COMP9444 Neural Networks and Deep Learning

Assignment 1

Term 2, 2024

Submitted by

zID: z5325799

Name: Lyu shaoqiu

Part 1: Japanese Character Recognition

1. Answer question 1

```
[[763.   5.   9.  12.  30.  64.   2.  64.  33.  18.]
 [  5. 672. 107.  17.  32.  23.  58.  11.  25.  50.]
 [  9.  60. 685.  26.  26.  22.  48.  37.  47.  40.]
 [  4.  35.  57. 759.  14.  56.  15.  18.  30.  12.]
 [ 58.  53.  76.  20. 629.  20.  32.  35.  22.  55.]
 [  8.  26. 121.  17.  20. 729.  27.   8.  33.  11.]
 [  5.  21. 145.  10.  25.  26. 722.  22.  10.  14.]
 [ 17.  31.  28.  10.  81.  16.  53. 625.  90.  49.]
 [ 12.  37.  91.  41.   5.  30.  45.   7. 711.  21.]
 [  8.  50.  87.   3.  54.  31.  19.  31.  40. 677.]]

Test set: Average loss: 1.0096, Accuracy: 6972/10000 (70%)
```

Final Accuracy:

Test set: Average loss: 1.0096, Accuracy: 6972/10000 (70%)

the number of independent parameters

Input size = 28 *28 = 784

Weight : 10 * 784 = 7840

Biases = 10

Total number of parameters : 10 * 784 +10 = 7850

2. Answer question 2

```
[[864.   4.   1.   5.  23.  32.   4.  38.  26.   3.]
 [  4. 821.  34.   3.  14.  14.  56.   5.  17.  32.]
 [  8.  11. 834.  44.  14.  19.  24.  12.  19.  15.]
 [  4.   9.  24. 920.   2.  12.   4.   4.   9.  12.]
 [ 39.  27.  20.   6. 819.   7.  29.  19.  20.  14.]
 [  9.  13.  70.  11.  13. 846.  20.   1.  12.   5.]
 [  3.  13.  37.  10.  13.   6. 896.  10.   2.  10.]
 [ 20.  10.  19.   4.  19.  11.  31. 826.  32.  28.]
 [  9.  25.  25.  57.   1.  12.  25.   5. 832.   9.]
 [  3.  16.  41.   7.  32.   5.  16.  18.  15. 847.]]

Test set: Average loss: 0.4921, Accuracy: 8505/10000 (85%)
```

In fc1

Input size = 28 * 28 = 784

Weight = 784 *512= 401408

Biases = 512

In fc2

Input size = 512

Weight = 512*10 = 5120

Biases = 10

Total number of parameters = 401408+512+5120+10 = 407050

3. Answer question 3

```
[[952.   2.   1.   0.  30.   1.   1.   6.   2.   5.]
 [  0. 939.   6.   0.   6.   0.  29.   4.   5.  11.]
 [  9.  11. 903.  26.   4.  14.  14.  10.   3.   6.]
 [  0.   3.  18. 958.   3.   9.   6.   0.   1.   2.]
 [ 14.   8.   4.   2. 945.   3.   8.   4.   7.   5.]
 [  4.  10.  33.   5.   5. 916.  13.   5.   3.   6.]
 [  4.   4.  17.   3.   5.   4. 958.   2.   1.   2.]
 [  1.   3.   2.   1.   1.   0.   8. 965.   4.  15.]
 [  4.  19.   8.   5.  10.   4.   2.   4. 940.   4.]
 [  7.   5.   9.   2.   3.   0.   4.   3.   4. 963.]]
```

Test set: Average loss: 0.2193, Accuracy: 9439/10000 (94%)

convolutional layer

conv1

Input channels = 1,

output channels = 16,

convolution kernel size = 5*5 =25

Weight = 16 * 25 * 1 = 400

Biases = 16

All = 400 + 16 = 416

Conv2

Input channels = 16,

output channels = 32,

convolution kernel size = 5*5 =25

Weight = 32 * 25 * 16 = 12800

Biases = 32

All = 12800 +32 = 12832

Fully connected layer

Fc1

Input size = 32 * 7 * 7 = 1568

output size = 500

Weight = 1568 * 500 = 784000

Biases = 500

All = 784000+500 = 784500

Fc2

Input size = 500
Input size =10
Weight = 500 *10 =5000
Biases = 10
All = 5000+10=5010

Total number of parameters = 416 + 12832 +784500 +5010 = 802758

4. Answer question 4

1
The relative accuracy from the three models in order is improving. The "NetLin" has the lowest accuracy, ranked 3rd, at about 70%. The second ranked model is "NetFull " with about 85%. "NetConv" has the highest relative accuracy and is ranked first with about 94%.
2
Number of independent parameters "NetLin" is 7850
Number of independent parameters "NetFull" is 407050
Number of independent parameters "NetConv" is 802758

Discussion: As the complexity of the model increases the model accuracy increases and also the time taken increases. Of the three models, netlin uses only linear features for segmentation and has the worst performance in terms of accuracy, and it has the lowest number of parameters. netfill is in the middle. Netfill is in the middle. The best model is netconv, which is the most complex and time-consuming for data processing.

3
Netlin
Confusion matrix:
```
[[763.   5.   9.  12.  30.  64.   2.  64.  33.  18.]
 [  5. 672. 107.  17.  32.  23.  58.  11.  25.  50.]
 [  9.  60. 685.  26.  26.  22.  48.  37.  47.  40.]
 [  4.  35.  57. 759.  14.  56.  15.  18.  30.  12.]
 [ 58.  53.  76.  20. 629.  20.  32.  35.  22.  55.]
 [  8.  26. 121.  17.  20. 729.  27.   8.  33.  11.]
 [  5.  21. 145.  10.  25.  26. 722.  22.  10.  14.]
 [ 17.  31.  28.  10.  81.  16.  53. 625.  90.  49.]
 [ 12.  37.  91.  41.   5.  30.  45.   7. 711.  21.]
 [  8.  50.  87.   3.  54.  31.  19.  31.  40. 677.]]
```
In this section, both "ha" and "ma" are more likely to be mistaken for "su".

Net full
Confusion matrix:

```
[[864.   4.   1.   5.  23.  32.   4.  38.  26.   3.]
 [  4. 821.  34.   3.  14.  14.  56.   5.  17.  32.]
 [  8.  11. 834.  44.  14.  19.  24.  12.  19.  15.]
 [  4.   9.  24. 920.   2.  12.   4.   4.   9.  12.]
 [ 39.  27.  20.   6. 819.   7.  29.  19.  20.  14.]
 [  9.  13.  70.  11.  13. 846.  20.   1.  12.   5.]
 [  3.  13.  37.  10.  13.   6. 896.  10.   2.  10.]
 [ 20.  10.  19.   4.  19.  11.  31. 826.  32.  28.]
 [  9.  25.  25.  57.   1.  12.  25.   5. 832.   9.]
 [  3.  16.  41.   7.  32.   5.  16.  18.  15. 847.]]
```

In this section "ki" is easily mistaken for "ma"." "ha" is easily mistaken for "su". "re" is easily mistaken for "tsu".

Netconv

Confusion matrix:

```
[[952.   2.   1.   0.  30.   1.   1.   6.   2.   5.]
 [  0. 939.   6.   0.   6.   0.  29.   4.   5.  11.]
 [  9.  11. 903.  26.   4.  14.  14.  10.   3.   6.]
 [  0.   3.  18. 958.   3.   9.   6.   0.   1.   2.]
 [ 14.   8.   4.   2. 945.   3.   8.   4.   7.   5.]
 [  4.  10.  33.   5.   5. 916.  13.   5.   3.   6.]
 [  4.   4.  17.   3.   5.   4. 958.   2.   1.   2.]
 [  1.   3.   2.   1.   1.   0.   8. 965.   4.  15.]
 [  4.  19.   8.   5.  10.   4.   2.   4. 940.   4.]
 [  7.   5.   9.   2.   3.   0.   4.   3.   4. 963.]]
```

In this section "o" can easily be mistaken for "na"." "ki" is easily mistaken for "ha". "ha" can easily be mistaken for "su".

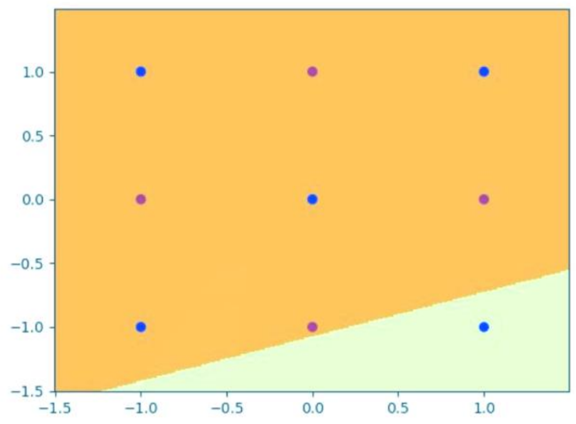
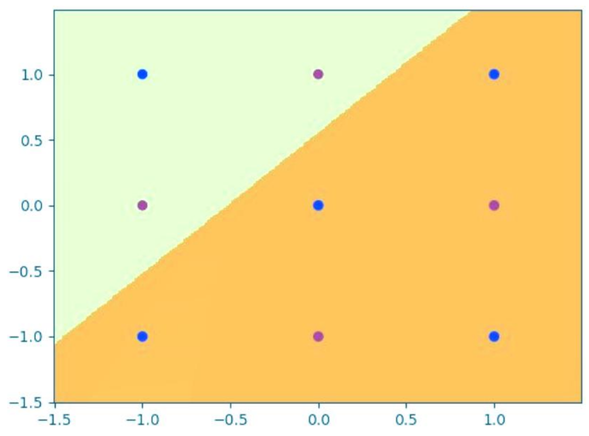
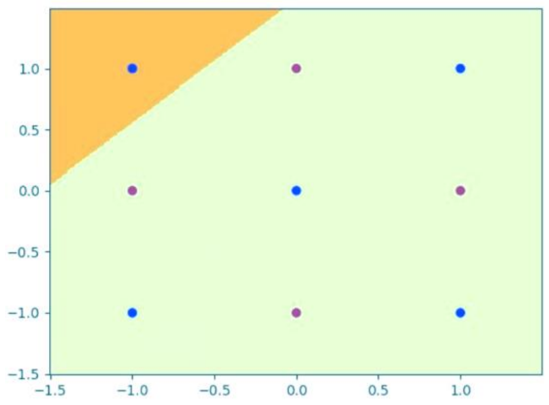
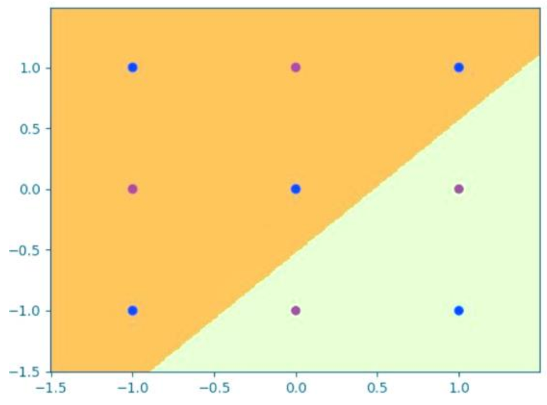
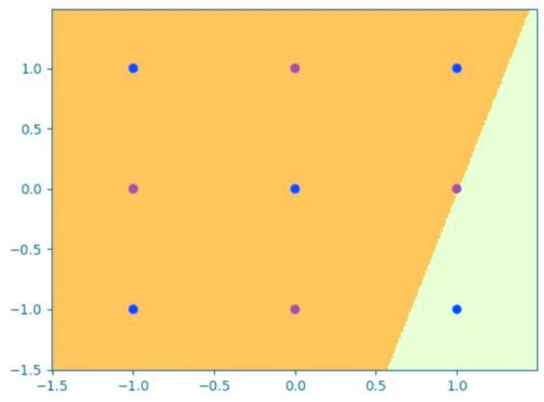
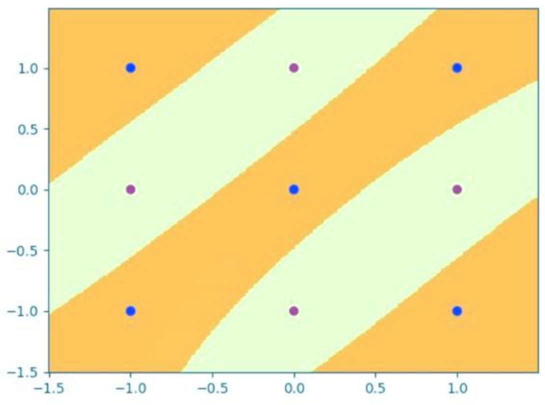
1, there are many similar features between Japanese kanji characters. Especially these characters are easy to be confused, once the parameter extraction settings are not appropriate, it is easy to have the features confused.

2, we found that some misrecognized characters, such as "su", "ma", etc., which have a lot of similar details in handwriting, appear in high frequency.

3, Japanese kanji characters show a high degree of non-linearity in visualization, so the accuracy of the linear "NetLin" is low. "NetFull" uses a fully connected layer, but it lacks the ability to deal with localized structures in complex images (e.g., complex stroke details in Japanese kanji). "NetConv", on the other hand, has a convolutional layer that learns small regions in the image, allowing it to better differentiate Japanese kanji characters that are morphologically similar.

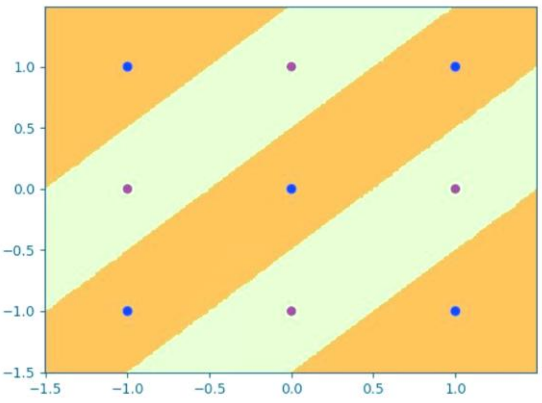
Part 2: Multi-Layer Perceptron

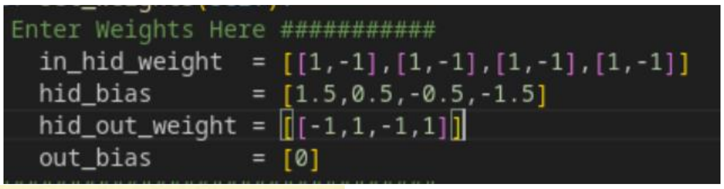
1. Answer question 1

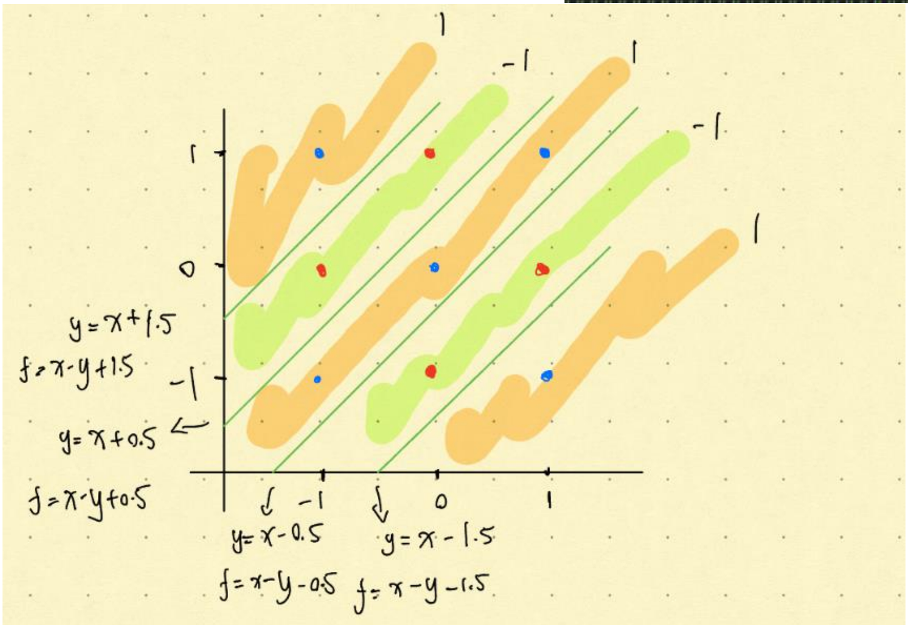
```
ep: 9800 loss: 0.1022 acc: 100.00
Final Weights:
tensor([[ 4.7294, -5.2481],
        [ 6.9408, -7.1242],
        [ 4.8425, -4.8899],
        [-4.6648,  4.3664],
        [ 0.9838,  6.9861],
        [ 5.4404,  1.3140]])
tensor([-2.6461,  3.5051, -7.7563,  7.2449, -0.6150,  0.2597])
tensor([[-5.2367,  6.2912,  6.3951, -4.9857,  5.1537, -4.9799]])
tensor([1.9824])
Final Accuracy:  100.0
```

2. Answer question 2

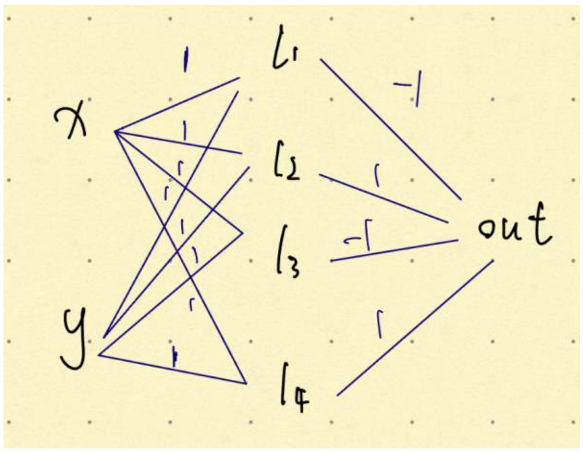




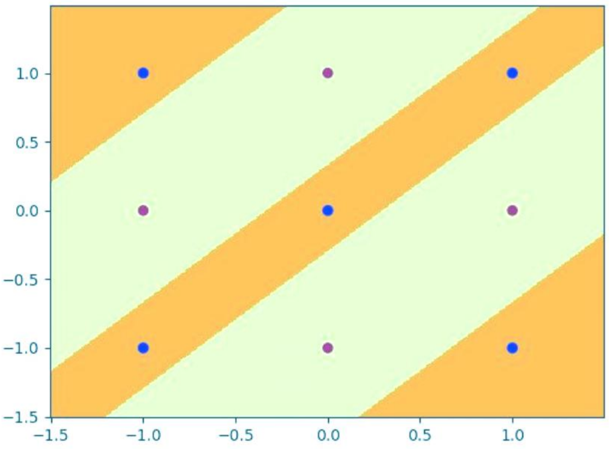
```
Enter Weights Here ###########
  in_hid_weight   = [[1,-1],[1,-1],[1,-1],[1,-1]]
  hid_bias        = [1.5,0.5,-0.5,-1.5]
  hid_out_weight = [[-1,1,-1,1]]
  out_bias        = [0]
```

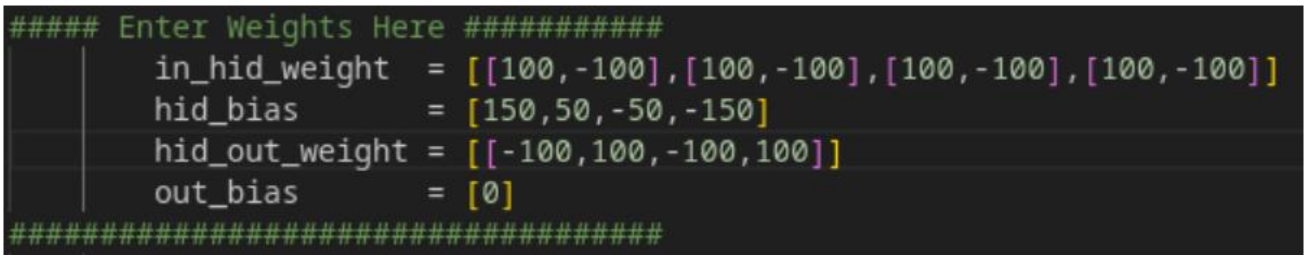


Point	$f: x-y+1.5$		$f: x-y+0.5$		$f: x-y-0.5$		$f\ x-y-1.5$	
(-1,-1)	1.5	0	0.5	1	-0.5	1	-1.5	0
(-1,0)	0.5	0	-0.5	0	-1.5	1	-2.5	0
(-1,1)	-0.5	1	-2.5	0	-2.5	1	-3.5	0
(0,-1)	2.5	0	1.5	1	0.5	0	-0.5	0
(0,0)	1.5	0	0.5	1	-0.5	1	-1.5	0
(0,1)	0.5	0	-0.5	0	-1.5	1	-2.5	0
(1,-1)	3.5	0	2.5	1	1.5	0	0.5	1
(1,0)	2.5	0	1.5	1	0.5	0	-0.5	0
(1,1)	1.5	0	0.5	1	-0.5	1	-1.5	0
hid W	-1		1		-1		1	



3. Answer question 3

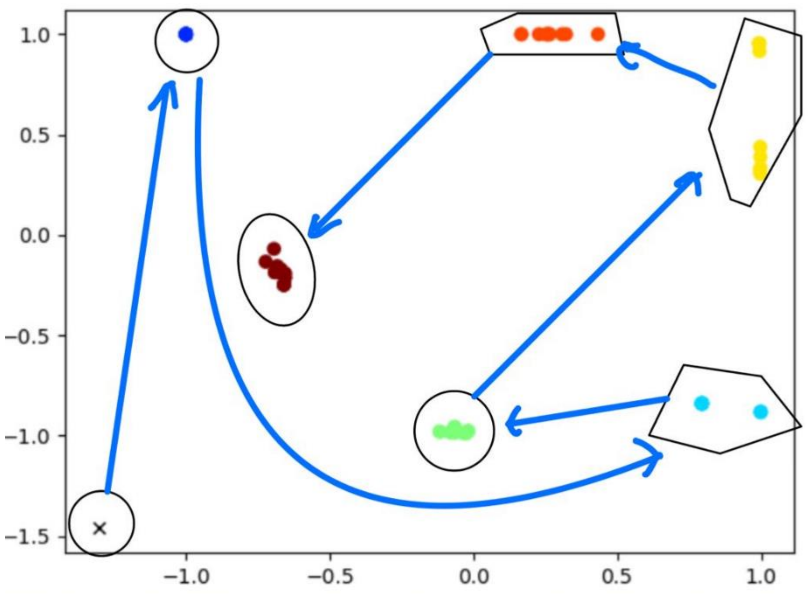




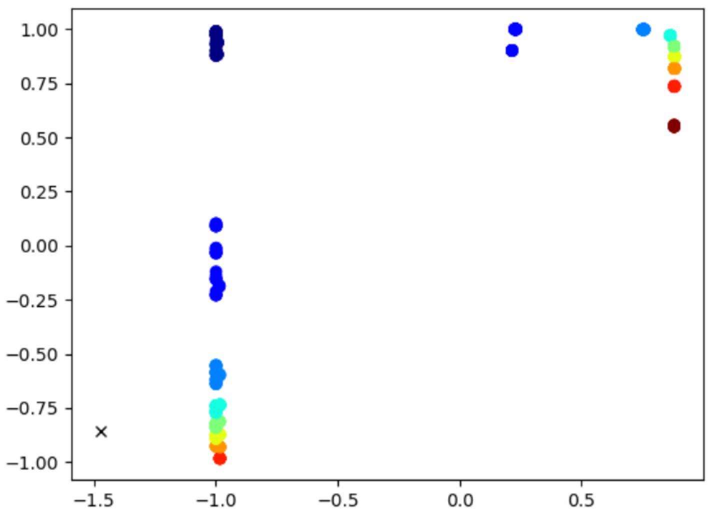
```
##### Enter Weights Here ###########
        in_hid_weight  = [[100,-100],[100,-100],[100,-100],[100,-100]]
        hid_bias       = [150,50,-50,-150]
        hid_out_weight = [[-100,100,-100,100]]
        out_bias       = [0]
####################################
```

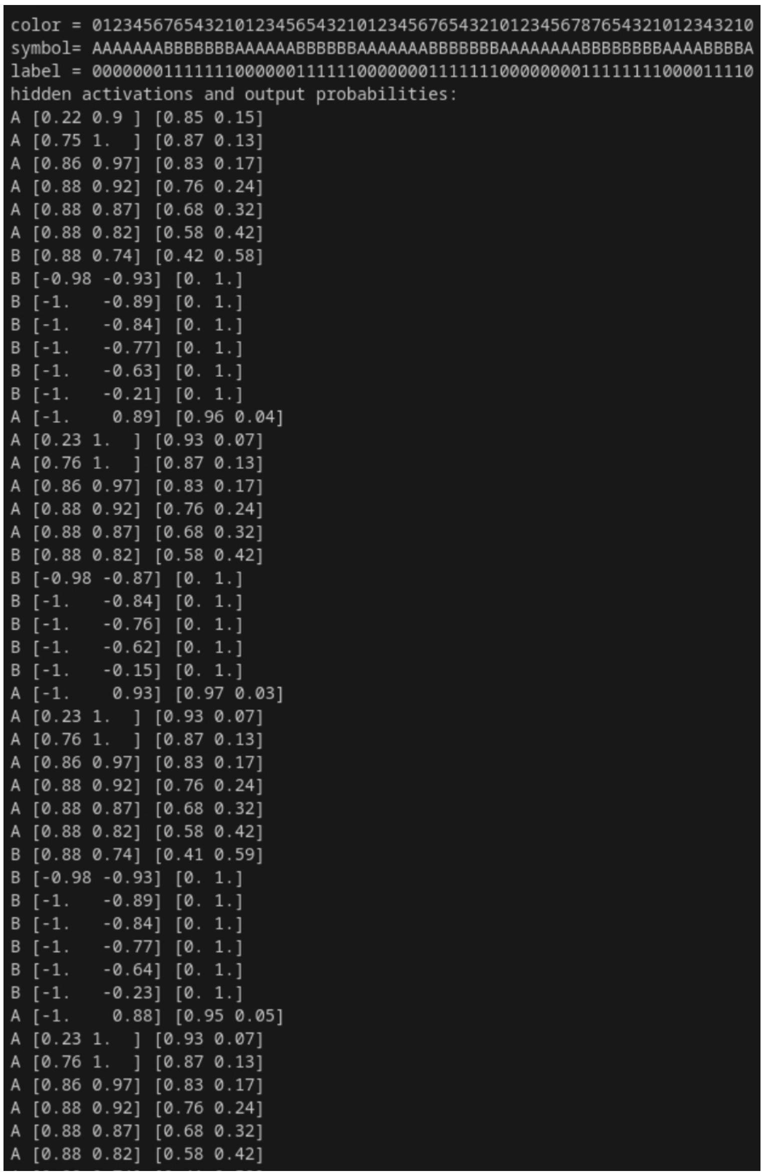
Part 3: Hidden Unit Dynamics for Recurrent Networks

1. Answer question 1



2. Answer question 2





```
color = 01234567654321012345654321012345676543210123456787654321012343210
symbol= AAAAAAABBBBBBBAAAAAABBBBBBAAAAAAAABBBBBBBAAAAAAAAABBBBBBBBAAAABBBBA
label = 0000000111111100000011111100000000111111100000000011111111000011110
hidden activations and output probabilities:
A [0.22 0.9 ] [0.85 0.15]
A [0.75 1.  ] [0.87 0.13]
A [0.86 0.97] [0.83 0.17]
A [0.88 0.92] [0.76 0.24]
A [0.88 0.87] [0.68 0.32]
A [0.88 0.82] [0.58 0.42]
B [0.88 0.74] [0.42 0.58]
B [-0.98 -0.93] [0. 1.]
B [-1.   -0.89] [0. 1.]
B [-1.   -0.84] [0. 1.]
B [-1.   -0.77] [0. 1.]
B [-1.   -0.63] [0. 1.]
B [-1.   -0.21] [0. 1.]
A [-1.    0.89] [0.96 0.04]
A [0.23 1.  ] [0.93 0.07]
A [0.76 1.  ] [0.87 0.13]
A [0.86 0.97] [0.83 0.17]
A [0.88 0.92] [0.76 0.24]
A [0.88 0.87] [0.68 0.32]
B [0.88 0.82] [0.58 0.42]
B [-0.98 -0.87] [0. 1.]
B [-1.   -0.84] [0. 1.]
B [-1.   -0.76] [0. 1.]
B [-1.   -0.62] [0. 1.]
B [-1.   -0.15] [0. 1.]
A [-1.    0.93] [0.97 0.03]
A [0.23 1.  ] [0.93 0.07]
A [0.76 1.  ] [0.87 0.13]
A [0.86 0.97] [0.83 0.17]
A [0.88 0.92] [0.76 0.24]
A [0.88 0.87] [0.68 0.32]
A [0.88 0.82] [0.58 0.42]
B [0.88 0.74] [0.41 0.59]
B [-0.98 -0.93] [0. 1.]
B [-1.   -0.89] [0. 1.]
B [-1.   -0.84] [0. 1.]
B [-1.   -0.77] [0. 1.]
B [-1.   -0.64] [0. 1.]
B [-1.   -0.23] [0. 1.]
A [-1.    0.88] [0.95 0.05]
A [0.23 1.  ] [0.93 0.07]
A [0.76 1.  ] [0.87 0.13]
A [0.86 0.97] [0.83 0.17]
A [0.88 0.92] [0.76 0.24]
A [0.88 0.87] [0.68 0.32]
A [0.88 0.82] [0.58 0.42]
```

1. All sequences follow anbn's rule, where an A of random length is followed by a B of the same length.

2, The goal of the program is to predict whether the next character is A or B at each time point.

For each character in the sequence, the SRN updates its hidden state and predicts the next character based on the current hidden state and input.

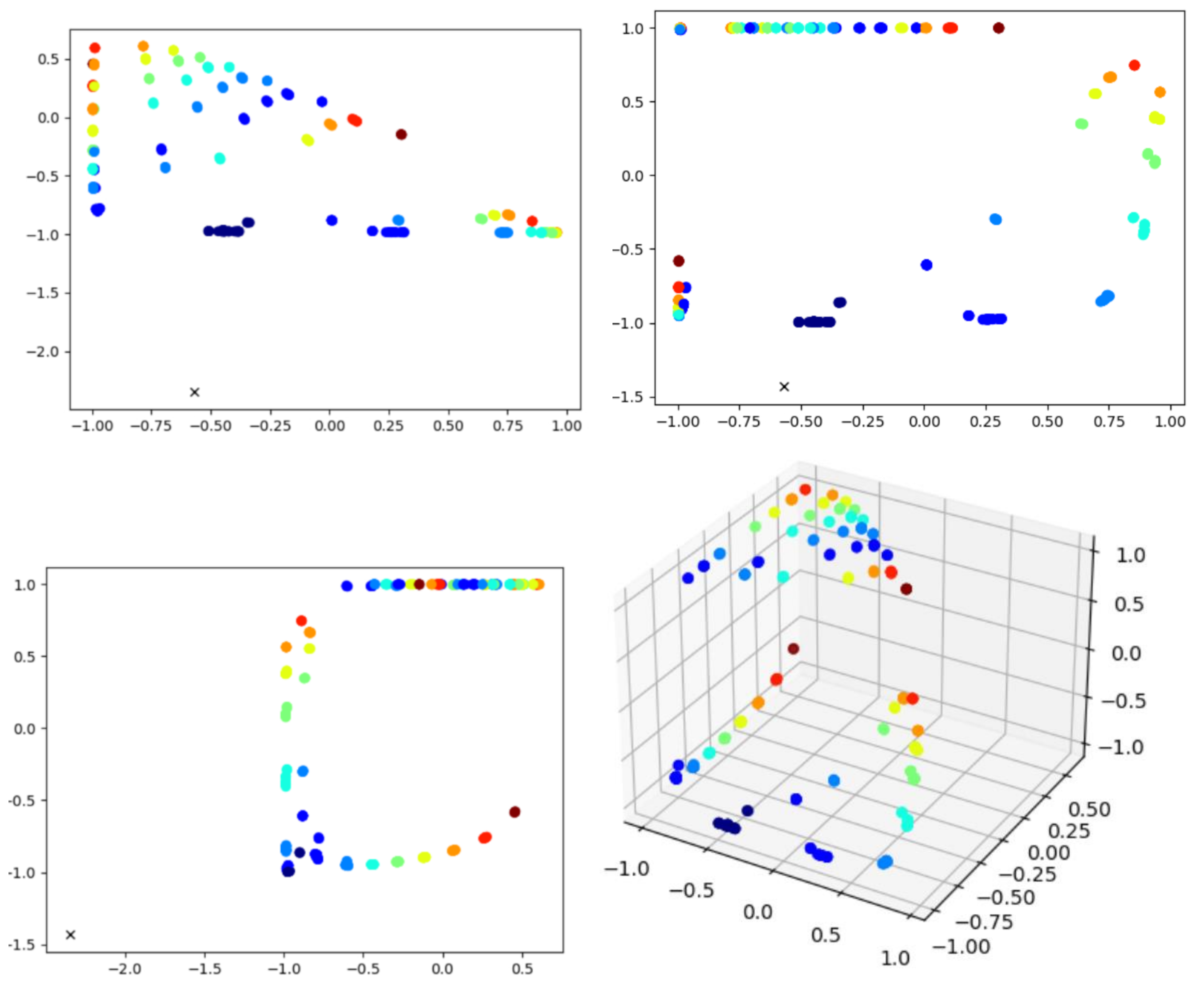
3, In training loss is calculated based on the output of the network and the actual next character while updating the network weights using the back propagation algorithm.

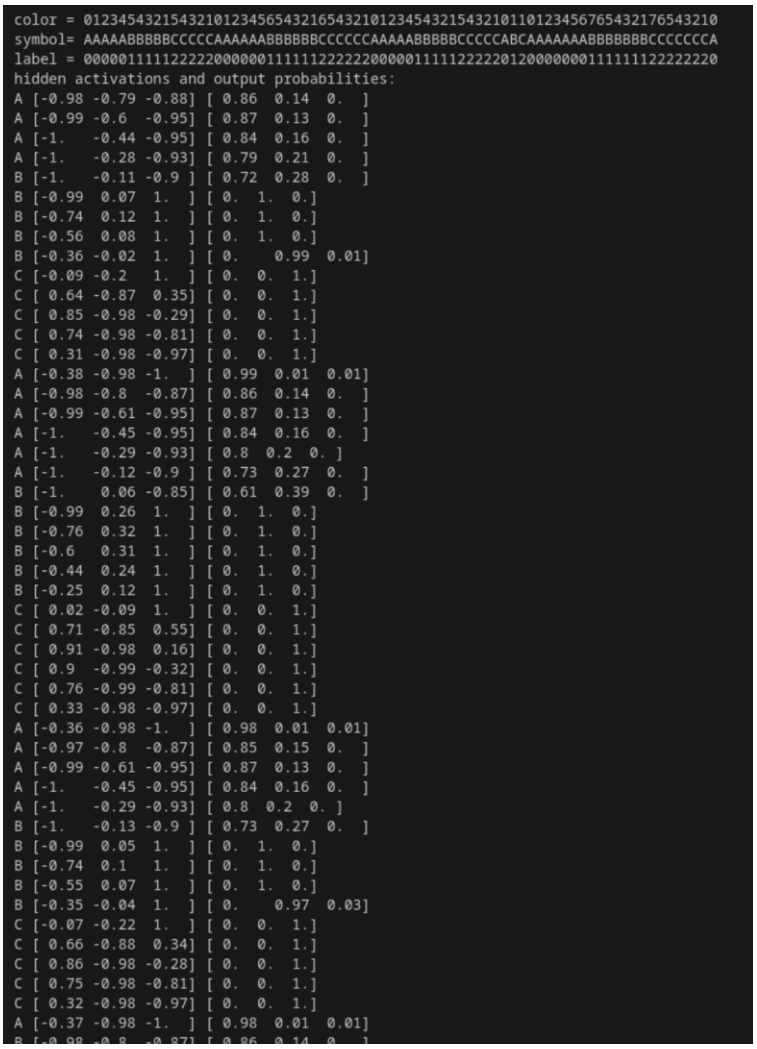
4, Because the length of the A character appears probabilistically, we can't really infer how many A's there will be, but once the first B appears, according to the generative principle, we can know how many B's there will be, and how many A's will follow the last B. The network will be able to predict the next character based on the current hidden state and input.

A successfully trained model should be able to accurately capture the relationship between the number of a's and b's after training.

Also in the output responses I find that the probability of an A keeps changing as it runs, but once the 1st B appears, the probability of a B is certain, and the probability of the 1st A is certain (both must be 1).

3. Answer question 3





```
color = 012345432154321012345654321654321012345432154321011012345676543217654321 0
symbol= AAAAABBBBBCCCCCAAAAAABBBBBBCCCCCCAAAAABBBBBCCCCCABCAAAAAAABBBBBBBCCCCCCCA
label = 000001111122222000000111111222222000001111122222012000000011111112222222 0
hidden activations and output probabilities:
A [-0.98 -0.79 -0.88] [ 0.86  0.14  0.  ]
A [-0.99 -0.6  -0.95] [ 0.87  0.13  0.  ]
A [-1.   -0.44 -0.95] [ 0.84  0.16  0.  ]
A [-1.   -0.28 -0.93] [ 0.79  0.21  0.  ]
B [-1.   -0.11 -0.9 ] [ 0.72  0.28  0.  ]
B [-0.99  0.07  1.  ] [ 0.  1.  0.]
B [-0.74  0.12  1.  ] [ 0.  1.  0.]
B [-0.56  0.08  1.  ] [ 0.  1.  0.]
B [-0.36 -0.02  1.  ] [ 0.    0.99  0.01]
C [-0.09 -0.2   1.  ] [ 0.  0.  1.]
C [ 0.64 -0.87  0.35] [ 0.  0.  1.]
C [ 0.85 -0.98 -0.29] [ 0.  0.  1.]
C [ 0.74 -0.98 -0.81] [ 0.  0.  1.]
C [ 0.31 -0.98 -0.97] [ 0.  0.  1.]
A [-0.38 -0.98 -1.  ] [ 0.99  0.01  0.01]
A [-0.98 -0.8  -0.87] [ 0.86  0.14  0.  ]
A [-0.99 -0.61 -0.95] [ 0.87  0.13  0.  ]
A [-1.   -0.45 -0.95] [ 0.84  0.16  0.  ]
A [-1.   -0.29 -0.93] [ 0.8  0.2  0. ]
A [-1.   -0.12 -0.9 ] [ 0.73  0.27  0.  ]
B [-1.    0.06 -0.85] [ 0.61  0.39  0.  ]
B [-0.99  0.26  1.  ] [ 0.  1.  0.]
B [-0.76  0.32  1.  ] [ 0.  1.  0.]
B [-0.6   0.31  1.  ] [ 0.  1.  0.]
B [-0.44  0.24  1.  ] [ 0.  1.  0.]
B [-0.25  0.12  1.  ] [ 0.  1.  0.]
C [ 0.02 -0.09  1.  ] [ 0.  0.  1.]
C [ 0.71 -0.85  0.55] [ 0.  0.  1.]
C [ 0.91 -0.98  0.16] [ 0.  0.  1.]
C [ 0.9  -0.99 -0.32] [ 0.  0.  1.]
C [ 0.76 -0.99 -0.81] [ 0.  0.  1.]
C [ 0.33 -0.98 -0.97] [ 0.  0.  1.]
A [-0.36 -0.98 -1.  ] [ 0.98  0.01  0.01]
A [-0.97 -0.8  -0.87] [ 0.85  0.15  0.  ]
A [-0.99 -0.61 -0.95] [ 0.87  0.13  0.  ]
A [-1.   -0.45 -0.95] [ 0.84  0.16  0.  ]
A [-1.   -0.29 -0.93] [ 0.8  0.2  0. ]
B [-1.   -0.13 -0.9 ] [ 0.73  0.27  0.  ]
B [-0.99  0.05  1.  ] [ 0.  1.  0.]
B [-0.74  0.1   1.  ] [ 0.  1.  0.]
B [-0.55  0.07  1.  ] [ 0.  1.  0.]
B [-0.35 -0.04  1.  ] [ 0.    0.97  0.03]
C [-0.07 -0.22  1.  ] [ 0.  0.  1.]
C [ 0.66 -0.88  0.34] [ 0.  0.  1.]
C [ 0.86 -0.98 -0.28] [ 0.  0.  1.]
C [ 0.75 -0.98 -0.81] [ 0.  0.  1.]
C [ 0.32 -0.98 -0.97] [ 0.  0.  1.]
A [-0.37 -0.98 -1.  ] [ 0.98  0.01  0.01]
B [-0.98 -0.8  -0.87] [ 0.86  0.14  0.  ]
```

Training.

1. all sequences follow anbncn's rule, where A of random length is followed by B and C of the same length.

2, The goal of the program is to predict whether the next character is A or B or C at each time point.

For each character in the sequence, the SRN updates its hidden state and predicts the next character based on the current hidden state and inputs.The three characters ABC correspond to three inputs, three hidden units and three outputs of the SRN model.

3, In training loss is calculated based on the output of the network and the actual next character while updating the network weights using the back propagation algorithm. Training is continued for up to 200,000 epochs until the network is able to reliably predict all 'C' characters and subsequent 'A' characters. Accuracy is also guaranteed.

4, For each epoch, it receives an input as well as a target output.

5, because the length of an A character is probabilistic, we can't really infer how many A's there will be, but once the first B occurs, according to the generative principle, we know how many B's there will be and how many C's and A's will surely follow the last one.

A successfully trained model should be able to accurately capture the relationship between the number of a's and b's after training.

Also in the output responses we find that the probability of A changes constantly when it is run, but once the first B is present, the probabilities of B and C are certain, as well as the probability of the first A (both must be 1).

The presentation on the image is from a point will gradually diverge to three face directions.

4. Answer question 4

LSTM network is a is a special kind of recurrent neural network that incorporates an oblivious design.

Four main parameter matrices are defined at the time of initialization, which are input layer, between hidden layers, hidden layer bias, and output layer.

After setting up the initialization __init__, init_weights, init_hidden

Then use the forword method. In this process each time step 't' of the input data will be checked. The model will use matrix operations to calculate the values of the 4 special gates by combining the state of the current and previous inputs. (i_t,f_t,g_t,o_t).The input gate determines which values are to be updated.The forget gate determines which values are to be rounded.The new values are calculated.The output gate controls the output. At the same time these gates are constantly updating the hidden state and finally the final result is calculated by matrix weights and bias values.

I think that in this way some special values in the training cases can be filtered out. For example a stretch of identical characters that is too long. The forgetting pattern of the model can, help us to better judge some long strings for the task. So that the model will not be in the middle of the specific value to bring the direction of his training.