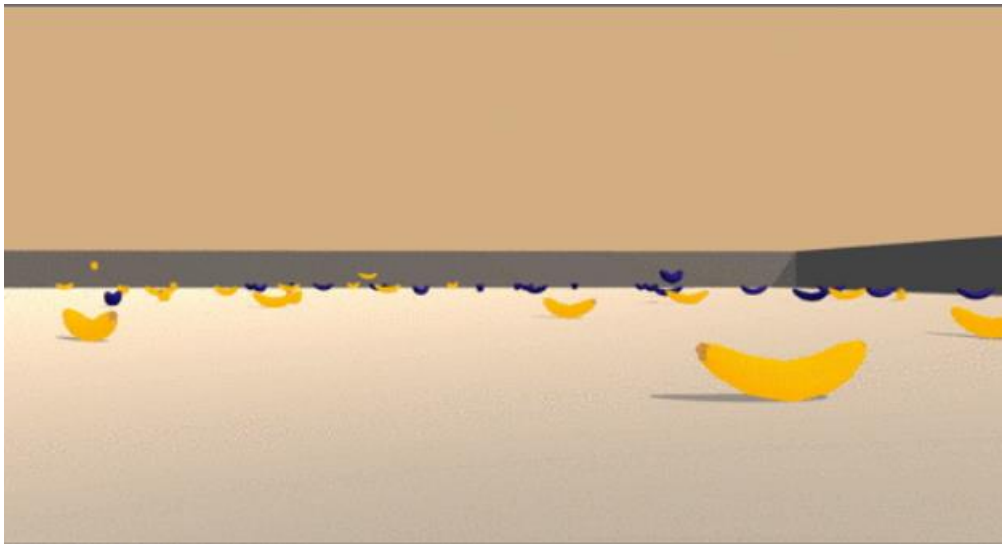
REPORT

Introduction

This project involves solving an environment in which an agent is required to collect 13 yellow bananas all the while avoiding blue bananas. I use Deep Reinforcement learning to solve the problem. In specific, I use Deep Q networks, where in the value policy is determined by a neural network which would take in state of the environment and we receive action probabilities distribution as the output of the network

Environment

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 your 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, your agent must get an average score of +13 over 100 consecutive episodes.

Implementation

The repository consists of following files:

1) Navigation.ipynb
2) dqn_agent.py
3) model.py

Navigation.ipynb is the jupyter notebook where the training happens. We setup the environment and do the basic analysis of it, state vector size, possible actions at every step. We instantiate an Agent (Class present in dqn_agent.py) with state_size = 37, possible actions = 4. This agent takes in the state as the input and gives the prospective action as output considering the epsilon greedy policy. We can take this output and pass it to the environment as the next step and update itself.

Dqn_agent.py contains thee class Agent which contains many functions like step (adds the step into memory cache and calls learn function), learn (Update parameters), act (takes in the state vector and outputs the action).

Model.py contains the network architecture of deep Q network. It is given as follows.

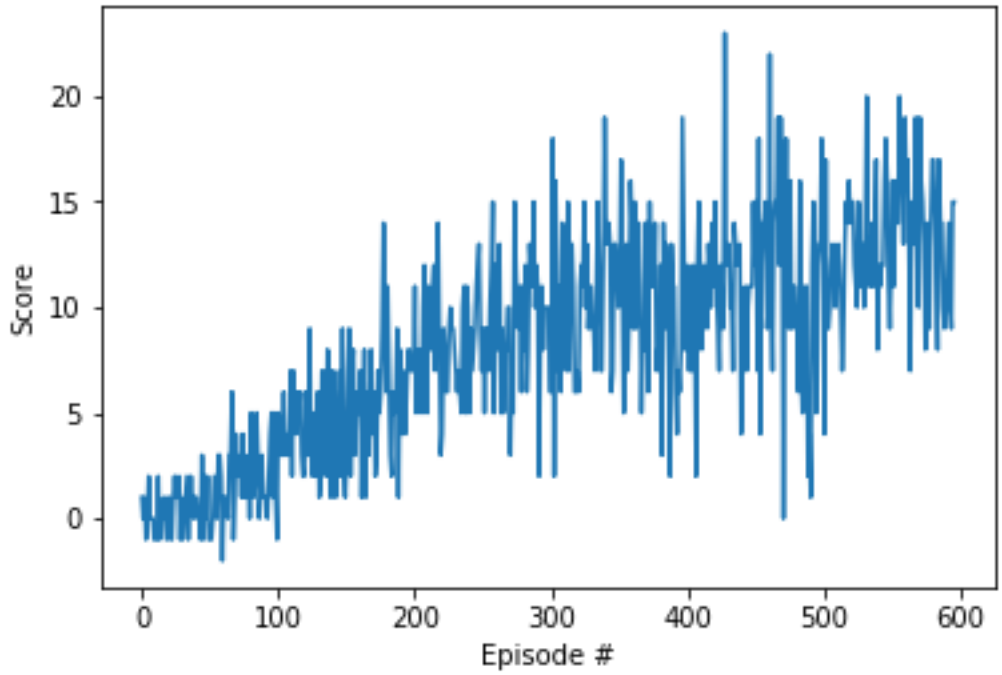
```
Sequential(
  (0): Linear(in_features=37, out_features=64, bias=True)
  (1): ReLU()
  (2): Linear(in_features=64, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=32, bias=True)
  (5): ReLU()
  (6): Linear(in_features=32, out_features=64, bias=True)
  (7): ReLU()
  (8): Linear(in_features=64, out_features=4, bias=True)
)
```

Learning Algorithm

I use the Adam optimizer to reduce the loss. Loss function of choice is MSE Loss between the Action Label and Action predicted.

Parameter	Value
Learning rate	5e-4
num_episodes	4000
gamma	0.99
tau	1e-3
epsilon_start	1
epsilon_end	0.01
epsilon_decay	0.995
Batch Size	64
Update every	4

Rewards




```
Episode 100     Average Score: 1.06
Episode 200     Average Score: 5.01
Episode 300     Average Score: 8.79
Episode 400     Average Score: 10.43
Episode 500     Average Score: 11.06
Episode 596     Average Score: 13.01
Environment solved in 496 episodes!     Average Score: 13.01
```

Ideas for Future Work

In this project, I have committed to implement the agent using Deep Q-learning algorithm. As the state, I have used the pre-defined 37 input features, that stack up with history. Learning from pixels is another alternative one could experiment with. It has been shown that a family of DQN algorithms have shown super-human capacity leaning from raw pixels.

In addition to DDQN, we can experiment with using the next generation of DQNs, such as Dueling Network Architectures, Dynamic Frame skip Deep Q Network, Rainbow, Deep Recurrent Q-Learning among others.

In addition, I would also hypothesis that we could use both hand crafted features and raw pixel as inputs to further experiment with and solve the problem similar to Siamese Neural Networks for example.