Project Report: Navigation

Rui Ding

March 2020

1 Introduction

In this project, I applied a Double DQN agent to play the banana game and complete the task of navigation and collecting yellow bananas while avoiding blue bananas. The environment is built through Unity.

2 Description of Environment

In this game, 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 the 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 forward, backward, left, and right. The environment is episodic and an average score is computed every 100 episodes.

3 Algorithm

In this project, I applied a Double-DQN agent to solve the task. A replay buffer is used to store prior experiences with capacity 100000 and mini-batch learning is employed while the neural networks update their weights.

We start by building two neural networks for computing the Q values for each state and action. Both the local and the target networks have two hidden layers of size 64 and an output size of 4 corresponding to each action. The activation function is chosen to be ReLu.

After gathering enough sample experiences, the agent samples a mini-batch of 64 experiences every learning step and the network values are updated every 4 steps. To act according to the current Q estimates, the agent uses a ϵ-greedy policy which takes the action corresponding to the highest Q value for the current state with probability $1-\epsilon$ and takes a random action with probability

ϵ. The parameter ϵ starts from 1.0 and decays every subsequent episode by 0.995 until it hits the threshold 0.01.

For the learning step the agents will compute the differences between the Q targets for current states from the batch experiences and the expected Q values using the local model at the current states.

We know that the DQN agent with fixed targets sets the target Q value for each experience tuple $e = (S_t, A_t, r_t, S_{t+1})$ as(while episode not done):

$$Q_{target} = r_t + \gamma * max_a Q(S_{t+1}, a, w^{target})$$

and

$$Q_{target} = r_t$$

if episode is done.

Here w^{target} are the weights for the target network. The expected Q values are:

$$Q_{expected} = Q(S_t, A_t, w)$$

where w are the weights for the local network.

In my implementation a Double-DQN is used so the Q target now has the expression:

$$Q_{target} = r_t + \gamma * Q(S_{t+1}, argmax_a(Q(S_{t+1}, a, w)), w^{target})$$

Now having computed these values our local Q network updates its weights w by minimizing the loss:

$$L = \sum_e (r_t + \gamma * Q(S_{t+1}, argmax_a(Q(S_{t+1}, a, w)), w^{target}) - Q(S_t, A_t, w))^2$$

Here the parameter γ is 0.99 and the neural network is updated using the Adam optimizer with a learning rate of 0.0005.

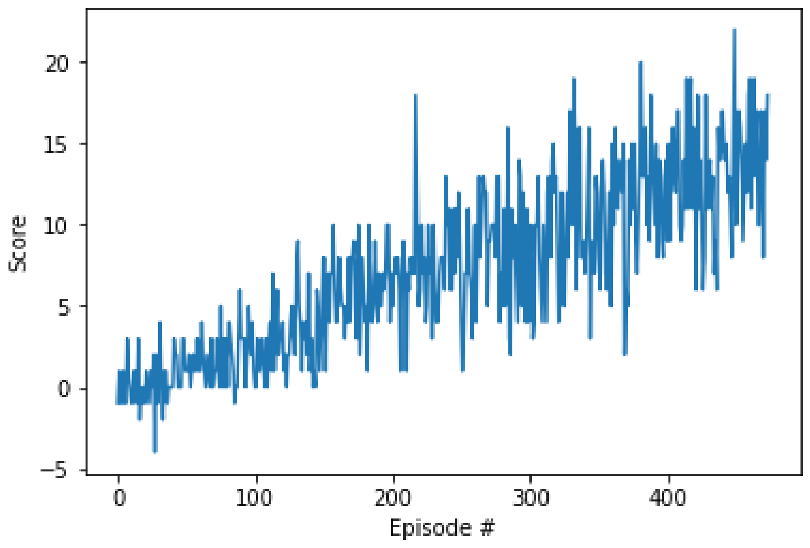
The target network weights w^{target} are updated following a soft update rule:

$$w^{target} = \tau * w + (1 - \tau) * w^{target}$$

where τ is taken to be 0.001.

4 Training Results

The number of episodes is set to be 2000 and each episode will be terminated if it is not terminated after 1000 time steps. Every 100 episodes the average score will be printed and the last 100 episodes will serve as a test score for the previously trained Double-DQN which should exceed 13.0 for the agent to be considered successful. For my algorithm, it takes 374 steps to solve this problem and the average score over the next 100 episodes is 13.06. The average score plot is shown below. The Double-DQN agent is able to solve the environment very quickly.



Finally, the weights of the local Q network is saved as the file ”model.pth”.

5 Future Extensions

One important feature I would like to implement is prioritized experience replay. This would hopefully increase the learning speed of the Double-DQN agent. A good combination of techniques like the Rainbow algorithm is also something of interest.

6 References

1. Course Materials and Codes

2. Human-level control through deep reinforcement learning, Mnih et. al, 2015.

3. Prioritized Experience Replay, Schaul et. al., 2015.

4. Deep Reinforcement Learning with Double Q-learning, van Hasselt et. al., 2016.