ECE448

MP4

Perceptrons and Reinforcement Learning

Date:2017/12/10

Credit: 3

Section: R

Team Member:

Jiaying Wu (jwu86)

Qin Li (qinli2)

Yuchen Liang (yliang35)

Part 1.1:Digit Classification with Perceptrons

Description:

In this part, we build a multi-class (non-differentiable) perceptron, which is made by several one-layer perceptron for each digital class. The images are of size 28*28, so there are 784 pixels (image vector **x** in length of 784) in total, and the value of the pixel is either 1 or 0. For each pixel, there are one number to represent the weight corresponding that pixel. Therefore, the length for the weight vector **w** is also 784. The output of such perceptron is represented as $\mathbf{w}^T\mathbf{x}+\mathbf{b}$, in which **b** represents a bias number for the current class. We found out that adding a bias number for each perceptron of the class will increase the accuracy of the model. The argmax of the output values for each class will be the final output of the classifier.

During the training process, we firstly set a learning rate **α** as $\alpha = \frac{1}{number\ of\ epoch^2}$ (the reason for choosing this number will be demonstrated below). For each training sample, if the image is in class **c** and it is misclassified as **c'**, update the value of **w** using

$$\mathbf{w_c} = \mathbf{w_c} + \alpha\mathbf{x}$$

$$\mathbf{w_{c'}} = \mathbf{w_{c'}} - \alpha\mathbf{x}$$

When training the bias value, we treat the bias as a weight number corresponding to a pixel with fixed value 1. Therefore, similar to the training process for the weight,

$$\mathbf{b_c} = \mathbf{b_c} + \alpha*1$$

$$\mathbf{b_{c'}} = \mathbf{b_{c'}} - \alpha*1$$

The weight and bias are initialized by random numbers since it will gain higher accuracy compared to initialize to zero although the accuracy varies a bit due to randomness. Also, we choose random samples during the training, since the accuracy will be decreased when the ordered samples are used for training.

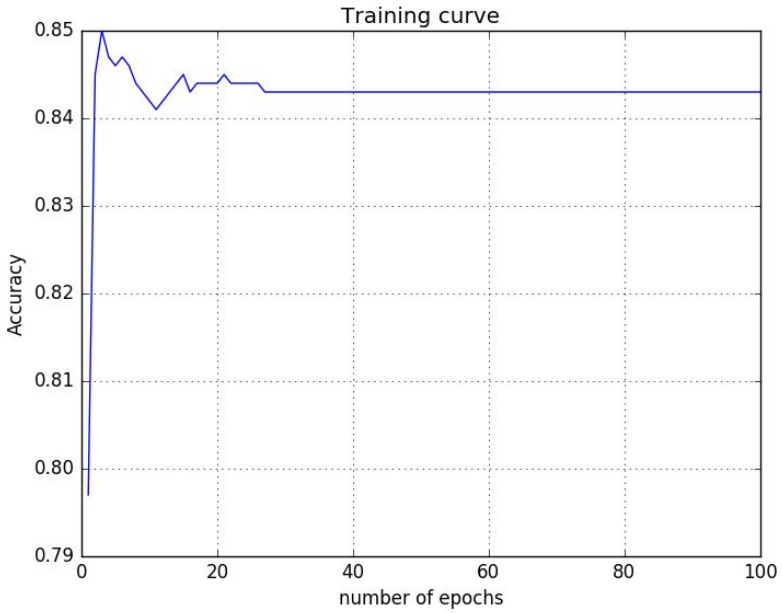
When choosing the value of the parameters, we firstly analyze different learning rate decay function and the training curve. Finally, we decided to use $\alpha = \frac{1}{number\ of\ epoch^2}$, since this equation can obtain the highest accuracy as well as a relatively stable training curve.

The number of epoch we choose is 3. According to the training curve below, the accuracy will achieve its maximum when the number of epoch for training is 3.

The overall accuracy is 0.85.

Compared to the accuracy of Naive Bayes method(0.772), the accuracy of Digit Classification with Perceptrons(0.85) is higher.

The **Training Curve** and **Confusion Matrix** are shown below:




```
The overall accuracy is: 0.85
The confusion matrix is:
[[ 0.911  0.     0.011  0.     0.     0.     0.011  0.011  0.033  0.022]
 [ 0.     0.981  0.009  0.     0.     0.     0.009  0.     0.     0.   ]
 [ 0.     0.01   0.864  0.01   0.01   0.     0.058  0.019  0.029  0.   ]
 [ 0.     0.     0.02   0.84   0.     0.04   0.03   0.06   0.01   0.   ]
 [ 0.     0.     0.019  0.009  0.85   0.     0.037  0.009  0.019  0.056]
 [ 0.011  0.     0.011  0.065  0.     0.804  0.     0.022  0.087  0.   ]
 [ 0.011  0.011  0.033  0.     0.033  0.011  0.868  0.022  0.011  0.   ]
 [ 0.     0.019  0.075  0.019  0.     0.     0.     0.802  0.     0.085]
 [ 0.01   0.01   0.039  0.068  0.019  0.058  0.019  0.019  0.748  0.01 ]
 [ 0.     0.     0.     0.04   0.04   0.03   0.     0.05   0.01   0.83 ]]
```

EXTRA:Part 1.2:Digit Classification with Nearest Neighbor

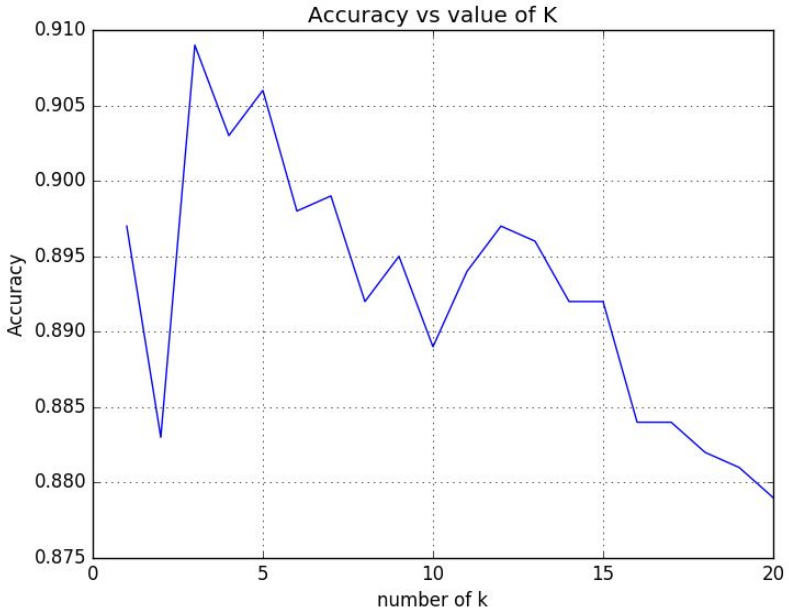
Description:

In this part, we implement a k-nearest-neighbor classifier for the digit classification. Firstly, we load all of the sample images into a huge matrix. Since there are 5000 samples in total, and there are 784 pixels for each image, we build a 5000*784 matrix to store all the data. The value of the pixel is either 1 or 0. Also, we made a vector of length 5000 to store all of the correct label for those training samples.

During the testing, we firstly convert the image into a vector of length 784, and then calculate the distance between the vector and the 5000 training sample vectors respectively. After calculating the values, we find the k nearest values and find its labels. The most common label will be the output for the classifier.

The method for calculating the distance we use is the manhattan distance. Thus, we calculate the absolute difference for each element of the vectors and sum them together.

The value of k we choose is 3, since it will reach the maximum accuracy of the classifier. The relationship between k and the accuracy is shown below.



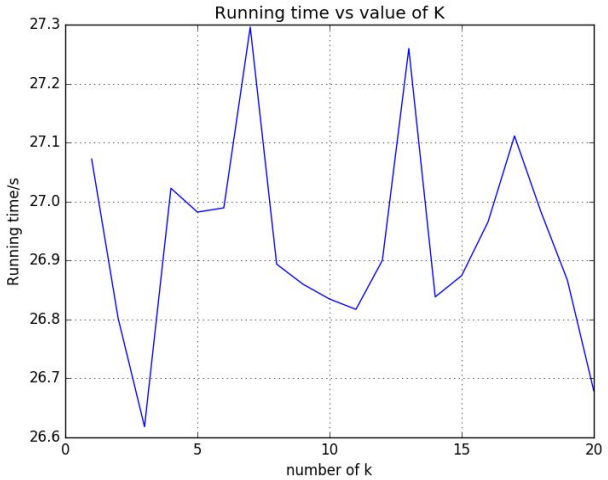
The overall accuracy is 0.909.

Compare to the accuracy of perceptron classifier (0.85) and Naive Bayes model(0.772), the k-nearest-neighbor classifier has the highest accuracy(0.909).

Confusion Matrix:

```
The overall accuracy is: 0.909
The confusion matrix is:
[[ 1.     0.     0.     0.     0.     0.     0.     0.     0.     0.   ]
 [ 0.     1.     0.     0.     0.     0.     0.     0.     0.     0.   ]
 [ 0.01   0.019  0.922  0.029  0.     0.     0.01   0.01   0.     0.   ]
 [ 0.     0.01   0.     0.84   0.     0.05   0.     0.03   0.05   0.02 ]
 [ 0.     0.019  0.     0.     0.907  0.     0.028  0.     0.     0.047]
 [ 0.022  0.     0.     0.033  0.     0.859  0.033  0.011  0.033  0.011]
 [ 0.     0.011  0.     0.     0.     0.011  0.978  0.     0.     0.   ]
 [ 0.     0.075  0.009  0.     0.019  0.     0.     0.83   0.     0.066]
 [ 0.01   0.01   0.     0.039  0.029  0.019  0.     0.019  0.864  0.01 ]
 [ 0.     0.     0.     0.01   0.03   0.02   0.     0.01   0.03   0.9  ]]
```

The value of running time for different value of k are similar (about 27 seconds), since we used the sorting function in python to sort the distances between the current test sample and all of the samples respectively. For different k values, we just take different amount of numbers from those sorted array. Thus, the time it takes for different k values are similar.



Part 2.1. Single Player Pong

Description:

In this part of the MP, we implement a Q learning algorithm which trains our AI to bounce the ball at maximum rate. The Pong game is represented by state parameters: ball's x position (bx), ball's y position (by), ball's x velocity (vx), ball's y velocity (vy) and paddle-top's y position (py). In every iteration (round of game), the states representing the ball being bounced back from the wall have reward +1, and those representing the paddle missing the ball have reward -1. Besides, the latter states are directed to an ending state.

In the Q learning, the states are divided into discrete states. Within the five continuous state parameters, bx, by, and py are subdivided into 12 discrete states. Vx is divided with respect to its sign (2 states), and vy is subdivided with respect to its magnitude and sign (3 states, 0 if the magnitude is too small). We use 12*12*2*3*12+1 array to store the state-utility matrix in the Q learning (the total number of states plus an ending state). We calculate the index for every state with the five parameters bx,by,vx,vy and py by hashing them onto an integer number. Then the state index is used in the Q matrix which helps us to determine the next action. The learning rate is defined by C/(C+N(s,a)), where N(s,a) is the number of times that the state-action pair (s,a) has appeared. The equation used for Q learning is

$$Q^{new}(s,a) = Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$. In the next section, we choose the optimal hyper-parameters.

2.1.1:Choice of parameter:

C	5
γ	0.9

exploration function	Ne = 10, R = 0.3
Number of stimulation to learn a good policy	100000

2.1.2:Justification of our choices:

Choice of α and γ:

C1	Gamma	Average Bouncing
5	0.7	8.836
5	0.8	8.219
5	0.9	8.951
5	0.95	7.499

C2	Gamma	Average Bouncing
7	0.7	8.871
7	0.8	8.015
7	0.9	8.605
7	0.95	7.073

C3	Gamma	Average Bouncing
10	0.7	7.634
10	0.8	8.936
10	0.9	7.954
10	0.95	7.771

C4	Gamma	Average Bouncing
15	0.7	6.683
15	0.8	5.586
15	0.9	7.324
15	0.95	6.98

C5	Gamma	Average Bouncing
20	0.7	6.899
20	0.8	7.063
20	0.9	7.637
20	0.95	6.363

According to the testing, we get the data as shown in the table above. From the tables, we can see that with same α value, when γ=0.9, the highest average bouncing rate is achieved. Similarly, with same γ value, we can observe that when α is 5, the highest bouncing rate can be achieved.

Choice of exploration function:

C	gamma	R	Ne	Average Bouncing
5	0.9	-0.1	5	0
5	0.9	0	5	0
5	0.9	0.1	5	8.779
5	0.9	0.2	5	8.933
5	0.9	0.3	5	7.748
5	0.9	0.4	5	7.79
5	0.9	0.5	5	9.571
5	0.9	0.2	10	7.989
5	0.9	0.3	10	9.8
5	0.9	0.4	10	9.726

With alpha=5 and gamma=0.9, we have tried different combination of R and Ne, and the result is shown on the table above. According to the table, we can see that when R=0.3, the highest average bouncing rate is achieved. Among different choice of Ne, Ne=10 is the way that can keep highest average bouncing.

2.1.3:Training number of Convergence

Training number	Average time of Bouncing

10	0
100	0
1000	1.261
10000	3.319
50000	8.777
100000	8.988
200000	9.475
300000	9.563

According to the table shown above, we can see that the average time of bouncing increases as the training time increases. Our average number of bouncing converges at around 200,000 trials. The average bouncing becomes around 9.5.

2.1.4: Change of MDP and constants:

In our testing, we have tried to change the MDP and constants used to discretize the continuous state space we have made. The result is shown in the table below:

boundary of velocity	state table size	paddle height	running time(s)	average bouncing rate
0.15	12*12*3*2*12	0.8	1573.39	8.988
0.15	14*14*3*2*14	0.8	1625.85	8.913
0.15	12*12*3*2*12	0.4	77.524	0
0.15	12*12*3*2*12	1.2	1297.11	7.113
0.3	12*12*3*2*12	0.8	1527.63	8.927
0.15	8*8*3*2*8	0.8	1016.49	4.956

From the table, we can observe that as the size of state table increase, the running time of execution increases while the average bouncing rate does not increase. When the size of the state table decrease, the bouncing rate significantly decreases though its running time decreases. The change of boundary when determine the velocity seems do not have much effect on the accuracy and running time. The paddle

height have considerable effect on the running time and bouncing rate. When the height of paddle is too low, the accuracy goes to zero. When the paddle height is too high, the bouncing rate decreases as well.

Part 2.2.Double player Pong (Extra Credit)

Description:

In this part of project, we made a second hard code AI, which moves toward the ball. To beat the second hard code AI , we have tried to make several changes which are speed of paddles, size of state table, and considering hardcode AI's loss in the reward function. The analyze of these changes is described in the following:

Speed of paddles:

Speed of paddle	Winning rate
0.04	45.7%
0.08	53.4%
0.12	39.2%

This is the only change that makes our AI have better performance. According to the table above, we can see that when we change the speed of paddle, the winning rate increases when the paddle speed doubles. However, when the paddle speed triples, the negative effects shows up. With speed 0.12, the paddle moves too fast that it cannot be stable at one point to bounce the ball off, and it will always moves around that makes it lose the ball. Therefore, the winning rate goes down.

Size of state table：

We have changed the size of state table, but 12*12 gives us the best performance.

Considering the hard code AI in our AI's reward:

We have made a change to reward our AI, if the hardcode AI did not get the ball. However, after we include it in our code, the performance of our AI becomes worse. Our guess is because when we do not include hard code AI, we are assuming the ball will always be bounced back from the opposite side, so our AI will focus on how to get the ball; while training our AI to give beat the hardcode AI by giving the hardcode AI a ball in a extreme angle will make our AI performs worse.

Extra Point: Part2 Make an animation of Pong game !

Statement of individual contribution:

Qin Li: Part1 Coding, Part1 report, Part 2.1 Testing

Yucheng Liang: Part2.1 Coding, Part 2.1 Testing, Part 2 Extra

Jiaying Wu: Part 2.1 Testing, Part 2.2, Part 2 Report