Project: Navigation

Submitter:

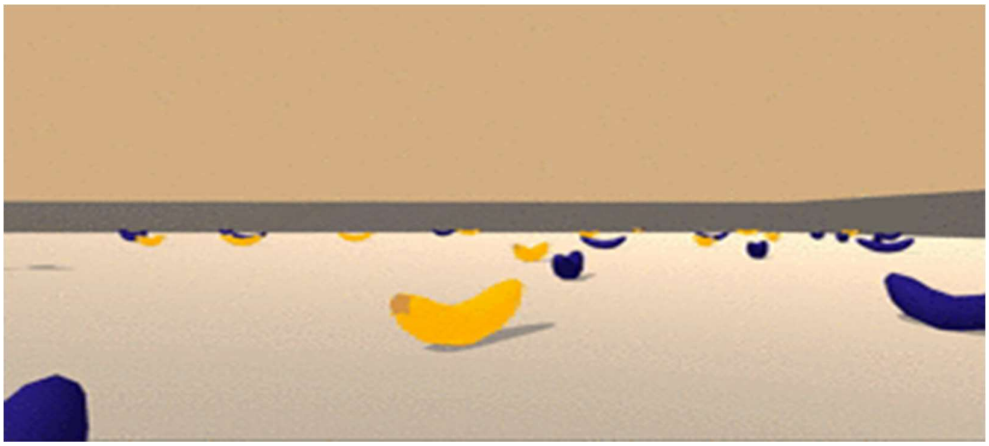
Monimoy Deb Purkayastha (monimoyd@yahoo.com | monimoyd@gmail.com)

Aim:

For fulfilment of Udacity Deep Reinforcement Learning Nano Degree

I. Project Overview

Problem Statement



In this project I have to train an agent 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 my 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, and in order to solve the environment, my agent must get an average score of +13 over 100 consecutive episodes.

The observations space is 37-dimension continuous space. Out of 37 dimensions 35 dimensions are based on of ray-based perception of objects around the agent's forward direction. Rest of two 2 dimensions are for velocity.

For the 35 dimension of ray perception, they are 7 rays projected. Rays are projected at angles: 20, 45, 70, 90 , 110, 135, 160 respectively.

Each ray has 5 dimensions out of these 4 dimensions correspond to four type of objects (agent, wall, blue banana, yellow banana) and the last dimension correspond to the distance measure which is a fraction of ray length.

I have used Unity Machine Learning Agent Toolkit to train, evaluate and test the machine Reinforcement Learning algorithm

II. Solution approach

In this project I have used Deep Q Learning (DQN) Algorithm for solving the agent to collect bananas to get the requisite score.

Before DQN algorithm I will first explain Q learning

Q Learning

Q learning is a off policy TD(0) control. Goal of Q learning is learn a policy based on a Q table. The core of the algorithm is a value iteration update using the weighted average of the old value and the new information as below:

$$Q(St, At) = Q(St, At) + \alpha * (Rt+1 + \gamma * \max(a) Q(St+1, a) - Q(St,At))$$

Where

$Q(St, At)$: Q value for action, value pair at time t

α : Learning rate

γ : Discount rate

During the start all the Q-table values are 0. Using the above function, Q values are iteratively updated. As the algorithm to explore the environment**,** the Q-function gives us better and better approximations by continuously updating the Q-values in the table.

Deep Q Learning

DQN is based upon Q-learning algorithm with a deep neural network as the function approximator**.** A deep Q network (DQN) is a multi-layered neural network that for a given state s outputs a vector of action values Q(s; w), where w are the parameters of the network.

Two important properties of Deep Q learning are:

1. Fixed Q Targets: In the Q-learning algorithm using a function approximator, the TD target is dependent on the network parameter w. As w parameter sis learn/updated, it can lead to instabilities. To address this, separate target neural network is used which is updated a number of steps are done

2. Experience Replay: When agent interacts with environment, the sequence of experience tuple can be highly correlated. To solve this experience replay is used. Here experience tuples (S,A,R,S`) are added to replay buffer and are randomly sampled from the replay buffer so that samples are not correlated

III. Methodology

In this project, I have used Deep Q Learning (DQN)algorithm to train the agent to pick the yellow banana. The DQN uses both experience replay and a separate target Neural Network

For Deep Q learning I have used pytorch for Neural Network. I have used 37 dimension state vector as input and 4 actions as output.

I tried two different neural networks one with hidden layer dimensions of 32x64x32 and another network of 64x64x64.

I used Adam optimizer and used the following hyper parameters:

Batch Size: 64
Discount Rate (gamma) : 0.99
Soft update of target parameters (Tau) : 0.001
Neural Network Initial Learning Rate: 0.0005
Buffer Size: 100000

As the network tend to overfit after training for many episodes, I have implemented a method to reduce the learning rate by 0.7 if the mean over

consecutive 100 episodes does not improve by 1.0 after 100 episodes. Once learning rate is not changed, next time will be changed only after another 100 episodes.

Also I gathered following metrics while performing training

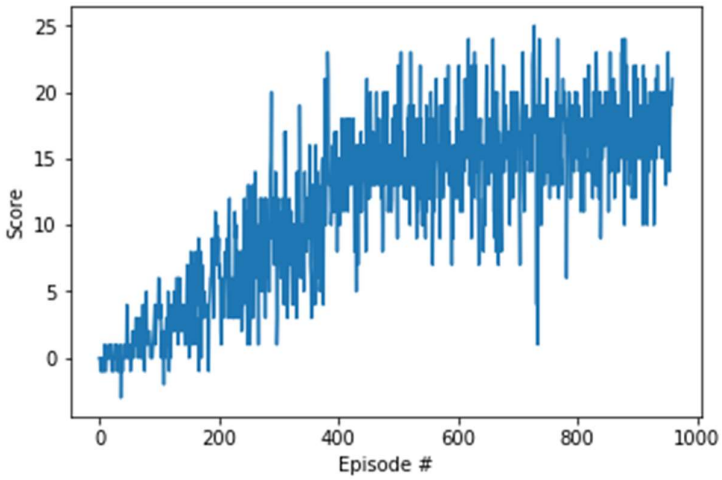
i. **Score** : Score received in each episode
ii. **Mean Score:** Mean score over last consecutive 100 episodes
iii. **Steps:** Number of steps used in each episode

IV. Results

i. DQN Network 32x64x32

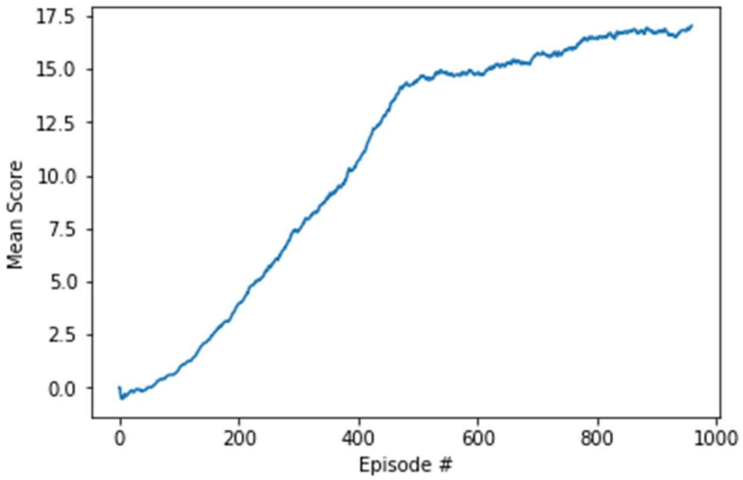
The environment is solved in 960 episodes i.e mean score over 100 episodes were achieved (The target that I set for the problem)

- **Plot of scores vs episodes**



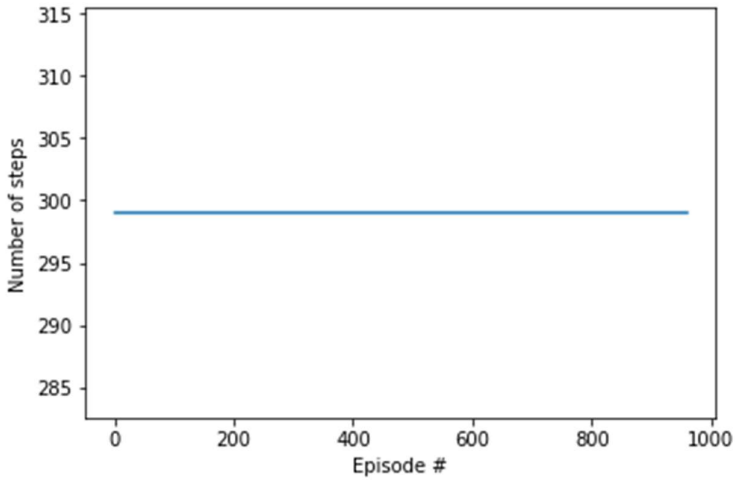
Observation: The scores gradually increase over episodes but there is a variance in scores

- **Plot for Average Score over 100 consecutive episodes vs episodes**



Observation: The mean scores over 100 episodes gradually increase but goes flat after around 500 episodes

- **Plot for number of steps vs episodes**

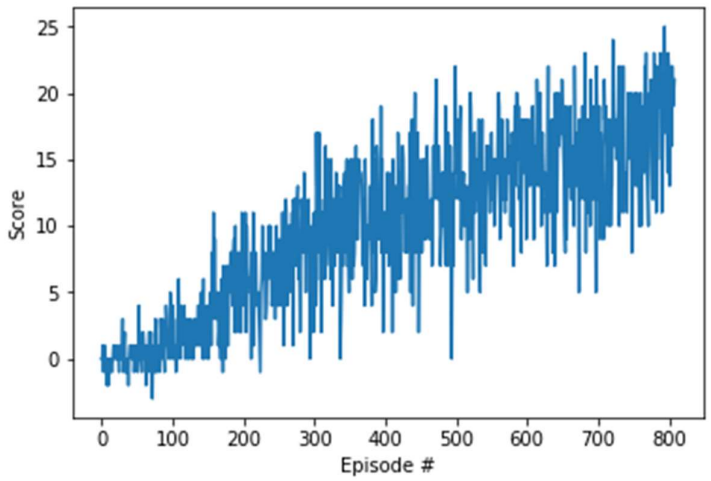


Observation: The number of steps are constant over episodes. Not very useful for analysis

ii. DQN Network 64x64x64

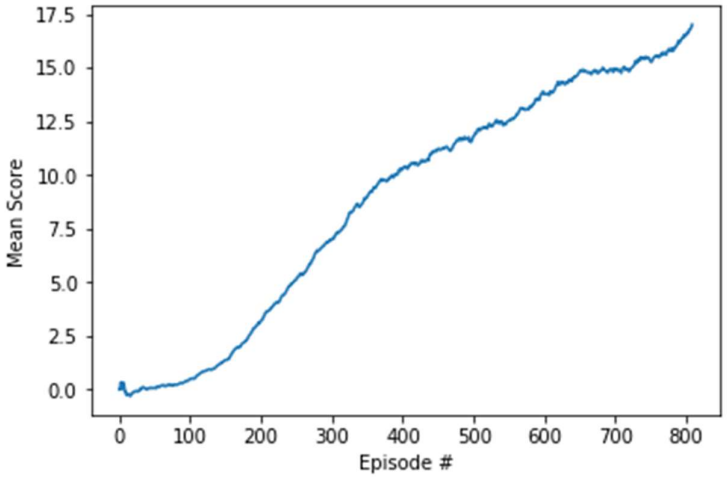
The environment is solved in 809 episodes i.e mean score over 100 episodes were achieved score of 17 (The target that I set for the problem)

- **Plot of scores vs episodes**



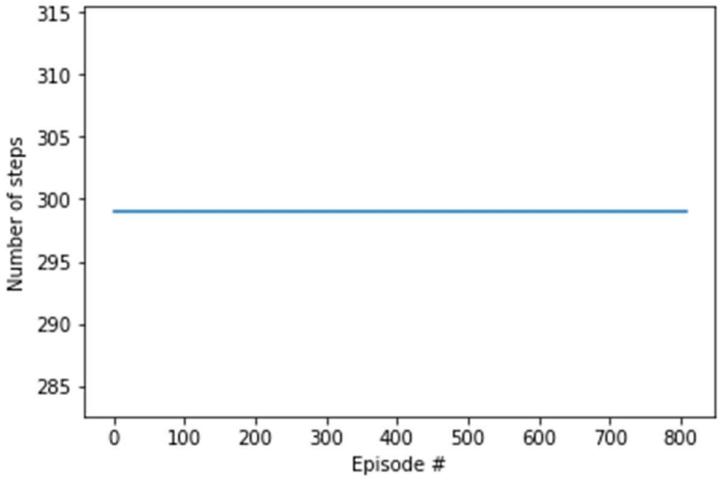
Observation: Scores increase as episodes incease but score variance is less compared to 64x32x64 network

- **Plot for Average Score over 100 consecutive episodes vs episodes**



Observation The mean scores over 100 episodes gradually increase but slope is always positive

- **Plot for number of steps vs episodes**



Observation: The number of steps are constant over episodes. Not very useful for analysis

V. Conclusion:

In this project I have used Deep Q Learning algorithm and tried two different neural networks 32x64x32 and 64x64x64.

i. Neural network having 64x64x64 achieved mean score of 17 over 100 episodes in 809 episodes which is 151 episodes less than 32x64x32 which took 960 episodes to solve the same problem
ii. Scores (Rewards) earned in each episode by 64x64x64 is having less variance than rewards earned by 32x64x32
iii. Slope of mean scores vs episodes in 64x64x64 network is much higher than slope of mean scores vs episodes curve used by 32x64x32

So, overall I found that 64x64x64 is performing better than 32x64x32 network

Project can be further enhanced by using Double DQN algorithm and Prioritized Experience Replay