Project 1: Navigation

Yafeng Cheng

April 3, 2019

This is a report to describe the implementation of the deep Q-learning algorithm to solve the navigation project.

1 The project introduction

The environment of the project is created using the Unity Machine Learning Agents v0.4. To create the environment on the local machine, we can follow the instruction on the following link .

The target of the project is to train an agent 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 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 a 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.

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

2 Algorithm

To algorithm used in this project is a Deep Q-network with ϵ-greedy policy. The algorithm works as following:

1. Initialize replay buffer $\mathbf{B}$ with size N_B.
2. Initialize a local action-value function $\mathbf{M}_l$ with a neural network model. Randomly set the model weights to be w.
3. Initialize a target action-value function $\mathbf{M}_t$ with a neural network model. Randomly set the model weights to be w^-.
4. Initialize the ϵ for the ϵ-greedy policy as 1, the decay of ϵ as $\rho_\epsilon = 0.995$ and set the lower bound of ϵ as $\epsilon_{min} = 0.1$.
5. Repeat the following steps $N_{episode}$ times. In each episode, set the score to be 0:
   (a) play n_step in the game until reach the predefined maximum number of steps (1000).
   (b) In each step, collect state, action, reward and new station as one tuple using the local action-value function $\mathbf{M}_l$

(c) Update the target action-value function $\mathbf{M}_t$ once every 4 steps using 64 samples randomly selected from the replay buffer B if we have enough tuples. This is a soft update with proportion $\tau = 0.0005$.

(d) add the reward to score to update score.

6. Check the performance of the target model by checking the average of the last 100 episodes.

2.1 The architecture of the neural networks

The core of the algorithm is the neural network models. Both $\mathbf{M}_l$ and $\mathbf{M}_t$ have the same architecture. In the notebook, a vanilla neural network with three hidden layers and 32,64,32 hidden units for each of the layers, respectively. The model of the choice is based on the performances of a number of different hidden layers and the number of hidden units.

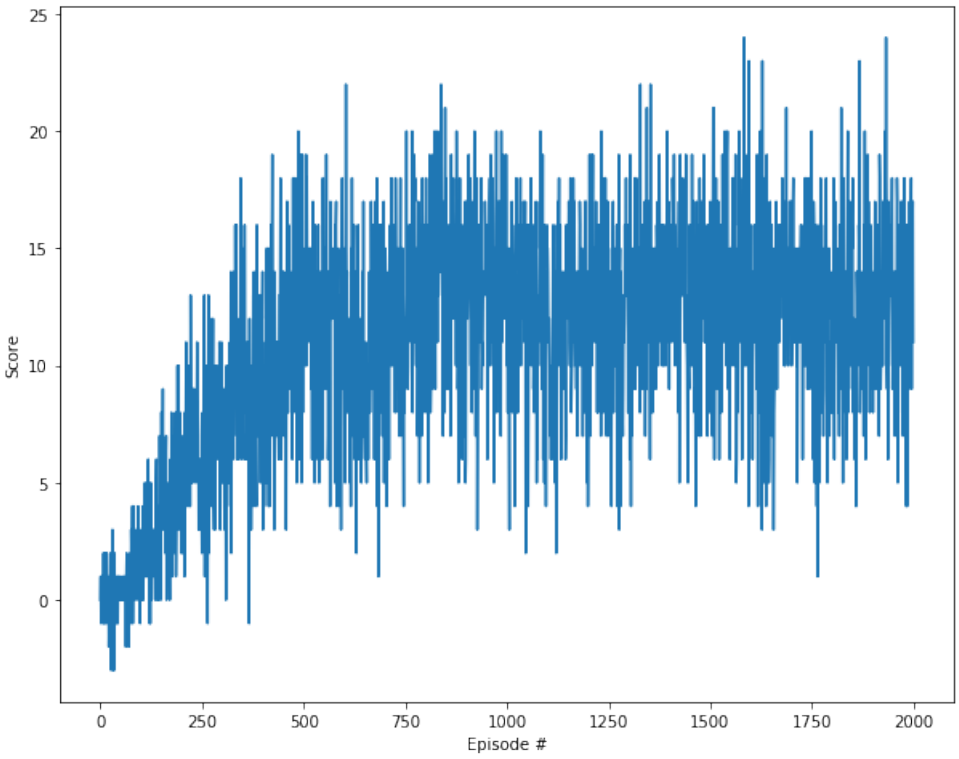
2.2 The choice of hyper-parameters

The hyperparameters used in the algorithm are listed in the following table:

argument	value	explanation
n_episodes	2000	
tau	0.0005	update rate?
LR	5e-4	learning rate
eps_decay	0.995	decay rate for epsilon greedy policy gradient
eps_start	1	starting rate for epsilon greedy policy gradient
eps_end	0.1	end rate for epsilon greedy policy gradient
BATCH_SIZE	256	batch size
gamma	0.99	decay rate for future return
max_t	1000	maximum number of steps in one episode
hiden_layer_size	[32,64,32]	neural network hidden layer and cells in each layer
hiden_layer_act	[nn.ReLU(),nn.ReLU(),nn.ReLU()]	neural network activation function in each hidden layer
memory_size	10000	replay buffer size
update_every_t	4	learn after every x generation steps
n_updates	1	number of updates every x generation steps
trace	100	print out summary every trace episodes
seed	1	

3 Outcome

The plot of the score from all the episodes is shown as following:



4 Conclusion and future work

The algorithm with the above setting successfully reached the required average score. However, during the learning process, there is a noticeable high variance on the score from each episode. The result is also sensitive to the listed hyper-parameters, including the seed. This implies that the settings have a lot of room for improvement.

The algorithm only implemented reply buffer, fixed Q-target, double DQN, and soft update. There are a number of other adjustments we can implement in this project.