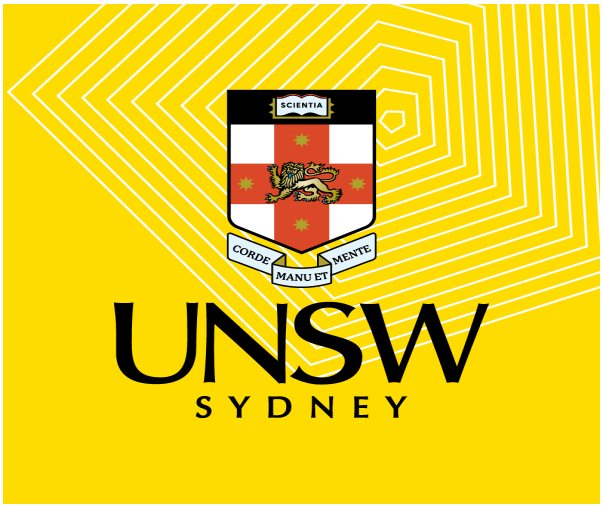


COMP9444 Neural Networks and Deep Learning

Assignment 1

Term 2, 2024

Submitted by

zID: z5524306

Name: Ziyu Yang

I declare that:

This assessment item is entirely my own original work, except where I have acknowledged use of source material [such as books, journal articles, other published material, the Internet, and the work of other student/s or any other person/s.

This assessment item has not been submitted for assessment for academic credit in this, or any other course, at UNSW or elsewhere.

I understand that:

The assessor of this assessment item may, for the purpose of assessing this item, reproduce this assessment item and provide a copy to another member of the University.

The assessor may communicate a copy of this assessment item to a plagiarism checking service (which may then retain a copy of the assessment item on its database for the purpose of future plagiarism checking).

I certify that I have read and understood the University Rules in respect of Student Academic Misconduct.

Ziyu Yang, z5524306, signature and date

Part 1: Japanese Character Recognition

Note: *To reproduce the same result in this project, random seed has been set in the code, using" torch.manual_seed(666)"*

1. **Answer question 1**

   The confusion matrix is:

   [[763. 5. 8. 13. 32. 63. 2. 63. 30. 21.]

   [7. 670. 107. 18. 26. 23. 57. 15. 24. 53.]

   [5. 59. 694. 27. 29. 20. 45. 36. 47. 38.]

   [4. 39. 58. 755. 16. 57. 15. 18. 26. 12.]

   [60. 53. 77. 21. 624. 21. 32. 35. 20. 57.]

   [8. 27. 123. 15. 19. 726. 28. 8. 34. 12.]

   [5. 21. 148. 9. 28. 24. 721. 21. 9. 14.]

   [17. 29. 29. 10. 83. 17. 53. 626. 89. 47.]

   [10. 37. 96. 41. 9. 31. 42. 7. 707. 20.]

   [8. 50. 85. 3. 54. 33. 18. 30. 41. 678.]]

   Test set: Average loss: 1.0094,
   Accuracy: 6964/10000 (70%)

2. **Answer question 2**

   After trying a lot of hidden layer nodes, the outcome of the model's final accuracy are as follows. When the number of hidden nodes is 220, the accuracy reaches the summit.

Hidden Layer Node Number	Accuracy (%)
20	76.32
40	79.76
80	83.10
100	83.59
140	84.23
180	84.78
220	85.06
260	84.99
360	84.57
500	84.66

When the Hidden layer Node Number is 220:

The confusion matrix is:

[[867. 3. 1. 5. 28. 21. 3. 36. 29. 7.]
[6. 833. 25. 2. 15. 12. 53. 7. 16. 31.]
[9. 11. 841. 44. 14. 18. 22. 9. 18. 14.]
[3. 11. 21. 928. 1. 16. 4. 1. 6. 9.]
[43. 28. 20. 4. 822. 8. 22. 16. 20. 17.]
[11. 14. 77. 9. 12. 831. 21. 2. 16. 7.]
[3. 15. 51. 7. 14. 6. 881. 11. 2. 10.]
[22. 17. 21. 5. 23. 7. 30. 825. 19. 31.]
[11. 30. 29. 47. 3. 9. 26. 3. 831. 11.]
[4. 15. 50. 6. 30. 4. 15. 14. 15. 847.]]

Test set: Average loss: 0.4907,

Accuracy: 8506/10000 (85%)

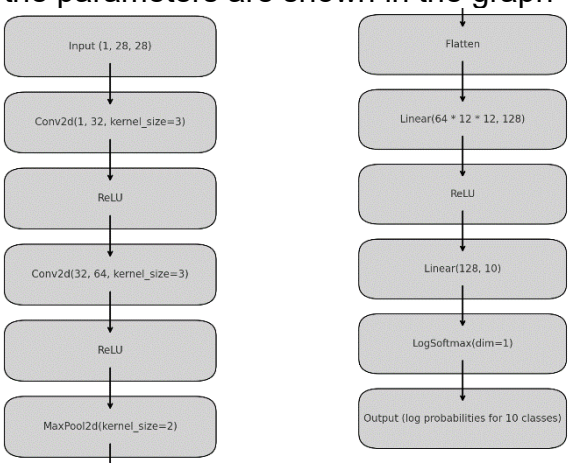
Calculation of total parameters

1) Input layer to hidden layer
   a) Input nodes: 28 x 28= 784
   b) Hidden layer: 784 x 220
   c) Numbers of bias: 220
   d) Total parameter of this layer = 784 x 220+220=172700
2) Hidden layer to Output layer
   a) Hidden layer nodes: 220
   b) Output nodes: 10
   c) Numbers of weights: 220 x 10
   d) Total parameter of this layer = 220 x 10+10=2210

3) Total parameters for both layers = 172700 +2210=174910

3. Answer question 3

In my convolutional layer, the parameters are shown in the graph



The confusion matrix is:

[[950. 3. 1. 0. 26. 2. 1. 10. 5. 2.]

[3. 905. 5. 0. 17. 1. 39. 4. 4. 22.]

[11. 13. 885. 28. 8. 10. 18. 5. 9. 13.]

[3. 0. 12. 962. 4. 4. 9. 1. 2. 3.]

[21. 7. 1. 6. 939. 0. 11. 1. 9. 5.]

[8. 6. 38. 5. 6. 903. 22. 0. 1. 11.]

[4. 4. 13. 2. 11. 1. 959. 5. 0. 1.]

[7. 6. 3. 0. 3. 2. 18. 933. 4. 24.]

[1. 20. 9. 4. 11. 3. 15. 1. 925. 11.]

[10. 2. 10. 4. 13. 3. 12. 4. 9. 933.]]

Test set: Average loss: 0.3030,
Accuracy: 9294/10000 (93%)

Calculation of total parameters:

1) Convolutional Layer 1 (conv1)
   - Input channels: 1
   - Output channels: 32
   - Kernel size: 3x3

   Number of weight parameters: =1 x 32 x 3 x 3 = 288

   Number of bias parameters: 32
2) Convolutional Layer 2 (conv2)
   - Input channels: 32

• Output channels: 64

• Kernel size: 3x3

Number of weight parameters:32 x 64 x 3 x 3 = 18432

Number of bias parameters:64

3. Fully Connected Layer 1 (fc1)

• Input features: 64 x 12 x 12

• Output features: 128

Number of weight parameters:64 x 12 x 12 x 128 = 1179648

Number of bias parameters:128

4. Fully Connected Layer 2 (fc2)

• Input features: 128

• Output features: 10

Number of weight parameters: 128 x 10 = 1280

Number of bias parameters: 10

Summary

Adding up the number of parameters for each layer gives us the total number of independent parameters in the network:

- Parameters in Convolutional Layer 1: 288 (weights) + 32 (biases) = 320
- Parameters in Convolutional Layer 2: 18432 (weights)+ 64 (biases) = 18496
- Parameters in Fully Connected Layer 1: 1179648 (weights) + 128 (biases) = 1179776
- Parameters in Fully Connected Layer 2: 1280 (weights)+ 10 (biases) = 1290

Total number of parameters: 320 + 18496 + 1179776 + 1290 = 1199882

4. Answer question 4

1) After analysing of the three models' output, it could be seen that Conv > fully connected 2-layer network>1-layer linear network.

   The analysis is as follows.

   a) NetLin (linear model):

   it is the simplest linear model with only one fully connected layer. Due to its simple structure, it usually performs poorly when dealing with complex datasets. Its accuracy will probably be the lowest of the three models.

b) NetFull (fully connected model):

this model has two fully connected layers and uses the tanh activation function. It has more complexity and representational power compared to the linear model, so the accuracy will be higher than the linear model in most cases.

c) NetConv (Convolutional Neural Network):

this model contains two convolutional layers and one fully connected layer using the ReLU activation function. The convolutional layer can efficiently extract spatial features of the image and therefore usually performs best and has the highest accuracy when processing image data.

2) Compare parameters

NetLin:

Number of parameters: 28 x 28 x 10 + 10 (bias) = 7840 + 10 = 7850

NetFull:

a) Layer 1: 28 x 28 x 220+220(bias)= 172,700
b) Layer 2: 220 x 10+10 (bias) =2210
c) Total: 172,700 + 2210 = 174,910

NetConv:

a) Layer 1 convolution: 3 x 3 x 1 x 32 + 32 (bias) = 288 + 32 = 320
b) Second layer convolution: 3 x 3 x 32 x 64 + 64 (bias) = 18432 + 64 = 18496
c) First layer fully connected: 64 x 12 x 12 x 128 + 128 (bias) = 1179648 + 128 = 1179776
d) Second layer fully connected: 128 * 10 + 10 (bias) = 1280 + 10 = 1290
e) Total: 320 + 18496 + 1179776 + 1290 = 1199882

Model accuracy is positively correlated with the number of unique model parameters, and increasing the number of model parameters is beneficial to the improvement of accuracy. However, a small dataset meets a complex model, which may lead to the overfitting problem. In addition, the learning rate, batch size and other hyperparameters also affect the model evaluation accuracy.

3) Analysis of errors and failures

First, we get some definitions as follows.

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

NetLin:

The confusion matrix is:

[[763. 5. 8. 13. 32. 63. 2. 63. 30. 21.]

[7. 670. 107. 18. 26. 23. 57. 15. 24. 53.]

[5. 59. 694. 27. 29. 20. 45. 36. 47. 38.]

[4. 39. 58. 755. 16. 57. 15. 18. 26. 12.]

[60. 53. 77. 21. 624. 21. 32. 35. 20. 57.]

[8. 27. 123. 15. 19. 726. 28. 8. 34. 12.]

[5. 21. 148. 9. 28. 24. 721. 21. 9. 14.]

[17. 29. 29. 10. 83. 17. 53. 626. 89. 47.]

[10. 37. 96. 41. 9. 31. 42. 7. 707. 20.]

[8. 50. 85. 3. 54. 33. 18. 30. 41. 678.]]

- 0 is incorrectly recognised as 9: because both characters have a rounded structure.
- 1 is incorrectly recognised as 7: both characters are elongated and have similar shapes.
- 3 is incorrectly recognised as 5: both characters have similar curves

Netfull:

The confusion matrix is:

[[867. 3. 1. 5. 28. 21. 3. 36. 29. 7.]

[6. 833. 25. 2. 15. 12. 53. 7. 16. 31.]

[9. 11. 841. 44. 14. 18. 22. 9. 18. 14.]

[3. 11. 21. 928. 1. 16. 4. 1. 6. 9.]

[43. 28. 20.   4. 822.   8. 22. 16. 20. 17.]
[11. 14. 77.   9. 12. 831. 21.   2. 16.   7.]
[3. 15. 51.   7. 14.   6. 881. 11.   2. 10.]
[22. 17. 21.   5. 23.   7. 30. 825. 19. 31.]
[11. 30. 29. 47.   3.   9. 26.   3. 831. 11.]
[4. 15. 50.   6. 30.   4. 15. 14. 15. 847.]]

- 0 was incorrectly recognised as 9: Despite improvements, there is still an error because both characters have similar rounded shapes.
- 4 was incorrectly recognised as 1: probably because some of the writing forms of 4 are similar to 1.
- 5 was incorrectly recognised as 3: the curved part of both characters caused confusion.

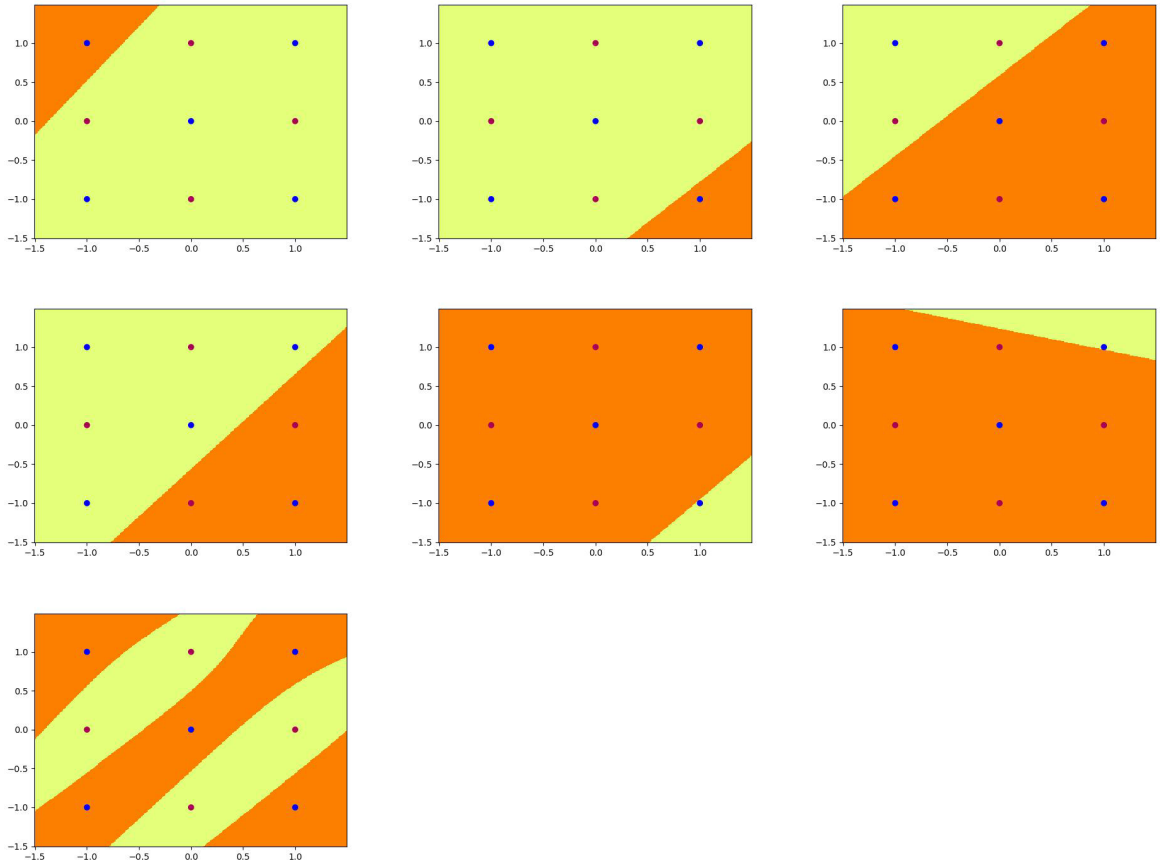
Netconv:

The confusion matrix is:
[[950.   3.   1.   0. 26.   2.   1. 10.   5.   2.]
[3. 905.   5.   0. 17.   1. 39.   4.   4. 22.]
[11. 13. 885. 28.   8. 10. 18.   5.   9. 13.]
[3.   0. 12. 962.   4.   4.   9.   1.   2.   3.]
[21.   7.   1.   6. 939.   0. 11.   1.   9.   5.]
[8.   6. 38.   5.   6. 903. 22.   0.   1. 11.]
[4.   4. 13.   2. 11.   1. 959.   5.   0.   1.]
[7.   6.   3.   0.   3.   2. 18. 933.   4. 24.]
[1. 20.   9.   4. 11.   3. 15.   1. 925. 11.]
[10.   2. 10.   4. 13.   3. 12.   4.   9. 933.]]

- 1 was incorrectly recognised as 7: despite the best performance, confusion with elongated characters still exists.
- 9 was incorrectly identified as 0: the circular structure of the two characters could still lead to confusion.
- 5 was incorrectly identified as 3: the curved part of the two characters led to confusion.

Part 2: Multi-Layer Perceptron

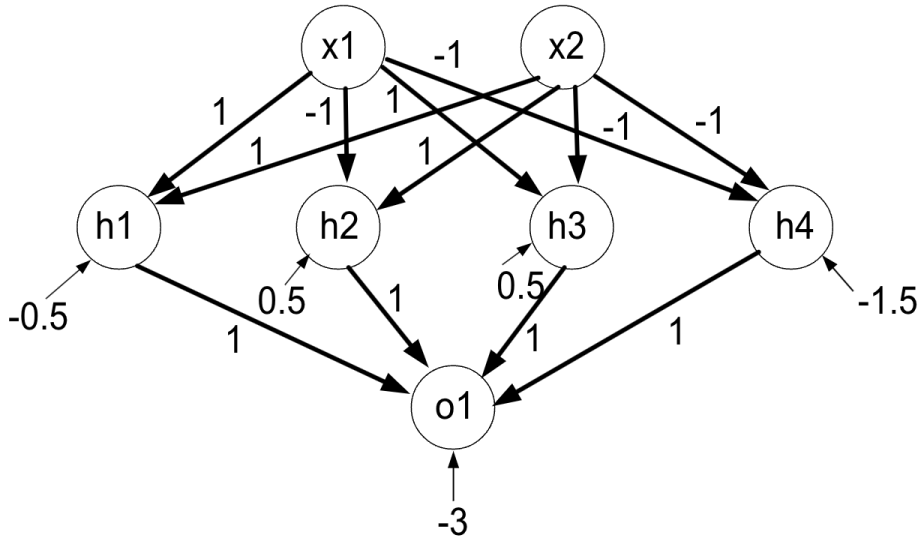
1. Answer question 1



These are figures from figure 0 to figure 6, I tried 3 times to get this result. We can use Weight initialization, optimization algorithms, learning rate scheduling, regularization, data augmentation, batch normalization, multiple training runs, model ensemble to prevent being stuck in a local minimum.

2. Answer question 2

Network design & node function:



1. *First hiddennode* : $1 \cdot x_1 + 1 \cdot x_2 - 0.5 = 0 \quad x_1 + x_2 = 0.5$
2. *Second hiddennode* : $-1 \cdot x_1 + 1 \cdot x_2 + 0.5 = 0 \quad -x_1 + x_2 = -0.5$
3. *Third hiddennode* : $1 \cdot x_1 - 1 \cdot x_2 + 0.5 = 0 \quad x_1 - x_2 = -0.5$
4. *Fourth hiddennode* : $-1 \cdot x_1 - 1 \cdot x_2 - 1.5 = 0 \quad -x_1 - x_2 = 1.5$

Weights and bias setup:

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

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

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

out_bias = [-3]

Result calculation:

(x1, x2)	hidden1	hidden2	hidden3	hidden4	O1
(-1, -1)	0	1	1	1	1
(0, -1)	0	0	1	1	0
(1, -1)	1	0	1	0	0
(-1, 0)	0	1	0	1	0
(0, 0)	0	1	1	0	0
(1, 0)	1	0	1	0	0
(-1, 1)	0	1	0	0	0
(0, 1)	1	1	0	0	0
(1, 1)	1	1	1	0	0

Green means correct detection, and red means wrong detection.

3. Answer question 3

Final weights setup:

in_hid_weight = [[10, 10], [-10, 10], [10, -10], [-10, -10]]

hid_bias = [-5, 5, 5, -15]

hid_out_weight = [[10, 10, 10, 10]]

out_bias = [-30]

graph outputs

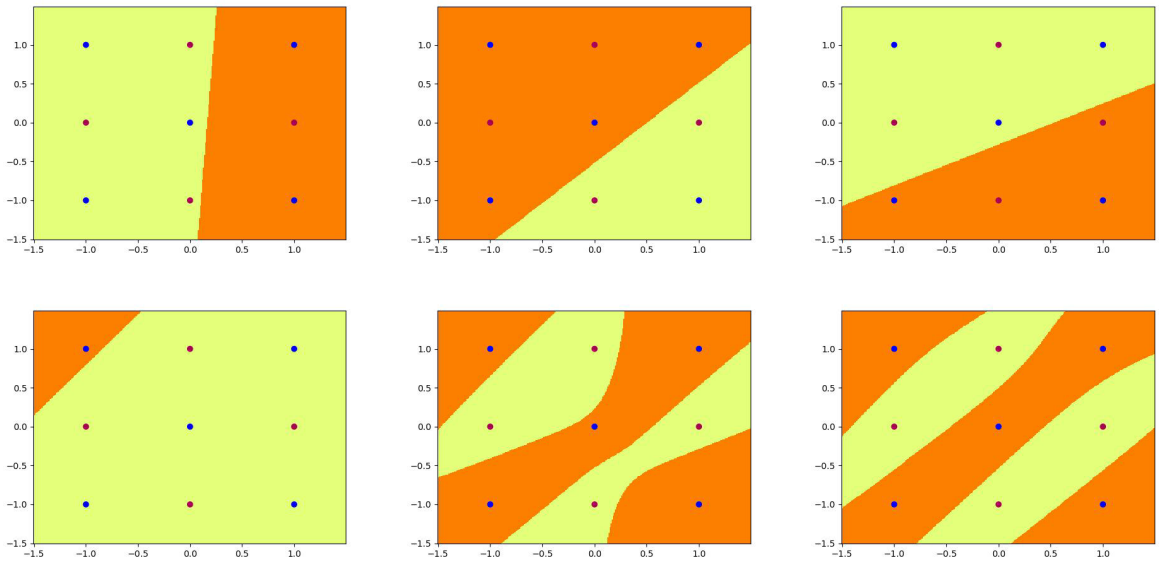


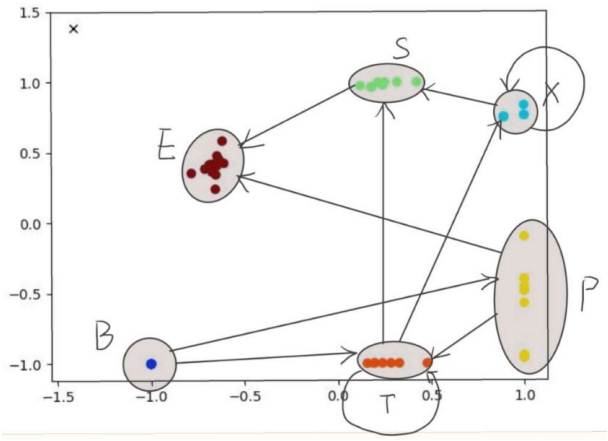
Figure 0 to figure 3 for hidden nodes.

Figure 4 to figure 5 for outputs.

Part 3: Hidden Unit Dynamics for Recurrent Networks

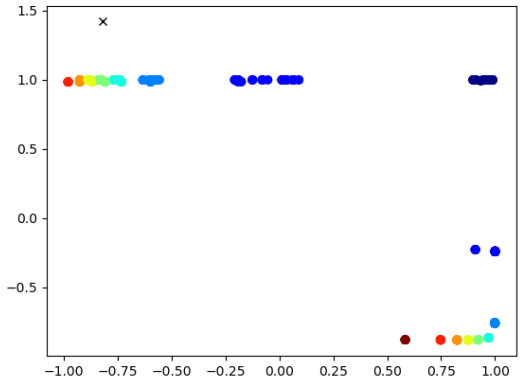
1. Answer question 1

The structure is shown in the following graph.



2. Answer question 2

The result graph shows:



In the $a^n b^n$ prediction task, the simple recurrent network (SRN) achieves accurate sequence prediction by dynamically adjusting its hidden unit activations as it processes the input string. Based on the generated figure, we can observe how these activations change and enable the network to predict the sequences correctly.

Hidden Unit Activations and Sequence Processing

1) **Processing 'a'**:

- The hidden unit activations for 'a' are shown as clusters of points at the top left of the plot.
- Each time an 'a' is processed, the activations move within this region, indicating that the network is incrementing a count of the 'a's seen so far.

2) **Processing 'b'**:

- When 'b' is processed, the hidden unit activations shift to a new region, seen as clusters on the right side of the plot.
- Each 'b' input causes the activations to move along this new trajectory, decrementing the count and matching it against the previously counted 'a's.

Predicting the Last B and Following A

- The network's ability to transition activations between distinct regions for 'a' and 'b' inputs allows it to maintain a correct count of both characters.

- The last 'b' in each sequence can be accurately predicted because the network ensures that the count of 'b's matches the count of 'a's by the time it processes the last 'b'.
- The correct prediction of the sequence is demonstrated by the distinct and orderly shifts in the activation clusters, ensuring that the counts are tracked accurately.

By effectively encoding the counts of 'a's and 'b's in its hidden unit activations and maintaining distinct regions for each character, the SRN can predict the $a^n b^n$ sequences correctly.

This capability demonstrates the network's proficiency in handling counting tasks and sequential pattern recognition.

3. Answer question 3

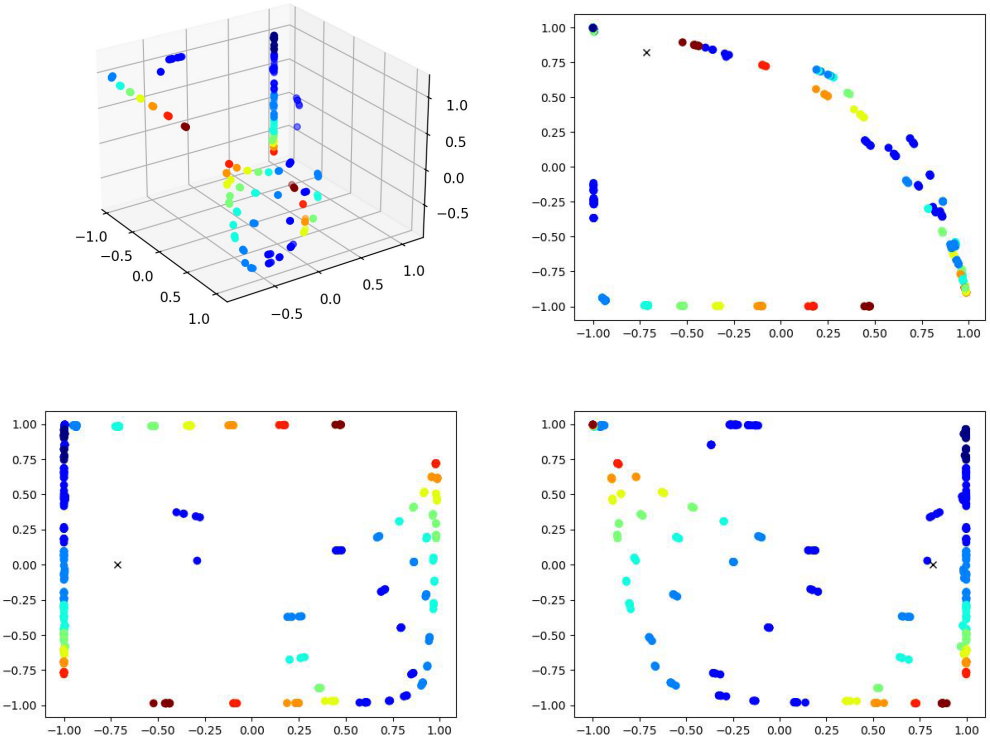


Image 2:

In Image 2, each point represents the hidden unit activations of the network at a specific input character. Different colours indicate different input characters:

- Blue Points: These points represent the hidden unit activations when the network processes 'a'. They cluster on the left side of the plot and move along a trajectory as more 'a's are input, indicating that the network is counting the number of 'a's.
- Red Points: These points represent the hidden unit activations when the network processes 'b'. They start from the blue cluster area and move into a new region, indicating that the network begins to decrement the count of 'a's and increment the count of 'b's.
- Yellow Points: These points represent the hidden unit activations when the network processes 'c'. They move from the red cluster into another new region, indicating that the network is counting the number of 'c's and ensuring it matches the previous counts of 'a's and 'b's.

Image 3:

In Image 3, we see the hidden unit activations as the network processes longer sequences:

- Blue Points: Continue to represent the activations for 'a'. As more 'a's are input, the state points follow a distinct trajectory.
- Red Points: When 'b' is input, the state points shift from the blue region to a new trajectory, showing that the network is counting 'b's.
- Yellow Points: When 'c' is input, the state points shift from the red region to a new trajectory, showing that the network is counting 'c's and matching the count with the previous 'a's and 'b's.

Image 4:

In Image 4, the network processes a complete $a^n b^n c^n$ sequence, showing the hidden unit activations for the entire sequence:

- Blue Points: Represent the activations for all input 'a's, forming a cluster in the plot.

- Red Points: Represent the activations for all input 'b's. The state points shift from the blue cluster, indicating the network is decrementing the count of 'a's and counting 'b's.
- Yellow Points: Represent the activations for all input 'c's. The state points shift from the red cluster, indicating the network is counting 'c's and ensuring the count matches the previous 'a's and 'b's.

Summary

Based on these images, we can conclude:

- Processing 'a': The hidden unit activations change within the blue region, counting the number of 'a's.
- Processing 'b': The activations shift significantly to the red region, counting 'b's and ensuring they match the count of 'a's.
- Processing 'c': The activations shift to the yellow region, counting 'c's and matching the counts of both 'a's and 'b's.

This mechanism allows the recurrent neural network to accurately predict the sequences in $a^n b^n c^n$. The hidden unit activations encode the counts of 'a's, 'b's, and 'c's, enabling the network to predict the last 'B', all 'C's, and the subsequent 'A' correctly. This demonstrates the network's capability to learn and handle complex sequence patterns and counting tasks.

4. Answer question 4

1) Analyse:

The Reber task involves generating and recognizing strings that follow Reber grammar. LSTM (Long Short-Term Memory) networks manage the flow of sequence information through their gating mechanisms to accomplish this task. Below is a detailed explanation of LSTM behaviour and how it completes the Reber task.

2) Behavior of LSTM in the Reber Task

- Initialization:

At the beginning of processing the string, LSTM has an initial hidden state h_0 and cell state c_0, usually initialized to zero. These states are used to store and transmit sequence information.

- Processing the input sequence:

When the LSTM processes each character, the input gate writes information about the current character into the cell state, the forget gate decides to keep or discard previous information, and the output gate decides which information is passed to the hidden state.

For example, when processing the string "BT", the LSTM learns 'B' and then predicts the next character 'T' according to the rules.

- Predicting the next character:

According to Reber syntax, LSTM predicts the next character using the information in the cell state. For example, after processing 'B', the possible characters are 'T' or 'P'. LSTM learns these rules and adjusts its cell state and hidden state to reflect the possible outputs.

- tracking sequence progress:

The LSTM tracks the progress of the sequence through its internal state to ensure that the generated string conforms to the Reber syntax. For example, after processing 'BT', the LSTM predicts the next character that should conform to the rules, such as 'S' or 'X'.