CS 440 Assignment 4

R section 3 credits

Group members:
Chendi Lin, Zhuoyue Wang, Zhaoyu Wu

1. Overview

In this assignment, we first continue to utilize the digit classification by discriminative machine learning methods--perceptrons and nearest neighbor, implementing different algorithms from those in Assignment 3, and compare the results given by same inputs from all methods we have implemented.

In the second part, we need to implement various Pong games using Q-Learning. For 3-credit part, the agent is trained to play alone against the wall. For 4-credit part, two agents compete with each other, with one trained by Q-learning and the other one hard-coded. Graphical representations and animations of agents playing the game are also created.

2. Work distribution

Chendi Lin: Q-learning, report

Zhuoyue Wang: Digit Classification, Q-learning, report

Zhaoyu Wu: Digit Classification, report

3. Section 1: Digit Classification using Discriminative Machine Learning Methods

3.1. Digit Classification with Perceptrons

(1) Implementation:

We follow the steps of multi-class non-differentiable perceptron learning rule to build our machine learning model. First, we trained all of the digit images by finding the best prediction and updating the weights, and report training accuracy per epoch out of 100 epoch. And then, we use the result to test our test images, to get the overall accuracy and confusion matrix.

(2) Parameters:

1. Learning rate decay function:

$$\alpha = 5/(5 + current\ epoch)$$

We get the best constant 5 for this function in trials, considering both converging speed and classification accuracy. We firstly set the constant to 100, discovering that from 1-100 epoch, the training accuracy approaches to 100% during 70-80 epoch. After that, we gradually decrease the constant, by 5 or 10, finally getting that 5 may be the best constant for this formula, given the result that during epoch 30-40, the training accuracy may approach to 100% and overall accuracy of testing is around 82% other than around 80% for other constants.

2. **Bias Factor: On**

From our perspective, bias factor may not influence the result a lot, because when we calculate the inner products of w vectors and F vectors, it is impossible to have a all-zero vector due to that every digit must have some foreground values. However, by trials, when we turn on the bias factor and set the bias value, which is added by previous inner product of w and F, to 1, the result is a little better than turning it off.

3. **Initialization of weights: zeros**

Same as the previous, we still mainly rely on continuous trials to make our decisions. However, we have some theoretical ideas of setting weights to zeros: during the training, we should traverse every pixel to get the inner product of weight vector and pixel value vector, while pixel value vector has either 1 or 0, corresponding to foreground and background. When value 0 times the weight, whatever zero weight or random weight, the result will be 0, while value 1 times the weight will not be 0 after the weight is updated. So setting the weights initially to all zeros is efficient enough to help us make clear classification.

4. **Ordering of examples: random**

As we stated above, we rely much on trials with the aim of getting better implementations and outcomes. If we set the ordering to be fixed and keep some other factors unchanged, the calculation of training may be similar which causes that we may get the similar results. It may not help us get a comprehensive idea of our implementation. Furthermore, there are some edge circumstances that hard to be discovered, which may decrease the overall accuracy or have a bad running time. More randomness may give us more chances to optimize our implementation and prevent from those edge circumstances.

5. **Number of epochs: 100**

During our trials, the training accuracy approaches to 100% during 30-40 epochs to 70-80 epochs, depending on different parameter settings. So it is safe to set the epochs to 100, both for helping us getting better opinions of parameter setting and for letting the algorithm implement better classification.

(3) Training curve:

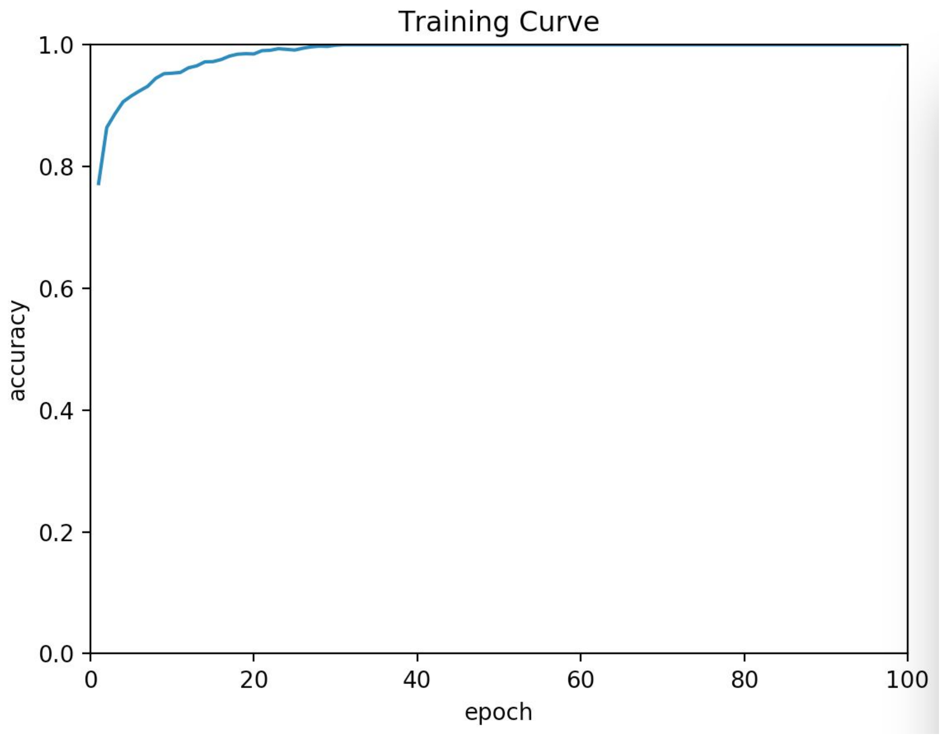


Fig 1. Training Curve

Learning rate constant = 5, bias = on, weights = zeros, ordering = random. We can discover that during about epoch 30, the training accuracy approaches to 100%.

(4) Overall Accuracy: 82.1%

Classification rates for each digit:

0: 91.11%

1: 98.15%

2: 81.55%

3: 79%

4: 84.11%

5: 69.57%

6: 86.81%

7: 78.3%

8: 67.96%

9: 84%

(5) Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	**91.11%**	0%	4.44%	0%	0%	0%	2.22%	0%	1.11%	1.11%
1	0%	**98.15%**	0%	0%	0.93%	0%	0.93%	0%	0%	0%
2	0%	1.94%	**81.55%**	3.88%	0.97%	0%	2.91%	2.91%	4.85%	0.97%
3	0%	0%	4%	**79%**	0%	6%	1%	5%	4%	1%
4	0%	0%	2.8%	1.87%	**84.11%**	0%	3.74%	0.94%	0.94%	5.61%
5	2.17%	0%	2.17%	5.44%	1.09%	**69.56%**	3.26%	3.26%	10.87%	2.17%
6	1.1%	1.1%	3.3%	0%	2.2%	1.1%	**86.81%**	2.2%	2.2%	0%
7	0.94%	2.83%	3.77%	1.89%	2.83%	0%	0%	**78.3%**	0.94%	8.49%
8	0%	2.91%	5.83%	5.83%	0.97%	6.8%	3.88%	1.94%	**67.96%**	3.88%
9	0%	0%	0%	5%	4%	1%	0%	4%	2%	**84%**

(6) Compare to Results from Naive Bayes

We can easily see perceptrons can have a better overall accuracy (82.1%) than Naive Bayes (77.24%) by about 5%. This advantage is from almost every digit's classification rate in perceptrons is more than that in Naive Bayes. Some digits, such as 5 and 8, which have relatively lower classification rates in Naive Bayes, still have lower classification rates in perceptrons. Consequently, the general trends (i.e. which digits get larger classification rate, while which get lower) in both algorithm are the same, but perceptrons have better accuracy for each digit, thus leading to a better overall accuracy.

What's more, by looking at confusion matrix, we can easily get the phenomenon that the high faulty classification rates in Assignment 3, such as classifying 4 to 9 with rate 16.82%, are much more lower in perceptrons. The highest faulty classification rate in this part is about 10%, compared to 4 pairs in Naive Bayes over 10%. Hence, this better overall accuracy may come from its ability to better distinguish pairs "similar-looking" digits.

3.2. Digit Classification with Nearest Neighbor

(1) Overall Accuracy: 91.1%

(2) Confusion Matrix with best k = 6:

	0	1	2	3	4	5	6	7	8	9
0	**100%**	0%	0%	0%	0%	0%	0%	0%	0%	0%
1	0%	**100%**	0%	0%	0%	0%	0%	0%	0%	0%
2	0.97%	3.88%	**89.32%**	1.94%	0.97%	0%	0.97%	0.97%	0.97%	0%
3	0%	0%	0%	**87%**	0%	5%	0%	3%	3%	2%
4	0%	0.93%	0%	0%	**93.49%**	0%	2.8%	0%	0%	1.87%
5	2.17%	1.09%	0%	4.35%	0%	**85.87%**	3.26%	1.09%	1.09%	1.09%
6	0%	1.1%	0%	0%	0%	1.1%	**97.8%**	0%	0%	0%
7	0%	8.49%	0.94%	0%	1.89%	0%	0%	**83.02%**	0%	5.56%
8	1.94%	1.94%	0%	2.91%	2.91%	3.88%	0%	1.94%	**82.52%**	1.94%
9	1%	0%	0%	3%	3%	2%	0%	0%	0%	**91%**

(3) Distance Function:

$\sqrt[n]{Manhattan\ Distance}$ for n-norm distance

(4) Running time:

The general trend of running time is that as K increases, the running time increases.

3.3 Extra credit: Advanced features

We adopted the ternary features from Assignment 3 into perceptrons without changing the parameters in part 1.1. We get the results:

(1) Training Curve:

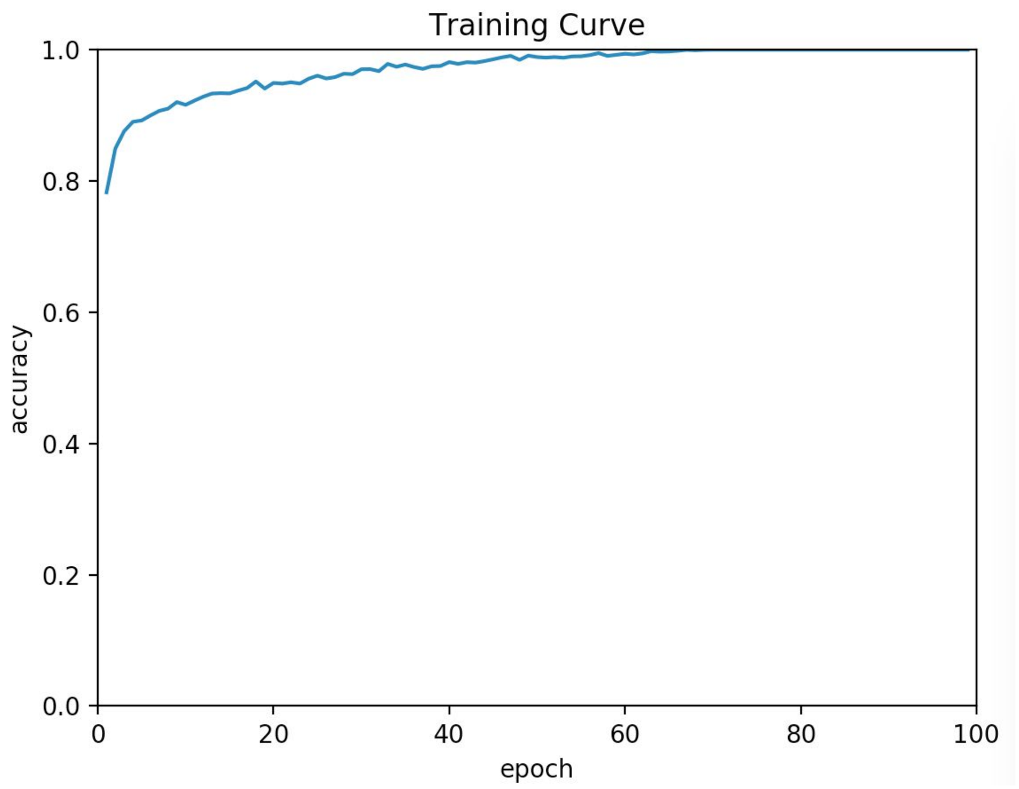


Fig 2. Training Curve

(2) Overall accuracy: 82.7%

(3) Classification Rates for Every Digit:

0: 93.33%

1: 95.37%

2: 83.5%

3: 76%

4: 85.05%

5: 75%

6: 92.31%

7: 76.42%

8: 67.96%

9: 83%

(4) Confusion Matrix:

	0	1	2	3	4	5	6	7	8	9
0	**93.33%**	0%	2.22%	0%	0%	0%	2.22%	0%	1.11%	1.11%
1	0.93%	**95.37%**	0%	0%	0.93%	0%	0.93%	0%	1.85%	0%
2	0%	1.94%	**83.49%**	1.94%	0.97%	0.97%	3.88%	3.88%	2.91%	0%
3	1%	1%	3%	**76%**	0%	6%	0%	7%	6%	0%
4	0%	0%	0%	0.94%	**85.05%**	0%	4.67%	1.87%	0%	7.48%
5	1.09%	0%	2.17%	4.35%	1.09%	**75%**	5.44%	1.09%	8.7%	1.09%
6	0%	1.1%	0%	0%	2.2%	2.2%	**92.31%**	1.1%	1.1%	0%
7	0.94%	2.83%	3.77%	1.89%	0.94%	0%	0%	**76.41%**	0.94%	12.26%
8	1.84%	2.91%	2.91%	8.74%	3.88%	5.83%	1.94%	0.97%	**67.96%**	2.91%
9	1%	0%	0%	4%	4%	3%	0%	5%	0%	**83%**

(5) Compare to Other Algorithms:

Comparing it to perceptrons with binary features, it gets a slightly better overall accuracy. However, during 70-80 epochs in ternary, compared to 30-40 epochs in binary, the training accuracy approaches to 100%. It may be because the complexity of value increases, which causes that corresponding weights are a little bit harder for machine learning to classify digits into different classes.

What’s more, comparing it to Naive Bayes with ternary, it still has a better overall accuracy, which may result from what we have discussed about the advantages of perceptrons over Naive Bayes algorithm.

3.4 Extra credit: Visualizations

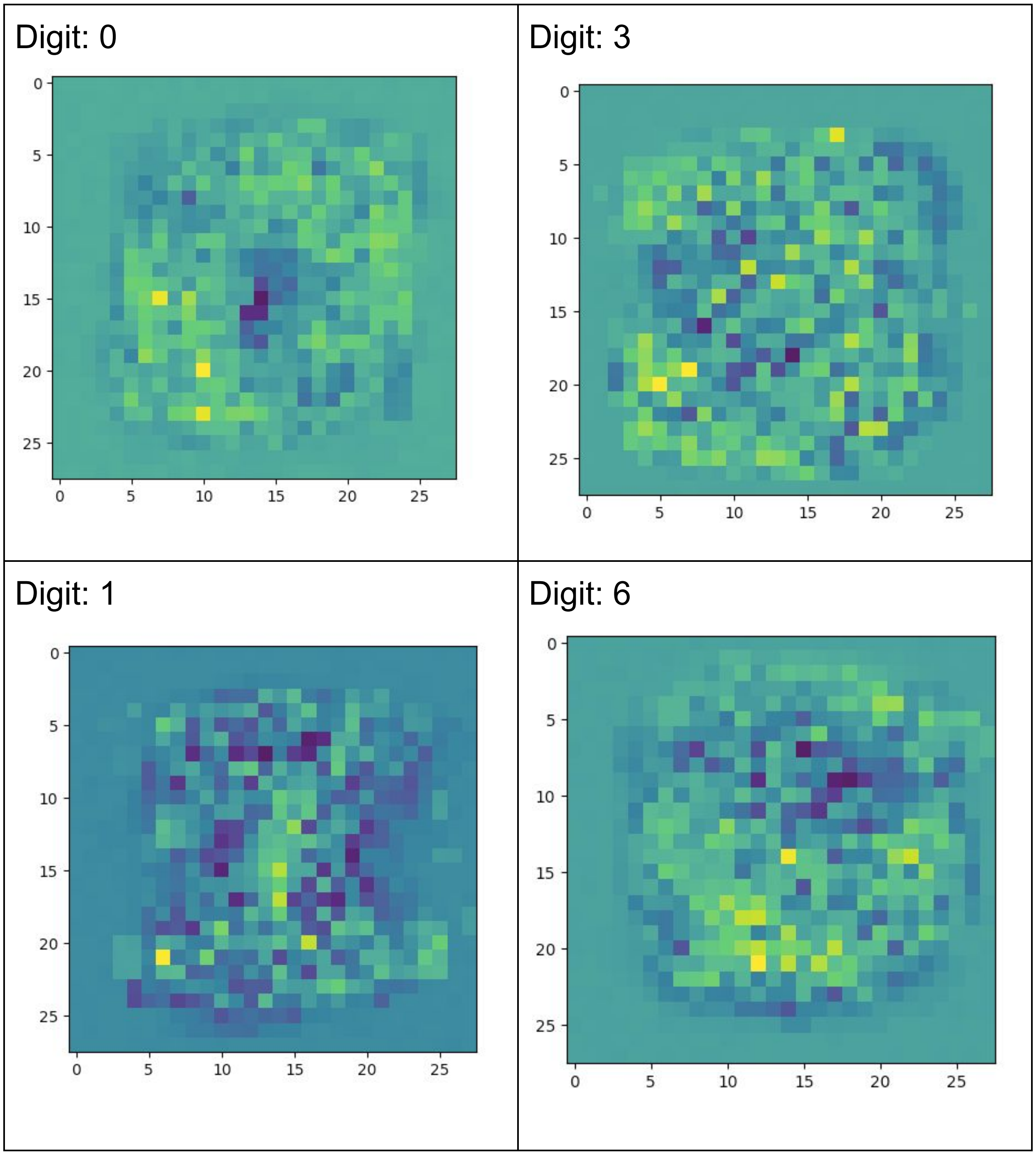


Fig 3. Visualization

In our weight visualization about the digit image, yellow color means the positive value, blue color means the negative value and green color means zero. It implies that the light green is slightly above zero and the dark green is slightly below zero vice versa.

We picked the four digits with highest classification rate to show the the perceptron weight in each digit. From the images above, we can clearly see that the overall shape about each specific digit. For example, the digit 0 has a circle formed by positive weight values. The visualization is more obvious by looking at these picture at a distance, that we can recognize several light green digits.

3.5 Extra credit: Differentiable perceptron

(1) Training Curve:

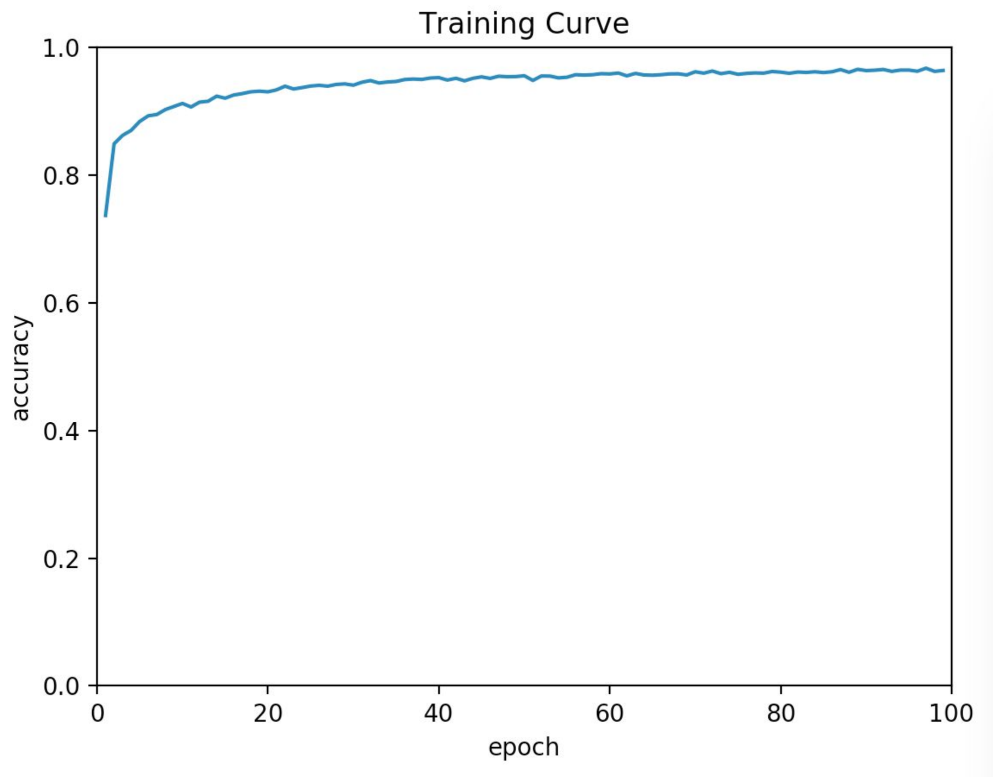


Fig 3. Training Curve

(2) Overall accuracy: 80.26%

(3) Confusion Matrix:

	0	1	2	3	4	5	6	7	8	9
0	**91.02%**	0%	1.11%	0%	0%	2.25%	2.25%	1.11%	1.12%	2.25%
1	0.00%	**99.05%**	0.92%	0%	0%	0%	0.93%	0%	0.03%	0%
2	0.98%	2.94%	**79.41%**	5.68%	2.94%	1.08%	1.96%	0.98%	1.96%	2.94%
3	0%	0%	2%	**79%**	0%	8%	2%	6%	3%	0%
4	0%	0.94%	3.88%	1.99%	**81.04%**	0.93%	2.83%	2.83%	0%	6.49%
5	4.38%	0%	0%	5.19%	1.39%	**80.25%**	2.22%	1.09%	3.31%	3.31%
6	0%	1.1%	4.4%	0%	5.5%	7.8%	**80.1%**	0%	1.1%	0%
7	0%	0.93%	4.78%	2.86%	0.95%	0.95%	0.95%	**79.05%**	0.94%	8.59%
8	0.93%	1.91%	7.76%	8.74%	0%	7.84%	1.94%	0.97%	**67.01%**	2.91%
9	0%	0%	1%	1%	11%	4%	1%	6%	0%	**76%**

(4) Compare to Other Algorithms:

Non-differentiable perceptrons gets a little bit lower overall accuracy as differentiable one. With same parameters, till epoch 100, the training accuracy will not be equal to 100%, which means that its training difficulty is greater than differentiable one. However, despite its slight disadvantage over differentiable perceptrons, it is still better than Naive Bayes.

3.6 Extra credit: Support Vector Machine

We also applied support vector machine for classification. The result is:

(1) Overall accuracy: 87.8%

(2) Confusion Matrix:

	0	1	2	3	4	5	6	7	8	9
0	**95.56%**	0%	0%	0%	2.22%	0%	1.11%	0%	1.11%	0%
1	0%	**99.07%**	0%	0%	0%	0%	0.93%	0%	0%	0%
2	0%	0.97%	**87.38%**	0.97%	1.94%	0%	4.85%	0%	1.94%	1.94%
3	0%	0%	1%	**86%**	0%	5%	1%	4%	1%	2%

4	0%	0.93%	0%	0%	**88.79%**	0%	1.87%	0.93%	0%	7.48%
5	2.17%	0%	0%	3.26%	0%	**89.13%**	2.17%	1.09%	2.17%	0%
6	2.2%	1.1%	0%	0%	3.3%	3.3%	**89.01%**	0%	1.1%	0%
7	0.94%	4.72%	4.72%	0%	1.89%	0%	0%	**78.3%**	0%	9.43%
8	0.97%	0.97%	2.91%	2.91%	2.91%	0.68%	0%	1.94%	**77.67%**	2.91%
9	1%	0%	0%	4%	3%	2%	0%	2%	0%	**88%**

4. Section 2: Q-Learning (Pong)

4.1 Single-Player Pong

In this section, we were required to implement a single-player Pong game. The agent is trained using Q-learning algorithm. The state in this problem is described as a tuple consisted of (ball_x, ball_y, velocity_x, velocity_y, paddle_y). The movements of the ball and the paddle are continuous. However, to use Q-learning, we need to discretize the problem. The instructed, the board is discretized as 12*12 grid, and we only consider the ball and the top of the paddle to be located in some cell. More details of the choice of MDP will be discussed in the next few sections.

(1) Choices for α, γ, and the exploration function

Following the instruction, the learning rate α is set to be $\alpha = C/(C+N(s,a))$, where $N(s,a)$ is the number of times the state-action pair visited. The C value is chosen to be 500, with sensitivity test and several manually trials. γ, the discount factor, is set to be 0.7, also with sensitivity test and several manually trials.

Both exploration and exploitation settings have been experimented. With exploitation formulation, the action is simply chosen as

$$a = argmax_{a'}(Q(s,a'))$$

While in exploration formulation, the action is chosen by

$$a = argmax_{a'} f(Q(s,a'), N(s,a'))$$

where

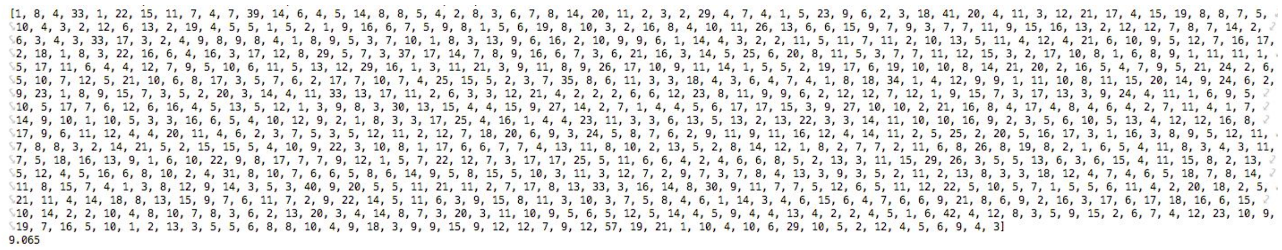
$$\begin{aligned} f(u,n) &= R^{+} \ (very\ large\ value)\ if\ n < N_e \\ &= u\ otherwise \end{aligned}$$

N_e is the optimal reward estimate. We chose it to be 5.

Using exploitation formulation, even with 100000 training games, the average bounces in the 1000 test games is barely 7, which is way lower than required. However, with exploration formulation, by tuning C and γ, the average varies from 9 to around 12. The results will be shown in the next section.

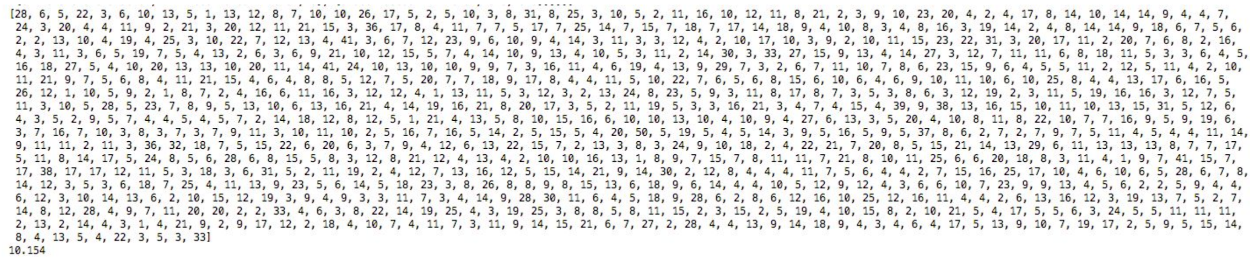
(2) Results with tuning α, γ

Several results with different parameters are shown below. The results are shown in screenshot, with the upper part as the bounce number of the 1000 test games and the last row as the average number.



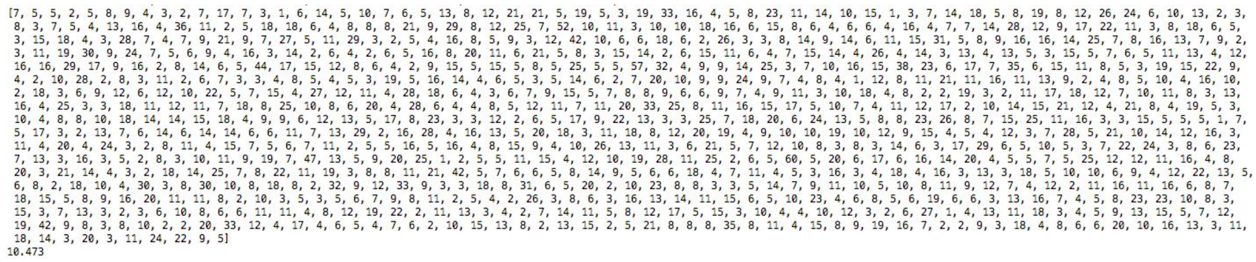
```
[1, 8, 4, 33, 1, 22, 15, 11, 7, 4, 7, 39, 14, 6, 4, 5, 14, 8, 8, 5, 4, 2, 8, 3, 6, 7, 8, 14, 20, 11, 2, 3, 2, 29, 4, 7, 4, 1, 5, 23, 9, 6, 2, 3, 18, 41, 20, 4, 11, 3, 12, 21, 17, 4, 15, 19, 8, 8, 7, 5,
10, 4, 3, 2, 12, 6, 13, 2, 19, 4, 5, 5, 1, 5, 2, 1, 9, 16, 6, 7, 5, 9, 8, 1, 5, 6, 19, 8, 10, 3, 2, 16, 8, 4, 10, 11, 26, 13, 6, 6, 15, 9, 7, 9, 3, 7, 7, 11, 9, 15, 16, 13, 2, 12, 12, 7, 8, 7, 14, 2,
6, 3, 4, 3, 33, 17, 3, 2, 4, 9, 8, 9, 8, 4, 1, 8, 9, 5, 3, 7, 10, 1, 8, 3, 13, 9, 6, 16, 2, 10, 9, 9, 6, 1, 14, 4, 3, 2, 2, 11, 5, 11, 7, 11, 2, 10, 13, 5, 11, 4, 12, 4, 21, 6, 10, 9, 5, 12, 7, 16, 17,
2, 18, 1, 8, 3, 22, 16, 6, 4, 16, 3, 17, 12, 8, 29, 5, 7, 3, 37, 17, 14, 7, 8, 9, 16, 6, 7, 3, 6, 21, 16, 3, 14, 5, 25, 6, 20, 8, 11, 5, 3, 7, 7, 11, 12, 15, 3, 2, 17, 10, 8, 1, 6, 8, 9, 1, 11, 11, 1,
5, 17, 11, 6, 4, 4, 12, 7, 9, 5, 10, 6, 11, 5, 13, 12, 29, 16, 1, 3, 11, 21, 3, 9, 11, 8, 9, 26, 17, 10, 9, 11, 14, 1, 5, 5, 2, 19, 17, 6, 19, 10, 10, 8, 14, 21, 20, 2, 16, 5, 4, 7, 9, 5, 21, 24, 2, 6,
5, 10, 7, 12, 5, 21, 10, 6, 8, 17, 3, 5, 7, 6, 2, 17, 7, 10, 7, 4, 25, 15, 5, 2, 3, 7, 35, 8, 6, 11, 3, 3, 18, 4, 3, 6, 4, 7, 4, 1, 8, 18, 34, 1, 4, 12, 9, 9, 1, 11, 10, 8, 11, 15, 20, 14, 9, 24, 6, 2,
9, 23, 1, 8, 9, 15, 7, 3, 5, 2, 20, 3, 14, 4, 11, 33, 13, 17, 11, 2, 6, 3, 3, 12, 21, 4, 2, 2, 2, 6, 6, 12, 23, 8, 11, 9, 9, 6, 2, 12, 12, 7, 12, 1, 9, 15, 7, 3, 17, 13, 3, 9, 24, 4, 11, 1, 6, 9, 5,
10, 5, 17, 7, 6, 12, 6, 16, 4, 5, 13, 5, 12, 1, 3, 9, 8, 3, 30, 13, 15, 4, 4, 15, 9, 27, 14, 2, 7, 1, 4, 4, 5, 6, 17, 17, 15, 3, 9, 27, 10, 10, 2, 21, 16, 8, 4, 17, 4, 8, 4, 6, 4, 2, 7, 11, 4, 1, 7,
14, 9, 10, 1, 10, 5, 3, 3, 16, 6, 5, 4, 10, 12, 9, 2, 1, 8, 3, 3, 17, 25, 4, 16, 1, 4, 4, 23, 11, 3, 3, 6, 13, 5, 13, 2, 13, 22, 3, 3, 14, 11, 10, 10, 16, 9, 2, 3, 5, 6, 10, 5, 13, 4, 12, 12, 16, 8,
17, 9, 6, 11, 12, 4, 4, 20, 11, 4, 6, 2, 3, 7, 5, 3, 5, 12, 11, 2, 12, 7, 18, 20, 6, 9, 3, 24, 5, 8, 7, 6, 2, 9, 11, 9, 11, 16, 12, 4, 14, 11, 2, 5, 25, 2, 20, 5, 16, 17, 3, 1, 16, 3, 8, 9, 5, 12, 11,
7, 8, 8, 3, 2, 14, 21, 5, 2, 15, 15, 5, 4, 10, 9, 22, 3, 10, 8, 1, 17, 6, 6, 7, 7, 4, 13, 11, 8, 10, 2, 13, 5, 2, 8, 14, 12, 1, 8, 2, 7, 7, 2, 11, 6, 8, 26, 8, 19, 8, 2, 1, 6, 5, 4, 11, 8, 3, 4, 3, 11,
7, 5, 18, 16, 13, 9, 1, 6, 10, 22, 9, 8, 17, 7, 7, 9, 12, 1, 5, 7, 22, 12, 7, 3, 17, 17, 25, 5, 11, 6, 6, 4, 2, 4, 6, 6, 8, 5, 2, 13, 3, 11, 15, 29, 26, 3, 5, 5, 13, 6, 3, 6, 15, 4, 11, 15, 8, 2, 13,
5, 12, 4, 5, 16, 6, 8, 10, 2, 4, 31, 8, 10, 7, 6, 6, 5, 8, 6, 14, 9, 5, 8, 15, 5, 10, 3, 11, 3, 12, 7, 2, 9, 7, 3, 7, 8, 4, 13, 3, 9, 3, 5, 2, 11, 2, 13, 8, 3, 3, 18, 12, 4, 7, 4, 6, 5, 18, 7, 8, 14,
11, 8, 15, 7, 4, 1, 3, 8, 12, 9, 14, 3, 5, 3, 40, 9, 20, 5, 5, 11, 21, 11, 2, 7, 17, 8, 13, 33, 3, 16, 14, 8, 30, 9, 11, 7, 7, 5, 12, 6, 5, 11, 12, 22, 5, 10, 5, 7, 1, 5, 5, 6, 11, 4, 2, 20, 18, 2, 5,
21, 11, 4, 14, 18, 8, 13, 15, 9, 7, 6, 11, 7, 2, 9, 22, 14, 5, 11, 6, 3, 9, 15, 8, 11, 3, 10, 3, 7, 5, 8, 4, 6, 1, 14, 3, 4, 6, 15, 6, 4, 7, 6, 6, 9, 21, 8, 6, 9, 2, 16, 3, 17, 6, 17, 18, 16, 6, 15,
10, 14, 2, 2, 10, 4, 8, 10, 7, 8, 3, 6, 2, 13, 20, 3, 4, 14, 8, 7, 3, 20, 3, 11, 10, 9, 5, 6, 5, 12, 5, 14, 4, 5, 9, 4, 4, 13, 4, 2, 2, 4, 5, 1, 6, 42, 4, 12, 8, 3, 5, 9, 15, 2, 6, 7, 4, 12, 23, 10, 9,
19, 7, 16, 5, 10, 1, 2, 13, 3, 5, 5, 6, 8, 8, 10, 4, 9, 18, 3, 9, 9, 15, 9, 12, 12, 7, 9, 12, 57, 19, 21, 1, 10, 4, 10, 6, 29, 10, 5, 2, 12, 4, 5, 6, 9, 4, 3]
9.065
```

$C = 200, \gamma = 0.7, ave = 9.063$



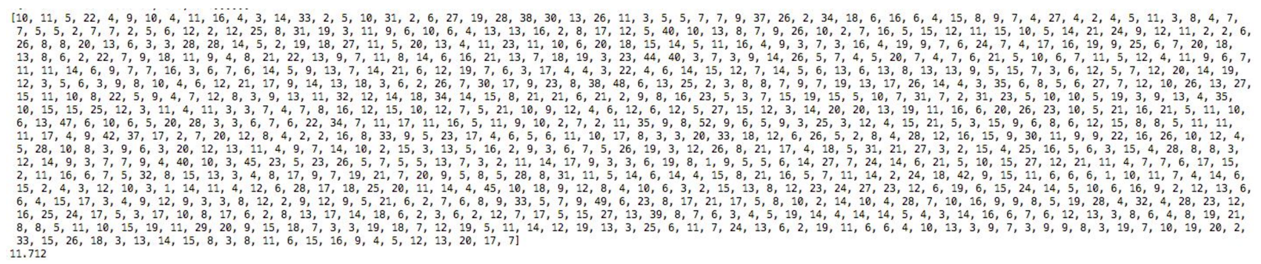
```
[28, 6, 5, 22, 3, 6, 10, 13, 5, 1, 13, 12, 8, 7, 10, 10, 26, 17, 5, 2, 5, 10, 3, 8, 31, 8, 25, 3, 10, 5, 2, 11, 16, 10, 12, 11, 8, 21, 2, 3, 9, 10, 23, 20, 4, 2, 4, 17, 8, 14, 10, 14, 14, 9, 4, 4, 7,
 24, 3, 20, 4, 4, 11, 9, 2, 21, 3, 20, 12, 11, 21, 15, 3, 36, 17, 8, 4, 11, 7, 7, 5, 17, 7, 25, 14, 7, 15, 7, 18, 7, 17, 14, 18, 9, 4, 10, 8, 3, 4, 8, 16, 3, 19, 14, 2, 4, 8, 14, 14, 9, 18, 6, 7, 5, 6,
 2, 2, 13, 10, 4, 19, 4, 25, 3, 10, 22, 7, 12, 13, 4, 41, 3, 6, 7, 12, 23, 9, 6, 10, 9, 4, 14, 3, 11, 3, 3, 12, 4, 2, 10, 17, 10, 3, 9, 2, 10, 11, 15, 23, 22, 31, 3, 20, 17, 11, 2, 20, 7, 6, 8, 2, 16,
 4, 3, 11, 3, 6, 5, 19, 7, 5, 4, 13, 2, 6, 3, 6, 9, 21, 10, 12, 15, 5, 7, 4, 14, 10, 9, 13, 4, 10, 5, 3, 11, 2, 14, 30, 3, 33, 27, 15, 9, 13, 4, 14, 27, 3, 12, 7, 11, 11, 6, 8, 18, 11, 5, 3, 3, 6, 4, 5,
 16, 18, 27, 5, 4, 10, 20, 13, 13, 10, 20, 11, 14, 41, 24, 10, 13, 10, 10, 9, 9, 7, 3, 16, 11, 4, 6, 19, 4, 13, 9, 29, 7, 3, 2, 6, 7, 11, 10, 7, 8, 6, 23, 15, 9, 6, 4, 5, 5, 11, 2, 12, 5, 11, 4, 2, 10,
 11, 21, 9, 7, 5, 6, 8, 4, 11, 21, 15, 4, 6, 4, 8, 8, 5, 12, 7, 5, 20, 7, 7, 18, 9, 17, 8, 4, 4, 11, 5, 10, 22, 7, 6, 5, 6, 8, 15, 6, 10, 6, 4, 6, 9, 10, 11, 10, 6, 10, 25, 8, 4, 4, 13, 17, 6, 16, 5,
 26, 12, 1, 10, 5, 9, 2, 1, 8, 7, 2, 4, 16, 6, 11, 16, 3, 12, 12, 4, 1, 13, 11, 5, 3, 12, 3, 2, 13, 24, 8, 23, 5, 9, 3, 11, 8, 17, 8, 7, 3, 5, 3, 8, 6, 3, 12, 19, 2, 3, 11, 5, 19, 16, 16, 3, 12, 7, 5,
 11, 3, 10, 5, 28, 5, 23, 7, 8, 9, 5, 13, 10, 6, 13, 16, 21, 4, 14, 19, 16, 21, 8, 20, 17, 3, 5, 2, 11, 19, 5, 3, 3, 16, 21, 3, 4, 7, 4, 15, 4, 39, 9, 38, 13, 16, 15, 10, 11, 10, 13, 15, 31, 5, 12, 6,
 4, 3, 5, 2, 9, 5, 7, 4, 4, 5, 4, 5, 7, 2, 14, 18, 12, 8, 12, 5, 1, 21, 4, 13, 5, 8, 10, 15, 16, 6, 10, 10, 13, 10, 4, 10, 9, 4, 27, 6, 13, 3, 5, 20, 4, 10, 8, 11, 8, 22, 10, 7, 7, 16, 9, 5, 9, 19, 6,
 3, 7, 16, 7, 10, 3, 8, 3, 7, 3, 7, 9, 11, 3, 10, 11, 10, 2, 5, 16, 7, 16, 5, 14, 2, 5, 15, 5, 4, 20, 50, 5, 19, 5, 4, 5, 14, 3, 9, 5, 16, 5, 9, 5, 37, 8, 6, 2, 7, 2, 7, 9, 7, 5, 11, 4, 5, 4, 4, 11, 14,
 9, 11, 11, 2, 11, 3, 36, 32, 18, 7, 5, 15, 22, 6, 20, 6, 3, 7, 9, 4, 12, 6, 13, 22, 15, 7, 2, 13, 3, 8, 3, 24, 9, 10, 18, 2, 4, 22, 21, 7, 20, 8, 5, 15, 21, 14, 13, 29, 6, 11, 13, 13, 13, 8, 7, 7, 17,
 5, 11, 8, 14, 17, 5, 24, 8, 5, 6, 28, 6, 8, 15, 5, 8, 3, 12, 8, 21, 12, 4, 13, 4, 2, 10, 10, 16, 13, 1, 8, 9, 7, 15, 7, 8, 11, 11, 7, 21, 8, 10, 11, 25, 6, 6, 20, 18, 8, 3, 11, 4, 1, 9, 7, 41, 15, 7,
 17, 38, 17, 17, 12, 11, 5, 3, 18, 3, 6, 31, 5, 2, 11, 19, 2, 4, 12, 7, 13, 16, 12, 5, 15, 14, 21, 9, 14, 30, 2, 12, 8, 4, 4, 4, 11, 7, 5, 6, 4, 4, 2, 7, 15, 16, 25, 17, 10, 4, 6, 10, 6, 5, 28, 6, 7, 8,
 14, 12, 3, 5, 3, 6, 18, 7, 25, 4, 11, 13, 9, 23, 5, 6, 14, 5, 18, 23, 3, 8, 26, 8, 8, 9, 8, 15, 13, 6, 18, 9, 6, 14, 4, 4, 10, 5, 12, 9, 12, 4, 3, 6, 6, 10, 7, 23, 9, 9, 13, 4, 5, 6, 2, 2, 5, 9, 4, 4,
 6, 12, 3, 10, 14, 13, 6, 2, 10, 15, 12, 19, 3, 9, 4, 9, 3, 3, 11, 7, 3, 4, 14, 9, 28, 30, 11, 6, 4, 5, 18, 9, 28, 6, 2, 8, 6, 12, 16, 10, 25, 12, 16, 11, 4, 4, 2, 6, 13, 16, 12, 3, 19, 13, 7, 5, 2, 7,
 14, 8, 12, 28, 4, 9, 7, 11, 20, 20, 2, 2, 33, 4, 6, 3, 8, 22, 14, 19, 25, 4, 3, 19, 25, 3, 8, 8, 5, 8, 11, 15, 2, 3, 15, 2, 5, 19, 4, 10, 15, 8, 2, 10, 21, 5, 4, 17, 5, 5, 6, 3, 24, 5, 5, 11, 11, 11,
 2, 13, 2, 14, 4, 3, 1, 4, 21, 9, 2, 9, 17, 12, 2, 18, 4, 10, 7, 4, 11, 7, 3, 11, 9, 14, 15, 21, 6, 7, 27, 2, 28, 4, 4, 13, 9, 14, 18, 9, 4, 3, 4, 6, 4, 17, 5, 13, 9, 10, 7, 19, 17, 2, 5, 9, 5, 15, 14,
 8, 4, 13, 5, 4, 22, 3, 5, 3, 33]
10.154
```

$C = 300, \gamma = 0.7, ave = 10.157$



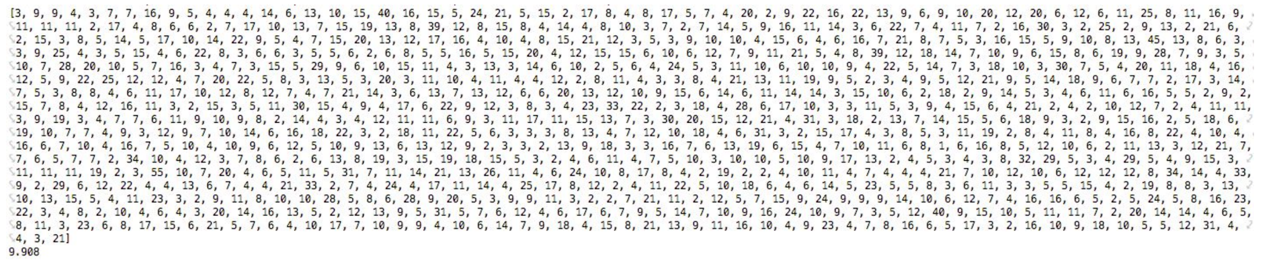
```
[7, 5, 5, 2, 5, 8, 9, 4, 3, 2, 7, 17, 7, 3, 1, 6, 14, 5, 10, 7, 6, 5, 13, 8, 12, 21, 21, 5, 19, 5, 3, 19, 33, 16, 4, 5, 8, 23, 11, 14, 10, 15, 1, 3, 7, 14, 18, 5, 8, 19, 8, 12, 26, 24, 6, 10, 13, 2, 3,
 8, 3, 7, 5, 4, 13, 16, 4, 36, 11, 2, 5, 18, 18, 6, 4, 8, 8, 8, 21, 9, 29, 8, 12, 25, 7, 52, 10, 11, 3, 10, 10, 18, 16, 6, 15, 8, 6, 4, 6, 6, 4, 16, 4, 7, 7, 14, 28, 12, 9, 17, 22, 11, 3, 8, 18, 6, 5,
 3, 15, 18, 4, 3, 28, 7, 4, 7, 9, 21, 9, 7, 27, 5, 11, 29, 3, 2, 5, 4, 16, 8, 5, 9, 3, 12, 42, 10, 6, 6, 18, 6, 2, 26, 3, 3, 8, 14, 9, 14, 6, 11, 15, 31, 5, 8, 9, 16, 16, 14, 25, 7, 8, 16, 13, 7, 9, 2,
 3, 11, 19, 30, 9, 24, 7, 5, 6, 9, 4, 16, 3, 14, 2, 6, 4, 2, 6, 5, 16, 8, 20, 11, 6, 21, 5, 8, 3, 15, 14, 2, 6, 15, 11, 6, 4, 7, 15, 14, 4, 26, 4, 14, 3, 13, 4, 13, 5, 3, 15, 5, 7, 6, 5, 11, 13, 4, 12,
 16, 16, 29, 17, 9, 16, 2, 8, 14, 6, 5, 44, 17, 15, 12, 8, 6, 4, 2, 9, 15, 5, 15, 5, 8, 5, 25, 5, 5, 57, 32, 4, 9, 9, 14, 25, 3, 7, 10, 16, 15, 38, 23, 6, 17, 7, 35, 6, 15, 11, 8, 5, 3, 19, 15, 22, 9,
 4, 2, 10, 28, 2, 8, 3, 11, 2, 6, 7, 3, 3, 4, 8, 5, 4, 5, 3, 19, 5, 16, 14, 4, 6, 5, 3, 5, 14, 6, 2, 7, 20, 10, 9, 9, 24, 9, 7, 4, 8, 4, 1, 12, 8, 11, 21, 11, 16, 11, 13, 9, 2, 4, 8, 5, 10, 4, 16, 10,
 2, 18, 3, 6, 9, 12, 6, 12, 10, 22, 5, 7, 15, 4, 27, 12, 11, 4, 28, 18, 6, 4, 3, 6, 7, 9, 15, 5, 7, 8, 8, 9, 6, 6, 9, 7, 4, 9, 11, 3, 10, 18, 4, 8, 2, 2, 19, 3, 2, 11, 17, 18, 12, 7, 10, 11, 8, 3, 13,
 16, 4, 25, 3, 3, 18, 11, 12, 11, 7, 18, 8, 25, 10, 8, 6, 20, 4, 28, 6, 4, 4, 8, 5, 12, 11, 7, 11, 20, 33, 25, 8, 11, 16, 15, 17, 5, 10, 7, 4, 11, 12, 17, 2, 10, 14, 15, 21, 12, 4, 21, 8, 4, 19, 5, 3,
 10, 4, 8, 8, 10, 18, 14, 14, 15, 18, 4, 9, 9, 6, 12, 13, 5, 17, 8, 23, 3, 3, 12, 2, 6, 5, 17, 9, 22, 13, 3, 3, 25, 7, 18, 20, 6, 24, 13, 5, 8, 8, 23, 26, 8, 7, 15, 25, 11, 16, 3, 3, 15, 5, 5, 5, 1, 7,
 5, 17, 3, 2, 13, 7, 6, 14, 6, 14, 14, 6, 6, 11, 7, 13, 29, 2, 16, 28, 4, 16, 13, 5, 20, 18, 3, 11, 18, 8, 12, 20, 19, 4, 9, 10, 10, 19, 10, 12, 9, 15, 4, 5, 4, 12, 3, 7, 28, 5, 21, 10, 14, 12, 16, 3,
 11, 4, 20, 4, 24, 3, 2, 8, 11, 4, 15, 7, 5, 6, 7, 11, 2, 5, 5, 16, 5, 16, 4, 8, 15, 9, 4, 10, 26, 13, 11, 3, 6, 21, 5, 7, 12, 10, 8, 3, 8, 3, 14, 6, 3, 17, 29, 6, 5, 10, 5, 3, 7, 22, 24, 3, 8, 6, 23,
 7, 13, 3, 16, 3, 5, 2, 8, 3, 10, 11, 9, 19, 7, 47, 13, 5, 9, 20, 25, 1, 2, 5, 5, 11, 15, 4, 12, 10, 19, 28, 11, 25, 2, 6, 5, 60, 5, 20, 6, 17, 6, 16, 14, 20, 4, 5, 5, 7, 5, 25, 12, 12, 11, 16, 4, 8,
 20, 3, 21, 14, 4, 3, 2, 18, 14, 25, 7, 8, 22, 11, 19, 3, 8, 8, 11, 21, 42, 5, 7, 6, 6, 5, 8, 14, 9, 5, 6, 6, 18, 4, 7, 11, 4, 5, 3, 16, 3, 4, 18, 4, 16, 3, 13, 3, 18, 5, 10, 10, 6, 9, 4, 12, 22, 13, 5,
 6, 8, 2, 18, 10, 4, 30, 3, 8, 30, 10, 8, 18, 8, 2, 32, 9, 12, 33, 9, 3, 3, 18, 8, 31, 6, 5, 20, 2, 10, 23, 8, 8, 3, 3, 5, 14, 7, 9, 11, 10, 5, 10, 8, 11, 9, 12, 7, 4, 12, 2, 11, 16, 11, 16, 6, 8, 7,
 18, 15, 5, 8, 9, 16, 20, 11, 11, 8, 2, 10, 3, 5, 3, 5, 6, 7, 9, 8, 11, 2, 5, 4, 2, 26, 3, 8, 6, 3, 16, 13, 14, 11, 15, 6, 5, 10, 23, 4, 6, 8, 5, 6, 19, 6, 6, 3, 13, 16, 7, 4, 5, 8, 23, 23, 10, 8, 3,
 15, 3, 7, 13, 3, 2, 3, 6, 10, 8, 6, 6, 11, 11, 4, 8, 12, 19, 22, 2, 11, 13, 3, 4, 2, 7, 14, 11, 5, 8, 12, 17, 5, 15, 3, 10, 4, 4, 10, 12, 3, 2, 6, 27, 1, 4, 13, 11, 18, 3, 4, 5, 9, 13, 15, 5, 7, 12,
 19, 42, 9, 8, 3, 8, 10, 2, 2, 20, 33, 12, 4, 17, 4, 6, 5, 4, 7, 6, 2, 10, 15, 13, 8, 2, 13, 15, 2, 5, 21, 8, 8, 8, 35, 8, 11, 4, 15, 8, 9, 19, 16, 7, 2, 2, 9, 3, 18, 4, 8, 6, 6, 20, 10, 16, 13, 3, 11,
 18, 14, 3, 20, 3, 11, 24, 22, 9, 5]
10.473
```

$C = 400, \gamma = 0.7, ave = 10.473$



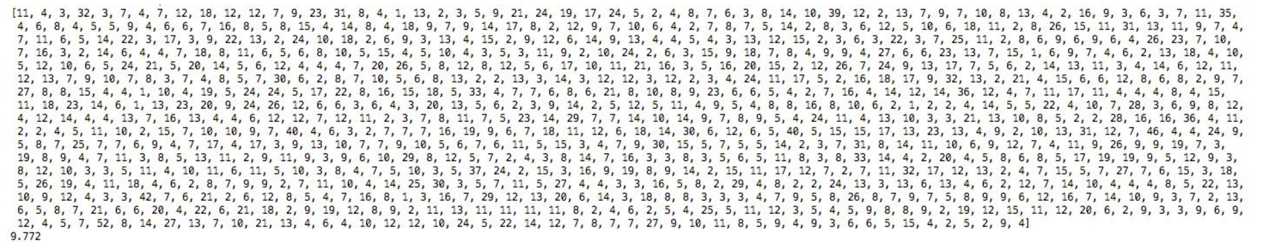
```
[10, 11, 5, 22, 4, 9, 10, 4, 11, 16, 4, 3, 14, 33, 2, 5, 10, 31, 2, 6, 27, 19, 28, 38, 30, 13, 26, 11, 3, 5, 5, 7, 7, 9, 37, 26, 2, 34, 18, 6, 16, 6, 4, 15, 8, 9, 7, 4, 27, 4, 2, 4, 5, 11, 3, 8, 4, 7,
 7, 5, 5, 2, 7, 7, 2, 5, 6, 12, 2, 12, 25, 8, 31, 19, 3, 11, 9, 6, 10, 6, 4, 13, 13, 16, 2, 8, 17, 12, 5, 40, 10, 13, 8, 7, 9, 26, 10, 2, 7, 16, 5, 15, 12, 11, 15, 10, 5, 14, 21, 24, 9, 12, 11, 2, 2, 6,
 26, 8, 8, 20, 13, 6, 3, 3, 28, 28, 14, 5, 2, 19, 18, 27, 11, 5, 20, 13, 4, 11, 23, 11, 10, 6, 20, 18, 15, 14, 5, 11, 16, 4, 9, 3, 7, 3, 16, 4, 19, 9, 7, 6, 24, 7, 4, 17, 16, 19, 9, 25, 6, 7, 20, 18,
 13, 8, 6, 2, 22, 7, 9, 18, 11, 9, 4, 8, 21, 22, 13, 9, 7, 11, 8, 14, 6, 16, 21, 13, 7, 18, 19, 3, 23, 44, 40, 3, 7, 3, 9, 14, 26, 5, 7, 4, 5, 20, 7, 4, 7, 6, 21, 5, 10, 6, 7, 11, 5, 12, 4, 11, 9, 6, 7,
 11, 11, 14, 6, 9, 7, 7, 16, 3, 6, 7, 6, 14, 5, 9, 13, 7, 14, 21, 6, 12, 19, 7, 6, 3, 17, 4, 4, 3, 22, 4, 6, 14, 15, 12, 7, 14, 5, 6, 13, 6, 13, 8, 13, 13, 9, 5, 15, 7, 3, 6, 12, 5, 7, 12, 20, 14, 19,
 12, 3, 5, 6, 3, 9, 8, 10, 4, 6, 12, 21, 17, 9, 14, 13, 18, 3, 6, 2, 26, 7, 30, 17, 9, 23, 8, 38, 48, 6, 13, 25, 2, 3, 8, 8, 7, 9, 7, 19, 13, 17, 26, 14, 4, 3, 35, 6, 8, 5, 6, 27, 7, 12, 10, 26, 13, 27,
 15, 11, 10, 8, 22, 5, 9, 4, 7, 12, 8, 3, 9, 13, 11, 32, 12, 14, 18, 34, 14, 15, 8, 21, 21, 6, 21, 2, 9, 8, 16, 23, 5, 3, 7, 15, 19, 15, 5, 10, 7, 31, 7, 2, 31, 23, 5, 10, 10, 5, 19, 3, 9, 13, 4, 35,
 10, 15, 15, 25, 12, 3, 11, 4, 11, 3, 3, 7, 4, 7, 8, 16, 12, 15, 10, 12, 7, 5, 21, 10, 9, 12, 4, 6, 12, 6, 12, 5, 27, 15, 12, 3, 14, 20, 20, 13, 19, 11, 16, 6, 20, 26, 23, 10, 5, 21, 16, 21, 5, 11, 10,
 6, 13, 47, 6, 10, 6, 5, 20, 28, 3, 3, 6, 7, 6, 22, 34, 7, 11, 17, 11, 16, 5, 11, 9, 10, 2, 7, 2, 11, 35, 9, 8, 52, 9, 6, 5, 9, 3, 25, 3, 12, 4, 15, 21, 5, 3, 15, 9, 6, 8, 6, 12, 15, 8, 8, 5, 11, 11,
 11, 17, 4, 9, 42, 37, 17, 2, 7, 20, 12, 8, 4, 2, 2, 16, 8, 33, 9, 5, 23, 17, 4, 6, 5, 6, 11, 10, 17, 8, 3, 3, 20, 33, 18, 12, 6, 26, 5, 2, 8, 4, 28, 12, 16, 15, 9, 30, 11, 9, 9, 22, 16, 26, 10, 12, 4,
 5, 28, 10, 8, 3, 9, 6, 3, 20, 12, 13, 11, 4, 9, 7, 14, 10, 2, 15, 3, 13, 5, 16, 2, 9, 3, 6, 7, 5, 26, 19, 3, 12, 26, 8, 21, 17, 4, 18, 5, 31, 21, 27, 3, 2, 15, 4, 25, 16, 5, 6, 3, 15, 4, 28, 8, 8, 3,
 12, 14, 9, 3, 7, 7, 9, 4, 40, 10, 3, 45, 23, 5, 23, 26, 5, 7, 5, 5, 13, 7, 3, 2, 11, 14, 17, 9, 3, 3, 6, 19, 8, 1, 9, 5, 5, 6, 14, 27, 7, 24, 14, 6, 21, 5, 10, 15, 27, 12, 21, 11, 4, 7, 7, 6, 17, 15,
 2, 11, 16, 6, 7, 5, 32, 8, 15, 13, 3, 4, 8, 17, 9, 7, 19, 21, 7, 20, 9, 5, 8, 5, 28, 8, 31, 11, 5, 14, 6, 14, 4, 15, 8, 21, 16, 5, 7, 11, 14, 2, 24, 18, 42, 9, 15, 11, 6, 6, 6, 1, 10, 11, 7, 4, 14, 6,
 15, 2, 4, 3, 12, 10, 3, 1, 14, 11, 4, 12, 6, 28, 17, 18, 25, 20, 11, 14, 4, 45, 10, 18, 9, 12, 8, 4, 10, 6, 3, 2, 15, 13, 8, 12, 23, 24, 27, 23, 12, 6, 19, 6, 15, 24, 14, 5, 10, 6, 16, 9, 2, 12, 13, 6,
 6, 4, 15, 17, 3, 4, 9, 12, 9, 3, 3, 8, 12, 2, 9, 12, 9, 5, 21, 6, 2, 7, 6, 8, 9, 33, 5, 7, 9, 49, 6, 23, 8, 17, 21, 17, 5, 8, 10, 2, 14, 10, 4, 28, 7, 10, 16, 9, 9, 8, 5, 19, 28, 4, 32, 4, 28, 23, 12,
 16, 25, 24, 17, 5, 3, 17, 10, 8, 17, 6, 2, 8, 13, 17, 14, 18, 6, 2, 3, 6, 2, 12, 7, 17, 5, 15, 27, 13, 39, 8, 7, 6, 3, 4, 5, 19, 14, 4, 14, 14, 5, 4, 3, 14, 16, 6, 7, 6, 12, 13, 3, 8, 6, 4, 8, 19, 21,
 8, 8, 5, 11, 10, 15, 19, 11, 29, 20, 9, 15, 18, 7, 3, 3, 19, 18, 7, 12, 19, 5, 11, 14, 12, 19, 13, 3, 25, 6, 11, 7, 24, 13, 6, 2, 19, 11, 6, 6, 4, 10, 13, 3, 9, 7, 3, 9, 9, 8, 3, 19, 7, 10, 19, 20, 2,
 33, 15, 26, 18, 3, 13, 14, 15, 8, 3, 8, 11, 6, 15, 16, 9, 4, 5, 12, 13, 20, 17, 7]
11.712
```

$C = 500, \gamma = 0.7, ave = 11.712$



```
[3, 9, 9, 4, 3, 7, 7, 16, 9, 5, 4, 4, 4, 14, 6, 13, 10, 15, 40, 16, 15, 5, 24, 21, 5, 15, 2, 17, 8, 4, 8, 17, 5, 7, 4, 20, 2, 9, 22, 16, 22, 13, 9, 6, 9, 10, 20, 12, 20, 6, 12, 6, 11, 25, 8, 11, 16, 9,
11, 11, 11, 2, 17, 4, 8, 6, 6, 2, 7, 17, 10, 13, 7, 15, 19, 13, 8, 39, 12, 8, 15, 8, 4, 14, 4, 8, 10, 3, 7, 2, 7, 14, 5, 9, 16, 11, 14, 3, 6, 22, 7, 4, 11, 7, 2, 16, 30, 3, 2, 25, 2, 9, 13, 2, 21, 6,
2, 15, 3, 8, 5, 14, 5, 17, 10, 14, 22, 9, 5, 4, 7, 15, 20, 13, 12, 17, 16, 4, 10, 4, 8, 15, 21, 12, 3, 5, 3, 9, 10, 10, 4, 15, 6, 4, 6, 16, 7, 21, 8, 7, 5, 3, 16, 15, 5, 9, 10, 8, 13, 45, 13, 8, 6, 3,
3, 9, 25, 4, 3, 5, 15, 4, 6, 22, 8, 3, 6, 6, 3, 5, 5, 6, 2, 6, 8, 5, 5, 16, 5, 15, 20, 4, 12, 15, 15, 6, 10, 6, 12, 7, 9, 11, 21, 5, 4, 8, 39, 12, 18, 14, 7, 10, 9, 6, 15, 8, 6, 19, 9, 28, 7, 9, 3, 5,
10, 7, 28, 20, 10, 5, 7, 16, 3, 4, 7, 3, 15, 5, 29, 9, 6, 10, 15, 11, 4, 3, 13, 3, 14, 6, 10, 2, 5, 6, 4, 24, 5, 3, 11, 10, 6, 10, 10, 9, 4, 22, 5, 14, 7, 3, 18, 10, 3, 30, 7, 5, 4, 20, 11, 18, 4, 16,
12, 5, 9, 22, 25, 12, 12, 4, 7, 20, 22, 5, 8, 3, 13, 5, 3, 20, 3, 11, 10, 4, 11, 4, 4, 12, 2, 8, 11, 4, 3, 3, 8, 4, 21, 13, 11, 19, 9, 5, 2, 3, 4, 9, 5, 12, 21, 9, 5, 14, 18, 9, 6, 7, 7, 2, 17, 3, 14,
7, 5, 3, 8, 8, 4, 6, 11, 17, 10, 12, 8, 12, 7, 4, 7, 21, 14, 3, 6, 13, 7, 13, 12, 6, 6, 20, 13, 12, 10, 9, 15, 6, 14, 6, 11, 14, 14, 3, 15, 10, 6, 2, 18, 2, 9, 14, 5, 3, 4, 6, 11, 6, 16, 5, 5, 2, 9, 2,
15, 7, 8, 4, 12, 16, 11, 3, 2, 15, 3, 5, 11, 30, 15, 4, 9, 4, 17, 6, 22, 9, 12, 3, 8, 3, 4, 23, 33, 22, 2, 3, 18, 4, 28, 6, 17, 10, 3, 3, 11, 5, 3, 9, 4, 15, 6, 4, 21, 2, 4, 2, 10, 12, 7, 2, 4, 11, 11,
3, 9, 19, 3, 4, 7, 7, 6, 11, 9, 10, 9, 8, 2, 14, 4, 3, 4, 12, 11, 11, 6, 9, 3, 11, 17, 11, 15, 13, 7, 3, 30, 20, 15, 12, 21, 4, 31, 3, 18, 2, 13, 7, 14, 15, 5, 6, 18, 9, 3, 2, 9, 15, 16, 2, 5, 18, 6,
19, 10, 7, 7, 4, 9, 3, 12, 9, 7, 10, 14, 6, 16, 18, 22, 3, 2, 18, 11, 22, 5, 6, 3, 3, 3, 8, 13, 4, 7, 12, 10, 18, 4, 6, 31, 3, 2, 15, 17, 4, 3, 8, 5, 3, 11, 19, 2, 8, 4, 11, 8, 4, 16, 8, 22, 4, 10, 4,
16, 6, 7, 10, 4, 16, 7, 5, 10, 4, 10, 9, 6, 12, 5, 10, 9, 13, 6, 13, 12, 9, 2, 3, 3, 2, 13, 9, 18, 3, 3, 16, 7, 6, 13, 19, 6, 15, 4, 7, 10, 11, 6, 8, 1, 6, 16, 8, 5, 12, 10, 6, 2, 11, 13, 3, 12, 21, 7,
7, 6, 5, 7, 7, 2, 34, 10, 4, 12, 3, 7, 8, 6, 2, 6, 13, 8, 19, 3, 15, 19, 18, 15, 5, 3, 2, 4, 6, 11, 4, 7, 5, 10, 3, 10, 10, 5, 10, 9, 17, 13, 2, 4, 5, 3, 4, 3, 8, 32, 29, 5, 3, 4, 29, 5, 4, 9, 15, 3,
11, 11, 11, 19, 2, 3, 55, 10, 7, 20, 4, 6, 5, 11, 5, 31, 7, 11, 14, 21, 13, 26, 11, 4, 6, 24, 10, 8, 17, 8, 4, 2, 19, 2, 2, 4, 10, 11, 4, 7, 4, 4, 4, 21, 7, 10, 12, 10, 6, 12, 12, 12, 8, 34, 14, 4, 33,
9, 2, 29, 6, 12, 22, 4, 4, 13, 6, 7, 4, 4, 21, 33, 2, 7, 4, 24, 4, 17, 11, 14, 4, 25, 17, 8, 12, 2, 4, 11, 22, 5, 10, 18, 6, 4, 6, 14, 5, 23, 5, 5, 8, 3, 6, 11, 3, 3, 5, 5, 15, 4, 2, 19, 8, 8, 3, 13,
10, 13, 15, 5, 4, 11, 23, 3, 2, 9, 11, 8, 10, 10, 28, 5, 8, 6, 28, 9, 20, 5, 3, 9, 9, 11, 3, 2, 2, 7, 21, 11, 2, 12, 5, 7, 15, 9, 24, 9, 9, 9, 14, 10, 6, 12, 7, 4, 16, 16, 6, 5, 2, 5, 24, 5, 8, 16, 23,
22, 3, 4, 8, 2, 10, 4, 6, 4, 3, 20, 14, 16, 13, 5, 2, 12, 13, 9, 5, 31, 5, 7, 6, 12, 4, 6, 17, 6, 7, 9, 5, 14, 7, 10, 9, 16, 24, 10, 9, 7, 3, 5, 12, 40, 9, 15, 10, 5, 11, 11, 7, 2, 20, 14, 14, 4, 6, 5,
8, 11, 3, 23, 6, 8, 17, 15, 6, 21, 5, 7, 6, 4, 10, 17, 7, 10, 9, 9, 4, 10, 6, 14, 7, 9, 18, 4, 15, 8, 21, 13, 9, 11, 16, 10, 4, 9, 23, 4, 7, 8, 16, 6, 5, 17, 3, 2, 16, 10, 9, 18, 10, 5, 5, 12, 31, 4,
4, 3, 21]
9.908
```

$C = 600, \gamma = 0.7, ave = 9.908$



```
[11, 4, 3, 32, 3, 7, 4, 7, 12, 18, 12, 12, 7, 9, 23, 31, 8, 4, 1, 13, 2, 3, 5, 9, 21, 24, 19, 17, 24, 5, 2, 4, 8, 7, 6, 3, 8, 14, 10, 39, 12, 2, 13, 7, 9, 7, 10, 8, 13, 4, 2, 16, 9, 3, 6, 3, 7, 11, 35,
 4, 6, 8, 4, 5, 5, 9, 4, 6, 6, 7, 16, 8, 5, 8, 15, 4, 14, 8, 4, 18, 9, 7, 9, 14, 17, 8, 2, 12, 9, 7, 10, 6, 4, 2, 7, 8, 7, 5, 14, 2, 8, 3, 6, 12, 5, 10, 6, 18, 11, 2, 8, 26, 15, 11, 31, 13, 11, 9, 7, 4,
 7, 11, 6, 5, 14, 22, 3, 17, 3, 9, 22, 13, 2, 24, 10, 18, 2, 6, 9, 3, 13, 4, 15, 2, 9, 12, 6, 14, 9, 13, 4, 4, 5, 4, 3, 13, 12, 15, 2, 3, 6, 3, 22, 3, 7, 25, 11, 2, 8, 6, 9, 6, 9, 6, 4, 26, 23, 7, 10,
 7, 16, 3, 2, 14, 6, 4, 4, 7, 18, 8, 11, 6, 5, 6, 8, 10, 5, 15, 4, 5, 10, 4, 3, 5, 3, 11, 9, 2, 10, 24, 2, 6, 3, 15, 9, 18, 7, 8, 4, 9, 9, 4, 27, 6, 6, 23, 13, 7, 15, 1, 6, 9, 7, 4, 6, 2, 13, 18, 4, 10,
 5, 12, 10, 6, 5, 24, 21, 5, 20, 14, 5, 6, 12, 4, 4, 4, 7, 20, 26, 5, 8, 12, 8, 12, 5, 6, 17, 10, 11, 21, 16, 3, 5, 16, 20, 15, 2, 12, 26, 7, 24, 9, 13, 17, 7, 5, 6, 2, 14, 13, 11, 3, 4, 14, 6, 12, 11,
 12, 13, 7, 9, 10, 7, 8, 3, 7, 4, 8, 5, 7, 30, 6, 2, 8, 7, 10, 5, 6, 8, 13, 2, 2, 13, 3, 14, 3, 12, 12, 3, 12, 2, 3, 4, 24, 11, 17, 5, 2, 16, 18, 17, 9, 32, 13, 2, 21, 4, 15, 6, 6, 12, 8, 6, 8, 2, 9, 7,
 27, 8, 8, 15, 4, 4, 1, 10, 4, 19, 5, 24, 24, 5, 17, 22, 8, 16, 15, 18, 5, 33, 4, 7, 7, 6, 8, 6, 21, 8, 10, 8, 9, 23, 6, 6, 5, 4, 2, 7, 16, 4, 14, 12, 14, 36, 12, 4, 7, 11, 17, 11, 4, 4, 4, 8, 4, 15,
 11, 18, 23, 14, 6, 1, 13, 23, 20, 9, 24, 26, 12, 6, 6, 3, 6, 4, 3, 20, 13, 5, 6, 2, 3, 9, 14, 2, 5, 12, 5, 11, 4, 9, 5, 4, 8, 8, 16, 8, 10, 6, 2, 1, 2, 2, 4, 14, 5, 5, 22, 4, 10, 7, 28, 3, 6, 9, 8, 12,
 4, 12, 14, 4, 4, 13, 7, 16, 13, 4, 4, 6, 12, 12, 7, 12, 11, 2, 3, 7, 8, 11, 7, 5, 23, 14, 29, 7, 7, 14, 10, 14, 9, 7, 8, 9, 5, 4, 24, 11, 4, 13, 10, 3, 3, 21, 13, 10, 8, 5, 2, 2, 28, 16, 16, 36, 4, 11,
 2, 2, 4, 5, 11, 10, 2, 15, 7, 10, 10, 9, 7, 40, 4, 6, 3, 2, 7, 7, 7, 16, 19, 9, 6, 7, 18, 11, 12, 6, 18, 14, 30, 6, 12, 6, 5, 40, 5, 15, 15, 17, 13, 23, 13, 4, 9, 2, 10, 13, 31, 12, 7, 46, 4, 4, 24, 9,
 5, 8, 7, 25, 7, 7, 6, 9, 4, 7, 17, 4, 17, 3, 9, 13, 10, 7, 7, 9, 10, 5, 6, 7, 6, 11, 5, 15, 3, 4, 7, 9, 30, 15, 5, 7, 5, 5, 14, 2, 3, 7, 31, 8, 14, 11, 10, 6, 9, 12, 7, 4, 11, 9, 26, 9, 9, 19, 7, 3,
 19, 8, 9, 4, 7, 11, 3, 8, 5, 13, 11, 2, 9, 11, 9, 3, 9, 6, 10, 29, 8, 12, 5, 7, 2, 4, 3, 8, 14, 7, 16, 3, 3, 8, 3, 5, 6, 5, 11, 8, 3, 8, 33, 14, 4, 2, 20, 4, 5, 8, 6, 8, 5, 17, 19, 19, 9, 5, 12, 9, 3,
 8, 12, 10, 3, 3, 5, 11, 4, 10, 11, 6, 11, 5, 10, 3, 8, 4, 7, 5, 10, 3, 5, 37, 24, 2, 15, 3, 16, 9, 19, 8, 9, 14, 2, 15, 11, 17, 12, 7, 2, 7, 11, 32, 17, 12, 13, 2, 4, 7, 15, 5, 7, 27, 7, 6, 15, 3, 18,
 5, 26, 19, 4, 11, 18, 4, 6, 2, 8, 7, 9, 9, 2, 7, 11, 10, 4, 14, 25, 30, 3, 5, 7, 11, 5, 27, 4, 4, 3, 3, 16, 5, 8, 2, 29, 4, 8, 2, 2, 24, 13, 3, 13, 6, 13, 4, 6, 2, 12, 7, 14, 10, 4, 4, 4, 8, 5, 22, 13,
 10, 9, 12, 4, 3, 3, 42, 7, 6, 21, 2, 6, 12, 8, 5, 4, 7, 16, 8, 1, 3, 16, 7, 29, 12, 13, 20, 6, 14, 3, 18, 8, 8, 3, 3, 3, 4, 7, 9, 5, 8, 26, 8, 7, 9, 7, 5, 8, 9, 9, 6, 12, 16, 7, 14, 10, 9, 3, 7, 2, 13,
 6, 5, 8, 7, 21, 6, 6, 20, 4, 22, 6, 21, 18, 2, 9, 19, 12, 8, 9, 2, 11, 13, 11, 11, 11, 11, 8, 2, 4, 6, 2, 5, 4, 25, 5, 11, 12, 3, 5, 4, 5, 9, 8, 8, 9, 2, 19, 12, 15, 11, 12, 20, 6, 2, 9, 3, 3, 9, 6, 9,
 12, 4, 5, 7, 52, 8, 14, 27, 13, 7, 10, 21, 13, 4, 6, 4, 10, 12, 12, 10, 24, 5, 22, 14, 12, 7, 8, 7, 7, 27, 9, 10, 11, 8, 5, 9, 4, 9, 3, 6, 6, 5, 15, 4, 2, 5, 2, 9, 4]
9.772
```

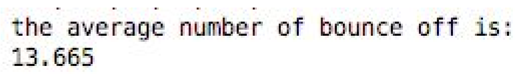
$C = 1000, \gamma = 0.7, ave = 9.772$

We can see from the results above that, when $C = 500, \gamma = 0.7$, the agent works the best. We think the reason why the *C* value is so large is that, the action of bouncing successfully is usually found after many failures. If *C* is small, meaning the learning rate decays quickly, the reward would be pretty small when the right action is found. That also leads to the idea of modifying the reward to get a better result, which will be discussed in the next sub-section.

(3) Change in MDP and other constants

Several possible changes of constants can be helpful. We can have a finer mesh of the board, and as mentioned above, have different reward values.

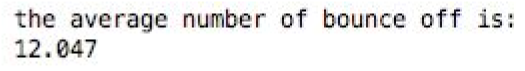
Firstly, we treat the board as a 15*15 grid. Now the number of states becomes 15*15*2*3*15+1 = 20251 instead of 10369. As a trade-off, the time taken is significantly longer, but it also shows better performance.



```
the average number of bounce off is:
13.665
```

The average number of bounce off is 13.665 now, higher than any cases above.

Changing the reward values can also be beneficial. Before, the rewards are set as instructed, +1 when the ball bounces off the paddle, -1 when the ball passes the paddle, and 0 otherwise. Because the final goal of the problem is to avoid the ball passing the paddle as much as possible, we set the reward of failure to be -5 to punish this case more.



```
the average number of bounce off is:
12.047
```

Both changes in parameters lead to better results. It is also trivial that if we increase the number of training games, the performance can also be improved.

4.2. Two-Player Pong (4-credit)

In part 2, we modify our Markov Decision Process by adding one more parameter in state: (ball_x, ball_y, velocity_x, velocity_y, paddle1_y, paddel2_y), which leads to the negative side-effect on increasing the time to train data. To fit our implementation, we add new check function, rebound function, and moving function for the new left paddle.

According to the result, we can see that the winning rate is still near 90%. But the average of bounce decreases to around 4.

In the condition such that $C = 500, \gamma = 0.7$, the relationship between the number of games and the average number of rebounce is like:

Number of games	Average number of rebounce	Wining rate
25000	3.71	78.1%
50000	4.02	82.4%

100000	4.06	86.3%
150000	4.11	89.1%

4.3 Extra Credit: Graphical Representation and Animation

The Pong game simulation is run on Python. We output all the states to a csv file and run the animation on Matlab. The screenshots of the animations for each part are shown below. The movies are included in the zip folder as .avi files, and the matlab code that generates the animation is also included in the folder.

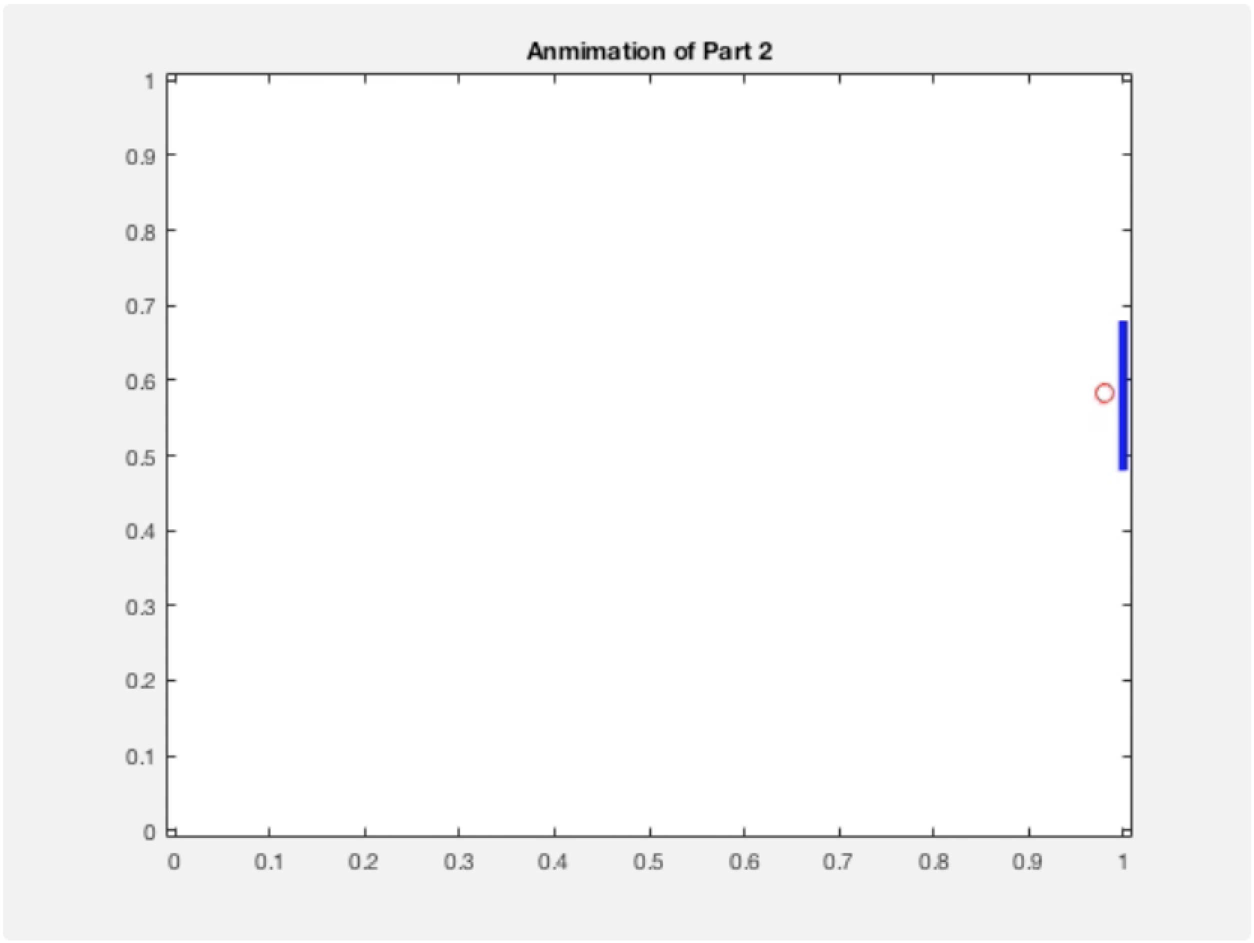


Fig 4. Screenshot of Animation for Part 2.1

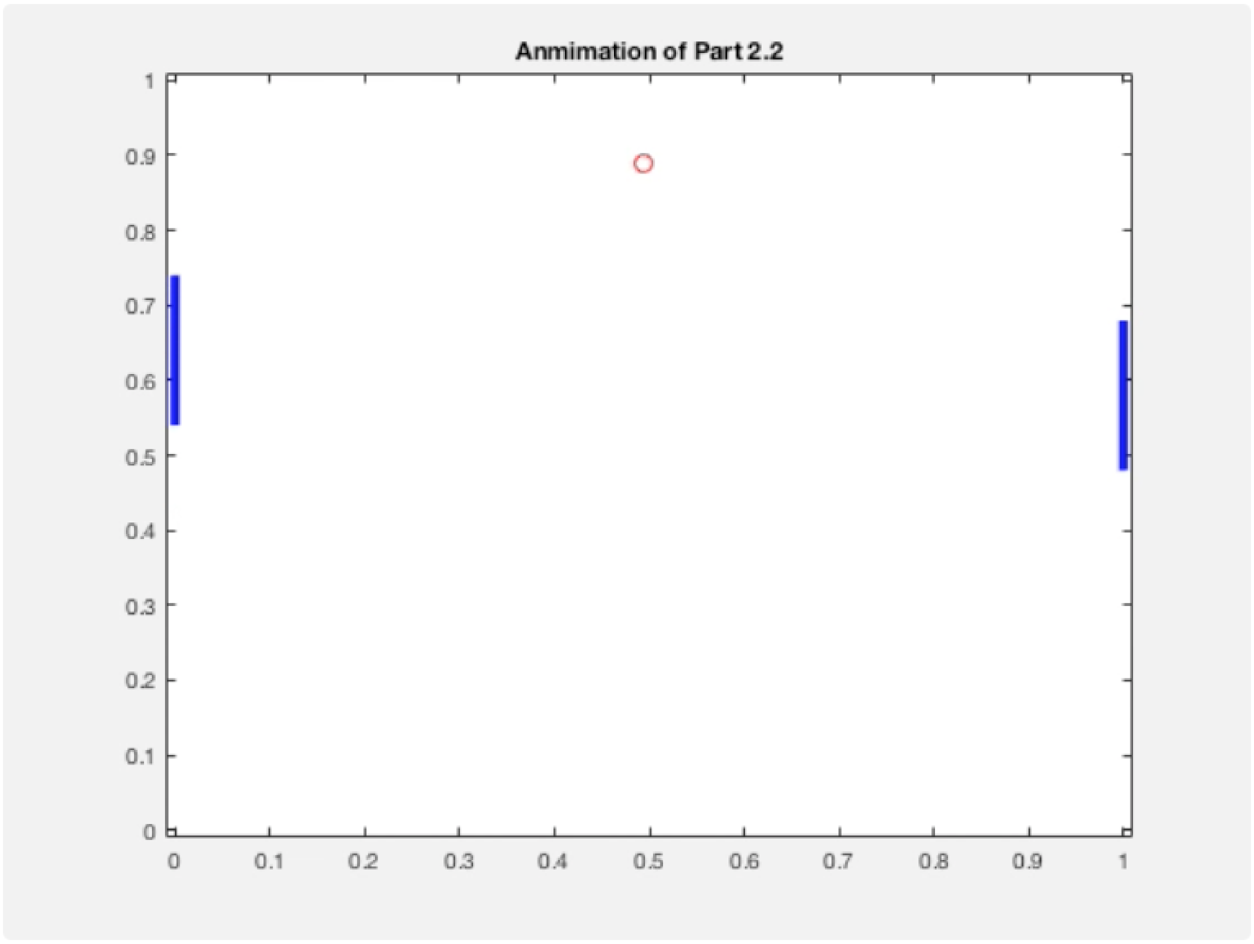


Fig 5. Screenshot of Animation for Part 2.2