pCOMP9444 Assignment 1 – Report

Name: Yiming Zhu

zID: z5571436

Part 1:

Part 1. 1NetLin

Model structure: Single-layer linear fully connected, with an output of log softmax.

```
class NetLin(nn.Module):
  def __init__(self):
    super(NetLin, self).__init__()
    self.fc = nn.Linear(28 * 28, 10)

  def forward(self, x):
    x = x.view(-1, 28 * 28)
    return F.log_softmax(self.fc(x), dim=1)
```

final test accuracy:

Accuracy: 6961 / 10000 = 69.61%

Confusion matrix (taken from epoch 10) :

```
[[765.  6.  9. 13. 31. 63.  2. 62. 30. 19.]
 [ 7. 668. 109. 18. 28. 23. 57. 16. 24. 50.]
 [ 7. 61. 688. 27. 28. 22. 46. 36. 46. 39.]
 [ 4. 34. 59. 758. 17. 57. 15. 19. 27. 10.]
 [ 59. 53. 77. 21. 625. 22. 31. 35. 20. 57.]
 [ 8. 28. 122. 17. 19. 727. 29. 7. 33. 10.]
 [ 5. 23. 145. 11. 26. 24. 721. 21. 11. 13.]
 [ 14. 29. 27. 11. 85. 17. 53. 626. 91. 47.]
 [ 10. 37. 93. 42. 8. 32. 45. 5. 704. 24.]
 [ 7. 49. 86. 3. 54. 31. 20. 30. 41. 679.]]
```

Part 1.2. NetFull

Model structure:

- Input layer: 784 (28×28 image pixels)
- Hidden Layer: 100 nodes, activated using tanh
- Output layer: 10 types, using log_softmax to output probabilities

```
class NetFull(nn.Module):
  def __init__(self, hidden_size=100):
    super(NetFull, self).__init__()
    self.fc1 = nn.Linear(28 * 28, hidden_size)
```

```
        self.fc2 = nn.Linear(hidden_size, 10)
    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = torch.tanh(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Accuracy: 8384 / 10000 = 83.84%

Confusion matrix (10th epoch) :

```
[[855.  5.  2.  4. 27. 25.  4. 41. 33.  4.]
 [ 5. 808. 24.  5. 20.  8. 70.  8. 23. 29.]
 [ 7. 10. 841. 41. 13. 16. 29. 13. 18. 12.]
 [ 2.  6. 34. 915.  0. 15.  6.  8.  6.  8.]
 [43. 23. 25.  7. 802.  9. 35. 16. 21. 19.]
 [ 8. 10. 94. 11. 11. 814. 25.  1. 17.  9.]
 [ 3. 14. 51.  7. 17.  5. 882.  7.  3. 11.]
 [18. 12. 19.  4. 27.  9. 43. 806. 29. 33.]
 [ 8. 28. 24. 44.  4. 12. 37.  4. 829. 10.]
 [ 4. 18. 55.  3. 30.  4. 27. 13. 14. 832.]]
```

First layer parameters： 784 × 100 + 100 (Weight + bias)

second layer parameters： 100 × 10 + 10

Total number of parameters = 784×100 + 100 + 100×10 + 10 = 78400 + 100 + 1000 + 10 = 79510

Part 1.3. NetConv

Model structure design: NetConv is a classic CNN structure which use to classify picture. It's include convolutional layer 1, convolutional layer 2, connection layer and output layer.

- Conv1： Number of input channels: 1 (grayscale image), Number of output channels: 10 , Convolution kernel size: 5×5, Activation function: ReLU
  followed by: 2×2 Max pooling (pool1)
- conv2： Number of input channels: 10, Number of output channels: 20, Convolution kernel size: 5×5, Activation function: ReLU,
  followed by: 2×2 Max pooling (pool2)
- fully connection layer：
  Input dimension: 320 after flatten from the convolutional layer
  Output dimension: 10 (corresponding to 10 categories)
- output layerout： using log_softmax to output the logarithmic probability of each category

Layers	Calculate	The number of parameters
conv1	1 × 10 × 5 × 5 + 10	260

Layers	Calculate	The number of parameters
conv 2	10 × 20 × 5 × 5 + 20	5020
Fully connection layer	320 × 10 + 10	3210
Total		**8490**

Final accuracy rate (Test Set of Round 10) :89.59%
Average loss: 0.3656
Confusion matrix:
[[929 2 2 1 39 11 3 10 2 1]
[4 869 15 0 19 5 68 4 5 11]
[10 5 850 50 19 16 15 17 9 9]
[3 4 16 927 11 11 11 8 3 6]
[34 4 3 7 887 10 21 12 15 7]
[7 10 42 12 6 875 22 17 6 3]
[5 10 26 3 21 7 914 10 1 3]
[15 1 9 4 19 2 25 884 9 32]
[12 10 12 14 13 6 9 10 912 2]
[9 13 17 3 19 5 4 8 10 912]]

To conclusion, the most common confusion occurs between characters with similar shapes (such as class 0 and class 4).Compare with NetLin and NetFull, this model confusion is focused in the easily confused categories, and the accuracy rate has been significantly improved.

Analysis:

The accuracy of NetLin	70%
The accuracy of NetFull	83.84%
The accuracy of NetConv	**89.6%**

- **NetLin** is a simple linear model which only can capture input pixel linear combination , it cannot deal with non-linear structure between characters. This model has worse performance.
- **NetFull** has good performance via add a hidden layer and nonlinear activation， and its performance has been significantly improved.
- **NetConv** extracts local spatial characteristics by conv structure. It keeps less parameters and also get best performance which proves that modeling of image structure is more effective.

The parameters number of NetLin	7,850
The parameters number of NetFull	79,510
The parameters number of NetConv	**8,490**

- The parameter number of NetFull is the largest. But it is not with highest accuracy which indicates that too many parameters may lead to redundancy.
- NetConv relies on the weight sharing and local receptive field mechanism in convolution operations. It reduces parameters significantly whicle improves recognition accuracy. This is the model with the highest parameter efficiency.

Confusion matrix error analysis:

- All models has higher confusion between certain categories, especially between similar shape of characters, like o, re, ki and ma. In addition,the confusion of NetLin is boarder, which shows that a variety of categories cannot be distinguished.
- NetFull significantly reduces the errors of most categories, but certain categories still confusion seriously.
- NetConv has the most focused confusion matrix and the error distribution is closest to the ideal diagonal, which indicate that it can capture the local differences between characters better.

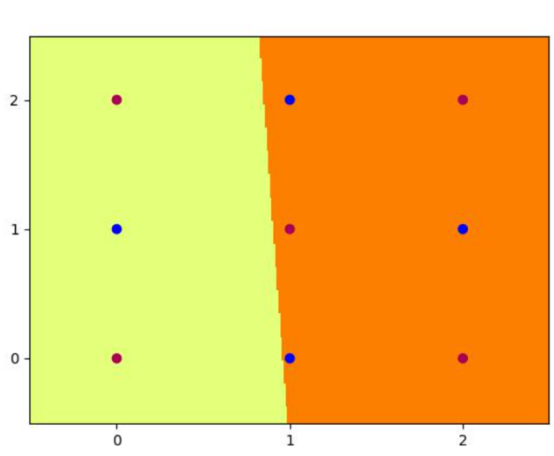
Part 2:

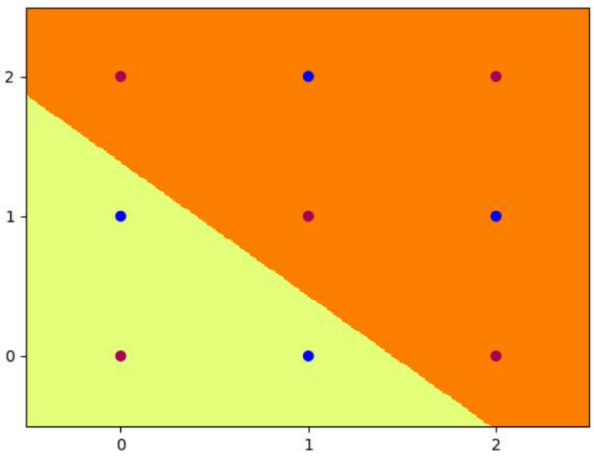
Part 2.1.MLP

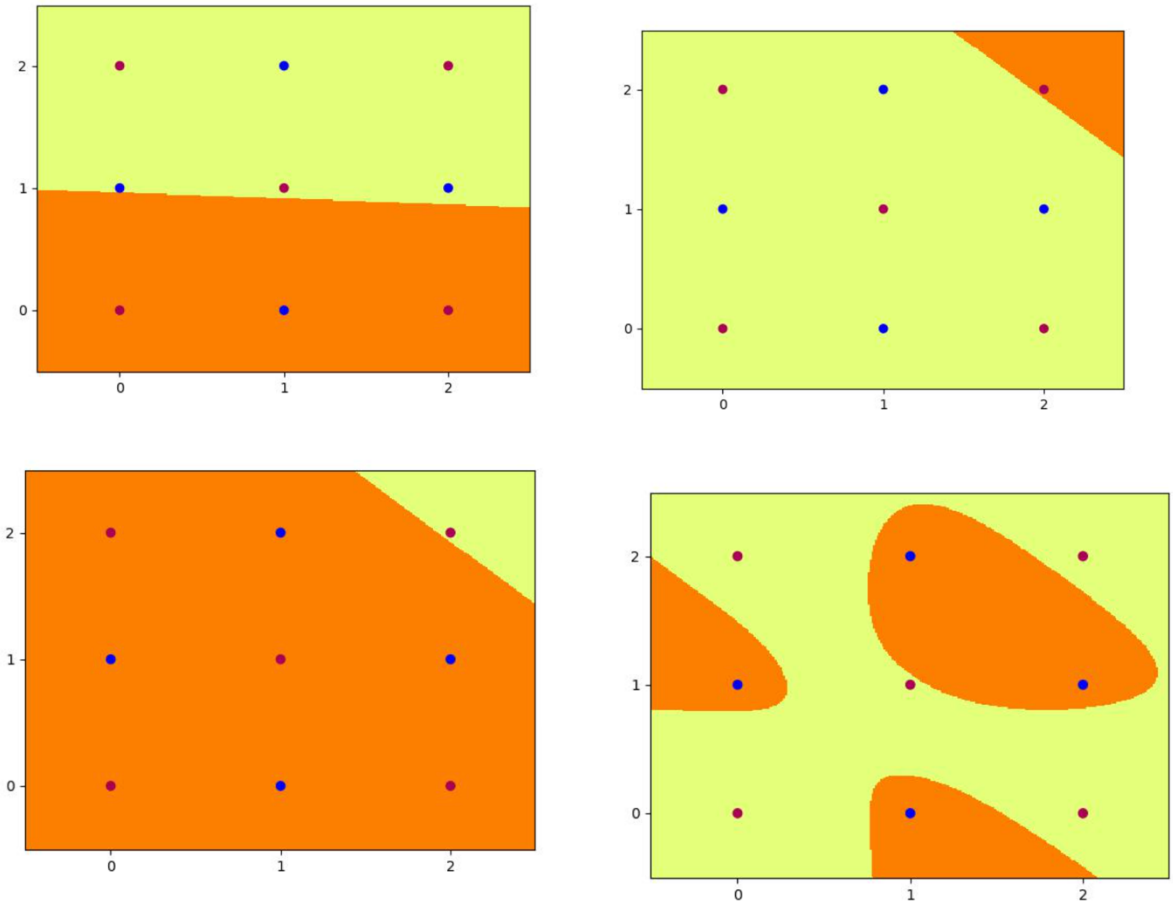
In Part 2.1, the a 2-layer neural network has been trained with 5 hidden units using sigmoid activation, which classify the dataset stored in check.csv.

```
Final Weights:
tensor([[ 5.5374,  5.5565],
        [ 0.0808, -4.0305],
        [-2.7526, -2.7529],
        [ 2.7677,  2.7676],
        [-4.1011,  0.0802]])
tensor([ -7.6509,   3.6446,  11.1285, -11.1739,   3.7199])
tensor([[ -8.6656,  -9.6047,  10.8095, -10.3538,  -9.5678]])
tensor([5.0466])
Final Accuracy:  100.0
```

The decision boundaries learned by the network and the functions computed by each hidden unit were visualized. The following plots were saved: plot/hid_5_0.jpg， plot/hid_5_1.jpg，plot/hid_5_2.jpg， plot/hid_5_3.jpg， plot/hid_5_4.jpg， plot/out_5.jpg







Part2.2

```
Initial Weights:
tensor([[1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.]])
tensor([-1., -2., -3., -4.])
tensor([[ 1., -1.,  1., -1.]])
tensor([-0.5000])
Initial Accuracy:  100.0
```

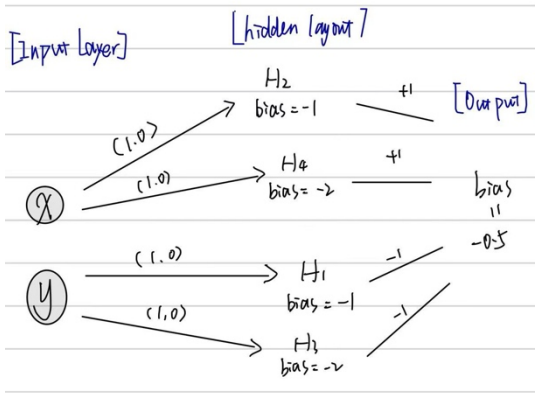


Figure 1Network Diagram

(x, y)	H1	H2	H3	H4	Output	Target
0, 0	0	0	0	0	0	0
0, 1	1	0	0	0	1	1
0, 2	1	1	0	0	0	0
1, 0	1	0	0	0	1	1
1, 1	1	1	0	0	0	0

(x, y)	H1	H2	H3	H4	Output	Target
1, 2	1	1	1	0	1	1
2, 0	1	1	0	0	0	0
2, 1	1	1	1	0	1	1
2, 2	1	1	1	1	0	0

Table 1 Activation Table

Hidden Node Decision Boundaries

Hidden Unit 1: $z1=1\times x+1\times y-1\geq 0$
Hidden Unit 2: $z2=1\times x+1\times y-2\geq 0$
Hidden Unit 3: $z3=1\times x+1\times y-3\geq 0$
Hidden Unit 4: $z4=1\times x+1\times y-4\geq 0$
When $z \geqslant 0$, hidden_j outputs 1, otherwise, output 0
a output=step(1·h1+(−1)·h2+1·h3+(−1)·h4−0.5)

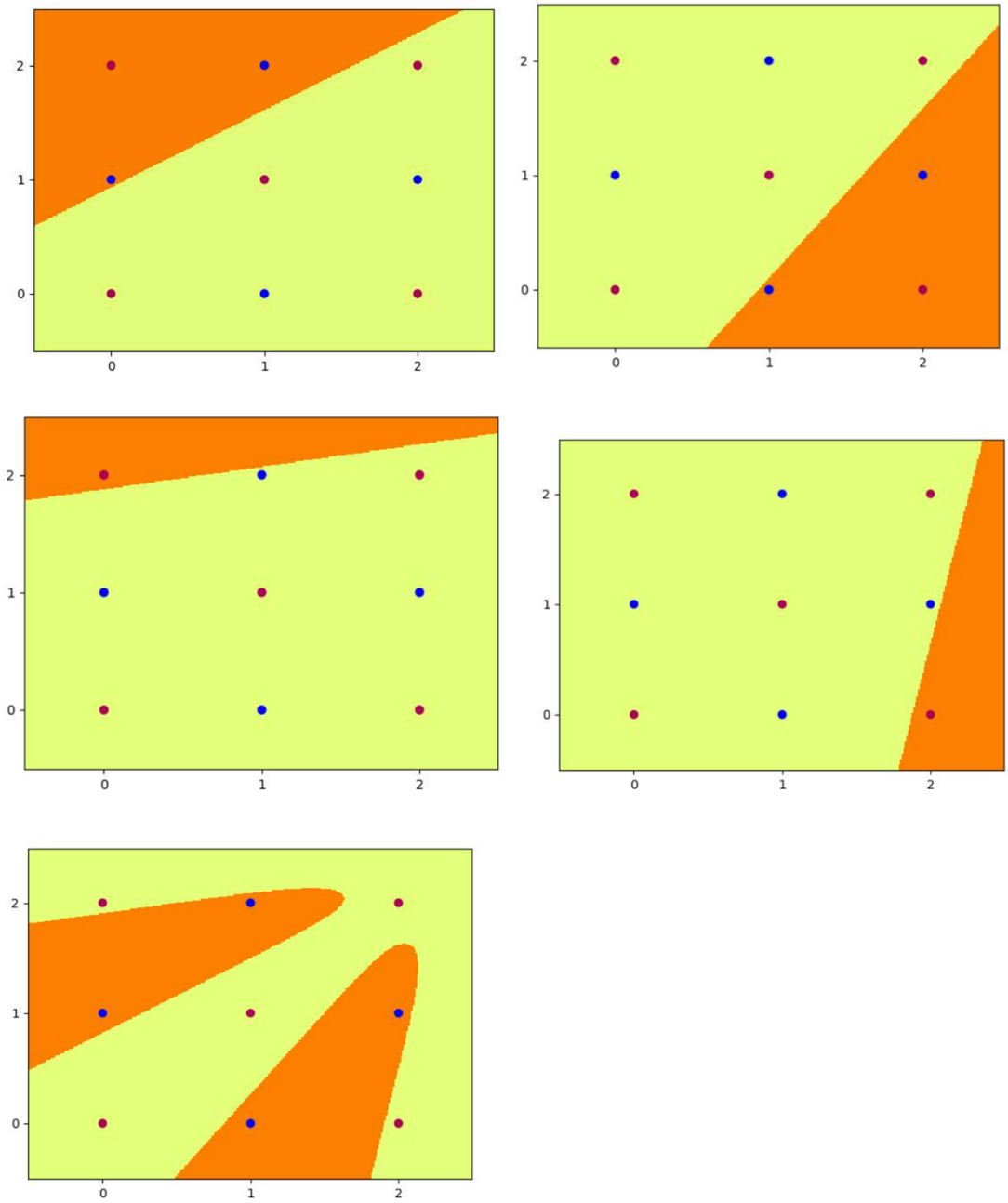
The weights of the output layer are [1, -1, 1, -1], and the bias is -0.5. The final network classified all 9 samples correctly with an accuracy rate of 100%.

Part2.3

I rescaled the weights and biases from the hand-crafted step network in Part 2.2 by a large factor so that the sigmoid activation behaves like a step function. This helps push activations toward either 0 or 1.

```
ep: 7400 loss: 0.0488 acc: 100.00
Final Weights:
tensor([[-5.0616,  7.4924],
        [ 7.4924, -5.0616],
        [-1.9825, 10.3817],
        [10.3817, -1.9825]])
tensor([ -6.9818,  -6.9818, -19.4931, -19.4931])
tensor([[ 11.1506,  11.1506, -13.5786, -13.5786]])
tensor([-3.3726])
Final Accuracy:  100.0
(base) vannizhu@Vannis-MacBook-Air-2 a1 %
```

The plots of hidden units and the overall network output confirm that the sigmoid network replicates the original step-function boundaries. The following plots were saved:
plot/hid_4_0.jpg， plot/hid_4_1.jpg， plot/hid_4_2.jpg， plot/hid_4_3.jpg，
plot/hid_4_4.jpg， plot/out_4.jpg



Part 3

Part 3.1

SRN has been trained with 2 hidden nodes on the anb2n language prediction task.
The final loss after 100,000 epochs was:

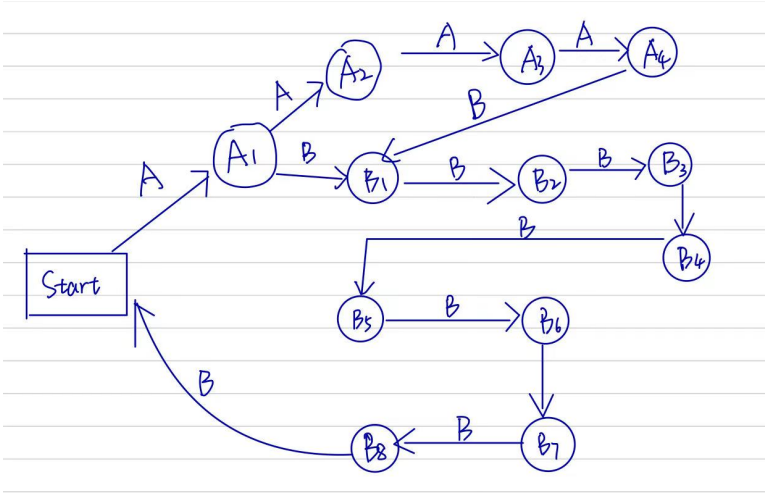
```
A [ 0.83   0.67] [ 0.83   0.17]
B [-1.     0.43] [ 0.   1.]
B [ 0.89   0.95] [ 0.85   0.15]
A [ 0.83   0.65] [ 0.82   0.18]
A [ 0.65   0.26] [ 0.64   0.36]
B [-1.    -0.12] [ 0.   1.]
B [ 0.37   0.69] [ 0.15   0.85]
B [-0.95   0.75] [ 0.   1.]
B [ 0.95   0.98] [ 0.9   0.1]
epoch: 100000
 loss: 0.0430
```

The hidden unit activations and output probabilities were printed for multiple test sequences. For example:

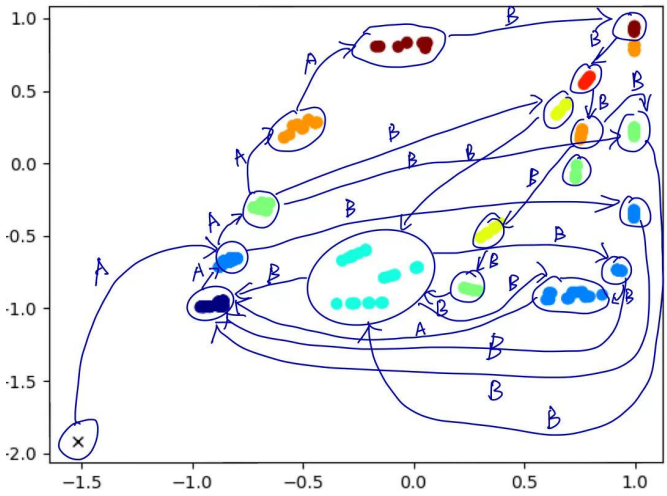
A [0.78 0.64] [0.77 0.23]
A [0.71 0.30] [0.73 0.27]
...
B [-1.   0.43] [0. 1.]
The network successfully converged with a loss below the required 0.05 threshold.

Part 3.2

For the anb2n language with n=4, the FSM consists of 4 states for reading A symbols (A1 to A4), followed by 8 states for reading B symbols (B1 to B8). After reading the last B, the machine transitions back to the start state, ready to process the next sequence. Each state corresponds to the number of symbols read so far. B.



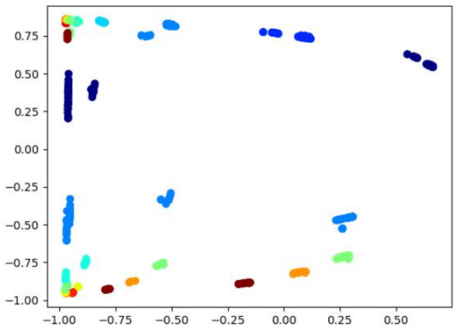
Below is the annotated state diagram showing how the SRN hidden units cluster into different states and transition between them for each input A or B. The initial state is marked with “x”.

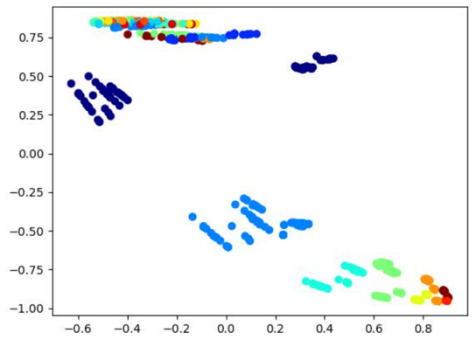


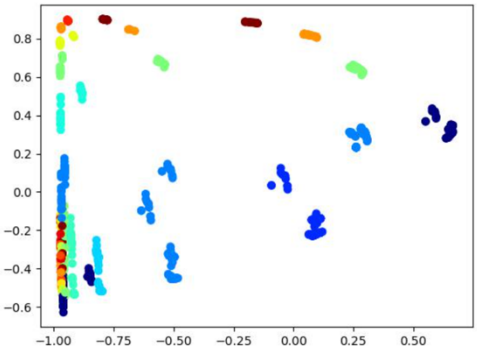
Part 3.3

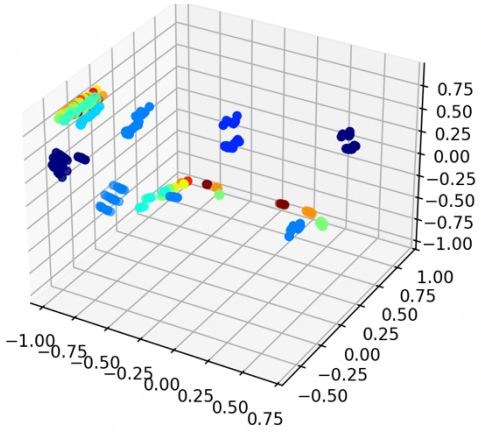
At anb2n task, SRN net predicts the input sequence accurately by hiding the changes in the activation values of the units. Specifically, when reading A, the activation value of hiding units will increase, then the number of A has been counted. But the number of hiding units has been decreased when input B. Which means, calculate the reciprocal of the previously recorded quantity of A to ensure that the output quantity of B is exactly twice that of A. After all B output, the net count becomes 0, then the next prediction should be A. Overall, this network stores information by hiding states to model and predict accurately.

Part 3.4









Part 3.5

In this task, the LSTM net with three hidden units has been trained to predict the sequence of anb^2nc^3n. This sequence includes three-section structure which are several A', then twice the number of B, and then three times the number of C. It requires segmented recognition ability, count accumulation, and symbol switching, which has high demands on the long-term and short-term memory of the model.

1.Cell state(short-term memory)

Cell state as the counter, the value of that gradually rising in A stage. The cell state continues to increase or remains which is twice in the quantity of A during B stage. At the stage of C, the value of cell state keep increasing which accumulates to three times the quantity of A. It rectify that the model's mastery of the proportion of the number of characters. It can be observed that cell3 rises from 0.49 to 3.0 and decreases at C stage. The numerical range of Hidden State is limited by tanh to (-1, 1), which can prevent gradient explosion.

2.Hidden State (long-term memory)

Hidden state mainly use to capture input stage. According to the visible picture, The hidden state forms clustering regions of different colours in the three-dimensional space (A,B,C), which indicates LSTM has learned to distinguish different semantic states in the hidden space. And there is obvious switching when switching from A to B or B to C. And The fluctuation range of the Cell State value is relatively large, which can reach approximately [-4, 5]

3. LSTM's gating mechanism

A input stage: The forgetting gate $f_t \approx 1$ retains previous memories. Then, Input the cumulative A quantity of gate i_t. And than output gate o_t is open, and the output is mainly dominated by H1/H3.

B input stage: The forgetting gate f_t retains part of the A information, and the input gate i_t rises strongly, which reflects an increase in the B count. And then output gate o_t switches to H2/H3 dominance, indicating that the state has changed from A to B.

C input stage: Forget Gate f_t clears the information of segment B and prepares to enter C. The input gate i_t rises to track the count of C. After that, output gate o_t is mainly dominated by H3 which reflects model is entering the C segment.

stage	Forget gate f_t	Input gate i_t	Output gate o_t	Behavior
A	f_t ≈ 1.0	i_t accumlate	o_t release H1/H3	Long-term memory accumulation

stage	Forget gate f_t	Input gate i_t	Output gate o_t	Behavior
B	f_t remains A	i_t rise	o_t release H2/H3	Switch and track B
C	f_t clears B	i_t track C	o_t release H3	Complete 3n output

4. Code modify

Name	Modify content	Reason
seq_models.py class LSTM_model → forward method	Initialize cell_seq, Save as cell state, returen cell state	- allowing store cell state while training - provide support to visualize and data analysis
seq_train.py	Extract the cell state during the training forward propagation	Ensure that the cell state can be obtained correctly during the training period
seq_train.py	extract cell state again	Ensure that the cell state is recorded throughout the training process
seq_train.py	print cell state	Used to monitor internal memory status of LSTM
seq_plot.py	record cell state data	Prepare for subsequent visualization
seq_plot.py	save cell state data	Implement the persistent storage of cell state
seq_plot.py	Realize 2D visualization	Show the variation trajectory of the cell state in the two-dimensional space
seq_plot.py	Realize 3D visualization	Display the state performance of LSTM in a high-dimensional space
anb2n.py	Print the training log	Output information (eg.hidden state, cell state, and output) probability in the training to facilitate analysis

5. Training process and result analysis：

At the beginning train, like 5k epoch, cell state fluctuates greatly. In addition, model haven't formed A distinct three-cluster state clustering, which is hard to distinguish A, B and C stages. After enter 20k epoch, hidden state cluster emerges, but still fluctuate.

After enter 50k epoch, the cell state shows a linear trend especially in the B stage, and the third cell state rises 1.0 unit per step continuously. Finally, the Loss converged to 0.012, and the probability of the model predicting symbols approached 1.0 when enter 100k epochs. Then, the model had fully mastered the sequence structure.

6. Conclusion

In conclusion, the grammar structure of anb^2nc^3n has been LSTM learned successfully and it flexibly switched between Hidden State and Cell State by using the gating mechanism to achieve accurate recognition of the three different stages. Additionally, cell state undertakes the key counter function and realizes the symbolic ratio counting of 1:2:3. In addition, hidden state separates the different dynamic modes of stages A, B, and C, which proves short-term memory and switching of the model. Moreover, T gating mechanism (f_t, i_t, o_t) is the key for the model to accurately grasp the long-term and short-term dependencies, which make sure that LSTM can maintain high accuracy during long sequence tasks.