Project – Navigation (Udacity Deep Reinforcement Learning Nanodegree).

Introduction

This is part of Navigation project of Udacity Deep Reinforcement Learning Nanodegree. Goal of the project is to train the Agent to navigate and pick-up bananas in the Banana environment provided by Unity Environments. After training it should go for yellow bananas and avoid blue ones. This is an episodic environment is considered solved when agents get an average score of 13 over 100 consecutive episodes.

Implementation

Two algorithms, DQN and Double DQN, with single and dueling network structure have been implemented. High-level overview – Initialize the environment, train the agent and review the actions taken by agent.

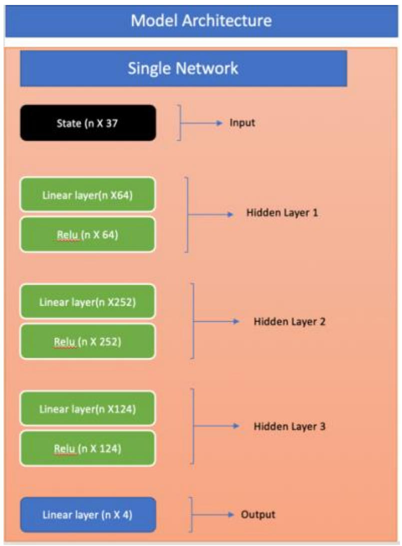
Hyperparameters.

Hyperparameter	Value	Description
n_episodes	3000	max number of the episodes in the training
max_t	2700	max number of time steps in the training episode
eps_start	1	start value of the epsilon parameter (epsilon-greedy strategy)
eps_end	0.01	end value of the epsilon parameter (epsilon-greedy strategy)
eps_decay	0.999	decrease of the epsilon parameter (epsilon-greedy strategy)
BUFFER_SIZE	1.00E+05	replay memory size
BATCH_SIZE	16	minibatch replay buffer size
GAMMA	0.9	discount factor (for future rewards)
TAU	1.00E-03	soft update of the target Q-network parameter
LR	5.00E-04	learning rate
UPDATE_EVERY	8	the frequency of updating the online Q-network

Model Architecture

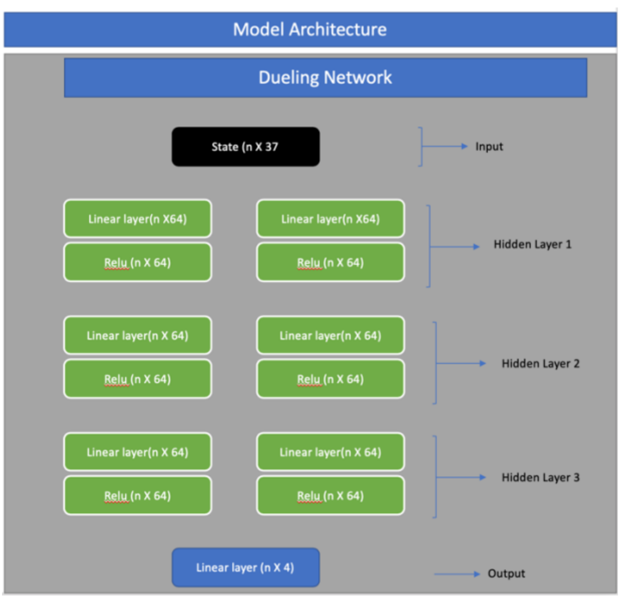
Single network

Single network structure consists of one input layer, 3 hidden layers and one output layer. Detail is shown in diagram below:



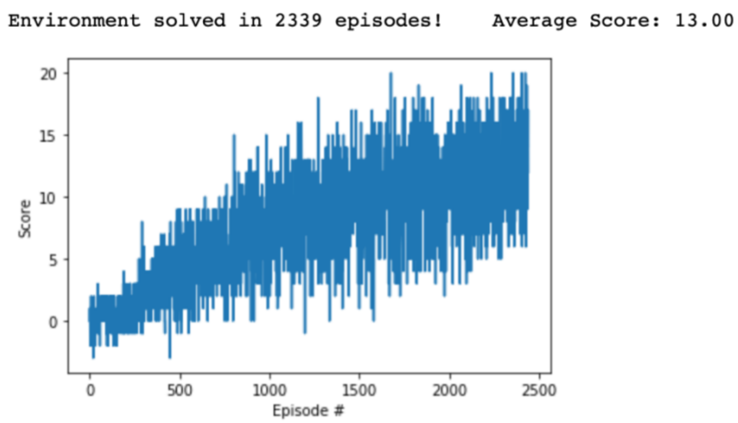
Dueling network

Dueling network consists of one input layer which is then shared by 2 networks. Each network consists of 3 hidden layers and one output layers. One network is outputting a single value V which represents the state value. The other one is outputting values which number equals to number of actions. This represents the advantage function value adv(i) = Q(i)-V for each action. Then the two output layers are linked up to form the final output layers by mathematical operation Q = V+(adv(i)- mean of adv). Detail is shown in diagram below

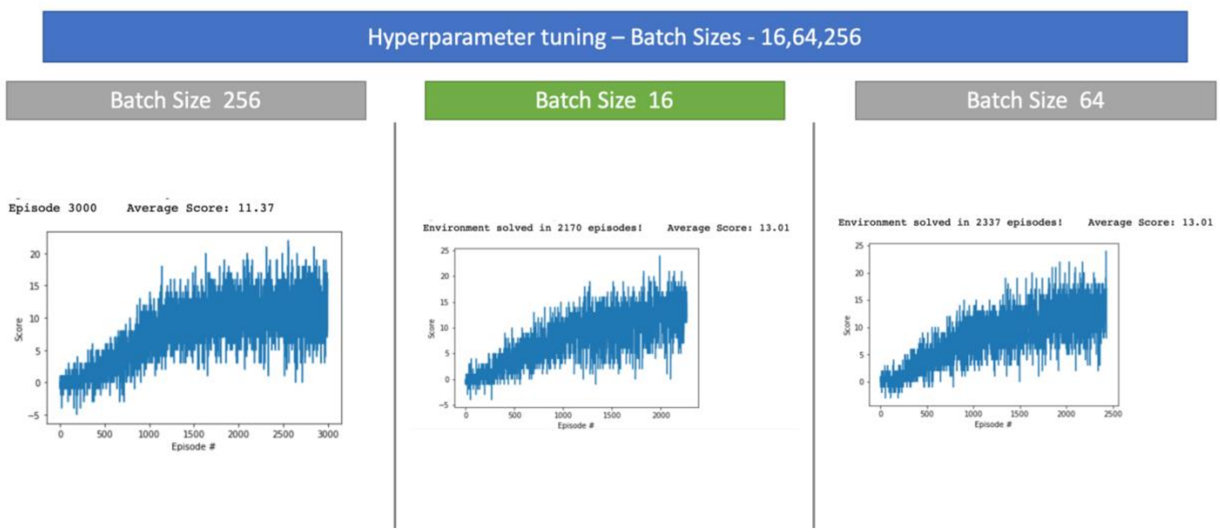


<u>Results</u>

Initial training.



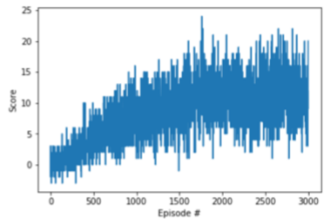
Hyperparameter Tuning



Hyperparameter tuning – Gamma Values – 1, 0.95, 0.9

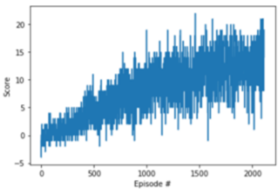
Gamma 1

Episode 3000 Average Score: 10.78



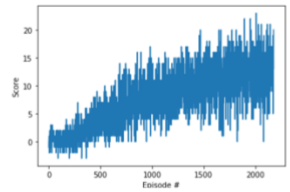
Gamma 0.95

Environment solved in 2013 episodes! Average Score: 13.00



Gamma 0.90

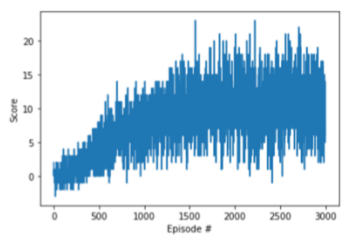
Environment solved in 2077 episodes! Average Score: 13.04



Hyperparameter tuning – Update every – 4, 6, 8

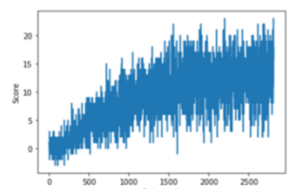
Update Every 4

Episode 3000 Average Score: 10.09



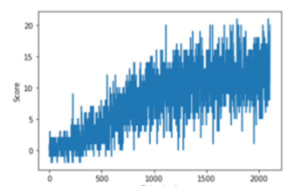
Update Every 6

Environment solved in 2717 episodes! Average Score: 13.01



Update Every 8

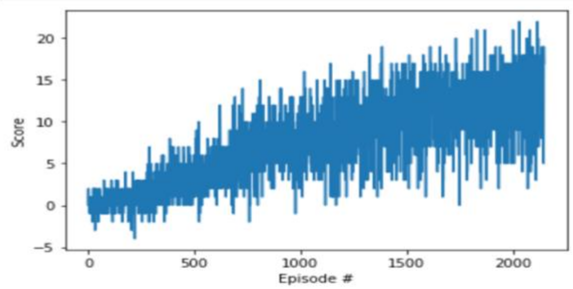
Environment solved in 2008 episodes! Average Score: 13.01



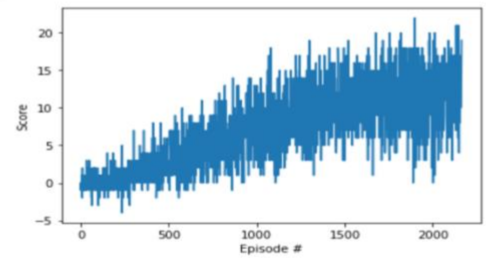
Comparison between different models.

Agent is trained using different settings of isDDQ and isDuel. Overall outperformed setting is isDDQ=True,isDuel=False.

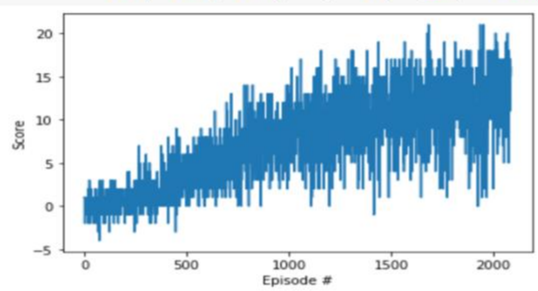
```
# train an agent with deep Q learing in optimized hyperparameters for 3000 steps
agent = Agent(state_size=37, action_size=4, seed=32,BUFFER_SIZE = int(1e5),BATCH_SIZE = 16,
              GAMMA = 0.9,TAU = 1e-3,LR = 5e-4,UPDATE_EVERY = 8,isDDQ=False,isDuel=False)
```



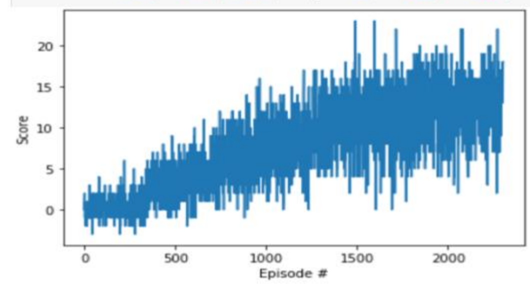

```
# train an agent with deep double Q learing with deuling network in optimized hyperparameters for 3000 steps
agent = Agent(state_size=37, action_size=4, seed=32,BUFFER_SIZE = int(1e5),BATCH_SIZE = 16,
              GAMMA = 0.9,TAU = 1e-3,LR = 5e-4,UPDATE_EVERY = 8,isDDQ=True,isDuel=True)
```

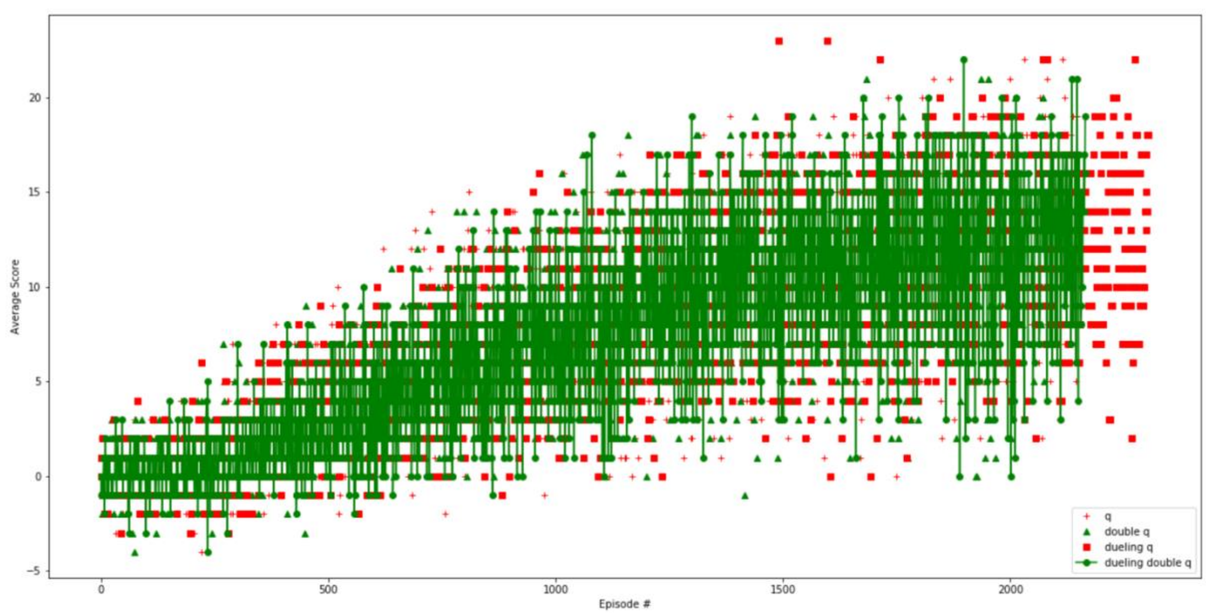



```
# train an agent with deep double Q learing in optimized hyperparameters for 3000 steps
agent = Agent(state_size=37, action_size=4, seed=32,BUFFER_SIZE = int(1e5),BATCH_SIZE = 16,
              GAMMA = 0.9,TAU = 1e-3,LR = 5e-4,UPDATE_EVERY = 8,isDDQ=True,isDuel=False)
```




```
# train an agent with deep Q learing with deuling network in optimized hyperparameters for 3000 steps
agent = Agent(state_size=37, action_size=4, seed=32,BUFFER_SIZE = int(1e5),BATCH_SIZE = 16,
              GAMMA = 0.9,TAU = 1e-3,LR = 5e-4,UPDATE_EVERY = 8,isDDQ=False,isDuel=True)
```





Future improvements

The agent can be further improved by the following:

1. Implementing prioritized experience replay so that the agent can focus more on experience which has larger error.
2. Implementing Expected SARSA. For

   a(t+1)=ArgMax(Q_target(s(t+1),a))

   y= r(t)+Gamma*Q_target(s(t+1),a(t+1))

   instead of only taking account of the maximum Q value in the next state, we can take epsilon into account so that

   a1(t+1)=ArgMax(Q_target(s(t+1),a))

   a2(t+1)=random(4)

   y=r(t)+Gamma*((1-epsilon)*Q_target(s(t+1),a1(t+1)) + epsilon*Q_target(s(t+1),a2(t+1)))

   This will change the algorithm from off-policy to on-policy which can help to stabilize the results.

3. Implementing Rainbow Algorithm [https://arxiv.org/pdf/1710.02298.pdf] which combines good features from different algorithms to form an integrated agent.

Conclusion

1. Hyper parameters tuning is really key.
2. Running the model on Udacity GPU is really a challenge as it disconnects often while model is running.
3. Topic is more complex than initially anticipated.