This project implements the Google DeepMind DQN algorithm that was used to train an RL agent to play classic Atari games, using experience replay (to store experience tuples) and fixed targets method (two neural networks are utilized, a local one which is used to generate experience tuples for a fixed set of weights, and a target one which is gradually copied from the local one after each batch of training/weights adjustment)

Here is how it works

1. The RL agent receives a state vector from the Unity environment (37 long vector including position, velocity etc …)
2. An epsilon-greedy policy is applied to determine the next action (4 actions possible)
3. The action is sent to the Unity environment, and the agent gets back a next_state, reward and done (a completion flag indicating whether the episode is over or not)
4. Experience tuples (state, action, reward, next_state, done) are gathered within the experience replay pool
5. Once enough experience tuples are present in the experience replay, a random batch of experience tuples is sampled and used to train the local neural network, using MSE as the loss function.
6. Each given step (in our case 4), the local neural network weights are updated/saved into the target network
7. The state is next updated to the next_state and we carry on the process from step 2. until the episode is over

Details on the neural network architecture

As we don't deal with pixel images (as Google did) but rather with state vectors, we don't need any convolutional layer in our neural network, and we can hence use fully connected layers instead

For this project, we are simply using two fully connected layers with ReLU (as activation function)

Here is the network architecture for both local and target nets

```
      QNetwork(
        (fc1): Linear(in_features=37, out_features=64, bias=True)
        (fc2): Linear(in_features=64, out_features=64, bias=True)
        (fc3): Linear(in_features=64, out_features=4, bias=True)
)
```

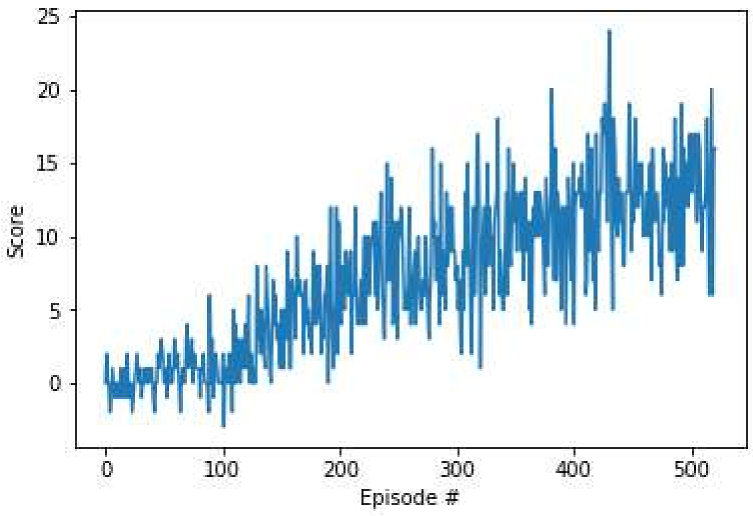
Hyperparameters

1. Batch size for training : 64
2. Gamma : 0.99
3. Epsilon : 1.0(initial) * 0.995(decay ratio)
4. Learning rate : 5e-4
5. Update frequency for neural networks (local->target) : 4

Results

With this configuration, we were able to solve the environment in only 421 episodes

```
Episode 100     Average Score: 0.55
Episode 200     Average Score: 3.82
Episode 300     Average Score: 7.74
Episode 400     Average Score: 10.01
Episode 500     Average Score: 12.59
Episode 521     Average Score: 13.00
Environment solved in 421 episodes!     Average Score: 13.00
```



<u>Improvement ideas</u>

One of the things we could certainly try it to use pixel data like googlemind did for their implementation. What it would take here is to modify the architecture of the local and the target neural networks to add two convolutional layers to process the images before proceeding with fully connected layers.

Another interesting idea would be to experiment with prioritized experience replay, to help focus on the most important transition and to optimize the learning.