Part 1: Japanese Character Recognition

1.1 Model NetLin final accuracy and confusion matrix

a)Accuracy: 6965/10000 (70%)

```
Test set: Average loss: 1.0093, Accuracy: 6965/10000 (70%)
```

b)Confusion matrix:

```
[[768.   5.   9.  12.  30.  62.   2.  62.  31.  19.]
 [  7. 672. 109.  18.  29.  22.  56.  14.  24.  49.]
 [  8.  61. 691.  25.  26.  21.  47.  36.  46.  39.]
 [  4.  37.  57. 762.  15.  57.  14.  18.  25.  11.]
 [ 59.  52.  81.  19. 619.  21.  33.  35.  23.  58.]
 [  8.  28. 127.  17.  20. 722.  28.   8.  32.  10.]
 [  5.  23. 148.  10.  27.  23. 720.  22.   9.  13.]
 [ 17.  30.  27.  11.  84.  17.  52. 625.  90.  47.]
 [ 11.  37.  95.  40.   6.  30.  45.   6. 706.  24.]
 [  8.  52.  84.   4.  53.  31.  20.  29.  39. 680.]]
```

1.2 A fully connected 2-layer network NetFull

a) Accuracy: 8443/10000 (84%)

```
Test set: Average loss: 0.5159, Accuracy: 8443/10000 (84%)
```

b)Confusion matrix:

```
[[846.   3.   4.   6.  31.  35.   4.  39.  26.   6.]
 [  6. 809.  36.   5.  22.  12.  59.   2.  20.  29.]
 [  6.  10. 838.  46.  11.  20.  23.  14.  19.  13.]
 [  3.   7.  29. 923.   2.  12.   5.   5.   5.   9.]
 [ 40.  28.  23.   7. 816.   7.  32.  14.  21.  12.]
 [ 10.  12.  78.  11.  12. 831.  24.   2.  14.   6.]
 [  3.  12.  50.   9.  15.   7. 892.   4.   2.   6.]
 [ 18.  12.  19.   4.  28.   8.  30. 822.  28.  31.]
 [ 12.  23.  32.  50.   4.   9.  27.   3. 833.   7.]
 [  6.  12.  60.   5.  25.   5.  25.  18.  11. 833.]]
```

c)Total Number of Independent Parameters in NetFull

Input dimension: 28 × 28 = 784 pixels

Hidden layer: 150 neurons with tanh activation

Output layer: 10 neurons with log_softmax activation

Layer-by-Layer Calculation:

1)Input layer to hidden layer:

Weight parameters: 784 inputs × 150 hidden units = 117,600

Bias parameters: 150

Total parameters: 117,600 + 150 = 117,750

2)Hidden layer to output layer:

Weight parameters: 150 hidden units × 10 outputs = 1,500

Bias parameters: 10

Total parameters: 1,500 + 10 = 1,510

Total Number of Independent Parameters in NetFull:

117,750 + 1,510 = 119,260 parameters

1.3 A NetConv convolutional network

a) Accuracy: 9265/10000 (93%)

```
Test set: Average loss: 0.2731, Accuracy: 9265/10000 (93%)
```

b)Confusion matrix:

```
[[962.   1.   1.   1.  15.   4.   0.  11.   3.   2.]
 [  3. 918.   7.   1.  11.   0.  40.   5.   7.   8.]
 [ 11.  11. 846.  52.  10.   8.  34.  16.   5.   7.]
 [  3.   3.  25. 949.   2.   4.   8.   3.   0.   3.]
 [ 24.   5.   6.   5. 919.   1.  10.  11.  12.   7.]
 [  6.  20.  40.   5.   8. 879.  18.  13.   9.   2.]
 [  3.   6.  13.   3.  11.   2. 953.   5.   2.   2.]
 [ 10.   3.   5.   2.   9.   1.   4. 937.   8.  21.]
 [ 10.  11.   6.   3.   4.   1.   1.   2. 960.   2.]
 [ 10.  12.   7.   4.  13.   1.   0.   3.   8. 942.]]
```

c)Conv Layer 1 Weight: 1 input channel × 16 output channels × 5 × 5 = 400

Bias: 16

Total: 416

Conv Layer 2 Weight: 16 input channels × 32 output channels × 5 × 5 = 12,800

Bias: 32

Total: 12,832

Fully Connected Layer Input size: 32 × 7 × 7 = 1,568

Weight: 1,568 × 10 = 15,680

Bias: 10

Total: 15,690

Therefore, the total number of independent parameters in NetConv is:

416 + 12,832 + 15,690 = 28,938

1.4 Briefly discuss the following points:

a) the relative accuracy of the three models

Model Name	Accuracy
NetLin	70%
NetFull	84%
NetConv	93%

Analysis:

1)**NetLin** is the simplest model, which only uses linear transformation and cannot effectively extract the spatial structure features of images, thus having the lowest accuracy rate.

2)**NetFull** introduces a hidden layer and a nonlinear activation function (tanh), which can extract more pattern information, thus significantly improving the accuracy rate.

3)**NetConv** employs a convolutional structure, capable of capturing local spatial features of images. It is most suitable for processing handwritten images, offering the best performance with an accuracy rate of up to 93%.

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

Model Name	Independent parameters
NetLin	7,850
NetFull	119,260
NetConv	28,938

Analysis: Although NetFull has the most parameters, it ignores spatial structure information, so its performance is still weaker than that of NetConv. NetConv has fewer parameters but a more reasonable structure, making full use of the convolutional feature extraction capability to achieve higher performance. Although NetLin has the fewest parameters, its expressive power is the weakest.

c) the confusion matrix for each model: which characters are most likely to be mistaken for which other characters, and why?

For the simplest NetLin model, due to the use of only linear mapping and the lack of nonlinear modeling and spatial feature extraction capabilities, the error rate is relatively high. **Specifically, the character "su" is often misjudged as the morphologically similar "tsu" and "ki", while "ki" is frequently misclassified as "re", "ma" or "su".** There are strong visual similarities among these characters, such as stroke length and bending Angle. Without nonlinear transformation, it is difficult for the model to distinguish them.

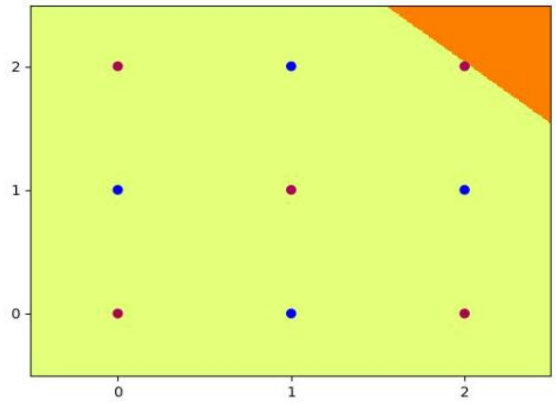
With addition of a nonlinear hidden layer in the NetFull model, over all recognition accuracy has been increased and confusion has been considerably decreased. **Nevertheless, some mixed categories are possible to notice. As an example, classification between "su" and "tsu" or "ki" is very fluid and the sound of "wo" is misjudged as either "ki" or "su".** It denotes that, in spite of the enhanced expressive power by the fully connected structure, the perception of the local spatial structure is missing as well as it cannot effectively capture the local differences of characters.

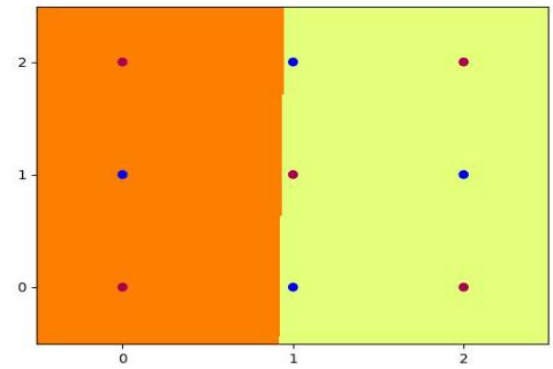
In the most powerful NetConv model, by leveraging the convolutional structure and the characteristics of the local receptive field, the model achieves a higher recognition accuracy rate on the vast majority of characters. **For instance, the recognition accuracy rate of "su" reached 846/1000, which is much higher than that of other models. However, there are still a few confusions. For instance, the confusion between "su" and "tsu" has not been completely eliminated, and "ki" may still be mistaken for "ma" or "re".** Most of these errors occur in samples with extremely similar stroke shapes or diverse handwriting styles, indicating that even convolutional networks still have room for improvement when dealing with characters with very similar shape details.

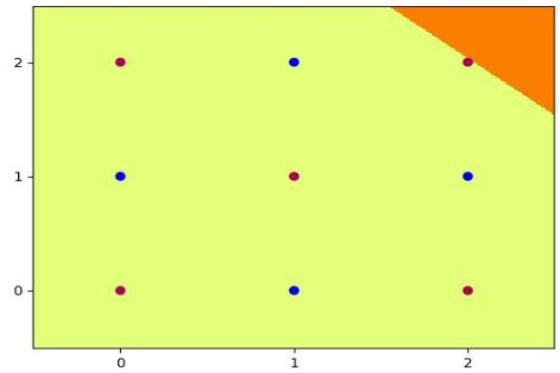
In general, the more intricate the model structure, the more the perception ability, the fewer there will be misclassifications in a confusion matrix and the more distinguishable between characters, as well.

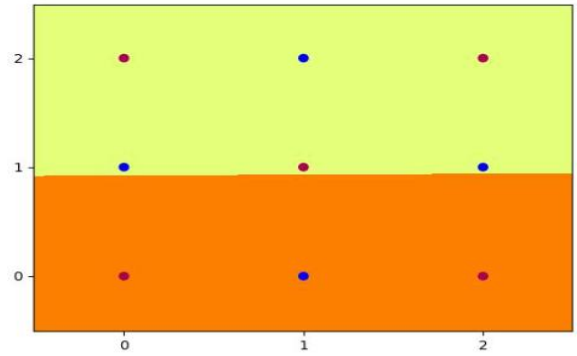
Part 2: Multi-Layer Perceptron

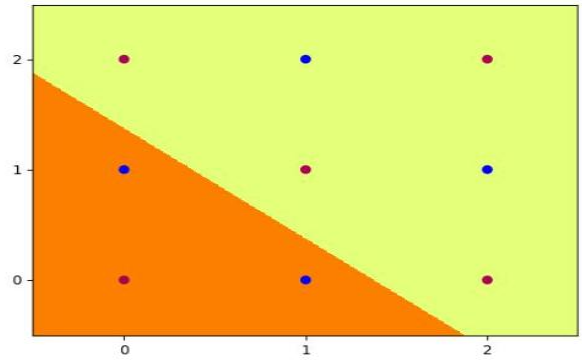
2.1 hid_5_0.jpg – hid_5_4.jpg are shown below



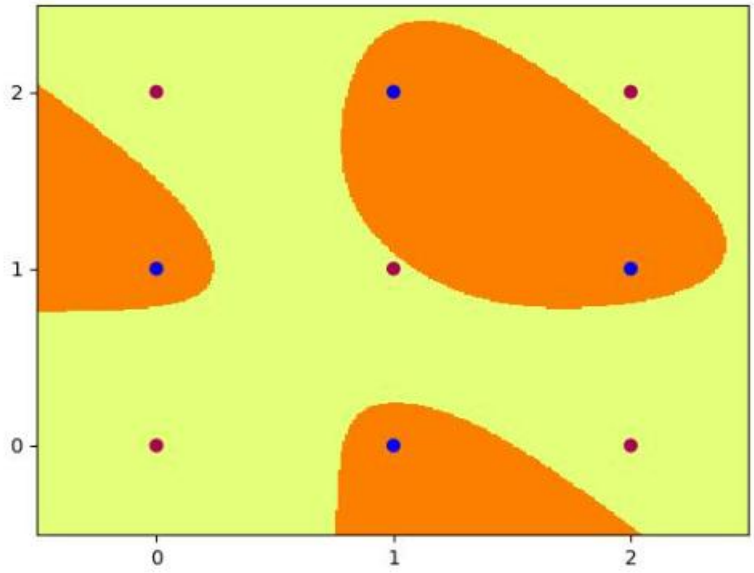






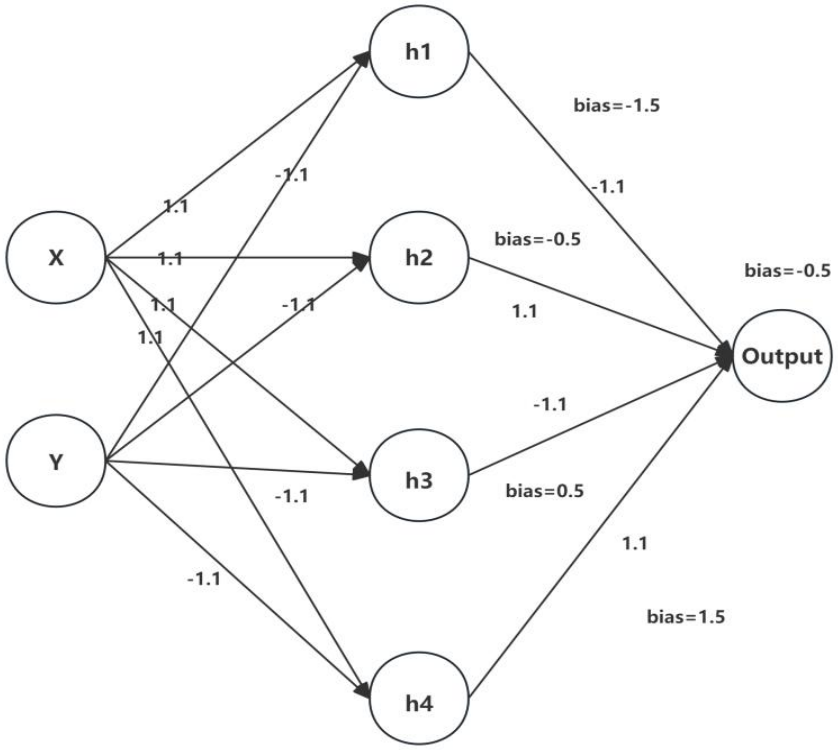


out_5.jpg is shown below



After several runs, the network successfully achieved **100% classification accuracy** on all 9 data points.The images above show the function computed by each hidden node (hid_5_?.jpg) and the final classification result of the network (out_5.jpg).

2.2 A diagram of the network



Equations for the dividing line

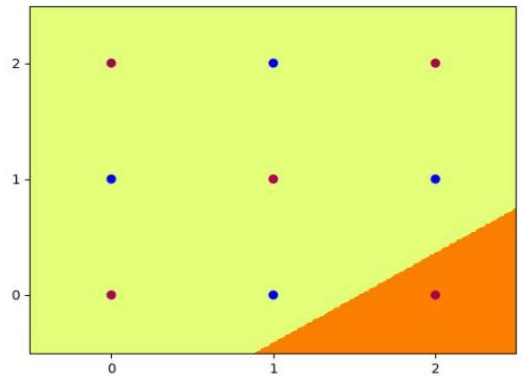
Hidden Node	Equation of Dividing Line
h_1	$x-y=1.36 \rightarrow y= x-1.36$
h_2	$x-y=0.45 \rightarrow y= x-0.45$
h_3	$x-y=-0.45 \rightarrow y= x+0.45$
h_4	$x-y=-1.36 \rightarrow y= x+1.36$

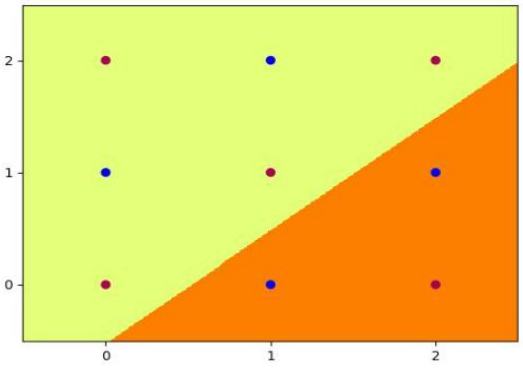
Activation Table

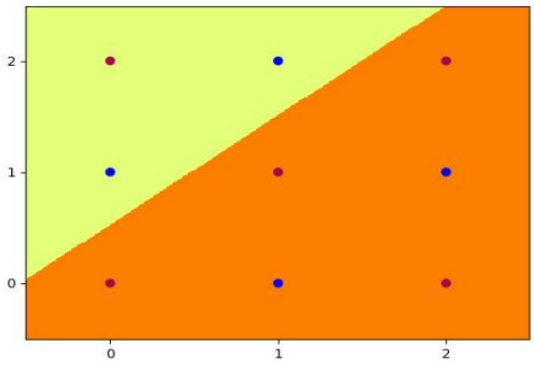
x	y	h_1	h_2	h_3	h_4	Output
0	0	0	0	1	1	1
0	1	0	0	0	1	0
0	2	0	0	0	0	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	2	0	0	0	1	0
2	0	1	1	1	1	1
2	1	1	1	1	1	1
2	2	0	1	1	1	1

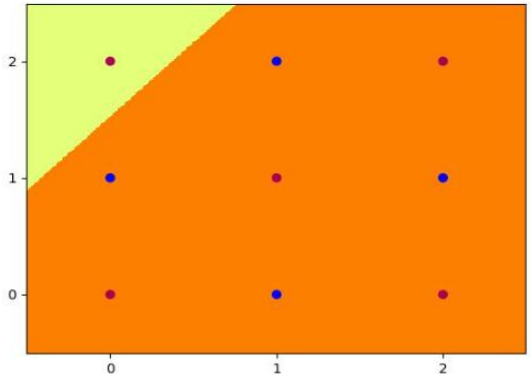
The hidden node divides the input space into diagonal bands through the step activation function. Each hidden node responds to a linear boundary of the form x -y=c, thus being activated only in a specific area. The four hidden units jointly divide the data into mutually exclusive regions, where positive and negative classes can be distinguished. The output node combines the weighted summation with another step function to integrate these activation results into the final binary decision. This network structure achieves perfect classification of 9 training samples by combining multiple step-based linear partitions.

2.3 hid_4_0.jpg、hid_4_1.jpg、hid_4_2.jpg、hid_4_3.jpg are shown below

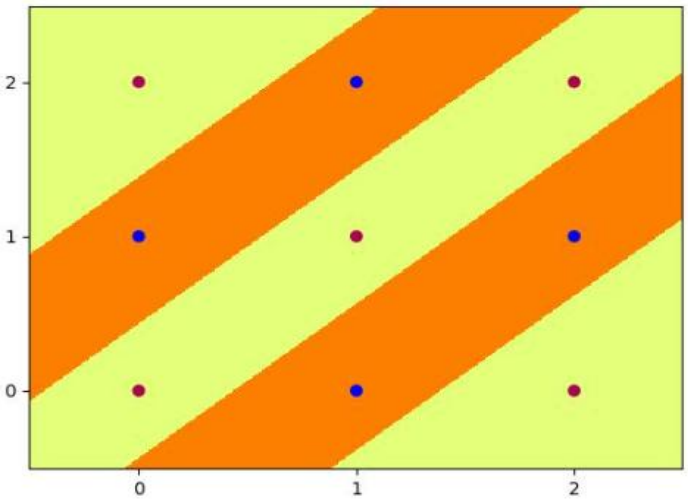








out_4.jpg is shown below

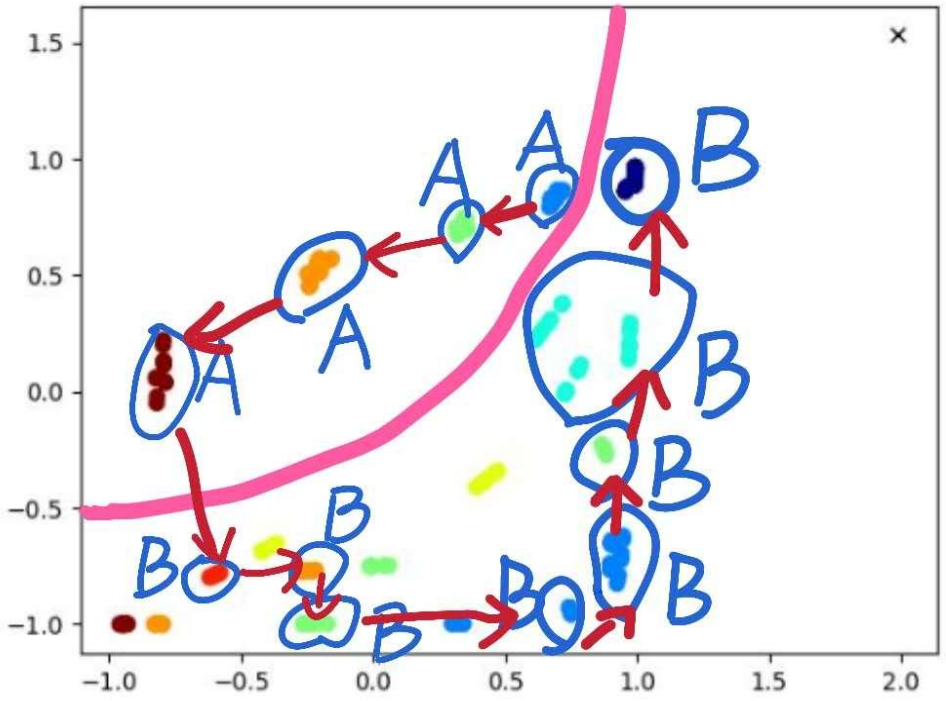


Explanation: The re-scaled weights enable the sigmoid network to approximate the behavior of the stepping network by generating sharp transitions between active states. By multiplying the ownership weight and bias by 10, the response of the sigmoid activation function to the input almost becomes binary. Therefore, the Sigmoid-based network successfully replicated the decision boundaries created by the step function network and achieved a 100% classification accuracy on the training data.

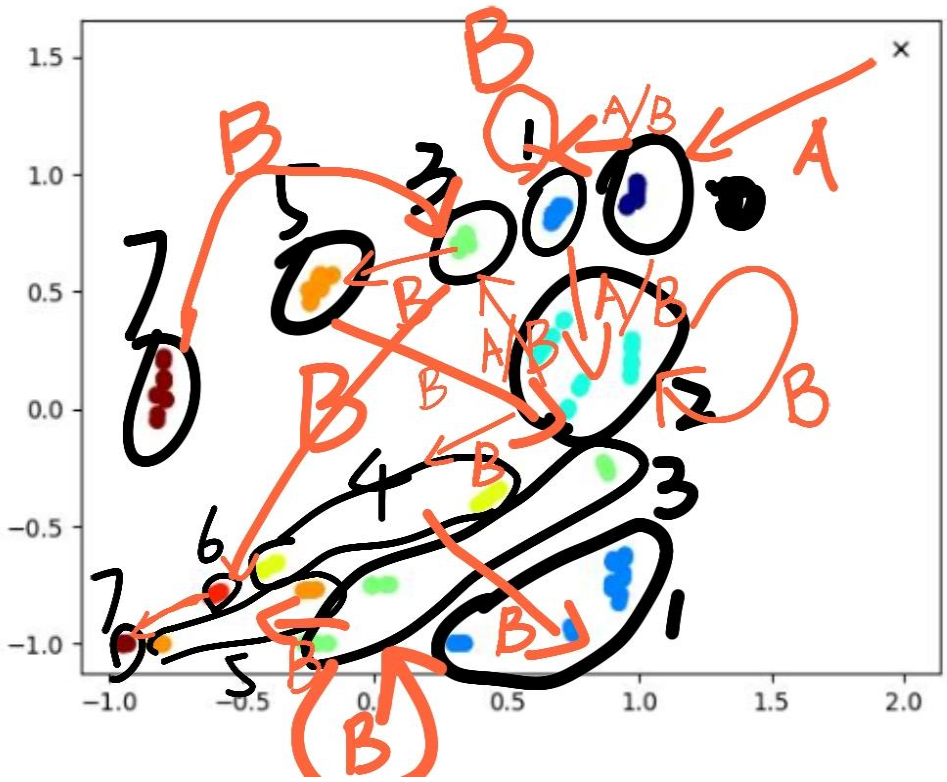
Part 3: Hidden Unit Dynamics for Recurrent Networks

3.1 Annotated image

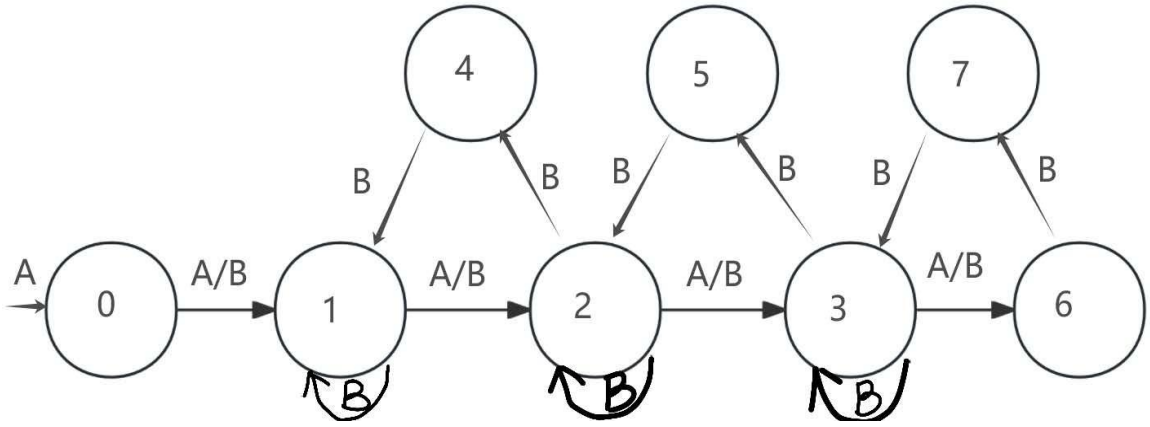
The first type



The second type



3.2 Finite state machine



3.3 How the network accomplishes the $a^n b^{2n}$ task

When processing $a^n b^{2n}$ language sequences, the network effectively captures the temporal dependencies between characters by learning how **hidden state (hidden unit activations) changes over time.**

We observe that **before the first B appears**, although the network's prediction probability for A remains high, its **confidence (i.e., probability peak) gradually decreases.** This indicates that the network's hidden states are gradually approaching the critical point for state transition, i.e., **shifting from outputting A to outputting B.**

After the last A is input, the network can accurately predict the arrival of the first B, indicating that the network has **learned how to represent and utilise the 'number of consecutive A inputs.'**

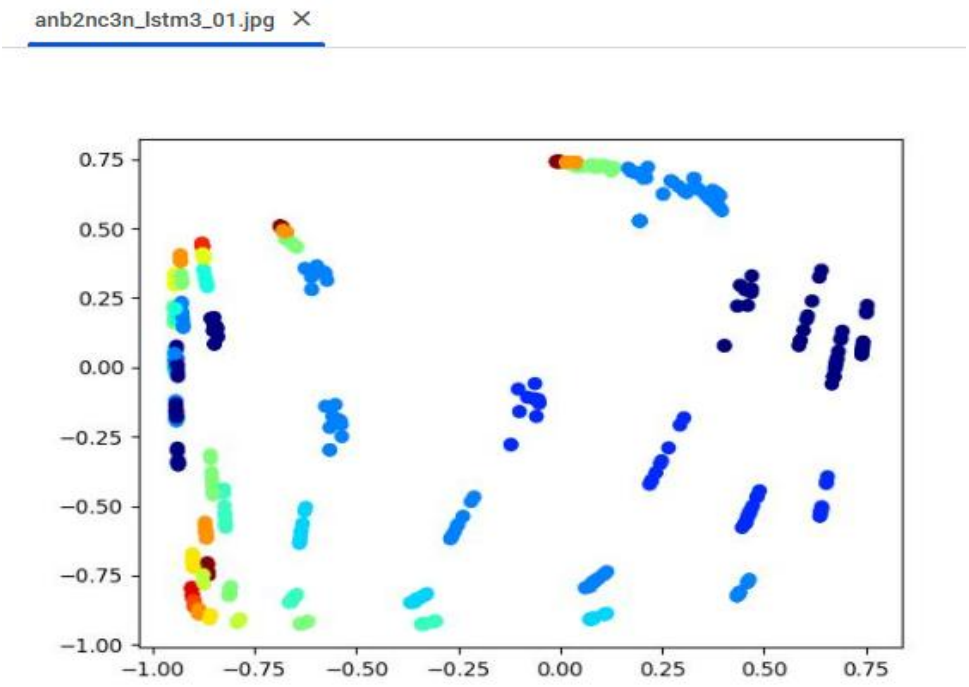
After entering the B phase, the network can accurately and continuously output 2n B, with predictions from the second B onwards being almost deterministic. This indicates that after receiving the first B, the network has **'locked' the previous n values** and uses this memory to control the number of B outputs, demonstrating a **certain 'counting' ability.**

Additionally, after outputting all B, the network can accurately **predict the next A** at the beginning of the sequence, **indicating that its internal state can identify the sequence endpoint and reset the state.**

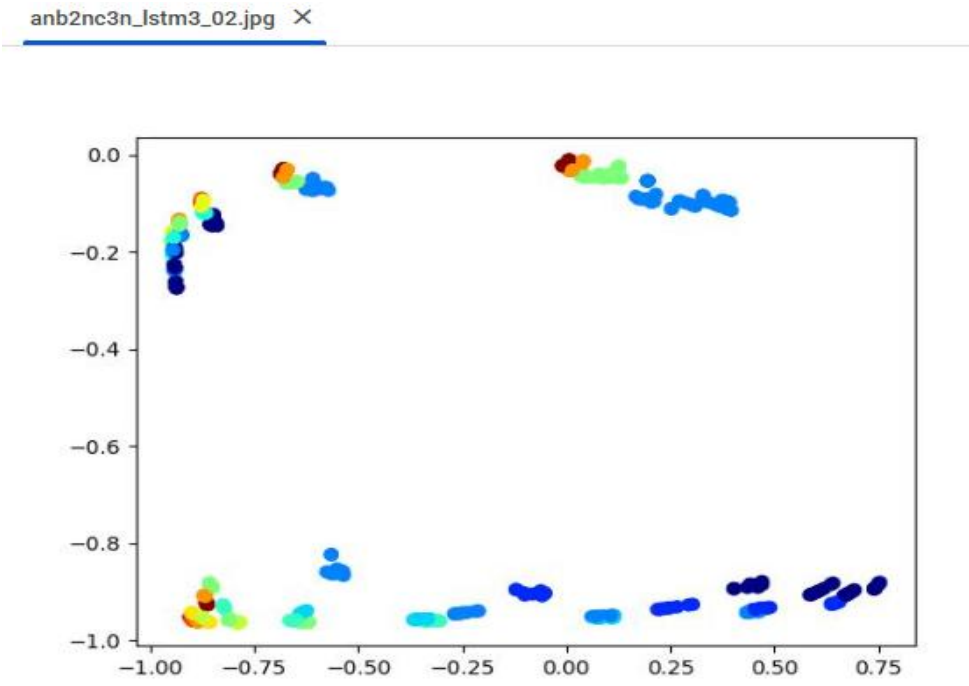
Overall, the SRN model implicitly remembers the number of As in the input through continuously evolving hidden states and generates the correct number of B based on this information, demonstrating certain temporal modelling and memory capabilities.

3.4 Four images

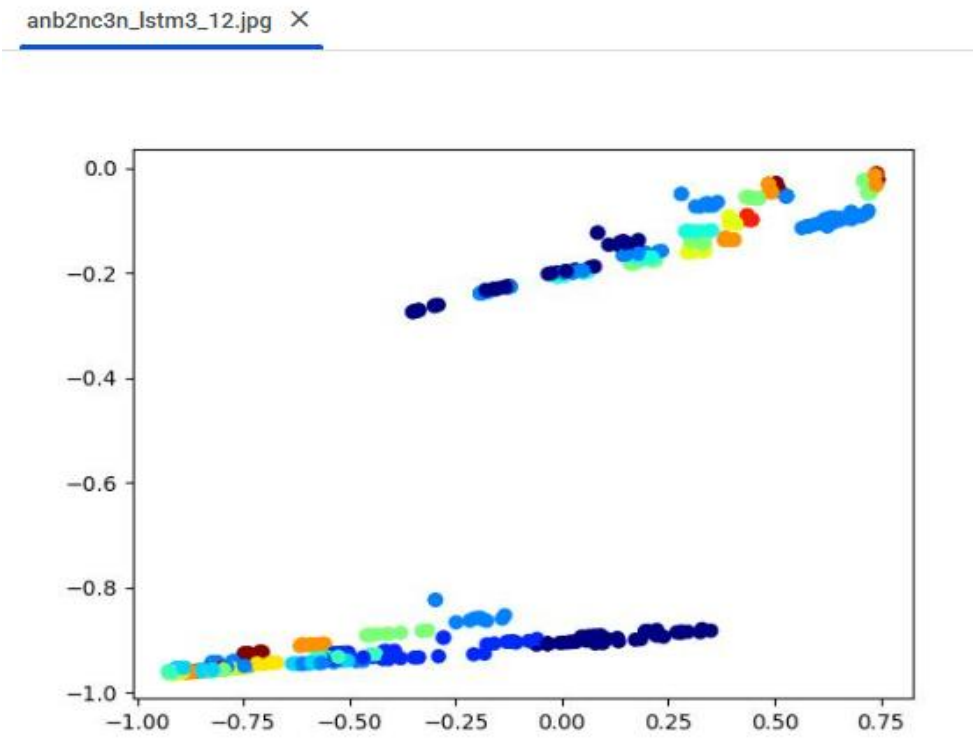
anb2nc3n_lstm3_01



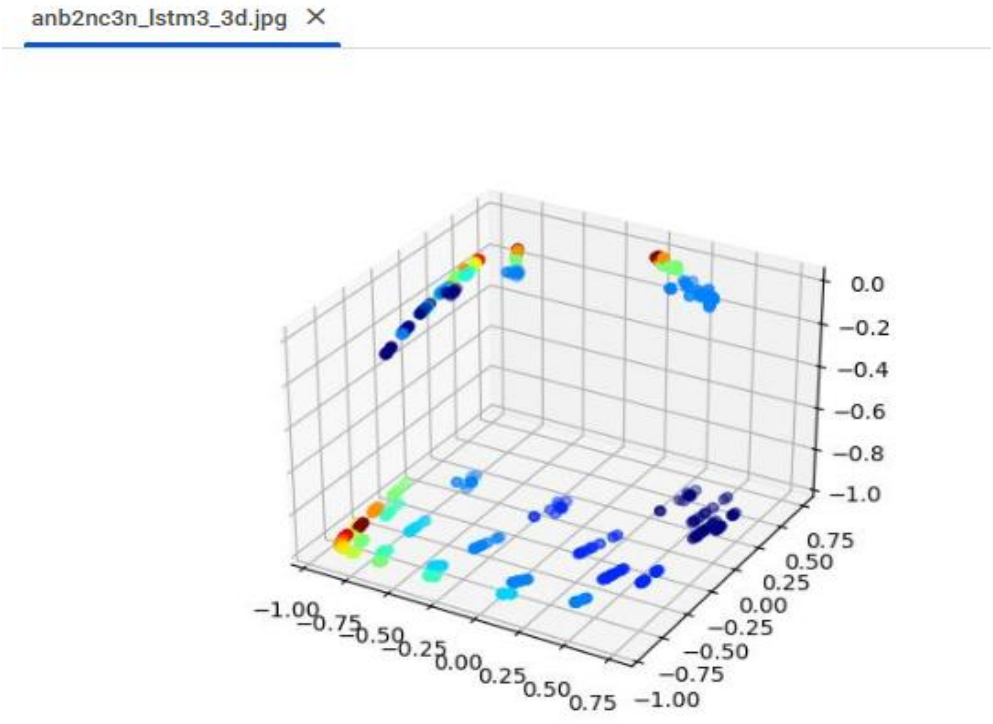
anb2nc3n_lstm3_02



anb2nc3n_lstm3_12



3D image



3.5 How LSTM successfully completes prediction tasks

a) analyse the dynamics of the hidden and/or context units

In $a^n b^{2n} c^{3n}$, the input sequence consists of three parts: n consecutive A, followed by 2n B, and finally 3n C. To accurately complete the prediction task, the network needs

to remember the value of n to determine when to switch from A to B and then to C, and predict the next character category.

Compared with ordinary RNN, LSTM can retain useful information in the sequence for a long time by introducing context cells (cell state), and is particularly suitable for handling such tasks with strict counting and boundary switching requirements.

Dynamic Behavior analysis

1.When dealing with stage A: LSTM accumulates state changes by using the context unit. Each time an "A" is read in, the value of the context unit gradually increases, similar to the current "counting" which "A" it is. The trajectory of the hidden unit activations will slowly advance within a certain area in 3D space.

2.After entering stage B: When LSTM recognizes the end of segment A, the hidden/state state switches.The "memory value" in the context unit no longer increases but is used to determine the output quantity of B. Hidden activations suddenly jump in space, forming another set of clustering trajectories, which usually become smooth or oscillating, indicating that they are in the "middle segment".

3.When entering stage C: LSTM, based on the previous state retention, determines that the current stage is C and switches to output prediction.The context value participates in state adjustment again, and the hidden unit jumps to a new area.The third group of significantly different state trajectories can be observed in the 3D image.

4.Predict the next A (new sequence begins): LSTM reset or jump back to the initial state;The area near the initial state in the figure is usually marked with an '×', which is also the starting point of the next sequence.

b) how the LSTM successfully accomplishes the $a^n b^{2n} c^{3n}$ prediction task

First of all, both RNN and LSTM belong to recurrent neural networks, **but RNN has a fatal problem - vanishing gradient and exploding gradient.** In simple terms, RNN can only "remember" short-distance patterns. When encountering long-distance dependencies (such as anb2nc3n, which are particularly far apart from each other), its hidden state will gradually "forget" the input information at the very beginning during the propagation process. For instance, if you count a few a's earlier, when it comes to the final c, the **RNN has actually forgotten the A's earlier, and thus cannot determine exactly how many c's there should be.**

However, the original intention of **LSTM (Long Short-Term Memory) design was to solve the memory problem of RNNS. In LSTM, there is a dedicated cell state, like an "information superhighway", along with three gate control mechanisms: input gate, forget gate, and output gate,** which specifically control the preservation, forgetting, and output of information. **The information of the cell state can be retained all the way** and will not be overwritten by irrelevant information in the middle.

Specifically for $a^n b^{2n} c^{3n}$: The hidden state of an RNN attenuates information at each step (especially when n is very large), **and it cannot precisely "remember" the number of each segment a, b, and c. Therefore, errors are prone to occur during b2n and c3n.**

LSTM is different. **The cell state of LSTM can explicitly "remember" the quantity of a and the quantity of b, and then selectively "forget" or "retain" the previous information through the gating mechanism.** When LSTM inputs c, it can also **recall through the cell state how many a and b there are in front of it, thereby correctly outputting the corresponding number of c.**

To sum up, LSTM can perform well in information transmission tasks such as anb2nc3n that require long-distance dependencies mainly because its cell state can retain key information instead of being constantly "diluted" like RNN. So when performing such grammatical structure prediction tasks, LSTM can successfully complete them.