## Project's goal

In this project, the goal is to train an agent to navigate a virtual world and collect as many yellow bananas as possible while avoiding blue bananas

## Environment details

The environment is based on Unity ML-agents

Note: The project environment provided by Udacity is similar to, but not identical to the Banana Collector environment on the Unity ML-Agents GitHub page.

The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents. Agents can be trained using reinforcement learning, imitation learning, neuroevolution, or other machine learning methods through a simple-to-use Python API.

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

## Methodology

This work implements the [Deep Q Network(DQN)] https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf, where a function approximate the state space because it's too costly to make discretization of it.

Deep Neural network represent the mapping Q(s,a). Using Bellman equations we train the network while we train the parameters to produce the Q values.

## Learning algorithm

A range of neural models were tried exploring wide, deep and shallow configurations. Overall, the simpler models performed as well or better than deeper ones and wide models performed worse than narrow ones. All models started with a 37 x 1 input vector from the environment, constructed two or more fully connected hidden layers and ended with a fully connected layer outputing 4 outputs, one for each action.

The final solution used two hidden layers of 64 and 128 neurons respectively.

```
QNetwork(
  (fc1): Linear(in_features=37, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=128, bias=True)
  (out): Linear(in_features=128, out_features=4, bias=True)
)
```

The following configurations were also tested

64 64 64 32 - Solved in 395 episodes

64 64 - Solved in 426 episodes

128 64 - Solved in 482 episodes

128 256 - Did not solve in under 1000 episodes

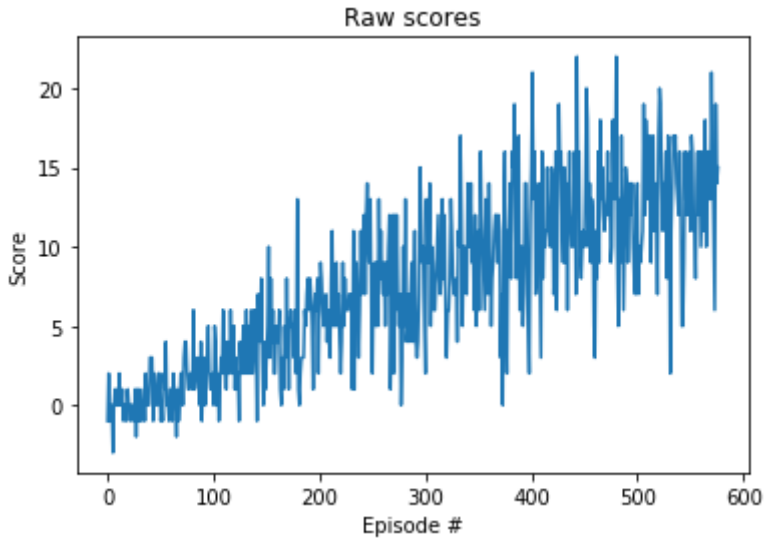
The interesting observation from this is that narrow and shallow networks worked best and the wide networks failed to converge.

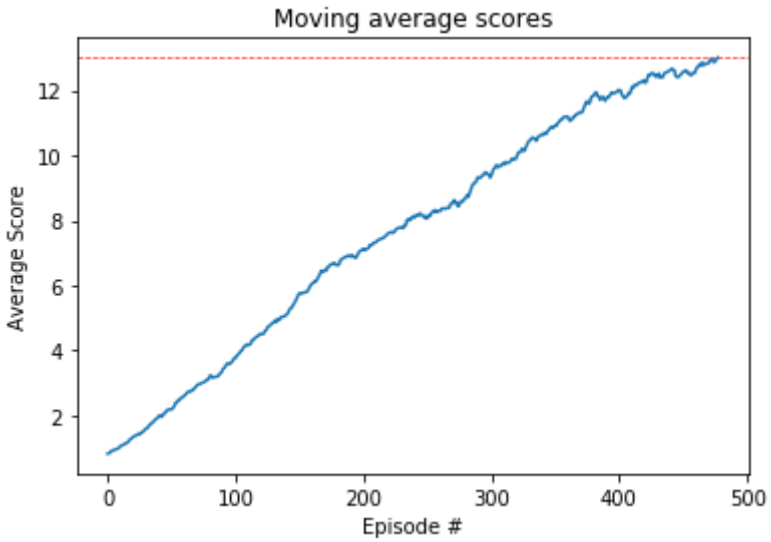
## Hyper parameters

```
11    BUFFER_SIZE = int(1e5)  # replay buffer size
12    BATCH_SIZE = 64         # minibatch size
13    GAMMA = 0.99            # discount factor
14    TAU = 1e-3              # for soft update of target parameters
15    LR = 5e-4               # learning rate
16    UPDATE_EVERY = 4        # how often to update the network
```

## Results

It was easy to achieve the goal score with almost no modifications. The consistent solving of the problem when training, and the low number of episodes required, was not expected.




```
Episode 100     Average Score: 0.83
Episode 200     Average Score: 3.77
Episode 300     Average Score: 7.13
Episode 400     Average Score: 9.39
Episode 500     Average Score: 12.01
Episode 577     Average Score: 13.01
Environment solved in 477 episodes!     Average Score: 13.01
```

## Ideas for future work

Overall, this environment was easily solved using a simple DQN DNN model with a very shallow configuration of layers.

I believe that implementation of Dueling DQNs or other enhancements to the standard implementation would be overkill.

Rather than extend the DQN itself, we would be very interested in exploring how bayesian sampling could be used (Thompson Sampling) in the policy evaluation step rather than using a Epsilon Greedy approach.