# Project 1 : Navigation

## Summary:

This project implements DQN Algorithm to train an agent to navigate (and collect bananas!) in a large, square world.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around 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 goal of our agent is to collect as many as yellow bananas while skipping blue bananas. A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana

## DQN Algorithm:

Deep Q-network (DQN) combines reinforcement learning with a class of artificial neural network known as deep neural networks. Notably, recent advances in deep neural networks, in which several layers of nodes are used to build up progressively more abstract representations of the data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data.

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values. We can address these instabilities with a novel variant of Q-learning, which uses two key ideas.

1. Experience Replay
2. Fixed Q Targets

**Experience Replay:**

When the agent interacts with the environment, the sequence of experience tuples can be highly correlated. The naive Q-learning algorithm that learns from each of these experience tuples in sequential order runs the risk of getting swayed by the effects of this correlation. By instead keeping track of a replay buffer and using experience replay to sample from the buffer at random, we can prevent action values from oscillating or diverging catastrophically.

The replay buffer contains a collection of experience tuples (S*S*, A*A*, R*R*, S'*S'*). The tuples are gradually added to the buffer as we are interacting with the environment.

The act of sampling a small batch of tuples from the replay buffer in order to learn is known as experience replay. In addition to breaking harmful correlations, experience replay allows us to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of our experience.

**Fixed Q-Targets:**

In Q-Learning, we *update a guess with a guess*, and this can potentially lead to harmful correlations. To avoid this, we can update the parameters w in the network *q*^ to better approximate the action value corresponding to state *S* and action A with the following update rule:

$$\Delta w = \alpha \cdot \overbrace{(\underbrace{R + \gamma \max_a \hat{q}(S', a, w^-)}_{\text{TD target}} - \underbrace{\hat{q}(S, A, w)}_{\text{old value}})}^{\text{TD error}} \nabla_w \hat{q}(S, A, w)$$

where *w*− are the weights of a separate target network that are not changed during the learning step, and (S, A, R, S′) is an experience tuple.

**HyperParameters and Results:**

BUFFER_SIZE = int(1e5) # replay buffer size

BATCH_SIZE = 64 # minibatch size

GAMMA = 0.99 # discount factor

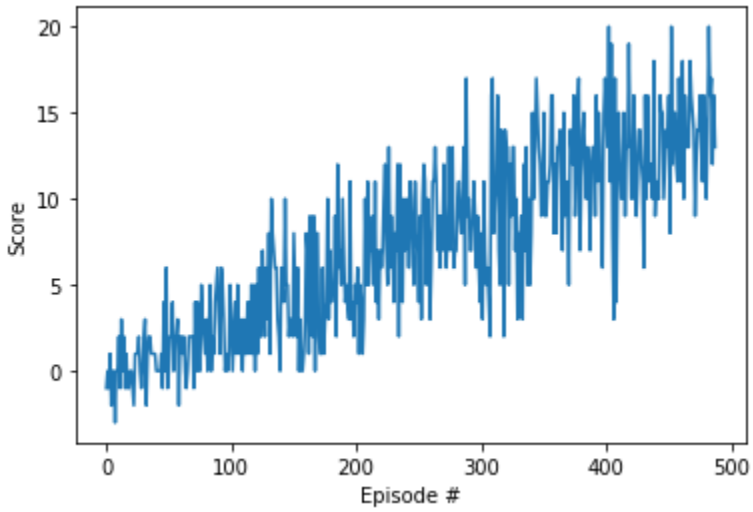
TAU = 1e-3 # for soft update of target parameters

LR = 5e-4 # learning rate

UPDATE_EVERY = 4 # how often to update the network

```
scores = dqn()
```

```
Episode 100     Average score:  1.18
Episode 200     Average score:  4.17
Episode 300     Average score:  7.86
Episode 400     Average score:  10.68
Episode 488     Average Score: 13.00
Environment solved in  388 episodes!    Average Score:  13.00
```




**Ideas for future work**

1. Double Deep Q Networks
2. Prioritized Experience Replay
3. Dueling Deep Q Networks
4. Learning from pixels
5. May be tuning hyper parameters rigorously.