COMP9444 Project1

Part1

1:

The screen shot for the confusion matrix and final accuracy:

```
 [ 61.  54.  80.  18. 622.  21.  33.  36.  19.  56.]
 [  8.  28. 125.  17.  20. 727.  26.   7.  34.   8.]
 [  5.  22. 147.  11.  27.  25. 719.  19.  10.  15.]
 [ 16.  28.  28.   9.  83.  18.  57. 621.  91.  49.]
 [ 10.  38.  94.  42.   7.  31.  46.   8. 702.  22.]
 [  7.  53.  88.   3.  51.  31.  20.  34.  38. 675.]]

Test set: Average loss: 1.0083, Accuracy: 6951/10000 (70%)
```

2:

I choose the number of hidden nodes to be 256.And it achieves the 85% final accuracy.

```
 [  7.  11. 843.  46.  13.  16.  26.  11.  13.  14.]
 [  3.  14.  26. 915.   2.  18.   7.   1.   6.   8.]
 [ 34.  32.  21.   4. 822.   7.  28.  19.  21.  12.]
 [ 10.  14.  76.   8.  12. 839.  18.   2.  14.   7.]
 [  2.  17.  45.   8.  17.   4. 891.   7.   1.   8.]
 [ 23.  13.  18.   3.  20.  10.  30. 828.  22.  33.]
 [  8.  27.  26.  50.   5.   8.  29.   3. 840.   4.]
 [  2.  19.  47.   4.  28.   5.  26.  11.  10. 848.]]

Test set: Average loss: 0.4945, Accuracy: 8492/10000 (85%)
```

3:

I have defined two convolutional layers (conv1 and conv2), one max

pooling layer (max_pool), and two fully connected layers (fc_layer_1 and fc_layer_2). The total number of independent parameters in NetConv network is 228831, which is calculated as 1664 + 38424 + 187143 + 1600.

```
[[968.   2.   2.   0.  16.   1.   0.   5.   4.   2.]
 [  2. 932.   5.   0.   4.   0.  36.   6.   5.  10.]
 [ 11.   6. 877.  42.  10.  10.  19.   7.   8.  10.]
 [  1.   0.  15. 960.   1.   3.  10.   2.   3.   5.]
 [ 16.   6.   2.   6. 937.   3.  16.   6.   5.   3.]
 [  6.  10.  48.   6.   3. 894.  19.   2.   4.   8.]
 [  4.   2.  14.   2.   2.   2. 969.   3.   1.   1.]
 [  8.   5.   4.   0.   4.   0.   6. 954.   4.  15.]
 [  5.  12.   6.   1.   6.   2.   4.   0. 958.   6.]
 [  6.   7.   2.   2.   7.   0.   4.   4.   4. 964.]]

Test set: Average loss: 0.2312, Accuracy: 9413/10000 (94%)
```

4:

a:

The accuracy rate for NetLin after 10 training epochs is around 70%.

The accuracy rate for NetFull after 10 training epochs is around 85%.

The accuracy rate for NetConv after 10 training epochs is around 94%.

Among those three models,the NetConv model achieves the highest accuracy rate on test set(94%),followed by NetFull model,which can reach 85% overall accuracy rate.The NetLin model has the lowest accuracy rate on test set,only 70%.

b:

==========

lin_layer.weight 7840

lin_layer.bias 10

The total number of parameters in the NetLin is 7850

==========

fc_layer_1.weight 200704

fc_layer_1.bias 256

fc_layer_2.weight 2560

fc_layer_2.bias 10

The total number of parameters in the NetFull is 203530

==========

conv1.weight 1600

conv1.bias 64

conv2.weight 38400

conv2.bias 24

fc_layer_1.weight 186984

fc_layer_1.bias 159

fc_layer_2.weight 1590

fc_layer_2.bias 10

The total number of parameters in the NetConv is 228831

c:

NetLin:

Looking at confusion matrix for question1,I find ma is most likely to be mistaken for su,because the biggest number except numbers on the

diagonal in the confusion matrix is 147,which located at the seventh row and the third column.This means there are 147 cases that seventh row character(ma) is mistaken for the third row character(su).

NetFull:

Ha is most likely to be mistaken for su.

Looking at confusion matrix of q2,the biggest number except diagonal is 78,which is at the sixth row and the third column.

NetConv:

Ha is most likely to be mistaken for su.

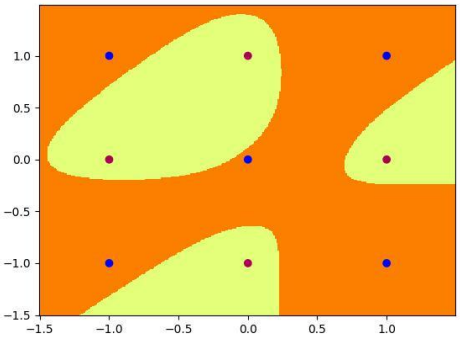
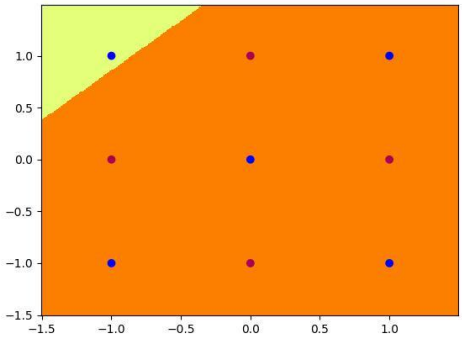
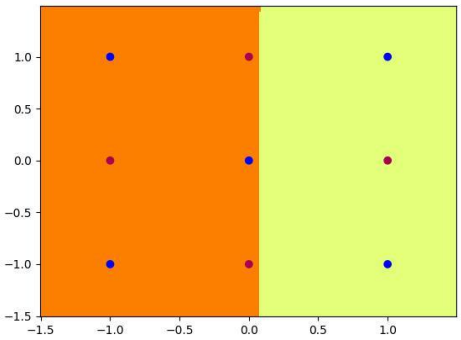
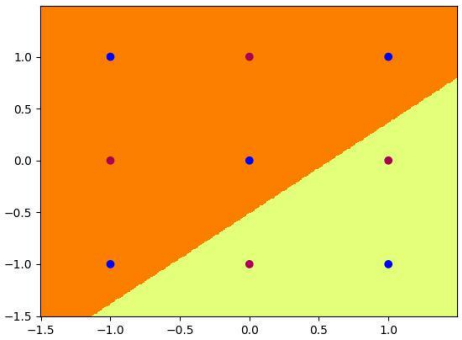
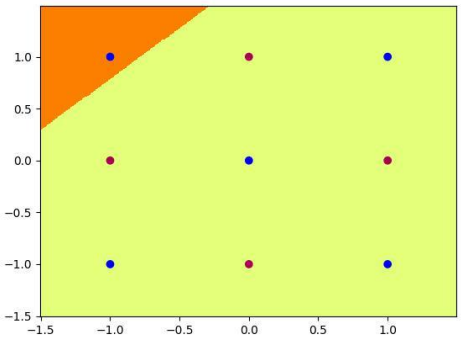
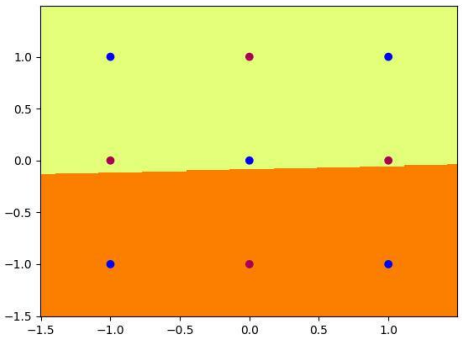
Looking at confusion matrix for q3, the biggest number except diagonal is 34,also located at the sixth row and the third column.

Part2

1:

I have chosen 5 hidden nodes for training.

```
tensor([[ 5.8292,  0.7010],
        [-0.2689, -4.7753],
        [ 5.2606, -4.8564],
        [-4.2140, -4.7878],
        [ 5.7875,  4.6056]])
tensor([-4.4478, -3.0796, -2.9529, -1.9176, -2.5498])
tensor([[ 7.5477,  6.1119, -6.5467, -5.9081, -6.5920]])
tensor([3.2299])
Final Accuracy:  100.0
```

2:

The equation of the boundary determined by each hidden node:

N1: *Heaviside(-0.5x+0.5y+0.8)*

N2: *Heaviside(-0.7x+0.7y+0.3)*

N3: *Heaviside(-0.7x+0.7y-0.3)*

N4: *Heaviside(-0.5x+0.5y-0.8)*

Output node:

Heaviside(-1.5N_1+1.5N_2-1.5N_3+1.5N_4+0.7)

X	***Y***	***T***	***N1***	***N2***	***N3***	***N4***
-1	-1	1	0.8	0.3	-0.3	-0.8
-1	0	0	1.3	1	0.4	-0.3
-1	1	1	1.8	1.7	1.1	0.2
0	-1	0	0.3	-0.4	-1	-1.3
0	0	1	0.8	0.3	-0.3	-0.8
0	1	0	1.3	1	0.4	-0.3
1	-1	1	-0.2	-1.1	-1.7	-1.8
1	0	0	0.3	-0.4	-1	-1.3
1	1	1	0.8	0.3	-0.3	-0.8

Heaviside:

X	***Y***	***T***	***N1***	***N2***	***N3***	***N4***	***Out_node***
-1	-1	1	1	1	0	0	0.7=>1
-1	0	0	1	1	1	0	-0.8=>0
-1	1	1	1	1	1	1	0.7=>1

0	-1	0	1	0	0	0	-0.8=>0
0	0	1	1	1	0	0	0.7=>1
0	1	0	1	1	1	0	-0.8=>0
1	-1	1	0	0	0	0	0.7=>1
1	0	0	1	0	0	0	-0.8=>0
1	1	1	1	1	0	0	0.7=>1

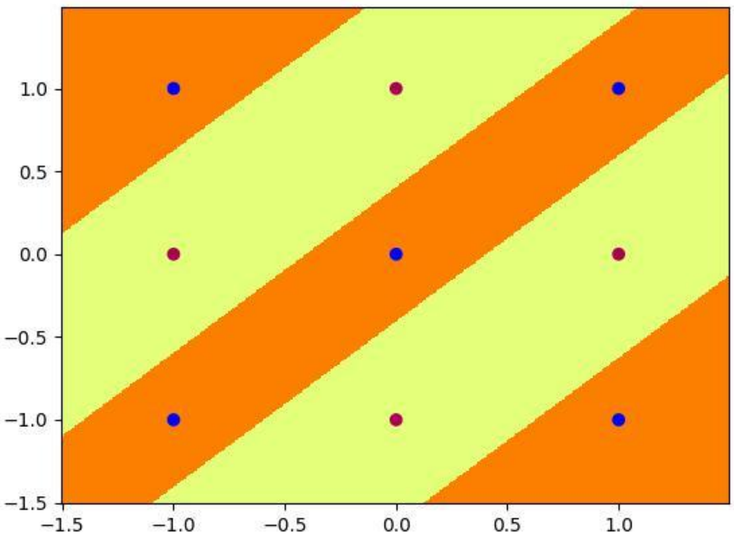
Set_weights:

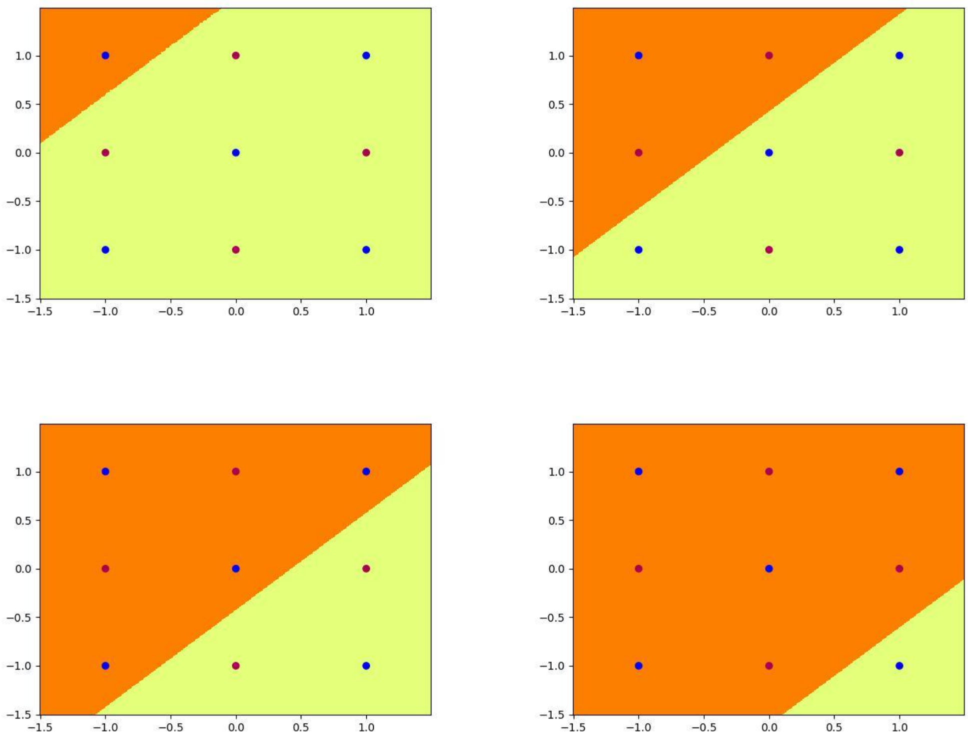
```
Initial Weights:
tensor([[-0.5000,  0.5000],
        [-0.7000,  0.7000],
        [-0.7000,  0.7000],
        [-0.5000,  0.5000]])
tensor([ 0.8000,  0.3000, -0.3000, -0.8000])
tensor([[-1.5000,  1.5000, -1.5000,  1.5000]])
tensor([0.7000])
Initial Accuracy:  100.0
```

3:

Multiply all parameters by ten

```
Initial Weights:
tensor([[-5.,  5.],
        [-7.,  7.],
        [-7.,  7.],
        [-5.,  5.]])
tensor([ 8.,  3., -3., -8.])
tensor([[-15.,  15., -15.,  15.]])
tensor([7.])
Initial Accuracy:  100.0
```

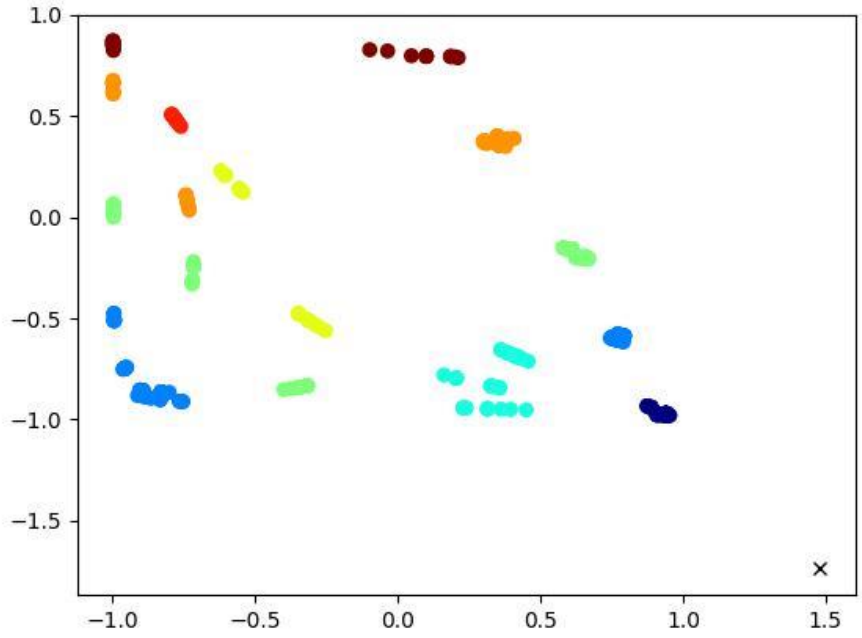

1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5 −1.0 −0.5 0.0 0.5 1.0


1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5 −1.0 −0.5 0.0 0.5 1.0

Part3

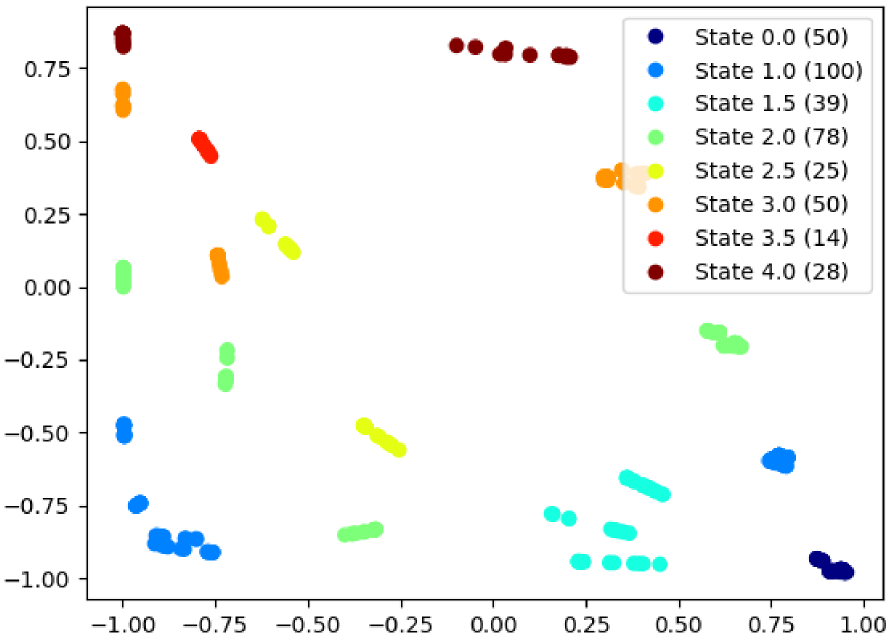
1:

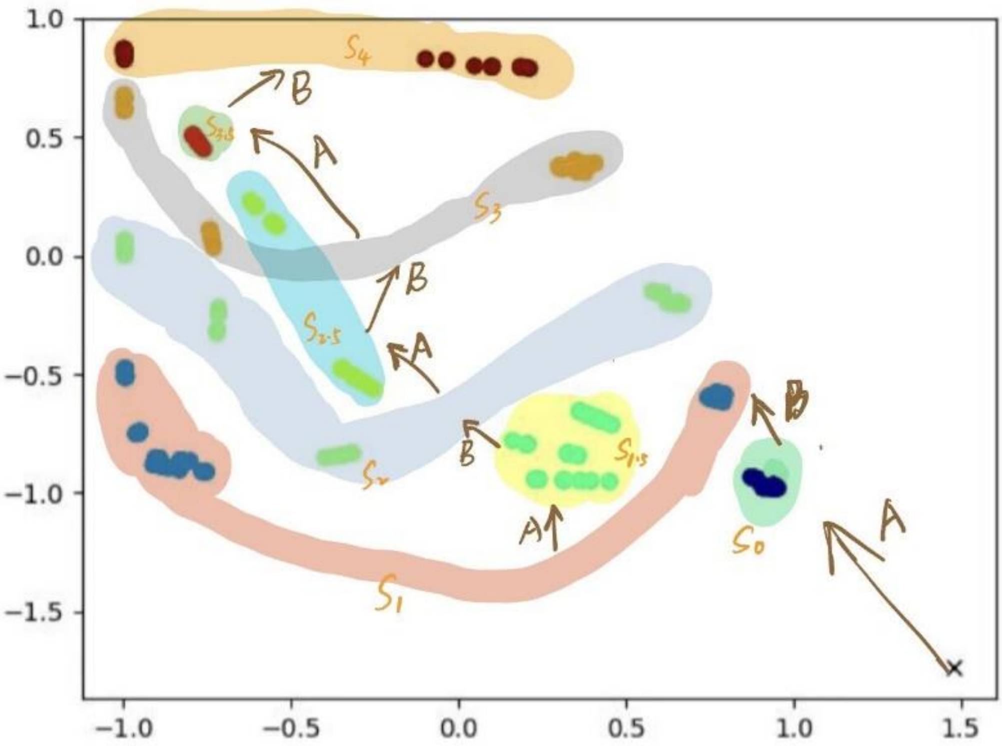
The anb2n_srn2_01.jpg:



2:

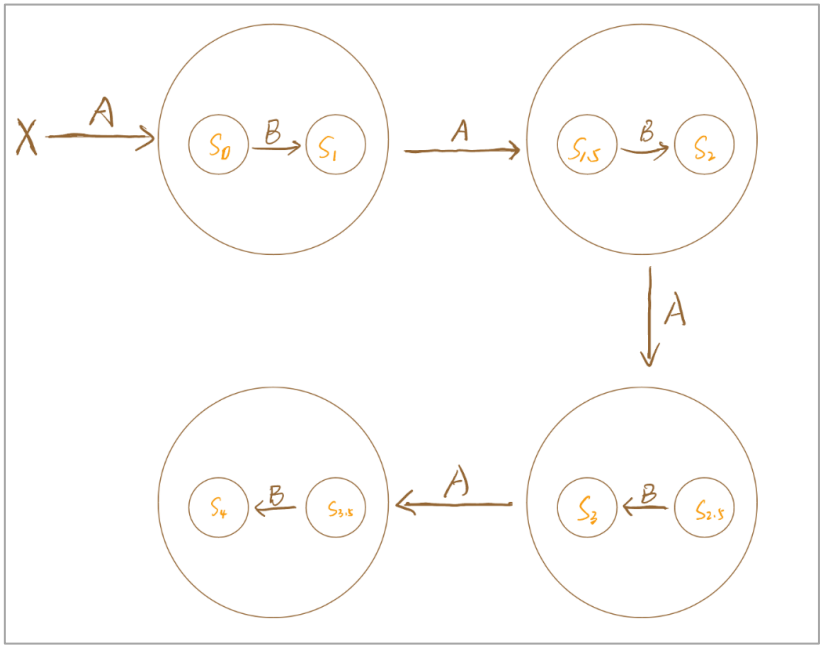
I attempted to print the quantity of states and the number of points for each state. There are eight states, and there is a clear twofold relationship between their quantities.


State 0.0 (50)
State 1.0 (100)
State 1.5 (39)
State 2.0 (78)
State 2.5 (25)
State 3.0 (50)
State 3.5 (14)
State 4.0 (28)



3：

finite state machine



4：

the network leverages distinct hidden activation states and state transitions to track the input pattern, enabling it to follow the $a^n b^{2n}$ grammar and make accurate predictions. The evolution of hidden states shows how recurrent networks can capture temporal dependencies.

- The hidden units form two main activation clusters - state A and state B.
- When processing a sequence of A's, the hidden activations stay in state A.
- When the input switches from A to B, the hidden activations rapidly transition from state A to state B. This allows the network to detect the

first B.

- In state B, the hidden activations iteratively cycle through substates that correspond to the number of B's seen so far. This allows the network to keep track of the count.
- By staying in these substates, the network predicts continuing B's until the count is fulfilled. This allows it to predict all B's after the first B.
- After the full count of B's, the hidden activations quickly switch back to state A. This enables the network to detect the end of the B sequence.
- In state A, the network predicts A as the next symbol. This allows it to predict the initial A after the last B.

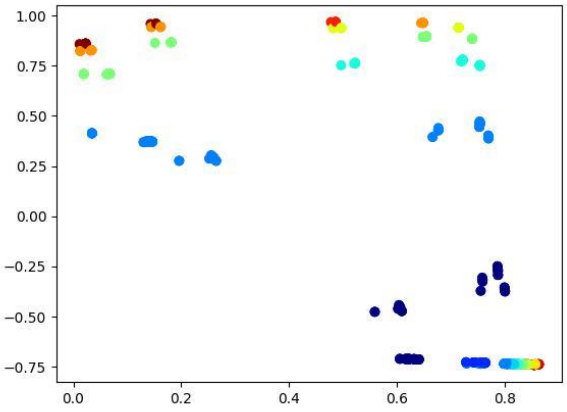
5:

After training for 100000 epochs, the loss decreased to 0.0087.

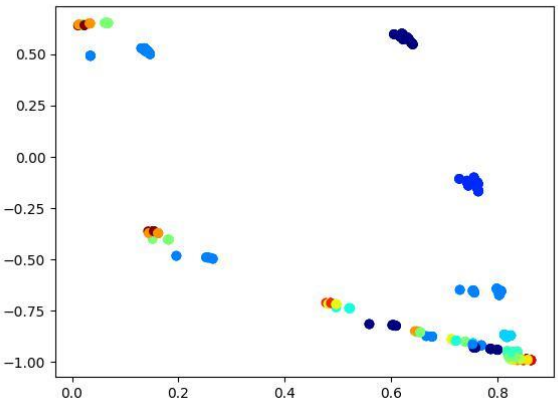
```
C [ 0.64 -0.71  0.55] [ 0.95  0.    0.05]
A [ 0.15  0.37  0.5 ] [ 0.73  0.27  0.  ]
B [ 0.26  0.28 -0.5 ] [ 0.01  0.95  0.05]
B [ 0.61 -0.47 -0.82] [ 0.    0.01  0.99]
C [ 0.76 -0.73 -0.66] [ 0.  0.  1.]
C [ 0.74 -0.73 -0.14] [ 0.05  0.    0.95]
C [ 0.62 -0.71  0.57] [ 0.96  0.    0.04]
epoch: 100000
 loss: 0.0087
```

I have chosen the default setting of 100,000 epochs, plot the hidden unit

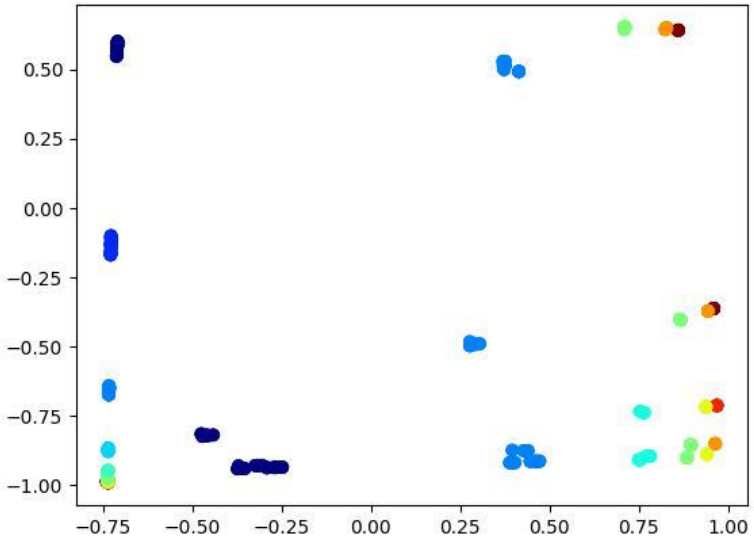
activations at epoch 100000.



anb2nc3n_lstm3_01.jpg



anb2nc3n_lstm3_02.jpg



anb2nc3n_lstm3_12.jpg

6:

LSTM model is a special type of recurrent neural network (RNN) that handles information in a unique way, allowing it to maintain high performance when dealing with long sequence tasks. This is particularly important in tasks like anb2nc3n, where the model needs to remember the early inputs (i.e., the number of A's) in order to generate the correct number

of subsequent characters (B's and C's).

In LSTM, information flows through a path called the "cell state," which serves as the model's long-term memory. The model can discard old information in the cell state using a structure called the "forget gate," add new information to the cell state using the "input gate," and determine which information should be output to the next state's hidden units and the final network output using the "output gate."

In the anb2nc3n task, LSTM can utilize the cell state to remember the number of A's seen so far. When the model starts reading B's, it knows how many B's to generate because this information is stored in the cell state. Similarly, when the model starts reading C's, it can determine the number of C's to generate based on the stored number of A's in the cell state.

The key to LSTM's success in completing the anb2nc3n task lies in its design to handle long-term dependencies in sequential data. The following are the specific mechanisms and steps:

- Long-term memory: LSTM remembers long-term dependencies in the input through its cell state. In the anb2nc3n task, this allows the network to remember the number of "A's" it has seen and generate the correct number of "B's" and "C's."

- Forget gate and input gate: When the symbols in the input sequence change, such as from "A" to "B," the forget gate is activated to discard

the stored number of "A's" in the cell state, while the input gate starts recording the number of "B's."

- Output gate: The output gate can output information from the cell state at the right time. It prevents information leakage when it is not needed and releases it when necessary, such as when transitioning to the input of "B" or "C."
- Gate mechanism: This LSTM structure provides finer control over the network state. It enables LSTM to learn counting and sequential dependencies in input patterns, allowing it to generate the correct number of "B's" and "C's" based on the number of "A's" seen.
- State transition: When a character sequence ends, such as "B," LSTM resets its cell state and starts recording a new character sequence, such as "C."
- Flexible counting and boundary recognition: By combining the above mechanisms, LSTM can flexibly count different lengths of input sequences and recognize boundaries between different symbols, enabling it to generate the correct anb2nc3n sequence.
- Complex sequential dependencies: Compared to simple RNNs, LSTM's stronger state control and memory capacity, thanks to the cell state and gate mechanism, allow it to handle more complex sequential dependency problems, such as nested counting problems present in the anb2nc3n task.