COMP9444 Neural Networks and Deep Learning

Term 2, 2025

Assignment – Characters and Hidden Unit Dynamics

Student Name: Zepeng Cui
Student ID: z5576489

Part 1: Japanese Character Recognition

Part 1.1 Linear Model (NetLin)

> **Model description:**
>
> Input: 28×28 grayscale image, flattened into a vector of length 784.

> **Network structure:**
>
> A linear layer `fc` that projects the 784-dimensional input to 10 classes.
>
> A `LogSoftmax` layer outputs the log-probabilities for the 10 classes.

> **Result:**
>
> Test set: Average loss: 1.0100, Accuracy: 6963/10000 (70%)

> **Confusion Matrix:**
>
> Rows = True labels, Columns = Predicted labels

```
[[773.   5.   7.  12.  29.  61.   2.  63.  30.  18.]
 [  7. 669. 109.  16.  28.  23.  57.  13.  25.  53.]
 [  8.  62. 691.  26.  25.  20.  48.  37.  45.  38.]
 [  4.  35.  58. 754.  14.  60.  14.  18.  30.  13.]
 [ 61.  51.  83.  20. 625.  20.  32.  32.  20.  56.]
 [  8.  28. 124.  17.  20. 725.  27.   9.  33.   9.]
 [  5.  24. 147.   9.  26.  25. 721.  20.   9.  14.]
 [ 17.  29.  28.  11.  84.  16.  56. 622.  89.  48.]
 [ 10.  36.  95.  42.   7.  31.  46.   6. 704.  23.]
 [  8.  51.  85.   3.  54.  33.  18.  30.  39. 679.]]
```

Part 1.2 Two-Layer Fully Connected Model (NetFull)

Model description:

Input: 28×28 grayscale image, flattened into a vector of length 784.

Structure:

Fully connected layer `fc1`: 784 → 200, with `tanh` activation function.
Fully connected layer `fc2`: 200 → 10, followed by `LogSoftmax` output.

Result:

Test set: Average loss: 0.4959, Accuracy: 8486/10000 (85%)

Confusion Matrix:

Rows = True labels, Columns = Predicted labels

```
[[854.   4.   1.   5.  25.  33.   2.  40.  29.   7.]
 [  6. 822.  29.   4.  19.  12.  52.   7.  20.  29.]
 [  9.  12. 833.  39.  10.  19.  26.  13.  23.  16.]
 [  4.   9.  31. 917.   2.  16.   2.   2.   7.  10.]
 [ 36.  32.  20.  10. 812.   7.  29.  13.  22.  19.]
 [ 11.  13.  65.   7.  10. 849.  20.   1.  16.   8.]
 [  3.  18.  45.   8.  17.   4. 888.   6.   3.   8.]
 [ 19.  13.  16.   4.  21.  10.  32. 831.  23.  31.]
 [  8.  25.  29.  45.   4.   8.  29.   3. 840.   9.]
 [  4.  17.  50.   6.  28.   8.  20.  15.  12. 840.]]
```

Independent Parameter Count:

Input → Hidden layer

- Weights: 784 × 200 = 156800
- Biases: **200**

Hidden → Output layer

- Weights: 200 × 10 = 2000
- Biases: **10**

Total parameters = 156800 + 200 + 2000 + 10 = 159010

Part 1.3 Convolutional Network (NetConv)

Model description:
Input: 28×28 grayscale image, processed by two convolutional layers and one fully-connected classifier, all with ReLU activations, and a final LogSoftmax.

Network structure:

1. **Conv1:** 1 → 32 channels, 5×5 kernel (padding=2)
   - BatchNorm2d(32) → ReLU → 2×2 MaxPool (28→14)
2. **Conv2:** 32 → 64 channels, 3×3 kernel (padding=1)
   - BatchNorm2d(64) → ReLU → 2×2 MaxPool (14→7)
3. Flatten → Dropout(p=0.25) → Fully connected 64·7·7 → 10 → LogSoftmax

Result:
Test set: Average loss: 0.2511, Accuracy: 9360/10000 (94%)

Confusion Matrix:

Rows = True labels, Columns = Predicted labels

```
[[960.   5.   3.   1.   8.   1.   1.  14.   3.   4.]
 [  2. 925.   5.   3.   7.   1.  38.   6.   4.   9.]
 [  9.   4. 850.  54.   6.   9.  38.  14.   9.   7.]
 [  0.   1.  12. 968.   3.   3.   5.   2.   3.   3.]
 [ 26.   1.   4.  11. 904.   1.  23.  10.  13.   7.]
 [  1.  10.  32.   6.   5. 904.  34.   3.   2.   3.]
 [  2.   0.   4.   3.   1.   1. 984.   3.   0.   2.]
 [  1.   5.   0.   0.   7.   2.   7. 950.  11.  17.]
 [  3.  12.   0.   4.   3.   1.  10.   4. 960.   3.]
 [  7.   6.   5.   3.  10.   0.   0.   3.  11. 955.]]
```

Independent Parameter Count:

- **Conv1 (1→32, 5×5):**
  - Weights: 1×5×5×32 = 800
  - Biases: 32
- **BatchNorm1:**
  - γ (scale): 32
  - β (shift): 32
- **Conv2 (32→64, 3×3):**
  - Weights: 32×3×3×64 = 18432
  - Biases: 64
- **BatchNorm2:**
  - γ: 64
  - β: 64
- **Fully Connected (64·7·7→10):**
  - Weights: 64×7×7×10 = 31360
  - Biases: 10

  Total parameters = 800 + 32 + 32+ 32 + 18432 + 64 + 64 + 64 + 31 360 + 10 = 50890

Part 1.4 Discussion: Neural Network Analysis

a.Relative Accuracy of the Three Models

NetLin (Linear): ~70%

NetFull (2-layer MLP): ~85%

NetConv (2-conv + FC): ~94%

From 70% to 85% and then to 94%, the accuracy improves significantly as the model capacity and structural complexity increase:

The pure linear model can only capture the simplest pixel-level differences, resulting in the lowest performance.The two-layer fully connected network learns hidden-layer feature combinations, improving accuracy by around 15%.The convolutional network leverages local translation invariance and hierarchical feature abstraction, further boosting accuracy by about 9%.

b.Number of independent parameters

Model	Parameters	Total
NetLin	`784×10` weights + `10` biases	**7850**
NetFull	`784×200` + `200` `200×10` + `10`	**159010**

Model	Parameters	Total
NetConv	800 + 32 + 32+ 32 + 18432 + 64 + 64 + 64 + 31 360 + 10 = 50890	**50890**

c.Confusion Matrix Analysis

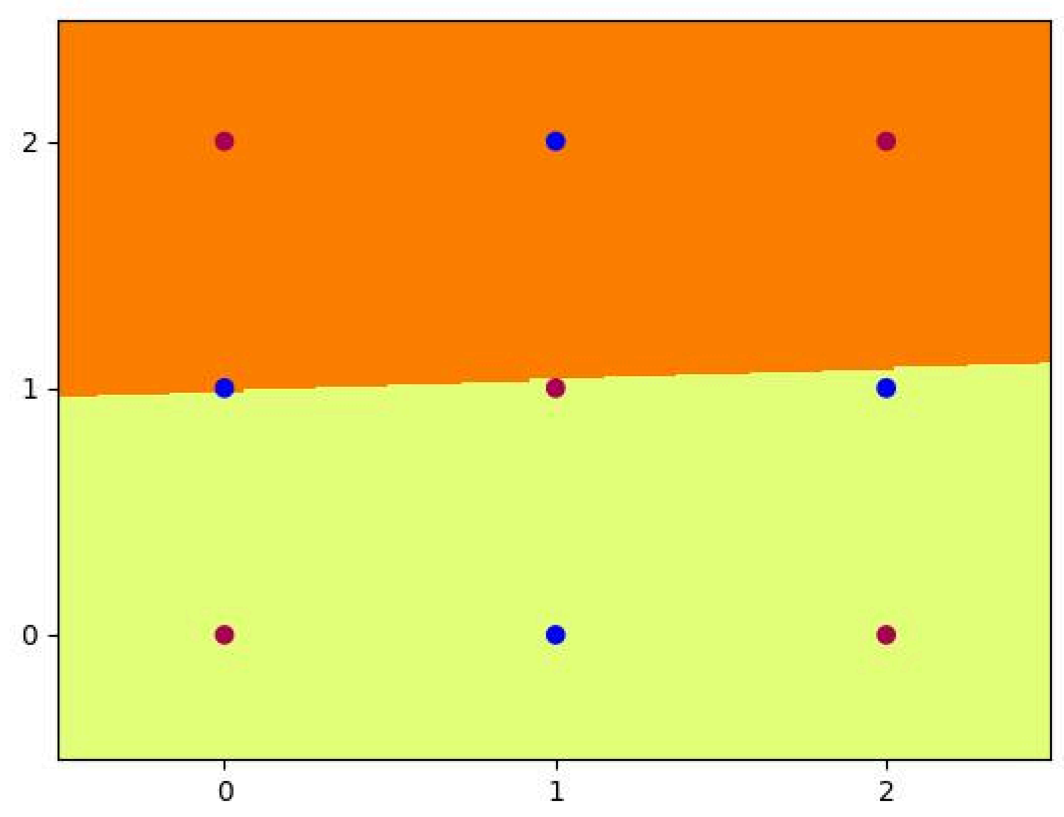
Across all three models, the most persistent errors occur between characters with very similar stroke patterns—e.g. “su”(2) vs. “tsu”(3), “re”(8) vs. “wo”(9), and pairs sharing two vertical strokes (“ha”(5) vs. “na”(4))—because Kuzushiji’s handwritten forms exaggerate historical variations and individual writing styles. NetLin, being purely linear, confuses many classes indiscriminately; NetFull’s hidden layer helps distinguish broad shape differences but still trips over curved-stroke pairs; and NetConv, thanks to its local feature detectors and pooling, best resolves these subtle distinctions, though it still misclassifies the visually closest pairs.

Part 2: Multi-Layer Perceptron

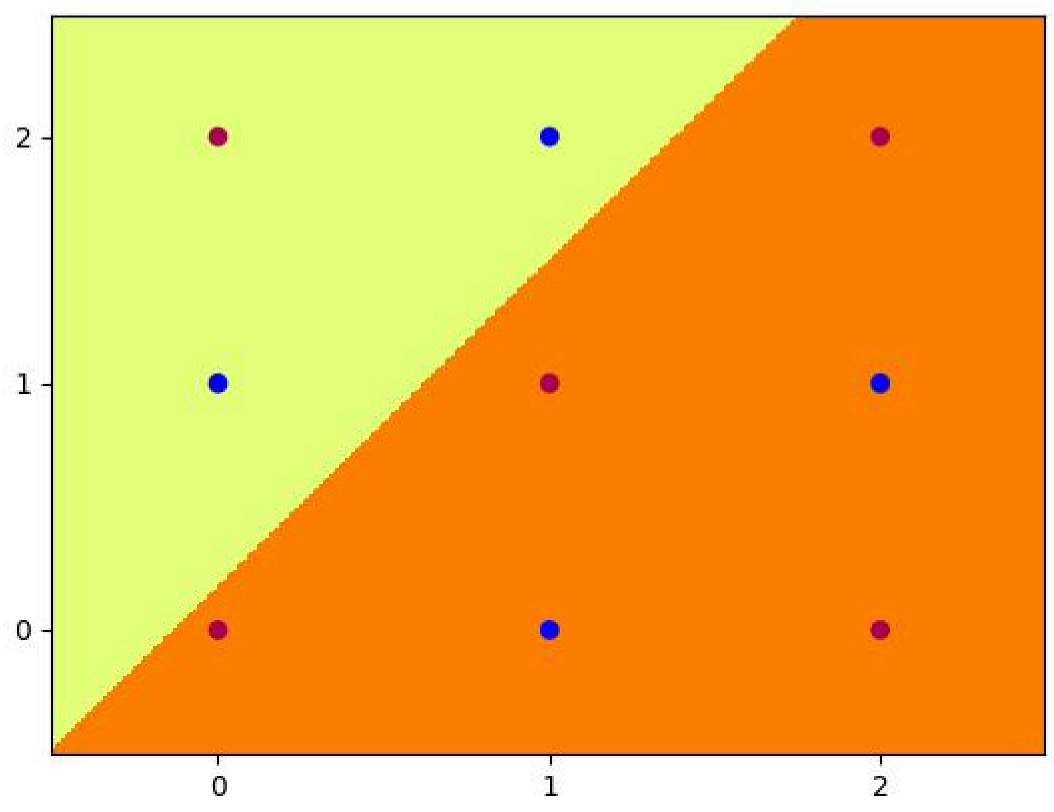
Part 2.1

Hidden Layer Activations (5 nodes):

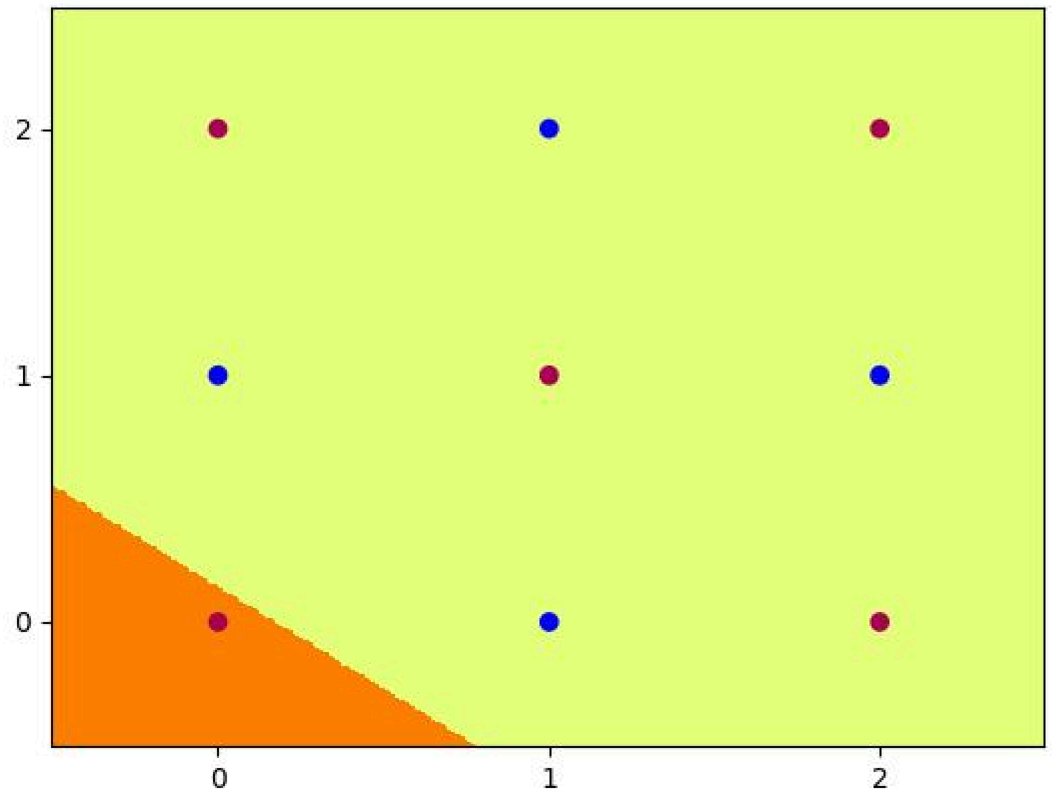
- **Node 0:**



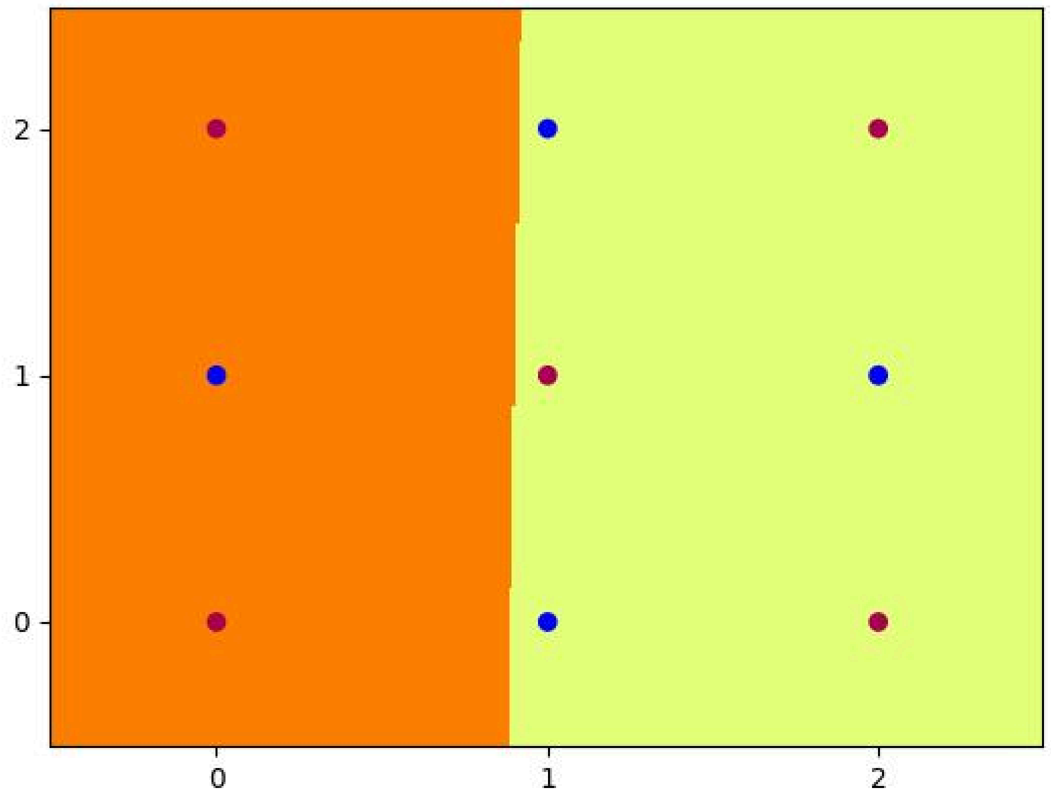
- **Node 1:**



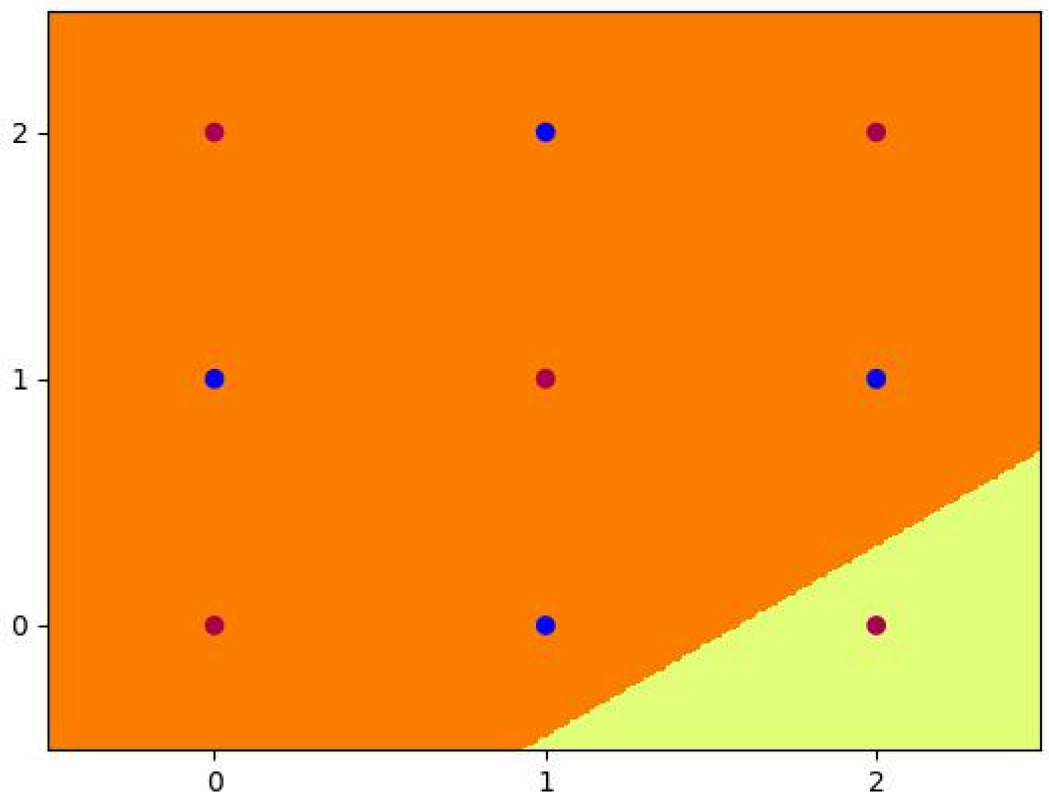
- **Node 2:**



- **Node 3:**

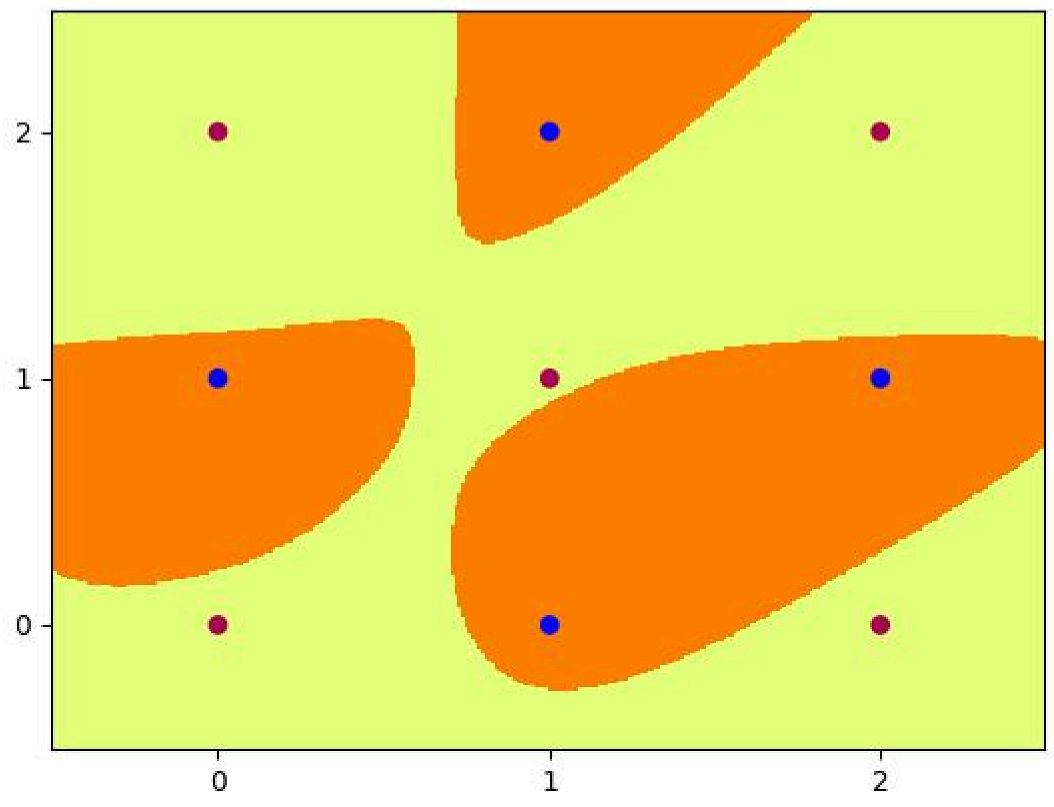


- **Node 4:**



Final Network Output:

I trained a 2-layer neural network with 5 hidden units and sigmoid activation on both layers. The final model reached 100% accuracy. The decision boundary learned by the network is shown below. Each colored region indicates the predicted output, and all training points are correctly classified, demonstrating that the model has successfully learned the classification task.

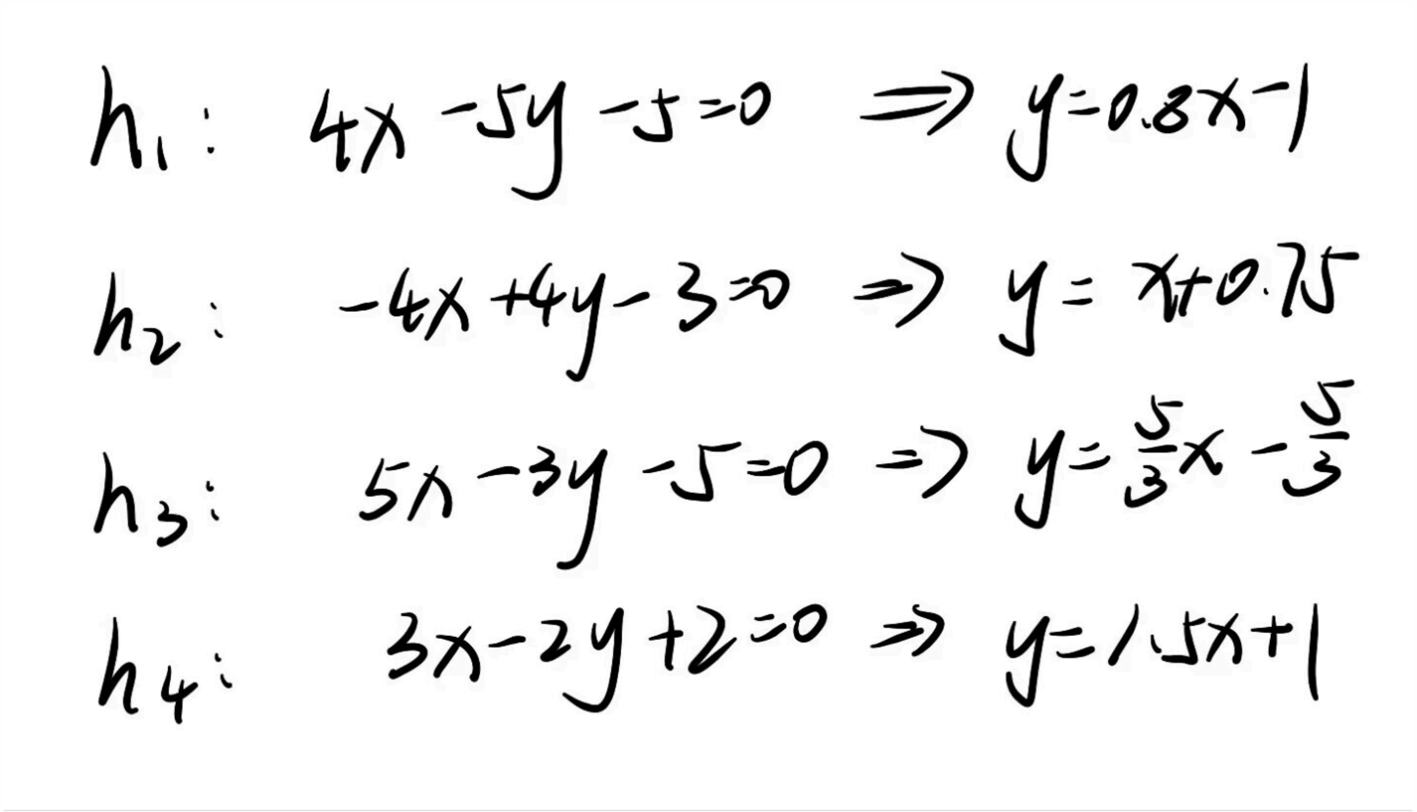


Part 2.2

Weights and biases

Layer	Node	w_1	w_2	Bias
Hidden	h_1	**4**	–5	–5
	h_2	–4	**4**	–3
	h_3	**5**	–3	–5
	h_4	**3**	–2	+2
Output	$\hat{y}$	–2, 1, 1, 4	—	–5

Dividing-line equations



$h_1: \quad 4x - 5y - 5 = 0 \Rightarrow y = 0.8x - 1$

$h_2: \quad -4x + 4y - 3 = 0 \Rightarrow y = x + 0.75$

$h_3: \quad 5x - 3y - 5 = 0 \Rightarrow y = \frac{5}{3}x - \frac{5}{3}$

$h_4: \quad 3x - 2y + 2 = 0 \Rightarrow y = 1.5x + 1$

Activation table (step network)

(x,y)	h1	h2	h3	h4	Output	Target
(0,0)	0	0	0	1	0	0
(0,1)	0	1	0	1	1	1
(0,2)	0	1	0	0	0	0
(1,0)	0	0	1	1	1	1
(1,1)	0	0	0	1	0	0
(1,2)	0	1	0	1	1	1
(2,0)	1	0	1	1	0	0
(2,1)	0	0	1	1	1	1
(2,2)	0	0	0	1	0	0

Part 2.3

Implementation

I rescaled all weights and biases from Part 2 by multiplying them by a factor of 10. This scaling makes the sigmoid function approximate the step function behavior.

Initial Rescaled Weights:

```
Hidden Layer Weights: [[40, -50], [-40, 40], [50, -30], [30, -20]]
Hidden Layer Biases: [-50, -30, -50, 20]
Output Layer Weights: [[-20, 10, 10, 40]]
Output Layer Bias: [-50]
```

Results

Initial Accuracy: 66.67% (the rescaled weights don't immediately solve the problem with sigmoid)

Training Progress: The network required additional training to reach 100% accuracy

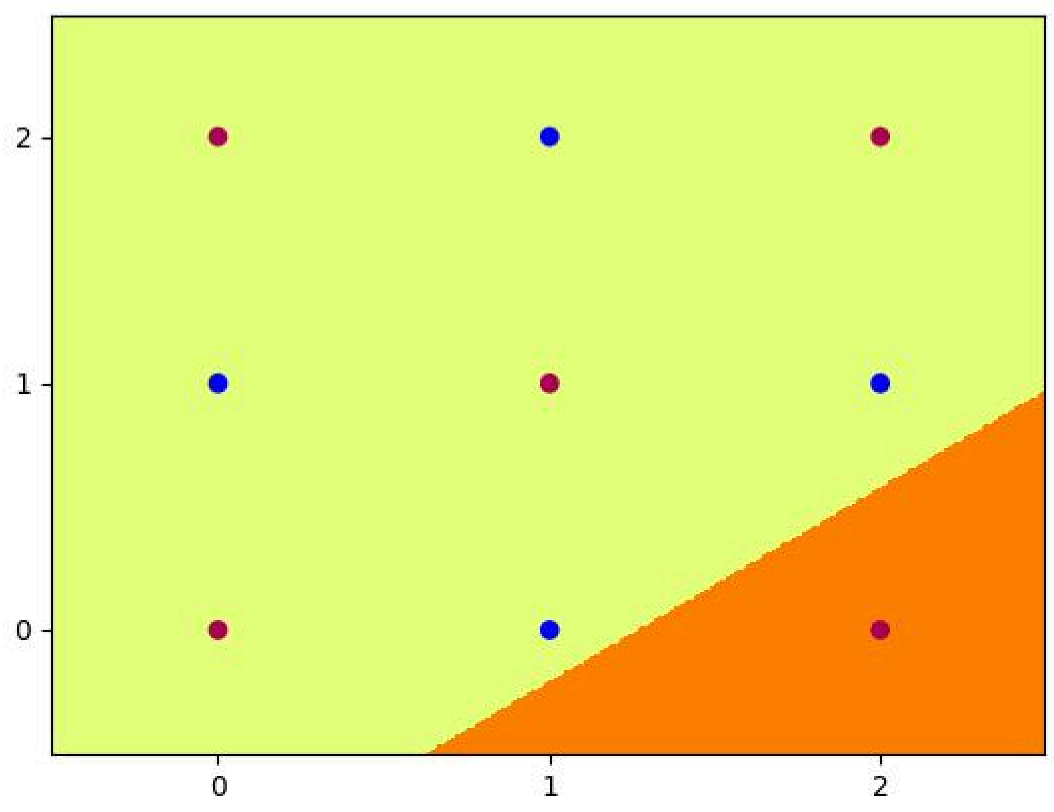
Convergence: Achieved 100% accuracy after approximately 1600 epochs

Final Accuracy: 100%

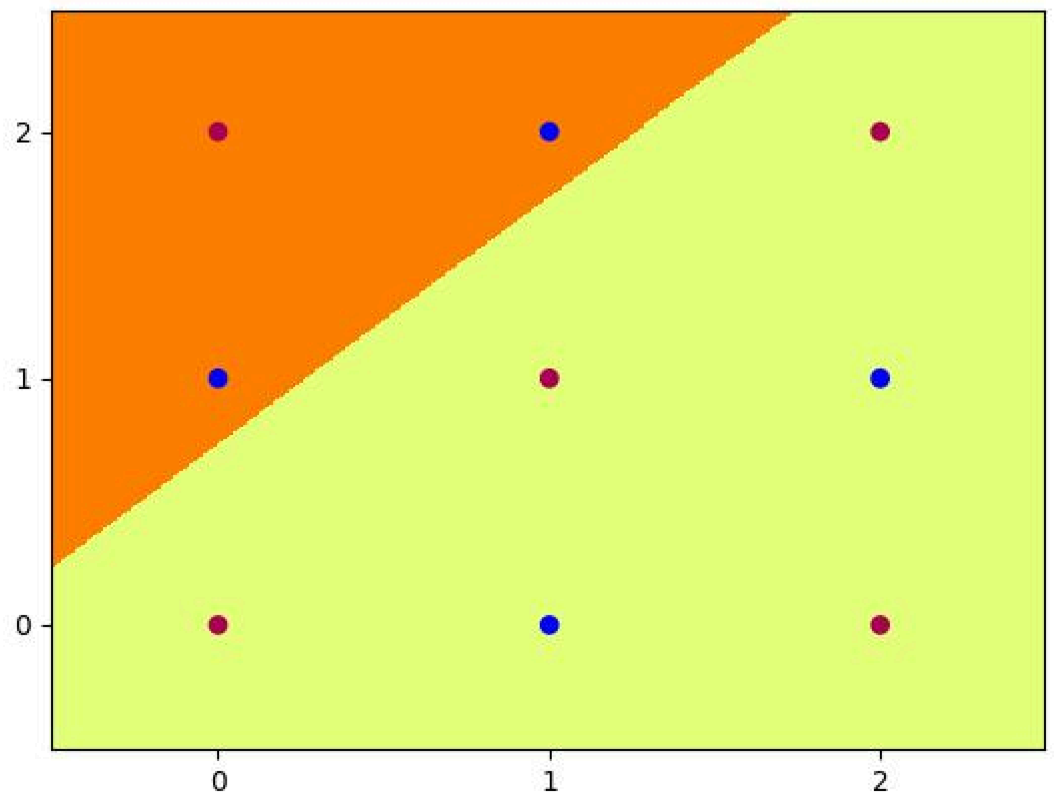
Visual Analysis

Hidden unit activations (hid_4_0.jpg - hid_4_3.jpg): Display sharp but continuous transitions at decision boundaries, contrasting with the discrete jumps in step function version

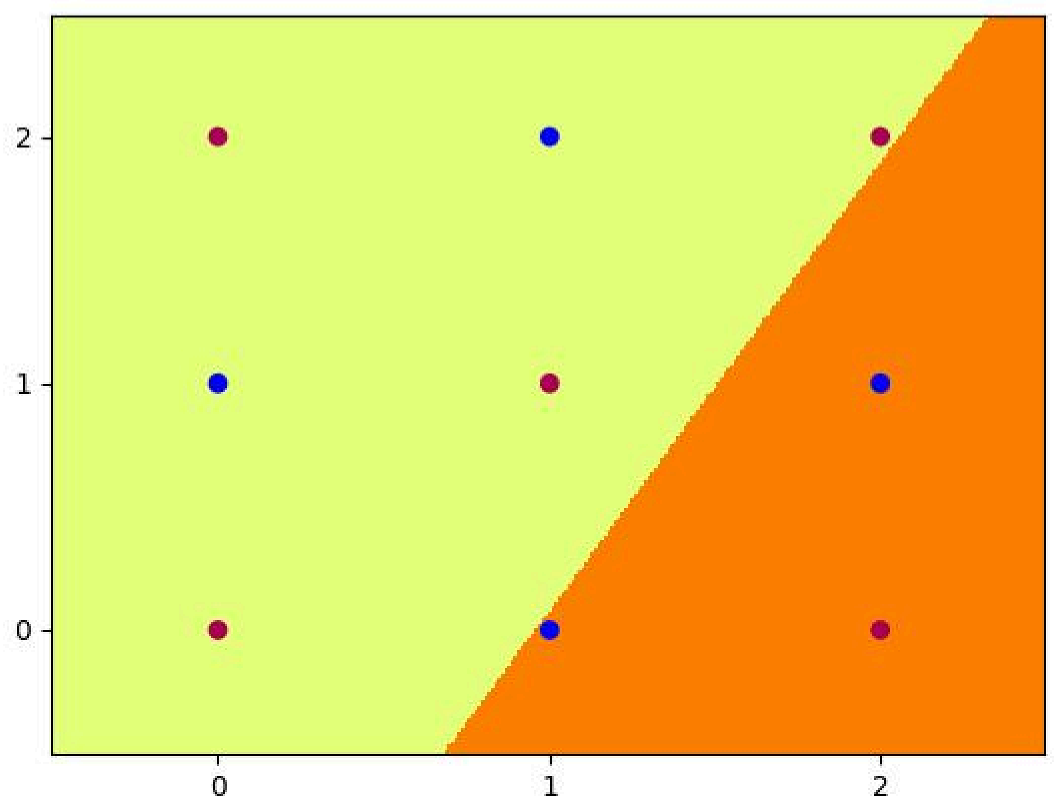
hid_4_0



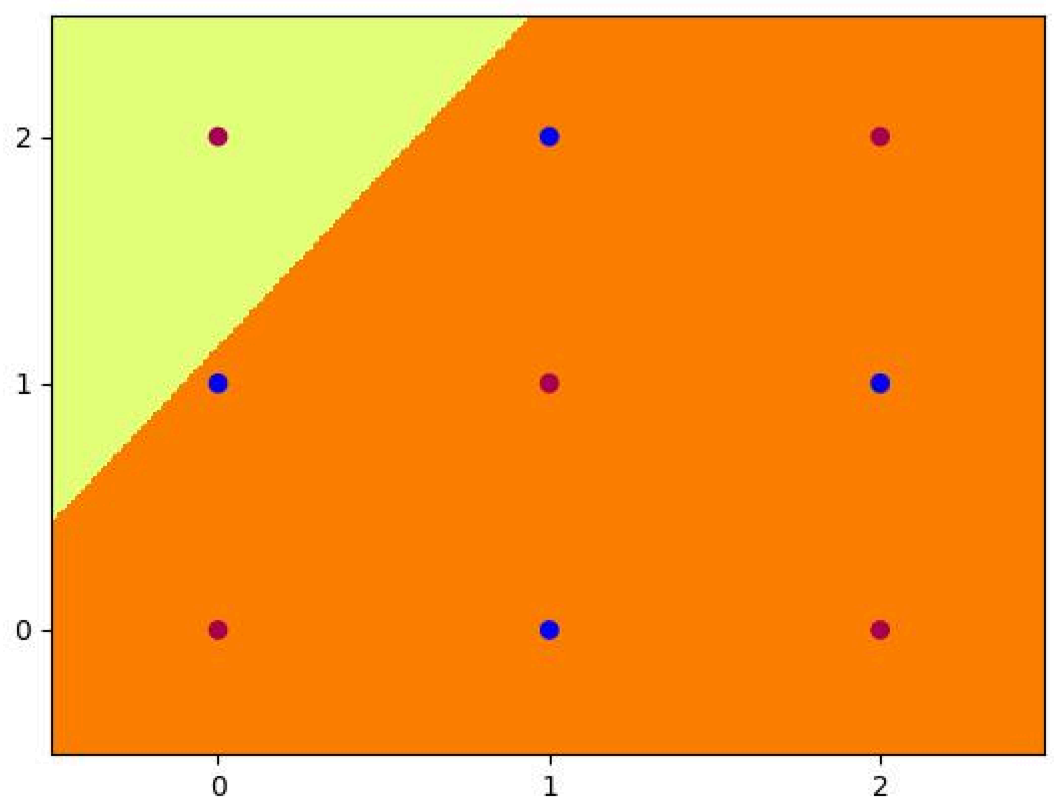
hid_4_1



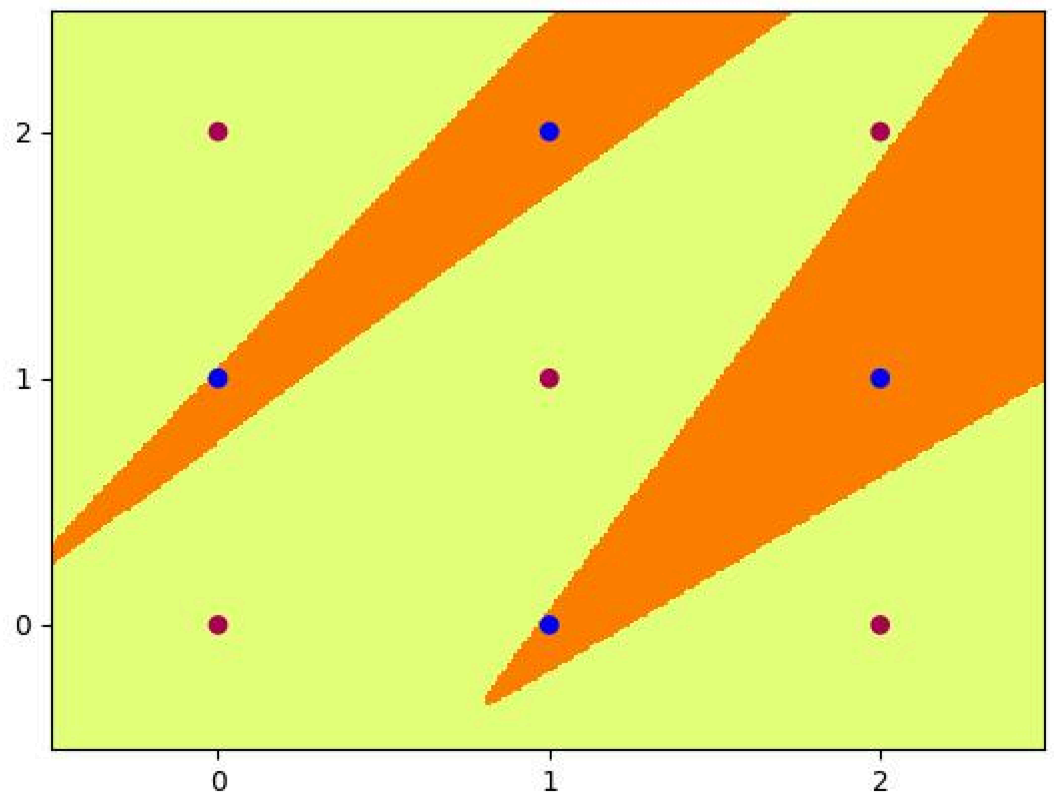
hid_4_2



hid_4_3

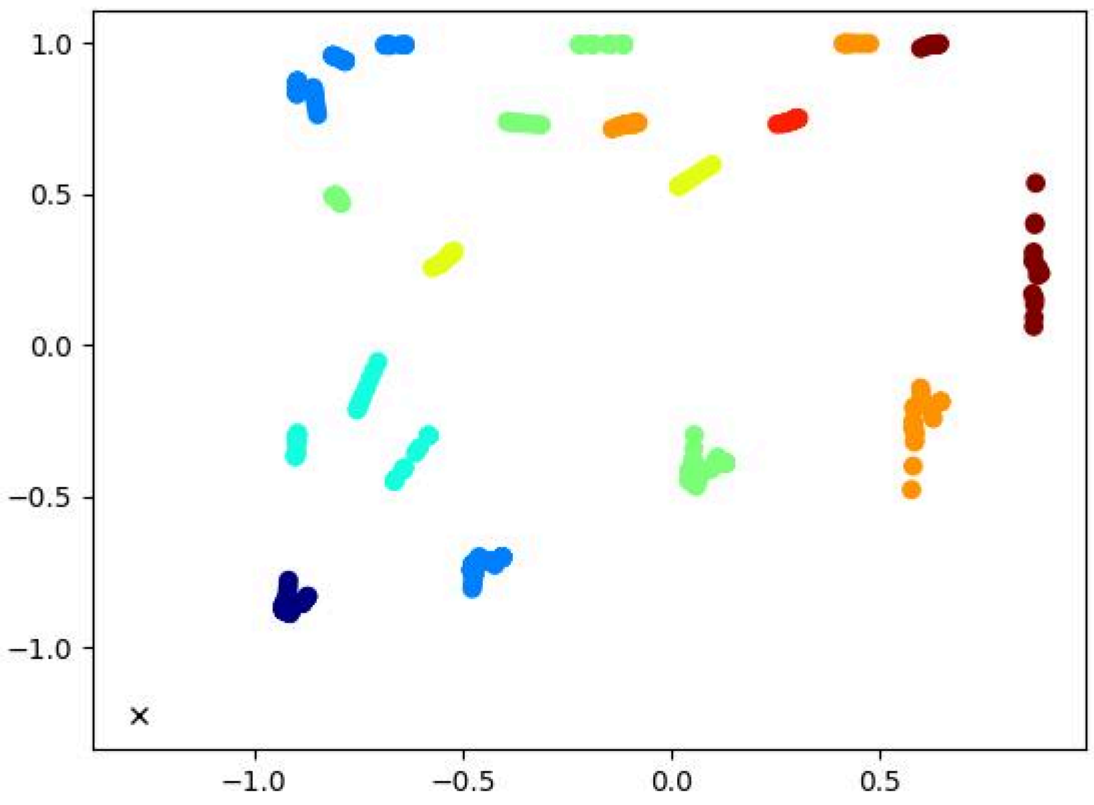


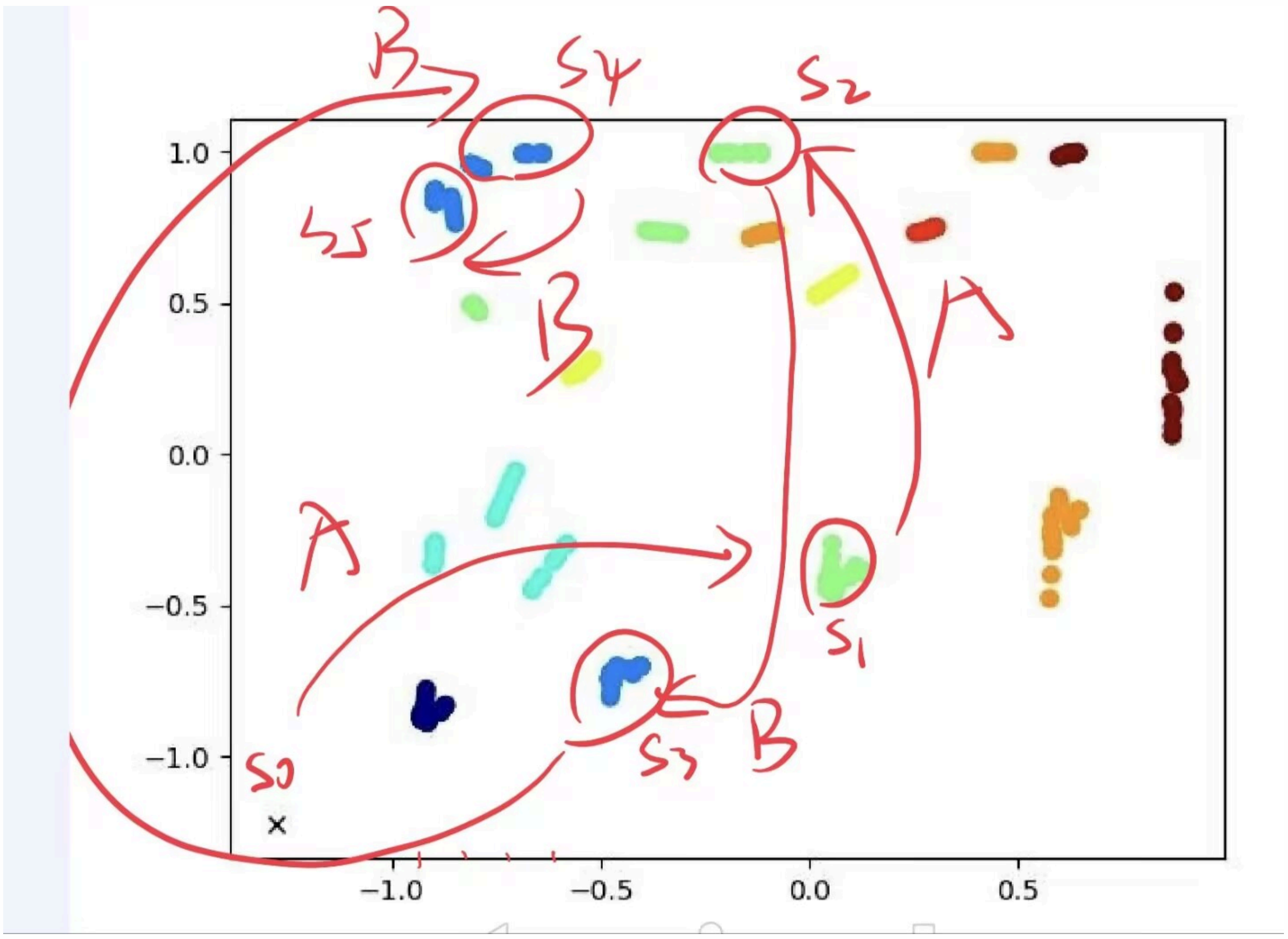
Overall classification (out_4.jpg): Shows the same diagonal band pattern as the step function network, confirming that the rescaling successfully mimics the step function behavior



Part 3: Hidden Unit Dynamics for Recurrent Networks

Part 3.1+3.2


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
×

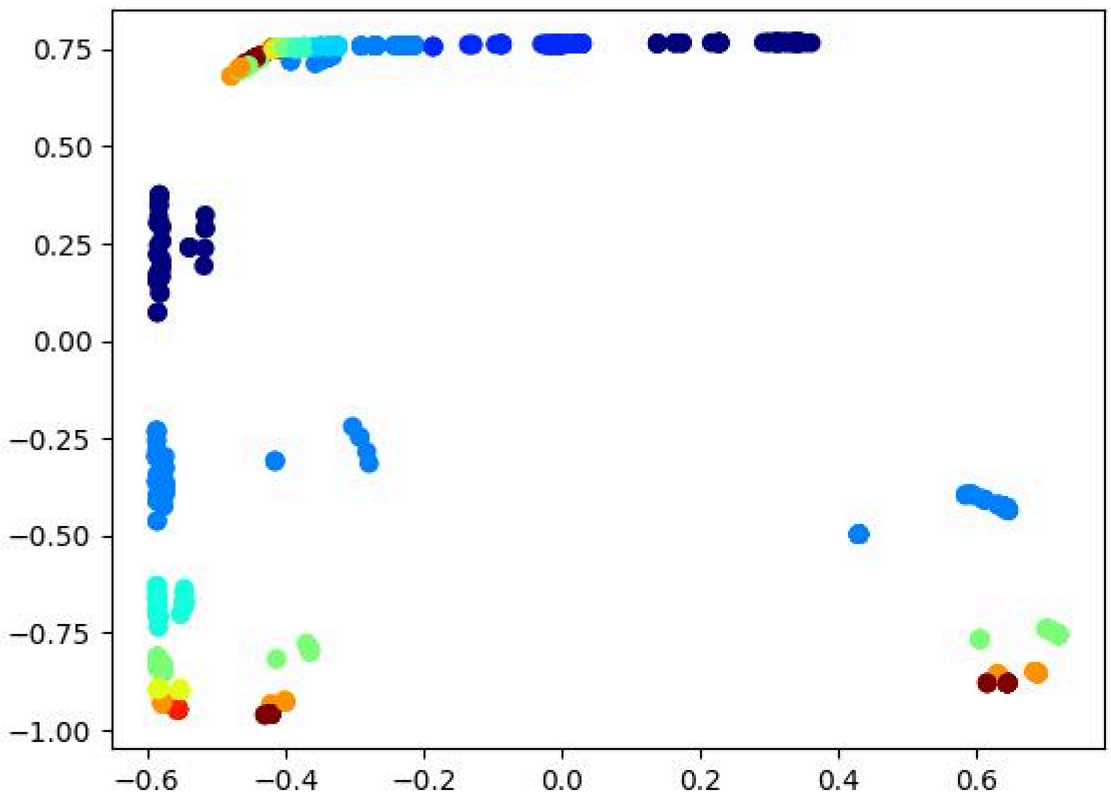


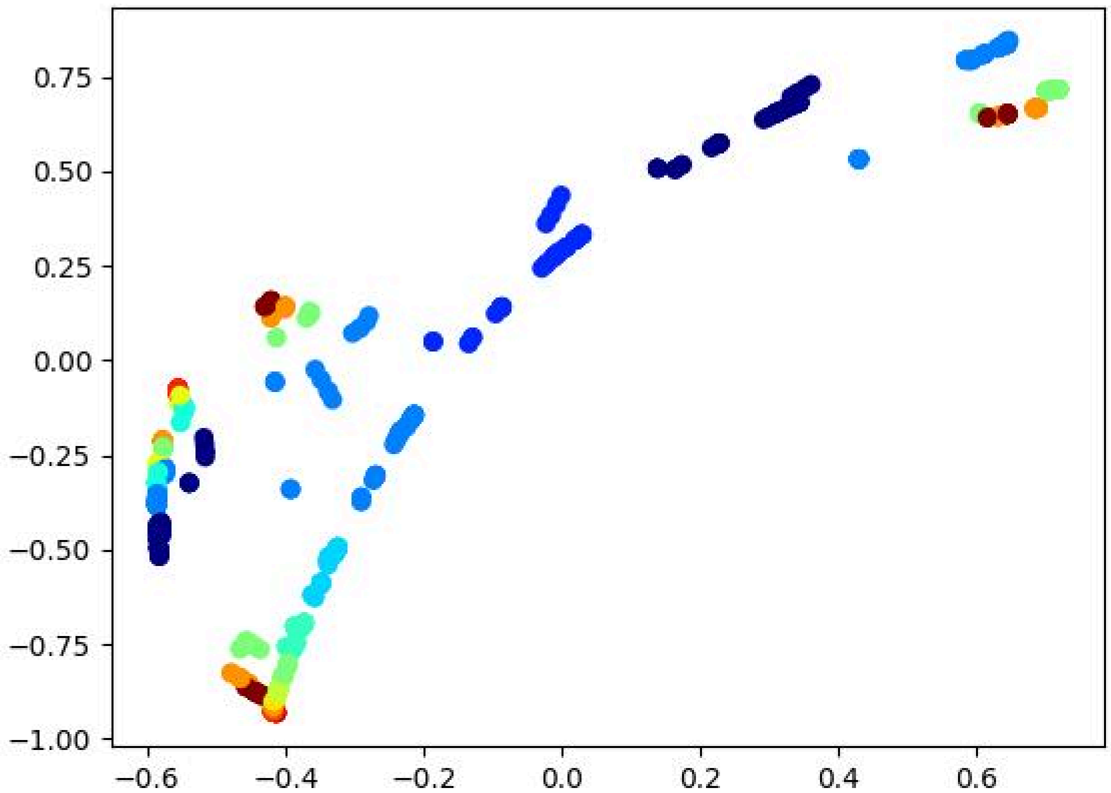
Part 3.3

When processing the *anb₂ₙ* sequence, the SRN (Simple Recurrent Network) activates its hidden units step by step along an "A-path" from the initial state S_0 to S_n, representing that it has seen *n* A's. Upon encountering the first B, it switches from the "S_B start" state to a separate "B-path," where it uses a finite cluster of hidden states to precisely count down the remaining $2n-1$ B's. After reading the final B, the network returns to its initial state, at which point it predicts the next A with high probability.

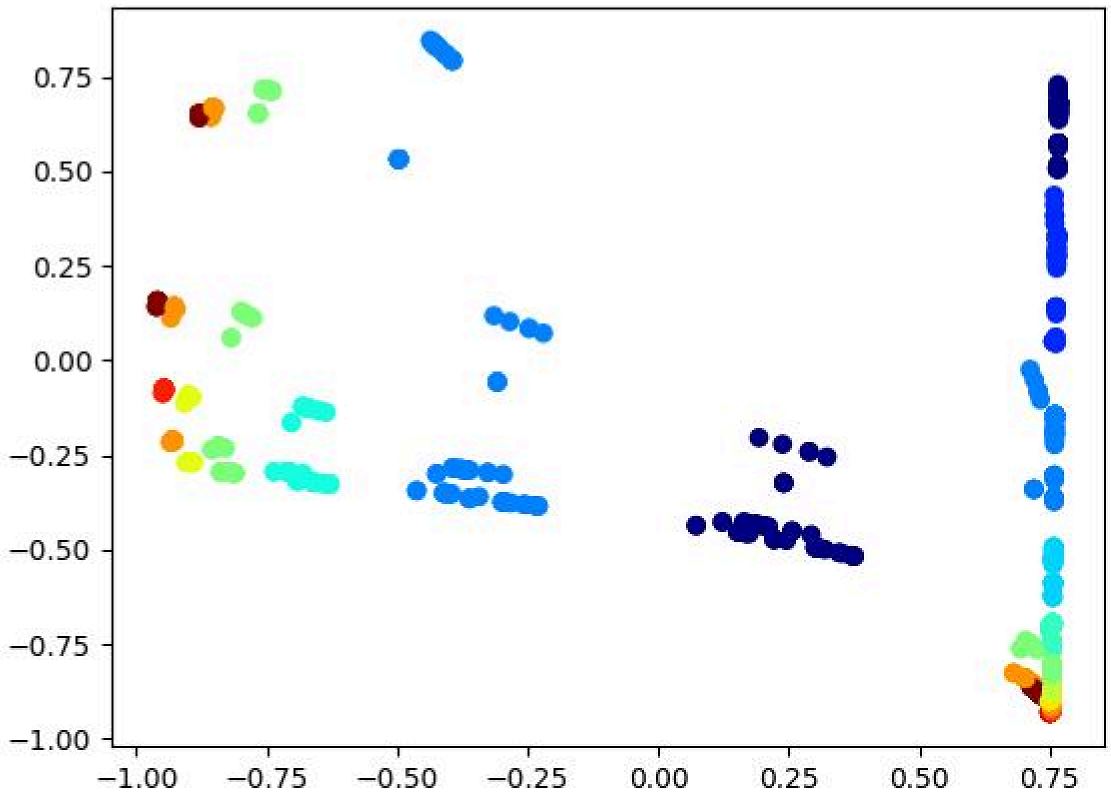
Although, in theory, recognizing *anb₂ₙ* requires a true pushdown automaton to handle arbitrarily large *n*, the SRN only needs a limited set of clustered states to "implicitly count" and make accurate predictions during training because *n* is restricted to a small range (1 to 4). However, if *n* becomes unbounded or the sequence grows longer, a simple SRN struggles to maintain such memory and would need a stack-like structure—similar to a pushdown mechanism—for proper support.

Part 3.4

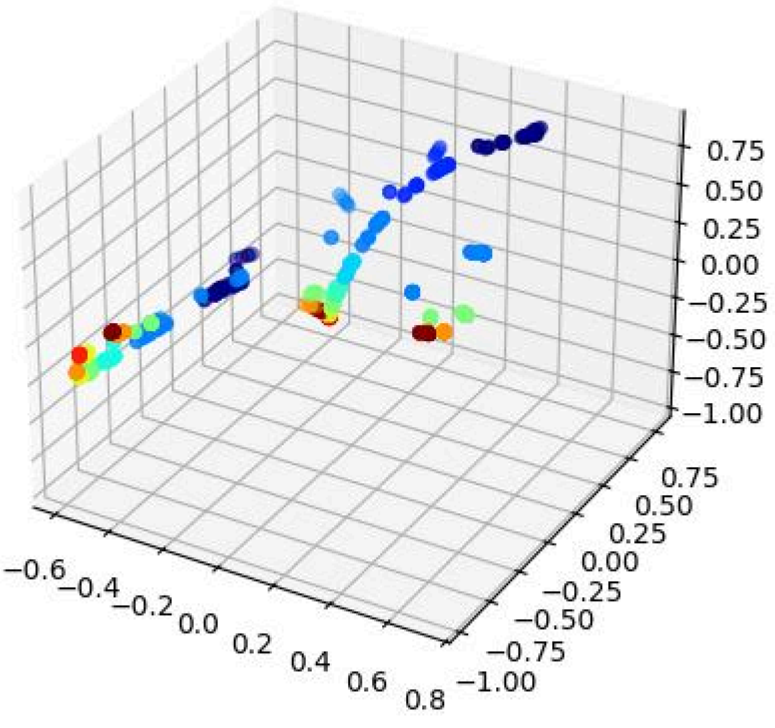




0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

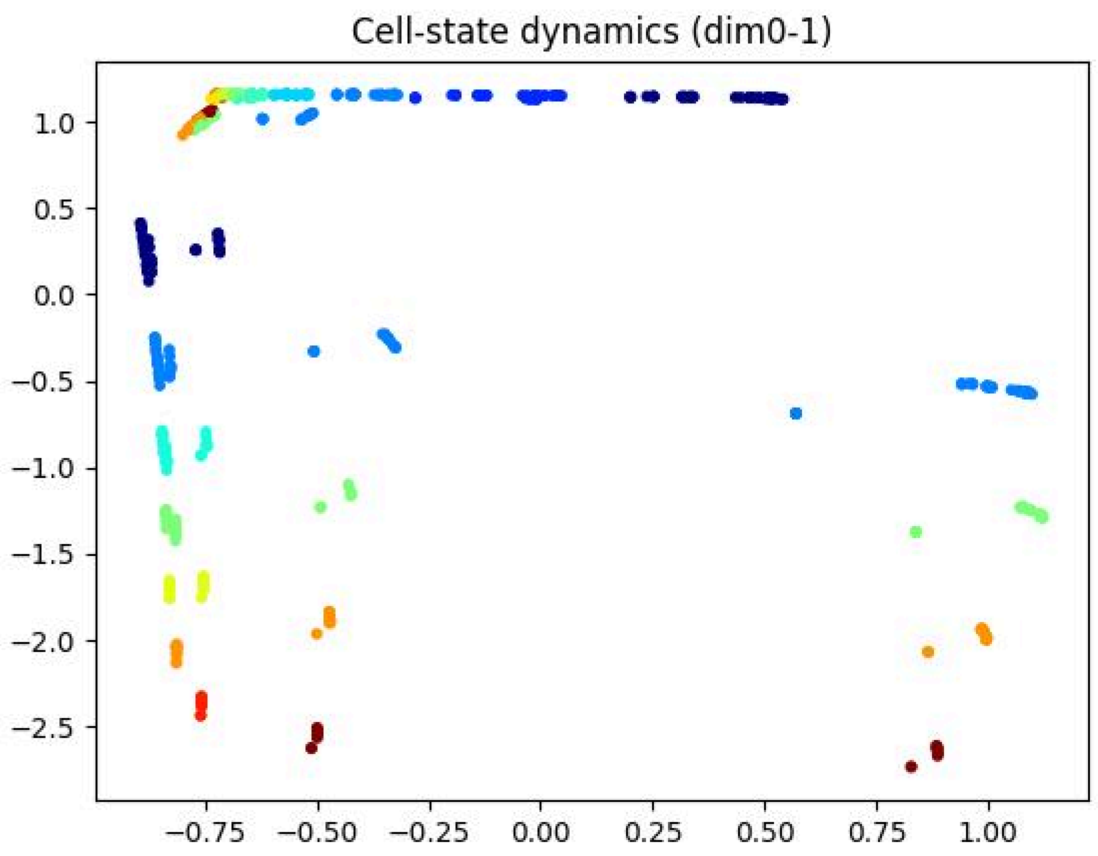


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75



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
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

Part 3.5



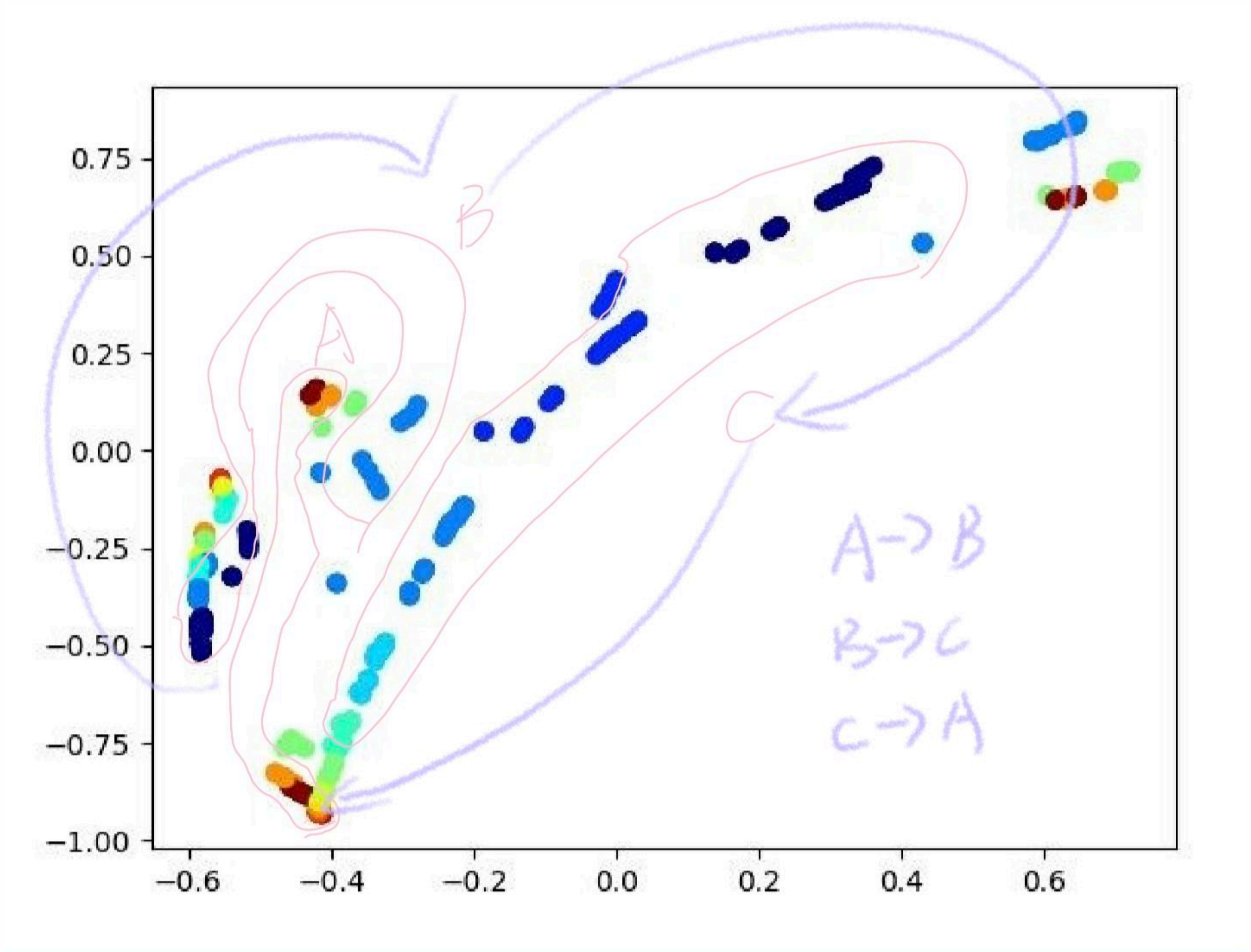
The cell states show clear horizontal stratification, with different "levels" corresponding to different sequence phases. The color progression from red/orange (early) to blue (late) indicates systematic state transitions as sequences progress.

How the LSTM Accomplishes the Task

The LSTM successfully handles the anb2nc3n prediction task through a multi-phase strategy:

1. **Counting Phase (A's):** During the initial A's, the LSTM builds an internal representation of the count n. This information is preserved in the cell state throughout the entire sequence.
2. **Production Phases (B's and C's):** The network uses the stored count to determine exactly how many B's (2n) and C's (3n) to predict. The clustering patterns show stable trajectories during these predictable phases.
3. **Phase Transitions:** Sharp movements in the state space occur at A→B and B→C transitions, indicating the network recognizes phase boundaries and adjusts its internal state accordingly.

4. **Sequence Boundaries:** The LSTM correctly predicts the return to A after exactly 3n C's, demonstrating long-term memory maintenance across sequences of length 6n.


A
B
C
A->B
B->C
C->A
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6