🍌BANANA🍌 DQN AGENT

Overview

This project is an assignment from the Udacity Deep Reinforcement Learning course. [p1_navigation] found here. The objective is to train a banana collecting agent using a DQN. The environment is made in the unity engine consisting of an state action space of 37 [0 , 1.] (continues) / 4 (discrete) the objective of the game is to collect as many yellow bananas as possible while avoiding the blue ones during a limited time frame. The assignment is considered passed if the agent reaches an average score of over 13 over 100 episodes.

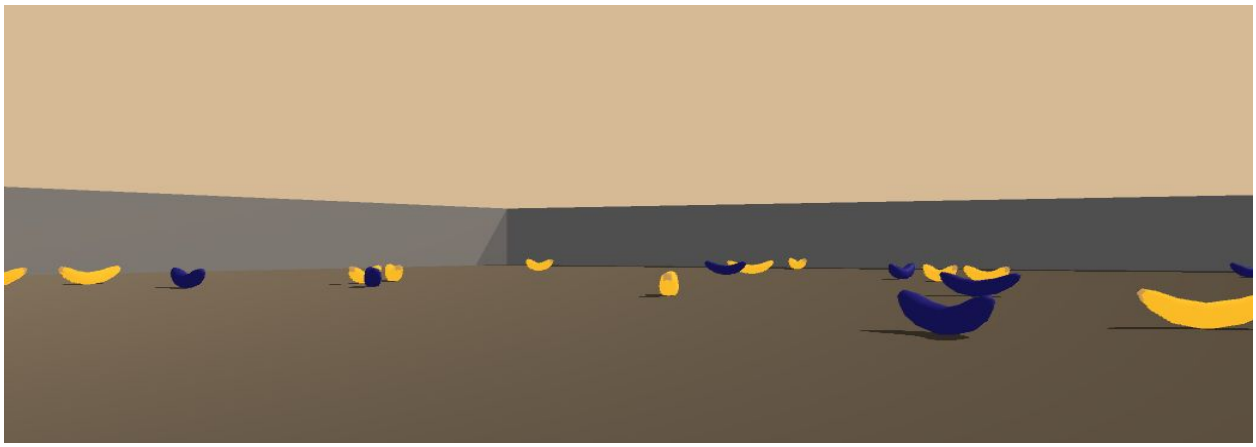
Implementation

1. agent.py
Contains the agent that is trained, it's learning code and the experience replay.

2. model.py
Contains a pytorch model wich is trained in Agent.py

3. Training notebook.ipynb
Contains the code to train an agent and save its model

4. Watching notebook.ipynb
Contains code to load the model and watch the agent play a round of the game



Learning algorithm and model architecture

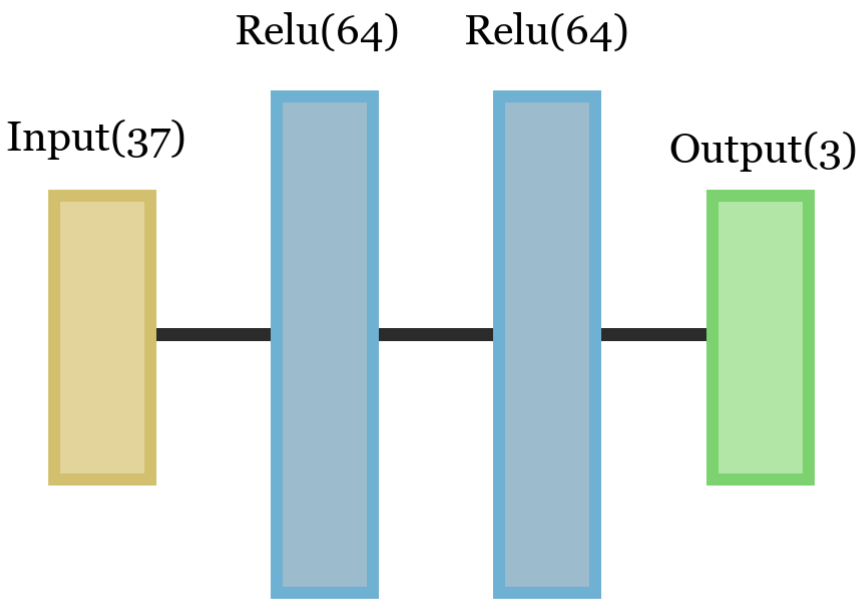
The model is a feed forward neural network trained on a unity environment using a DQN learning algorithm. The models dimensions are 37 -> 64 -> relu -> 64 -> relu -> 3 where the input layer is the state and the output the estimated Q values.

The learning algorithm uses gradient descent to train the network, where the loss is the mean squared error between the expected Q values and the reward plus the discounted expected Q value of the next state.

Learning is done offline and using fixed Q targets, using Experience replay where the training pool is randomized .

The hyper parameters used are:

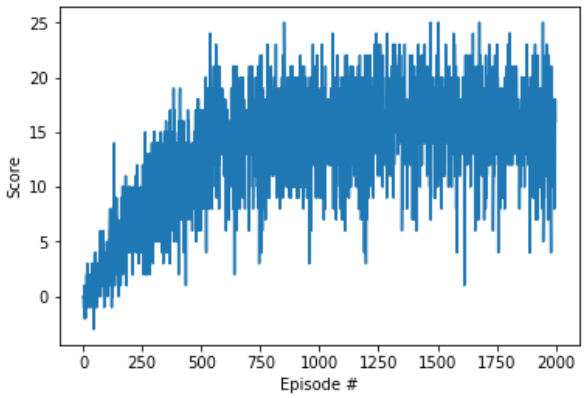
```
Learning_rate =  5e-4
Discount_factor = 0.99
Epsilon (for epsilon greedy) = 1.0
Epsilon_decay = 0.995
Epsilon_min = 0.01
```



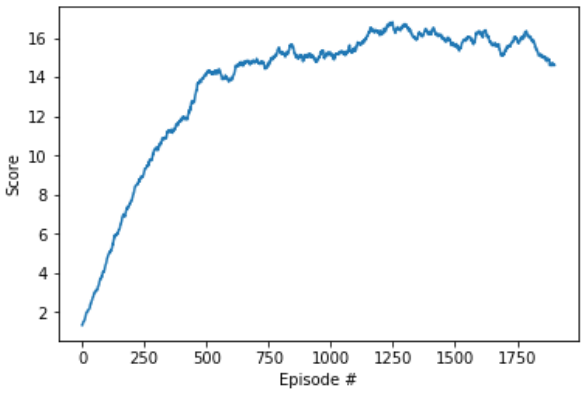
Performance

The agent reaches an average score of over 13 at around 600 episodes and seems to converge to fluctuating between 14 and 16 which is most likely caused by the inherit random nature of the game.

1.



2.



(1. Score per episode, 2. Moving average score with a window of 100 episodes)

Improve performance

Improvement could likely be made in both Training and model performance, where:
Training performance could be improved by implementing some or all of the improvements made to the DQN algorithm. See for example Rainbow dqn wich aims to implement a large selection of these improvements.
Model performance could be improved by implementing an RNN or other architecture aimed at dealing with time series data.