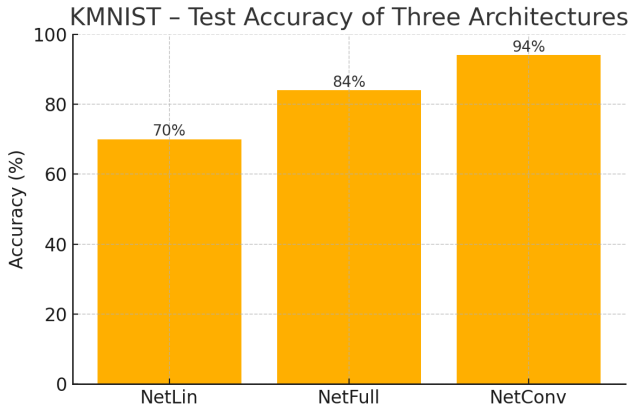
Part1

(a) the relative accuracy of the three models

The Convolutional network (**Conv**) achieved the highest test accuracy (94%), outperforming both the Linear (**Lin**, 70%) and Fully connected (**Full**, 84%) networks. This indicates that the convolutional architecture, which captures spatial information effectively, is significantly more suitable for image classification tasks on the KMNIST dataset compared to simpler linear or fully connected models.



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

Linear model (lin): 7,850 parameters

Fully connected model (full): 101,770 parameters

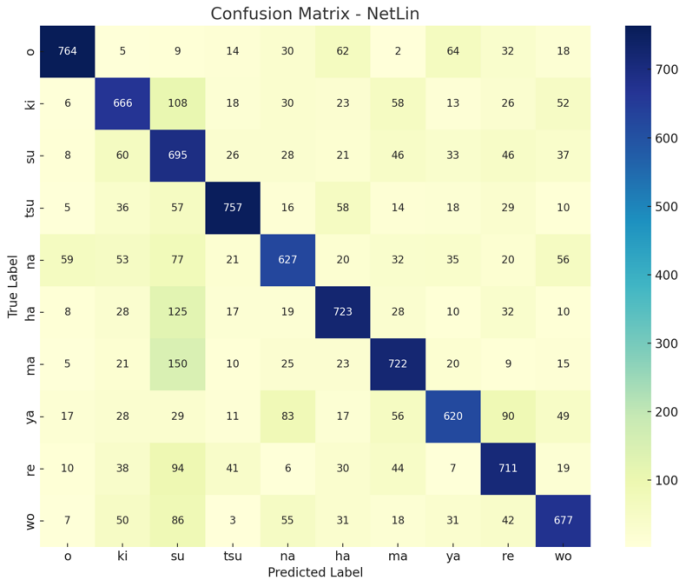
Convolutional model (conv): 421,642 parameters

The Conv model has the highest number of parameters (421,642), significantly exceeding the Lin (7,850) and Full (101,770) models. Although Conv has the most parameters, the increased complexity translates directly into improved performance, justifying the higher computational cost. Conversely, the simplest Lin model with the fewest parameters shows the lowest accuracy.

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

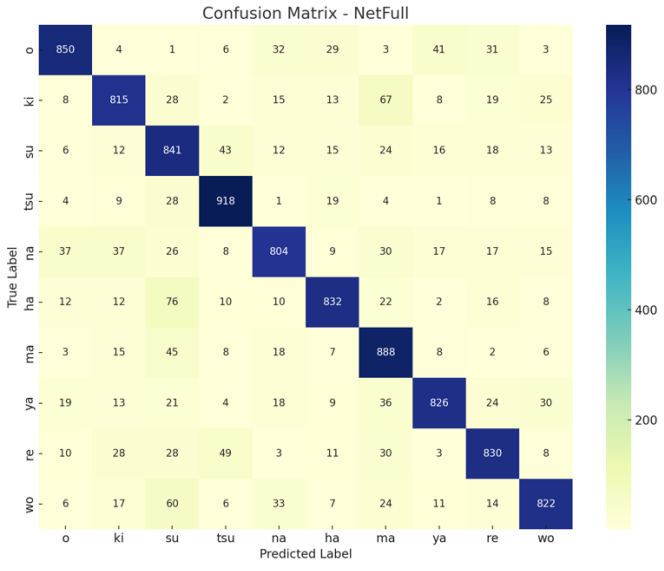
NetLin:

In the NetLin model, '**ma**', '**ha**', and '**ki**' are often mistaken for '**su**'. These characters share similar curved or looped strokes, which a simple linear model struggles to distinguish. Without the ability to capture spatial patterns, the model treats these visually similar shapes as the same. This leads to frequent confusion, especially when the handwriting is unclear. Although '**ki**' and '**ma**' are not the highest, they still have a very high error rate.



NetFull:

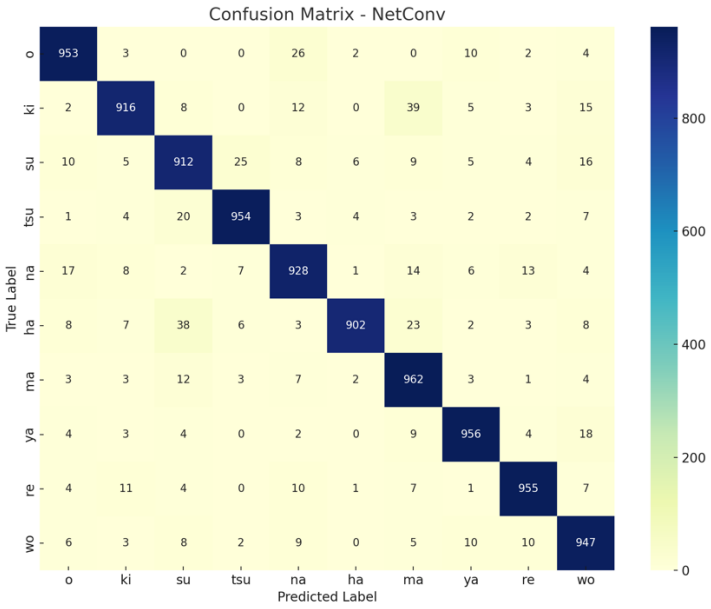
In the NetFull model, we observe that **'ha'** and **'wo'** are still frequently misclassified as **'su'**, though with reduced frequency compared to the linear model. This is likely because these characters share similar sweeping curves and loop structures, which can confuse a fully connected architecture lacking spatial awareness. Additionally, **'ki'** is still often mistaken for **'ma'**, as both contain multiple short horizontal lines stacked in a similar fashion. While the overall confusion has decreased compared to the linear model, these examples highlight how subtle stroke similarities continue to challenge the model's ability to distinguish characters accurately.



NetConv:

In the NetConv model, two of the few remaining confusions include **'ha'** being misclassified as '**su**' and '**ki**' as '**ma**'. These errors are likely due to shared structural features, '**ha**' and '**su**' both contain curved strokes that can appear similar in handwritten form, especially when

written hastily or imprecisely. Likewise, '**ki**' and '**ma**' share horizontal line patterns, which may still confuse the model in edge cases. Nonetheless, compared to the linear and fully connected models, these mistakes are rare, indicating that the convolutional architecture has effectively learned to extract and utilize spatial features for better discrimination.

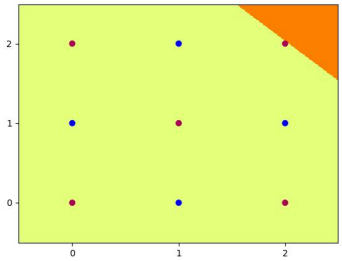


Three models:

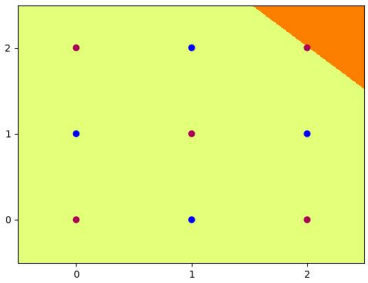
Across all three models, '**ha**' is frequently misclassified as '**su**'. This is likely because the right-side component of '**ha**' closely resembles the shape of '**su**', especially when handwritten with slight variations. Similarly, '**ki**' is often mistaken for '**ma**', as their overall structures are quite alike. Both characters share similar upper portions, and the only distinguishing feature lies in the lower strokes — a difference that simple models may fail to capture. These confusions persist even in more advanced models like NetConv, although the overall misclassification rate is reduced.

Part2

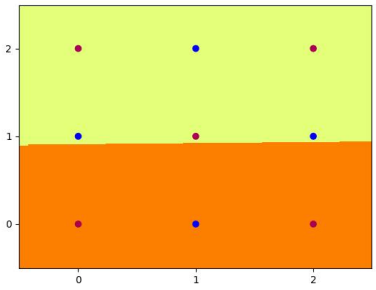
2.1



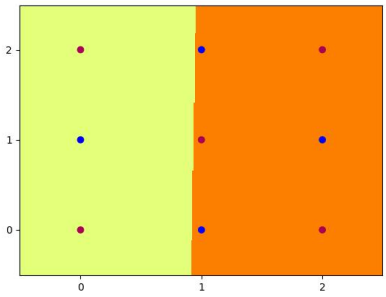
hid_5_0.jpg



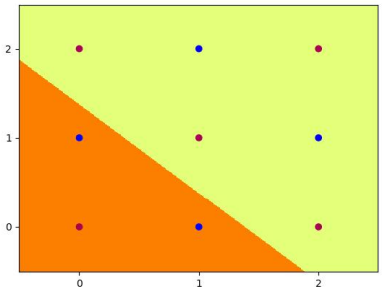
hid_5_1.jpg



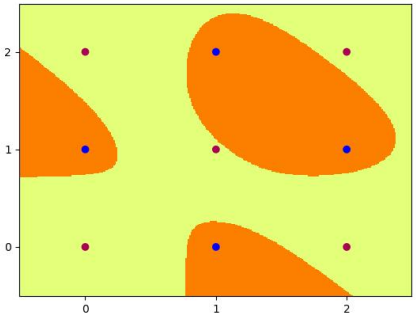
hid_5_2.jpg



hid_5_3.jpg



hid_5_4.jpg



out_5.jpg

```
Final Weights:
tensor([[-6.3665,  5.0988],
        [-0.0380,  4.5473],
        [-5.8864,  7.9285],
        [-5.3448,  0.3483],
        [-6.4586, -6.3601]])
tensor([-7.9172, -4.0172,  1.0098,  5.1912,  1.3715])
tensor([[-7.6434,  6.1671, -7.3206,  7.4092, -6.0049]])
tensor([-1.7245])
Final Accuracy:  100.0
```

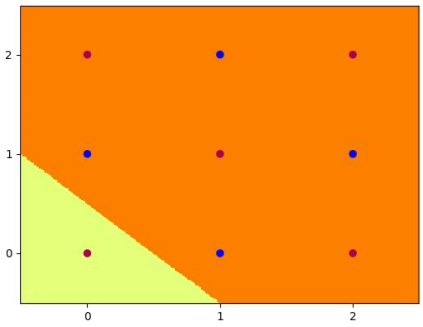
2.2

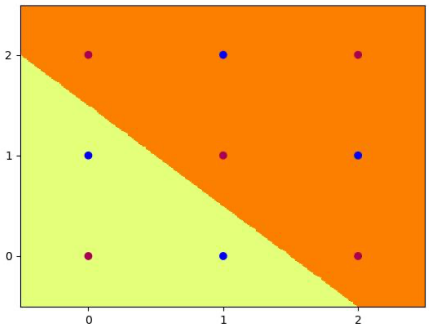
in_hid_weight = [[1,1],[1,1],[1,1],[1,1]]

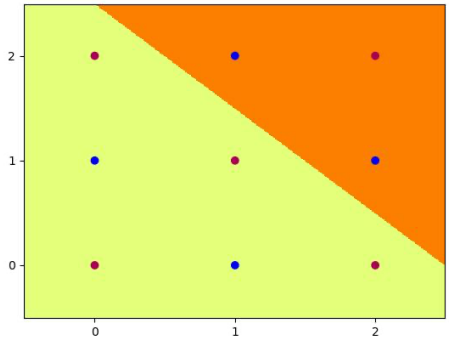
hid_bias = [-0.5, -1.5, -2.5, -3.5]

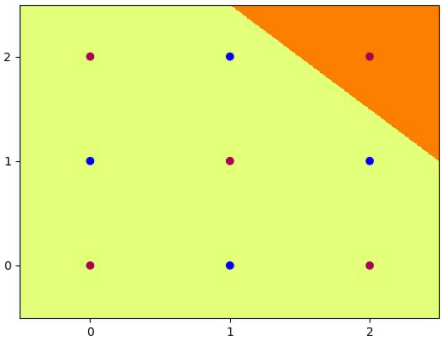
hid_out_weight = [[1, -1, 1, -1]]

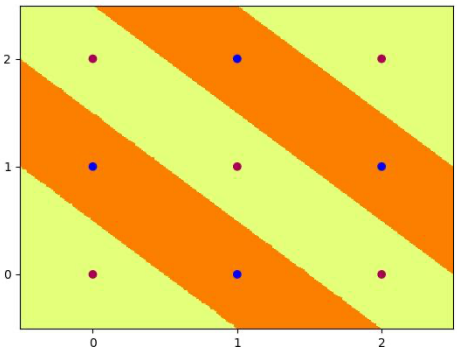
out_bias = [-0.5]



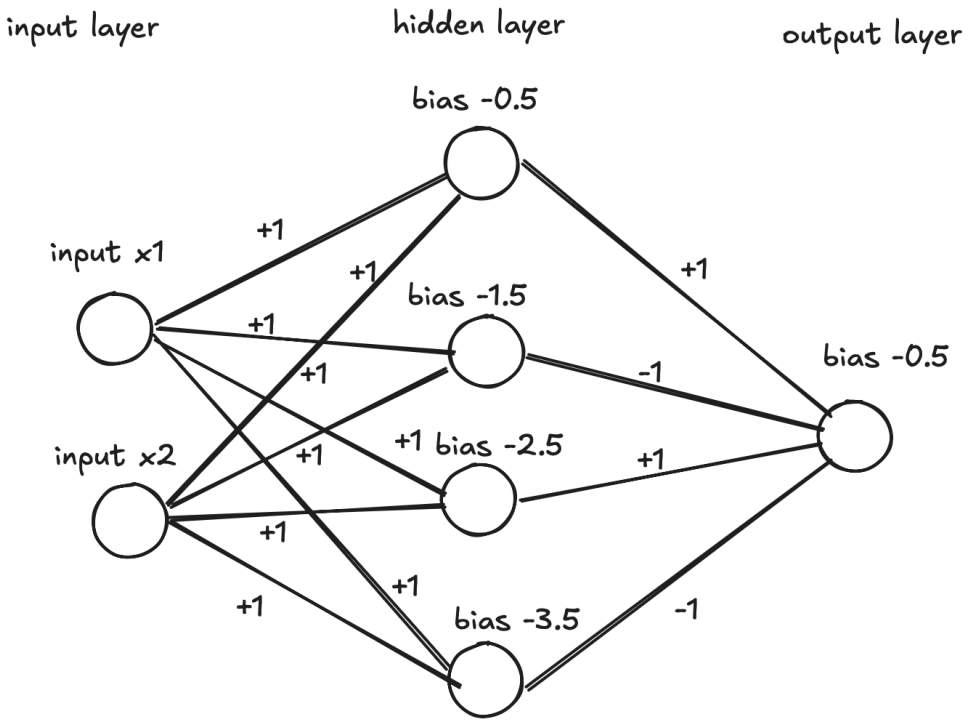








```
Initial Weights:
tensor([[1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.]])
tensor([-0.5000, -1.5000, -2.5000, -3.5000])
tensor([[ 1., -1.,  1., -1.]])
tensor([-0.5000])
Initial Accuracy:  100.0
```



Equations:

$x + y = 0.5$

$x + y = 1.5$

$x + y = 2.5$

$x + y = 3.5$

Activation table for the 9 training points

(x , y)	Target T	H_1	H_2	H_3	H_4	Output
0 , 0	0	0	0	0	0	**0**
0 , 1	1	1	0	0	0	**1**
0 , 2	0	1	1	0	0	**0**
1 , 0	1	1	0	0	0	**1**
1 , 1	0	1	1	0	0	**0**
1 , 2	1	1	1	1	0	**1**
2 , 0	0	1	1	0	0	**0**
2 , 1	1	1	1	1	0	**1**
2 , 2	0	1	1	1	1	**0**

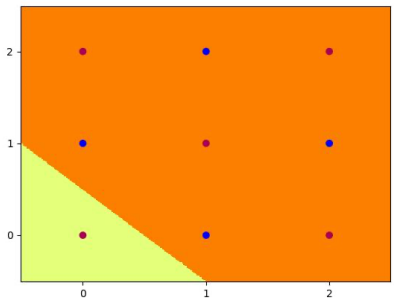
2.3

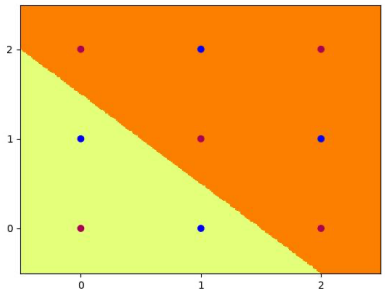
in_hid_weight = [[10,10],[10,10],[10,10],[10,10]]

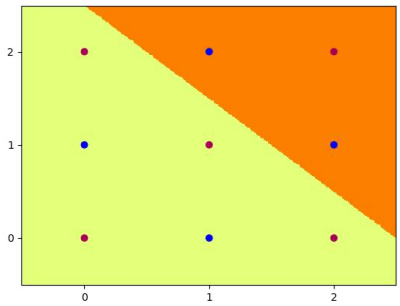
hid_bias = [-5, -15, -25, -35]

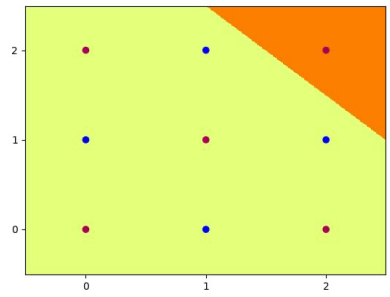
hid_out_weight = [[10, -10, 10, -10]]

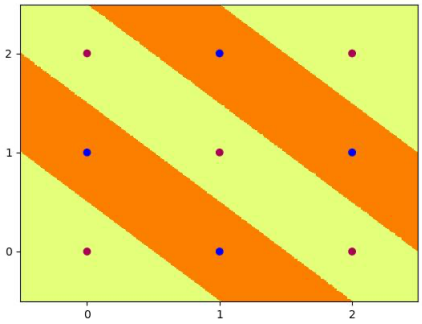
out_bias = [-5]









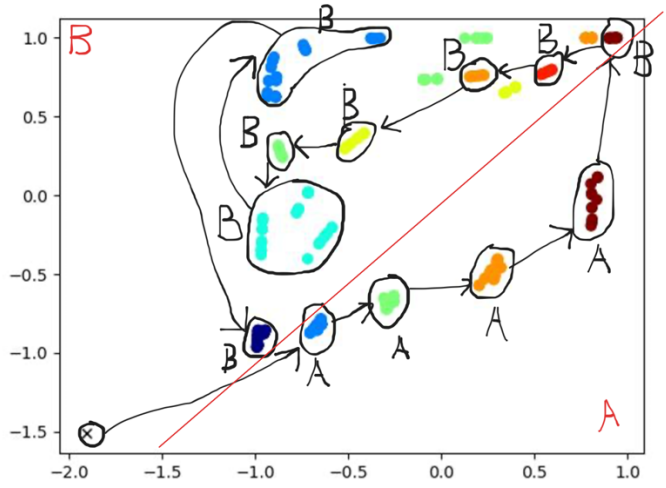



```
Initial Weights:
tensor([[10., 10.],
        [10., 10.],
        [10., 10.],
        [10., 10.]])
tensor([ -5., -15., -25., -35.])
tensor([[ 10., -10.,  10., -10.]])
tensor([-5.])
Initial Accuracy:  100.0
```

Part3

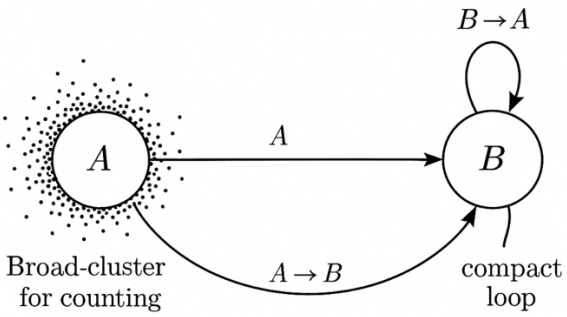
3.1

```
symbol= AABBBBABBABBAAAABBBBBBBBABBA
hidden activations and output probabilities:
A [-0.65 -0.82] [0.81 0.19]
A [-0.26 -0.67] [0.66 0.34]
B [0.24 1.  ] [0. 1.]
B [-0.59 -0.2 ] [0.06 0.94]
B [-0.73  0.92] [0. 1.]
B [-0.99 -0.95] [0.9 0.1]
A [-0.65 -0.8 ] [0.79 0.21]
B [-0.36  1.  ] [0. 1.]
B [-0.95 -0.87] [0.84 0.16]
A [-0.66 -0.85] [0.84 0.16]
B [-0.33  1.  ] [0. 1.]
B [-0.95 -0.86] [0.83 0.17]
A [-0.67 -0.85] [0.85 0.15]
A [-0.26 -0.64] [0.6 0.4]
A [ 0.28 -0.53] [0.49 0.51]
A [0.84 0.11] [0.01 0.99]
B [0.9 1. ] [0. 1.]
B [0.54 0.77] [0. 1.]
B [0.16 0.75] [0. 1.]
B [-0.51  0.31] [0. 1.]
B [-0.88  0.31] [0. 1.]
B [-0.97 -0.35] [0.12 0.88]
B [-0.91  0.86] [0. 1.]
B [-0.99 -0.96] [0.91 0.09]
A [-0.65 -0.79] [0.78 0.22]
B [-0.37  1.  ] [0. 1.]
B [-0.95 -0.87] [0.84 0.16]
epoch: 9
 loss: 0.0330
```

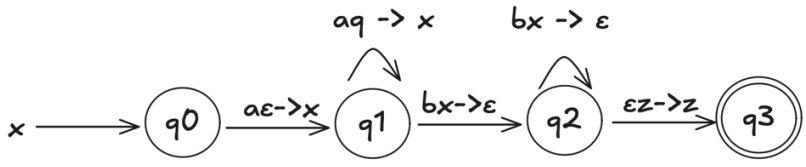


We divided it into two parts, A and B, based on all the points.

3.2



Perhaps we can express it more clearly using a pushdown automaton（PDA）.



States:

q_0: initial state

q_1: read **a** and push it onto the stack

q_2: read **b** and push it onto the stack

q_3: acceptance state

Stack operation:

x: Indicate the marking symbol

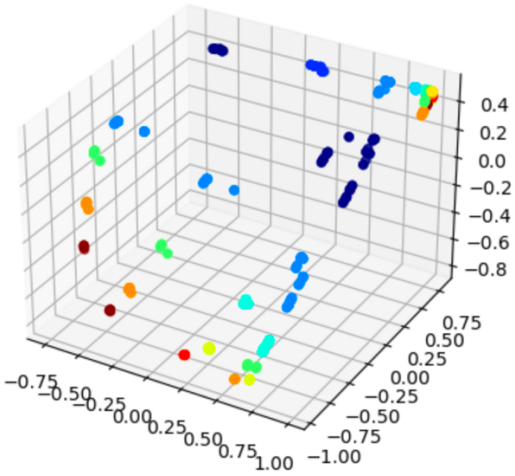
z: represents the bottom symbol of the stack

ε: indicate emptiness

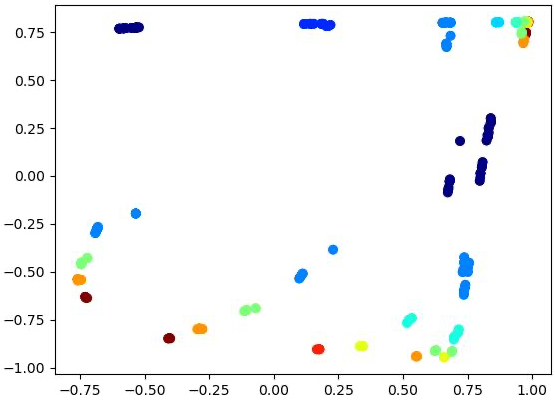
3.3

Before the first B appears, the model's output shows a clear trend: the probability of A steadily decreases while the probability of B rises. This means the network has learned that the longer the run of A's, the more likely a B is imminent, capturing the temporal dependency that defines the language. As soon as the first B is encountered, the hidden state jumps into a distinct "B loop," effectively locking in the previously observed count n. From there, the state machine advances deterministically for the remaining 2n–1 B's, then moves back to the initial region in readiness to predict the next leading A. These dynamics demonstrate that the SRN has internalized both (i) the growing uncertainty of continued A's, and (ii) the precise transition that fixes n once the first B is seen.

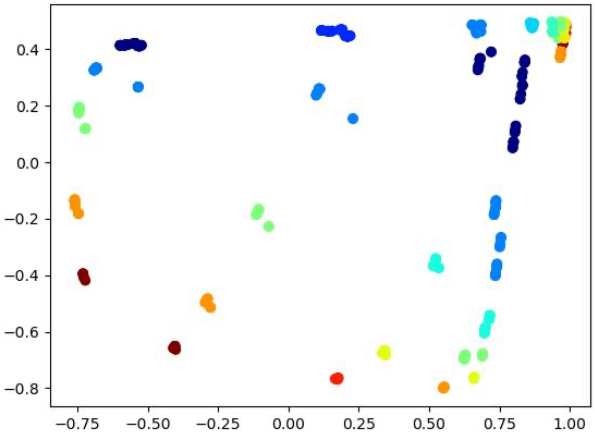
3.4



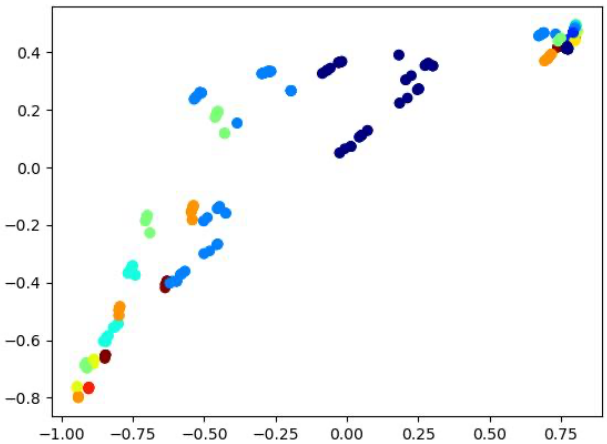
interactive 3D plot



anb2nc3n_lstm3_01.jpg



anb2nc3n_lstm3_02.jpg



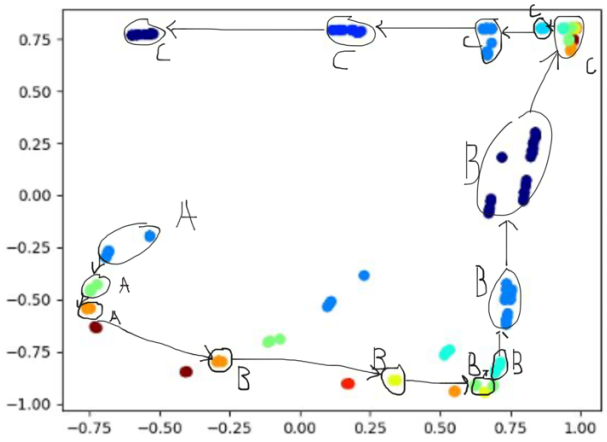
anb2nc3n_lstm3_12.jpg

```
symbol= AAABBBBBBCCCCCCCCCAAABBBBBBCCCCCCCCCAABBBBCCCCCCAABBBBCCCCCCAAABBBBBBCCCCCCCCCA
hidden activations and output probabilities:
A [-0.53 -0.2   0.27] [0.7  0.29 0.01]
A [-0.72 -0.43  0.12] [0.56 0.44 0.  ]
A [-0.75 -0.54 -0.18] [0.31 0.69 0.  ]
B [-0.28 -0.8  -0.51] [0.01 0.99 0.  ]
B [ 0.34 -0.89 -0.68] [0. 1. 0.]
B [ 0.63 -0.91 -0.7 ] [0. 1. 0.]
B [ 0.7  -0.84 -0.6 ] [0. 1. 0.]
B [ 0.74 -0.6  -0.4 ] [0.   0.96 0.04]
B [0.8  0.01 0.07] [0.   0.13 0.86]
C [0.97 0.7  0.38] [0. 0. 1.]
C [0.98 0.8  0.44] [0. 0. 1.]
C [0.98 0.81 0.47] [0. 0. 1.]
C [0.97 0.81 0.48] [0. 0. 1.]
C [0.94 0.8  0.49] [0. 0. 1.]
C [0.87 0.8  0.49] [0. 0. 1.]
C [0.69 0.8  0.49] [0. 0. 1.]
C [0.19 0.79 0.47] [0.06 0.   0.94]
C [-0.54  0.77  0.42] [0.92 0.   0.07]
A [-0.68 -0.27  0.33] [0.77 0.23 0.  ]
A [-0.74 -0.45  0.19] [0.59 0.41 0.  ]
A [-0.76 -0.54 -0.13] [0.35 0.65 0.  ]
B [-0.29 -0.8  -0.49] [0.01 0.99 0.  ]
B [ 0.34 -0.89 -0.67] [0. 1. 0.]
B [ 0.63 -0.91 -0.68] [0. 1. 0.]
B [ 0.7  -0.84 -0.59] [0.   0.99 0.01]
B [ 0.74 -0.58 -0.37] [0.   0.95 0.05]
B [0.81 0.05 0.11] [0.  0.1 0.9]
```

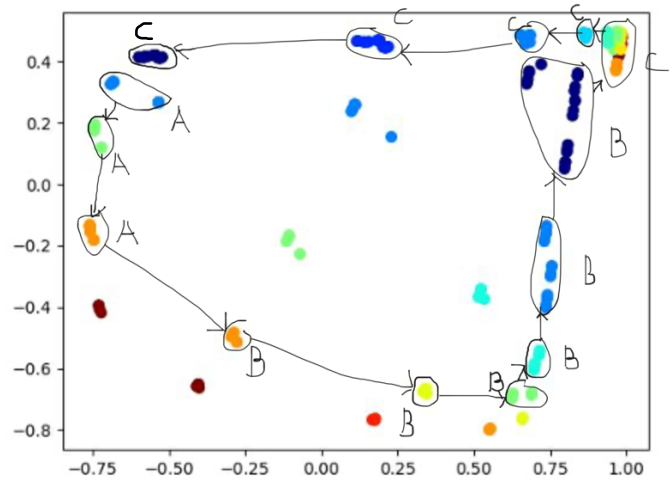
```
C [0.97 0.71 0.39] [0. 0. 1.]
C [0.98 0.8  0.44] [0. 0. 1.]
C [0.98 0.81 0.47] [0. 0. 1.]
C [0.97 0.81 0.48] [0. 0. 1.]
C [0.94 0.8  0.49] [0. 0. 1.]
C [0.87 0.8  0.49] [0. 0. 1.]
C [0.68 0.8  0.49] [0. 0. 1.]
C [0.19 0.79 0.47] [0.06 0.   0.93]
C [-0.55  0.77  0.42] [0.93 0.   0.07]
A [-0.68 -0.28  0.33] [0.77 0.23 0.  ]
A [-0.74 -0.45  0.19] [0.59 0.41 0.  ]
B [-0.11 -0.7  -0.17] [0.02 0.98 0.  ]
B [ 0.52 -0.75 -0.34] [0.   0.99 0.01]
B [ 0.74 -0.45 -0.14] [0.   0.85 0.15]
B [0.84 0.28 0.36] [0.   0.01 0.98]
C [0.96 0.75 0.45] [0. 0. 1.]
C [0.94 0.8  0.47] [0. 0. 1.]
C [0.87 0.8  0.48] [0. 0. 1.]
C [0.66 0.8  0.48] [0. 0. 1.]
C [0.14 0.79 0.46] [0.08 0.   0.91]
C [-0.58  0.77  0.41] [0.94 0.   0.06]
A [-0.69 -0.29  0.33] [0.76 0.24 0.  ]
A [-0.74 -0.46  0.18] [0.58 0.42 0.  ]
B [-0.11 -0.71 -0.19] [0.02 0.98 0.  ]
B [ 0.52 -0.77 -0.37] [0.   0.99 0.01]
B [ 0.73 -0.5  -0.19] [0.  0.9 0.1]
B [0.83 0.21 0.3 ] [0.   0.02 0.97]
C [0.96 0.74 0.44] [0. 0. 1.]
C [0.94 0.8  0.46] [0. 0. 1.]
C [0.87 0.8  0.48] [0. 0. 1.]
C [0.67 0.8  0.48] [0. 0. 1.]
C [0.14 0.79 0.46] [0.08 0.   0.91]
C [-0.58  0.77  0.41] [0.94 0.   0.06]
```

```
A [-0.69 -0.29  0.33] [0.76 0.24 0.  ]
A [-0.74 -0.46  0.18] [0.58 0.42 0.  ]
A [-0.76 -0.54 -0.15] [0.34 0.66 0.  ]
B [-0.3 -0.8 -0.5] [0.01 0.99 0.  ]
B [ 0.33 -0.89 -0.68] [0. 1. 0.]
B [ 0.62 -0.91 -0.69] [0. 1. 0.]
B [ 0.7  -0.85 -0.6 ] [0. 1. 0.]
B [ 0.74 -0.62 -0.4 ] [0.   0.97 0.03]
B [ 0.8  -0.03  0.05] [0.   0.17 0.82]
C [0.97 0.69 0.37] [0. 0. 1.]
C [0.98 0.8  0.44] [0. 0. 1.]
C [0.98 0.81 0.47] [0. 0. 1.]
C [0.97 0.81 0.48] [0. 0. 1.]
C [0.94 0.8  0.49] [0. 0. 1.]
C [0.87 0.8  0.49] [0. 0. 1.]
C [0.68 0.8  0.49] [0. 0. 1.]
C [0.18 0.79 0.47] [0.06 0.   0.93]
C [-0.55  0.77  0.42] [0.93 0.   0.07]
epoch: 9
 loss: 0.0090
```

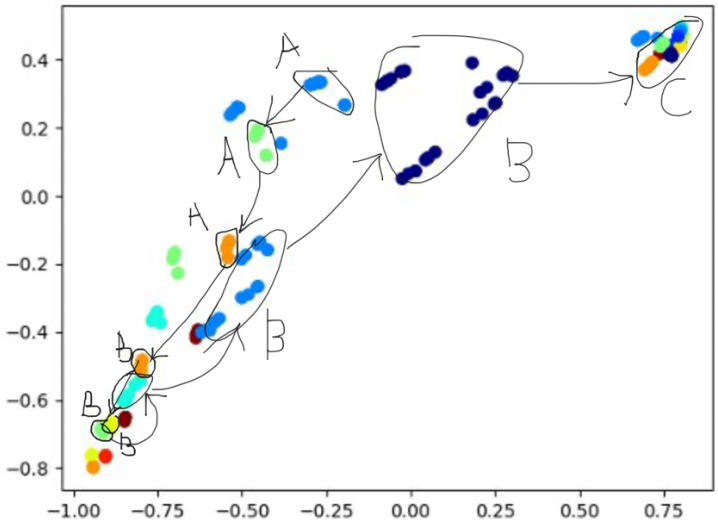
3.5



anb2nc3n_lstm3_01.jpg



anb2nc3n_lstm3_02.jpg



anb2nc3n_lstm3_12.jpg

When processing complex sequence patterns like anb^2nc^3n, standard **RNNs** often struggle due to their reliance on a single hidden state that gets continuously overwritten at each timestep. This makes it difficult to retain early information (such as the number of A's) until much later in the sequence, especially during the prediction of the C's, where long-range dependencies are required. Furthermore, training standard RNNs involves backpropagation through time, where the repeated application of the chain rule can lead to vanishing or exploding gradients. As a result, early inputs like the number of A's are easily lost or distorted by the time the model reaches the C-phase.

In contrast, the **LSTM** overcomes these limitations by introducing a dedicated **memory cell** and a gating mechanism consisting of the **forget gate**, **input gate**, and **output gate**. These gates enable the network to selectively retain relevant past information, update memory when necessary, and control what is exposed to the output at each step. During the A-phase, the LSTM accumulates a memory trace representing the count n. In the B-phase, it uses this memory to output exactly $2n$ B's, and in the C-phase, it continues to rely on the same stored value to generate $3n$ C's. The transitions between phases are handled by smooth changes in internal activations without overwriting or resetting the cell state.

These dynamics—preservation of memory across long spans, controlled updates, and output regulation—demonstrate that the LSTM has successfully internalized the counting and switching behavior required by the grammar. Its ability to overcome vanishing gradients and maintain structured memory is precisely why it can model anb^2nc^3n correctly, while a standard RNN typically fails.