Project 1: Navigation

Ke Wang

October 2018

1 Introduction

In this project, I developed a deep Q-network algirhtm and its variants to solve a banana collection problem in Unity environment.

1.1 Environment

In this environment, an agent is able to navigate (and collect bananas!) in a large, square world.

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.

1.2 Solving criterion

The task is episodic, and in order to solve the environment, your agent must get an average score of +13 over 100 consecutive episodes.

2 Algorithm

Two algorithms have been developed. An original Deep Q-Network is described as Algorithm1. And a double DQN algorithm is developed where the action of next state is selected by Q_local network and the Q value of next state is evaluated by Q_target network.

3 Neural network

I proposed a three-layer fully connected linear neural network. The first two layers are hidden layers with a size N. The last layer is the output layer with

Algorithm 1 Deep Q-Network algorithm

Initialize Q-value memory Q-buffer
Initialize two Q-networks: target and local with random weights
Agent.tstep = 0
for $episodei = 1$ to M **do**
    Reset Environment
    for $t = 1$ to T **do**
        $R_t = env(S_t)$
        $A_t = argmax(Q_local(S_t))$
        $Q\_target\_next = max(Q_target(S_t))$
        $Q\_target = R_t + \gamma * Q\_target\_next$
        $Q\_expected = Q_local(S_t)$
        $TD\_error = |Q_expected - Q_target|$
        Q-buffer.push (S_t, A_t, R_t)
        if $Mod(Agent.tstep, UPDATE\_EVERY) = 0$ **then**
            Sample Q-buffer
            Calculate TD-errors
            Back propagate Q_local
            Update Q_target with Q_local

Algorithm 2 Double Deep Q-Network algorithm

Initialize Q-value memory Q-buffer
Initialize two Q-networks: target and local with random weights
Agent.tstep = 0
for $episodei = 1$ to M **do**
    Reset Environment
    for $t = 1$ to T **do**
        $R_t = env(S_t)$
        $A_t = argmax(Q_target(S_t))$
        $A_t^{next} = argmax(Q_local(S_t))$
        $Q\_target\_next = Q_local(S_t)[A_t^{next}]$
        $Q\_target = R_t + \gamma * Q\_target\_next$
        $Q\_expected = Q_local(S_t)$
        $TD\_error = |Q_expected - Q_target|$
        Q-buffer.push (S_t, A_t, R_t)
        if $Mod(Agent.tstep, UPDATE\_EVERY) = 0$ **then**
            Sample Q-buffer
            Calculate TD-errors
            Back propagate Q_local
            Update Q_target with Q_local

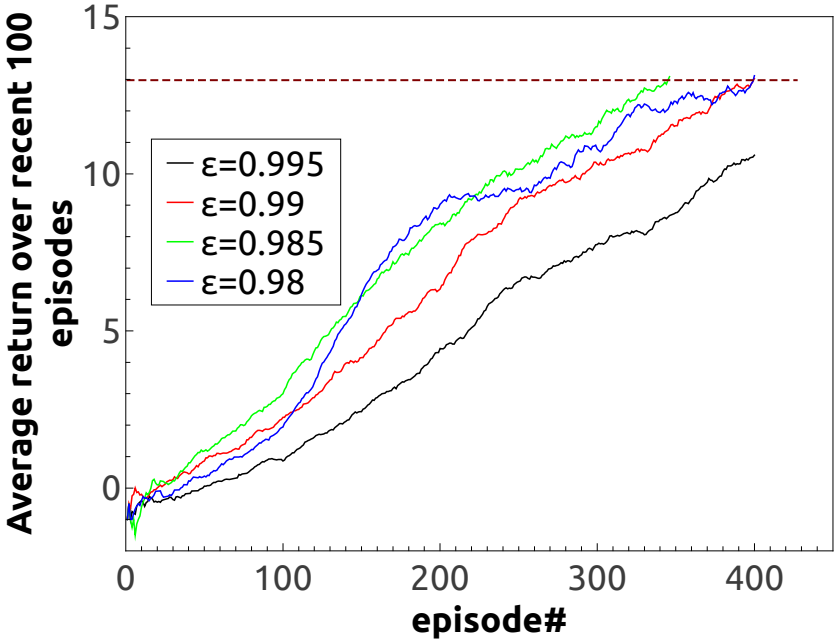

Figure 1: The effect of different learning rate decay

size 4. The first two layers adopt relu as their activation function.

4 Results

In this section, grid search of parameters are carried out to find the find the best parameter set. The convergence curves are shown and compared.

The initial set of parameters are: learning batch size B=64, neural network size of each hidden layer is $N = 64$, $\gamma = 0.99$, network update factor $\tau = 0.001$, network learning rate LR=0.0005, network training interval = 4.

Figure 1 compares different epsilon decay rate in epsilon-greedy strategy. $\epsilon = 0.985$ leads to the best result.

Figure 2 compares the neural network with different sizes. $N = 48$ leads to the best result.

I also found that network update factor$\tau = 0.002$ and network training interval = 3 will lead to a better results.

The final best set of parameters are: learning batch size B=64, neural network size of each hidden layer is $N = 48$, $\gamma = 0.99$, network update factor$\tau = 0.002$, network learning rate LR=0.0005, network training interval = 3, epsilon decay rate in epsilon-greedy strategy $\epsilon = 0.985$. The best parameters set lead to a convergence at 288 episode.

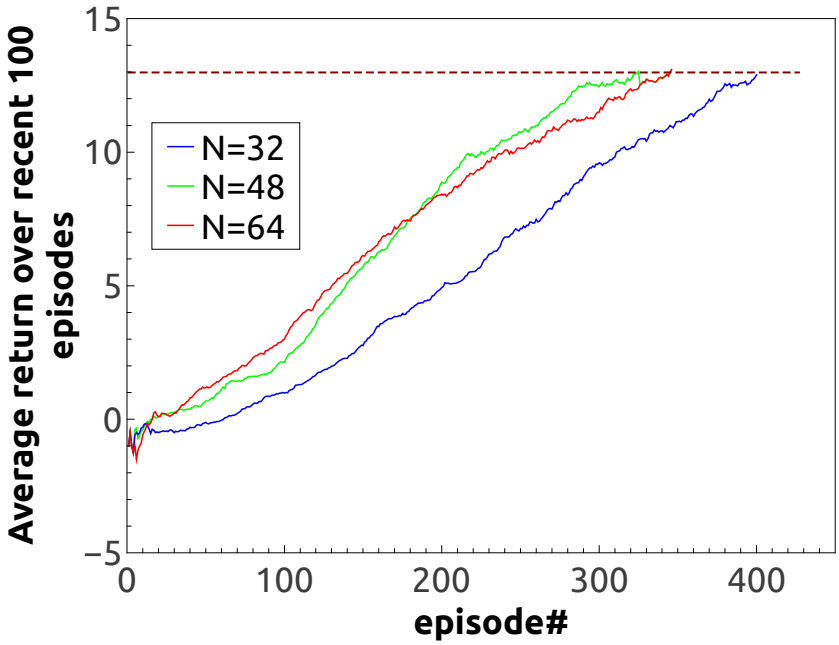

Figure 2: The effect of neuron numbers of each layer

Figure 3 compare DQN and DDQN methods. Original DQN outperforms DDQN in this study.

5 Conclusion

Two deep Q-network algorithms are developed to solve a banana collection problem in Unity environment. Parameters of the algorithms are tuned to get the best result which converges at 288 episodes. Original DQN outperforms DDQN in this study.

Next step, a dueling DQN and prioritized experience replay will be implemented to further improve the performance.

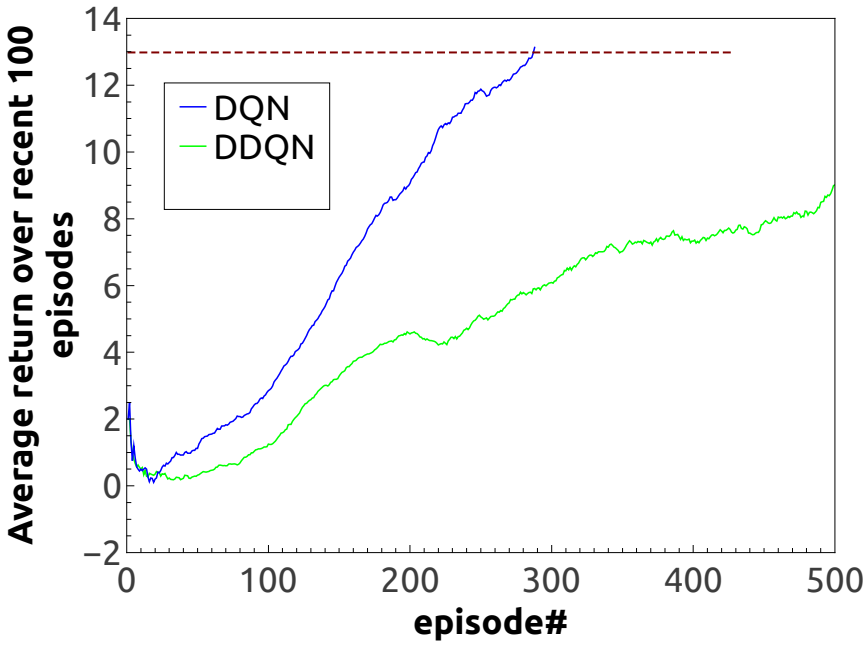

Figure 3: Comparison of original DQN and Double DQN