Deep reinforcement learning with pixel features in Pong Game

Jiaqian Yu(jy2880)
Junkai Zhang(jz2929)
Wenshan Wang(ww2468)
Xiaoxiao Guo(xg2282)

1 Introduction

In this project, we implemented two methods of deep reinforcement learning to play game Pong: Policy Gradient and Deep Q-Network, which learn directly from screen images as in the network. We use convolutional neural network to extract features from the screen images and then apply Policy Gradient and Deep Q-Network to predict actions of game Pong.

1.1 Background

Pong is one of the earliest arcade video games. It is a table tennis sports game featuring simple two-dimensional graphics. The environment is from openai gym pong (https://gym.openai.com/envs/Pong-v0/).

We trained an AI to play as one of the paddles and fight against the other paddle (controlled by a decent AI).

In every step, we receive an image frame ($210 \times 160 \times 3$ byte array) and conduct convolutional neural network to extract features from it. Then, we train our AI based on these features using two different reinforcement learning methods: deep Q-network and policy network. After about three-days training with each method, we compare the performances of these two methods.

1.2 Goals

The goal of our AI is to win the game, in other words, to get a positive reward.

The goal of this project is to compare two different methods of reinforcement learning: Deep Q-Network and Policy Gradient and to see which method can result in a better reward after three-days training.

2 Model

Here, we briefly talk about our procedure to implement these two methods. We use Adam stochastic optimization method for all models with learning rate = 0.00025. We calculate the discounted moving average reward which is a weighted average of reward for current episode and reward achieved before. We use this discounted moving average reward to measure the performance of our AI learner.

2.1 Deep Q-Network

[2][3] For Deep Q-Network, in every step, after preprocessing images from ($210 \times 160 \times 3$) to ($80 \times 80 \times 1$), we receive a state as our input and use convolutional neural network to simulate the

Q-function, then we choose our action according to an ϵ-greedy policy, which is, with probability $1 - \epsilon$, we choose the action which has the largest $q - value$ and with probability ϵ, we choose a random action.

2.1.1 First Trial of Deep Q-Network

In our first trial, we try to implement Deep Q-Network based on some online resource and professors' lecture notes.

Our first trial of using Deep Q-Network failed but we will still describe our procedure and highlight the keys.

Memory Buffer First, we generate a memory buffer with a number of 80000 tuples (s, a, r, s') (we try to store more observations in the memory buffer but failed based on the memory limit in Colab and GCP, so we just choose the largest number within the memory limit). The memory buffer is important because it will reduce the dependence of our input and improve the stability of our model. Since the sequential observations $S_t, S_t + 1$ generated by Deep Q-Network are dependent and will cause higher variance, so we need to use mini-batch method to choose several states from the memory buffer in every stage in order to eliminate the dependence of data. So, within our storage ability, we want the size of memory buffer as large as possible.

Observation At first, we follow a tutorial[reference] online and generate one state as four consecutive observations after preprocessing (each with a dimension of 80×80) as the whole input for one step in the neural network. We choose four consecutive observations as one state because more information such as the action of the ball will be gathered with more observations.

CNN For the convolution neural network, we use four convolution layers followed by one dense layer. The first convolution layer takes an input of shape [None, 80, 80, 4], because each state contains 4 consecutive observations, filters=32, kernel size=[8, 8], strides=4; the second layer filters=64, kernel size=[4, 4], strides=2; the third layer filters=64, kernel size=[3, 3], strides=1; filters=hidden, kernel size=[3, 3], strides=1; and the dense layer has units equal to the number of actions 6.

Action We find out from environment that there are six possible actions in every steps. So in each step, we provide the $q - values$ for these six possible actions calculated from our convolutional neural network.

Exploration Rate Exploration rate is defined as the probability of taking a random action. Because of the property of greedy policy, we are always reaching to the local optimal point. However, the local optimal point may differ from global point so we should generate a probability of "exploring" other actions in order to see if this can actually provide a better optimum.

The function of ϵ is really important. We want our AI learner to explore more at first, and with the increasing of episodes, we decrease the exploration rate and then remain a constant value to force our AI learner to have a small probability to explore.

So, in every step, our AI learner will explore a random action with probability ϵ or choose the best action based on $q - values$ with probability $1 - \epsilon$.

We choose a linear function and choose 0.1 as the lowest exploration rate which means our AI learner will always explore a random action.

$$\epsilon = max(0.1, \epsilon - \tfrac{1}{10000})$$

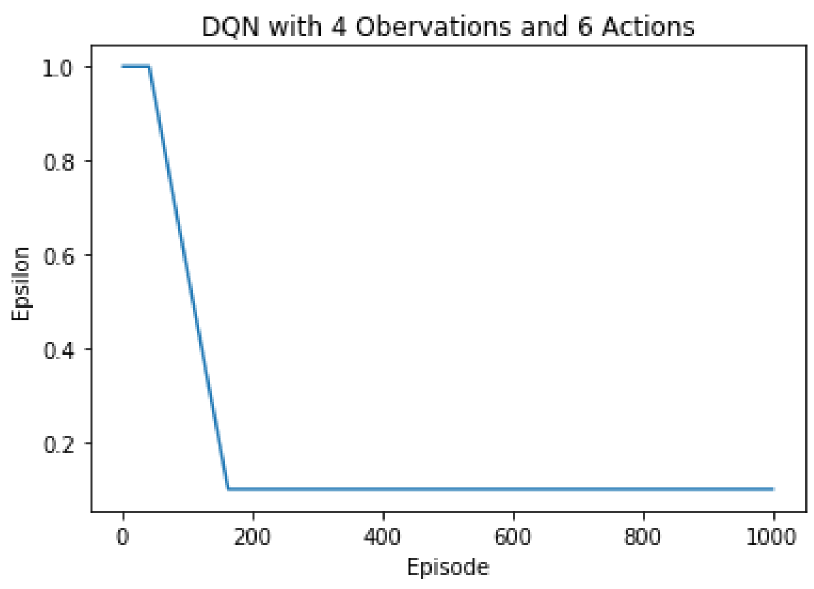

Figure 1: Exploration Rate Function of First DQN Trail

Result The result of our first trial is shown below:

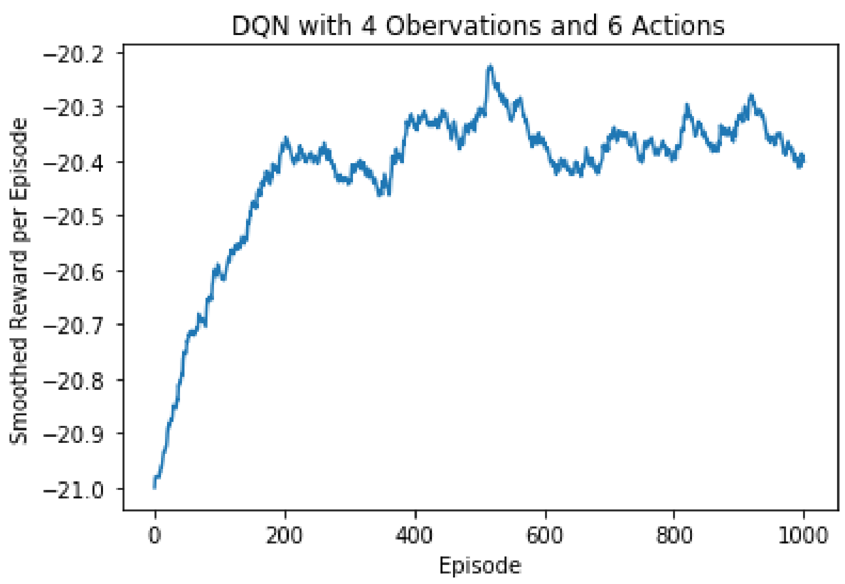

Figure 2: Performance of First DQN Trail

Discussion After training with about 24 hours, we find out that the discounted moving average reward are really lower at about -20 and it seems that our AI learner is not learning at all.

We think the reasons are: 1. With four observations as a whole state, the size of memory buffer is too small. 2. The exploration function still decreases too fast, so our AI learner didn't get explore enough random actions and reach out to some local optimal points. 3. We use too many actions, six, provided by the environment, some of them may be invalid, so it may cause low convergence rate of our algorithm.

2.1.2 Second Trial of Deep Q-Network

Based on the results of our first trial, we begin our second trial of Deep Q-Network.

Memory Buffer This time, We generate a memory buffer with a number of 300000 tuples (s, a, r, s').

Observation We decide to use the difference of two observations as one state in the convolutional neural network this time.

CNN For the convolution neural network, we use similar structure as our first attempt, except that the input layer takes in only one observation instead of four and that the output layer returns 2 values instead of 6. The first convolution layer takes an input of shape [None, 80, 80, 1], filters=32, kernel size=[8, 8], strides=4; the second layer filters=64, kernel size=[4, 4], strides=2; the third layer filters=64, kernel size=[3, 3], strides=1; filters=hidden, kernel size=[3, 3], strides=1; and the dense layer has units equal to the number of actions 2.

Action Here, we choose only two actions (UP and $DOWN$). So the output of our convolutional neural network is the $q - values$ for these two actions.

Exploration Rate [figure] Here, we change the linear function which decreases much more slower than our previous one. Also, we choose 0.01 as the lowest exploration rate as suggested by OpenAi Baselines for DQN.

Our exploration rate function is shown below:

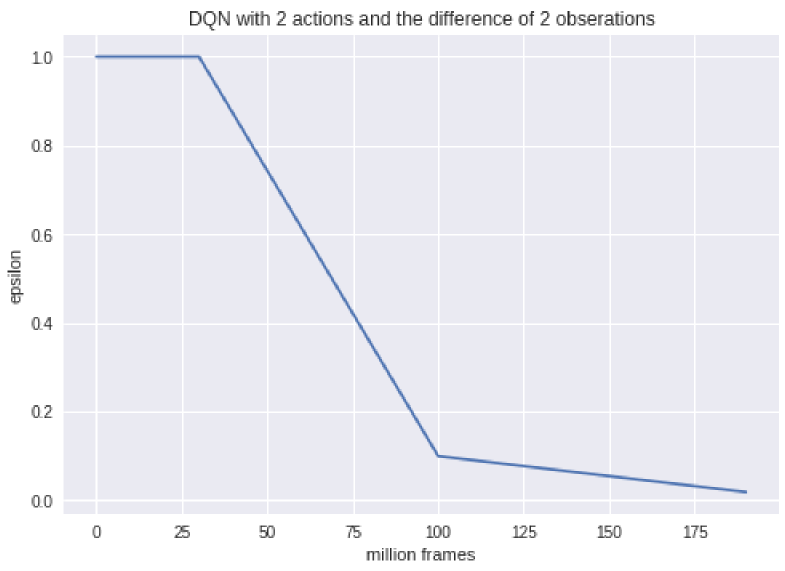

Figure 3: Exploration Rate Function of Second DQN Trail

Result The result of our first trial is shown below:

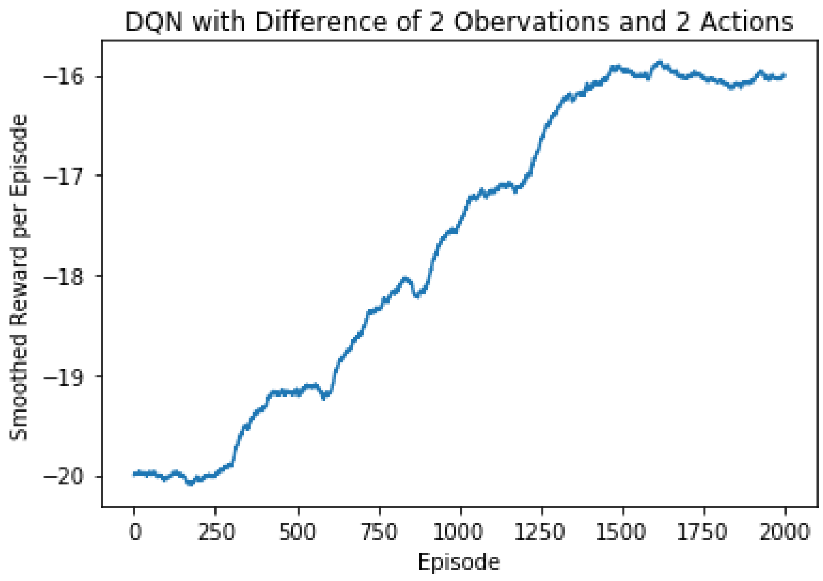

Figure 4: Performance of Second DQN Trail

Discussion After training with two days, we find out that the discounted moving average reward have some improvements and can reach to about -16. To some extent, it is learning but the result is still not good.

We think the reasons are: 1. The exploration rate function still decreases too fast and maybe linear deceasing function is not a good choice. If we have time, we should test more and more decreasing function and to find one with the best performance. 2. We use the structure of Deep Q-Network from lecture notes and it is a really simple one in DQN. It has a good performance in class because professor used a really easy game "cartpole" but our game is much more complicated so we may need more advanced Deep Q-Network methods such as Double-Dueling-DQN. We may explore this in the future. But in this project, we decide to replace Deep Q-Network with another method in reinforcement learning: Policy Gradient.

2.2 Policy Gradient

[4] For Policy Gradient, after preprocessing the image to $(80 \times 80 \times 1)$, we take the difference of two consecutive observations as input for our convolutional neural network and the output of our CNN is a probability of choosing going UP as the action in this step.

Then we sample an action based on this probability and move to the next step. After several steps, one episode finished which means we have won or lost the game.

Next, we will talk about the keys and results in the Policy Gradient model.

CNN The convolution neural network contains 1 input layer that takes in a tensor with shape [None, 80, 80, 1], a convolution layer with filters = 5, strides = [1, 1], padding = ’SAME’, kernel size = [3, 3], a max pooling layer with pool size=[2, 2], strides=2, a convolution layer with filters = 5, strides = [1, 1], padding = ’SAME’, kernel size = [3, 3], a max pooling layer with pool size=[2, 2], strides=2, a dense layer that has 20 hidden units, and finally a dense layer that outputs a single unit, which is the probability of going up.

Observation Here, we also use the difference of two consecutive observations as our input.

Result The results of our trial with Policy Network are:

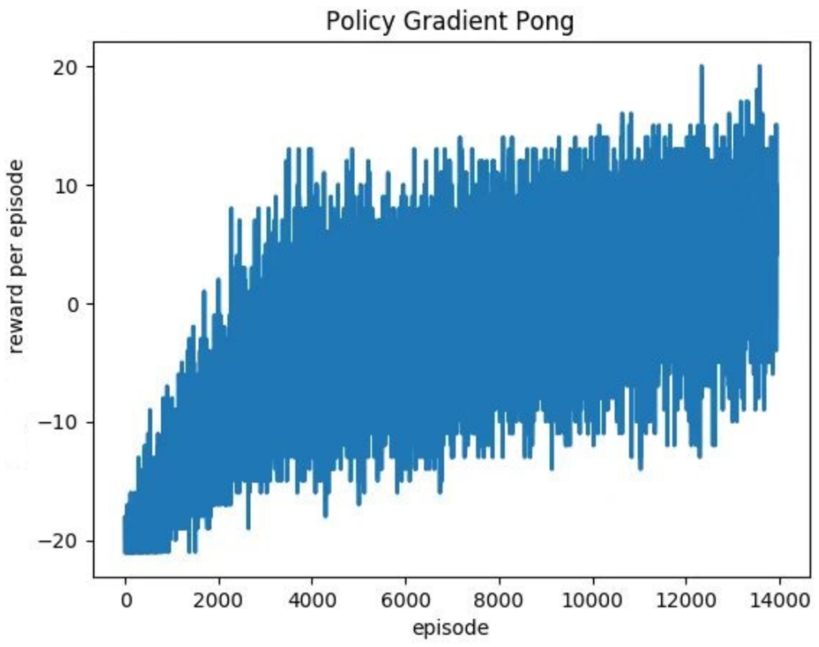

Figure 5: Performance of PG

3 Discussion and Conclusion

We were successfully able to train agents for pong using policy gradient and deep q network, among which the agent trained by policy gradient has the ability to achieve consistent wins.

The reasons why policy gradient outperforms deep q network are considered as follows: deep q network is a value-based algorithm that calculates the value of the action on every step and choose the action corresponding with the largest value, while Policy Network is a policy-based algorithm that takes state as input and then outputs action instead of q values.

$$a = \pi(a|s, \theta)$$

In other words, deep q network outputs assertive actions (although it uses $\epsilon - greedy$ to explore), while policy gradient outputs a probability. In many cases in the pong game, given a certain state, we do not have a single right action but a lot reasonable actions, so the algorithm that outputs a probability is more reasonable. As a result, the policy network performs better than deep q network when training agent to play Pong.

When comparing two attempts to build deep q network, we notice that changing the states from consecutive four observations (resulting state shape [80, 80, 4]) to the difference between two observations (resulting state shape [80, 80, 1]), changing the $\epsilon - greedy$ to make the ϵ to decrease slower, as well as changing number of actions from 6 to 2 (only allowing going up or down) can significantly increase the speed that the agent learns and improves. This is probably because, firstly, with four observations as a whole state, the size of memory buffer is too small. Secondly, the exploration function decreases too fast so that our AI learner cannot get explore enough random actions and reach out to some local optimal points. Finally, there are actually only two valid actions among the six Pong action space. A future work on improving the deep q network could be increasing the maximum memory replay buffer size and decreasing the rate that ϵ decreases.

References

[1] P.A.Somnuk, Learning to Play Pong using Policy Gradient Learning https://arxiv.org/pdf/1807.08452.pdf

[2] M.Volodymyr, K.Koray, S.David, Human-level control through deep reinforcement learning Nature 518, 52-533,(26 February 2015)

[3] https:https://github.com/fg91/Deep-Q-Learning/blob/master/DQN.ipynb

[4] http://karpathy.github.io/2016/05/31/rl