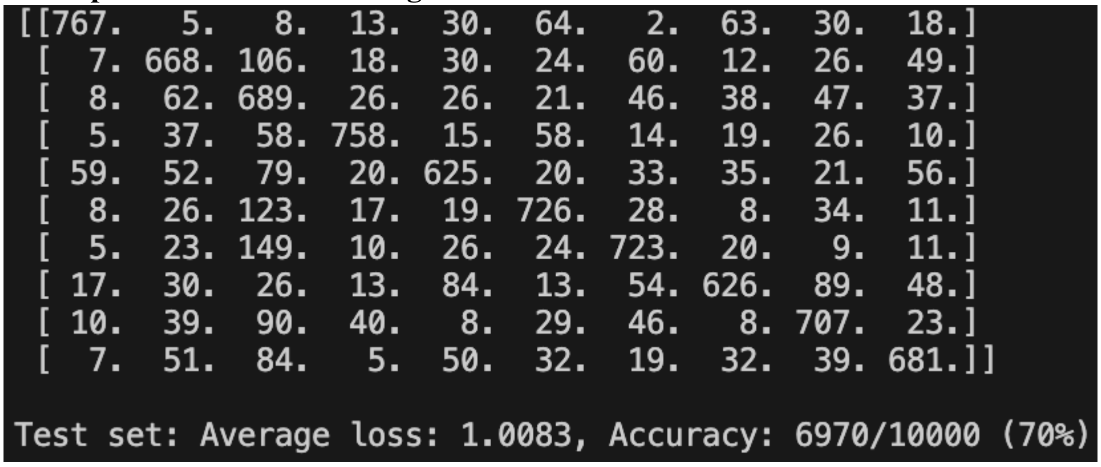
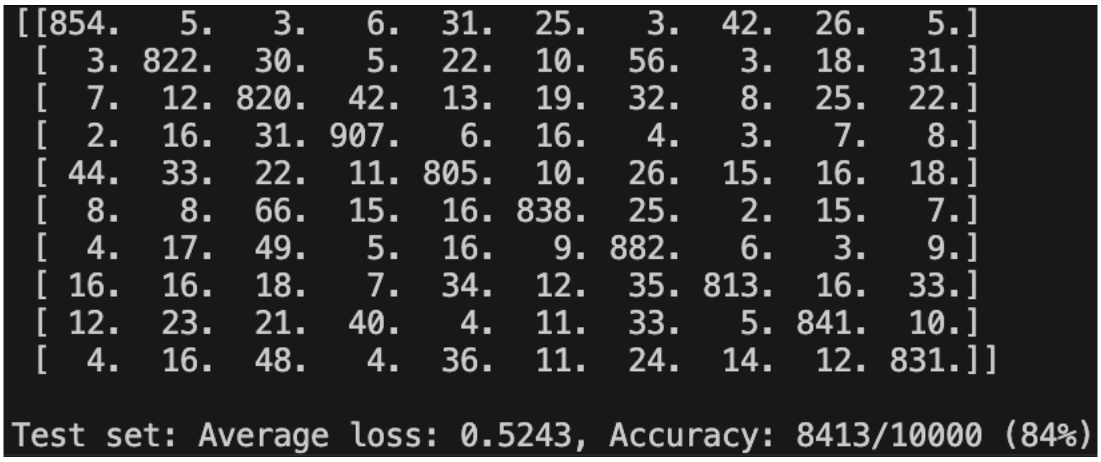
Part 1: Japanese Character Recognition

1. 

```
[[767.   5.   8.  13.  30.  64.   2.  63.  30.  18.]
 [  7. 668. 106.  18.  30.  24.  60.  12.  26.  49.]
 [  8.  62. 689.  26.  26.  21.  46.  38.  47.  37.]
 [  5.  37.  58. 758.  15.  58.  14.  19.  26.  10.]
 [ 59.  52.  79.  20. 625.  20.  33.  35.  21.  56.]
 [  8.  26. 123.  17.  19. 726.  28.   8.  34.  11.]
 [  5.  23. 149.  10.  26.  24. 723.  20.   9.  11.]
 [ 17.  30.  26.  13.  84.  13.  54. 626.  89.  48.]
 [ 10.  39.  90.  40.   8.  29.  46.   8. 707.  23.]
 [  7.  51.  84.   5.  50.  32.  19.  32.  39. 681.]]

Test set: Average loss: 1.0083, Accuracy: 6970/10000 (70%)
```

2. 

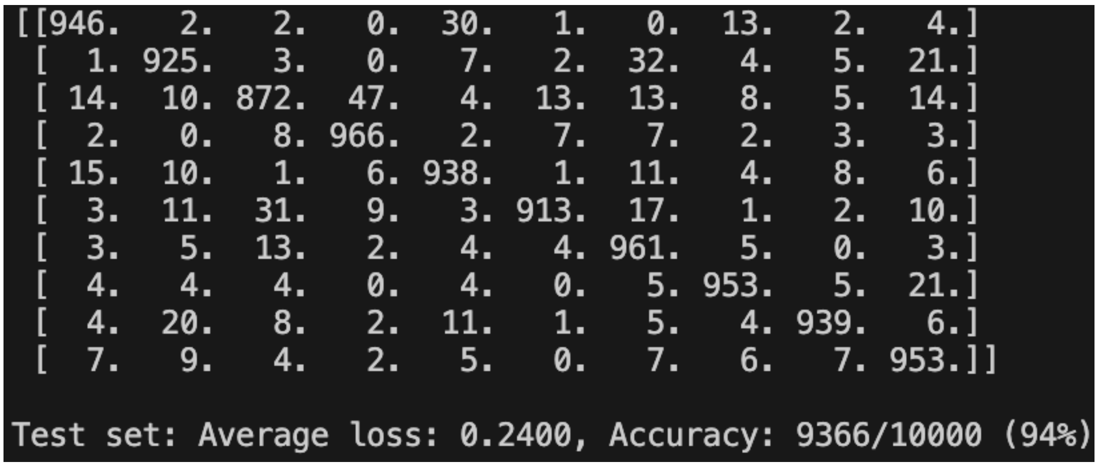
```
[[854.   5.   3.   6.  31.  25.   3.  42.  26.   5.]
 [  3. 822.  30.   5.  22.  10.  56.   3.  18.  31.]
 [  7.  12. 820.  42.  13.  19.  32.   8.  25.  22.]
 [  2.  16.  31. 907.   6.  16.   4.   3.   7.   8.]
 [ 44.  33.  22.  11. 805.  10.  26.  15.  16.  18.]
 [  8.   8.  66.  15.  16. 838.  25.   2.  15.   7.]
 [  4.  17.  49.   5.  16.   9. 882.   6.   3.   9.]
 [ 16.  16.  18.   7.  34.  12.  35. 813.  16.  33.]
 [ 12.  23.  21.  40.   4.  11.  33.   5. 841.  10.]
 [  4.  16.  48.   4.  36.  11.  24.  14.  12. 831.]]

Test set: Average loss: 0.5243, Accuracy: 8413/10000 (84%)
```

hidden_units = 95

- fc1 = nn.Linear(28 * 28, 95)
  Weight parameters: 28 * 28 * 95 + 95 = 74,575
- fc2 = nn.Linear(95, 10)
  Weight parameters: 95 * 10 + 10 = 960

Total parameters = 74,575 + 960 = 75,535

3. 

```
[[946.   2.   2.   0.  30.   1.   0.  13.   2.   4.]
 [  1. 925.   3.   0.   7.   2.  32.   4.   5.  21.]
 [ 14.  10. 872.  47.   4.  13.  13.   8.   5.  14.]
 [  2.   0.   8. 966.   2.   7.   7.   2.   3.   3.]
 [ 15.  10.   1.   6. 938.   1.  11.   4.   8.   6.]
 [  3.  11.  31.   9.   3. 913.  17.   1.   2.  10.]
 [  3.   5.  13.   2.   4.   4. 961.   5.   0.   3.]
 [  4.   4.   4.   0.   4.   0.   5. 953.   5.  21.]
 [  4.  20.   8.   2.  11.   1.   5.   4. 939.   6.]
 [  7.   9.   4.   2.   5.   0.   7.   6.   7. 953.]]

Test set: Average loss: 0.2400, Accuracy: 9366/10000 (94%)
```

- Conv1 layer: in_channels = 1, out_channels = 16, kernel_size = 3 * 3
  parameters = kernel_height * kernel_width * in_channels * out_channels +

out_channels (bias) = 3 * 3 * 1 * 16 + 16 = 160

- Conv2 layer: in_channels = 16, out_channels = 32, kernel_size = 3 * 3
  parameters = kernel_height * kernel_width * in_channels * out_channels + out_channels (bias) = 3 * 3 * 16 * 32 + 32 = 4,640
- FC1: in_features = 32 * 7 * 7, out_features = 128
  parameters = 32 * 7 * 7 * 128 + 128 = 200,832
- FC2: in_features = 128, out_features = 10
  parameters = 128 * 10 + 10 = 1,290

Total parameters = 160 + 4,640 + 200,832 + 1,290 = 206,922

4. Briefly discuss the following points

a. The relative accuracy of the three models

The three models show significant differences in classification performance:

- NetLin: Achieves approximately 70% accuracy. This is the most basic model and lacks the capacity to capture spatial features in images.
- NetFull: Reaches around 84% accuracy. The additional hidden layer allows for more complex decision boundaries, improving generalization.
- NetConv: Attains the highest accuracy of 94%. Its convolutional layers effectively capture local spatial patterns, leading to superior performance in image classification tasks.

Relative ranking in accuracy: NetLin < NetFull < NetConv

b. The number of independent parameters in each of the three models

The number of learnable parameters increases with the complexity of the model:

- NetLin:
  - Weight parameters = 28 * 28 * 10 = 7,840
  - Bias = 10
  - Total = 7,850
- NetFull:
  - First layer: 28 * 28 * 95 + 95 = 74,575
  - Second layer: 95 * 10 + 10 = 960
  - Total = 75,535
- NetConv:
  - Conv1: 3 * 3 * 1 * 16 + 16 = 160
  - Conv2: 3 * 3 * 16 * 32 + 32 = 4,640
  - FC1: 32 * 7 * 7 * 128 + 128 = 200,832
  - FC2: 128 * 10 + 10 = 1,290
  - Total = 206,922

Relative size of models: NetLin < NetFull < NetConv

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

Across all models, digits such as 2, 3, 4, 5, 7, and 9 are most frequently confused with one another. This is primarily due to visual similarity in their handwritten forms. For instance, the digit 2 is often mistaken for 1 and 7, likely because of the curved top and straight diagonal stroke that resemble aspects of both. Similarly, 4 and 9 are commonly

confused due to their overlapping angular structures, especially when written hastily or with stylistic variation. The digits 3 and 5 also exhibit frequent misclassification, as their loops and curves can appear nearly identical when written loosely.
From the confusion matrices of each model:

- NetLin (70% accuracy):

High confusion between visually similar digits, such as 2 and 1, 2 and 7, 3 and 5, 4 and 9.
Examples:

  - 62 instances of class 2 misclassified as 1
  - 38 instances of class 2 misclassified as 7
  - 58 instances of class 3 misclassified as 5
  - 56 instances of class 4 misclassified as 9

Reasons:

  - The model is a simple linear classifier and cannot extract local patterns or spatial features.
  - It lacks the ability to distinguish subtle differences in shapes (e.g., loops, angles, stroke directions).

- NetFull (84% accuracy):

Fewer misclassifications, but confusion still occurs between 2 and 3, 4 and 9, 3 and 5, 9 and 4.
Examples:

  - 42 instances of 2 misclassified as 3
  - 18 instances of 4 misclassified as 9
  - 16 instances of 3 misclassified as 5
  - 36 instances of 9 misclassified as 4

Reasons:

  - The fully connected network captures non-linear abstract patterns, reducing some errors.
  - However, it lacks spatial invariance, which limits its ability to differentiate between digits that share similar layout or components (e.g., curved vs. straight segments).

- NetConv (94% accuracy):

Most confusion is significantly reduced, though minor issues remain confusions often occur between digits with overlapping or ambiguous handwritten forms.
Examples:

  - 47 instances of 2 misclassified as 3
  - 4 instances of 4 misclassified as 7
  - 31 instances of 5 misclassified as 2
  - 5 instances of 9 misclassified as 4

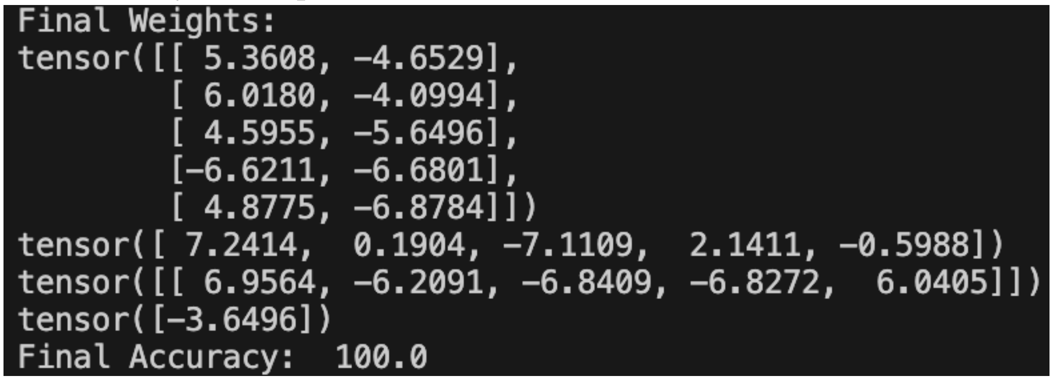
Reasons:

  - This model employs convolutional layers, which are effective at learning spatial hierarchies, local edge features, and translation-invariant patterns.

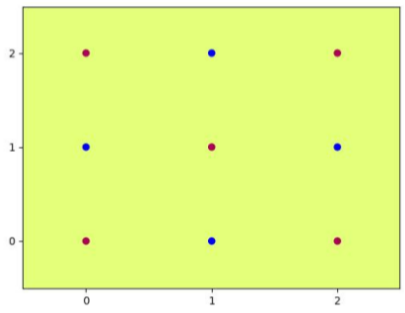
- ⑩ Remaining errors are typically due to poor handwriting quality (e.g., incomplete loops, slanted strokes) and inherent shape overlap between certain digits (e.g., 5 vs. 3, 9 vs. 4).

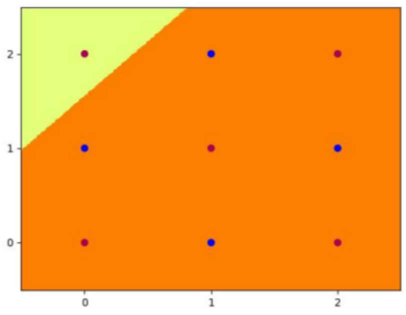
As model complexity and spatial awareness improve, confusion between similar-looking digits decreases. NetConv best resolves ambiguous patterns due to its convolutional feature extraction. However, digits with inherently similar shapes—such as 2 and 3, or 4 and 9—remain prone to confusion, particularly when handwriting is ambiguous or inconsistent. This highlights that while advanced models greatly reduce error, some digit pairs still challenge recognition systems due to visual overlap and writing variability.

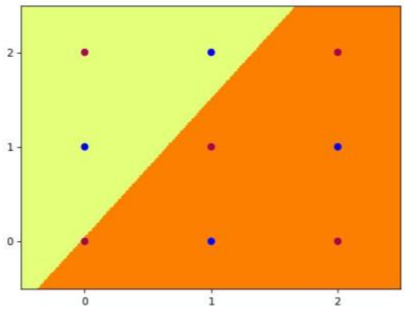
Part 2: Multi-Layer Perceptron

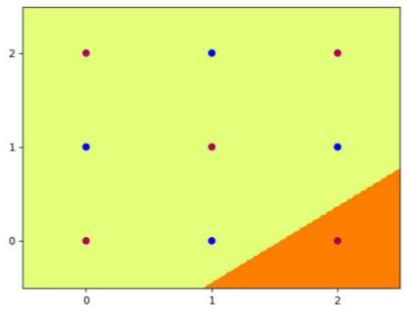
1. 

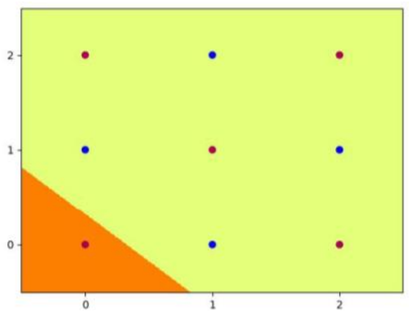
```
Final Weights:
tensor([[ 5.3608, -4.6529],
        [ 6.0180, -4.0994],
        [ 4.5955, -5.6496],
        [-6.6211, -6.6801],
        [ 4.8775, -6.8784]])
tensor([ 7.2414,  0.1904, -7.1109,  2.1411, -0.5988])
tensor([[ 6.9564, -6.2091, -6.8409, -6.8272,  6.0405]])
tensor([-3.6496])
Final Accuracy:  100.0
```


check.jpg

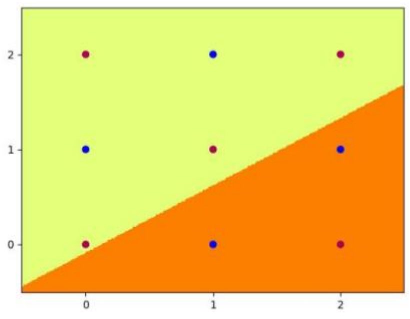

hid_5_0.jpg


hid_5_1.jpg

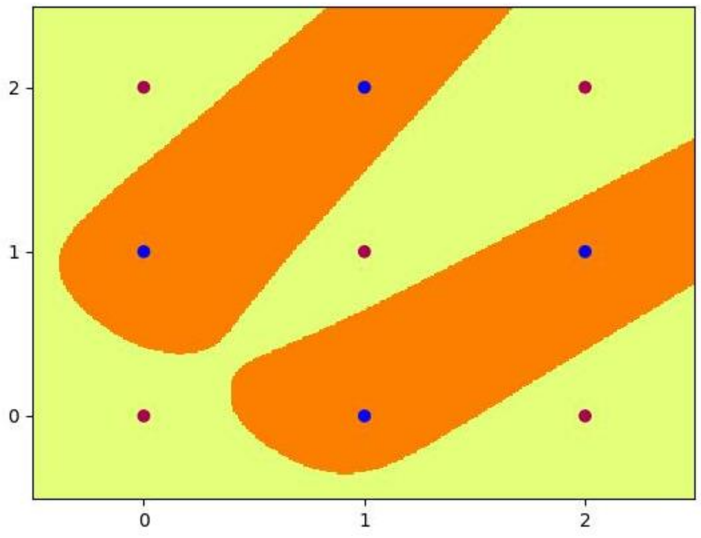

hid_5_2.jpg



hid_5_3.jpg

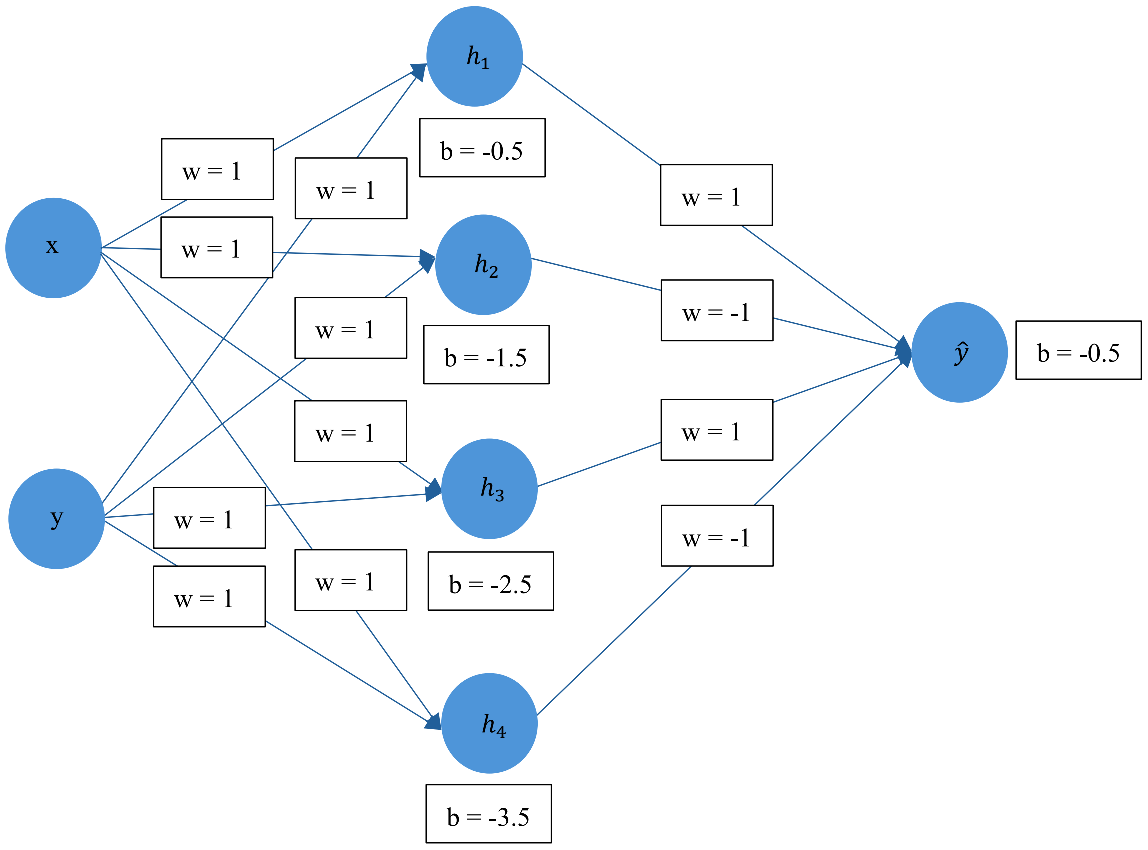


hid_5_4.jpg



out_5.jpg

2.



Each hidden unit h_i implements a linear threshold $\omega_x x + \omega_y y + b_i = 0$ with $\omega_x = \omega_y = 1$. Hence its decision boundary is the line $x + y + b_i = 0$.

Hidden1 (h_1: $b_1 = -0.5$):

$x + y - 0.5 = 0 \Rightarrow y = -x + 0.5$

Hidden2 (h_2: $b_2 = -1.5$):

$x + y - 1.5 = 0 \Rightarrow y = -x + 1.5$

Hidden3 (h_3: $b_3 = -2.5$):

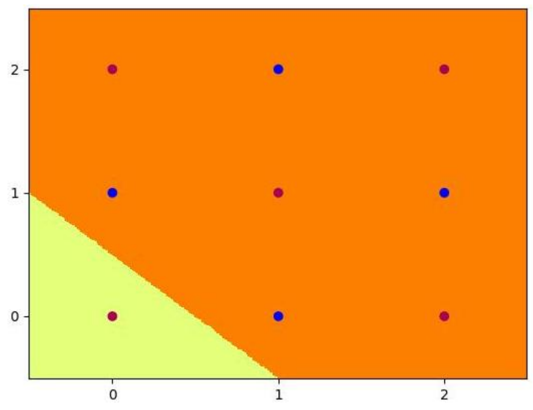
$x + y - 2.5 = 0 \Rightarrow y = -x + 2.5$

Hidden4 (h_4: $b_4 = -3.5$):

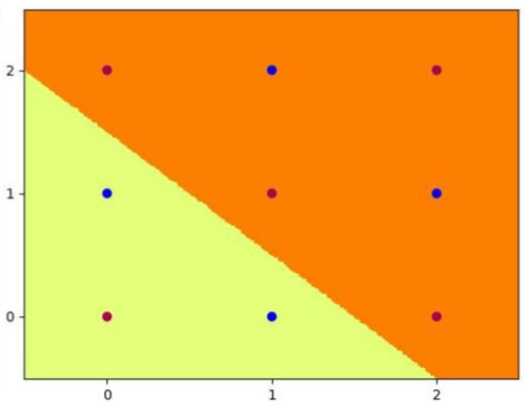
$x + y - 3.5 = 0 \Rightarrow y = -x + 3.5$

#	(x, y)	Hidden1	Hidden2	Hidden3	Hidden4	Output
1	(0, 0)	0	0	0	0	0
2	(0, 1)	1	0	0	0	1
3	(0, 2)	1	1	0	0	0
4	(1, 0)	1	0	0	0	1
5	(1, 1)	1	1	0	0	0
6	(1, 2)	1	1	1	0	1
7	(2, 0)	1	1	0	0	0
8	(2, 1)	1	1	1	0	1
9	(2, 2)	1	1	1	1	0

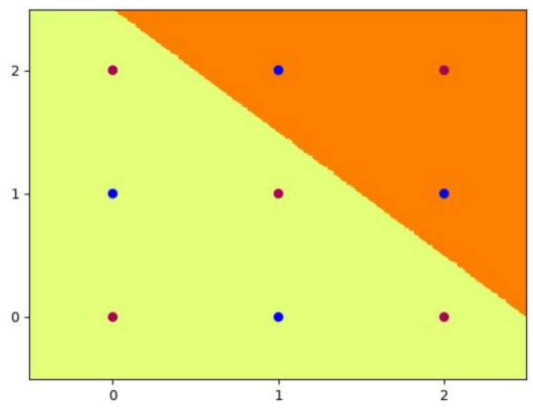
3.

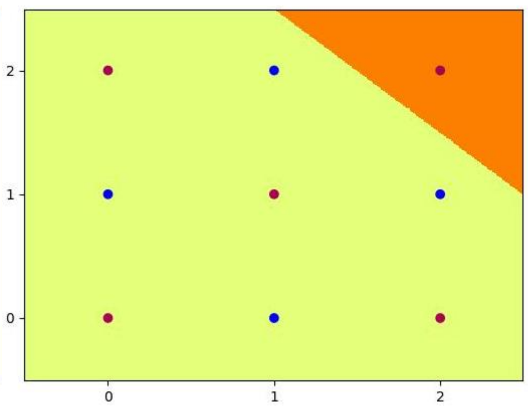


hid_4_0.jpg

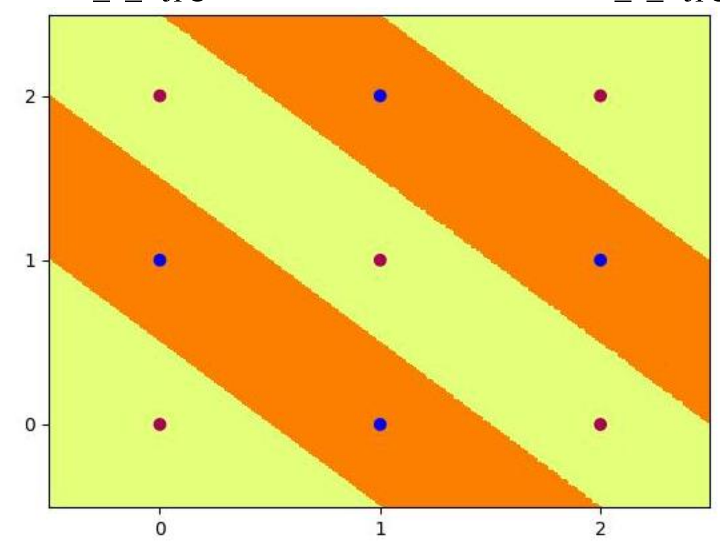


hid_4_1.jpg



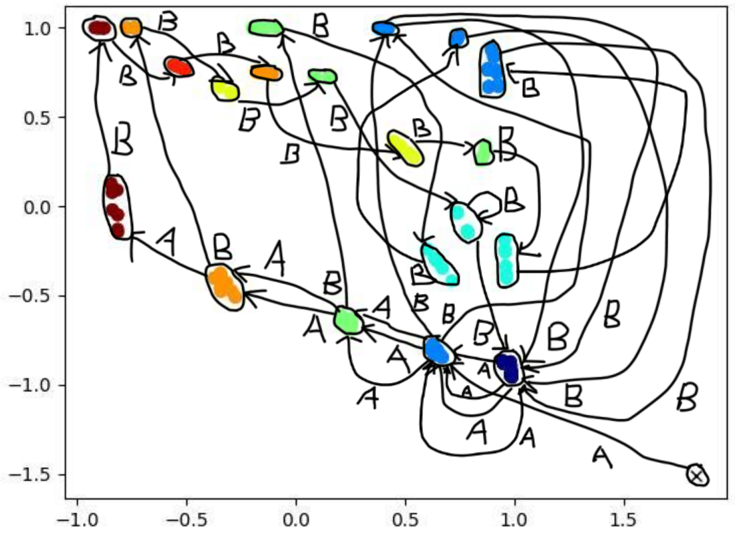


hid_4_2.jpg hid_4_3.jpg

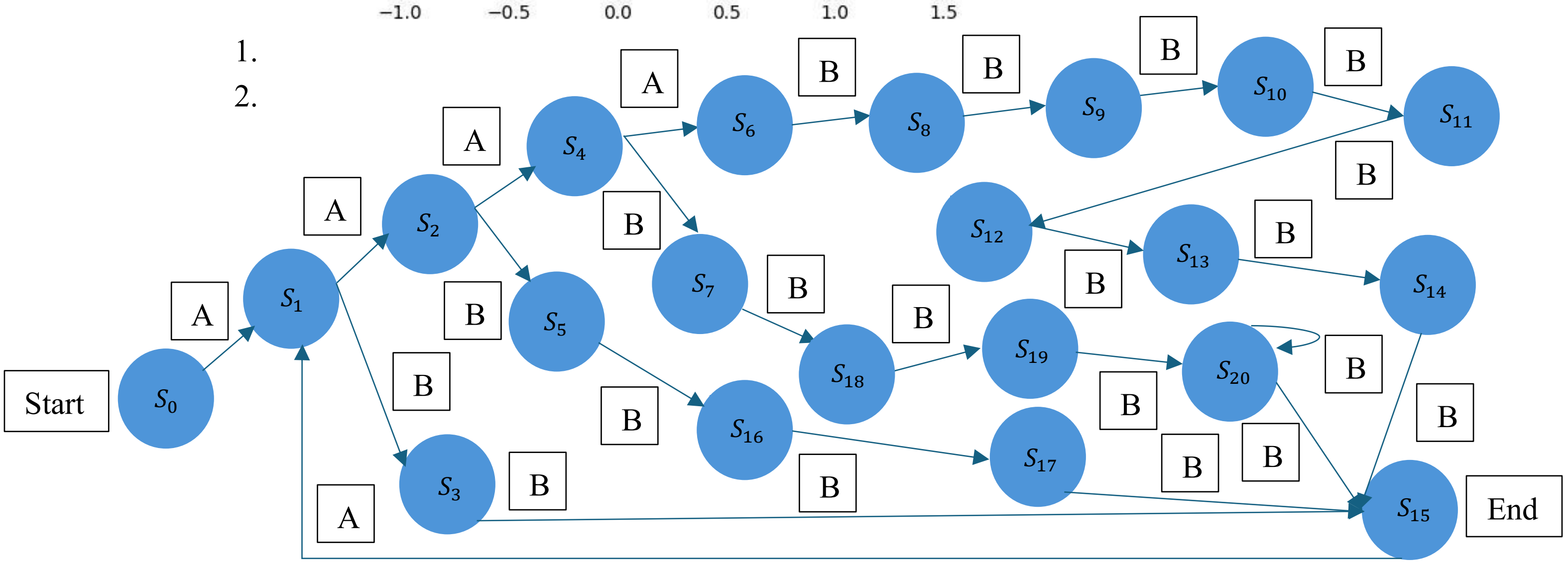


out_4.jpg

Part 3: Hidden Unit Dynamics for Recurrent Networks



1.
2.



3. The SRN network accomplishes the anb^{2}n task by dynamically adjusting hidden unit activations across different phases of the input sequence.

● Hidden unit activations during the A phase

  ❿ During the A phase, certain hidden units show increasing activation values, especially one unit that gradually “counts” the number of A's.

  ❿ As shown in the heatmap and trajectory visualization, hidden states transition from cooler to warmer colors (bottom-right to top-left), indicating accumulation of A's.

  ❿ These states encode the memory of how many A’s have been seen so far.

  ❿ As the network reads each A, the activation of one hidden unit gradually increases, indicating it is counting the number of A's.

  ❿ A [0.62 -0.8], A [0.21 -0.65], A [-0.37 -0.4]. One unit steadily increases with each A, reflecting the accumulation of A's (i.e., storing n).

  ❿ In the 2D hidden state plot, this is shown as a transition through clustered regions labeled A, moving in a loop-like trajectory, suggesting discrete internal memory states corresponding to the count of A's.

● Correctly predicting all B’s after the first B

  ❿ When the first B is encountered, the network experiences a sharp hidden state change.

  ❿ This is a key turning point where the model begins to "use" the stored A count to predict a matching number of B's.

  ❿ After the first B, the activation of a different hidden unit starts decreasing gradually, functioning as a countdown mechanism for the 2n B's.

  ❿ B [-0.77 1.] → B [-0.35 0.67] → B [0.09 0.72] → B [0.74 -0.04] → B [0.88 0.67]. This shows how the second unit (e.g., y-coordinate) goes through a structured decay phase, helping track how many B's are left to process.

  ❿ This allows the network to maintain high confidence in predicting B (e.g., output probabilities like [0.43 1.]) while avoiding premature switching back to A.

  ❿ Visually, in the state space diagram, B trajectories loop through a distinct cluster path, separate from A’s trajectory, further supporting the idea of structured countdown dynamics.

● Predicting the first A after the last B

  ❿ After the first B, a new hidden unit takes over with activations that gradually decrease.

  ❿ This unit acts like a reverse counter, decrementing as B's are processed.

  ❿ This allows the model to predict exactly 2n B’s by ensuring the activations don’t reach a terminal state too early or too late.

  ❿ After processing 2n B’s, the network resets its hidden state and prepares to process the next A: B [0.99, -0.95] → A [0.63, -0.83] → A [0.21, -0.62]

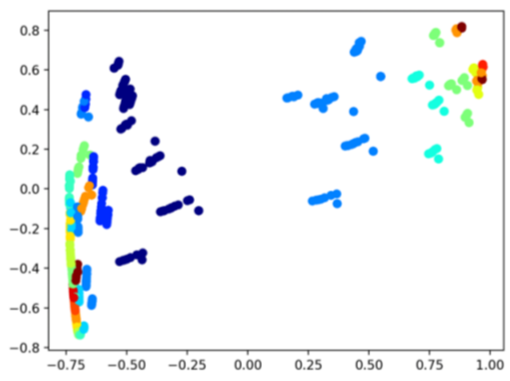
  ❿ The hidden activations return to patterns like the initial A phase, and output probabilities reflect this transition: [0.83, 0.17] → [0.62, 0.38] → [0.44, 0.56]

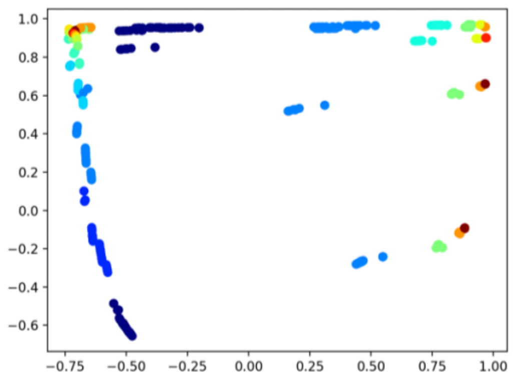
  ❿ This indicates the network recognizes the end of the B sequence and resets its internal counter to start processing the next A for the new anb^{2}n sequence.

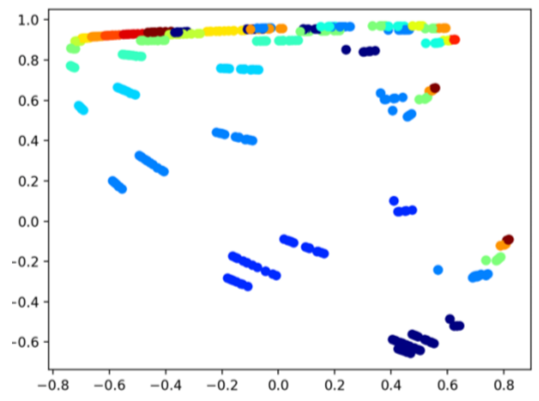
⑩ This is evident in the state diagram as the system loops back to the A region in hidden state space.

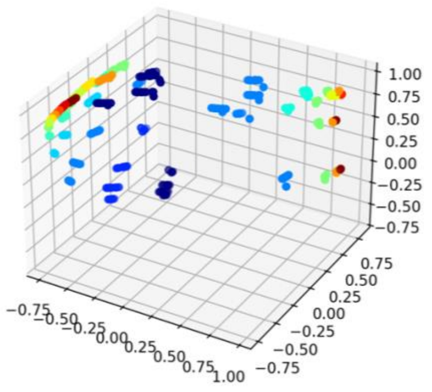
The network encodes A count during the A phase via increasing activation in one hidden unit. It counts down 2n B's using another hidden unit that decreases activation, allowing accurate B predictions. It resets the hidden state after the last B to prepare for the next sequence, ensuring correct prediction of the next A.

4.


anb2nc3n_lstm3_01.jpg


anb2nc3n_lstm3_02.jpg


anb2nc3n_lstm3_03.jpg


3D plot.jpg

5. It explains how the LSTM implements an b^2n c^3n task end to end.

- Task Background & Requirements
  - ⑩ Phase Recognition: Know when it's in the A phase (reading/gathering n A's), then switch to the B phase (emitting 2n B's), then to the C phase (emitting 3n C's), and finally automatically reset back to A for the next cycle.
  - ⑩ Precise Counting: Accurately accumulate the count of A's (n) in long-term memory, then count down in the B phase (2n steps), then count down in the C phase (3n steps).
  - ⑩ Clean Transitions: Seamlessly switch between phases, without mixing symbols or leaking counts.
- LSTM Core Mechanisms
  - ⑩ Hidden State (H): Under tanh, each H-dimension stays in [–1,1], and the network learns three well-separated clusters in H-space for phases A, B, and C.

❿ Cell State (C): Not limited by tanh, C can range widely (e.g. [–4,5]) so it can accumulate, or decay counts exactly.

❿ The Three Gates

Gate	Controls	Effect
Forget	$f_t = \sigma(Wf \cdot [H_{t-1}, x_t]+bf)$	Decides how much of previous C to keep vs. drop.
Input	$i_t = \sigma(Wi \cdot [H_{t-1}, x_t]+bi)$	Decides how much new input (x_t=A/B/C) to write into C.
Output	$o_t = \sigma(Wo \cdot [H_{t-1}, x_t]+bo)$	Decides how much of C to expose as the new H (and drive output).

- Phase-by-Phase Gate Patterns

Phase	Forget f_t	Input i_t	Output o_t	Intuition
A	≈1 (keep all memory)	≈1 (fully accumulate A)	Open H_1, close H_3	Accumulate A's count in C; H clusters in "A region."
B	Partial (release A)	≈0 (pause new B write)	Open H_2, close H_1	Hold/decay A's stored count; H jumps to "B region."
C	Clear B residue	≈1 (accumulate or track C)	Open H_3, close H_2	Use stored count to emit C's; H jumps to "C region."
Reset	$f_t \to 1$, $i_t \to 0$, $o_t \to H_1$ only	—	—	C returns near zero; H back in "A-start" cluster.

- Detailed Numerical Walk-through (n=4, first cycle)

❿ Phase (4 steps of A)

Step	$H = [H_1, H_2, H_3]$	SoftMax → [P(A), P(B), P(C)]	What's happening?
A_1	[0.55, 0.57, –0.24]	[0.63, 0.37, 0.00]	H_1/H_2 start rising; H_3 negative; output firmly A.
A_2	[0.79, 0.74, –0.19]	[0.48, 0.52, 0.00]	Output gate shifting slightly toward B but still <50%.
A_3	[0.86, 0.79, –0.12]	[0.37, 0.63, 0.00]	H moves deeper into A-cluster; output now slightly Bish.
A_4	[0.88, 0.81, –0.10]	[0.33, 0.67, 0.00]	Last A: H_1/H_2 peak; ready to switch to B next.

❿ B-Phase (8 steps of B)

Step	H	SoftMax	Interpretation
B_1	[0.97, 0.55, 0.66]	[0.01,0.99,0.00]	Forget gate partly releases A-memory; H_3 spikes.
…	…	…	B_2–B_6 all → [0,1,0], stable B output.
B_7	[0.37, –0.07, 0.96]	[0.00,0.92,0.08]	H_2 turns negative → C gets a toe-in probability.
B_8	[–0.44, –0.36, 0.94]	[0.00,0.02,0.98]	H_3 high; C nearly takes over.

❿ C-Phase (12 steps of C)

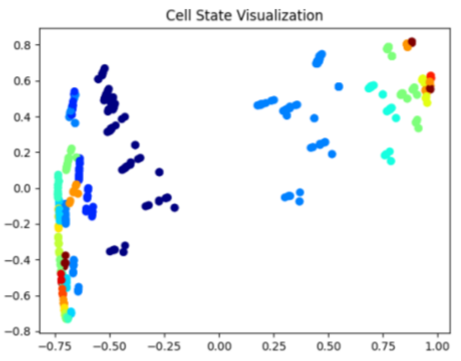
Step	H	SoftMax	Interpretation
C_1	[–0.70, –0.42, 0.94]	[0.00,0.00,1.00]	H_3 high → pure C output.
…	…	…	C_2–C_9 remain [0,0,1], robustly producing C.
C_{10}	[–0.64, –0.57, 0.17]	[0.10,0.01,0.89]	H_3 decays near threshold → A gets small probability.
C_{11}	[–0.58, –0.13, –0.31]	[0.10,0.01,0.89]	H_3 crosses zero → output gate begins favoring A.
C_{12}	[–0.48, 0.46, –0.65]	[0.89,0.00,0.10]	H_3 well negative → final switch to A, next cycle starts.

❿ New-Cycle A Reset

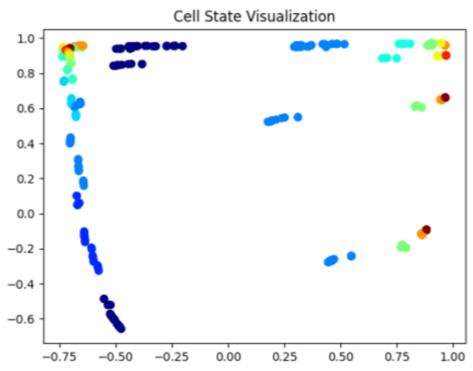
Step	H	SoftMax	Interpretation
A_1'	[0.45, 0.70, –0.27]	[0.78,0.22,0.00]	H returns to A-start region; new cycle begins.

- Key Conclusions & Outlook

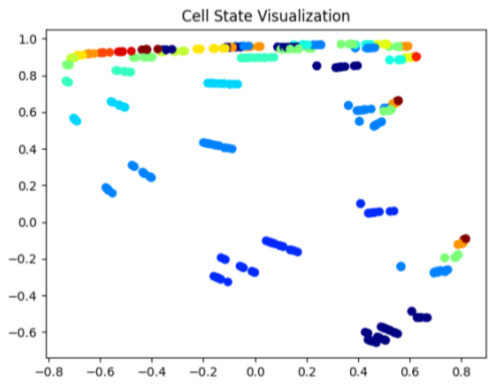
- ❿ Three-Phase FSM Learned: Hidden-state clusters + gate thresholds implement a clear finite-state machine (A→B→C→A).
- ❿ Accurate Counting: Cell state exactly accumulates “n” then counts down “2n” B’s and “3n” C’s.
- ❿ Highly Interpretable: Every dimension (H_1, H_2, H_3) and gate (f, i, o) has a clear, phase-specific role.
- ❿ End-to-End Training: SoftMax probabilities converge to near-perfect one-hot ($P \approx 1$) outputs, confirming the model generalizes the nested pattern.
- ❿ Extendibility: This same mechanism can be scaled to deeper nested structures or more complex grammars requiring long-term counting.



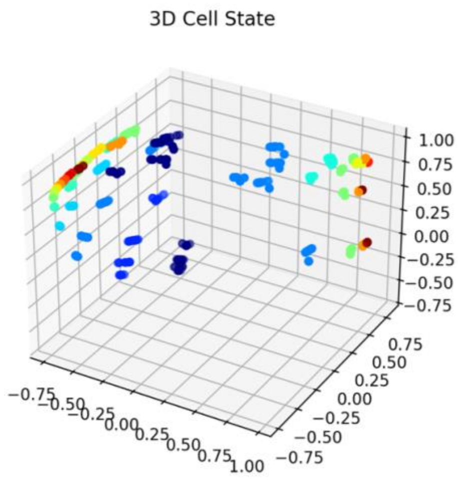
anb2nc3n_lstm3_01_cell state.jpg



anb2nc3n_lstm3_02_cell state.jpg



anb2nc3n_lstm3_03_cell state.jpg



3D plot_cell state.jpg