Learning Algorithm:

At each time step, the agent observes the unity environment which depicts the current state (s_t), then the agent selects an action based upon stochastic policy

The loss function: $$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

For each time step, {s,a,r,s'} is regarded as a transition, while θ_i is the weight at i-th time step, with y_i^{DQN} to be the target value to be updated only at the end of each episode. In the following Algorithm 1, the symbol ϕ is in fact a function with respect to θ, the same as the above-mentioned loss function.

Algorithm 1: deep Q-learning with experience replay.

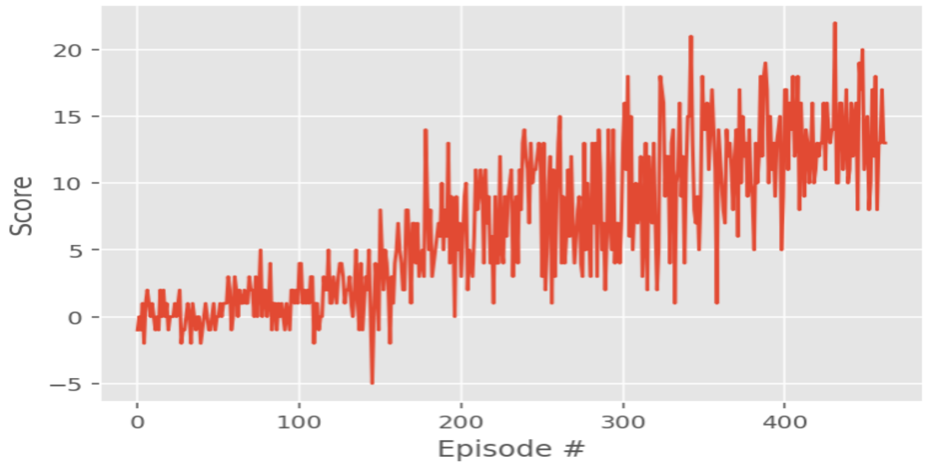
```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
**For** episode = 1, $M$ **do**
   Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
   **For** $t = 1$,T **do**
      With probability $\varepsilon$ select a random action $a_t$
      otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
      Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
      Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
      Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
      Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
      Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
      network parameters $\theta$
      Every $C$ steps reset $\hat{Q} = Q$
   **End For**
**End For**
```

Source

Plot of Rewards:

Plot and report the number of episodes needed to solve the environment:

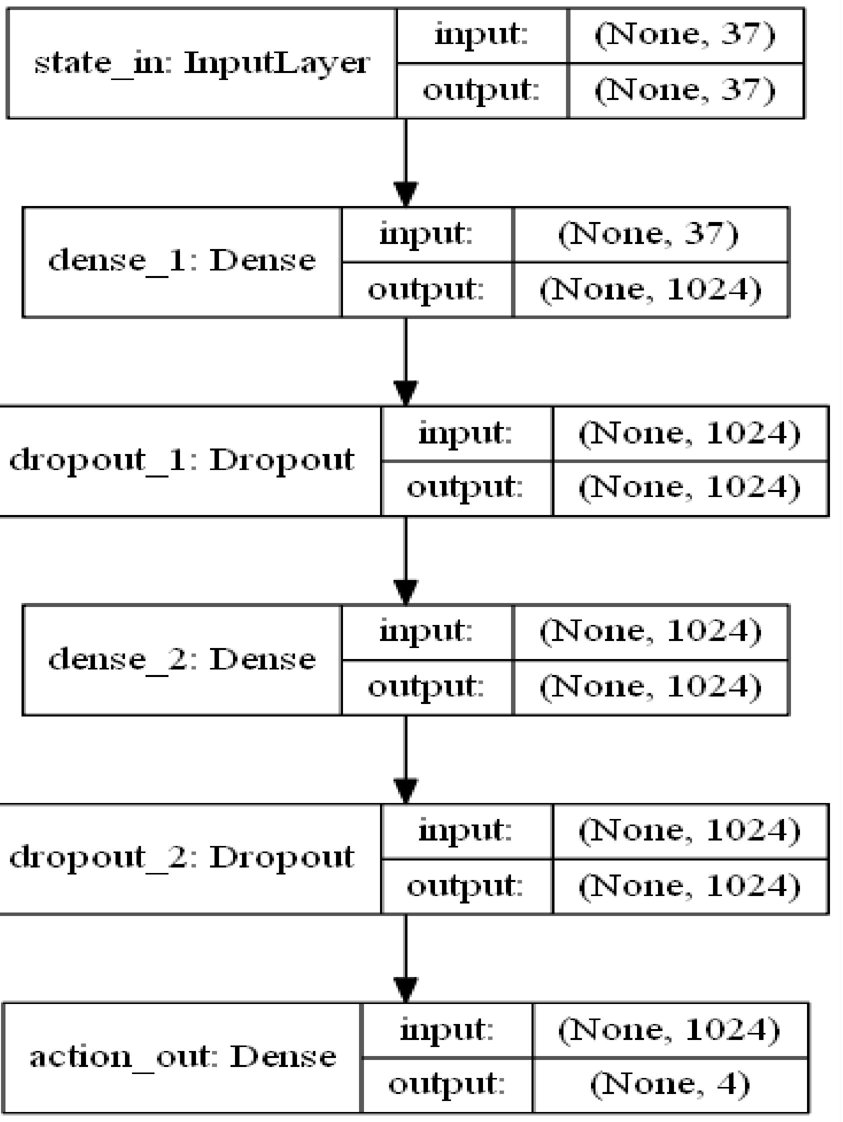
```
Episode 100     Average Score: 0.55
Episode 200     Average Score: 3.53
Episode 300     Average Score: 7.81
Episode 400     Average Score: 11.17
Episode 463     Average Score: 13.00
Environment solved in 363 episodes!     Average Score: 13.00

Total Training time = 9.9 min
```



The problem was solved by taking merely 363 episodes, comparable to a benchmark metric of 1800 episodes, within only 9.9 minutes training on an RTX 2060 GPU locally. See the result uploaded to YouTube of an agent collecting bananas efficiently.
Parameters are as follows (using Keras):

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
state_in (InputLayer)        (None, 37)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              38912
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
action_out (Dense)           (None, 4)                 4100
=================================================================
Total params: 1,092,612
Trainable params: 1,092,612
Non-trainable params: 0
_________________________________________________________________
```

Here's a brief architecture of DQN model (using Keras backend):



Note that initially in hidden layers only 64 units have been assigned for each layer, and the model fails to converge at an early stage. After adding units up to 1024 for each the performance still cannot converge ideally before 1800 episodes, but it is discovered that the score grew faster at early episodes than the 64-unit-layers model architecture, which might be caused by too much correlations produced between different state samples. As a result, two Dropout layers were introduced with 0.2 possibility for each to be dropped out immediately after each hidden layer as a solution. The model finally yields to a much better result as it not only converges faster than previous model structures, but also completed training by acquiring an average score of +13 points within 363 episodes, even better than many incumbent models.

Ideas of Future Work:

1. Lessening numbers of parameters might help increase the efficiency of Q-Network.
2. Integrating all DQNs in fashion into the Rainbow will be highly likely to optimize the final performance.