

**Liverpool University**

**COMP532 Machine Learning and Bioinspired Optimization**

**Assignment 2**

**Deep Reinforcement Learning**

**Computer science**

**Cheng Bi 201352473**

**Yuefeng Liang 201350099**

**Rui Zhang 201255800**

**Zheng Yan 201336720**

**11.4.2019**

**Content**

[**1. Import an OpenAI game 20% 1**](#_Toc5811267)

[**2. Creating a network 20% 2**](#_Toc5811268)

[**3. Connection of the game to the network 10% 5**](#_Toc5811269)

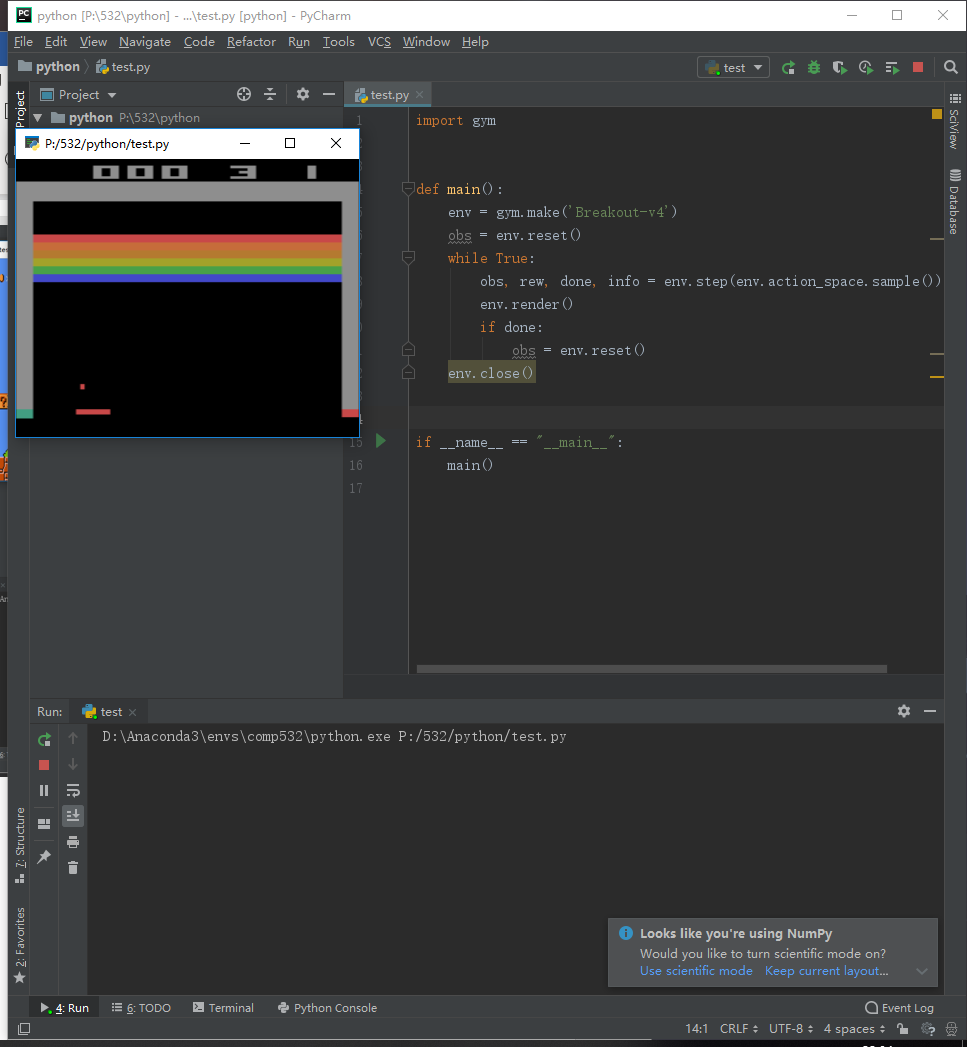
[**4. Deep reinforcement learning model 30% 8**](#_Toc5811270)

[**5. Experimental results 20% 9**](#_Toc5811271)

1. Import an OpenAI game 20%

In this assignment, we choose to use pure gym environment to train an agent to play an Atari video game.

At beginning, in order to test gym environment, we import the game Breakout-v4.

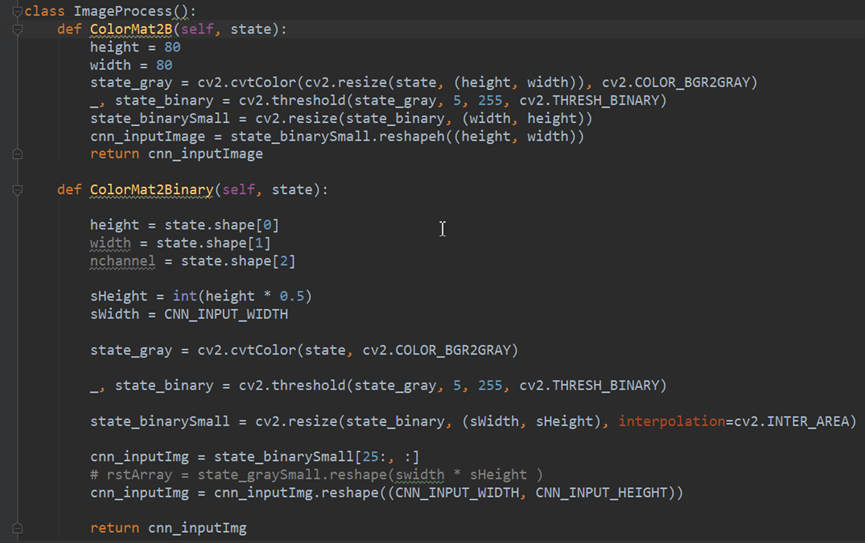


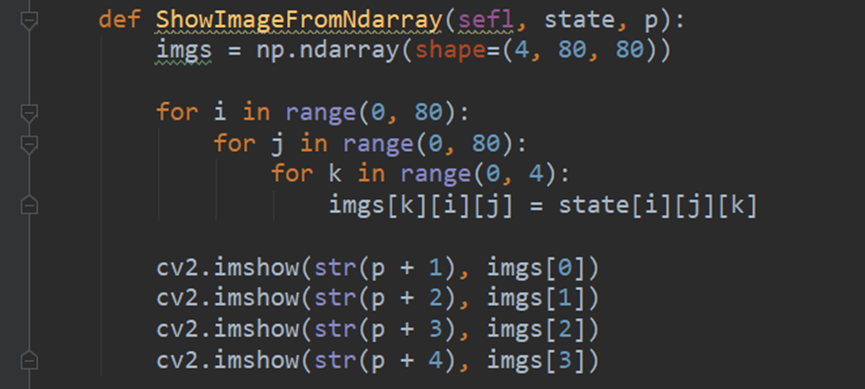
**Figure1. test gym env**

We could see from figure1, the main method creates an agent randomly, thus this agent just run stochastically. The score usually less than 5.

**Pre-process:**

This part is the game Image process. We use three method to analysis the game image and convert it to the 2D array.

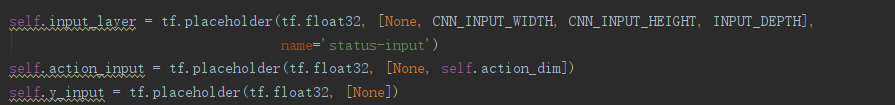




1. Creating a network 20%

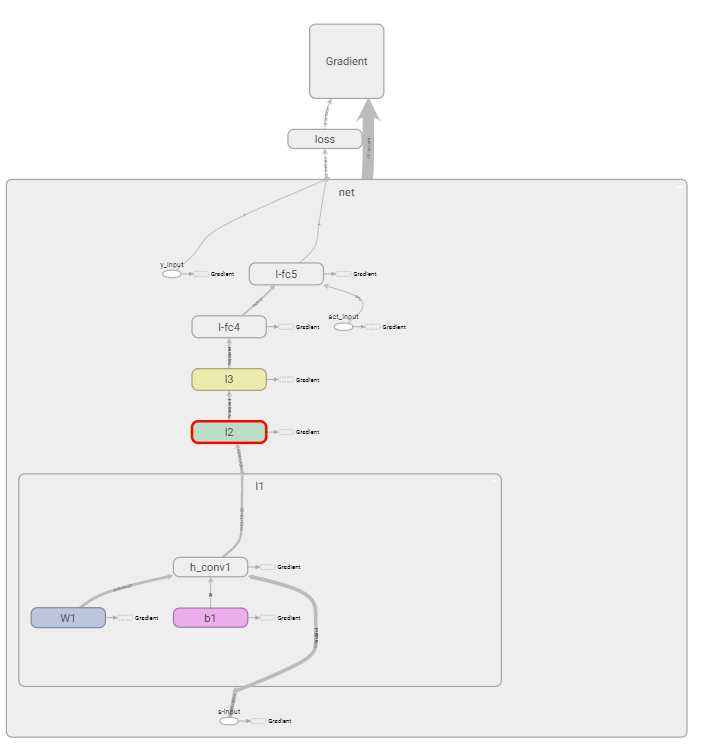
About this assignment, we want to use DQN to train an agent to play the game. Thus, we need to create the Q-network.

We used the create\_network method to build the net and initialize some value and place them in the tensorboard firstly.

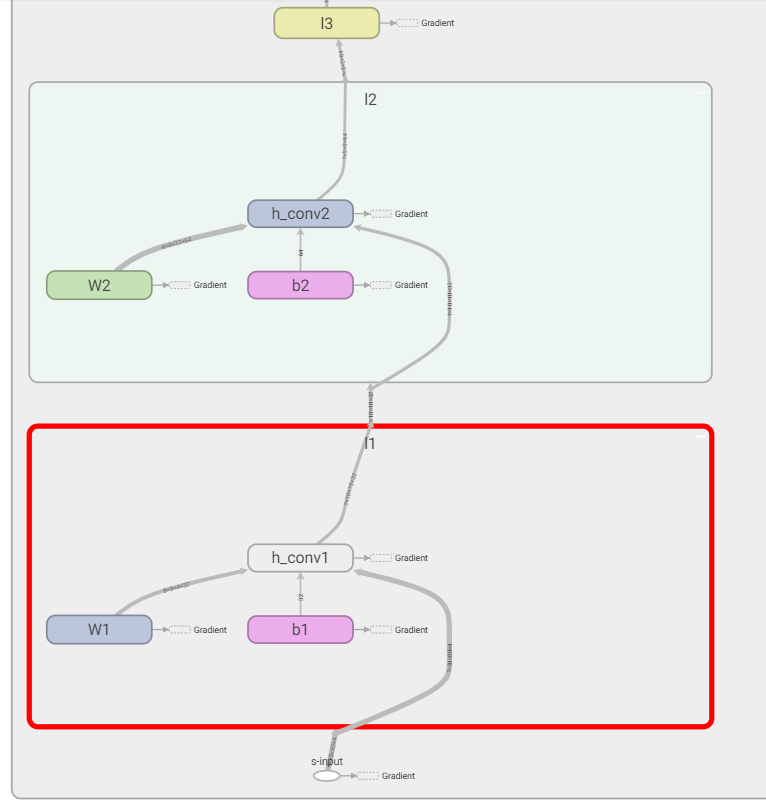


**Figure2. codes in the create\_network method**

Then, we define the three convolutional layers( l1,l2,l3 ) including hidden layers followed by two fully connected layer( l-fc4,l-fc5 ) to output each valid action. After each convolutional layer, we use max pooling method to reduce the dimensions of features. Fully connected layers are used to connect all the features as output value.



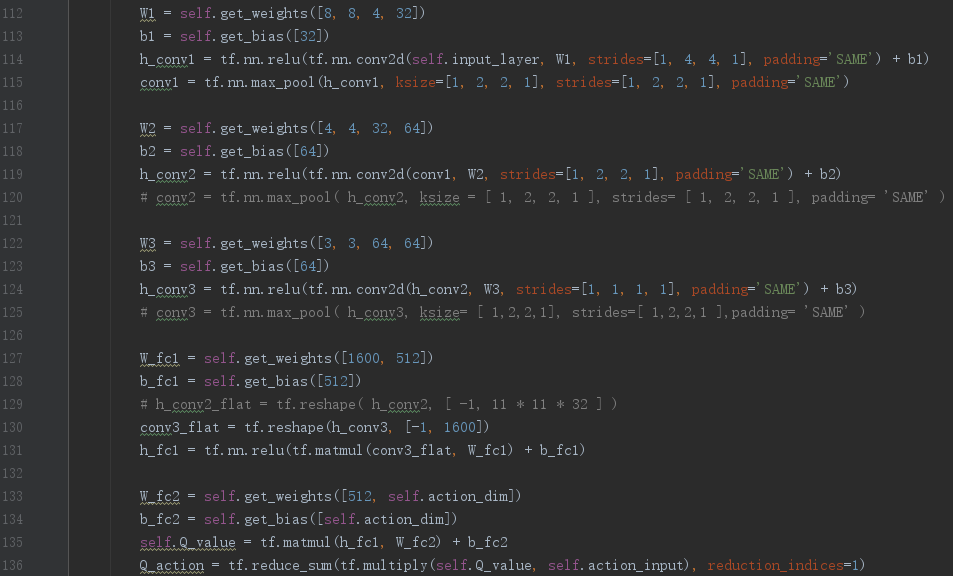
**Figure3. Structure of five convolutional layers**



**Figure4. The structure of hidden layers**

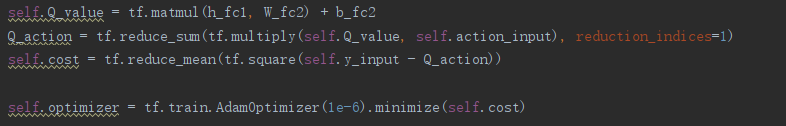
As shown in the figure4, in each convolutional layer, we sum the weight , bias and the input values to the hidden convolution layer. After that, it output the values to the next layer. Each convolutional layer has a hidden layer.

The code below shows that building the network including weights and bias of each layer, as well as max pooling method.



**Figure5. The codes for building network**

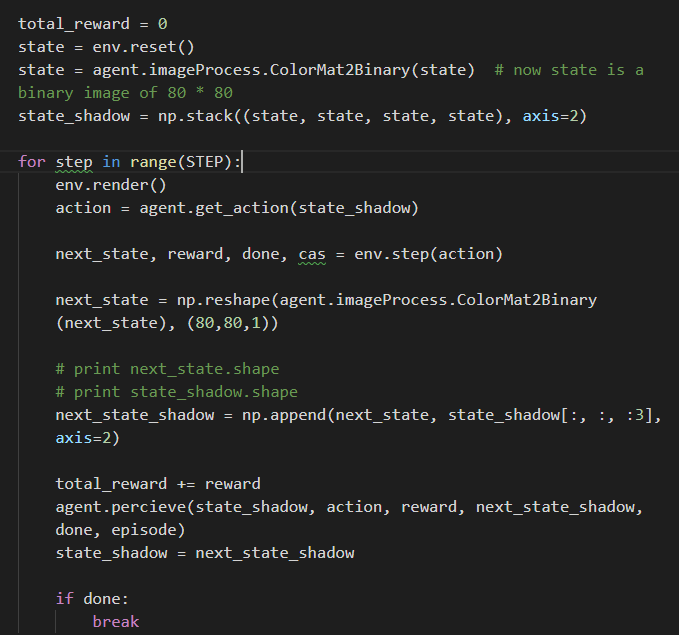
Calculate the loss and compute the mean of loss between y\_input and Q\_action. Then calculate the Gradient descent. The loss and the Gradient descent will use to upgrade the parameters of the network.



**Figure6. Methods of calculating the loss and the Gradient descent**

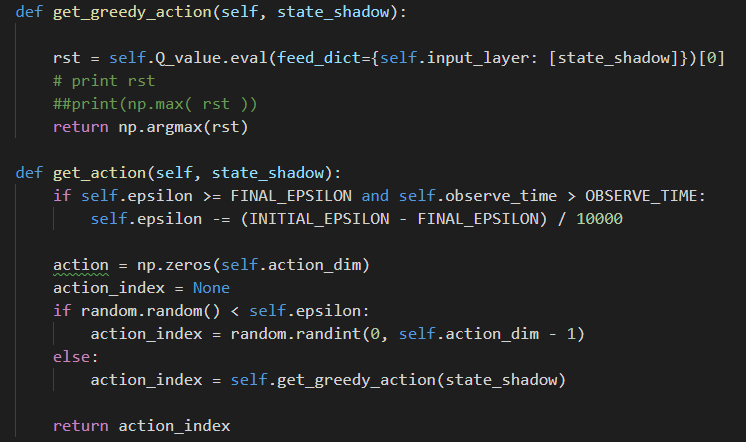
1. Connection of the game to the network 10%

To connect the game agent to our network, as shown in the first figure below. At very beginning, we define *state\_shadow* as initial state, then we choose the action of agent according to this state by using ε-greedy method (the second figure below) and get next state of agent. We set a method called *perceive* to observe the game and decide if the agent need to be trained by Neural Network (the third figure below).



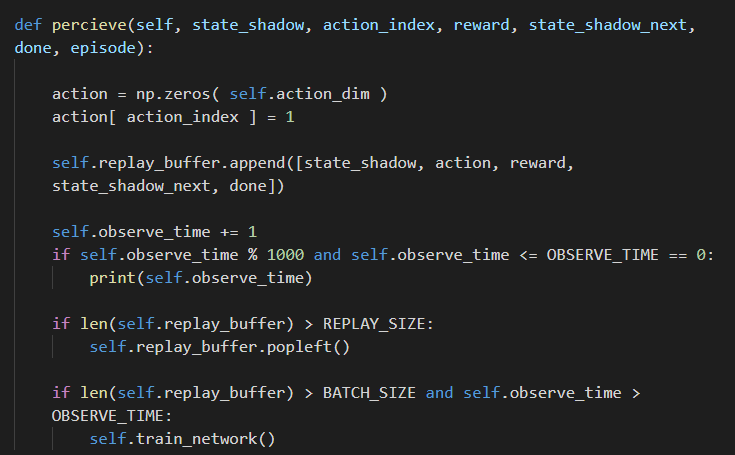
**Figure7. Input the values to network and record the next sate**

The get\_greedy\_action method was used to return the actions under the condition of usinggreedy, while the get\_action method chose the actions after using the Q-network.



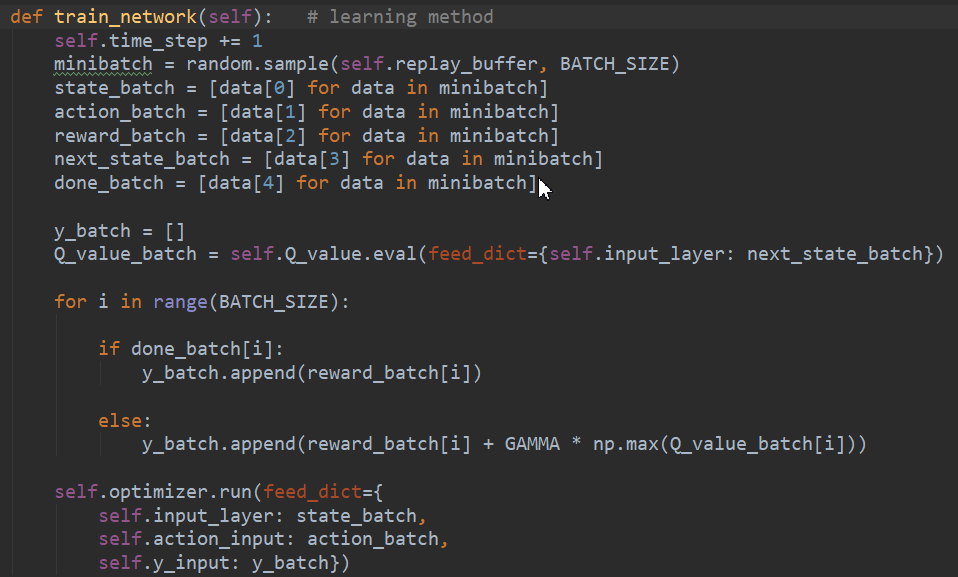
**Figure8 . Choose the action**

The precieve method was used to build the memory pool. The pool stores the values from network. When the agent plays a short time, it could not learn. Only when there is some memory in the pool, the agent could learn from these knowledge.



**Figure9. Memory pool**

The train\_method used to train the agent to play the game. It is the main learn method for the agent. The agent learns from the Q-network and update its action and state. Minibatch is the size of the memory extracted from the memory pool randomly. We could get the input\_layer, action\_layer and y\_input from the optimizer.run method, after the y\_input upgrading.



**Figure10. Learning method**

1. Deep reinforcement learning model 30%

Deep Q-Network is the model which uses Q-Learning to determine the Loss Function to find the gradient and update the parameters using random gradient descent. The update formula of Q-Learning is:

and the Loss Function is:

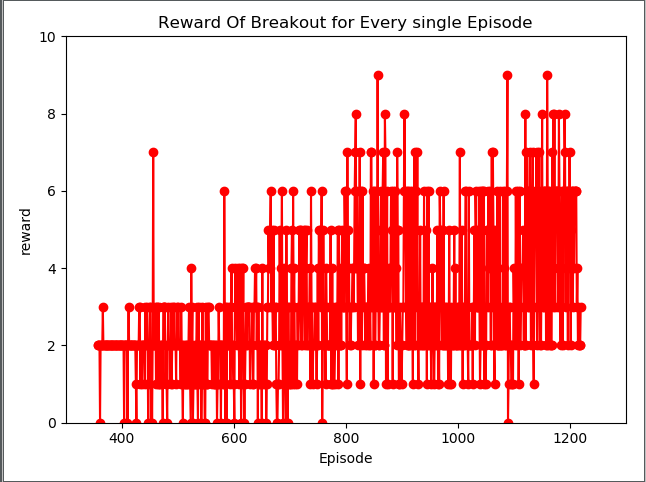
.

The function of the experience pool is mainly to solve the problem of correlation and non-static distribution. The specific method is to transfer the sample obtained by interacting each time step agent with the environment, save to the playback memory unit, then randomly take out some (minibatch) to train.

The neural network also includes in the model. The input to the neural network consists of an 84 × 84 × 4 image produced by the preprocessing map . The first hidden layer convolves 32 filters of 8 × 8 with stride 4 with the input image and applies a rectifier nonlinearity. The second hidden layer convolves 64 filters of 4 × 4 with stride 2, again followed by a rectifier nonlinearity. This is followed by a third convolutional layer that convolves 64 filters of 3 × 3 with stride 1 followed by a rectifier. The final hidden layer is fully-connected and consists of 512 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions varied between 4 and 18 on the games we considered.

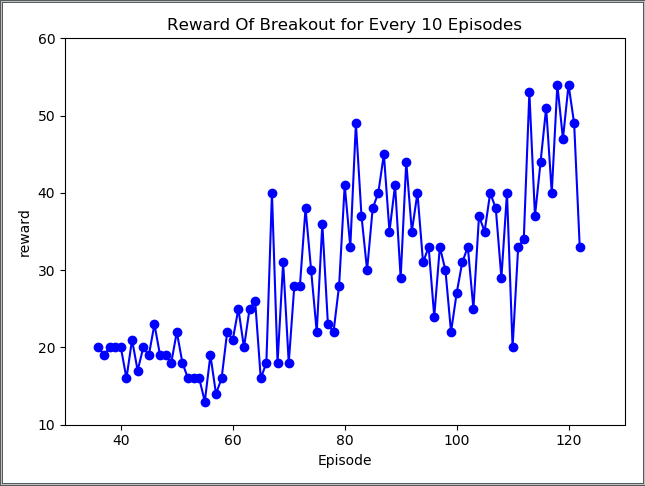
1. Experimental results 20%

We run code for about 24 hours and then get about 1200 episodes. There are some data is lost because they were not processed in time as the terminal cannot display too much data at same time.



**Figure 11. The reward for every single episode**

It can be seen clearly from the line chart, in general, there is almost no case where the score is 0 after about 800 episode and it is not until about 1100 episode that the score is 0. Besides, the highest score is getting higher and higher which shows that our training is effective.



**Figure11. Reward of every 10 episodes**

The line chart illustrates that there is a slightly growth before 600 episode which shows the agent learn quite slow. Subsequently, a significantly increase could be found between 600 and 1200 episode which indicated that the agent was getting to learn how to play this game.

Date 11.4.2019