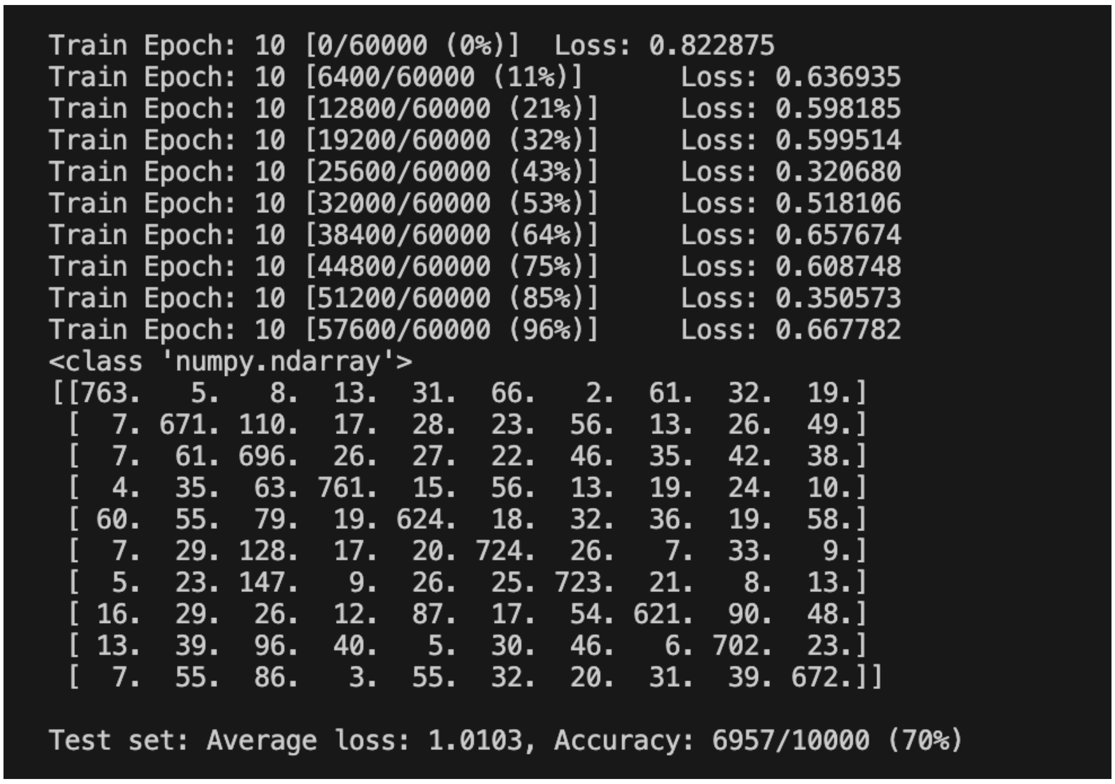
Part 1

Question 1

In this experiment, we implemented the NetLin model, which is a single-layer linear classifier. The model consists of a fully connected linear layer mapping the 784 input pixels (28 × 28 images) directly to the 10 output classes, followed by a log softmax activation. The network was trained on the KMNIST dataset for handwritten Japanese Hiragana character recognition.



```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.822875
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.636935
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.598185
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.599514
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.320680
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.518106
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.657674
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.608748
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.350573
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.667782
<class 'numpy.ndarray'>
[[763.   5.   8.  13.  31.  66.   2.  61.  32.  19.]
 [  7. 671. 110.  17.  28.  23.  56.  13.  26.  49.]
 [  7.  61. 696.  26.  27.  22.  46.  35.  42.  38.]
 [  4.  35.  63. 761.  15.  56.  13.  19.  24.  10.]
 [ 60.  55.  79.  19. 624.  18.  32.  36.  19.  58.]
 [  7.  29. 128.  17.  20. 724.  26.   7.  33.   9.]
 [  5.  23. 147.   9.  26.  25. 723.  21.   8.  13.]
 [ 16.  29.  26.  12.  87.  17.  54. 621.  90.  48.]
 [ 13.  39.  96.  40.   5.  30.  46.   6. 702.  23.]
 [  7.  55.  86.   3.  55.  32.  20.  31.  39. 672.]]

Test set: Average loss: 1.0103, Accuracy: 6957/10000 (70%)
```

Question 2

Calculation of the number of independent parameters:

the hidden layer size is 110.

Input to hidden layer:

(Number of input features + 1 for bias) × Hidden layer size

Hidden to output layer:

(Hidden layer size + 1 for bias) × Number of output classes

= (110 + 1) × 10 = 1,110

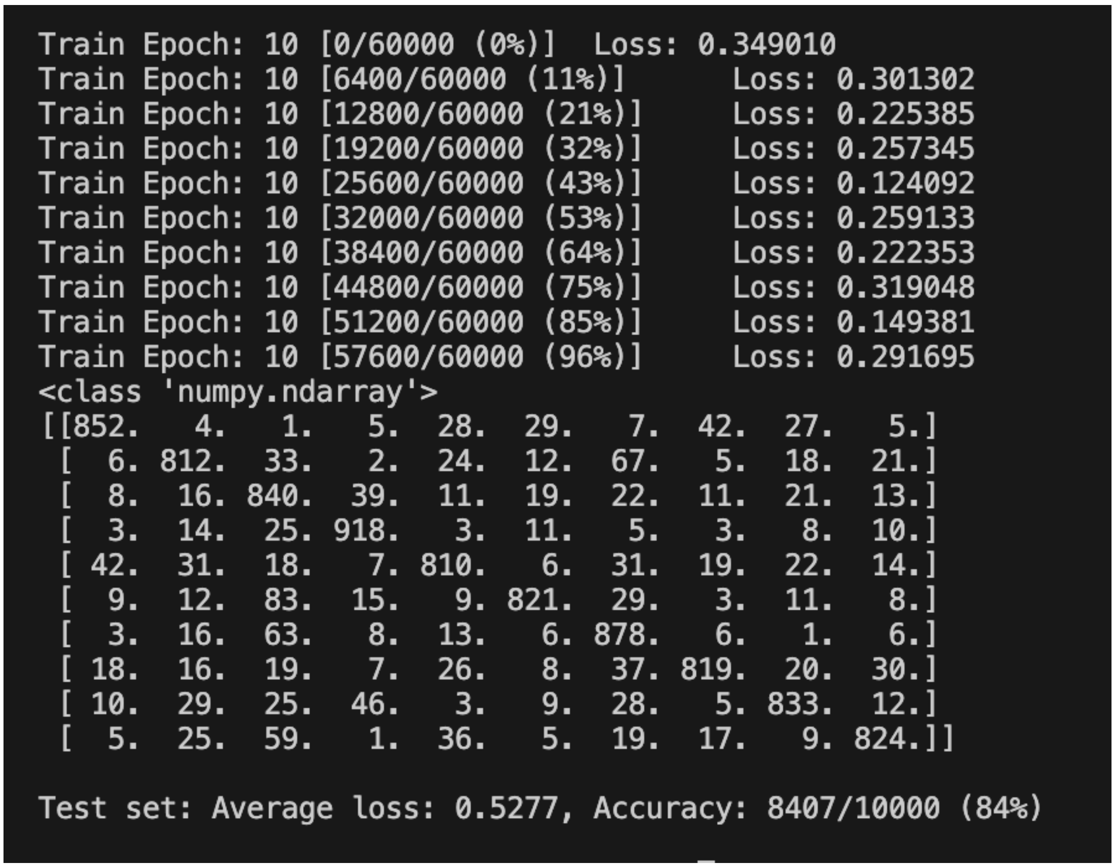
Total number of parameters:

86,350 + 1,110 = 87,460

Therefore, the NetFull model with a hidden size of 110 has 87,460 independent parameters.

The fully connected network performs significantly better than the linear model, as it is able to learn non-linear representations of the input data. However, it still struggles to distinguish some similar-looking classes, as seen in the confusion matrix.

Number of Hidden nodes	Accuracy
50	80%
80	83%
90	83.8%
100	83.9%
110	84.07%
150	84.15%
200	84.97%
300	85.15%



```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.349010
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.301302
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.225385
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.257345
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.124092
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.259133
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.222353
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.319048
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.149381
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.291695
<class 'numpy.ndarray'>
[[852.   4.   1.   5.  28.  29.   7.  42.  27.   5.]
 [  6. 812.  33.   2.  24.  12.  67.   5.  18.  21.]
 [  8.  16. 840.  39.  11.  19.  22.  11.  21.  13.]
 [  3.  14.  25. 918.   3.  11.   5.   3.   8.  10.]
 [ 42.  31.  18.   7. 810.   6.  31.  19.  22.  14.]
 [  9.  12.  83.  15.   9. 821.  29.   3.  11.   8.]
 [  3.  16.  63.   8.  13.   6. 878.   6.   1.   6.]
 [ 18.  16.  19.   7.  26.   8.  37. 819.  20.  30.]
 [ 10.  29.  25.  46.   3.   9.  28.   5. 833.  12.]
 [  5.  25.  59.   1.  36.   5.  19.  17.   9. 824.]]

Test set: Average loss: 0.5277, Accuracy: 8407/10000 (84%)
```

Question 3

In this part, I implemented the NetConv model, which is a convolutional neural network consisting of two convolutional layers followed by one fully connected layer. Each convolutional and fully connected layer uses the ReLU activation function, and the output layer applies log softmax.

Network Structure:

Conv Layer 1: 1 input channel, 32 output channels, 5×5 kernel, ReLU activation

Max Pooling: 2×2

Conv Layer 2: 32 input channels, 64 output channels, 5×5 kernel, ReLU activation

Max Pooling: 2×2

Fully Connected Layer 1: input 1024 (64×4×4), output 128, ReLU activation

Fully Connected Layer 2 (output): input 128, output 10, log softmax

Calculation of the number of independent parameters:

Conv Layer 1:

(Number of input channels × kernel height × kernel width + 1 for bias) × number of output channels

$= (1 \times 5 \times 5 + 1) \times 32 = (25 + 1) \times 32 = 832$

Conv Layer 2:

$(32 \times 5 \times 5 + 1) \times 64 = (800 + 1) \times 64 = 51{,}264$

Fully Connected Layer 1:

$(64 \times 4 \times 4 + 1) \times 128 = (1024 + 1) \times 128 = 131{,}200$

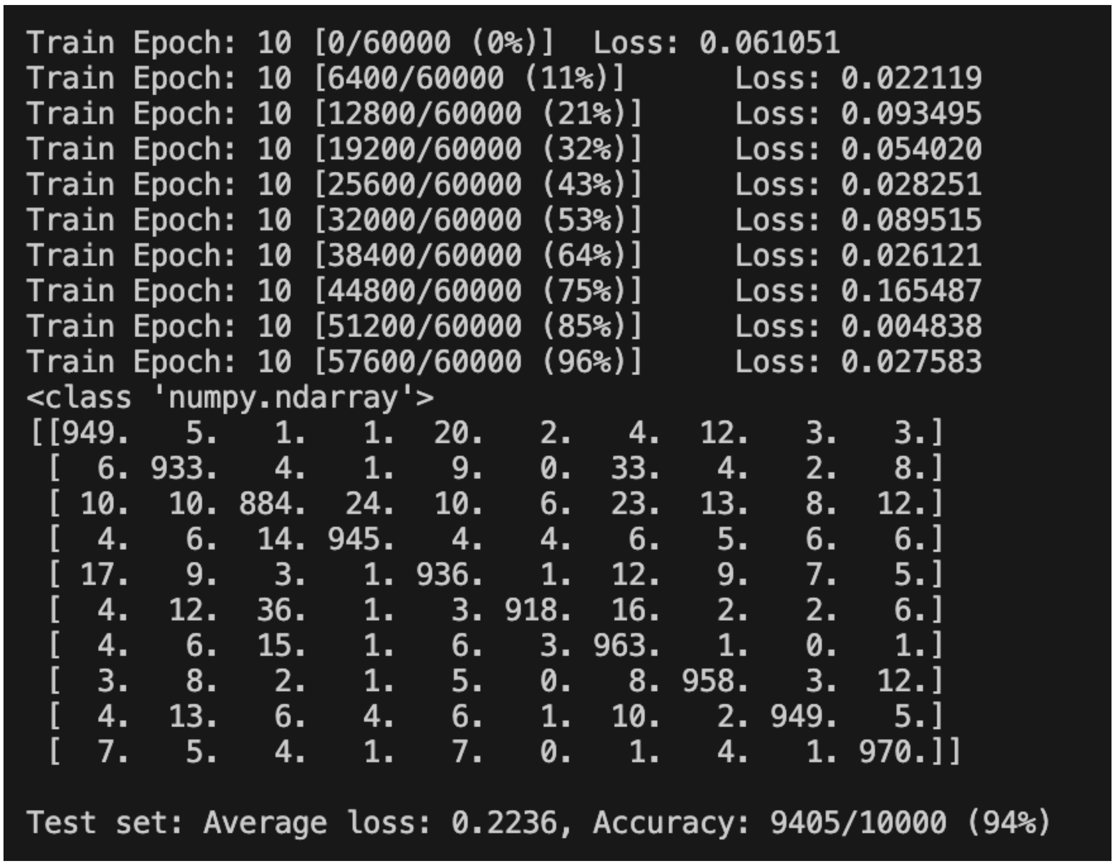
Fully Connected Layer 2:

$(128 + 1) \times 10 = 129 \times 10 = 1{,}290$

Total number of parameters:

$832 + 51{,}264 + 131{,}200 + 1{,}290 = 184{,}586$

Therefore, the NetConv model has 184,586 independent parameters



```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.061051
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.022119
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.093495
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.054020
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.028251
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.089515
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.026121
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.165487
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.004838
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.027583
<class 'numpy.ndarray'>
[[949.   5.   1.   1.  20.   2.   4.  12.   3.   3.]
 [  6. 933.   4.   1.   9.   0.  33.   4.   2.   8.]
 [ 10.  10. 884.  24.  10.   6.  23.  13.   8.  12.]
 [  4.   6.  14. 945.   4.   4.   6.   5.   6.   6.]
 [ 17.   9.   3.   1. 936.   1.  12.   9.   7.   5.]
 [  4.  12.  36.   1.   3. 918.  16.   2.   2.   6.]
 [  4.   6.  15.   1.   6.   3. 963.   1.   0.   1.]
 [  3.   8.   2.   1.   5.   0.   8. 958.   3.  12.]
 [  4.  13.   6.   4.   6.   1.  10.   2. 949.   5.]
 [  7.   5.   4.   1.   7.   0.   1.   4.   1. 970.]]

Test set: Average loss: 0.2236, Accuracy: 9405/10000 (94%)
```

Question 4

a)

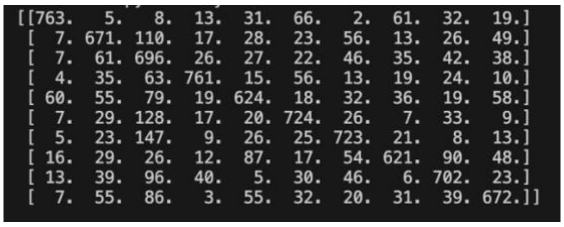
In this Kuzushiji-MNIST task, the three architectures exhibit a clear accuracy gradient. The linear model NetLin reaches only about 70 % accuracy because a single linear layer cannot capture non-linear or spatial features. Introducing a hidden tanh layer in NetFull greatly increases representational power, lifting accuracy to roughly 84 %; however, once the hidden size exceeds about 250 units, additional parameters yield little further gain. The best performer is the convolutional model NetConv, which leverages local receptive fields, weight sharing, and pooling to extract curved-stroke and hook patterns characteristic of cursive kana, pushing test accuracy to around 94 % within just ten epochs. In short, the more a network exploits spatial locality and non-linear representation, the higher its recognition performance—hence NetConv offers the best balance of accuracy and parameter efficiency.

b)

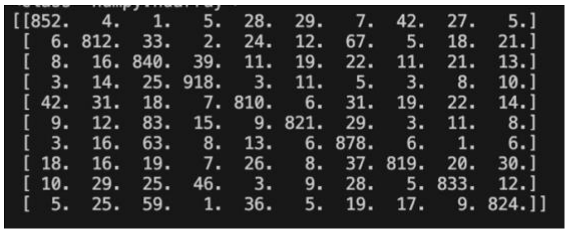
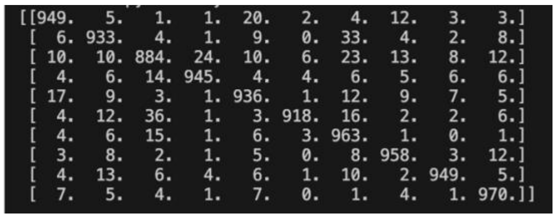
Model	Independent Parameters
Linear Network	7850
Full Network	87460
Convolution Network	184586

C)

0="o", 1="ki", 2="su", 3="tsu", 4="na", 5="ha", 6="ma", 7="ya", 8="re", 9="wo"

Model	Confusion Matrix	Most likely mistaken	Reason
Linear Network		1 and 2 5 and 2 6 and 2 8 and 2 9 and 2	1 and 2 were the most mistaken, perhaps due to the similarities in the left-hand side of the image. Both are quite thin characters with lots of "curls" and so appear similar to a simple linear network

```
[[763.   5.   8.  13.  31.  66.   2.  61.  32.  19.]
 [  7. 671. 110.  17.  28.  23.  56.  13.  26.  49.]
 [  7.  61. 696.  26.  27.  22.  46.  35.  42.  38.]
 [  4.  35.  63. 761.  15.  56.  13.  19.  24.  10.]
 [ 60.  55.  79.  19. 624.  18.  32.  36.  19.  58.]
 [  7.  29. 128.  17.  20. 724.  26.   7.  33.   9.]
 [  5.  23. 147.   9.  26.  25. 723.  21.   8.  13.]
 [ 16.  29.  26.  12.  87.  17.  54. 621.  90.  48.]
 [ 13.  39.  96.  40.   5.  30.  46.   6. 702.  23.]
 [  7.  55.  86.   3.  55.  32.  20.  31.  39. 672.]]
```

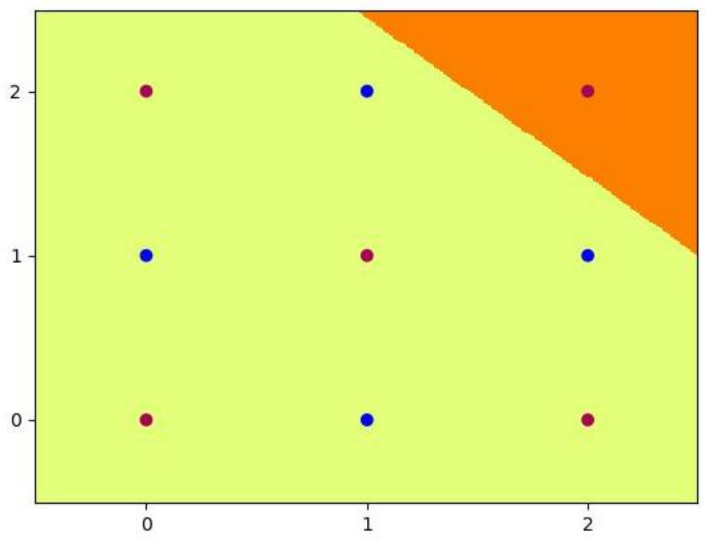
Full Network		2 and 5 2 and 6 2 and 9 3 and 8 6 and 1 7 and 0	2 and 5 were the most mistaken. The right side of the characters appear very similar, with a long, thin vertical line.
Convolution Network		2 and 5 6 and 1	There were not many mistakes here. Again, 2 and 5 were the most mistaken. Perhaps for the same reasons as above

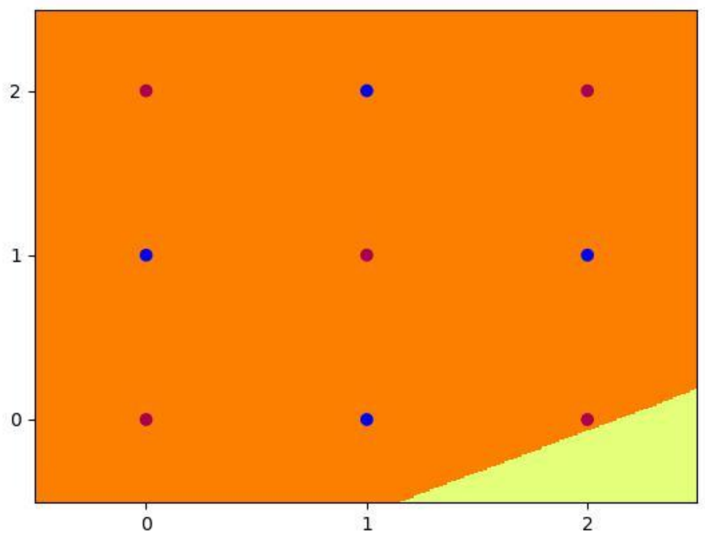
```
[[852.   4.   1.   5.  28.  29.   7.  42.  27.   5.]
 [  6. 812.  33.   2.  24.  12.  67.   5.  18.  21.]
 [  8.  16. 840.  39.  11.  19.  22.  11.  21.  13.]
 [  3.  14.  25. 918.   3.  11.   5.   3.   8.  10.]
 [ 42.  31.  18.   7. 810.   6.  31.  19.  22.  14.]
 [  9.  12.  83.  15.   9. 821.  29.   3.  11.   8.]
 [  3.  16.  63.   8.  13.   6. 878.   6.   1.   6.]
 [ 18.  16.  19.   7.  26.   8.  37. 819.  20.  30.]
 [ 10.  29.  25.  46.   3.   9.  28.   5. 833.  12.]
 [  5.  25.  59.   1.  36.   5.  19.  17.   9. 824.]]
```

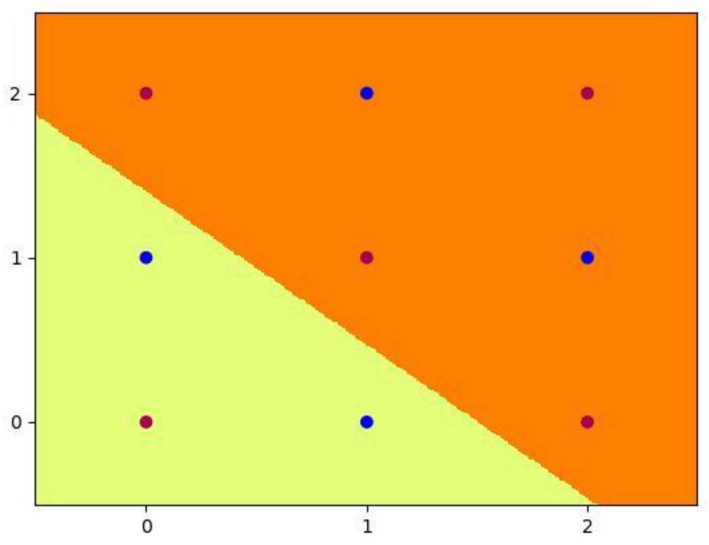
```
[[949.   5.   1.   1.  20.   2.   4.  12.   3.   3.]
 [  6. 933.   4.   1.   9.   0.  33.   4.   2.   8.]
 [ 10.  10. 884.  24.  10.   6.  23.  13.   8.  12.]
 [  4.   6.  14. 945.   4.   4.   6.   5.   6.   6.]
 [ 17.   9.   3.   1. 936.   1.  12.   9.   7.   5.]
 [  4.  12.  36.   1.   3. 918.  16.   2.   2.   6.]
 [  4.   6.  15.   1.   6.   3. 963.   1.   0.   1.]
 [  3.   8.   2.   1.   5.   0.   8. 958.   3.  12.]
 [  4.  13.   6.   4.   6.   1.  10.   2. 949.   5.]
 [  7.   5.   4.   1.   7.   0.   1.   4.   1. 970.]]
```

Part 2

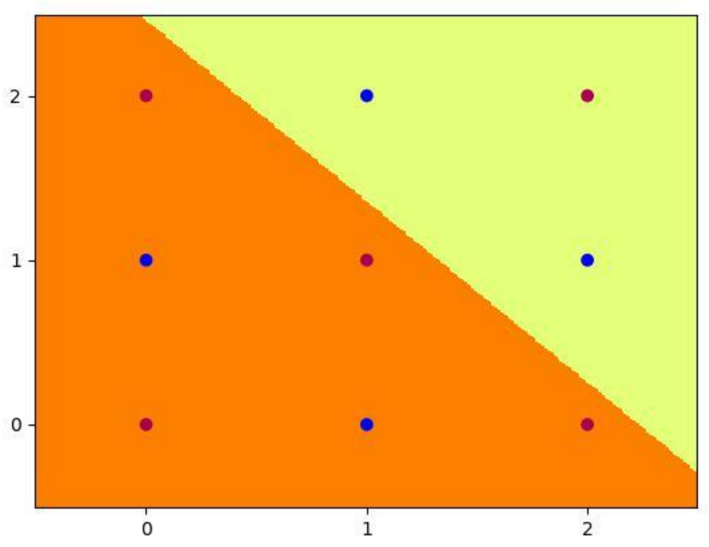
Question 1



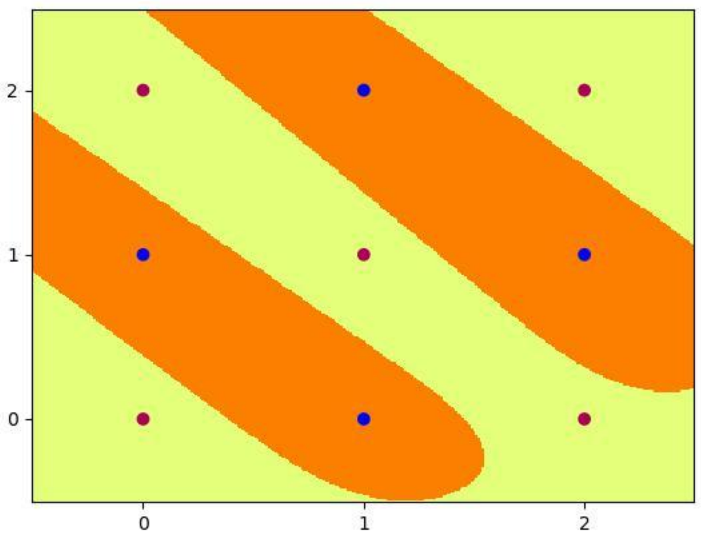




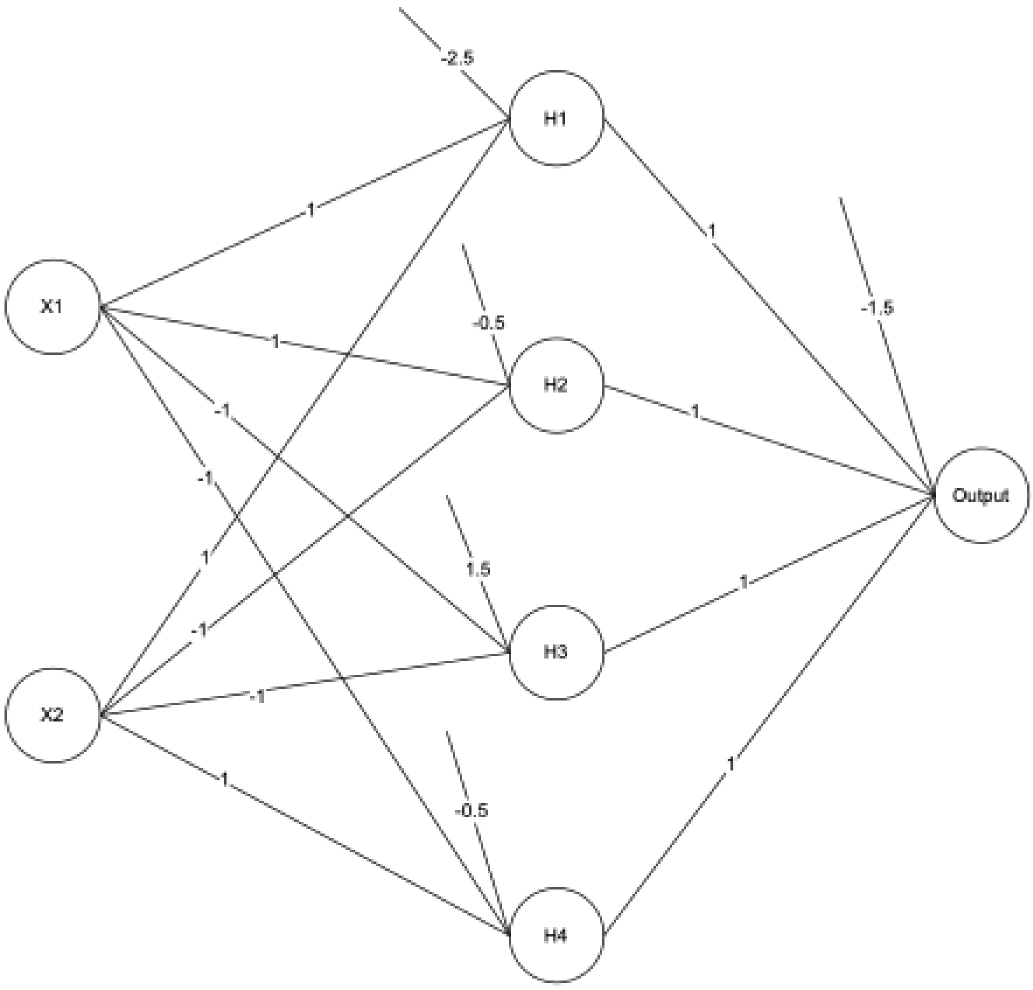
0
1
2
0
1
2



0
1
2
0
1
2



Question 2



Hidden Node H1: x1 + x2 - 2.5 = 0

Hidden Node H2: x1 - x2 - 0.5 = 0

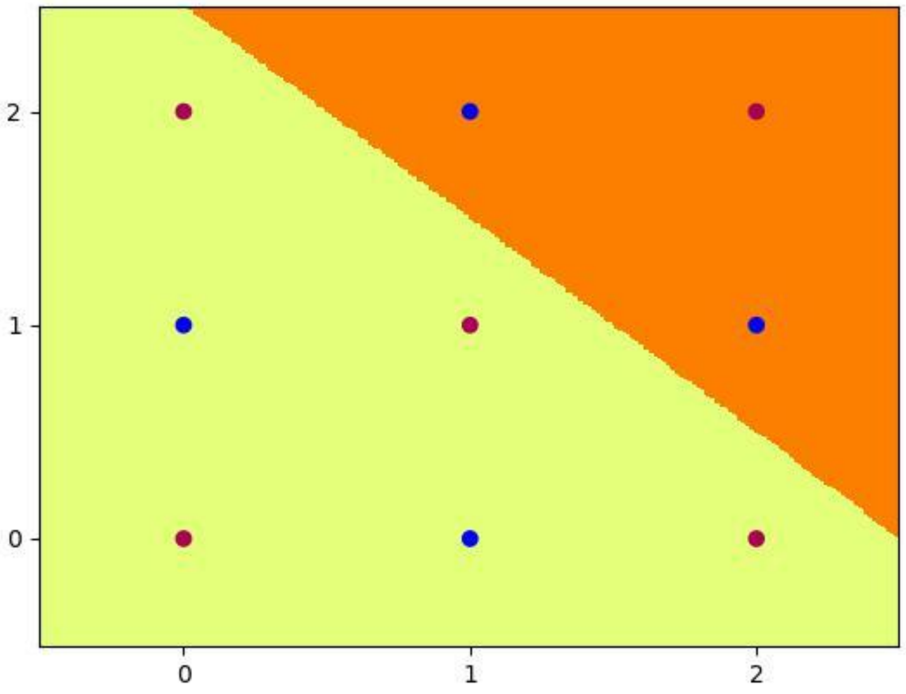
Hidden Node H3: -x1 – x2 + 1.5 = 0

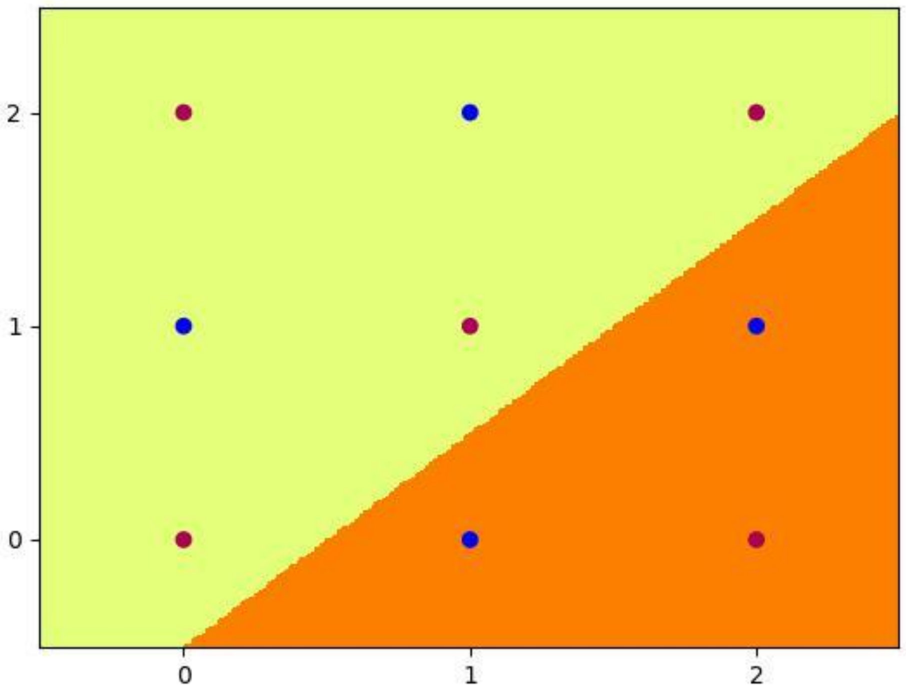
Hidden Node H4: :-x1 + x2 - 0.5 = 0

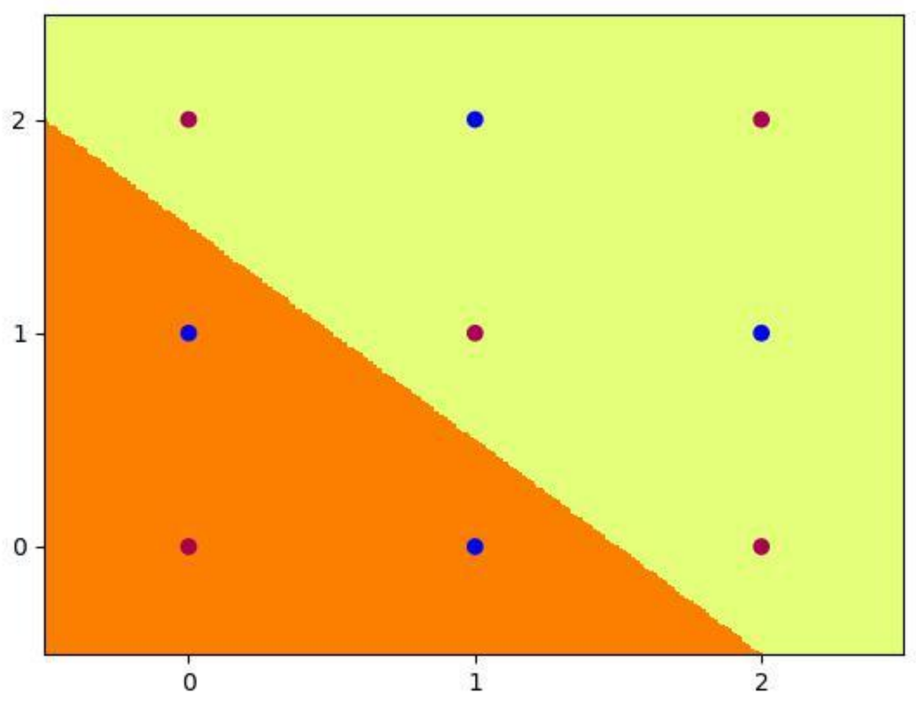
X1 X2	Target	H1 Act.	H2 Act.	H3 Act.	H4 Act.	Final Output
(0, 0)	0	0	0	1	0	0
(0, 1)	1	0	0	1	1	1
(0, 2)	0	0	0	0	1	0
(1, 0)	1	0	1	1	0	1
(1, 1)	0	0	0	0	0	0
(1, 2)	1	1	0	0	1	1
(2, 0)	0	0	1	0	0	0
(2, 1)	1	1	1	0	0	1
(2, 2)	0	1	0	0	0	0

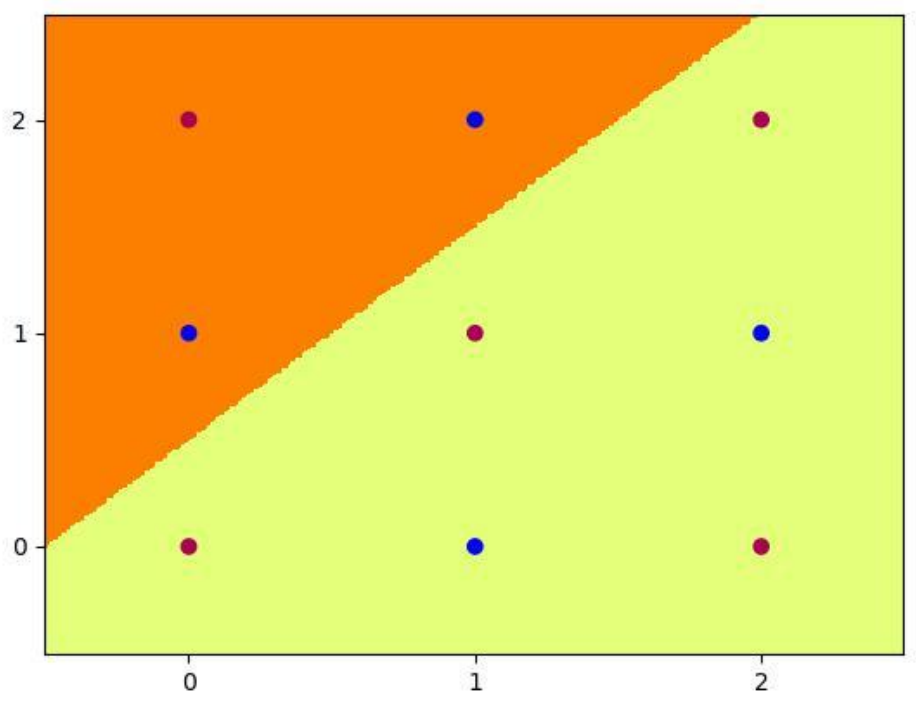
Question3

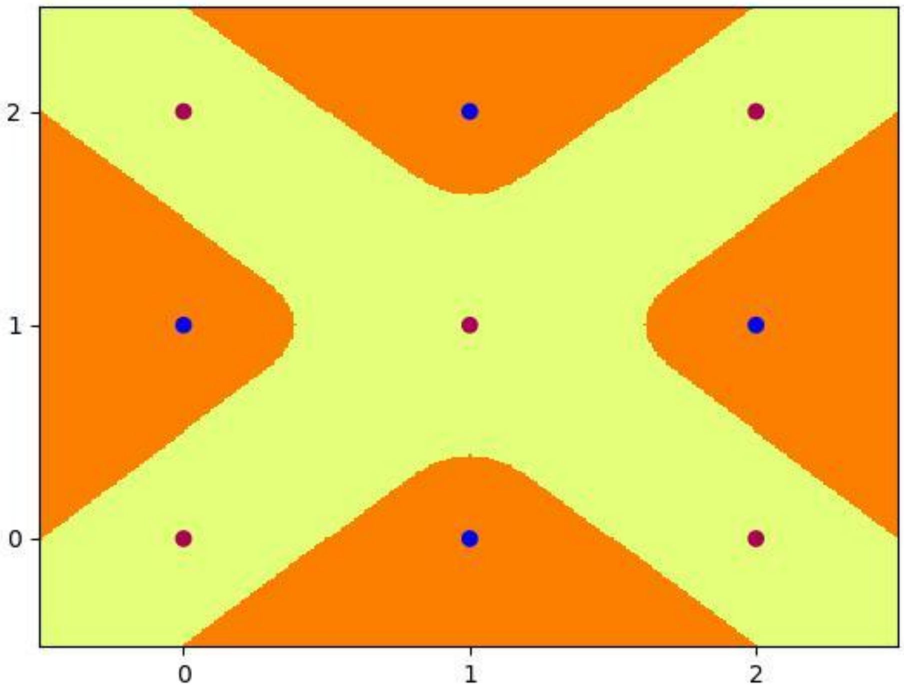
These images demonstrate that by multiplying the weights and biases by a large constant, the sigmoid activation function effectively approximates a step function. As a result, the network correctly classifies all data points, achieving the same outcome as the hand-designed step-function network.


0
1
2
0
1
2


0
1
2
0
1
2


0
1
2
0
1
2


0
1
2
0
1
2

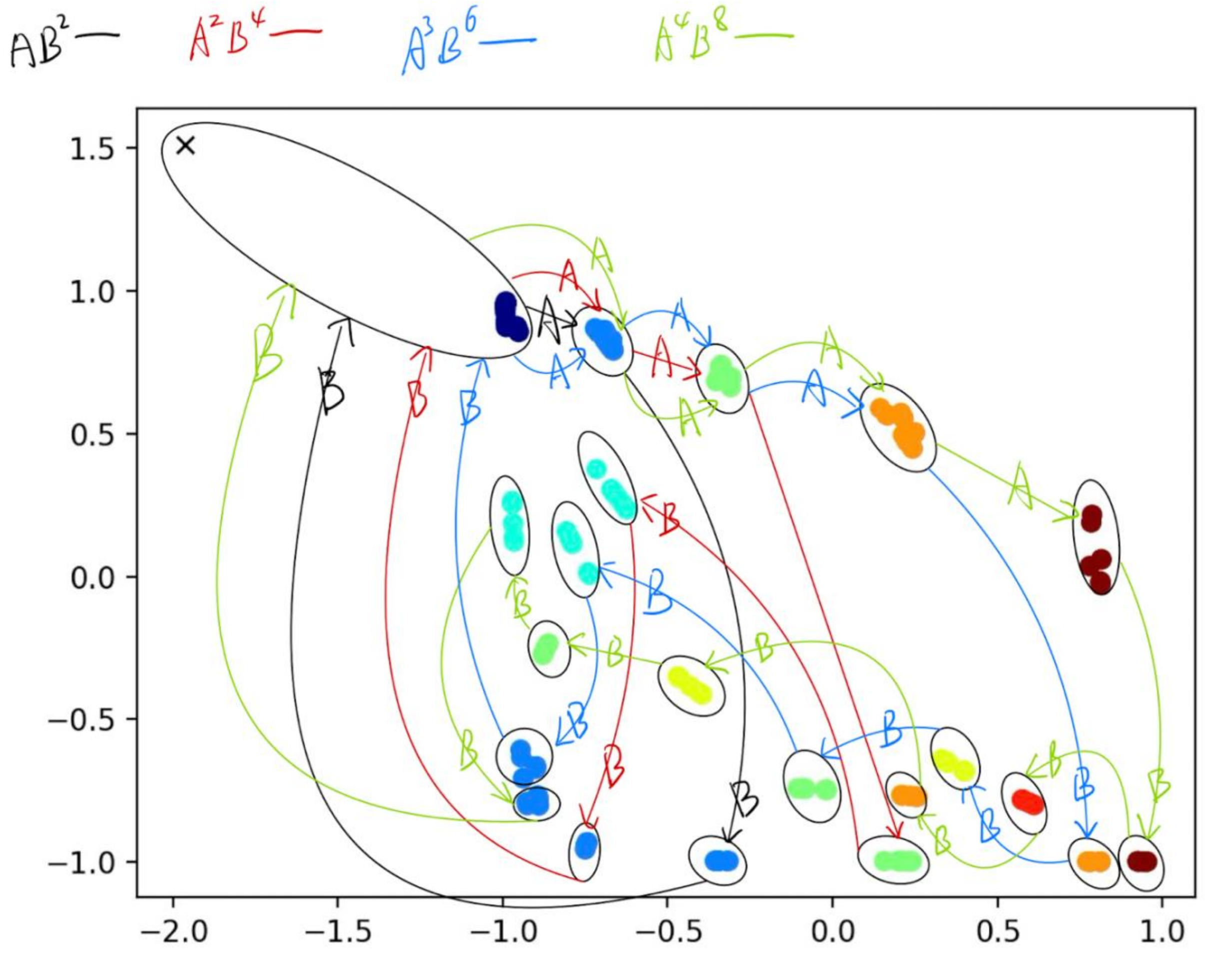

0
1
2
0
1
2

Part 3

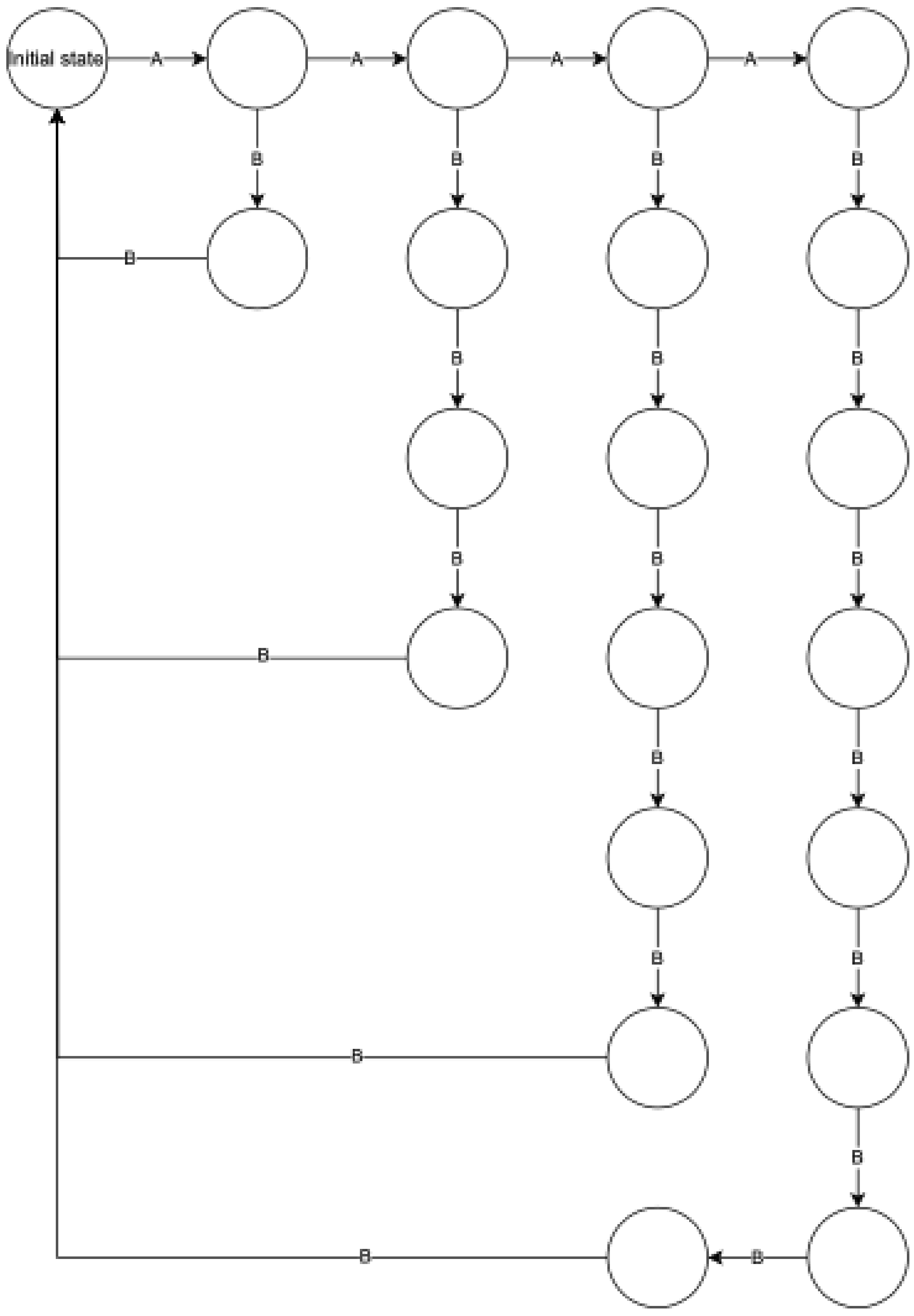
Question 1

The black “×” and the tight group of dark-blue dots are merged into a single initial state cluster S₀.

During every A^{n}B^{2n} sequence, the network’s hidden vector moves along the A-chain while reading the A’s, then along the B-chain while reading the 2n B’s; after the final B, the hidden vector collapses back into the dark-blue cluster, i.e. S₀, ready to start the next sequence.



Question 2



Question 3

The trained SRN turns its two hidden units into a very small finite-state counter:

1. Initialisation – At the start of every sequence the hidden vector sits in the cluster S_{0} (the “×” + dark-blue dots). Output weights make this point score A ≫ B, so the network confidently predicts the first A.

2. Reading the A-block – Each time an A is seen the vector steps outwards along the “A-ray” . The distance from S_{0} is therefore proportional to the running count n. Because the next symbol could still be A or B, the output layer can only give a probabilistic guess here—exactly the behaviour required by the spec. ￼

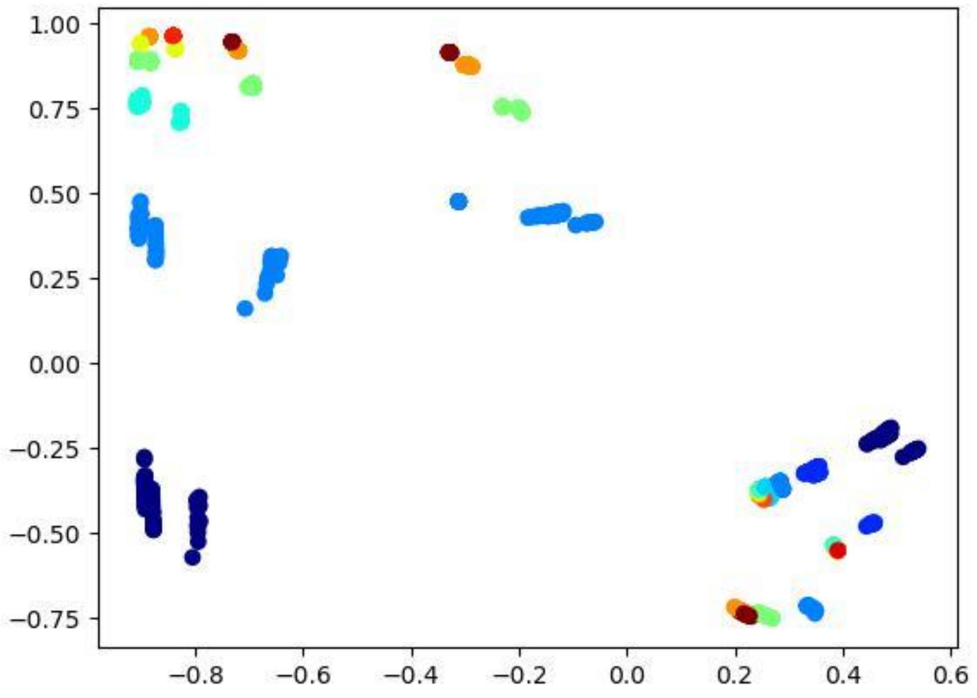
3. First B: direction flip – Encountering the first B rotates the update direction so the vector now moves back towards the origin along a “B-ray”. From this point on the hidden state encodes “how many of the required 2n B’s are still outstanding”.

4. Deterministic B-chain – Because the vector stays on the B-ray until it hits S_{0}, the output weights assign an overwhelming probability to B at every intermediate position. Thus all B’s after the first are predicted almost deterministically, as demanded. ￼

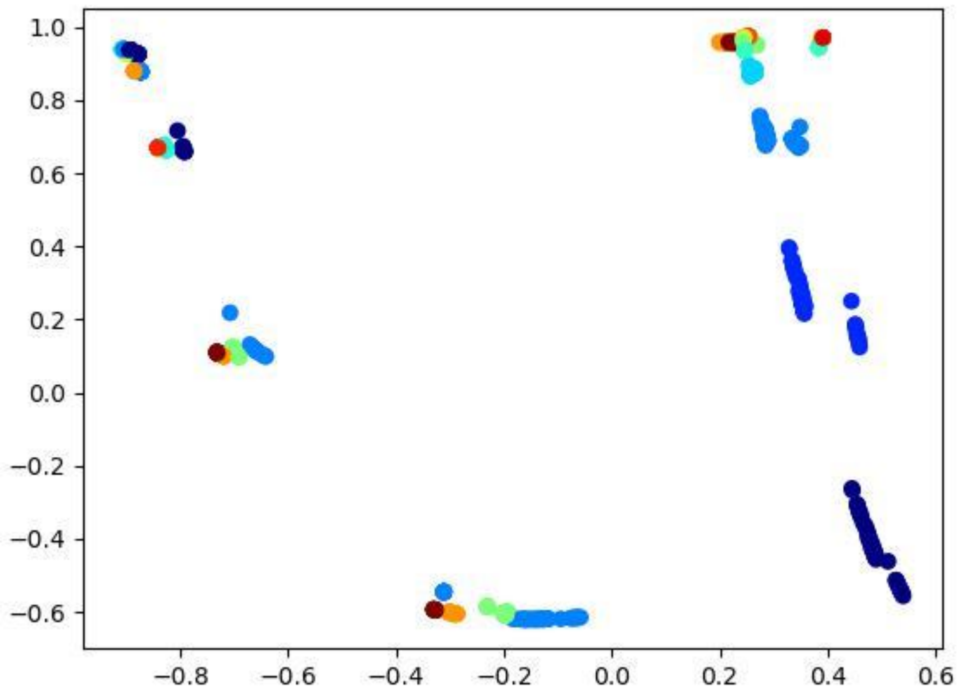
5. Last B → reset – After the 2n^th B the vector has walked exactly back into S_{0}. From that point the output layer again favours A, so the very first symbol of the next subsequence is predicted correctly. The cycle then repeats for the next random n.

In short, the SRN has learned a one-dimensional internal clock: move away from the origin while counting A’s, then march back with twice the step size for B’s. The geometry of these two line segments, together with fixed output weights, is sufficient to make every non-probabilistic prediction (all B’s except the first, and the post-sequence A) exactly right while still respecting the inherent uncertainty of the first B and the “middle” A’s.

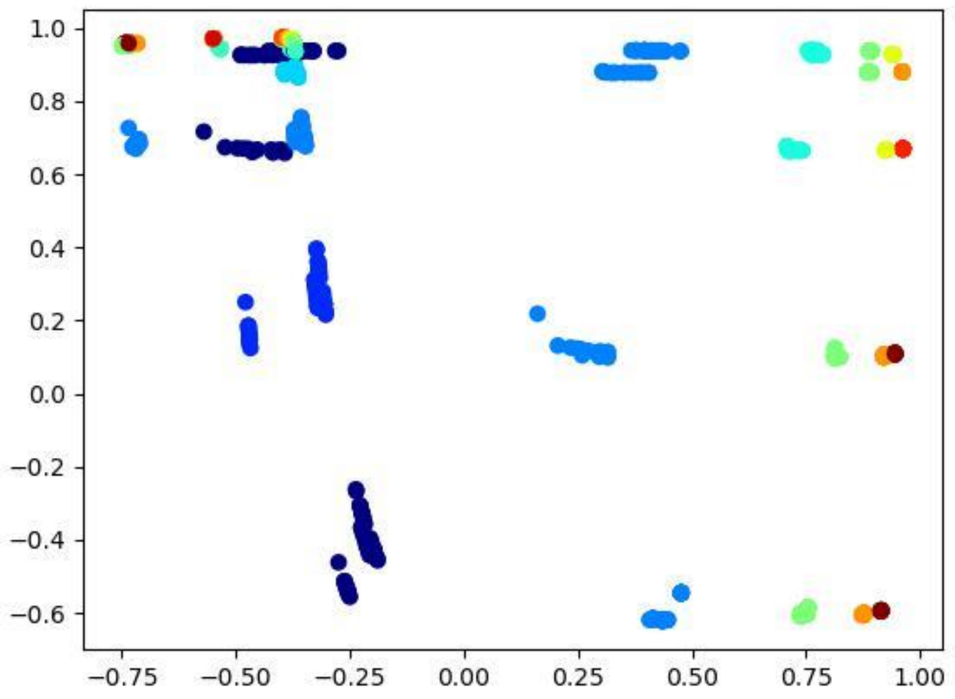
Question 4



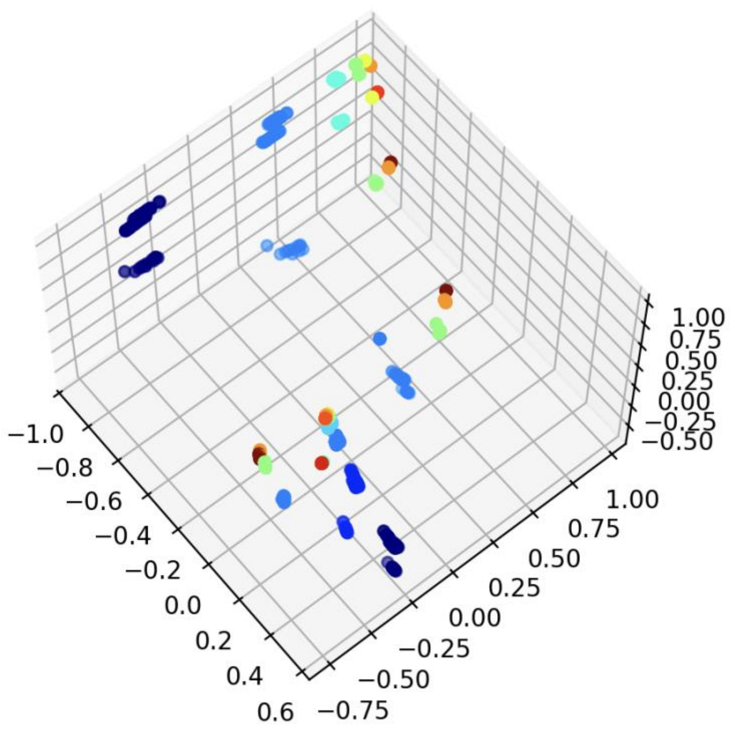
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6



1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6



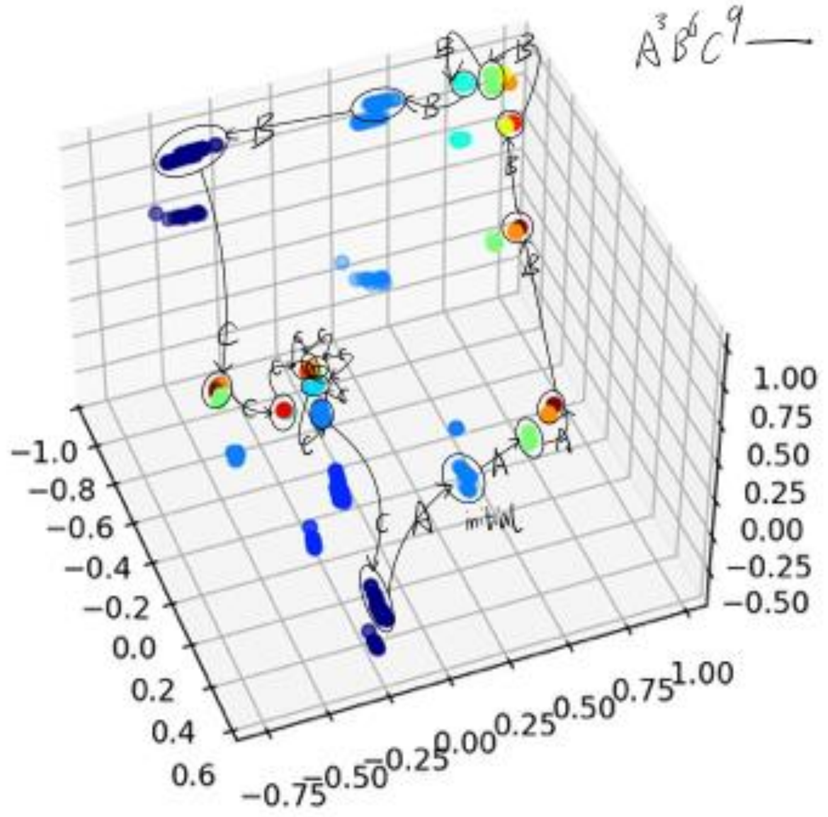
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



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

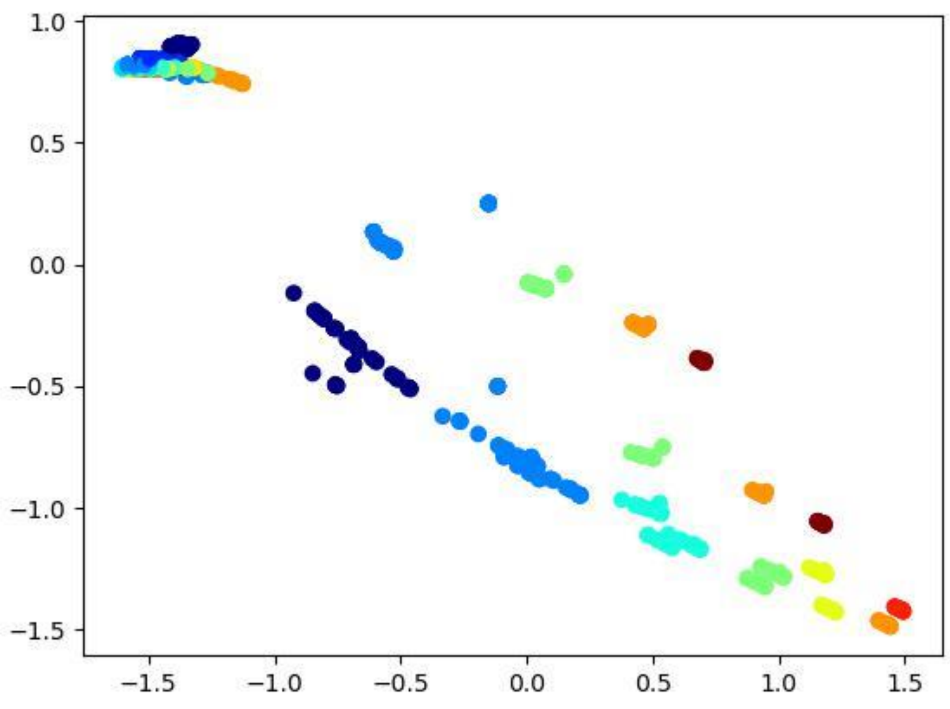
Question 5

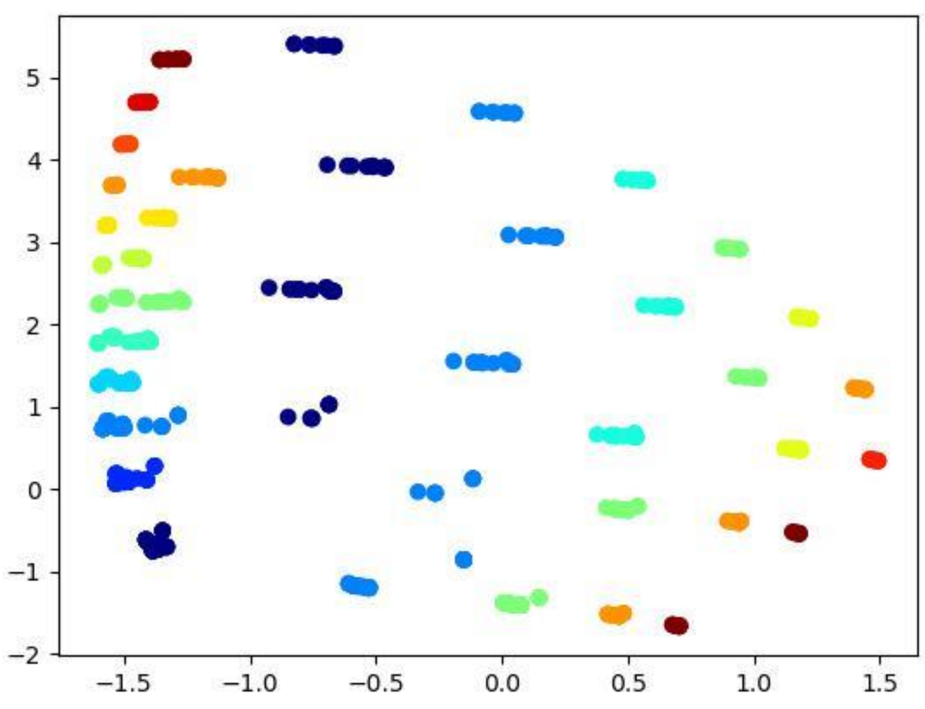
Hidden units:

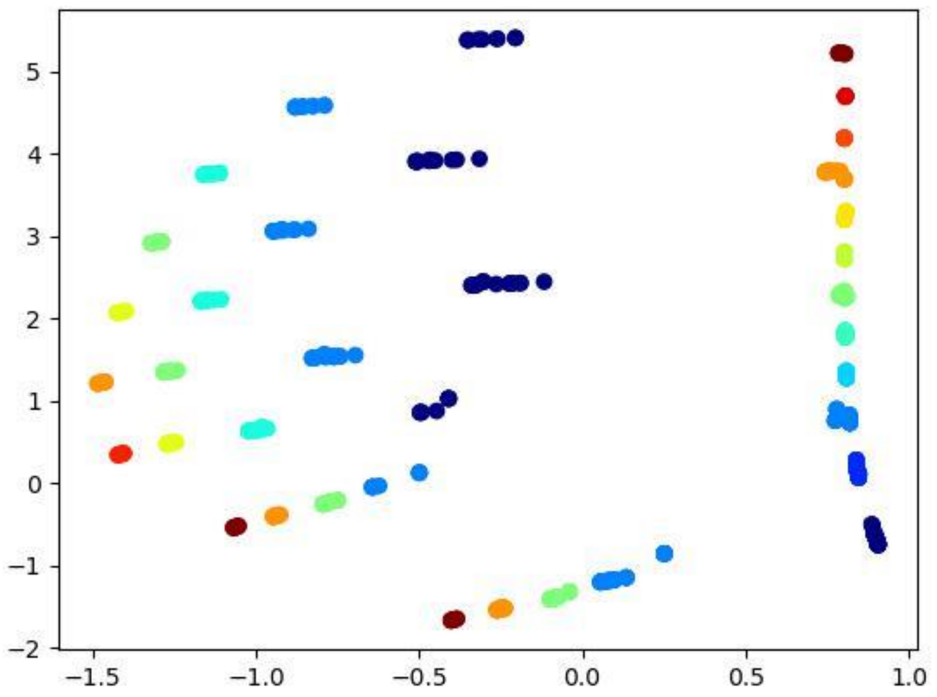


The hidden state (h_t) functions as a short-term mode indicator. The 3D activation plots required by Question 4 show that the hidden state occupies separate, well-defined regions of the state space, with each region corresponding to one of the A, B, or C segments of the sequence. This indicates that h_t is primarily concerned with the network's immediate action—such as predicting the current character type—rather than tracking the long-term count. Its location within a specific cluster effectively signals the current "phase" of the sequence to the output layer.

Context units:

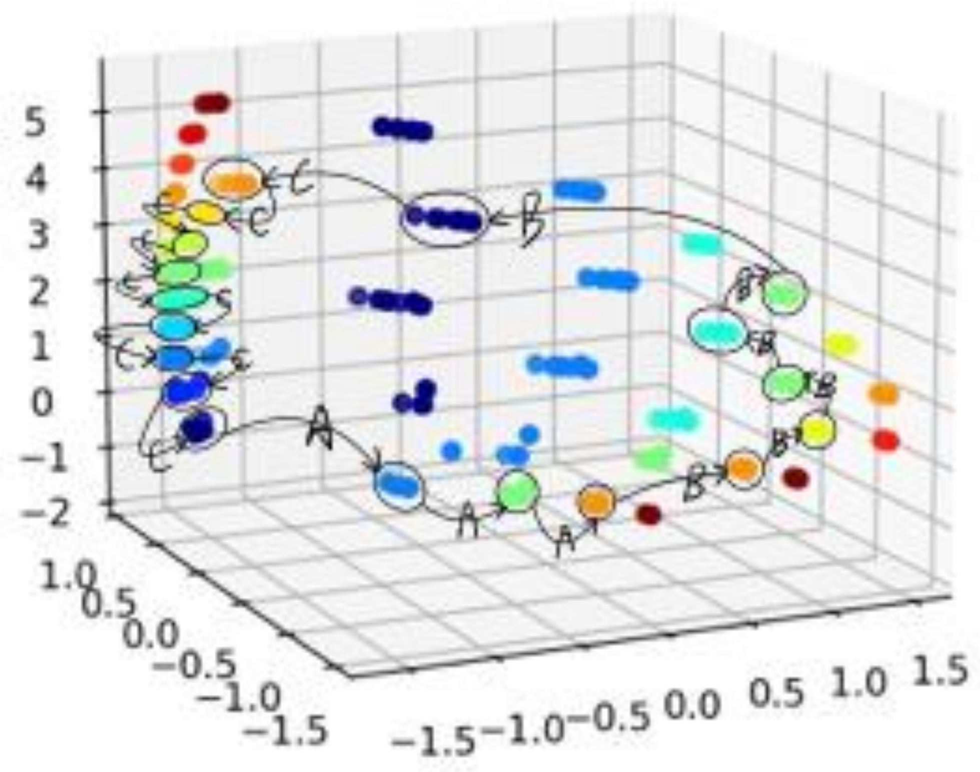






5
4
3
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0

$A^3B^6C^9$ —



C
C
B
B
B
B
B
A
A
A
C
5
4
3
2
1
0
−1
−2
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

In contrast, the cell state (c_t) serves as the network's long-term memory, as revealed by the analysis for Question 5. Its visualization shows not distinct clusters, but a single, continuous path through the state space. Along this trajectory, specific dimensions of the c_t vector exhibit monotonic behavior—steadily increasing during the 'A' phase, then systematically decreasing at different rates during the 'B' and 'C' phases. This strongly suggests that the cell state has learned to function as an analog counter, encoding both the value of n and the progress within each segment of the sequence.

In conclusion, the LSTM's success is attributable to this functional specialization. The cell state (c_t) reliably maintains the long-range context, such as the value of n, while the hidden state (h_t) uses this context to make localized, moment-to-moment decisions about the output. This separation of duties—where c_t handles the memory and h_t handles the action—is the core mechanism that allows the LSTM to generalize its learned structure to different sequence lengths and solve the complex $a^n b^{2n} c^{3n}$ task.