Part1 : Japanese Character Recognition

1.

Train Epoch: 10 [0/60000 (0%)] Loss: 0.823010

Train Epoch: 10 [6400/60000 (11%)] Loss: 0.639519

Train Epoch: 10 [12800/60000 (21%)] Loss: 0.600728

Train Epoch: 10 [19200/60000 (32%)] Loss: 0.595341

Train Epoch: 10 [25600/60000 (43%)] Loss: 0.324008

Train Epoch: 10 [32000/60000 (53%)] Loss: 0.517906

Train Epoch: 10 [38400/60000 (64%)] Loss: 0.664940

Train Epoch: 10 [44800/60000 (75%)] Loss: 0.608237

Train Epoch: 10 [51200/60000 (85%)] Loss: 0.347265

Train Epoch: 10 [57600/60000 (96%)] Loss: 0.668209

<class 'numpy.ndarray'>

[[765. 5. 8. 10. 30. 66. 2. 61. 32. 21.]

[7. 665. 108. 18. 29. 23. 58. 16. 25. 51.]

[6. 57. 691. 27. 27. 22. 47. 37. 47. 39.]

[4. 33. 60. 758. 16. 59. 13. 18. 27. 12.]

[62. 51. 80. 20. 622. 21. 33. 35. 20. 56.]

[8. 27. 120. 16. 18. 729. 29. 8. 33. 12.]

[5. 22. 147. 10. 26. 25. 722. 20. 10. 13.]

[15. 31. 28. 8. 83. 16. 54. 625. 93. 47.]

[9. 38. 96. 40. 7. 31. 43. 7. 708. 21.]

[7. 50. 82. 3. 53. 30. 19. 31. 42. 683.]]

Test set: Average loss: 1.0091, Accuracy: 6968/10000 (70%)

2.

Train Epoch: 10 [0/60000 (0%)] Loss: 0.414992

Train Epoch: 10 [6400/60000 (11%)] Loss: 0.254586

Train Epoch: 10 [12800/60000 (21%)] Loss: 0.283960

Train Epoch: 10 [19200/60000 (32%)] Loss: 0.251868

Train Epoch: 10 [25600/60000 (43%)] Loss: 0.120579

Train Epoch: 10 [32000/60000 (53%)] Loss: 0.262449

Train Epoch: 10 [38400/60000 (64%)] Loss: 0.310027

Train Epoch: 10 [44800/60000 (75%)] Loss: 0.331698

Train Epoch: 10 [51200/60000 (85%)] Loss: 0.123282

Train Epoch: 10 [57600/60000 (96%)] Loss: 0.300315

<class 'numpy.ndarray'>

[[845. 5. 2. 5. 34. 30. 3. 34. 34. 8.]

[5. 814. 31. 6. 21. 11. 61. 5. 22. 24.]

[8. 20. 826. 34. 11. 13. 30. 16. 24. 18.]

[3. 12. 29. 899. 3. 17. 8. 8. 7. 14.]

[43. 29. 27. 10. 800. 7. 25. 16. 21. 22.]

[7. 14. 73. 9. 11. 835. 22. 2. 19. 8.]

[3. 12. 52. 6. 18. 5. 886. 7. 3. 8.]

[11. 11. 26. 5. 28. 8. 46. 801. 32. 32.]

[9. 34. 25. 43. 7. 9. 25. 4. 834. 10.]

[2. 22. 48. 2. 31. 12. 25. 14. 17. 827.]]

Test set: Average loss: 0.5356, Accuracy: 8367/10000 (84%)

For the total number of independent parameters in the network

First layer(fc1): Second layer(fc2)

Input dim: 28x28 =784 95

Output dim : 95 10

Weights: 784 x 95 = 74480 95 x 10 = 950

Biases: 95 10

Subtotal: 74480 + 95 =74575 950 + 10 = 960

Total fc1+fc2=74575 + 960 =75535

So the total number of independent parameters is 75535.

3. Train Epoch: 10 [0/60000 (0%)] Loss: 0.014743

Train Epoch: 10 [6400/60000 (11%)] Loss: 0.034772

Train Epoch: 10 [12800/60000 (21%)] Loss: 0.099488

Train Epoch: 10 [19200/60000 (32%)] Loss: 0.024146

Train Epoch: 10 [25600/60000 (43%)] Loss: 0.020688

Train Epoch: 10 [32000/60000 (53%)] Loss: 0.063270

Train Epoch: 10 [38400/60000 (64%)] Loss: 0.031177

Train Epoch: 10 [44800/60000 (75%)] Loss: 0.201449

Train Epoch: 10 [51200/60000 (85%)] Loss: 0.019592

Train Epoch: 10 [57600/60000 (96%)] Loss: 0.035505

<class 'numpy.ndarray'>

[[953. 2. 0. 0. 24. 1. 0. 9. 7. 4.]

[3. 906. 11. 1. 14. 1. 39. 7. 4. 14.]

[11. 5. 895. 33. 3. 8. 14. 10. 5. 16.]

[3. 0. 11. 959. 2. 8. 10. 1. 1. 5.]

[17. 7. 2. 6. 936. 1. 12. 4. 13. 2.]

[5. 6. 33. 6. 4. 914. 16. 3. 4. 9.]

[3. 2. 9. 3. 10. 4. 965. 4. 0. 0.]

[1. 1. 5. 0. 4. 0. 7. 956. 6. 20.]

[3. 12. 6. 0. 12. 3. 9. 5. 940. 10.]

[3. 6. 5. 1. 9. 0. 10. 3. 8. 955.]]

Test set: Average loss: 0.2427, Accuracy: 9379/10000 (94%)

1. conv1 (in=1, out=16, 3×3)

   Weights: 1×16×3×3=144

   Biases: 16

   Subtotal: 144 + 16 = 160

2. conv2 (in=16, out=32, 3×3)

   Weights: 16×32×3×3=4608

   Biases: 32

   Subtotal: 4 608 + 32 = 4 640

3. fc1 (in=32×7×7=1 568, out=128)

   Weights: 1 568×128=200704

   Biases: 128

   Subtotal: 200704 + 128 = 20832

4. fc2 (in=128, out=10)

   Weights: 128×10=1280

   Biases: 10

   Subtotal: 1280 + 10 = 1290

Total parameters:

160+4640+200832+1290=206922

So the total number of independent parameters is 206922.

4.

a) Relative accuracy of three models.Expected accuracy ranking (from highest to lowest):

NetConv (highest) - Convolutional networks are designed specifically for image data and can capture spatial hierarchy and local patterns

NetFull (Medium) - A two-layer fully connected network with nonlinear activation can learn more complex patterns than linear models

NetLin (lowest) - Simple linear classifiers can only learn linearly separable patterns.

b) 2. Number of Independent Parameters

NetLin:

FC layer: (28×28) × 10 + 10 = 784 × 10 + 10 = 7,850 parameters

NetFull:

FC1: (28×28) × 95 + 95 = 784 × 95 + 95 = 74,575

FC2: 95 × 10 + 10 = 960

Total: 75,535 parameters

NetConv:

Conv1: (1×3×3) × 16 + 16 = 160

Conv2: (16×3×3) × 32 + 32 = 4,640

FC1: (32×7×7) × 128 + 128 = 200,832

FC2: 128 × 10 + 10 = 1,290

Total: 206,922 parameters

c)

Dataset characters:

あ (a), き (ki), す (su), つ (tsu), な (na), は (ha), ま (ma), や (ya), れ (re), を (wo)

NetLin (Linear Model)

Highest confusion pairs:

あ (a) ↔ お (o): Both contain circular strokes, so a purely linear model struggles to distinguish subtle curve differences.

き (ki) ↔ さ (sa): Shared vertical strokes with short horizontal elements.

す (su) ↔ む (mu): Very similar looping curves.

は (ha) ↔ ほ (ho): Nearly identical left-hand components.

Reason:

Linear classifiers can only form pixel-wise weighted sums and cannot capture stroke continuity or the overall spatial structure of each character.

NetFull (Fully Connected Network)

Moderate confusion pairs:

あ (a) ↔ お (o): Still the dominant confusion.

き (ki) ↔ さ (sa): Reduced but not eliminated.

つ (tsu) ↔ う (u): Both have similarly shaped arcs.

Reason:

Nonlinear activations allow richer feature combinations, but every pixel is still treated independently—spatial relationships remain implicit rather than explicitly modeled.

NetConv (Convolutional Network)

Lowest confusion pairs (minor remaining errors):

あ (a) ↔ お (o): A few misclassifications persist on this very similar pair.

き (ki) ↔ さ (sa): Dramatically improved, though occasional mistakes may occur.

Reason:

Convolutional layers learn local stroke patterns and their continuity, and pooling provides some spatial invariance—so the network better recognizes how strokes fit together to form each character.

Part 2 Multi-Layer Perceptron

1.

Final Weights:

tensor([[0.0783, -4.2822],

[2.9355, -0.0858],

[2.7455, 2.7451],

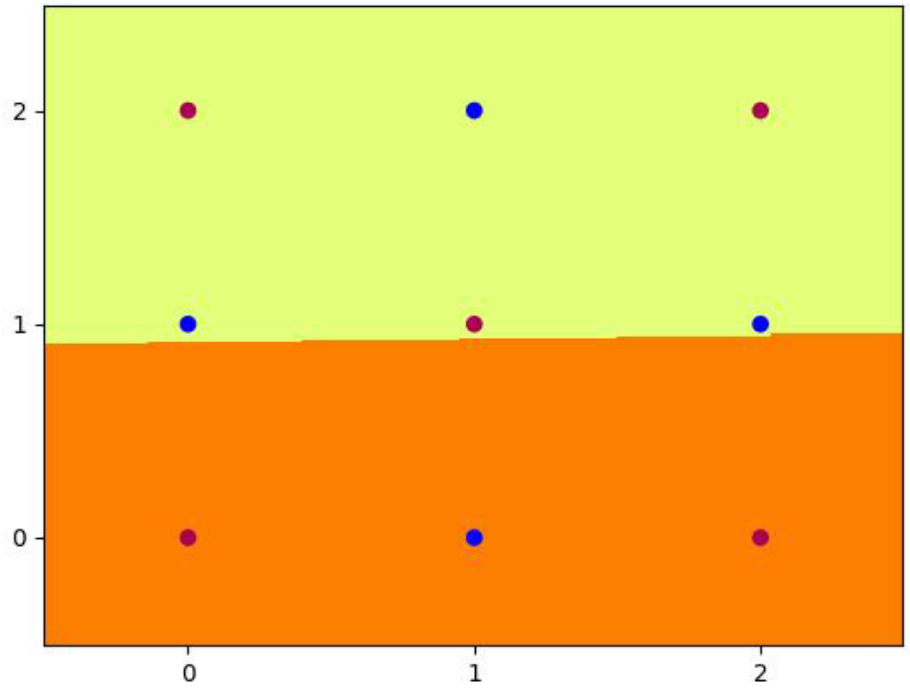
[5.5703, 5.5358],

[2.7757, 2.7757]])

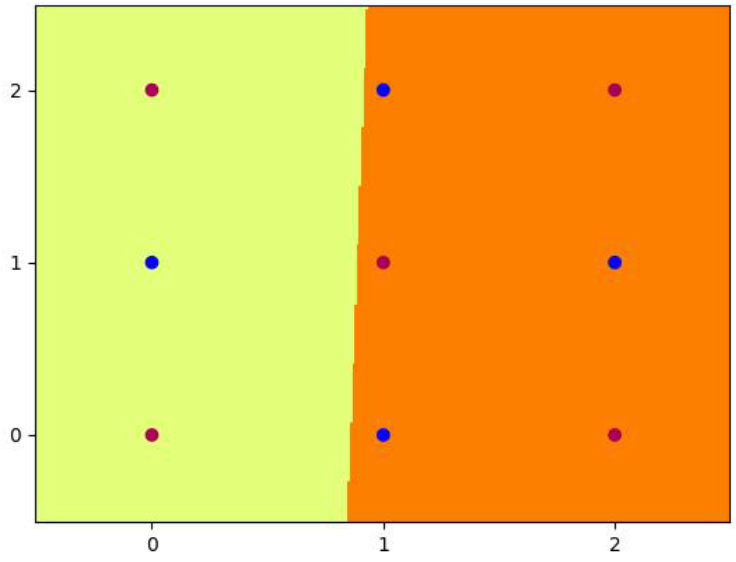
tensor([3.9085, -2.5182, -11.0748, -7.6629, -11.1676])

tensor([[-9.6010, 10.3991, -10.4415, -8.6959, -10.3989]])

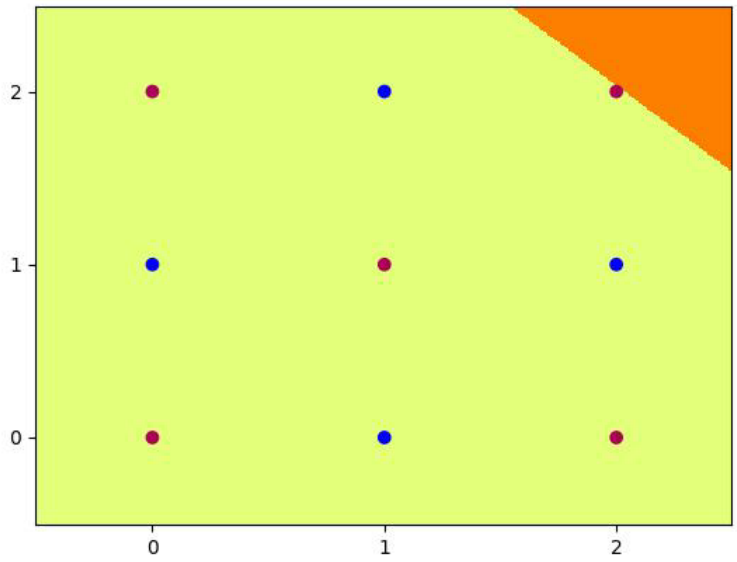
tensor([5.7844])



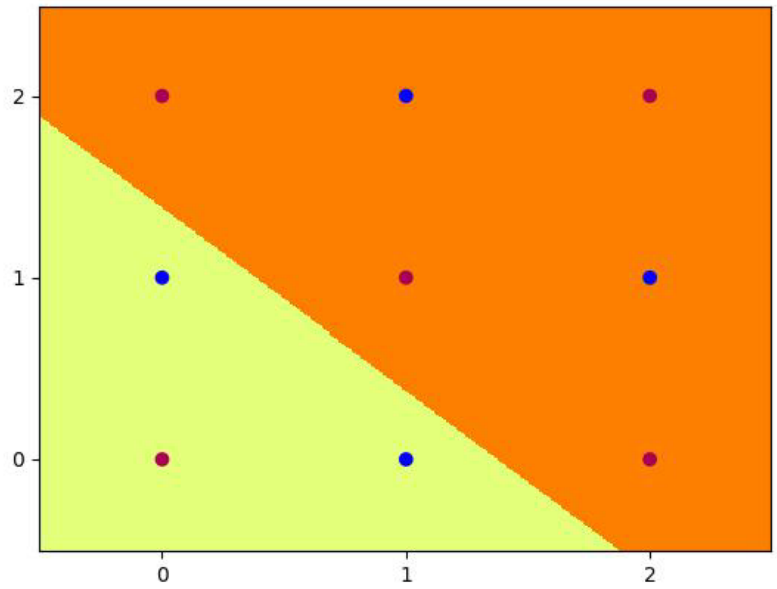
hid_5_0.jpg



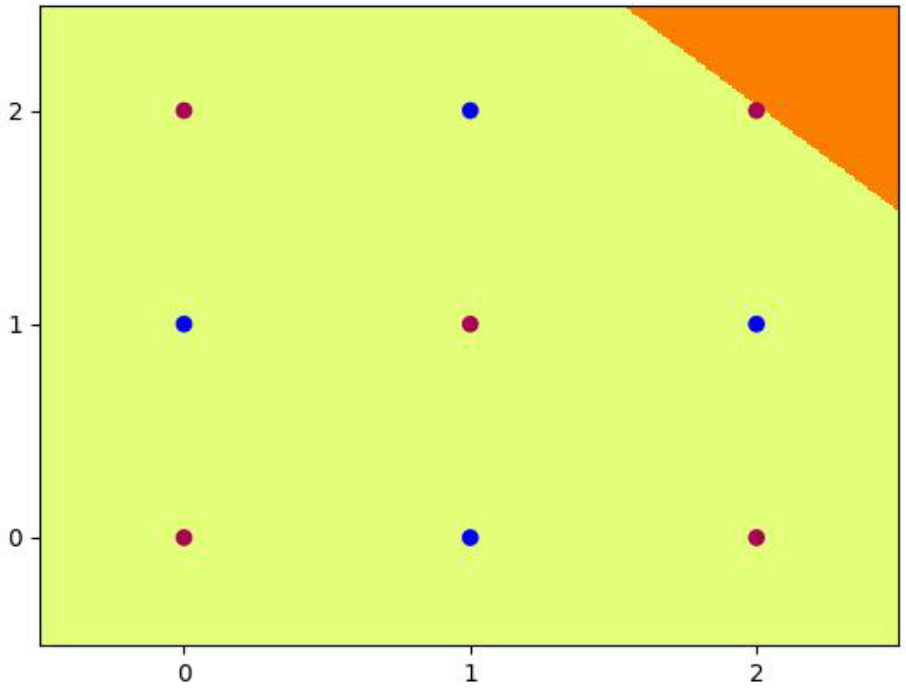
hid_5_1.jpg



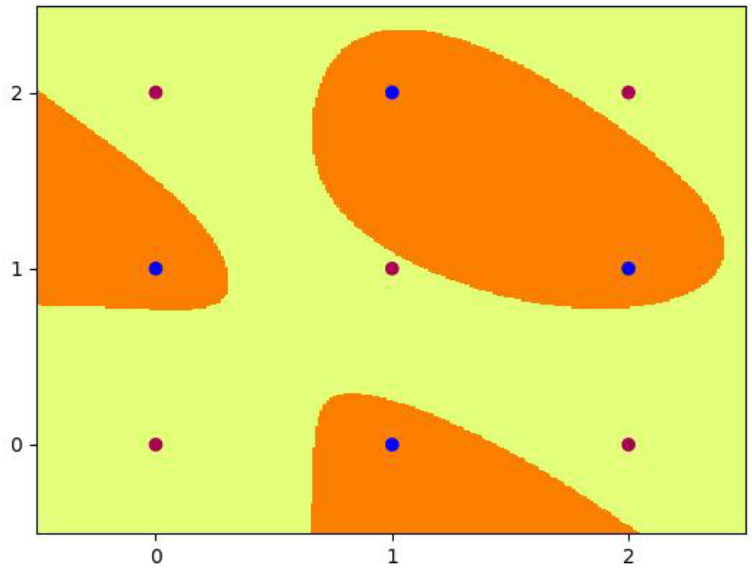
hid_5_2.jpg



hid_5_3.jpg



hid_5_4.jpg



Out_5.jpg

2. The 4 hidden nodes can be regarded as four dividing lines, and the formula is as follows:

$$x + y - 1.5 = 0 \quad (1)$$

$$x + y - 2.5 = 0 \quad (2)$$

$$x - y + 0.5 = 0 \quad (3)$$

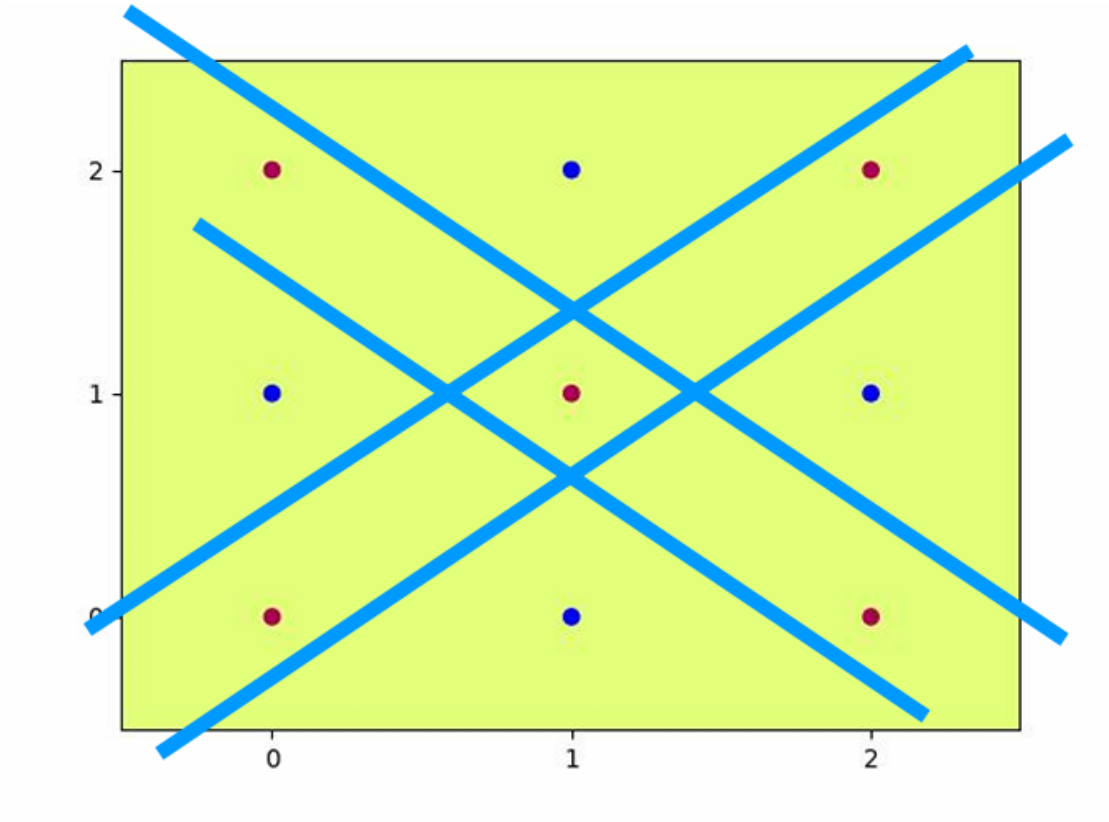
$$x - y - 0.5 = 0 \quad (4)$$

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

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

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

out_bias = [0]



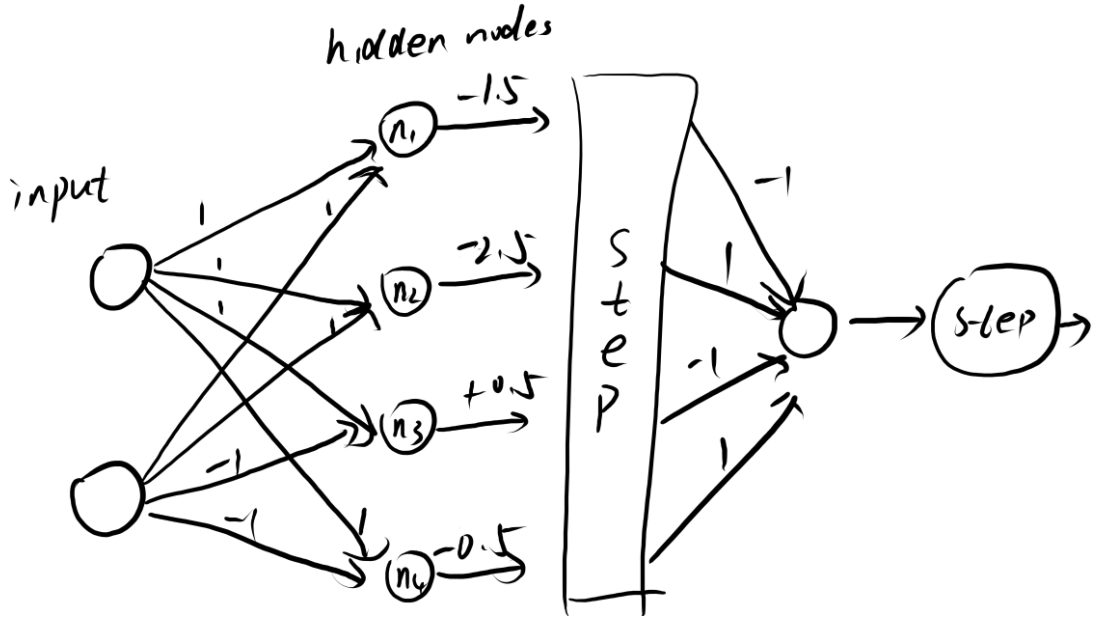


Table of activations of all the hidden nodes and the output node:

Input	Hidden Nodes	After Step Function	Output	Output Step
(0,0)	(-1.5,-2.5,0.5,-0.5)	(0,0,1,0)	-1	0
(0,1)	(-0.5,-1.5,-0.5,-1.5)	(0,0,0,0)	0	1

(0,2)	(0.5,-0.5,-1.5,-2.5)	(1,0,0,0)	-1	0
(1,0)	(-0.5,-1.5,1.5,0.5)	(0,1,0,1)	2	1
(1,1)	(0.5,-0.5,0.5,-0.5)	(1,0,1,0)	-2	0
(1,2)	(1.5,0.5,-0.5,-1.5)	(1,1,0,0)	0	1
(2,0)	(0.5,-0.5,2.5,1.5)	(1,0,1,1)	-1	0
(2,1)	(1.5,0.5,1.5,0.5)	(1,1,1,1)	0	1
(2,2)	(2.5,1.5,0.5,-0.5)	(1,1,1,0)	-1	0

3.

Final Weights:

tensor([[-6.5513, 7.3641],

[5.8746, 7.8048],

[2.1092, -1.5514],

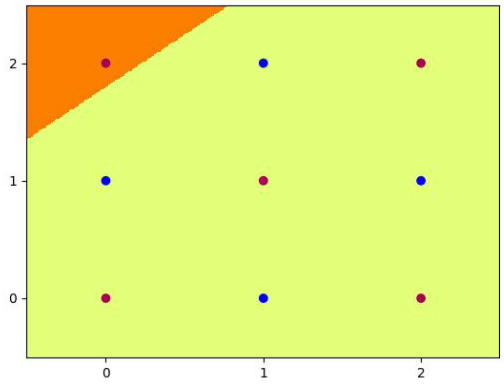
[12.9947, -12.9068]])

tensor([-13.2312, -16.8358, -0.9344, -6.4030])

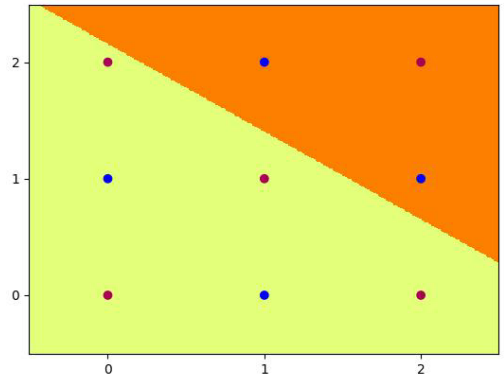
tensor([[-12.8772, 3.9815, -20.9687, 14.6100]])

tensor([3.5993])

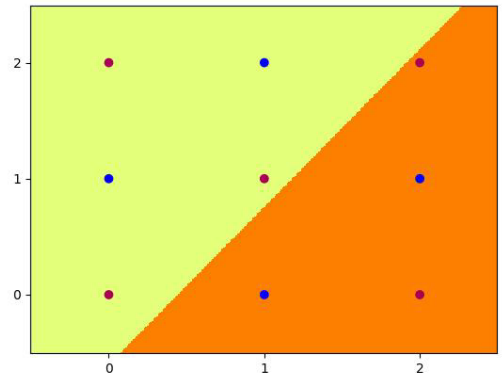
Final Accuracy: 100.0



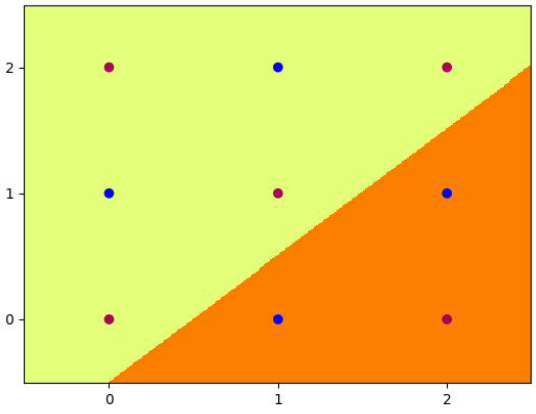
Hid_4_0



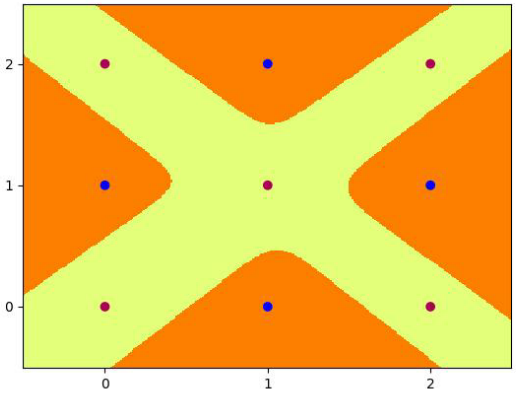
Hid_4_1



Hid_4_2



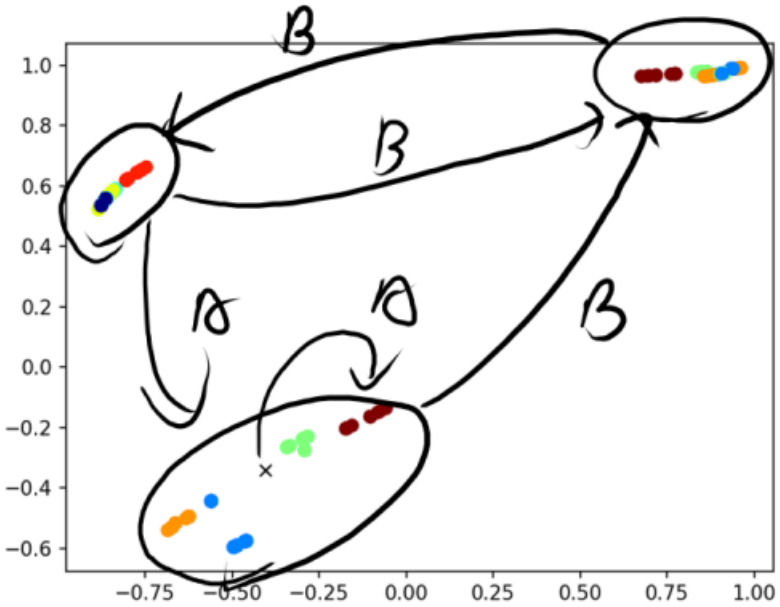
Hid_4_3


2
1
0
0
1
2

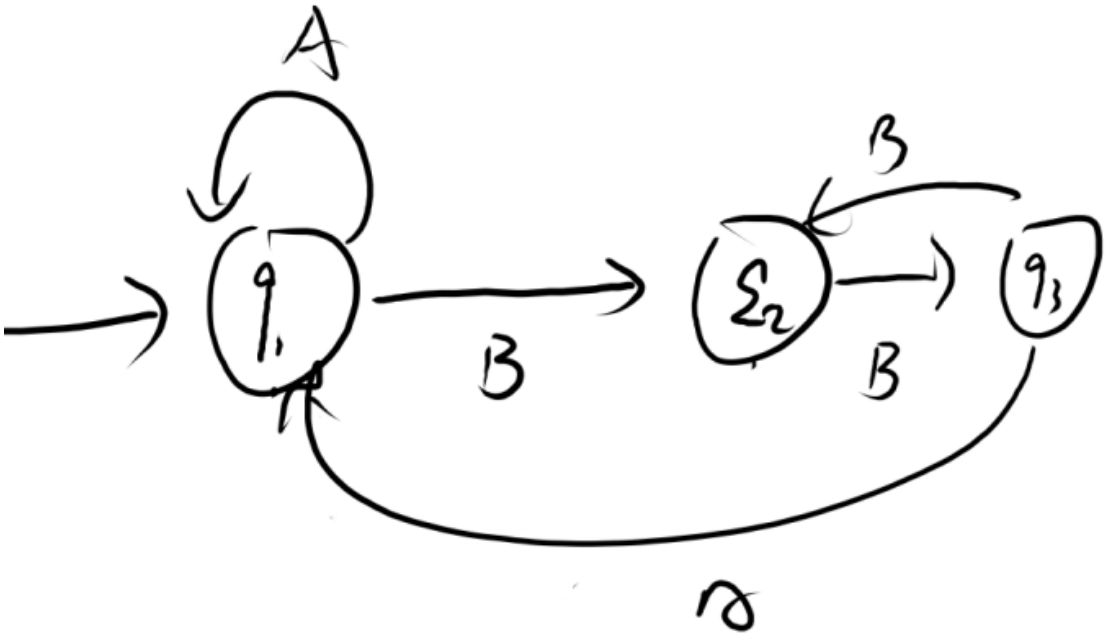
Out_4

Part3

1.

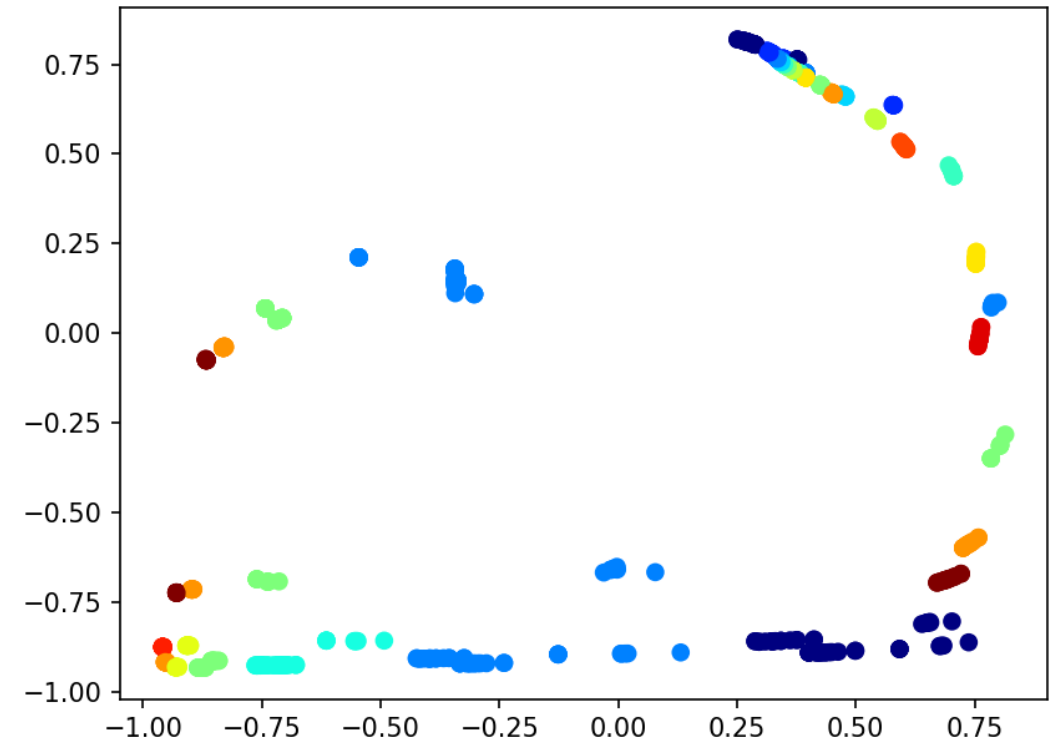

B
B
A
A
B
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

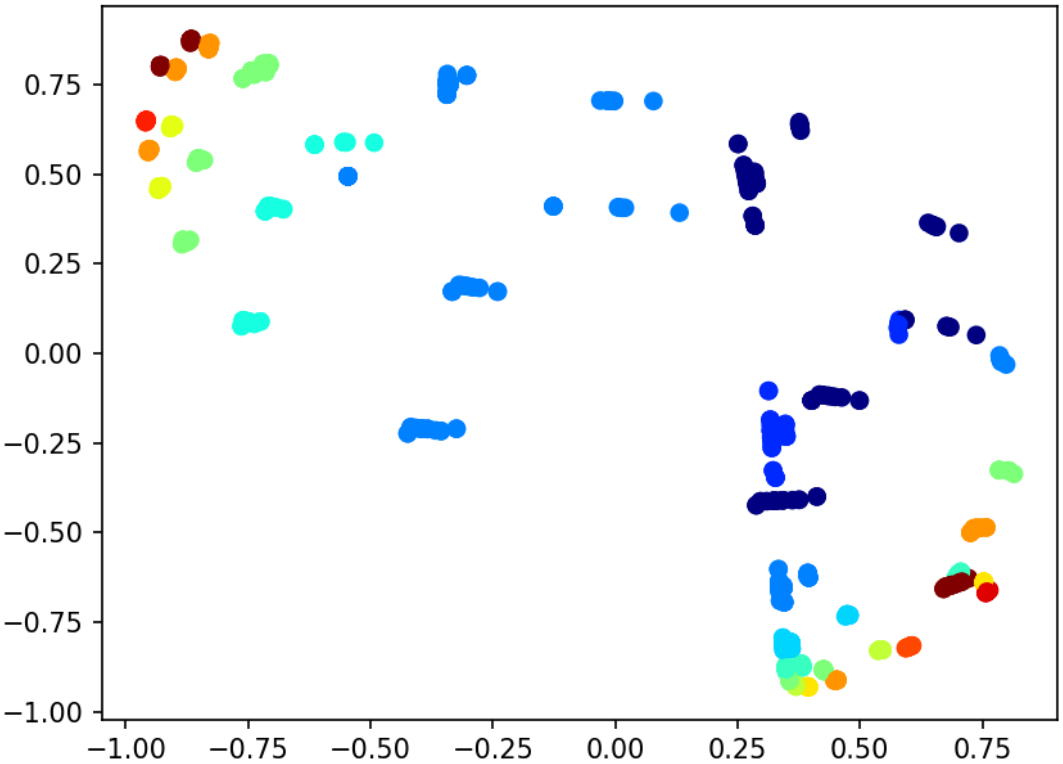
2.

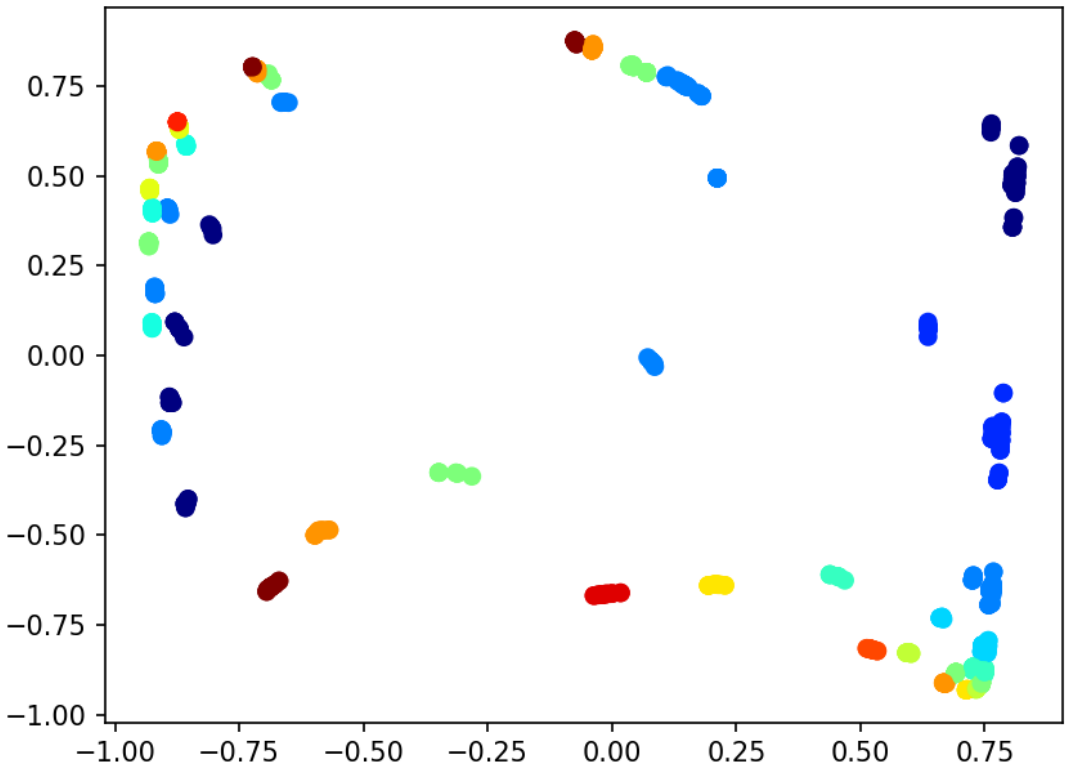


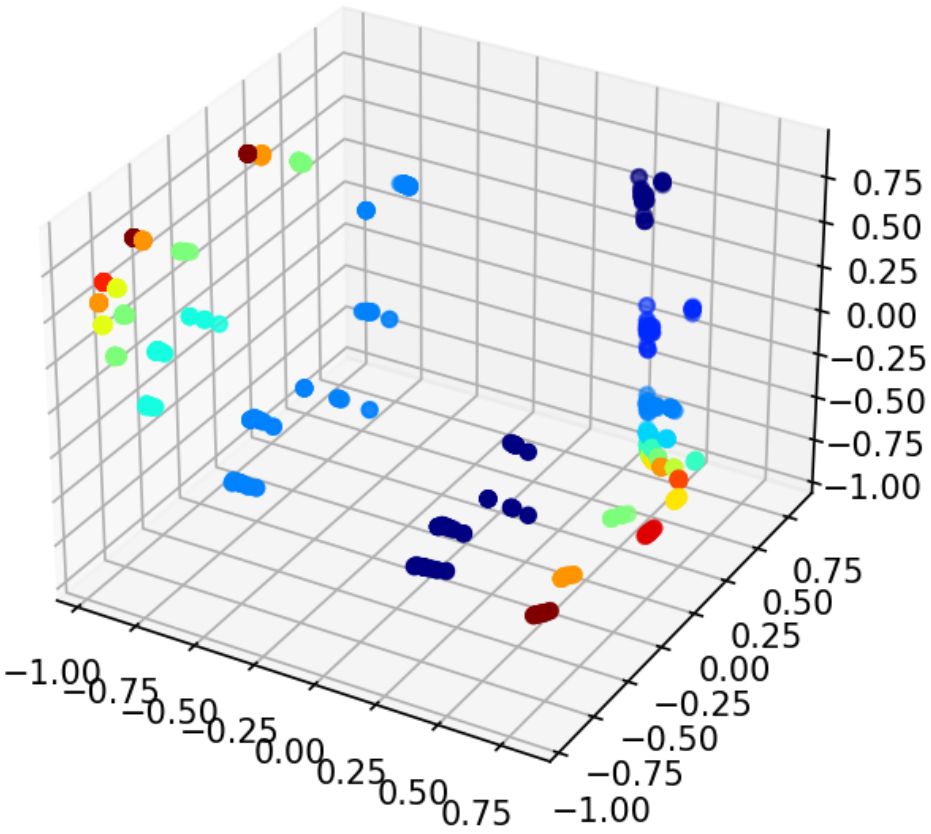
3. Initially, the probability of A is much greater than that of B. After the first A appears, the probability of A gradually decreases, while the probability of B gradually increases. After the first B appears, the probability of B is close to 1, while the probability of A is close to 0. As the number of Bs increases, the probability of B gradually decreases, while the probability of A gradually increases, and this cycle repeats.

4.


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

5. We investigate the formal language A^n B^2n C^3n, where each string contains n A, followed by 2n B and 3n C. The 3D LSTM network creation of language A, B, C requires the network to adequately identify the moment when this sentence takes place from letter to letter: from A to B and from B to C. It must precisely establish how many A are there in total to form double the amount of B and three times as much C. The neural network needs to sustain this tally for a comprehensive time through the C stage, though without erasing relevant data. Indeed, the hidden layer of the LSTM is small; however, it should cover several counter-independent issues, cope with sharp state changes at the boundaries, and preserve long-range dependencies. Such a system can therefore correctly identify the next symbol even despite incorporating hundreds of time steps into its state.

1. Hidden State: Go-Between of the Short-Term Connections

The hidden states clearly cluster during the training. The representations hike onto the same hidden variables for the same grammar phase, which proves that the LSTM correctly understands the transition rules between phases. The model's hidden state features the tanh restriction to the range [-1, 1] and ensures a balance between rich info encoding and protection against exploding senders. The different clustering of the state makes the hidden state able to discriminate phases of the input sequence from each other so that the next symbol to be predicted can be determined using the short-term context.

2. Cell State: Storehouse of the Permanent Memory

Its width is significantly greater (up to approximately [-4,5]) than the hidden state, and notable jumps depict critical points of transitions (e.g., A→B). Therefore, these jumps are observed as a new passage of long-term information. A few cell-state values are drifting out of the acceptable range for |3| (less than -3 or greater than 3), which might mean preservation of the exclusive count information, but they also indicate the possibility of a gradient explosion, which would be handled through gradient clipping. Smooth transitions (during repeated signs) combined with sharp fluctuations (when associated phases change) equip the LSTM with a tool to disentangle nested grammatical units that vary in complexity.

3. Gate Cooperation: Supplementation, Recording, and Reading

Forget Gate (ft): The disturbance of the current phase matching pattern with the first B after many settlers can trigger ft, and it will be obliged to decrease quickly from about 1 to consider the blank space at the phase. The forget gate of the LSTM removes elements that have no relation to the task at hand. This is to say, the LSTM can be seen in a way that every time an A is provided as an input, the respective count of A's will be in the memory. When the first B has entered, therefore, it is B's counting while it is still not erasing the memory of the letter A it has. It uses input C to keep the original memory of the total counts of A's and B's, then uses these cumulative memories to foresee following counts of B's and C's. Input Gate + Candidate (it, gt): As the second hand strikes the hour, the gate (input) opens that together with the current state (candidate) adds an appropriate increment to the cell (i.e., “+1” or “+2”) so that the count of the new symbol is stored. Output Gate (ot): The output ot is produced via ht = ot · tanh(ct) and allows the cell state (long-term count) to have a regulated impact over the hidden

state, providing an adjusted context at the short-term level. The working principle of the gates is that each time step, all three of them jointly go to preserve the long-term counting in the cell state as well as maintain the short-term phase recognition in the hidden state. The final take is to be a Softmax of the next-symbol probabilities.

4. Key Conclusions

With the help of gating tools, the LSTM victoriously learns non-linear languages a b^2 c^3 while it accurately identifies and moves among the recurrent phases. LSTM cell state were the exact “counter” determining the right ratio 1:2:3 of letters A’s, B’s, and C’s, while the hidden states presented the stable clusters for each symbol of the input. Training comes out to a close loss of approximately 0.012, and the output assignment agrees really well with the acceptance of ground truth distribution. This assurance brings forth the notion of both successful knowledge acquisition and great generalization.

Code change

1. seq_models.py In class LSTM_model.forward

Initialize a cell_seq container to hold each time-step’s cell state.

Store every hidden cell state into cell_seq as the LSTM runs.

Return cell_seq alongside the usual outputs so that downstream code can access and visualize the LSTM’s internal cell states.

2. seq_train.py

Line 87 & 102: During the forward pass, **extract** the returned cell_seq from the model so it’s available after each batch.

Line 107: **Print** current cell_seq

3. seq_plot.py

Line 68: **Record** the loaded cell_seq data into a structure suitable for plotting.

Line 81: **Save** the raw cell_seq values to disk (

Line 100: Add a **2D visualization** routine

Line 114: Add a **3D visualization**