COMP9444 Assignment

Part 1: Japanese Character Recognition

Q1 (Model NetLin):

```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.817107
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.638587
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.596563
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.598331
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.322625
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.516231
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.663297
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.609507
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.348288
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.678141
[[772.   5.   8.  10.  30.  60.   2.  64.  29.  20.]
 [  6. 668. 107.  19.  31.  23.  57.  13.  25.  51.]
 [  7.  62. 691.  28.  25.  19.  48.  38.  44.  38.]
 [  5.  35.  62. 756.  15.  57.  14.  18.  27.  11.]
 [ 60.  54.  80.  21. 620.  21.  33.  35.  19.  57.]
 [  8.  30. 126.  17.  19. 724.  27.   8.  33.   8.]
 [  5.  21. 149.  10.  27.  25. 721.  20.   8.  14.]
 [ 17.  28.  29.  10.  87.  19.  54. 619.  86.  51.]
 [ 12.  34.  97.  43.   5.  29.  45.   6. 707.  22.]
 [  8.  51.  90.   3.  51.  31.  19.  30.  40. 677.]]

Test set: Average loss: 1.0100, Accuracy: 6955/10000 (70%)

Model parameters: 7850
```

Q2 (Model NetFull):

In this case, we can see the total number of independent parameters is 101770

```
Test set: Average loss: 0.5345, Accuracy: 8377/10000 (84%)

Train Epoch: 10 [0/60000 (0%)]  Loss: 0.346262
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.284999
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.240954
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.210883
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.130341
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.275975
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.231662
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.352064
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.125497
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.258143
<class 'numpy.ndarray'>
[[851.   3.   1.   4.  25.  29.   4.  40.  36.   7.]
 [  6. 818.  38.   5.  15.   7.  58.   4.  19.  30.]
 [  7.  17. 830.  40.  11.  15.  24.  14.  23.  19.]
 [  3.   8.  33. 920.   2.  12.   3.   3.   7.   9.]
 [ 43.  26.  20.   4. 804.   8.  29.  17.  27.  22.]
 [  8.  12.  81.  11.  11. 836.  14.   1.  18.   8.]
 [  3.  13.  76.  12.  14.   5. 861.   8.   2.   6.]
 [ 13.  15.  22.   5.  26.  11.  32. 823.  25.  28.]
 [ 12.  28.  24.  49.   3.   9.  27.   4. 841.   3.]
 [  2.  20.  51.   8.  26.   4.  15.  19.  12. 843.]]

Test set: Average loss: 0.5156, Accuracy: 8427/10000 (84%)

Model parameters: 101770
```

Q3 (Model NetConv):

In this case, we can see the total number of independent parameters is 225034

```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.112775
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.015323
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.177137
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.089446
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.038564
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.058035
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.024076
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.140065
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.013225
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.038735
<class 'numpy.ndarray'>
[[945.   5.   1.   2.  28.   1.   2.  10.   5.   1.]
 [  2. 914.   6.   0.   9.   1.  45.   3.   5.  15.]
 [ 11.   5. 883.  36.   9.   8.  30.   6.   7.   5.]
 [  2.   1.   8. 963.   1.   5.  14.   2.   1.   3.]
 [ 16.   5.   4.   8. 927.   5.  15.   3.   7.  10.]
 [  4.  10.  34.  11.   2. 892.  31.   4.   4.   8.]
 [  3.   4.  12.   1.   2.   0. 973.   0.   1.   4.]
 [  7.  10.   7.   1.   3.   2.  23. 931.   5.  11.]
 [  4.  16.  14.   5.   9.   2.   9.   4. 931.   6.]
 [  5.  10.   5.   3.   8.   0.  16.   2.   9. 942.]]

Test set: Average loss: 0.2612, Accuracy: 9301/10000 (93%)

Model parameters: 225034
```

Q4 （Briefly discuss the following points）:

a) the relative accuracy of the three models:

From the three model's result screenshot, we can see the Linear model's (NetLin) accuracy in the test set is 70 percent. The fully connected 2-layer network model's (NetFull) accuracy in the test set is 84 percent, an increase of 14 precentage points compared to NetLin. This improvement from the 2 hidden layers and with additional non-linear activation function (e.g. tanh), can let the model learn more complex features combinations. The convolutional network model's (NetConv) accuracy in the test set is 93 percent, an increase of 9 precentage points compared to NetFull. This improvement is because the convolutional layers can efficiently extract local image features. Focusing on these structured patterns, the model avoids learning irrelevant or noisy pixel information.

b) the number of independent parameters in each of the three models:

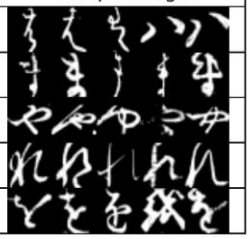
In the Linear network model (NetLin), the independent parameter is 7850.

In the fully connected 2-layer network model (NetFull), the independent parameter is 101770.

In the convolutional network model (NetConv), the independent parameter is 225034.

c) the confusion matrix of each model:

Hiragana	Unicode	Samples	Sample Images
お (o)	U+304A	7000	
き (ki)	U+304D	7000	
す (su)	U+3059	7000	
つ (tsu)	U+3064	7000	
な (na)	U+306A	7000	

Hiragana	Unicode	Samples	Sample Images
は (ha)	U+306F	7000	
ま (ma)	U+307E	7000	
や (ya)	U+3084	7000	
れ (re)	U+308C	7000	
を (wo)	U+3092	7000	

NetLin confusion matrix:

```
[[764.   6.   8.  14.  31.  62.   2.  63.  31.  19.]
 [  7. 664. 107.  20.  30.  24.  59.  14.  26.  49.]
 [  7.  62. 691.  27.  26.  21.  47.  36.  45.  38.]
 [  4.  38.  58. 760.  15.  55.  16.  18.  25.  11.]
 [ 58.  52.  76.  21. 629.  20.  32.  36.  20.  56.]
 [  8.  27. 123.  17.  20. 726.  27.   8.  33.  11.]
 [  5.  24. 148.  10.  26.  23. 723.  20.   8.  13.]
 [ 17.  28.  26.  12.  86.  15.  53. 626.  89.  48.]
 [ 11.  37.  93.  42.   8.  30.  45.   6. 706.  22.]
 [  8.  52.  87.   3.  52.  31.  20.  28.  40. 679.]]
```

NetFull confusion matrix:

```
[[851.   3.   1.   4.  25.  29.   4.  40.  36.   7.]
 [  6. 818.  38.   5.  15.   7.  58.   4.  19.  30.]
 [  7.  17. 830.  40.  11.  15.  24.  14.  23.  19.]
 [  3.   8.  33. 920.   2.  12.   3.   3.   7.   9.]
 [ 43.  26.  20.   4. 804.   8.  29.  17.  27.  22.]
 [  8.  12.  81.  11.  11. 836.  14.   1.  18.   8.]
 [  3.  13.  76.  12.  14.   5. 861.   8.   2.   6.]
 [ 13.  15.  22.   5.  26.  11.  32. 823.  25.  28.]
 [ 12.  28.  24.  49.   3.   9.  27.   4. 841.   3.]
 [  2.  20.  51.   8.  26.   4.  15.  19.  12. 843.]]
```

NetConv confusion matrix:

```
[[945.   5.   1.   2.  28.   1.   2.  10.   5.   1.]
 [  2. 914.   6.   0.   9.   1.  45.   3.   5.  15.]
 [ 11.   5. 883.  36.   9.   8.  30.   6.   7.   5.]
 [  2.   1.   8. 963.   1.   5.  14.   2.   1.   3.]
 [ 16.   5.   4.   8. 927.   5.  15.   3.   7.  10.]
 [  4.  10.  34.  11.   2. 892.  31.   4.   4.   8.]
 [  3.   4.  12.   1.   2.   0. 973.   0.   1.   4.]
 [  7.  10.   7.   1.   3.   2.  23. 931.   5.  11.]
 [  4.  16.  14.   5.   9.   2.   9.   4. 931.   6.]
 [  5.  10.   5.   3.   8.   0.  16.   2.   9. 942.]]
```

In this case, we can see in NetLin confusion matrix. The most likely to be mistaken is (***su***) and (**ma**), which the target character is (***ma***) but the model is mistake collect 148 (***ma***) to (***su***) characters. But this mistake is because of the model's performance. This is because we can see above that has three different model's confusion matrixes, the blue underline marked that although this mistake highly appeared in NetLin model's confusion matrix. When the model performance increases, we can see in NetFull model's and NetConv model's confusion matrix. This mistake is rapidly decreased (148->76->12).

However, we can see that in NetConv confusion matrix, the red underline marked the largest mistaken character in this confusion matrix which mistake character (***ki***) to character (***ma***). In this case, we can check that all those three models' confusion matrix has a similar number of mistaken characters. (58, 59, 45). This is not the model's fault, because those two handwritten characters are very similar.

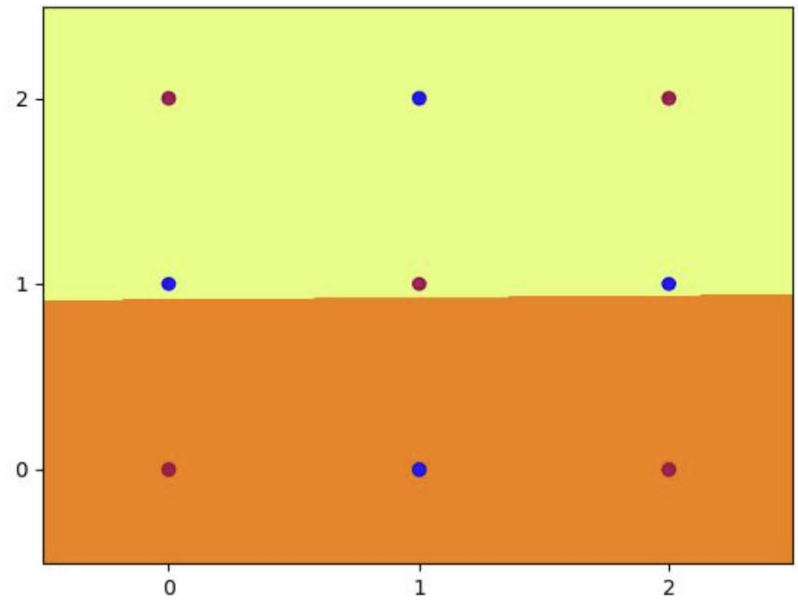
Ki:

Ma:

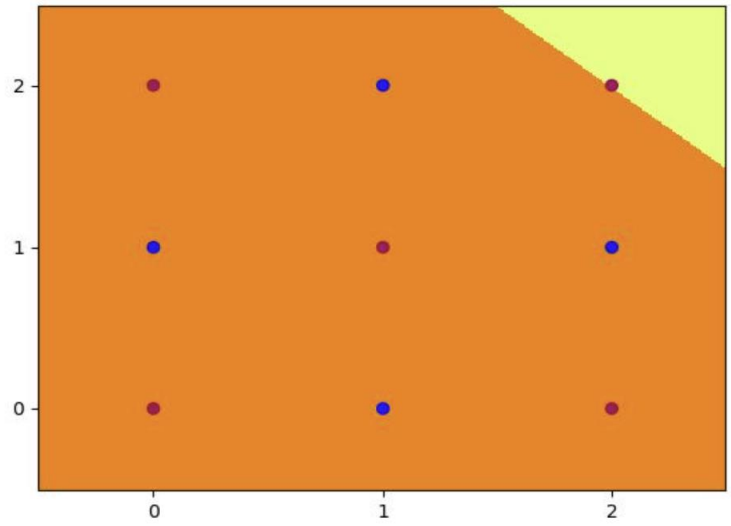
Part 2: Multi-Layer Perceptron

Q1:

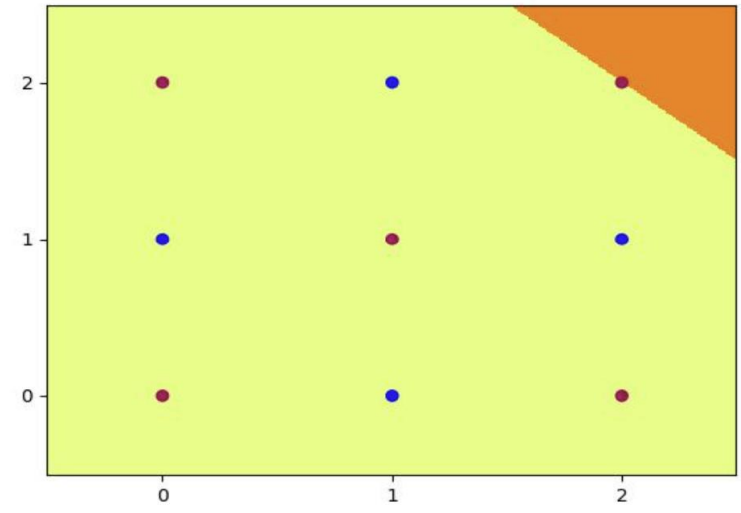
Hid_5_0:



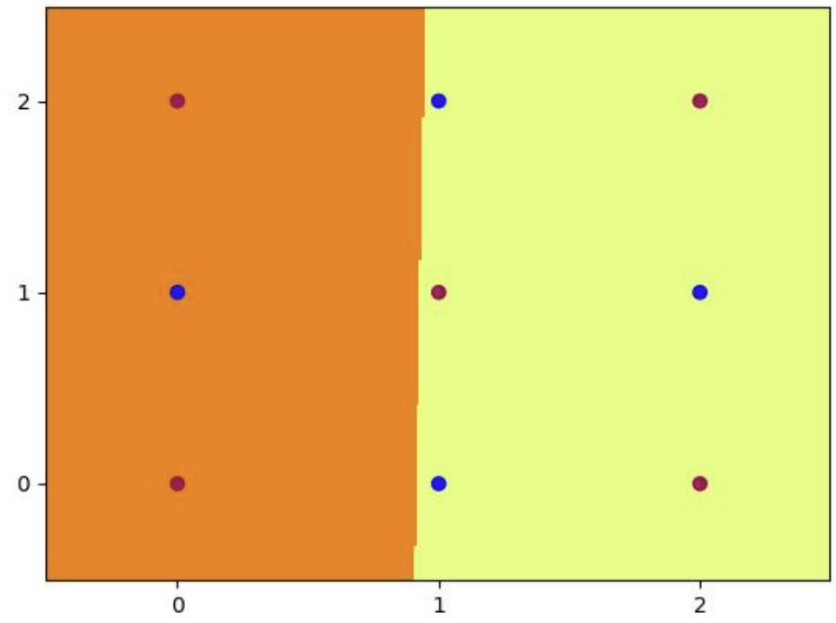
Hid_5_1:



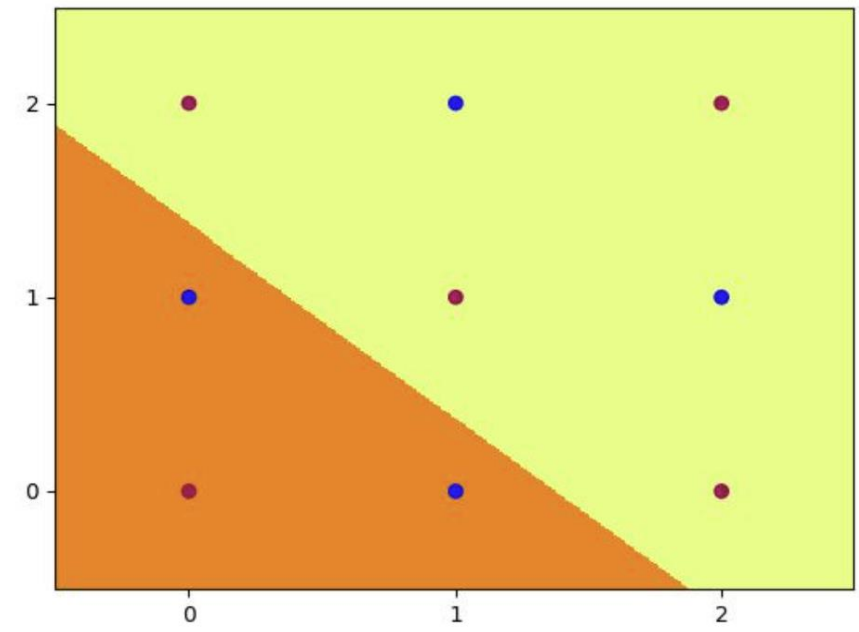
Hid_5_2:



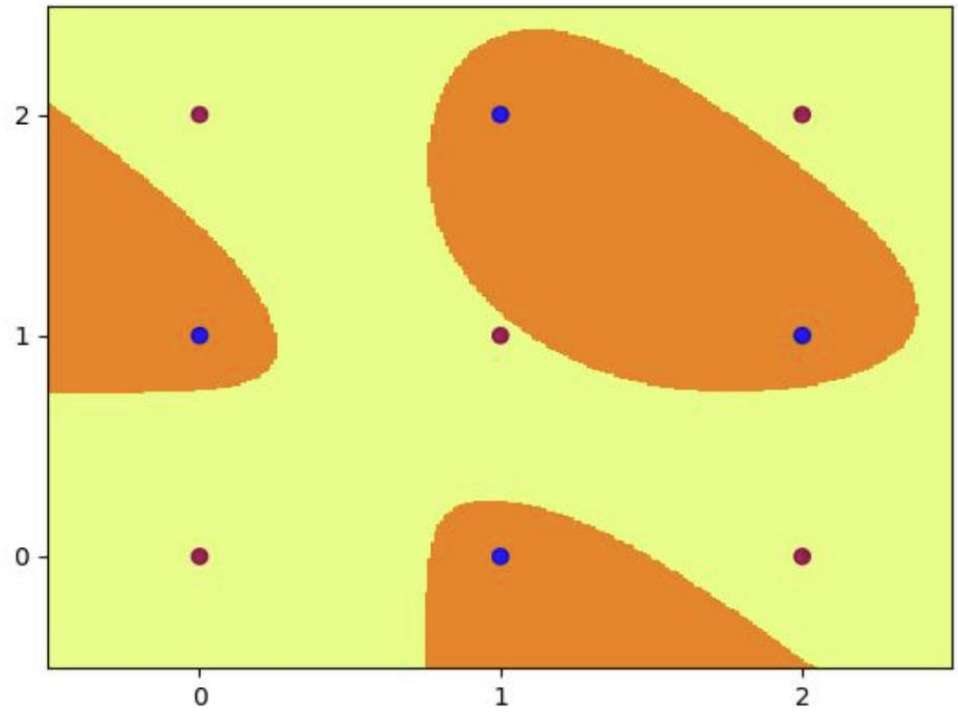
Hid_5_3:



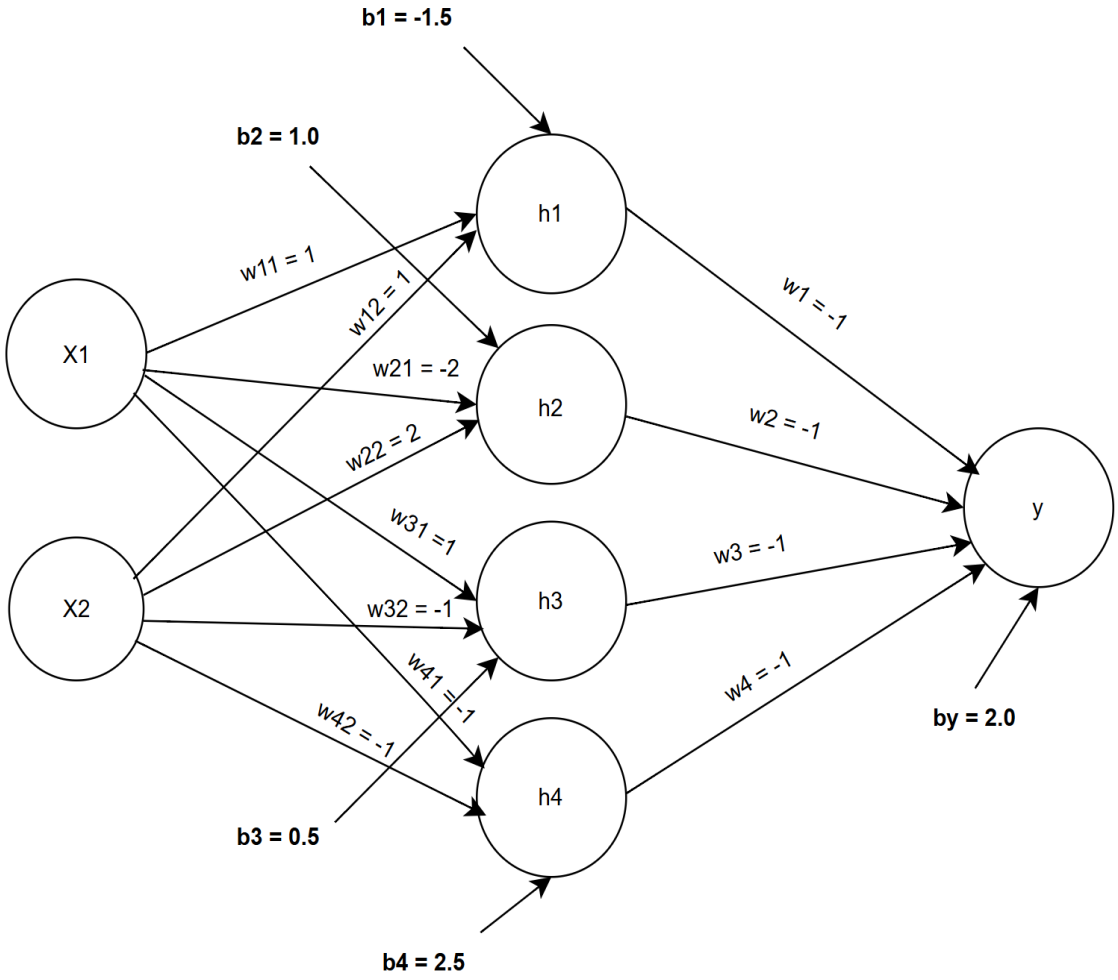
Hid_5_4:



Output_5:



Q2:



Check it is correct:

```
!python3 check_main.py --act step --hid 4 --set_weights
```

```
Initial Weights:
tensor([[ 1.,  1.],
        [-2.,  2.],
        [ 1., -1.],
        [-1., -1.]])
tensor([-1.5000,  1.0000,  0.5000,  2.5000])
tensor([[-1., -1., -1., -1.]])
tensor([2.])
Initial Accuracy:  100.0
```

Equations for the dividing line determined by each hidden node is:

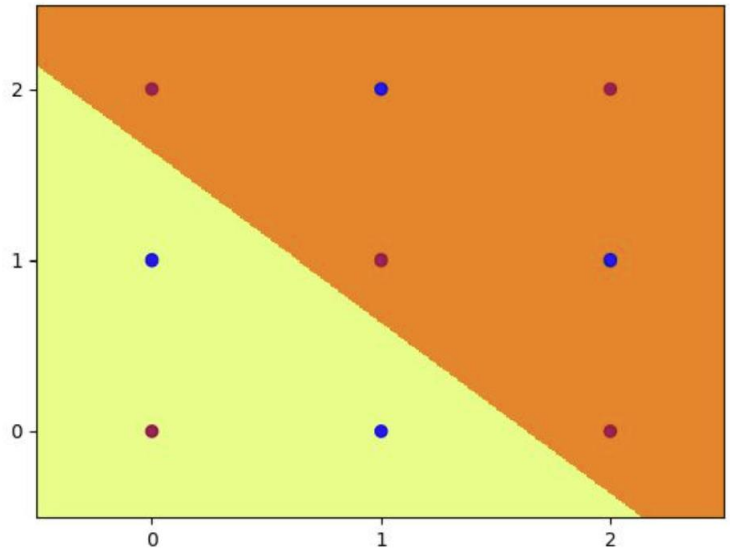
Hidden units	Weights	bias	Equations
h1	[1.0, 1.0]	-1.5	$1.0x1 + 1.0x2 - 1.5 = 0$
h2	[-2.0, 2.0]	1.0	$-2.0x1 + 2.0x2 + 1.0 = 0$
h3	[1.0, -1.0]	0.5	$1.0x1 - 1.0x2 + 0.5 = 0$
h4	[-1.0, -1.0]	2.5	$-1.0x1 - 1.0x2 + 2.5 = 0$

Table for the activations of all hidden nodes and output node (9 training points) :

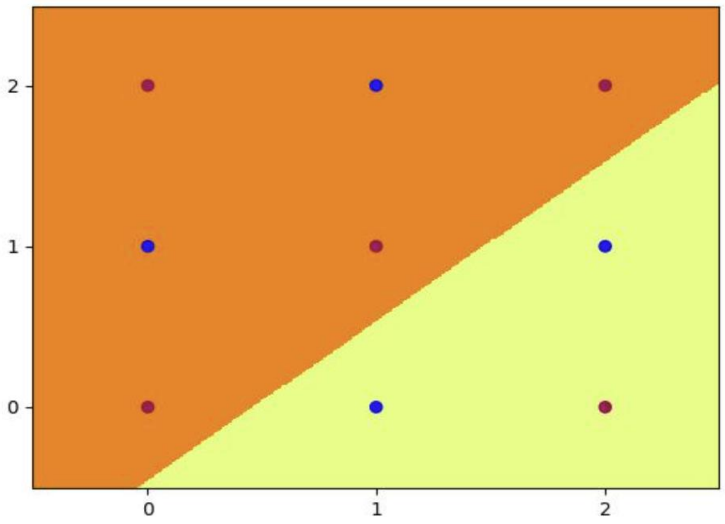
Sample	**x1**	**x2**	**h1**	**h2**	**h3**	**h4**	**y(output)**
1	0	0	1	0	0	0	0
2	0	1	1	0	1	0	1
3	0	2	0	0	1	0	0
4	1	0	1	1	0	0	1
5	1	1	0	0	0	0	0
6	1	2	0	0	1	1	1
7	2	0	0	1	0	0	0
8	2	1	0	1	0	1	1
9	2	2	0	0	0	1	0

Q3:

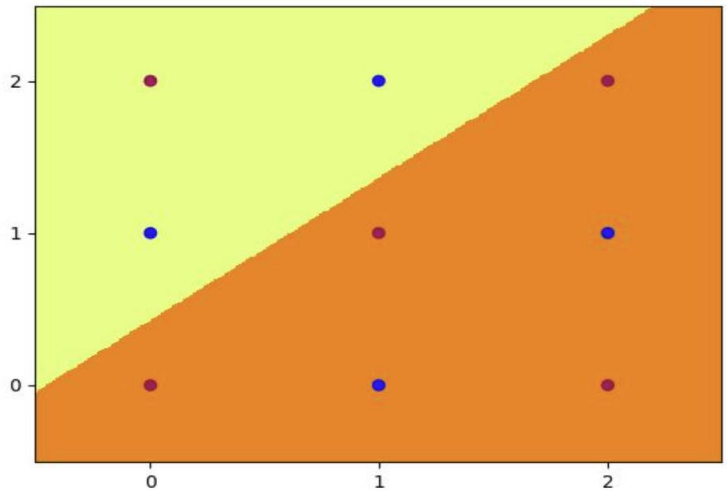
Hid4_0:



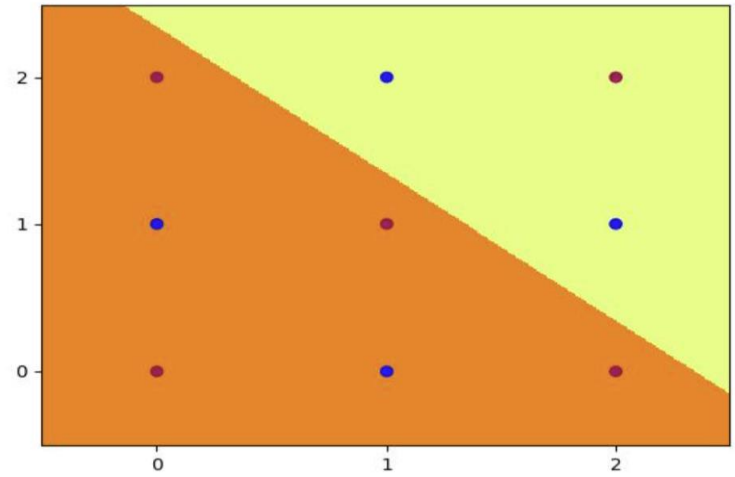
Hid4_1:



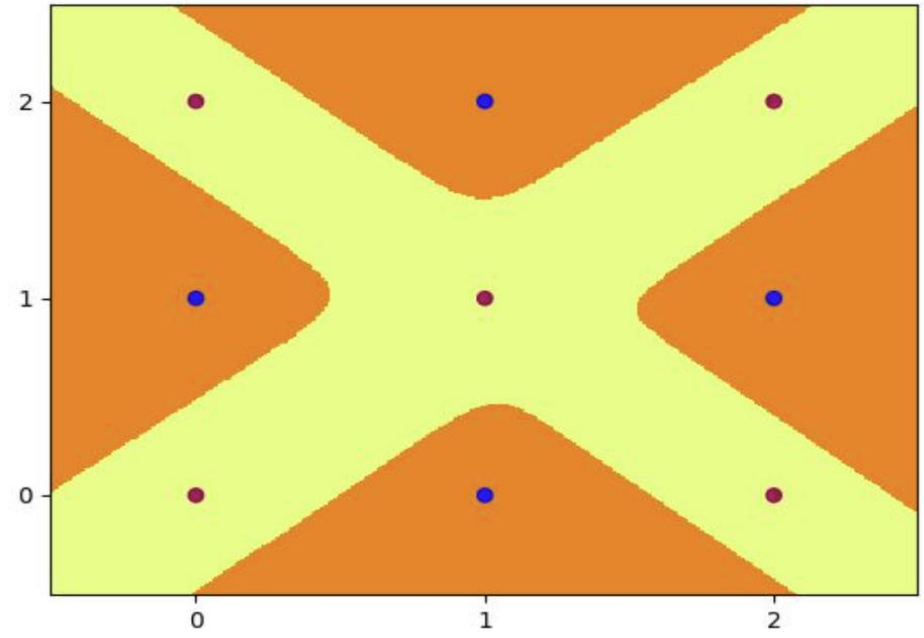
Hid4_2:



Hid4_3:

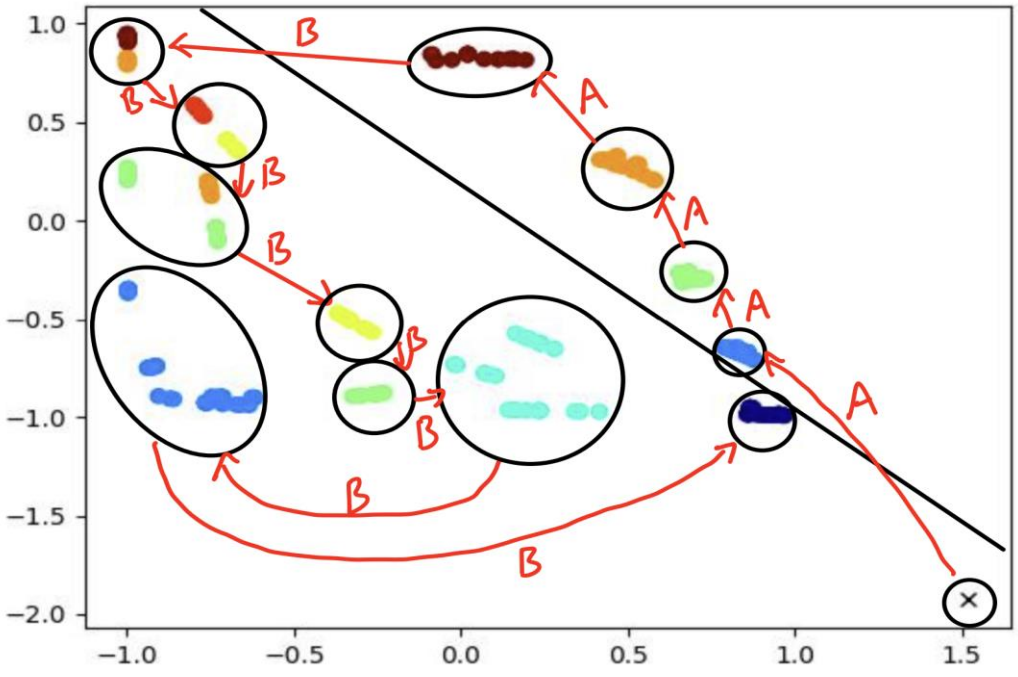


Output4:



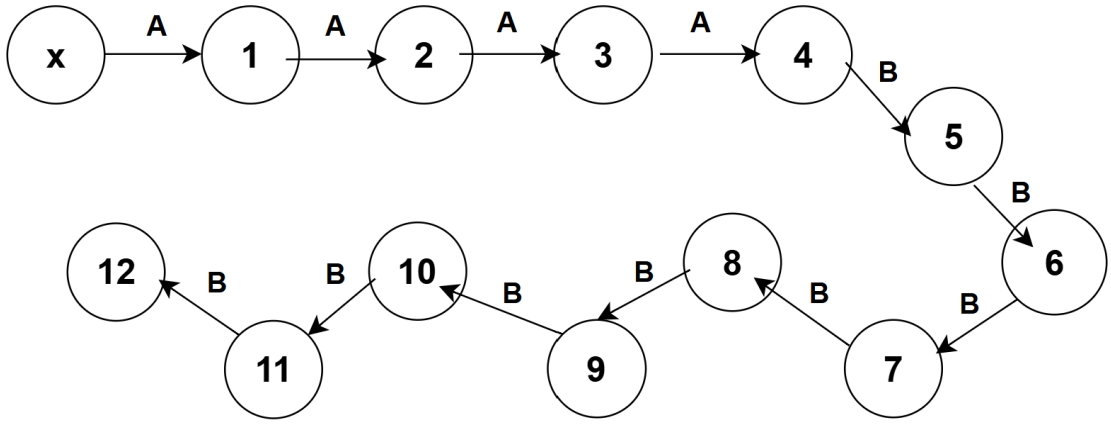
Part 3: Hidden Unit Dynamics for Recurrent Networks

Q1:



Q2:

Each number means cluster.



Q3:

In the anb2n task, in the initial stage (before the first B appears), the predicted probability of A gradually decreases, and the probability of B gradually increases, this shows that the hidden state is (accumulating demand) which means the more A there are, the more the network expects that more B is needed to meet the rule of "n A's correspond to 2n B's" in the task.

After the first B appears, the hidden state "determines" the number of Bs required (n), so the predicted probability of B stabilizes at a high level (close to 100%, confirming the count), while the probability of A is temporarily suppressed (close to 0%).

Then, after the last B appears, the hidden state is reset to start a new round, and the

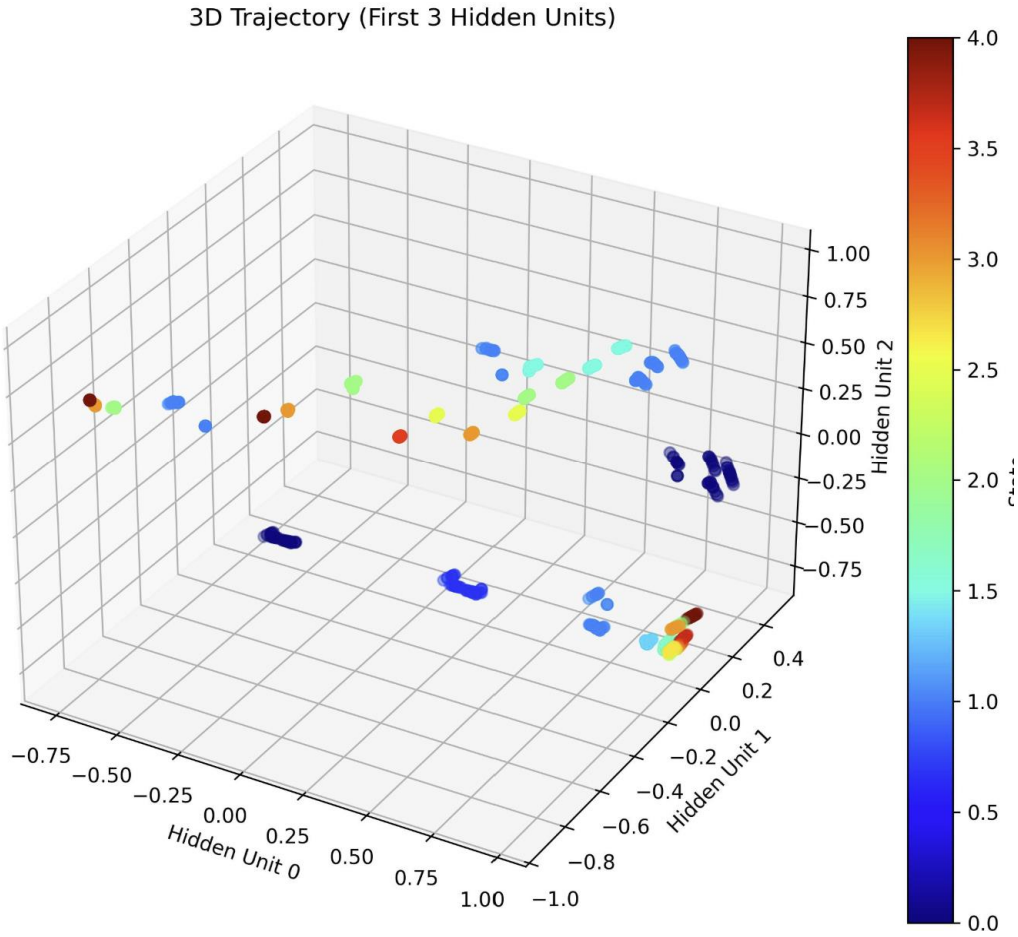
predicted probability of A rises sharply (close to 90%), marking the beginning of a new sequence.

Therefore, this process directly reflects the network's ability to learn temporal correlations.

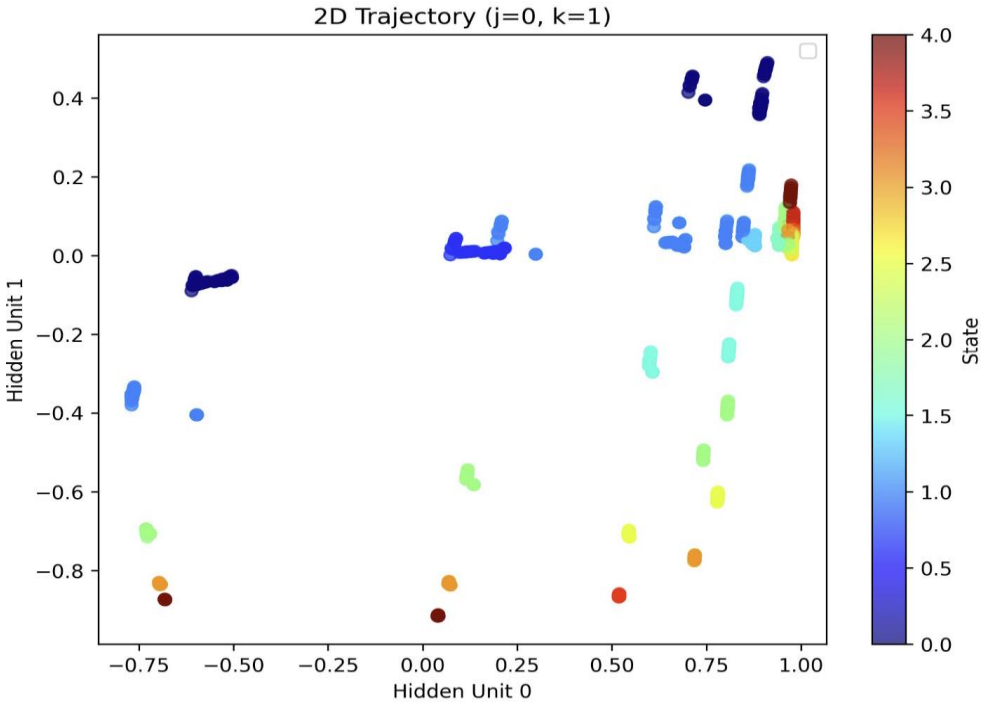
```
symbol= AAAABBBBBBBBAAAABBBBBBBBABBAAAABBBBBBBBABBA
hidden activations and output probabilities:
A [ 0.83 -0.65] [0.81 0.19]
A [ 0.68 -0.26] [0.65 0.35]
A [0.47 0.33] [0.35 0.65]
A [0.02 0.84] [0.03 0.97]
B [-1.    0.93] [0.  1.]
B [-0.79  0.57] [0.  1.]
B [-0.75  0.18] [0.  1.]
B [-0.33 -0.51] [0.  1.]
B [-0.26 -0.89] [0.  1.]
B [ 0.24 -0.97] [0.07 0.93]
B [-0.76 -0.93] [0.  1.]
B [ 0.94 -0.99] [0.89 0.11]
A [ 0.82 -0.67] [0.8 0.2]
```

Q4:

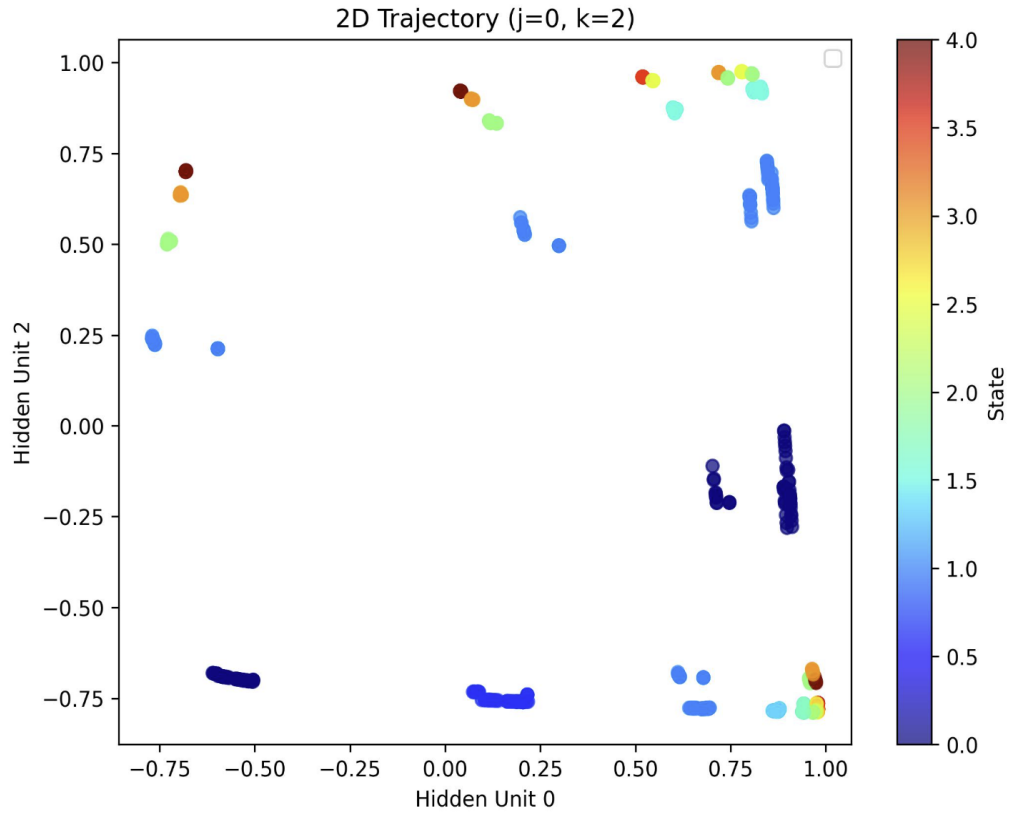
3D plot



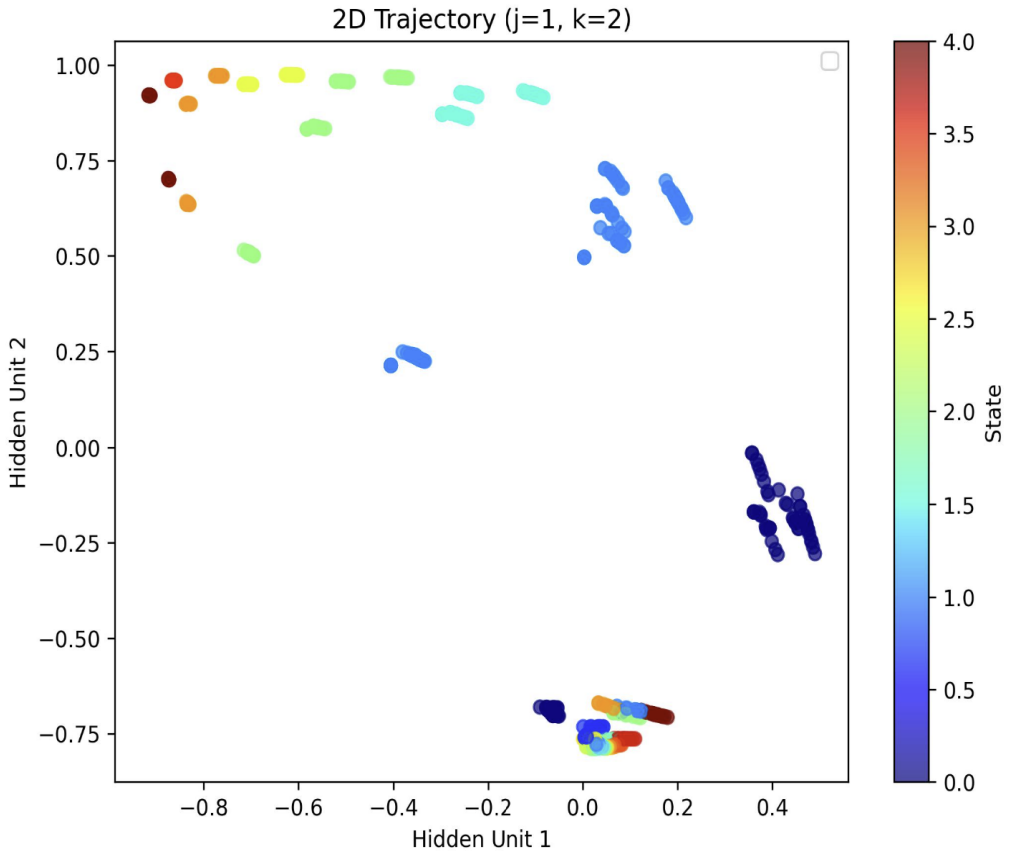
2D plot (j=0, k=1):



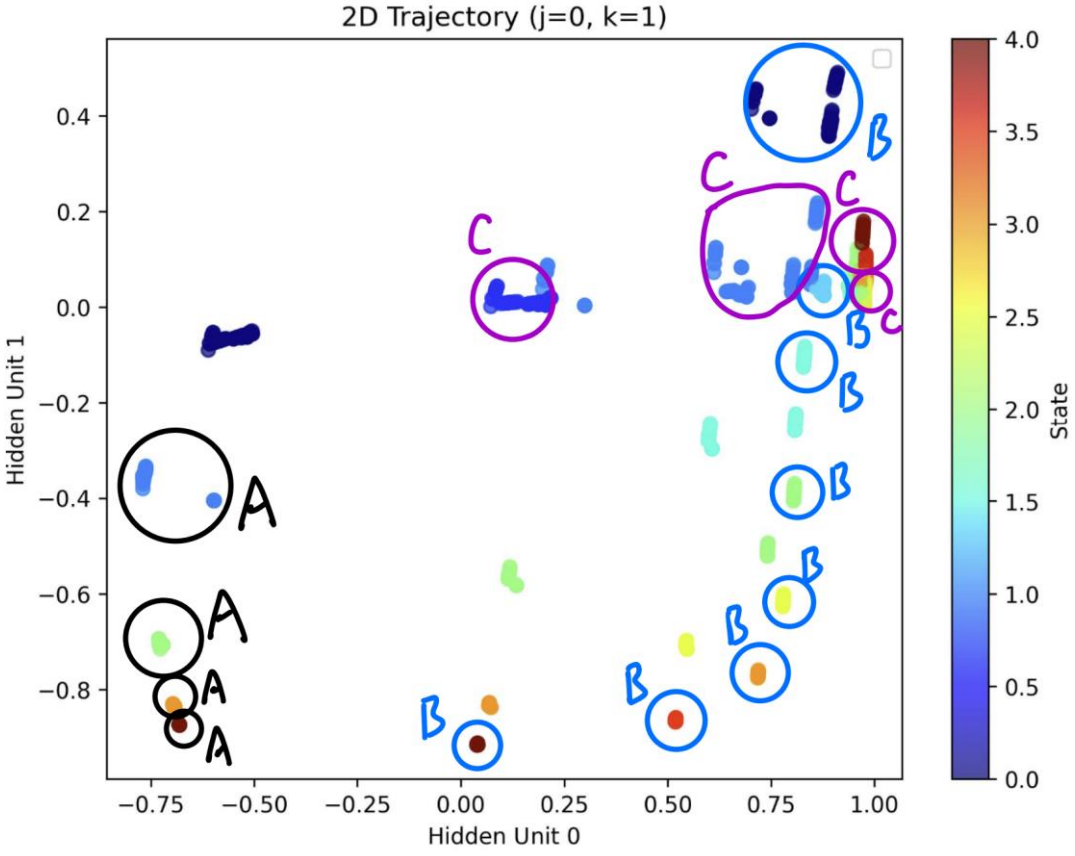
2D plot (j=0, k=2):



2D plot (j=1, k=2):



Annotated 2D plot (j=0, k=1):



Q5:

The reason why LSTM can complete long sequence prediction tasks such as $a^n b^{2n} c^{3n}$ well is the collaborative design of its memory cell and three-gate structure. As a "long-term memory bank", the memory cell dynamically controls the retention, writing and output of information through three gates (forget gate, input gate, and output gate), solving the long sequence forgetting problem caused by gradient disappearance or gradient explosion in normal RNN.

The functions of these three gates are:

1. Forget gate: suppress irrelevant historical information (for example, suppress early irrelevant A signals when B or C is generated), and only retain key memory related to the task.

2. Input gate: filter and write the core signal of the current input (such as the number of A requirements, the count of B) and strengthen the storage of task-related information.

3. Output gate: control the output amount of information in the memory cell to avoid redundant interference (such as only outputting the count result when C is generated).

Then, in the anb2nc3n task (generating the sequence "na→2nb→3nc→na"), the LSTM completes long dependency tracking in stages through three gates.

In Phase A: In the initial stage (before the first B appears), A's output probability drops from 0.7 to 0.3 (e.g., [0.7, 0.3, 0]-> [0.3, 0.7, 0]), while B's probability increases f rom 0.3 to 0.63. This reflects the hidden state accumulating demand: each A signals " more B is needed later" (specifically, 2n B's per A), so the network suppresses A predictions and strengthens B expectations.

In Phase B: Once the first B appears, B's output probability stabilizes near 100% (e.g., [0.01, 0.99, 0]-> [0, 0.97, 0.03]), confirming the "2n B" count. Meanwhile, A's probability decreases to 0% ~ 1% (e.g., [0.01, 0.99, 0]-> [0, 0.07, 0.93]). Here, the forget gate suppresses A's interference, and the input gate reinforces B's count signal, locking the "2n B" memory in the hidden state.

In Phase C: After the last B, C's output probability increases to 100% (e.g., [0, 0, 1], while A and B's probabilities drop to near 0% (e.g., [0, 0.03, 0.97]). The output gate triggers C generation using the "2n" count stored in the hidden state, converting it to "3n C's".

Restart phase: Finally, the hidden state resets: A's probability rebounds to 0.93 (e.g., [0.93, 0.1, 0.07]) after last C output, and B, C probabilities drop to 1% ~ 7%. This reset clears the "2n B" and "3n C" memory, preparing the network to start a new cycle (na→2nb→3nc→na).

```
hidden activations and output probabilities:
A [-0.6  -0.41  0.21] [0.7 0.3 0. ]
A [-0.72 -0.71  0.51] [0.53 0.47 0.  ]
A [-0.69 -0.84  0.64] [0.37 0.63 0.  ]
A [-0.68 -0.87  0.7 ] [0.3 0.7 0. ]
B [ 0.04 -0.92  0.92] [0.01 0.99 0.  ]
B [ 0.52 -0.87  0.96] [0.  1.  0.]
B [ 0.72 -0.77  0.97] [0.  1.  0.]
B [ 0.78 -0.62  0.97] [0.  1.  0.]
B [ 0.81 -0.4   0.97] [0.  1.  0.]
B [ 0.83 -0.12  0.93] [0.    0.99 0.01]
B [0.86 0.18 0.68] [0.    0.94 0.06]
B [ 0.9   0.46 -0.15] [0.    0.03 0.97]
C [ 0.97  0.13 -0.69] [0.  0.  1.]
C [ 0.98  0.07 -0.76] [0.  0.  1.]
C [ 0.98  0.06 -0.78] [0.  0.  1.]
C [ 0.98  0.05 -0.79] [0.  0.  1.]
C [ 0.98  0.05 -0.79] [0.  0.  1.]
C [ 0.97  0.05 -0.79] [0.  0.  1.]
C [ 0.97  0.05 -0.79] [0.  0.  1.]
C [ 0.94  0.04 -0.79] [0.  0.  1.]
C [ 0.86  0.04 -0.78] [0.  0.  1.]
C [ 0.64  0.03 -0.78] [0.  0.  1.]
C [ 0.1   0.01 -0.75] [0.09 0.01 0.9 ]
C [-0.59 -0.07 -0.69] [0.93 0.01 0.07]
A [-0.77 -0.36  0.24] [0.79 0.21 0.  ]
```