COMP9444 Neural Networks and Deep Learning

Term 2, 2025

Assignment – Characters and Hidden Unit Dynamics

Z5574599 - Shaoran Liu

Part 1: Japanese Character Recognition

1. The below screenshot is the final accuracy and confusion matrix result.

```
<class 'numpy.ndarray'>
[[768.   5.   8.  13.  31.  62.   2.  64.  30.  17.]
 [  7. 671. 106.  18.  29.  23.  58.  12.  26.  50.]
 [  7.  61. 688.  26.  26.  23.  46.  37.  47.  39.]
 [  5.  38.  56. 761.  14.  56.  13.  18.  27.  12.]
 [ 64.  54.  76.  21. 623.  20.  33.  35.  20.  54.]
 [  8.  28. 124.  17.  20. 725.  26.   9.  33.  10.]
 [  5.  21. 144.  10.  25.  24. 729.  20.  10.  12.]
 [ 18.  29.  26.  12.  83.  17.  54. 620.  92.  49.]
 [ 10.  36.  89.  43.   8.  31.  45.   6. 709.  23.]
 [  9.  54.  85.   3.  55.  32.  19.  32.  40. 671.]]

Test set: Average loss: 1.0064, Accuracy: 6965/10000 (70%)

z5574599@vx07:~/Documents/COMP9444/a1/a1$
```

2. First time I choose number 100 as hidden nodes, but only 83% accuracy, when I change to 300, it achieves 85% accuracy.

```
<class 'numpy.ndarray'>
[[860.   2.   2.   7.  25.  30.   2.  38.  28.   6.]
 [  4. 828.  30.   4.  16.   9.  60.   4.  17.  28.]
 [  7.  15. 841.  42.  12.  21.  23.   8.  18.  13.]
 [  3.  11.  28. 922.   0.  16.   5.   3.   6.   6.]
 [ 36.  32.  21.   5. 820.   7.  28.  19.  20.  12.]
 [  9.  15.  86.   8.  12. 832.  21.   2.  10.   5.]
 [  3.  13.  49.   9.  13.   3. 895.   8.   2.   5.]
 [ 15.  13.  19.   3.  24.  10.  30. 835.  21.  30.]
 [  9.  35.  32.  54.   2.   8.  30.   5. 818.   7.]
 [  3.  18.  44.   4.  31.   6.  18.  22.  11. 843.]]

Test set: Average loss: 0.4959, Accuracy: 8494/10000 (85%)

z5574599@vx07:~/Documents/COMP9444/a1/a1$
```

3. After setup a NetConv with two convolutional layers plus one fully connected layer, I achieve 94% accuracy on the test set after 10 training epochs. (this part run too slow on vlab, I use my pc to run this part code)

```
Windows PowerShell
 [ 12.   6.   3.   0.   8.   4.   9. 935.   5.  18.]
 [  7.  15.   6.   5.   9.   3.   2.   1. 949.   3.]
 [ 12.   6.   7.   4.   9.   1.   2.   7.  10. 942.]]

Test set: Average loss: 0.2737, Accuracy: 9383/10000 (94%)

Train Epoch: 10 [0/60000 (0%)]  Loss: 0.012215
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.008310
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.006599
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.006174
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.009212
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.012070
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.003409
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.010827
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.005722
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.006061
<class 'numpy.ndarray'>
[[963.   4.   2.   0.  17.   3.   0.   6.   1.   4.]
 [  1. 937.   6.   2.  10.   2.  25.   5.   5.   7.]
 [  9.   3. 901.  25.  12.  13.  11.   9.   6.  11.]
 [  1.   3.  15. 962.   1.   5.   3.   2.   1.   7.]
 [ 22.   7.   3.   9. 924.   3.  12.   8.   9.   3.]
 [  6.   4.  40.   5.   3. 913.  12.   2.   3.  12.]
 [  4.   6.  16.   1.   8.   1. 958.   4.   0.   2.]
 [ 14.   7.   3.   1.   6.   3.   7. 936.   5.  18.]
 [  8.  18.   9.   4.   8.   3.   1.   2. 944.   3.]
 [ 11.   7.  10.   5.   6.   2.   3.   3.   9. 944.]]

Test set: Average loss: 0.2861, Accuracy: 9382/10000 (94%)
```

4.a

The accuracy of NetLin after 10 train epoch is about 70%.

The accuracy of NetFull after 10 train epoch is about 85%.

The accuracy of NetConv after 10 train epoch is about 94%.

4.b

The NetLin only have one layer, the number of independent parameters is 7850.

(28*28*10+10=7850)

The NetFull have two layers, the number of independent parameters is 238510.

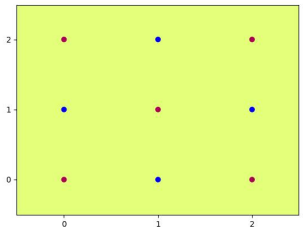
(28*28*300+300+300*10+10)

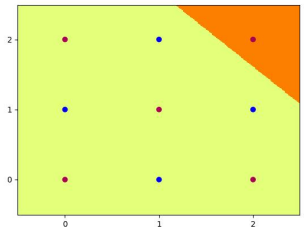
The NetConv have 3 layers, the number of independent parameters is 553866.
(32*1*5*5+32+64*32*5*5+64+64*28*28*10+10)

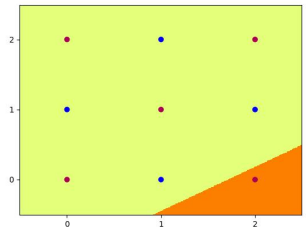
4.c
The confusion matrix for NetLin, I found a lot of `ha` and `ma` mistaken for `su`, because except the values on the diagonal, the larger number in figure are located in the third column of the sixth row and the third column of the seventh row, their values are 124 and 144 respectively, which means that the `ha` represented by the sixth row and the `ma` represented by the seventh row can be easily mistaken for `su`.
The confusion matrix for NetFull, I found a lot of `ha` mistaken for `su`, because except the values on the diagonal, the larger number in figure is located in the third column of the sixth row.
The confusion matrix for NetConv, I found a lot of `ha` mistaken for `su`, because except the values on the diagonal, the larger number in figure is located in the third column of the sixth row.

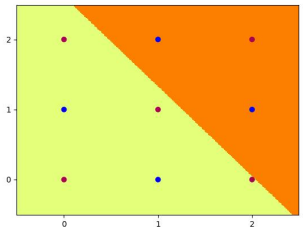
Part 2: Multi-Layer Perceptron

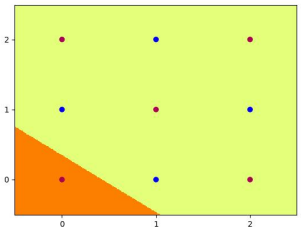
1.After run the code, it successfully achieves accuracy of 100%, the figure for each hidden node and the network as a whole are below.

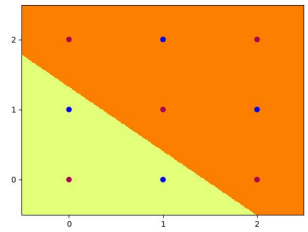

check.jpg

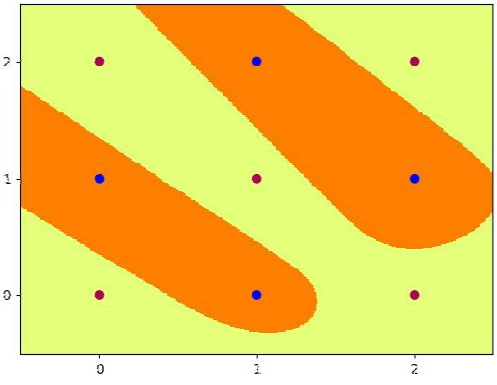

hid_5_0.jpg


hid_5_1.jpg


hid_5_2.jpg


hid_5_3.jpg


hid_5_4.jpg



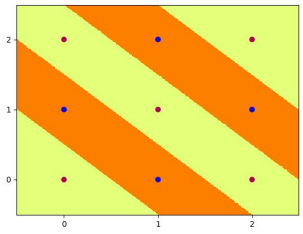
```
Windows PowerShell
ep: 4800 loss: 0.3432 acc: 100.00
ep: 4900 loss: 0.3297 acc: 100.00
ep: 5000 loss: 0.3152 acc: 100.00
ep: 5100 loss: 0.3013 acc: 100.00
ep: 5200 loss: 0.2888 acc: 100.00
ep: 5300 loss: 0.2774 acc: 100.00
ep: 5400 loss: 0.2666 acc: 100.00
ep: 5500 loss: 0.2563 acc: 100.00
ep: 5600 loss: 0.2464 acc: 100.00
ep: 5700 loss: 0.2369 acc: 100.00
ep: 5800 loss: 0.2277 acc: 100.00
ep: 5900 loss: 0.2190 acc: 100.00
ep: 6000 loss: 0.2106 acc: 100.00
ep: 6100 loss: 0.2025 acc: 100.00
ep: 6200 loss: 0.1948 acc: 100.00
ep: 6300 loss: 0.1874 acc: 100.00
ep: 6400 loss: 0.1802 acc: 100.00
ep: 6500 loss: 0.1733 acc: 100.00
ep: 6600 loss: 0.1667 acc: 100.00
Final Weights:
tensor([[ 2.0507,  1.9263],
        [ 4.1214, -6.5457],
        [ 3.6439,  2.8461],
        [-5.2748, -6.4905],
        [ 5.0388,  5.4980]])
tensor([-7.2064, -7.0510, -7.4288,  2.2637, -7.2660])
tensor([[-6.2403, -5.0347,  6.2209, -6.4385, -6.2780]])
tensor([3.2447])
Final Accuracy:  100.0
(venv) PS D:\MiniGG_Project\COMP9444\a1\a1>
```

out_5.jpg

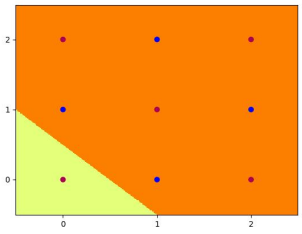
2.Table below presents the activations of the four hidden nodes and the output node for all 9 training items, and the following figure illustrate the decision computed by each hidden node (hid_4_?.jpg) and the network as a whole (out_4.jpg)

No.	(x, y)	check	hid_4_0	hid_4_1	hid_4_2	hid_4_3	out_4
1	(0, 0)	0	0	0	0	0	0
2	(1, 0)	1	1	0	0	0	1
3	(2, 0)	0	1	1	0	0	0
4	(0, 1)	1	1	0	0	0	1
5	(1, 1)	0	1	1	0	0	0
6	(2, 1)	1	1	1	1	0	1
7	(0, 2)	0	1	1	0	0	0
8	(1, 2)	1	1	1	1	0	1
9	(2, 2)	0	1	1	1	1	0

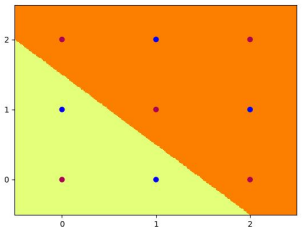
```
(venv) PS D:\MiniGG_Project\COMP9444\a1\a1> python check_main.py --act step --hid 4 --set_weights
Initial Weights:
tensor([[10., 10.],
        [10., 10.],
        [10., 10.],
        [10., 10.]])
tensor([ -5., -15., -25., -35.])
tensor([[ 10., -10.,  10., -10.]])
tensor([-5.])
Initial Accuracy:  100.0
```



check.jpg



hid_4_0.jpg

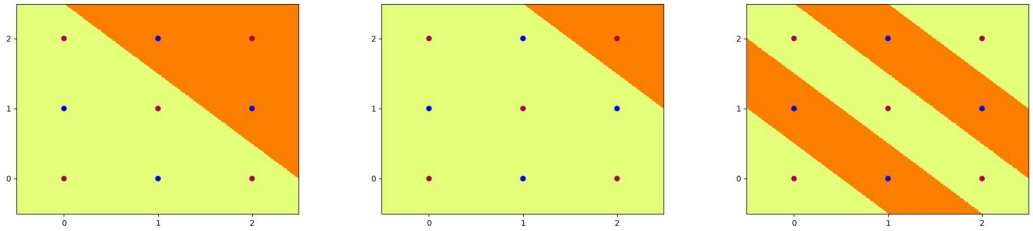


hid_4_1.jpg

hid_4_2.jpg

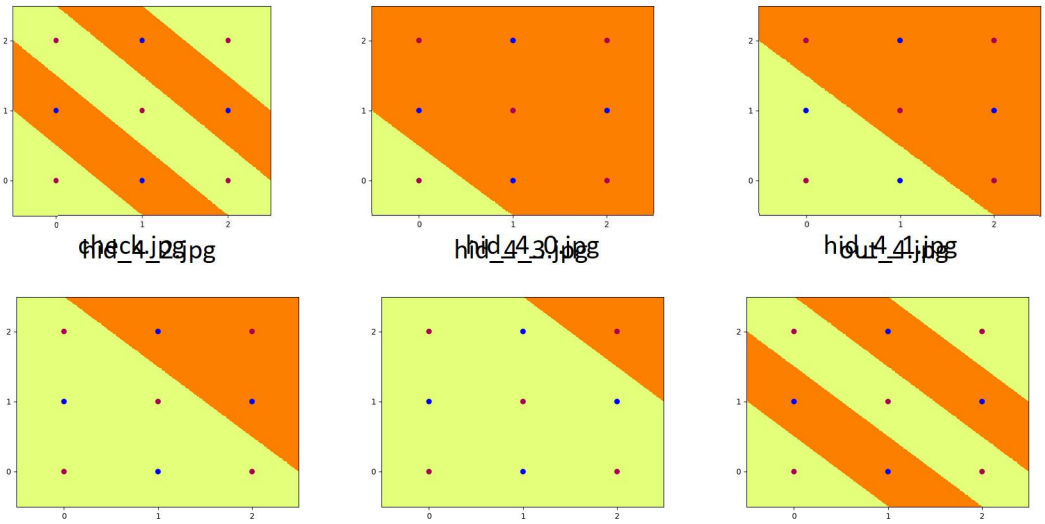
hid_4_3.jpg

out_4.jpg



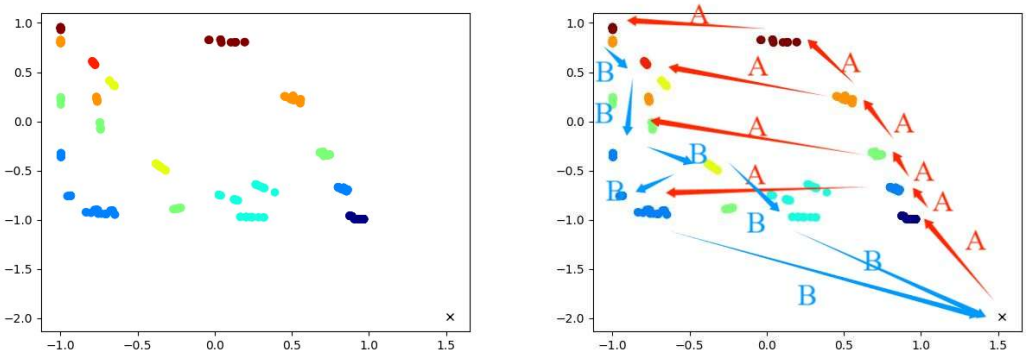
3.The below figure are the result of rescale your hand-crafted weights and biases from Part 2, multiplying all of them by 10.

```
(venv) PS D:\MiniGG_Project\COMP9444\a1\a1> python check_main.py --act sig --hid 4 --set_weights
Initial Weights:
tensor([[10., 10.],
        [10., 10.],
        [10., 10.],
        [10., 10.]])
tensor([ -5., -15., -25., -35.])
tensor([[ 10., -10.,  10., -10.]])
tensor([-5.])
Initial Accuracy:  100.0
(venv) PS D:\MiniGG_Project\COMP9444\a1\a1>
```

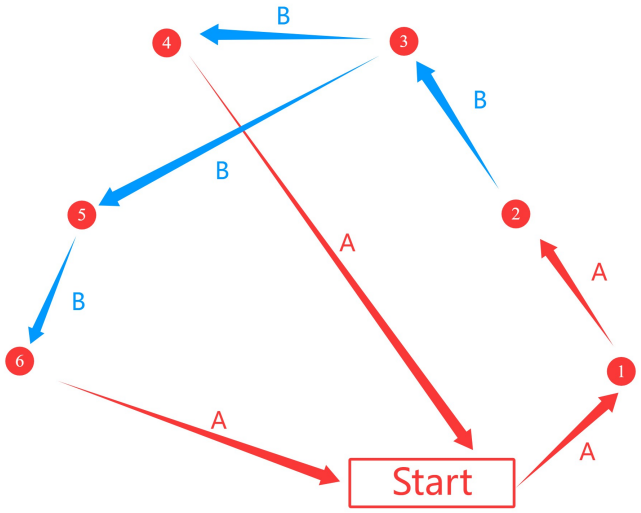


Part 3: Hidden Unit Dynamics for Recurrent Networks

1. A SRN with 2 hidden nodes on the anb2n language prediction task.

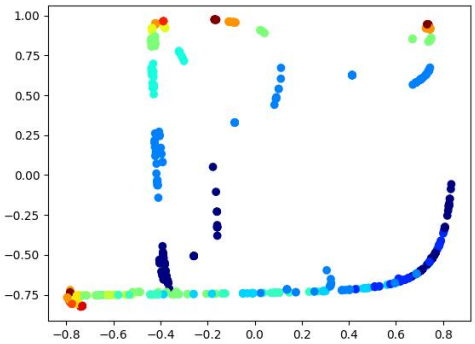


2. The following picture is equivalent to the finite state machine.

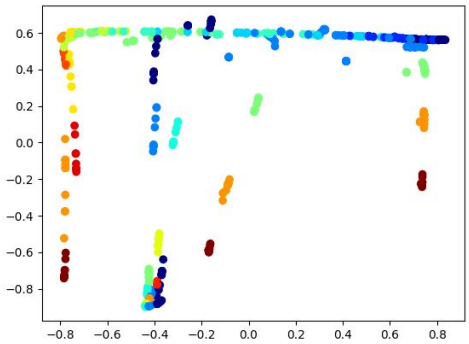


3. In the anb2n task, each input sequence consists of n A and 2n B. The A after the first A is uncertain, but the state of A changes continuously, so the prediction is probabilistic. The first B is uncertain, and the subsequent B is certain. After the last B, the model's hidden state knows that it has ended, and it is ready to enter the next sequence, so the probability of the first A in the next sequence can be accurately obtained.

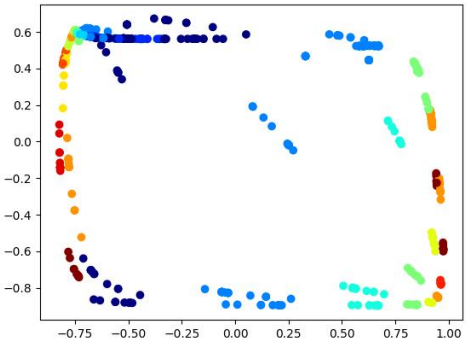
4. The below 4 figures is an LSTM with 3 hidden nodes on the anb2nc3n language prediction task.



anb2nc3n_lstm3_01.jpg



anb2nc3n_lstm3_02.jpg



anb2nc3n_lstm3_12.jpg

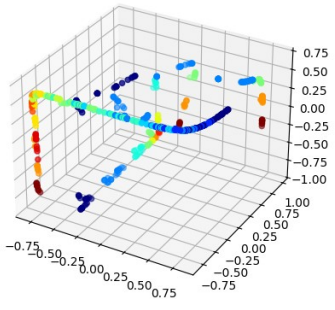
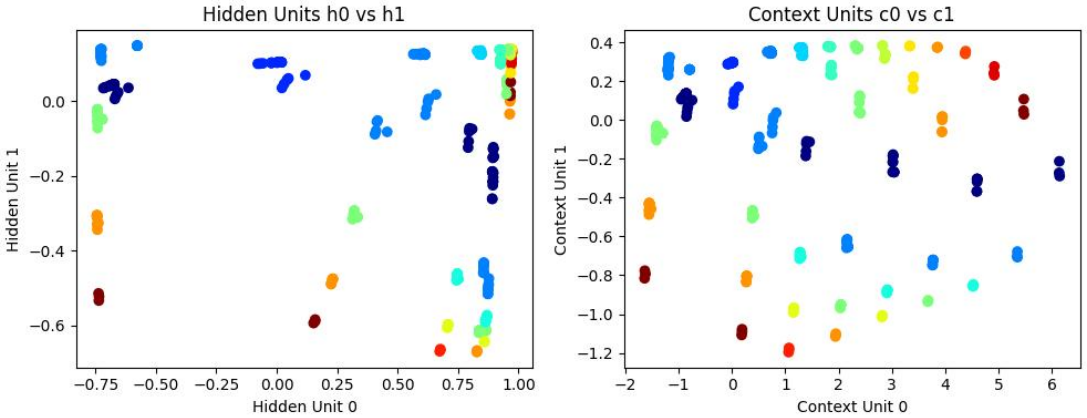
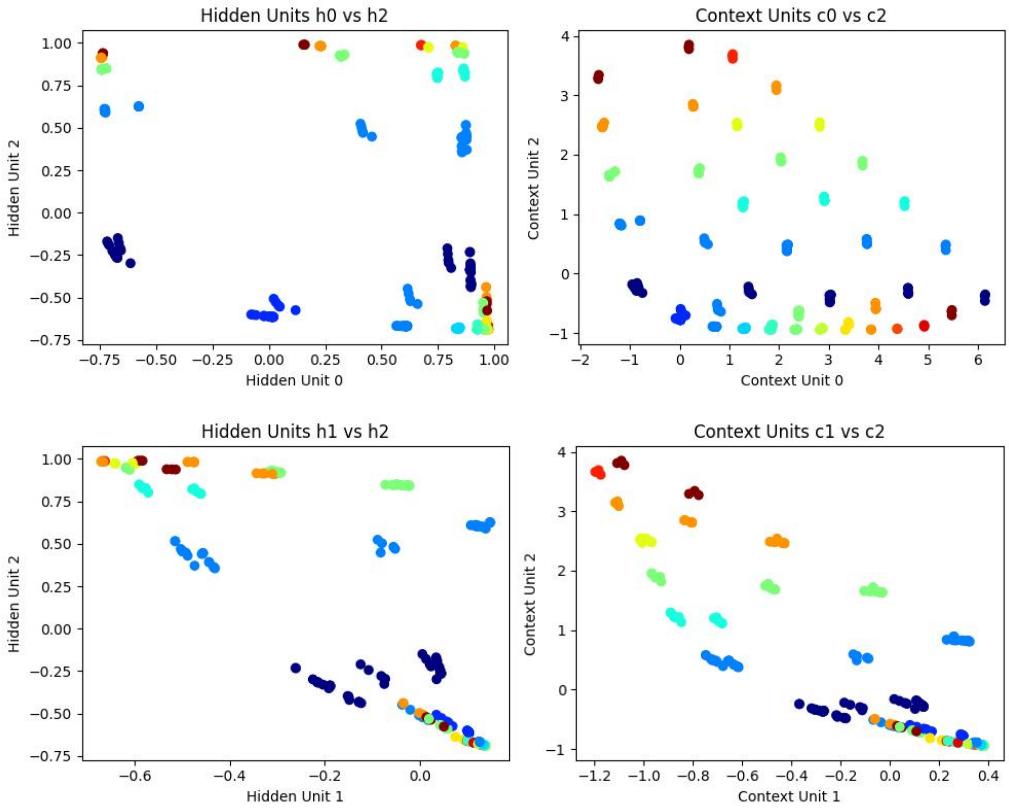


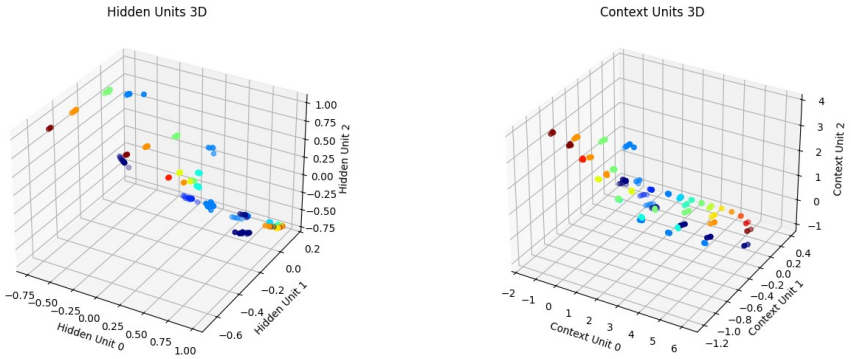
Figure_1.jpg

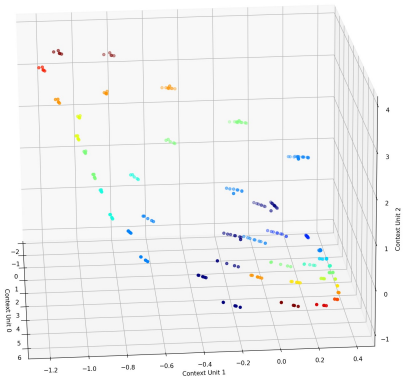
5. These 4 figures are generate after modifying the code, The Hidden Units diagram clearly reflects the stages of the sequence. The activations of each stage are concentrated in certain areas. For example, the points of the A input stage are mostly concentrated in the upper right corner, while the points of the C stage are distributed at the edge or the lower left corner. Context Units record smoother and longer-distance change trends and can be used as a "counter".

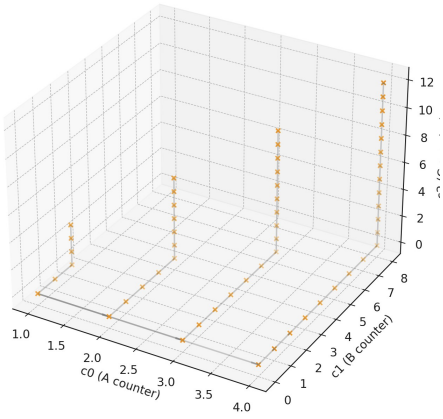




By rotating the three-dimensional trajectory diagram, we can intuitively observe the sequence changes of each stage over time. According to anb2nc3n and the variable ratio of 0(a) 1(b) 2(c) in the figure, it is simplified to the bottom right figure. The number of A is recorded by C0. At A, after h0 moves in a specific direction, it enters the "waiting for B" state (predicting the first B, uncertain height), and C1 increases to 2A when B appears. In the B stage, after h1 moves in a specific direction (predicting the next B, the height has been determined), h2 increases to 3A when C appears. Finally, in the C stage (predicting the next C, the height has been determined), h0 gradually becomes negative and h2 increases. Finally, C0 resets back to the A area and predicts a new A (uncertain height). Finally, the LSTM network successfully uses the context unit to memorize the number of A, and thus derives the prediction requirements of 2n B and 3n C.




Context Unit 0
Context Unit 1
Context Unit 2


c0 (A counter)
c1 (B counter)