

Part1:

1. [[767. 5. 9. 11. 31. 63. 2. 62. 30. 20.]
[7. 667. 110. 17. 28. 23. 59. 13. 24. 52.]
[7. 59. 694. 26. 28. 20. 46. 35. 47. 38.]
[6. 38. 57. 756. 14. 57. 16. 17. 27. 12.]
[61. 49. 81. 20. 627. 19. 31. 34. 21. 57.]
[8. 28. 123. 17. 20. 724. 27. 9. 33. 11.]
[5. 22. 148. 8. 25. 23. 727. 20. 8. 14.]
[16. 28. 27. 12. 83. 16. 54. 623. 92. 49.]
[11. 35. 98. 42. 8. 31. 43. 6. 702. 24.]
[7. 50. 91. 4. 51. 33. 18. 29. 40. 677.]]

Test set: Average loss: 1.0088, Accuracy: 6964/10000 (70%)

Number of parameters: $784 \times 10 + 10 = 7850$

2. [[849. 2. 2. 4. 35. 30. 4. 43. 24. 7.]
[6. 815. 39. 3. 19. 9. 59. 4. 16. 30.]
[7. 12. 840. 48. 11. 15. 23. 9. 18. 17.]
[5. 7. 26. 918. 2. 13. 3. 7. 11. 8.]
[42. 33. 23. 7. 791. 8. 37. 16. 21. 22.]
[10. 9. 78. 9. 13. 823. 26. 3. 19. 10.]
[3. 14. 73. 8. 16. 6. 858. 11. 2. 9.]
[15. 19. 20. 4. 24. 8. 39. 808. 32. 31.]
[8. 30. 27. 52. 4. 13. 27. 6. 826. 7.]
[3. 18. 56. 5. 30. 6. 21. 10. 14. 837.]]

Test set: Average loss: 0.5315, Accuracy: 8365/10000 (84%)

Number of parameters: $784 \cdot 100 + 100 + 100 \cdot 10 + 10 = 79510$

3. $\begin{bmatrix} 938. & 5. & 2. & 0. & 23. & 11. & 0. & 12. & 3. & 6. \\ 2.889. & 5. & 0. & 19. & 3. & 62. & 2. & 4. & 14. \\ 10. & 3.883. & 30. & 10. & 12. & 26. & 8. & 9. & 9. \\ 0. & 1. & 18.961. & 0. & 8. & 5. & 3. & 1. & 3. \\ 16. & 2. & 2. & 13.918. & 4. & 17. & 7. & 18. & 3. \\ 1. & 2. & 33. & 9. & 8.913. & 27. & 2. & 2. & 3. \\ 3. & 3. & 17. & 1. & 8. & 2.963. & 1. & 0. & 2. \\ 2. & 2. & 4. & 0. & 12. & 2. & 9.935. & 11. & 23. \\ 3. & 19. & 3. & 3. & 5. & 7. & 13. & 2.942. & 3. \\ 6. & 4. & 10. & 1. & 12. & 2. & 13. & 5. & 8.939. \end{bmatrix}$

Test set: Average loss: 0.2621, Accuracy: 9281/10000 (93%)

Number of parameters:

$1 \cdot 16 \cdot 3 \cdot 3 + 16 + 16 \cdot 32 \cdot 3 \cdot 3 + 32 + 1568 \cdot 128 + 128 + 128 \cdot 10 + 10 = 206922$

4.

a. the relative accuracy of the three models:

NetLin: 70%

NetFull: 84%

NetConv: 93%

b. the number of independent parameters in each of the three models:

NetLin: 7850

NetFull: 79510

NetConv: 206922

c.

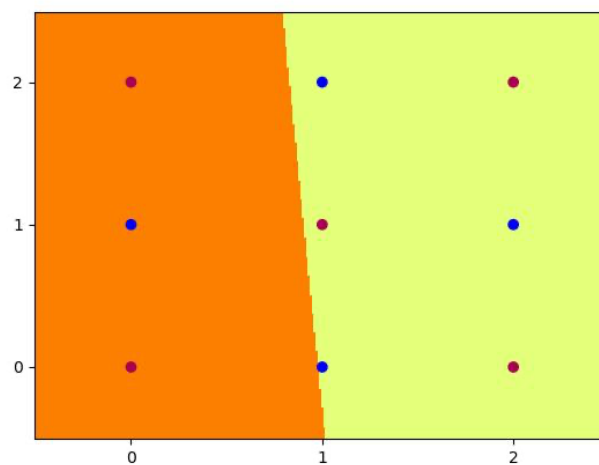
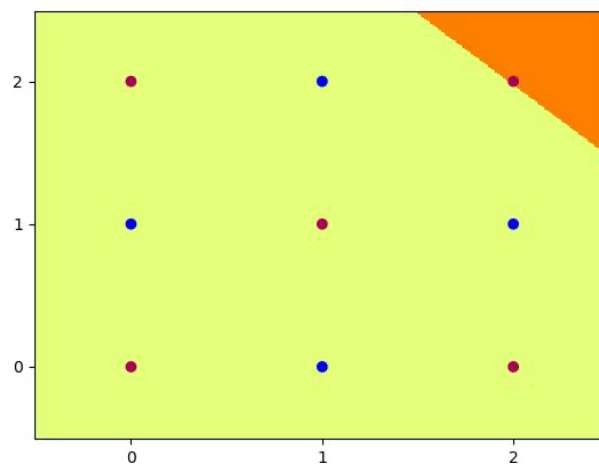
The confusion matrix as above. "su"(2) and "ha"(5) are easily confused because both have curved strokes, especially since the right half of "ha" closely resembles "su". "su"(2) and "tsu"(3) are easily confused because their stroke curves are similar, with the main body of "su" appearing very much like "tsu". "ki"(1) and "ma"(6) are easily confused due to their complex and extremely similar structures. Additionally, "na"(4) and "o"(0) both feature curved structures with a dot in the

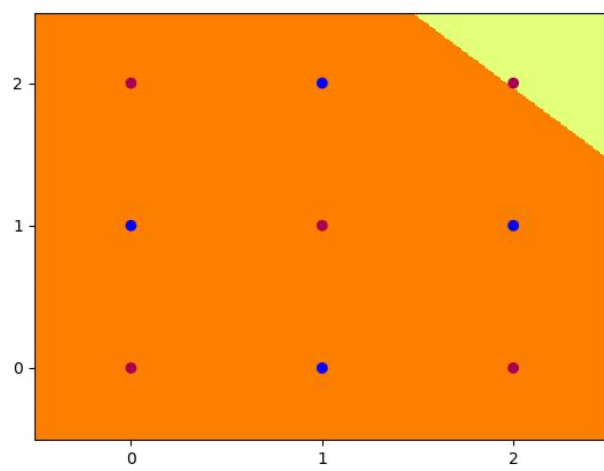
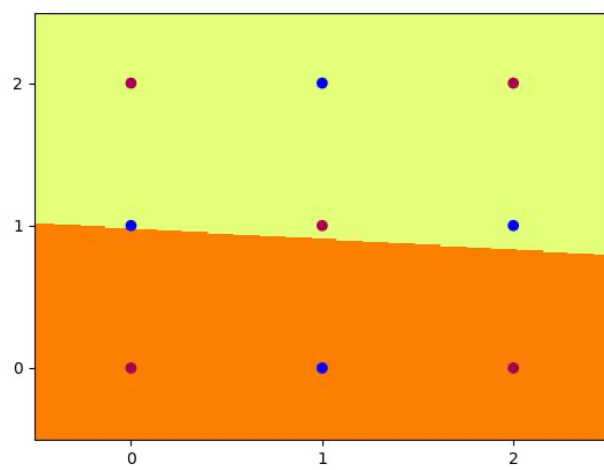
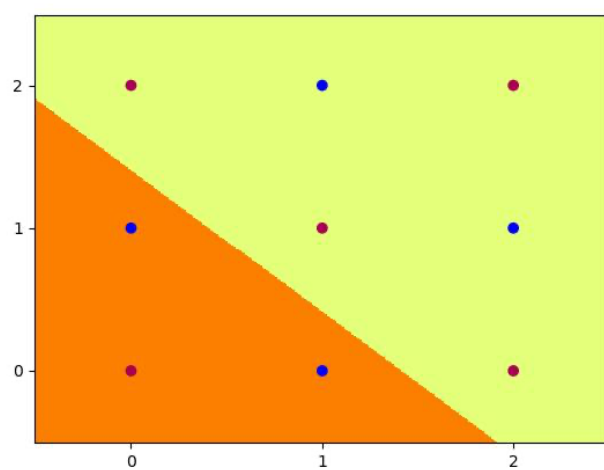
upper right corner.

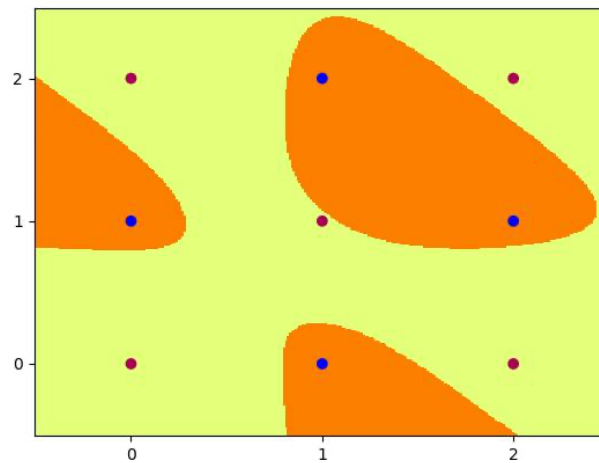
We can observe that in linear models, because the features of "su"(2) are distributed in the middle range, linear models have difficulty learning this distinction. However, convolutional neural networks learn this pattern much better, with error rates less than half that of linear models, because CNNs are superior at extracting features from complex characters and better at recognizing the spatial relationships between strokes.

Part2:

1.







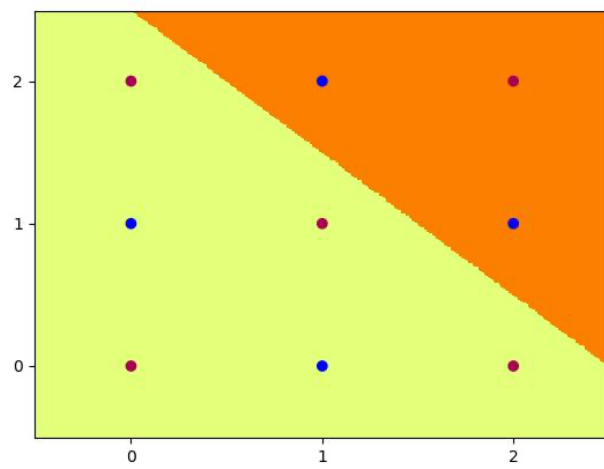
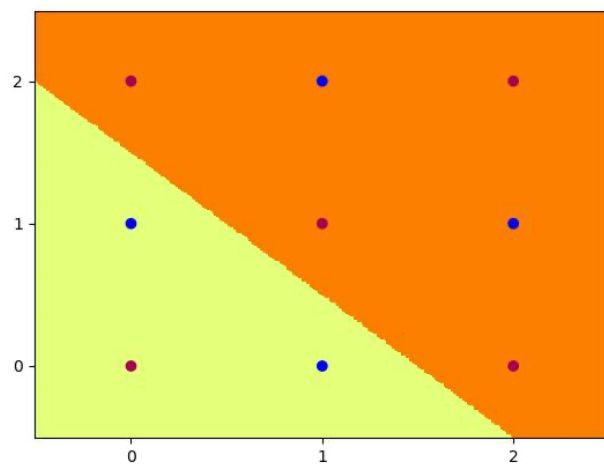
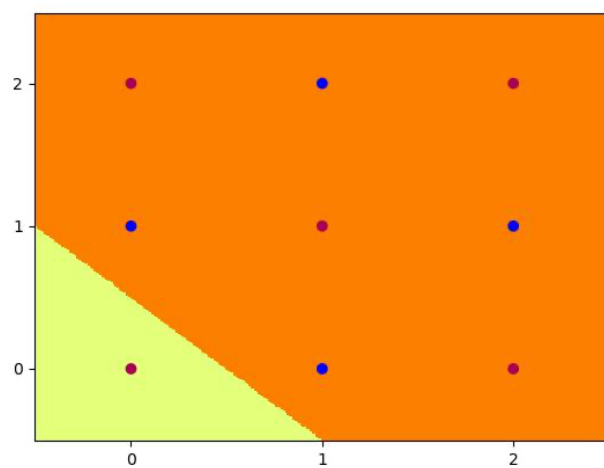
2.

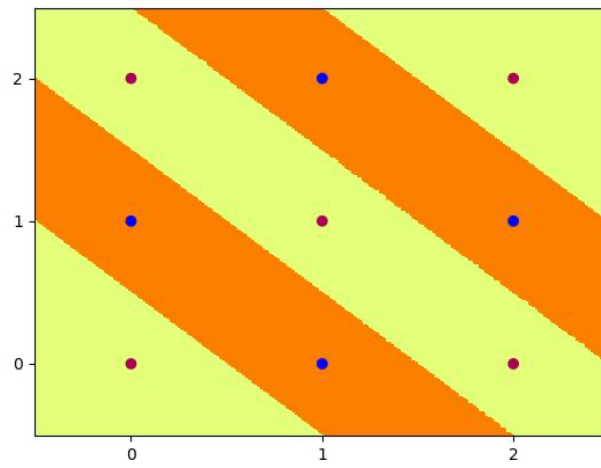
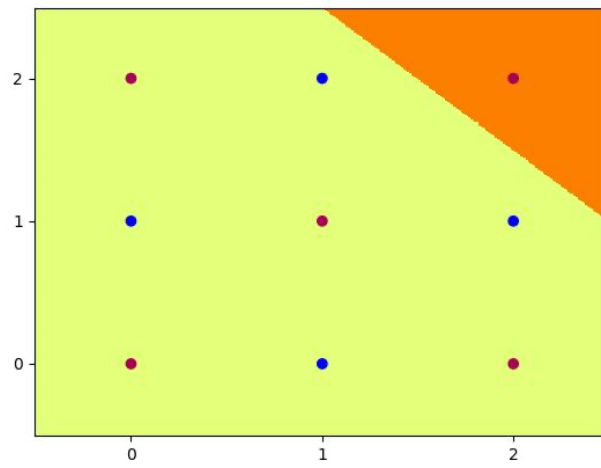
```
Initial Weights:
tensor([[1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.]])
tensor([-0.5000, -1.5000, -2.5000, -3.5000])
tensor([[ 1., -1.,  1., -1.]])
tensor([-0.5000])
Initial Accuracy: 100.0
```

H1: $X+Y=0.5$ H2: $X+Y=1.5$ H3: $X+Y=2.5$ H4: $X+Y=3.5$

X	Y	Hidden nodes	Output
0	0	0,0,0,0	0
0	1	1,0,0,0	1
0	2	1,1,0,0	0
1	0	1,0,0,0	1
1	1	1,1,0,0	0
1	2	1,1,1,0	1
2	0	1,1,0,0	0
2	1	1,1,1,0	1
2	2	1,1,1,1	0

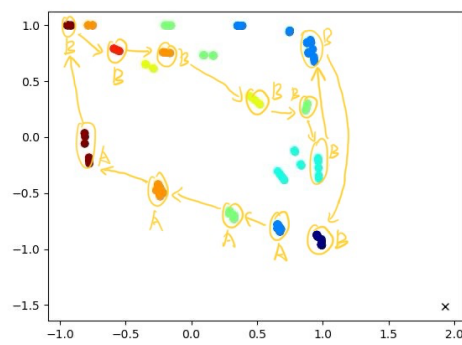
3.



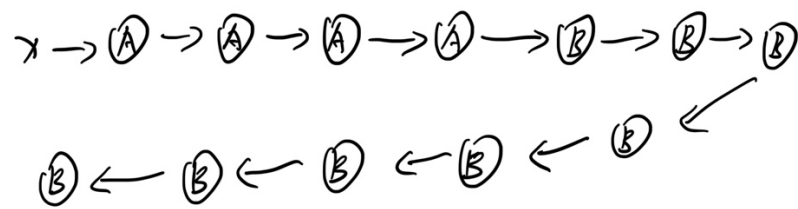


Part3:

1.

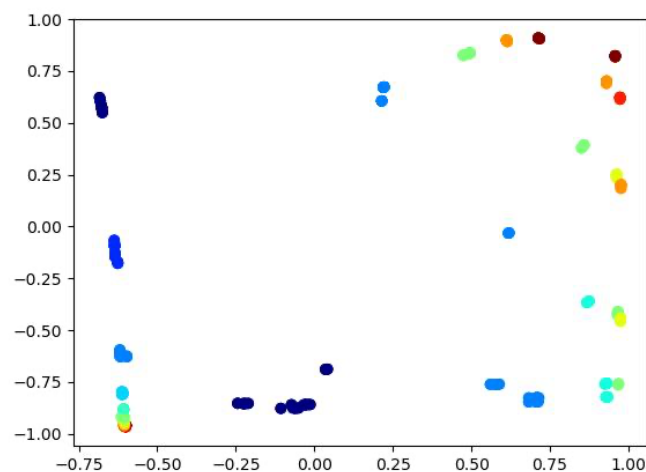


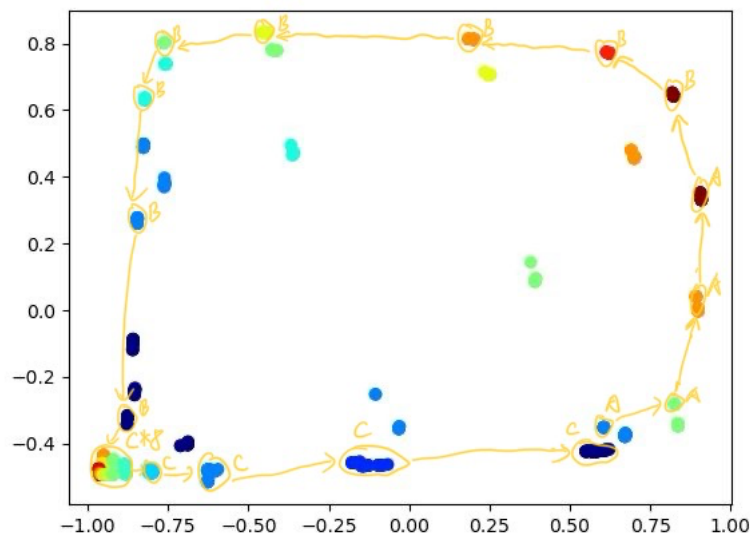
2.



3. When the network receives a character sequence beginning with A, its hidden state is progressively updated with each A input. As the number of A's increases, the hidden layer maintains a memory of the A count. For example, incrementing the hidden state by 1 for each A encountered. When the first B appears, the network can already determine the number of preceding A's through its current hidden state. Following the $a^n b^{2n}$ rule, the number of B's should be twice the number of A's. Therefore, when the network encounters the first B, it begins counting the B's. For each B it sees, it decrements the previously remembered $2N$ count. As long as the count in the hidden state hasn't reached zero, it continues predicting B. After all $2N$ B's have been predicted, the "counter" in its hidden state returns to zero, signaling that the next character should be A.

4.





The $a^n b^{2n} c^{3n}$ language is a quite complex counting task that requires the network to accurately output $2n$ B's and $3n$ C's after seeing n A's.

Traditional RNNs face serious difficulties when processing such long sequences. RNN's hidden state update mechanism is relatively simple, combining the previous hidden state with current input to calculate the new state at each step. This design easily leads to gradient vanishing or exploding problems when processing long sequences, preventing the network from effectively maintaining long-term memory. Especially when the network processes the C phase, it's difficult to accurately recall how many A's appeared earlier.

LSTM designs a forget gate to decide which information to forget from the memory cell, an input gate to control new information writing, and an output gate to manage how information in the memory cell affects the final output. It can extract the count information of A's from the memory cell, then output twice that number of B's and three times that number of C's. Hidden state handles current output and short-term information transfer, while cell state is specifically responsible for long-term information storage. Cell state is carefully protected through gating mechanisms, allowing it to stably preserve important information across multiple time steps.