The project work implements a method called Q-Network, which is able to combine reinforcement learning with a class of artificial neural network known as deep neural networks. 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. The implementation addresses the issue of instabilities with a variant of Q-learning, which uses two ideas: experience replay and target network. The project work implement another algorithm called, double learning, to alleviate the maximization bias – the overestimate of the Q function value. This report gives implementation details on network architecture, hyper parameters, main algorithm, and training results. For more theoretical discussions and mathematic foundations of subjects, please refer to the original papers, Human-level control through deep reinforcement learning and Deep Reinforcement Learning with Double Q-learning.

Model Architecture

Target and training neural networks use the same network architecture. The network model has three layers, two hidden layers and one output layer. Both hidden layers have 64 units and use Relu activation. The output layer has 4 units, corresponding to predicted Q values of four actions, move forward, move backward, turn left, and turn right. The input is a 37-dimension vector, representing environment state space of 37 dimensions, containing agent's velocity, along with ray-based perception of objects around agent's forward direction.
The work uses mini-batch gradient decent training algorithm with batch size 64 and learning rate 0.0005 (5e-4). The model parameters are optimized with the Adam method.

Experience Replay

Experience replay is an idea to stabilize the learning algorithm and speed up the learning process. The work uses memory buffer size 100,000. The learning algorithm saves every experience (s, a, r, s', done) into the memory. The memory refreshes old experiences with new ones when the memory buffer is full. The learning algorithm samples a batch of experiences from the memory every 4 steps.

Target Network Update

The target network is another idea to stabilize the learning algorithm and speed up the learning process. The weights of the target network are fixed for every 4 training steps. They are updated from the training network using following soft update formula below. The value of tau is 0.001 (1e-3).

```
target_weights = tau * trainig_weights + (1-tau) * target_weights
```

Double Learing

One problem in the DQN algorithm is that the agent tends to overestimate the Q function value, due to the *max* in the formula used to set targets:

$$Q^{-}(s,a) = r + gamma * max_{a}Q^{-}(s',a)$$

Double learning is one of solutions to the problem. In the work, it uses the training Q function to determine the maximizing action and target Q function to estimate its value.

$$Q^{-}(s,a) = r + gamma * Q^{-}(s',argmax_{a}Q(s'))$$

It was proven that by decoupling the maximizing action from its value in this way, one can indeed eliminate the maximization bias.

Agent Learning Algorithm

The Q learning algorithm trains an agent by interacting it with the environment. This work uses Unity reinforcement learning environment. Each iteration, the agent takes an action with epsilon-greedy policy. The policy chooses an action randomly with epsilon probability, and take an greedy action with 1 – epsilon probability. The algorithm uses a decay schedule for the epsilon. The value of epsilon starts with 1. It decays 0.995 every step until it reaches 0.1, which is the floor of the value.

Each iteration, the algorithm saves an experience, the state, reward, and terminate condition which are returned from the environment, into the memory. Every 4 iterations, it samples a batch of experiences from the memory, trains the training network, and update the weights of target network. It uses mean square error loss function to minimize the distance between Q(s, a), action value of training network and $Q^-(s,a)$, action value of target network. gamma is the discount factor and equal to 0.99.

$$Q^-(s,a) = r + gamma * Q^-(s', argmax_a Q(s'))$$

$$Loss = MSE(Q^-(s,a) - Q(s,a))$$

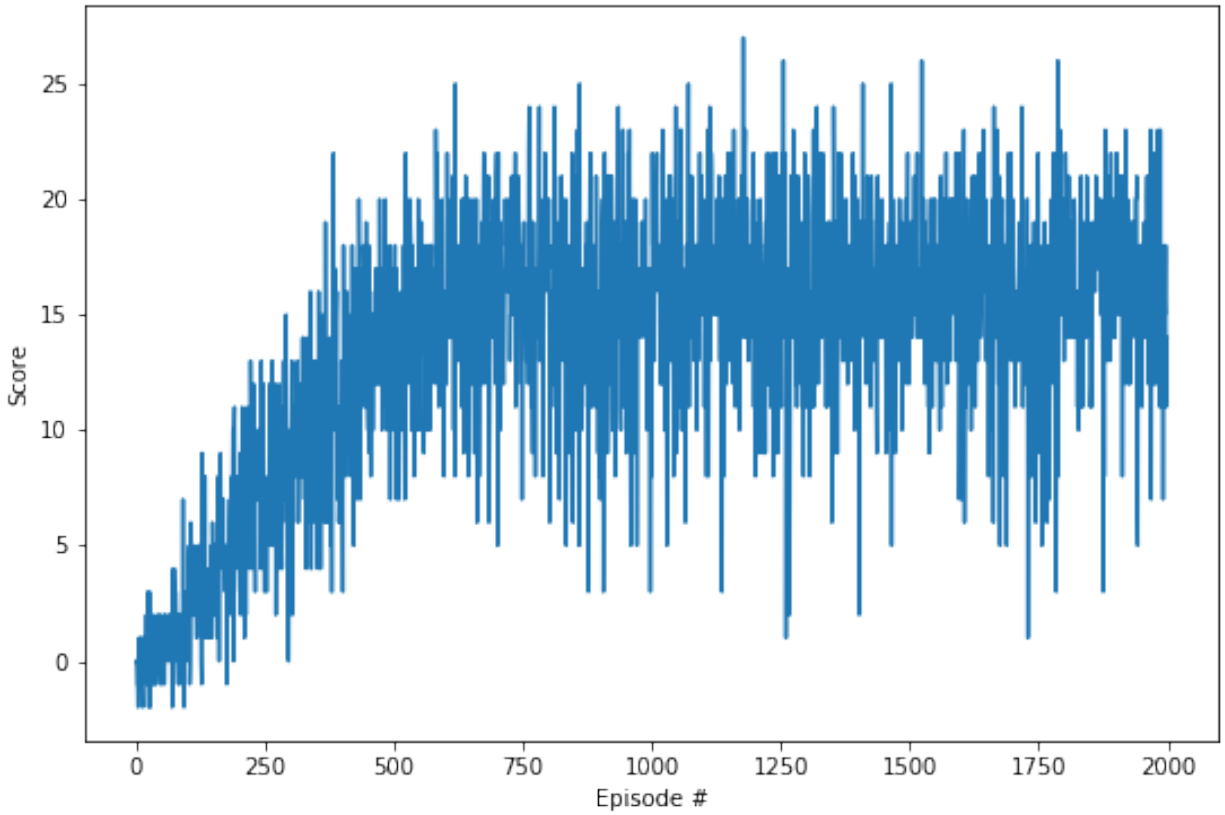
Pseudo code of Q-learning with experience replay and double learning

The training process runs 2000 episodes. Each episode, it iterates 1000 times. 'done' represents terminal condition.

Initialize replay memory D to capacity 100,000
Initialize action-value function Q with random weights theta
Initialize target action-value function Q^- with weights $theta^- = theta$
For episode from 1 to 2000 do
    Initialize sequence s1 = {x1}
    For t from 1 to 100 do
        With probability epsilon select a random action a_t
        otherwise select $a_t = argmax_a Q(s_t, theta)$
        Environment take the action a_t and return reward r_t, state s_{t+1}, and done
        Store the experience $(s_t, a_t, r_t, s_{t+1}, done)$ in D
        If t % 4 == 0 then
            Sample random minibatch of experiences $(s_j, a_j, r_j, s_{j+1}, done)$ from D
            Set $y_j = r_j + gamma * Q^-(s_{j+1}, argmax_a Q(s_{j+1}))$
            Perform a gradient descent step on $MSE(y_j - Q(s_j, a_j, theta))^2$ with respect to the training network weights
            soft update target network weights with training network weights
        End If
    End For
End For

Training Results

The learning algorithm implemented is able to solve the environment (average reward above 13 over 100 episodes) in 373 episodes. Below is the plot of rewards per episode over 2009 episodes.



Future Works:

To apply the implementation to the large scale and more complex problems, the work can be improved in following areas:

- To implement prioritized experience replay and dueling DQN.
- To use convolution neural network as the model to train the agent with the input of raw pixels.
- To refract the code, so all hyper parameters can be learned with grid search.