Part1

1.1

```
[[764.   5.   8.  13.  31.  66.   2.  60.  31.  20.]
 [  7. 670. 108.  18.  29.  23.  57.  13.  25.  50.]
 [  7.  61. 687.  27.  27.  20.  47.  37.  48.  39.]
 [  3.  35.  61. 757.  16.  56.  13.  18.  28.  13.]
 [ 60.  53.  76.  21. 625.  18.  33.  38.  21.  55.]
 [  8.  27. 126.  16.  19. 726.  28.   8.  32.  10.]
 [  5.  21. 148.  10.  27.  24. 722.  21.  10.  12.]
 [ 17.  28.  27.  10.  83.  18.  53. 626.  88.  50.]
 [ 10.  36.  95.  40.   8.  30.  47.   6. 708.  20.]
 [  9.  53.  86.   3.  53.  32.  19.  29.  41. 675.]]

Test set: Average loss: 1.0094, Accuracy: 6960/10000 (70%)
```

1.2

```
[[843.   4.   1.   6.  31.  33.   3.  41.  30.   8.]
 [  6. 812.  37.   3.  17.  10.  66.   5.  15.  29.]
 [  8.  20. 821.  45.  10.  20.  24.  15.  22.  15.]
 [  5.  12.  32. 898.   2.  15.  12.   5.   8.  11.]
 [ 43.  32.  27.  10. 783.   8.  33.  20.  20.  24.]
 [  7.   7.  73.  13.  11. 832.  31.   3.  13.  10.]
 [  3.  10.  54.   6.  18.   7. 889.   7.   2.   4.]
 [ 13.  13.  26.   4.  22.  12.  41. 808.  31.  30.]
 [  9.  26.  24.  47.   3.  12.  30.   6. 838.   5.]
 [  5.  22.  42.   4.  27.   8.  19.  15.  15. 843.]]

Test set: Average loss: 0.5369, Accuracy: 8367/10000 (84%)
```

The total number of independent parameters in NetFull is calculated as:

Input-to-hidden:

weights: 784 × 100 = 78400

biases: 100

Hidden-to-output:

weights: 100 × 10 = 1000

biases: 10

Hence, total parameters = 78400 + 100 + 1000 + 10 = 79510.

1.3

```
[[955.   2.   1.   0.  27.   2.   0.  10.   2.   1.]
 [  4. 915.   5.   0.   7.   1.  42.   8.   6.  12.]
 [ 12.   8. 900.  15.   5.   7.  32.  10.   4.   7.]
 [  1.   3.  18. 948.   2.  10.   9.   2.   2.   5.]
 [ 21.   9.   3.   6. 922.   1.  12.   8.  13.   5.]
 [  3.   8.  45.   5.   3. 899.  21.   2.   3.  11.]
 [  3.   2.  10.   1.   8.   2. 966.   6.   0.   2.]
 [  4.   0.   1.   0.   6.   0.   3. 968.   4.  14.]
 [  5.  25.   9.   0.  16.   1.   7.   7. 925.   5.]
 [  8.   6.   8.   2.  16.   2.   9.  10.   6. 933.]]

Test set: Average loss: 0.2587, Accuracy: 9331/10000 (93%)
```

The total number of independent parameters in NetConv is calculated as:

Conv1:

weights: 3×3×1×16 = 144

biases: 16

total: 160

Conv2:

weights: 3×3×16×32 = 4608

biases: 32

total: 4640

FC1:

weights: 1568×120 = 188160

biases: 120

total: 188280

FC2:

weights: 120×10 = 1200

biases: 10

total: 1210

Hence, the total number of parameters = 160 + 4640 + 188280 + 1210 = 194290.

1.4

(a) Relative accuracy of the three models

- NetLin (Linear model)

Accuracy: ~70%. Performs only linear transformation, insufficient to classify complex characters.

- NetFull (2-layer fully connected network)

Accuracy: ~84%. Adding a hidden layer (tanh activation) improves non-linear pattern fitting significantly.

- NetConv (Convolutional network)

Accuracy: ~93%. Convolution layers effectively extract local features and spatial structures, achieving the highest accuracy as required.

(b) Number of independent parameters

- NetLin

weights: 784 × 10 = 7,840

biases: 10

Total: 7,850

- NetFull (hidden units=100)

Input to hidden: (784 × 100) + 100 = 78,500

Hidden to output: (100 × 10) + 10 = 1,010

Total: 79,510

- NetConv (kernel size =3, padding=1)

Conv1: (3×3×1×16) + 16 = 160

Conv2: (3×3×16×32) + 32 = 4,640

FC1: (1568×120) + 120 = 188,280

FC2: (120×10) + 10 = 1,210

Total: 194,290

(c) Confusion matrix analysis

1. NetLin

- Major confusions:
- Class 3 (tsu) frequently misclassified as 2 (su) or 4 (na).
- Class 1 (ki) and 2 (su) often confused.
- Reason: Lacks non-linear capability to distinguish similar visual structures.

2. NetFull

- Major confusions:
- Class 4 (na) still misclassified as 1 (ki) or 2 (su).
- Class 5 (ha) occasionally misclassified as 1 (ki).
- Reason: Hidden layer improves non-linear fitting but still lacks local feature extraction capability.

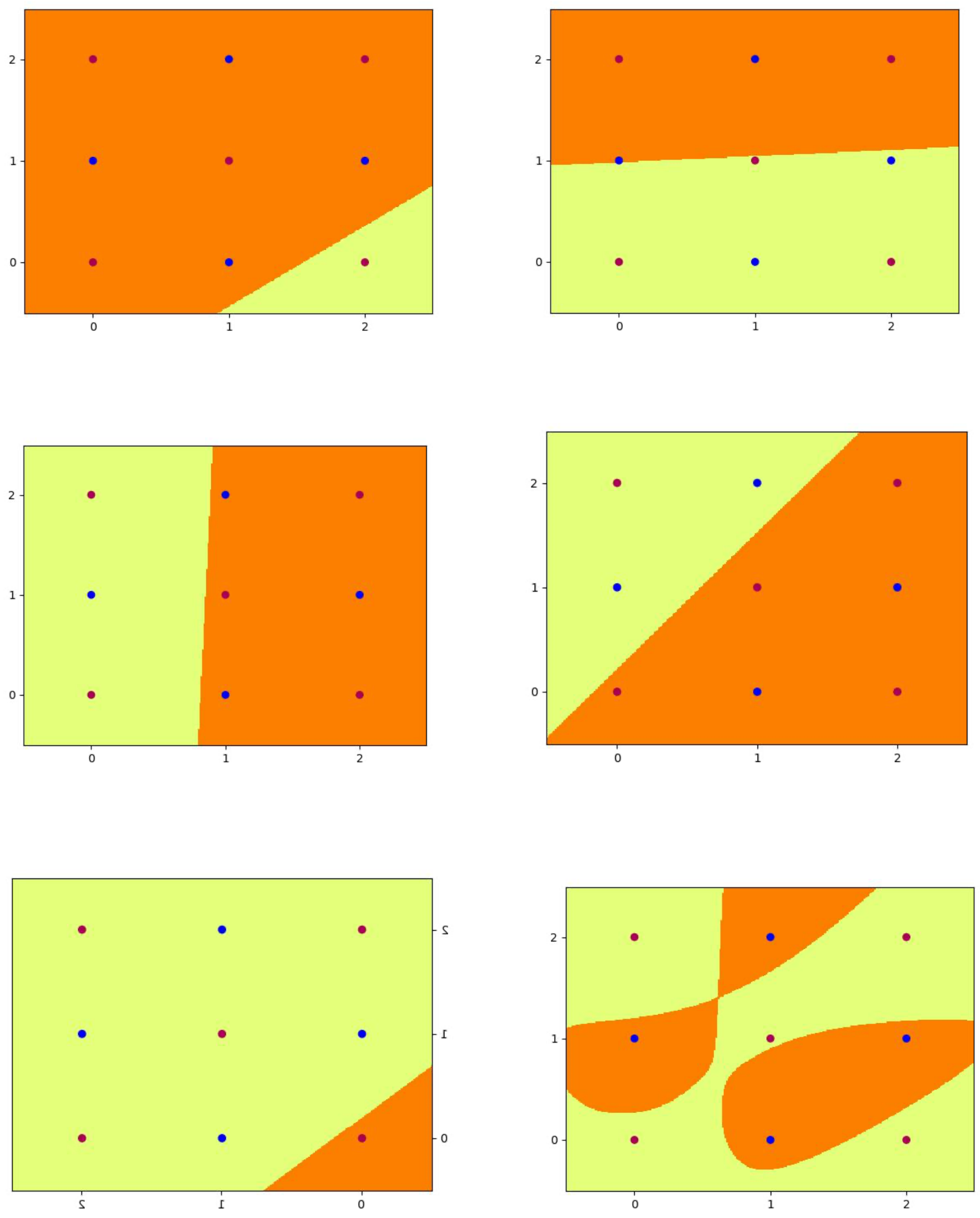
3. NetConv

- Major confusions:
- Class 4 (na) sometimes confused with 1 (ki) and 2 (su), but greatly reduced.
- Class 9 (wo) occasionally misclassified as 1 (ki) or 2 (su).
- Reason: CNN effectively captures edges, local patterns, and spatial structure, resulting in the best performance.

The three models show progressive improvement: NetLin < NetFull < NetConv, with increasing parameter counts and complexity. CNN significantly reduces misclassification among visually similar characters by extracting robust local features.

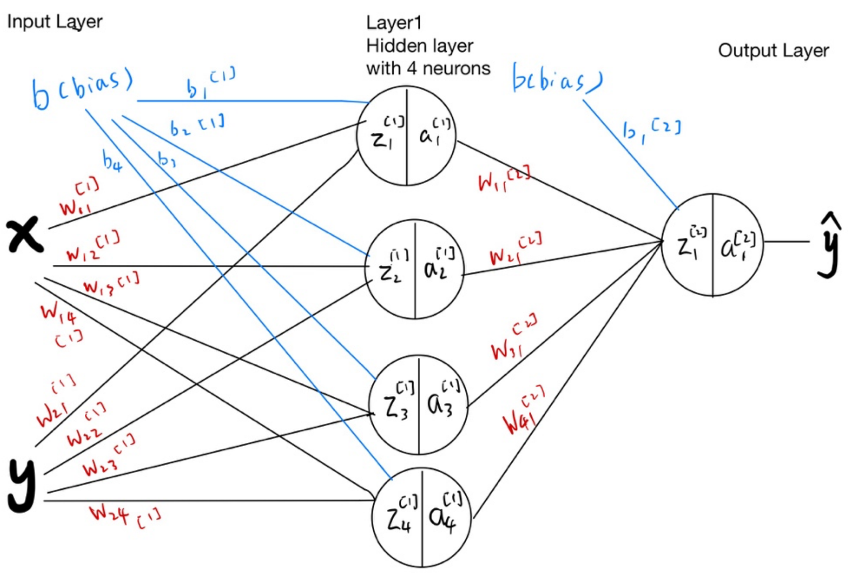
Part2

2.1



2.2

Network Diagram



Hidden Layer Decision Boundaries

Hidden Node	Equation	Weights	Bias
h1	x - y + 0.5 = 0	[1,-1]	-0.5
h2	x + y - 2.5 = 0	[1,1]	-2.5
h3	x - y - 0.5 = 0	[1,-1]	+0.5
h4	x + y - 1.5 = 0	[1,1]	-1.5

Output Layer Weights and Bias

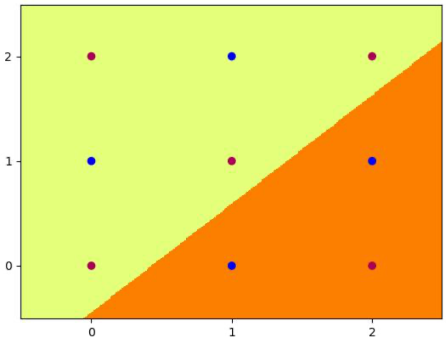
- weights: (1, 1, -1, -1)
- bias: 0

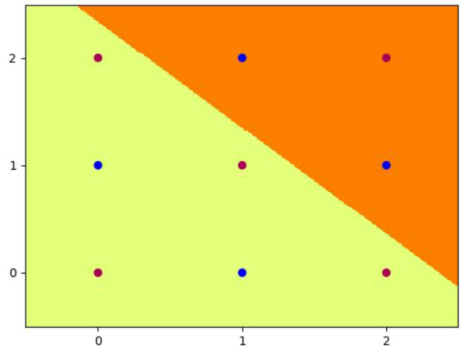
Activation Table

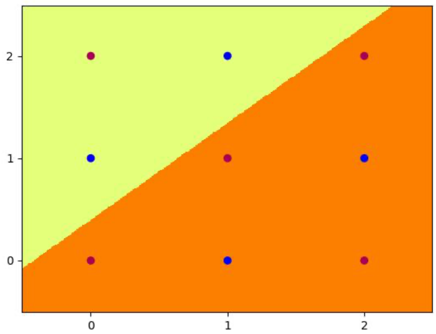
x	y	h1	h2	h3	h4	output
0	0	0	0	1	0	0
0	1	0	0	0	0	1
0	2	0	0	0	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	2	0	1	0	1	1
2	0	1	0	1	1	0
2	1	1	1	1	1	1
2	2	0	1	1	1	0

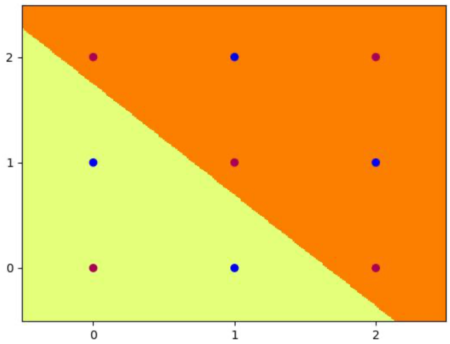
```
Initial Weights:
tensor([[ 1., -1.],
        [ 1.,  1.],
        [ 1., -1.],
        [ 1.,  1.]])
tensor([-0.5000, -2.5000,  0.5000, -1.5000])
tensor([[ 1.,  1., -1., -1.]])
tensor([0.])
Initial Accuracy:  100.0
```

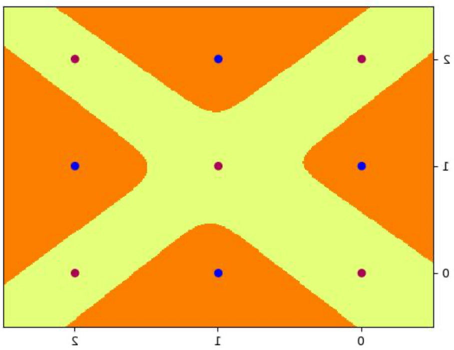
2.3







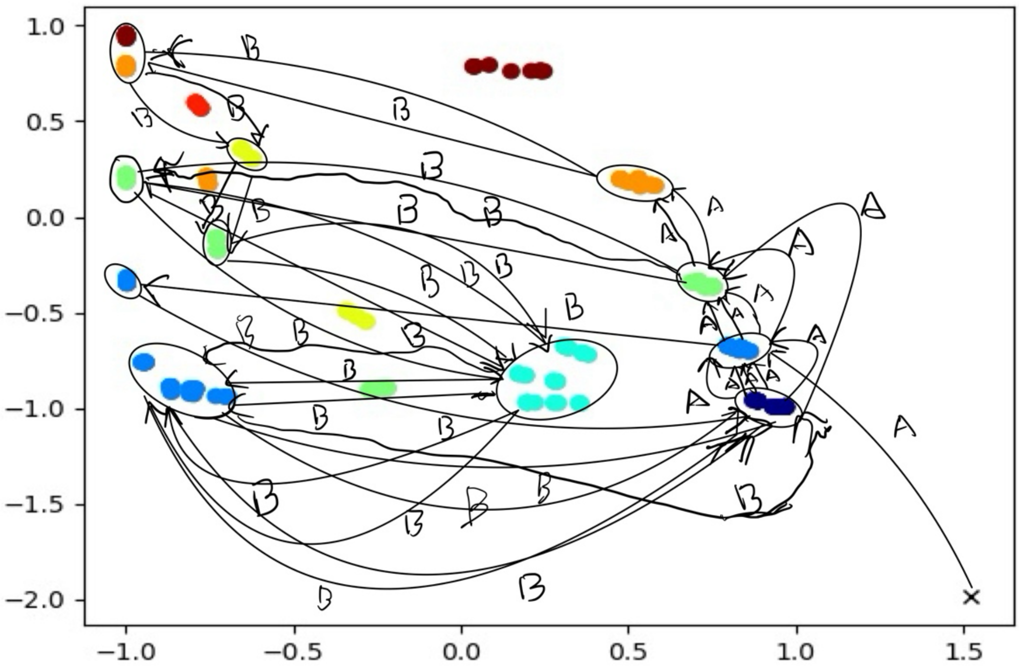




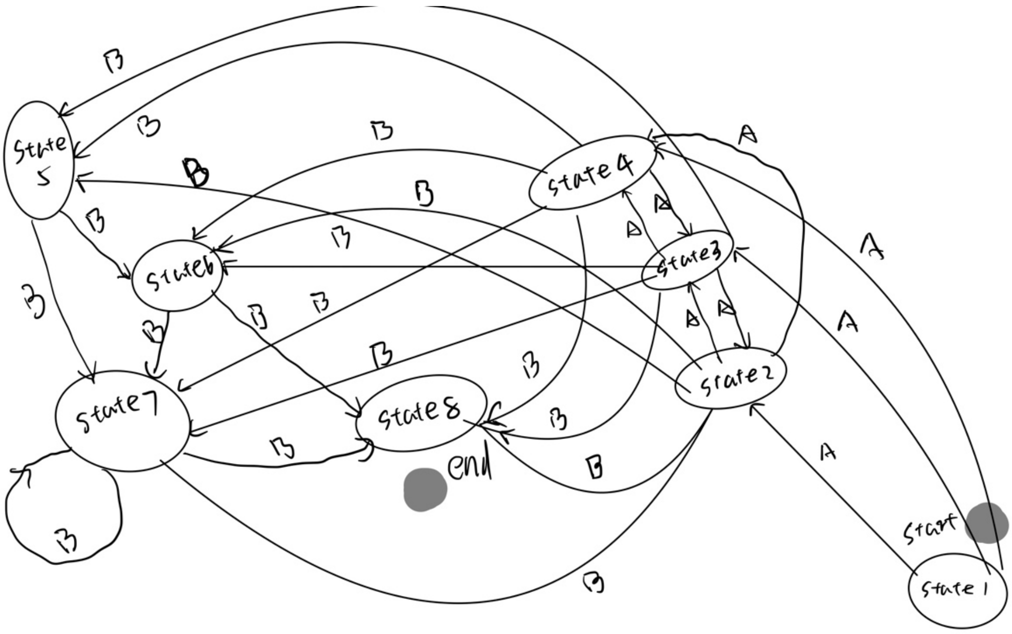

```
    def set_weights(self):
##### Enter Weights Here ###########
        in_hid_weight  = [[10,-10],[10,10],[10,-10],[10,10]]
        hid_bias       = [-5.0, -25.0, +5.0, -15.0]
        hid_out_weight = [[10, 10, -10, -10]]
        out_bias       = [0]
####################################
        self.in_hid_weight.data = torch.tensor(
```

Part3

3.1

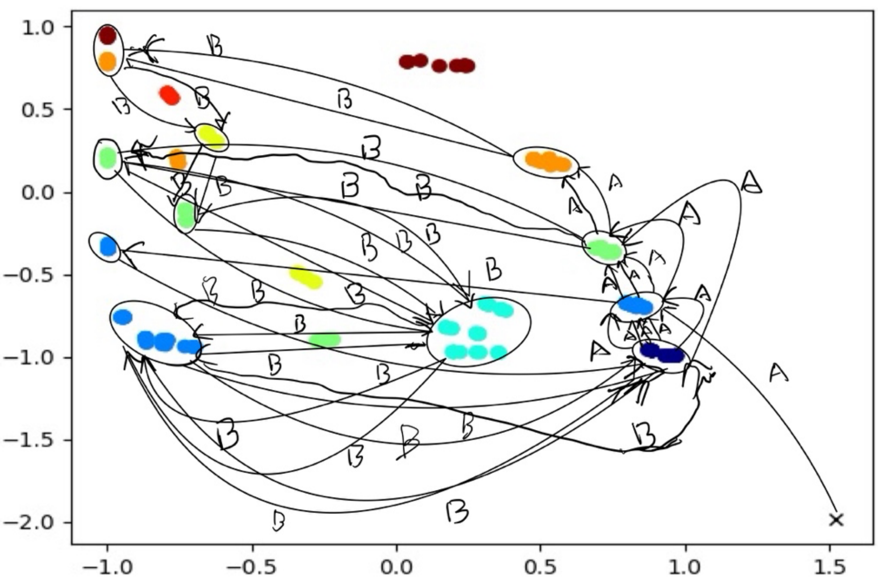


3.2



State 5
State 6
State 7
State 8
State 4
State 3
State 2
State 1
start
end
A
B

3.3



1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.0
−0.5
0.0
0.5
1.0
1.5
A
B

```
问题    输出    调试控制台    终端    端口

symbol= ABBABBABBAAAABBBBBBBBAAAABBBBBBBBA
hidden activations and output probabilities:
A [ 0.84 -0.68] [ 0.82  0.18]
B [-1.   -0.32] [ 0.  1.]
B [ 0.87 -0.96] [ 0.84  0.16]
A [ 0.86 -0.7 ] [ 0.85  0.15]
B [-1.   -0.32] [ 0.  1.]
B [ 0.88 -0.96] [ 0.85  0.15]
A [ 0.86 -0.7 ] [ 0.84  0.16]
B [-1.   -0.32] [ 0.  1.]
B [ 0.88 -0.96] [ 0.85  0.15]
A [ 0.86 -0.7 ] [ 0.84  0.16]
A [ 0.69 -0.34] [ 0.65  0.35]
A [ 0.58  0.16] [ 0.5  0.5]
A [ 0.02  0.79] [ 0.02  0.98]
B [-1.    0.93] [ 0.  1.]
B [-0.77  0.56] [ 0.  1.]
B [-0.75  0.16] [ 0.  1.]
B [-0.27 -0.56] [ 0.  1.]
B [-0.28 -0.91] [ 0.  1.]
B [ 0.38 -0.98] [ 0.14  0.86]
B [-0.88 -0.91] [ 0.  1.]
B [ 0.97 -1.  ] [ 0.91  0.09]
A [ 0.79 -0.67] [ 0.77  0.23]
A [ 0.76 -0.37] [ 0.74  0.26]
A [ 0.47  0.2 ] [ 0.32  0.68]
```

```
A [ 0.69 -0.34] [ 0.65  0.35]
A [ 0.58  0.16] [ 0.5  0.5]
A [ 0.02  0.79] [ 0.02  0.98]
B [-1.    0.93] [ 0.  1.]
B [-0.77  0.56] [ 0.  1.]
B [-0.75  0.16] [ 0.  1.]
B [-0.27 -0.56] [ 0.  1.]
B [-0.28 -0.91] [ 0.  1.]
B [ 0.38 -0.98] [ 0.14  0.86]
B [-0.88 -0.91] [ 0.  1.]
B [ 0.97 -1.  ] [ 0.91  0.09]
A [ 0.79 -0.67] [ 0.77  0.23]
A [ 0.76 -0.37] [ 0.74  0.26]
A [ 0.47  0.2 ] [ 0.32  0.68]
A [ 0.25  0.76] [ 0.11  0.89]
B [-1.    0.96] [ 0.  1.]
B [-0.79  0.6 ] [ 0.  1.]
B [-0.76  0.21] [ 0.  1.]
B [-0.34 -0.49] [ 0.  1.]
B [-0.22 -0.9 ] [ 0.  1.]
B [ 0.19 -0.97] [ 0.04  0.96]
B [-0.69 -0.94] [ 0.  1.]
B [ 0.93 -0.99] [ 0.89  0.11]
epoch: 100000
 loss: 0.0468
```

The anb2n language is:

• n consecutive A

• Next 2n B

The network needs to gradually predict each symbol.

1. Start state (initial × point):

• When the network starts the sequence, the hidden state is located in the initial state of the mark × in the graph.

2. Read Phase A:

• From the output, when A is input, the hidden layer activation value moves smoothly between the states in the right half of the figure.

• for example:

• A [0.84 -0.68] [0.82 0.18]

• A [0.87 -0.96] [0.84 0.16]

• These hidden activations of consecutive A are distributed in different "clusters" in the lower right.

• This means that the network records the number of A through different hidden states.

3. First B (non-deterministic prediction):

• Output:

• B [-1. -0.32] [0. 1.]

• The first B causes the hidden state to jump to a new cluster (usually in the lower left area of the figure), with the predicted probability close to [0,1], but due to the task definition, the first B itself is nondeterministic.

4. Subsequent B (deterministic prediction):

• Output:

Multiple B activation values remain in the B cluster area, for example:

• B [-1. -0.32] [0. 1.]

• B [-0.75 0.16] [0. 1.]

• B [-0.28 -0.91] [0. 1.]

• The hidden state switches at different locations in the B cluster, reflecting the network counting B, ensuring that all subsequent Bs can be correctly predicted.

5. A after the prediction sequence ends:

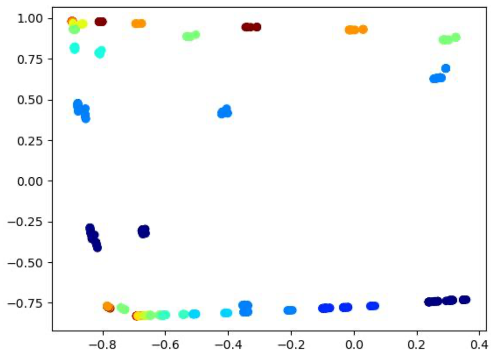
• After processing 2n Bs, the hidden state returns to the initial area, corresponding to the start of the new sequence:

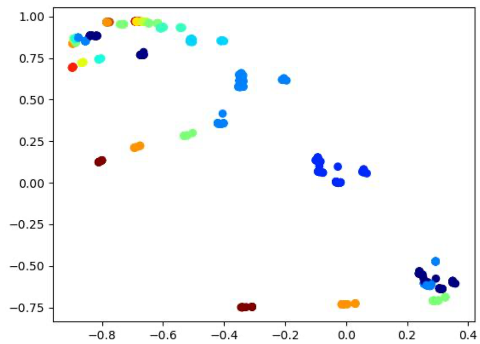
• A [0.79 -0.67] [0.77 0.23]

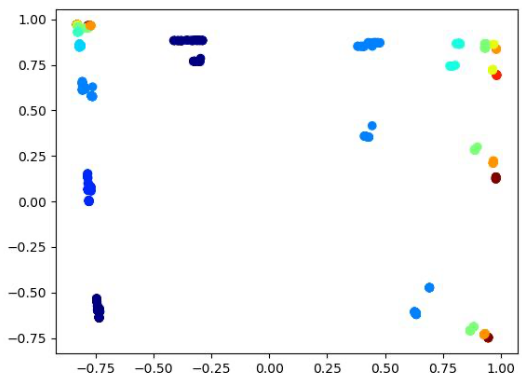
- A [0.76 -0.37] [0.74 0.26]

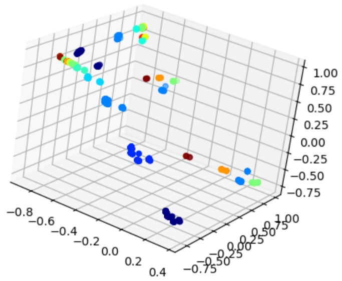
- The network is able to correctly predict the new A after the last B.

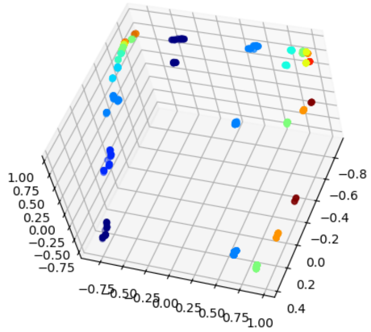
3.4











3.5

1. Table: LSTM Gate Operations at Each Stage

Stage	Input Symbol	Input Gate	Forget Gate	Output Gate	Function
A phase	A	OpenAccepts A input, updates cell state to accumulate n	Partially closedRetains previous cell state value	OpenOutputs hidden state for A prediction	Counts A's, updates cell state with n
B phase (transition)	First B	OpenReads n from cell state, initializes B counting	ClosedKeeps n stored	OpenOutputs B prediction	Switches from A to B phase, uses n for B counting
B phase (counting)	Subsequent B's	Closed or partially openNo update to n	ClosedRetains n	OpenOutputs B prediction	Generates 2n B's using n stored in cell state

C phase (transition)	First C	OpenReads n from cell state, initializes C counting	ClosedRetains n	OpenOutputs C prediction	Switches from B to C phase, uses n for C counting
C phase (counting)	Subsequent C's	Closed or partially open	Closed	Open	Generates 3n C's using n stored in cell state
End	Next A	Reset cell state as needed	OpenForgets previous n	Open	Resets state, prepares for next sequence

2. How LSTM uses gate mechanism to track nested structures

• Input Gate

Decide whether the current input symbol affects cell state

• In Phase A: Turn on, accumulate n

• In B/C phase: partially off, only on when phase switches to read n

• Forget Gate

Decide how much previous cell state information to retain

• Throughout the sequence: Mainly keeping it closed

• Only at the end of the sequence: Turn on, reset cell state, prepare for the next sequence

• Output Gate

Control cell state information flow to hidden state

• Stay on at all stages

• Make hidden state produce corresponding A, B, C output prediction

3. Hide state and cell state evolution (combined with 3D image analysis)

Hidden State:

• In the figure, the three stages A, B, and C correspond to the hidden state clusters of different regions

• It reflects the different gating combinations, causing hidden states to "jump" in the space, forming clear separation

Cell State:

• The n values accumulated in stage A are stored inside the LSTM, and the information is read through the input gate in stages B and C to guide the output quantity

• Since the forgetting door is closed, cell state can be saved for a long time n, completing the tracking of nested structures (A→B→C)

4. Long-range dependency and nested structure capture capability

The key to success of LSTM is:

• Capture long-range dependencies

Conventional RNNs tend to forget early inputs in sequences; LSTM maintains n values through cell state and gate mechanisms, and can be used from stage A of the sequence header to stage C at the end

• Handle nested structures

anb2nc3n task is essentially "nested counting"

• Stage A definition n

• Phase B dependency n generates 2n

• Stage C is relied on n to generate 3n

LSTM uses cell state, a design that can store information for a long time, to achieve complete tracking of nested structures.