.44
Episode 200     Average Score: 6.60
Episode 300     Average Score: 10.73
Episode 372     Average Score: 13.02
Environment solved in 272 episodes!     Average Score: 13.02
```

```
In [17]: # plot the scores
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(rewards_list)), rewards_list)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```



Ideas of future work:

1.priority experience replay:

This technique will help the agent to learn faster,because,with experience replay some old experiences may be lost when the memory is dequeued. Prioritized experience replay selects experiences with higher temporal differences more frequently. Hence the agent learns more efficiently.

References.

Tom Schaul, John Quan, Ioannis Antonoglou and David Silver(2016) Prioritized Experience Replay .