Assignment 1

Part 1

1. Final Test Accuracy: 6961 / 10000 （70%）
   Final Test Loss: 1.0104
   Confusion Matrix:
   [[764. 6. 8. 14. 30. 63. 2. 63. 32. 18.]
   [7. 667. 110. 18. 29. 22. 57. 15. 25. 50.]
   [7. 62. 689. 27. 26. 20. 47. 39. 45. 38.]
   [3. 39. 59. 760. 15. 55. 14. 18. 26. 11.]
   [58. 54. 80. 21. 624. 21. 32. 36. 19. 55.]
   [7. 28. 124. 17. 20. 724. 29. 7. 33. 11.]
   [5. 22. 146. 9. 24. 24. 726. 20. 10. 14.]
   [17. 30. 27. 12. 83. 14. 55. 625. 86. 51.]
   [10. 38. 97. 42. 6. 30. 43. 7. 703. 24.]
   [8. 50. 92. 4. 50. 31. 20. 29. 37. 679.]]

2. Final Accuracy: 8412/10000 (84%)
   Test Loss: 0.5282
   Hidden Units: 100
   Epochs Trained: 10
   Confusion Matrix:
   [[853. 8. 4. 4. 30. 25. 4. 39. 26. 7.],
   [4. 803. 23. 6. 26. 11. 65. 3. 27. 32.],
   [8. 14. 832. 38. 12. 21. 27. 14. 16. 18.],
   [3. 13. 29. 911. 1. 18. 5. 3. 6. 11.],
   [39. 27. 22. 5. 816. 9. 27. 16. 24. 15.],
   [11. 12. 75. 9. 12. 825. 23. 2. 17. 14.],
   [3. 17. 53. 7. 22. 7. 878. 6. 2. 5.],
   [15. 11. 20. 5. 32. 12. 33. 810. 25. 37.],
   [9. 26. 23. 45. 5. 11. 28. 3. 839. 11.],
   [4. 20. 40. 2. 33. 7. 23. 14. 12. 845.]]

   Parameter Count:
   Input to Hidden Layer: 784 × 100 + 100 = 78,500
   Hidden to Output Layer: 100 × 10 + 10 = 1,010
   Total: 79,510 parameters

3. Final Accuracy: 9381/10000 (94%)
   Confusion Matrix:
   [[951. 3. 1. 0. 28. 0. 0. 13. 2. 2.]
    [4. 916. 11. 0. 13. 1. 38. 6. 3. 8.]
    [11. 3. 895. 40. 10. 7. 17. 6. 6. 5.]
    [2. 1. 11. 967. 2. 3. 5. 4. 2. 3.]
    [17. 6. 1. 3. 934. 1. 9. 9. 13. 7.]
    [4. 8. 42. 7. 6. 906. 19. 5. 1. 2.]
    [3. 5. 10. 1. 6. 3. 968. 3. 0. 1.]
    [9. 3. 3. 1. 7. 1. 7. 947. 3. 19.]
    [5. 4. 7. 11. 9. 3. 11. 2. 945. 3.]
    [5. 4. 12. 4. 12. 0. 2. 5. 4. 952.]]

   Parameter Count:
   conv1: (1 * 5 * 5 + 1) * 32 = 832
   conv2: (32 * 5 * 5 + 1) * 64 = 51264
   fc1: (1024 + 1) * 128 = 131200
   fc2: (128 + 1) * 10 = 1290
   Total = 832 + 51264 + 131200 + 1290 = 184586 parameters

4. Relative Accuracy:
   NetLin: NetLin achieved around 70% accuracy, which was honestly what I expected. Since it's basically just a linear classifier with no hidden layers, it can only do linear transformations on the flattened image. For a task that needs to recognize spatial features like kana characters, this approach just doesn't have enough power.
   NetFull: NetFull performed much better at 85% accuracy. Adding that hidden layer with 128 neurons really made a difference - the tanh activation brought in the nonlinearity that let the model learn more complex patterns. The improvement was quite noticeable.
   NetConv: NetConv was clearly the winner, reaching 94% accuracy within just 10 epochs. I think this is mainly because the convolutional layers can automatically extract local features, which is perfect for recognizing structured characters like kana that have distinct stroke patterns.

   Number of Independent Parameters:
   NetLin: 7,850 (small, but weak performance)
   NetFull: 79,510 (moderate size, significant improvement)
   NetConv: 184,586 (largest, but excellent accuracy)

   Confusion Matrix Insight:
   NetLin: Confuses characters like ‘ki’ and ‘sa’, or ‘so’ and ‘n’, due to lack of structural understanding.
   NetFull: Some improvements, but still confused by characters with similar pixel-level

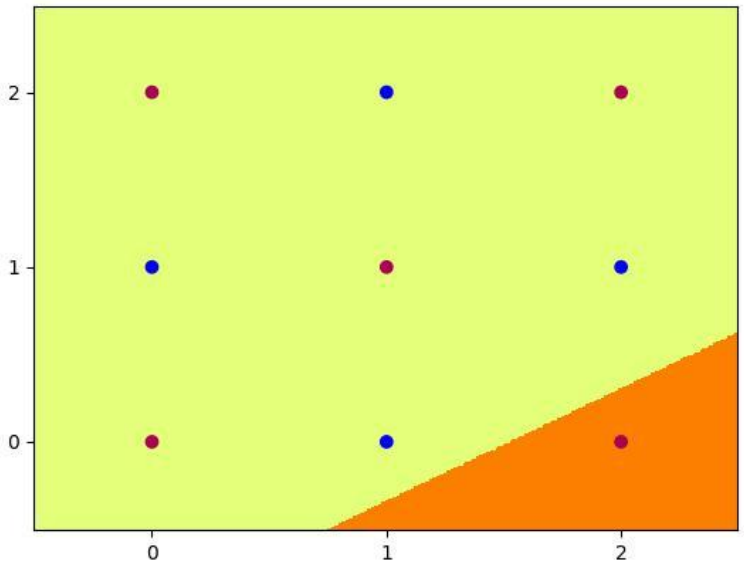
patterns.

NetConv: Most accurate, but still makes occasional errors between characters with subtle stroke differences (e.g. ‘me’ vs ‘nu’). These are hard even for humans to differentiate.

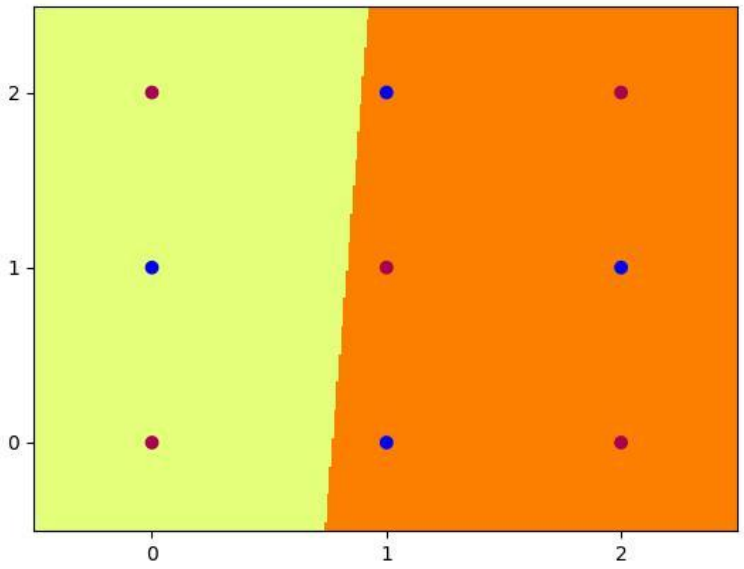
Part 2

1.

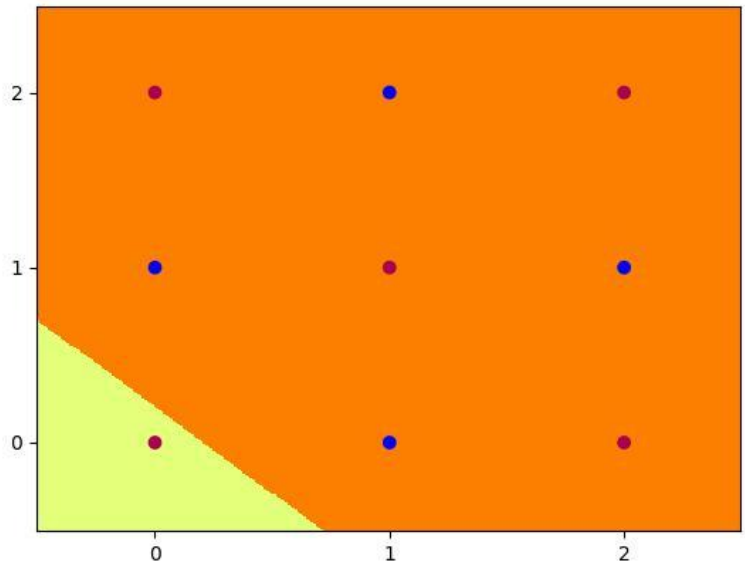
hid_5_0



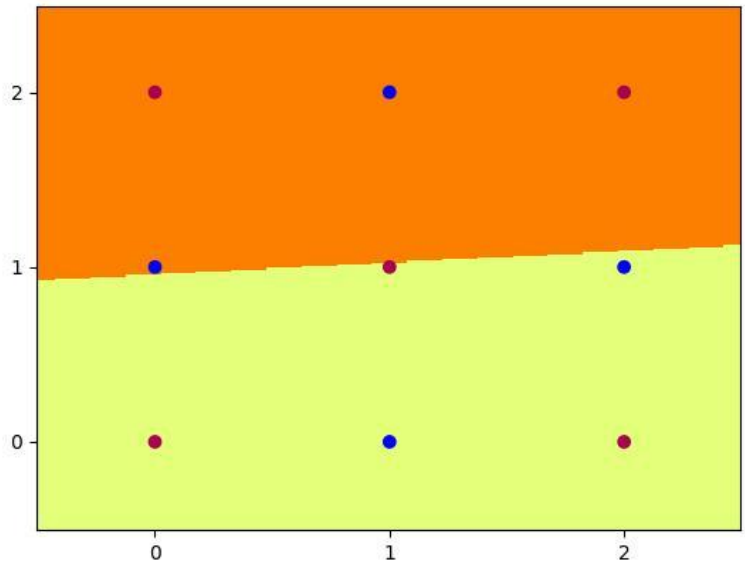
hid_5_1



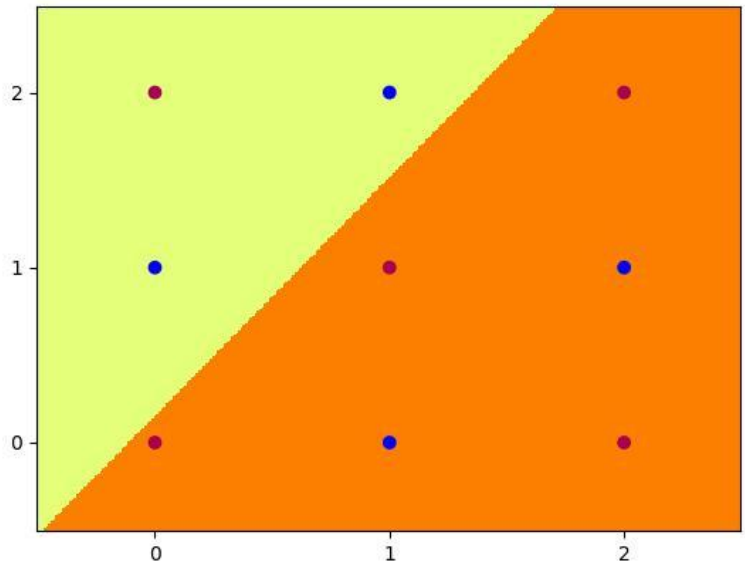
hid_5_2



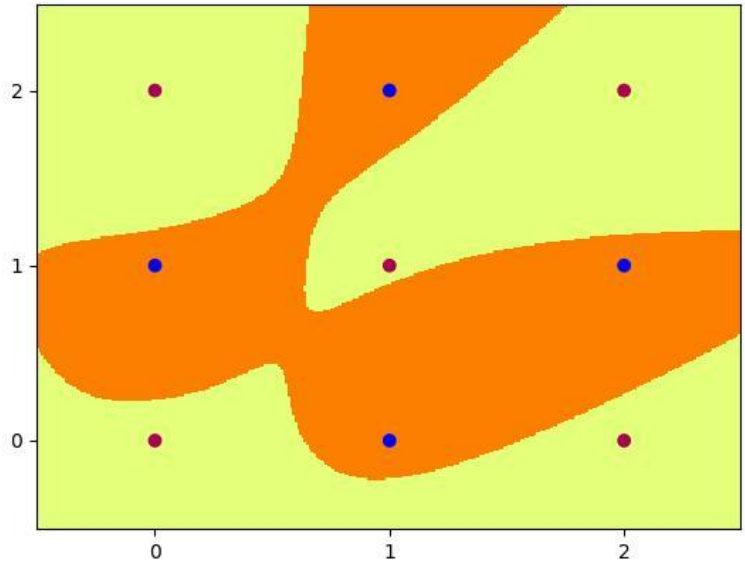
hid_5_3



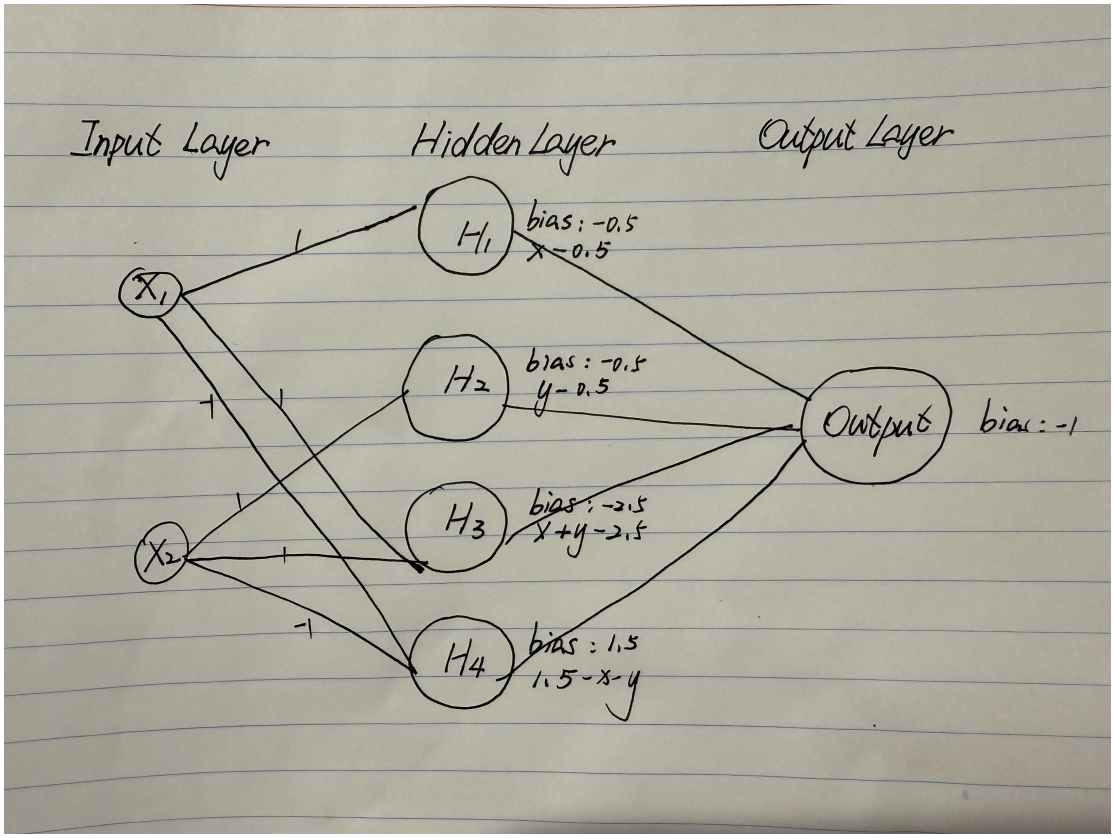
hid_5_4



out_5



2.



Hidden Layer

Node	w_1 (x-weight)	w_2 (y-weight)	Bias	Function
h1	1	0	-0.5	step(x - 0.5)
h2	0	1	-0.5	step(y - 0.5)
h3	1	1	-2.5	step(x + y - 2.5)
h4	-1	-1	1.5	step(1.5 - x - y)

Output Layer

From h1	From h2	From h3	From h4	Bias	Function
1	1	-1	1	-1	step($h_1 + h_2 - h_3 + h_4 - 1$)

Decision Line Equations:

h_1: x - 0.5 = 0 → x = 0.5 (vertical line)

h_2: y - 0.5 = 0 → y = 0.5 (horizontal line)

h_3: x + y - 2.5 = 0 → y = -x + 2.5 (diagonal line)

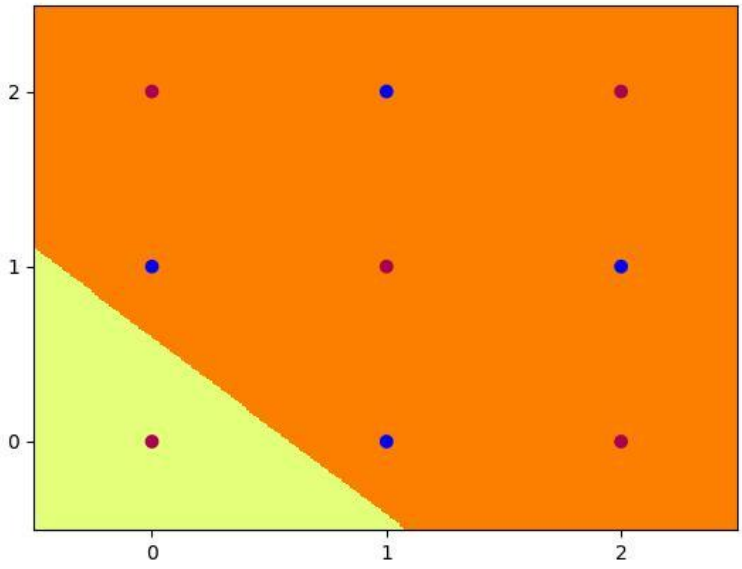
h_4: -x - y + 1.5 = 0 → y = -x + 1.5 (anti-diagonal line)

Activation Truth Table

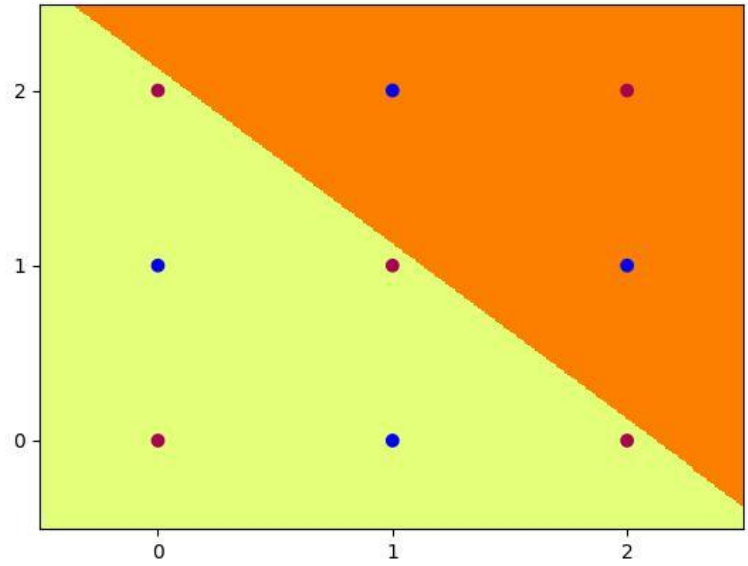
Point	x	y	h1	h2	h3	h4	Linear Combination	Step Output	Target
(0,0)	0	0	0	0	0	1	0+0-0+1-1 = 0	0	0
(0,1)	0	1	0	1	0	1	0+1-0+1-1 = 1	1	1
(0,2)	0	2	0	1	0	0	0+1-0+0-1 = 0	0	0
(1,0)	1	0	1	0	0	1	1+0-0+1-1 = 1	1	1
(1,1)	1	1	1	1	0	0	1+1-0+0-1 = 1	1	0
(1,2)	1	2	1	1	1	0	1+1-1+0-1 = 0	0	1
(2,0)	2	0	1	0	0	0	1+0-0+0-1 = 0	0	0
(2,1)	2	1	1	1	1	0	1+1-1+0-1 = 0	0	1
(2,2)	2	2	1	1	1	0	1+1-1+0-1 = 0	0	0

3.

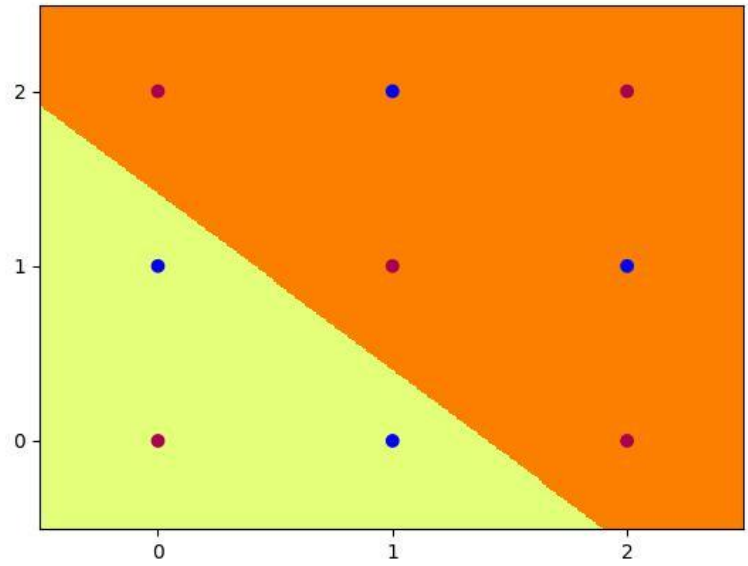
hid_4_0



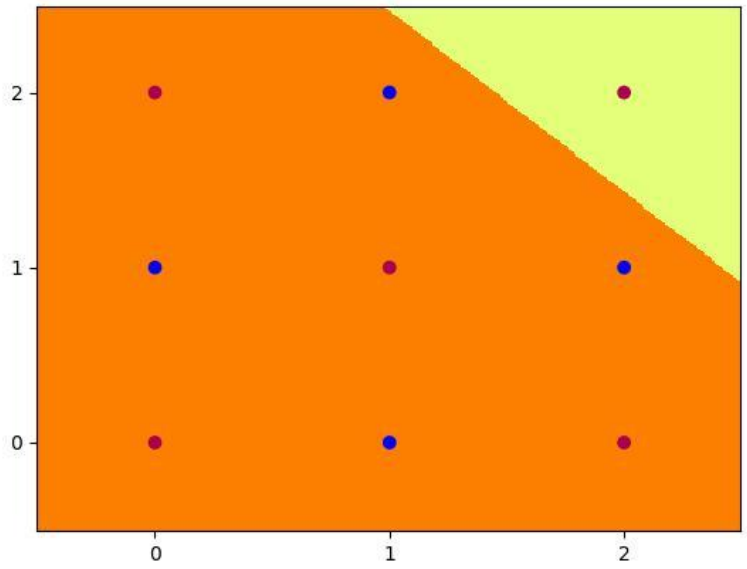
hid_4_1



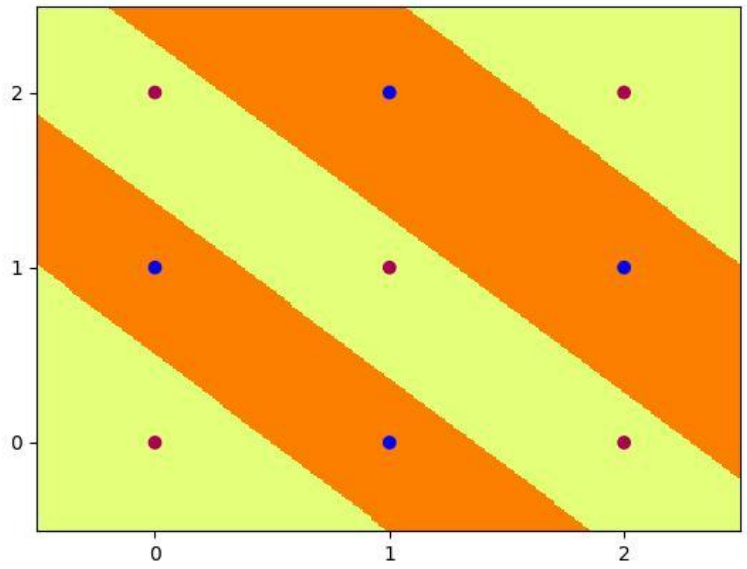
hid_4_2



hid_4_3

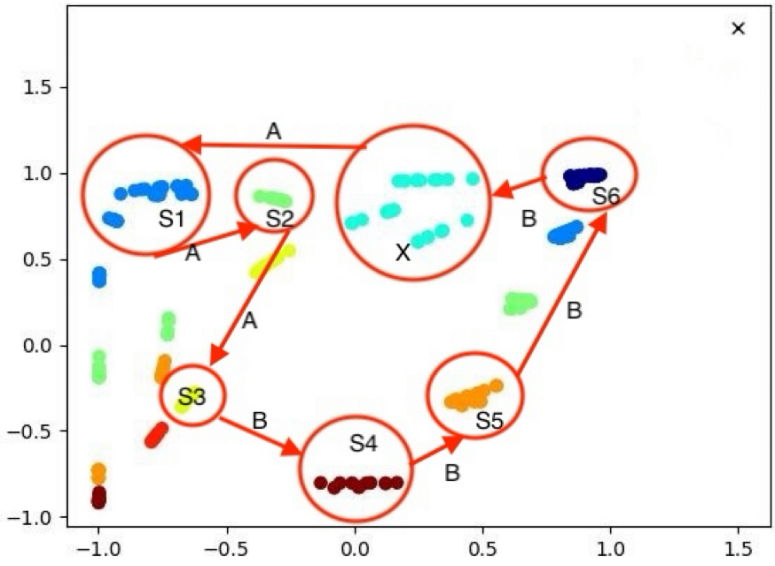


out_4

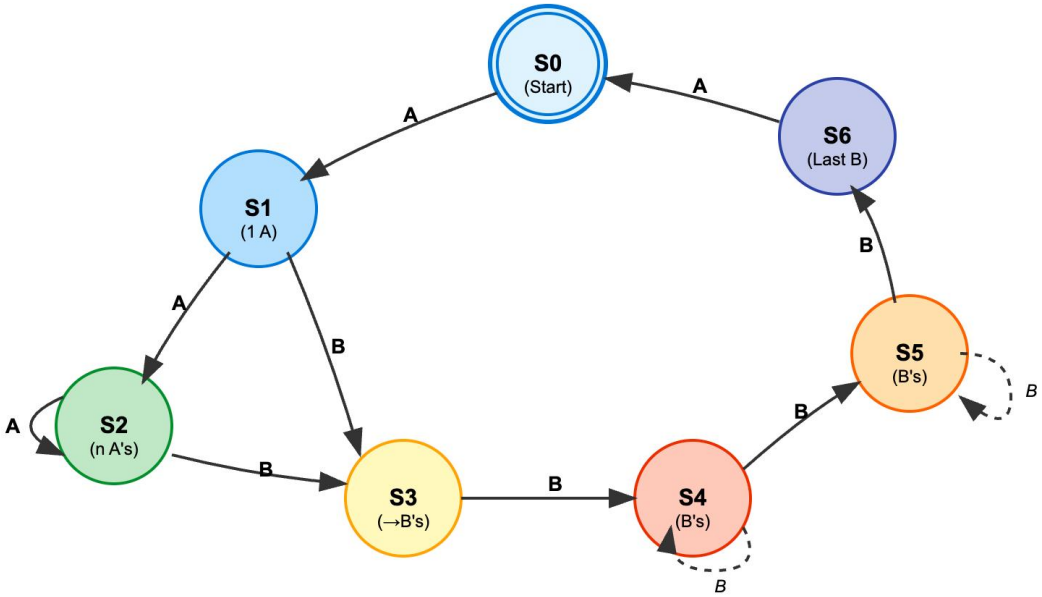


Part 3

1. epoch: 100000
   loss: 0.0417



2.



3. The SRN learns to solve the anb2n language task through a distributed counting mechanism implemented in its 2D hidden state space.

   During A Processing：

   The hidden unit activations form distinct state clusters that encode how many A's have been read 1, 2, 3, or 4. Each cluster represents a different count state, effectively implementing a neural counter. As more A's are processed, the activation point moves between these counting states in the hidden space.

   During B Processing：

   When the first B appears, the network transitions from "A - counting mode" to "B - output mode." Crucially, the hidden states retain memory of the original A count while simultaneously tracking progress through the B sequence. The network knows it must output exactly 2n B's.

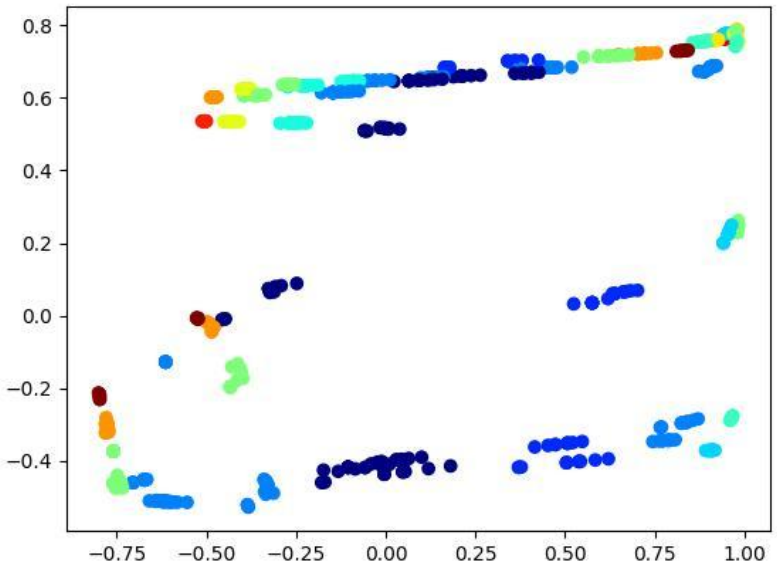
   Prediction Mechanism：

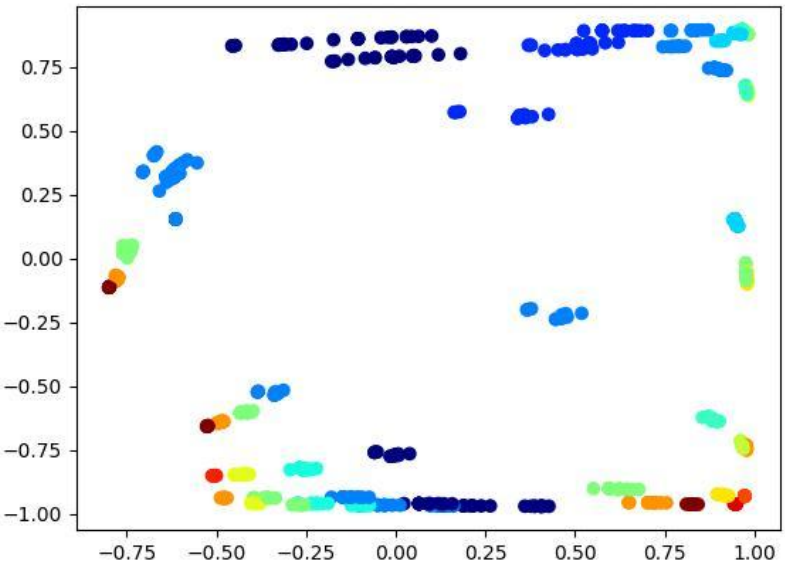
   Predicting B's after the first B: The hidden state encodes both the target number of B's 2n and current position in the B sequence. Since the network knows both values, it can deterministically predict the next symbol should be B until reaching the target count.

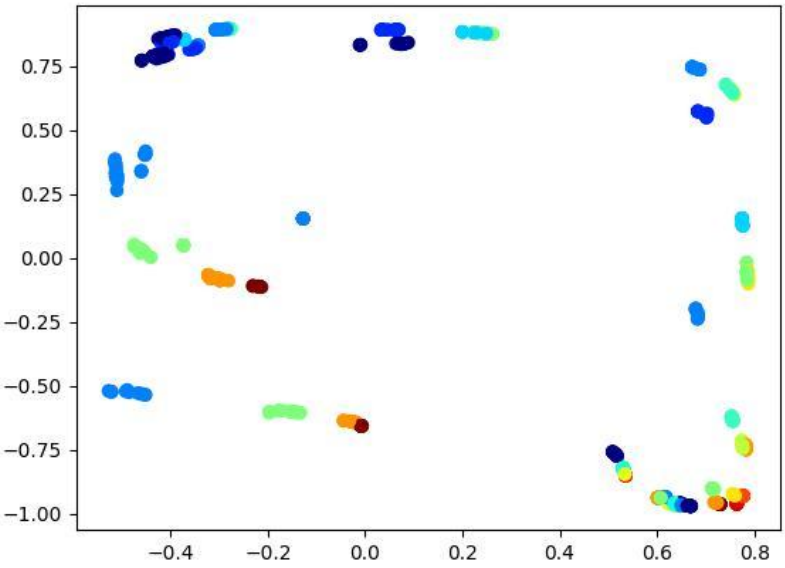
   Predicting A after the last B: When exactly 2n B's have been output, the hidden state recognizes sequence completion and predicts A to start the next sequence, returning to the initial state.

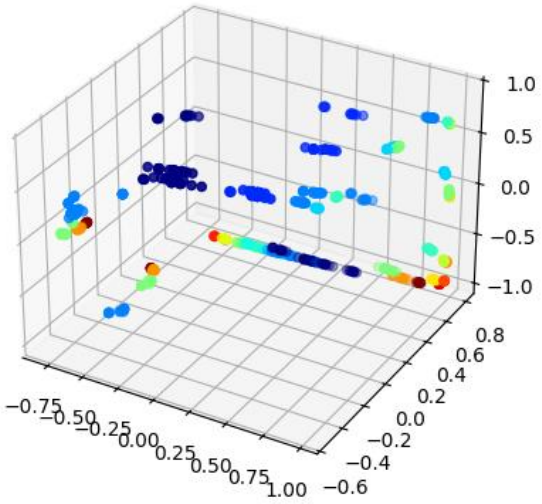
   This creates a distributed finite state machine that maintains counting information across the entire sequence.

4.




0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8



5.
This task asks the LSTM to learn sequences that follow a specific symbolic pattern:
n = 1 → a b b c c c
n = 2 → a a b b b b c c c c c c
n = 3 → a a a b b b b b b c c c c c c c c c

The LSTM needs to count the number of a's (which we call n), then generate exactly 2n b's followed by 3n c's. The key is for the network to remember how many a's it saw and use that count to guide how many b's and c's to produce.

Hidden State Dynamics – 3D Structure

Visualizations show that the LSTM organizes its hidden states into distinct spatial regions based on input type, sequence length, and position.

In horizontal view, states move from left to right across A → B → C phases:
- Left (red/orange) = processing a's
- Middle (blue) = processing b's
- Right (cyan) = processing c's

In the vertical dimension:
- Upper levels (y ≈ 0.8) = longer sequences (n = 3, 4)
- Lower levels (y ≈ -1.0) = shorter sequences (n = 1, 2)

Full 3D views show:

Symbol types (a/b/c) occupy different regions

The model uses all three dimensions to organize its processing logically

How the LSTM Solves the Task

(1) A-processing (Counting phase)

As it reads a's, the hidden state moves through a red/orange cluster. The cell state keeps track of how many a's have been seen (i.e., stores n).

(2) B-processing (Output 2n b's)

Once a's are done, the model enters a blue region. It uses the stored count n to decide when to stop outputting b's. The hidden state tracks how far along it is.

(3) C-processing (Output 3n c's)

The model enters a cyan region. Again, it uses n to determine how many c's to output and tracks progress with its hidden state. When 3n c's are complete, the sequence ends.

Internal Mechanisms of the LSTM

- Cell state: Stores the long-term memory, specifically the count n.
- Hidden state: Tracks current position in the output sequence.
- Gating mechanisms: Control when to update, forget, or output values.

Counting logic:

- During a's: accumulate n in cell state.
- During b's: output based on 2n target.
- During c's: output based on 3n target.
- At the end: reset state to prepare for a new sequence.

Why LSTM Succeeds (Compared to SRN)

- SRNs have limited memory and struggle with long-term dependencies.
- LSTMs maintain the count n over long sequences thanks to cell states.
- LSTMs use a clear 3D internal structure, separating A/B/C states across space.
- Gated updates help avoid vanishing gradients, enabling better training on longer inputs.

Prediction Behavior

After the first b, the network already knows:

The original n

How many b's it has generated

How many more to go

The same applies for c's: it knows the target (3n), tracks output, and stops at the right time.

After generating 3n c's, it predicts the next symbol as a, resets the state, and begins again.

3D Structure Benefits

One dimension encodes symbol type (a/b/c)
Another reflects sequence length (n = 1 to 4)
The third dimension tracks position within the phase
This separation allows the model to reason over sequence logic in a clean and distributed way.

Conclusion
The LSTM essentially learns to be a symbolic counter:

- It remembers n using its cell state
- It tracks progress using hidden state
- It organizes its hidden space by symbol type and sequence phase
- It performs symbolic-like reasoning without being explicitly told how

This example shows how LSTMs can generalize and learn structured behavior using internal memory and spatial organization, making them very effective for tasks that involve counting, sequences, and phase transitions.