Q1.

1.

[[766. 5. 9. 12. 31. 63. 2. 62. 31. 19.]

[6. 668. 109. 19. 27. 23. 56. 15. 26. 51.]

[8. 62. 695. 28. 25. 19. 46. 36. 45. 36.]

[5. 38. 59. 755. 15. 57. 15. 17. 29. 10.]

[59. 53. 77. 22. 623. 20. 33. 38. 20. 55.]

[8. 28. 126. 17. 20. 725. 26. 9. 32. 9.]

[5. 24. 146. 10. 25. 24. 723. 21. 9. 13.]

[17. 30. 26. 13. 80. 18. 54. 627. 88. 47.]

[12. 37. 93. 43. 6. 29. 45. 6. 707. 22.]

[8. 53. 88. 3. 51. 31. 20. 30. 39. 677.]]

Test set: Average loss: 1.0099, Accuracy: 6966/10000 (70%)

2.

[[849. 2. 1. 7. 36. 31. 4. 35. 27. 8.]

[5. 830. 26. 6. 20. 11. 53. 6. 17. 26.]

[8. 19. 843. 35. 14. 15. 22. 11. 20. 13.]

[5. 13. 32. 910. 2. 12. 6. 2. 7. 11.]

[39. 31. 19. 4. 810. 9. 29. 21. 21. 17.]

[8. 10. 75. 10. 10. 836. 26. 2. 16. 7.]

[3. 15. 49. 8. 15. 6. 892. 4. 2. 6.]

[15. 19. 19. 8. 31. 9. 34. 801. 31. 33.]

[9. 31. 29. 49. 3. 10. 28. 4. 832. 5.]

[5. 25. 47. 3. 34. 5. 21. 15. 13. 832.]]

Test set: Average loss: 0.5189, Accuracy: 8435/10000 (84%)

3.

[[946. 1. 2. 1. 18. 12. 2. 11. 3. 4.]

[4. 927. 7. 0. 5. 1. 37. 5. 4. 10.]

[11. 9. 871. 31. 7. 13. 32. 12. 5. 9.]

[1. 7. 25. 941. 5. 10. 3. 3. 3. 2.]

[27. 4. 7. 12. 907. 9. 17. 9. 6. 2.]

[3. 13. 29. 3. 5. 918. 16. 1. 6. 6.]

[3. 10. 13. 1. 2. 0. 963. 6. 1. 1.]

[15. 4. 4. 0. 3. 5. 10. 943. 4. 12.]

[5. 10. 4. 2. 12. 5. 4. 2. 955. 1.]

[4. 12. 6. 1. 4. 2. 1. 5. 8. 957.]]

Test set: Average loss: 0.3094, Accuracy: 9328/10000 (93%)

4.

We can see that the CNN model has the highest accuracy score followed by the fully connected 2-layer network followed by the 1-layer network. A simple model like 1-layer linear function network, the model underfits the data because a linear function can only separate the linear features of the data which is clearly not enough for the image classification task. Similarly, CNN can be specialized at the specific feature of the image which would perform a better job than the 2-layer fully connected network.

For the 1-layer network, there are 3 majority mistakes, misclassify 1 to 2, misclassify 5 to 2, and misclassify 6 to 2. But in general, all numbers can be misclassified to others. For the 2-layer network, the main mistake is misclassifying 5 to 2. For the CNN, the main mistakes are misclassifying 5 to 2 and misclassifying 2 to 3.

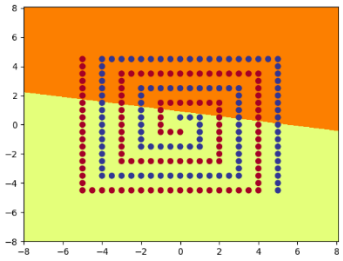
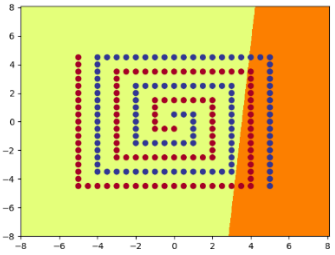
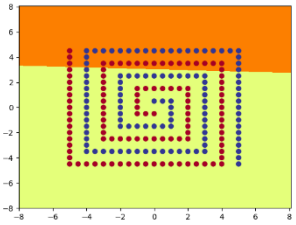
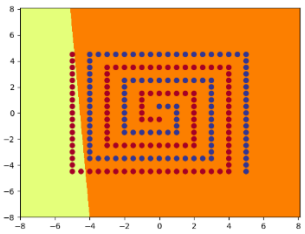
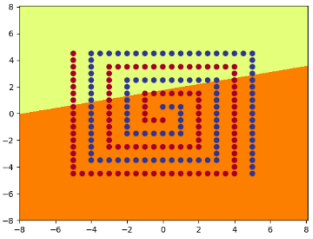
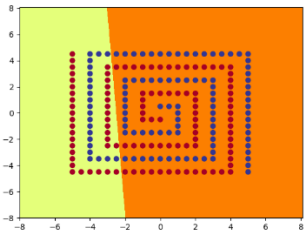
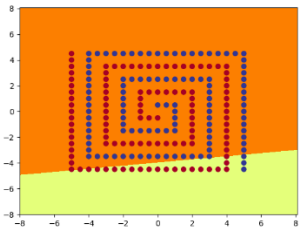
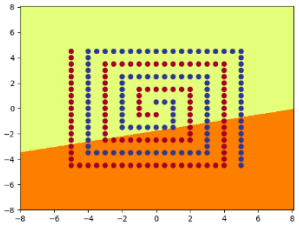
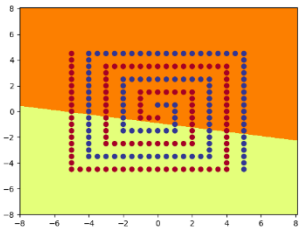
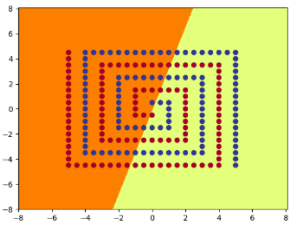
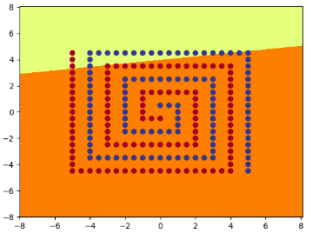
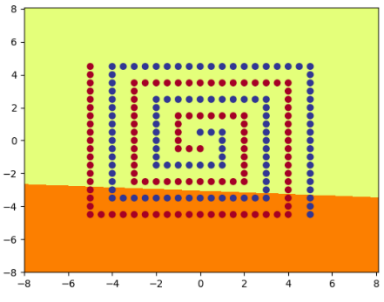
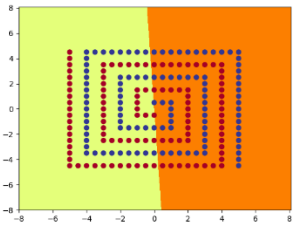
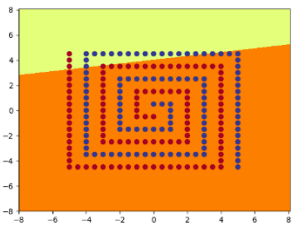
There are two reasons for misclassification. One is because models like 1-layer network is not good enough to generalize the data. Another main reason is because the noise in the data. The handwritten digits are very different from each other within the same group (e.g. digit 5 looks very different from each other) and might look similar to another digit of a different group (e.g. digit 2 looks very similar to a lot of other digits of different group).

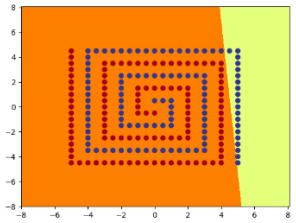
Q2.

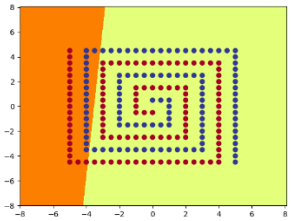
3.

The minimum hidden nodes required to train the network successfully is 17.

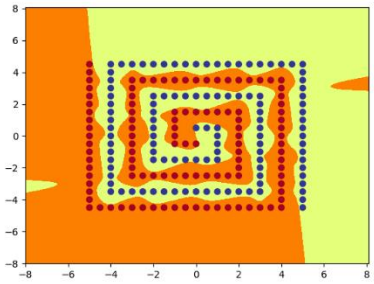
The output of graph_hidden() from node 0 to 16.





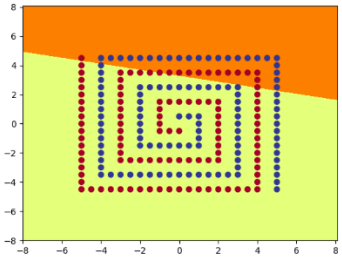
The output of graph_out() is.

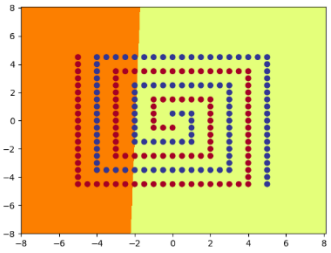


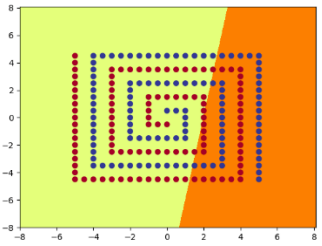
4.

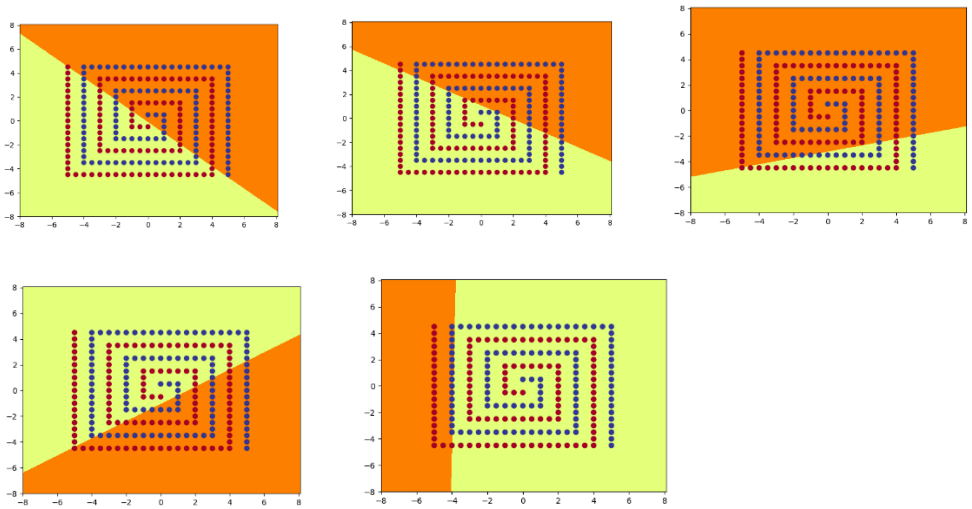
The minimum node per layer that can make the training successful is 8, hence in total 16 hidden nodes.

The output of get_hidden() for layer 1 from node 0 to node 7.

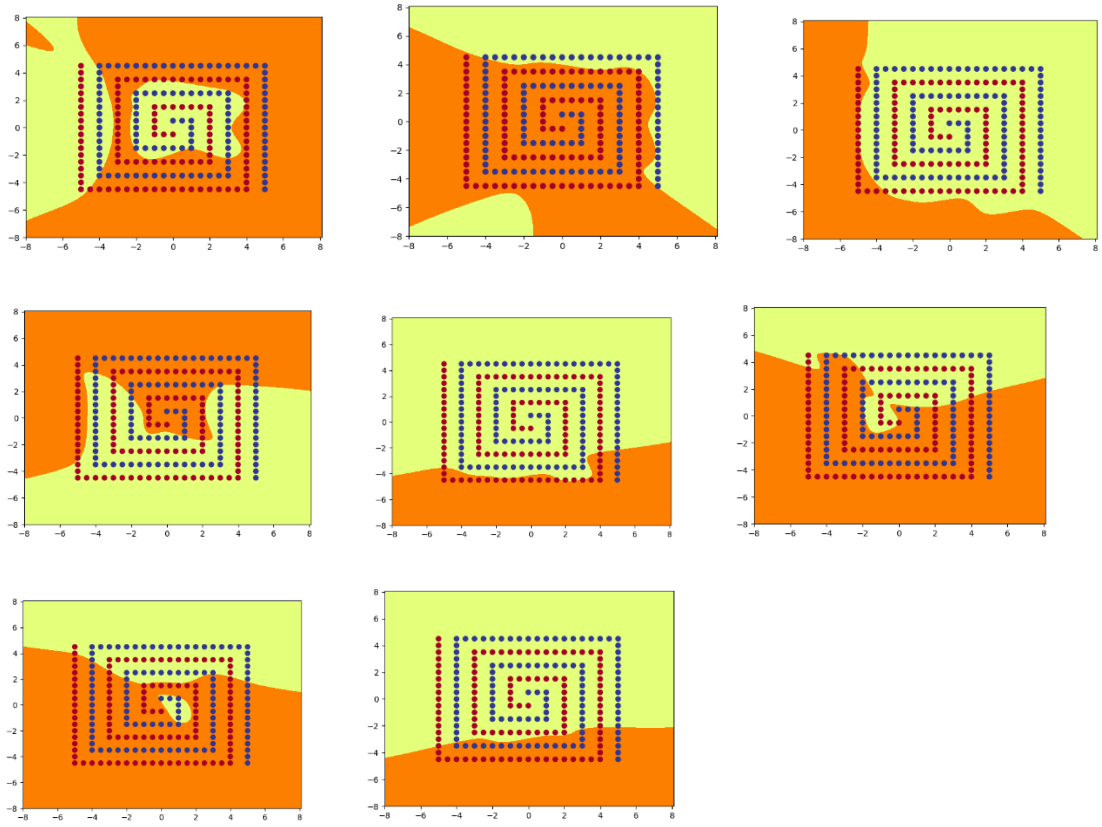




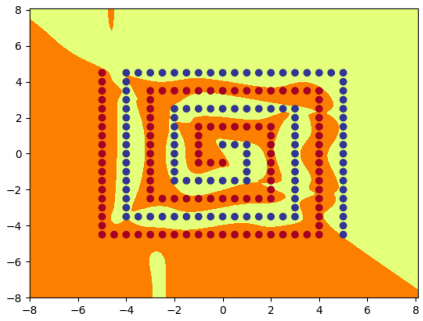




The output of get_hidden() for layer 2 from node 0 to 7.



The output of graph_out().

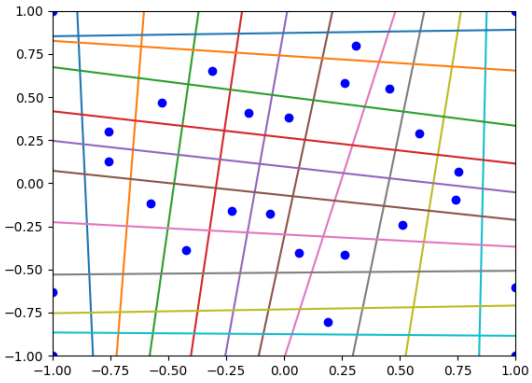


5.

1) We can observe that for the first hidden layer of the neural network, it can only learn linear separable functions, while the later layers can learn more complicated functions or the “convex” feature. For this classification task, whenever the layer is equal to 1 or 2, the target function can be learned by the output layer.

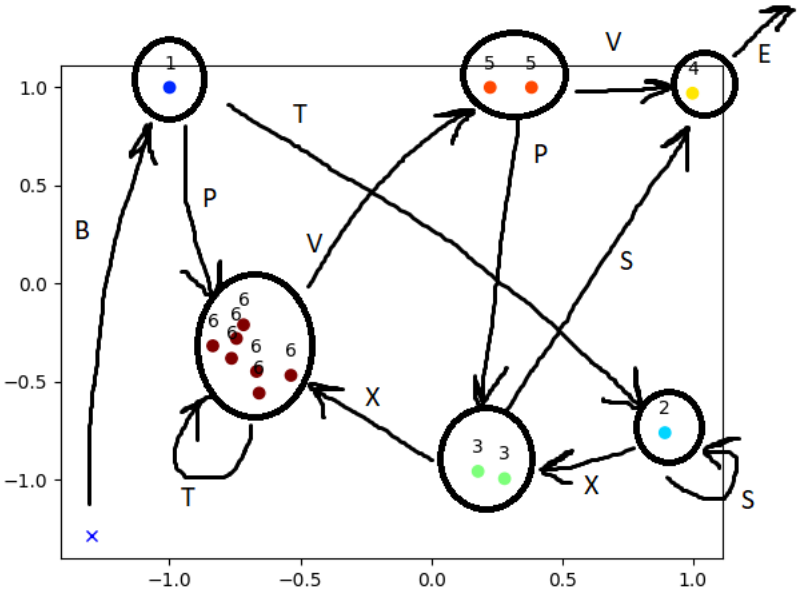
2) There is no significant difference between overall function in terms of learning the rectangular spiral problem, all successfully separate the positive examples and negative examples.

Q3.

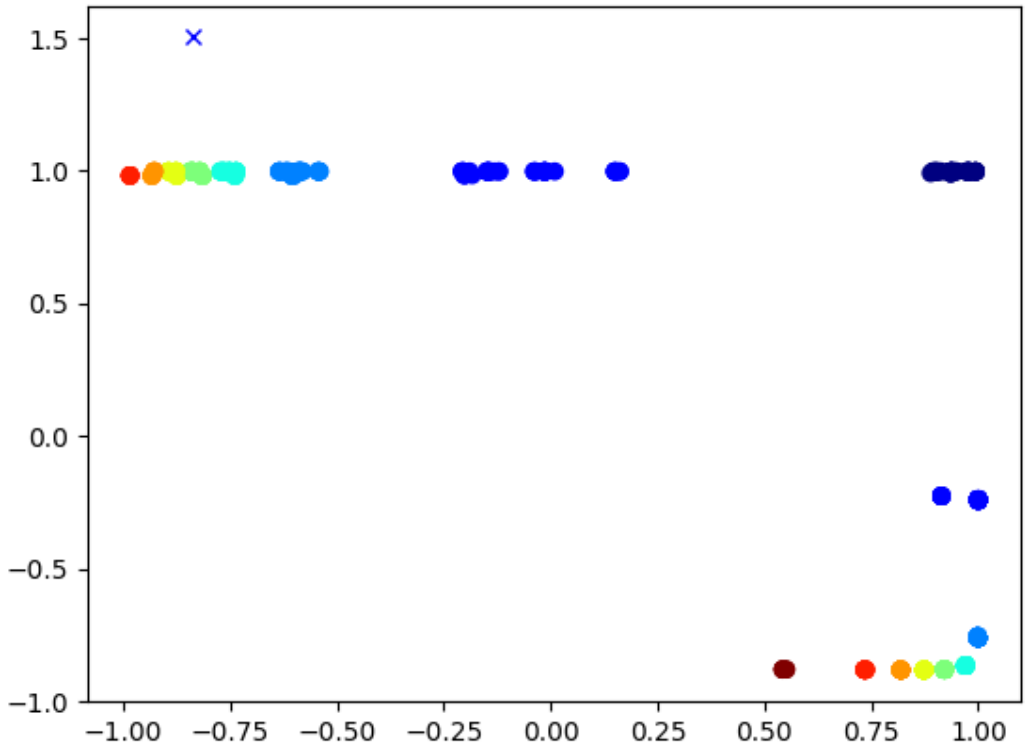


Q4.

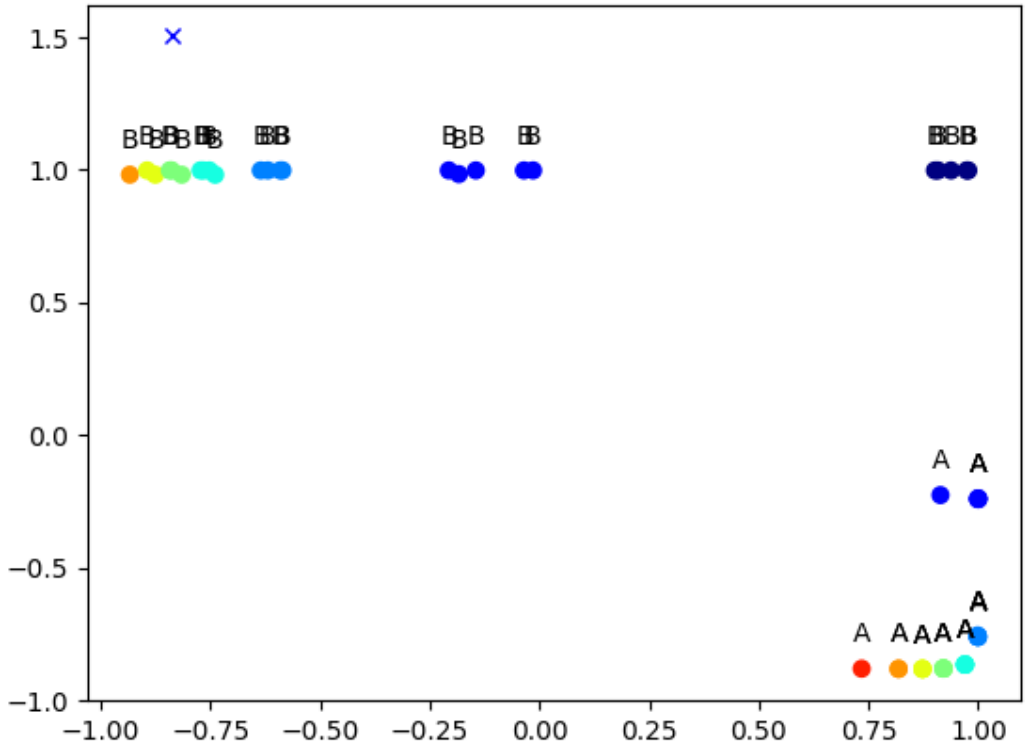
1.


1
5
5
4
V
E
T
P
B
P
V
S
6
6
6
6
6
6
6
X
2
3
3
X
S
T
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

2.

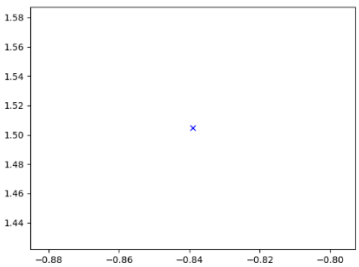

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

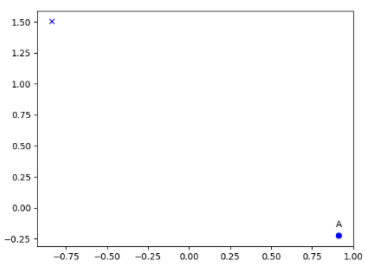
3.

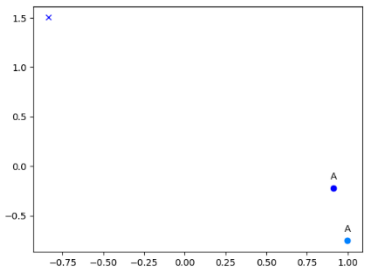


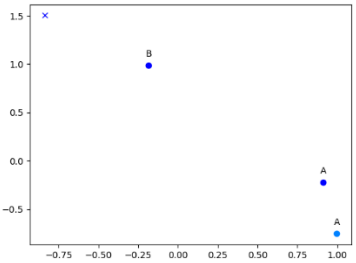
If we label what each of these points represents, we could see from the following figure that in the hidden unit space, the top region corresponds to B, the bottom region corresponds to A. The SRN must learn both the boundary and the transition of the pushdown automaton.

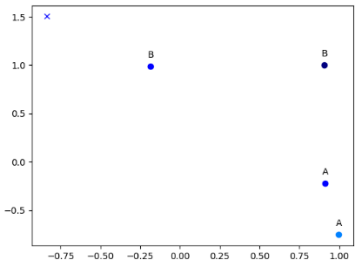
Since for each AnBn, the first B is unpredictable, the following B and the A followed by the n Bs are predictable. When we meet the first A (denote this point as X or repeller #1), we count up the number of As in the direction away from X. When we meet the first B, we are repelled to a point the B region (let this point be Y or repeller #2) that represents the expected Bs afterward is n – 1 (n is the number of As seen), we then predict Bs, count down the number of expected Bs and move in the direction away from Y until the count of A drops to 0 (at this point we reach one of the dark purple points). When this condition is met, we are repelled once more to the A region in a point corresponds to the first A (one of the deep blue points). The above procedure is repeated until the entire string is predicted.

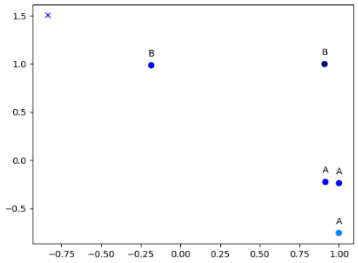






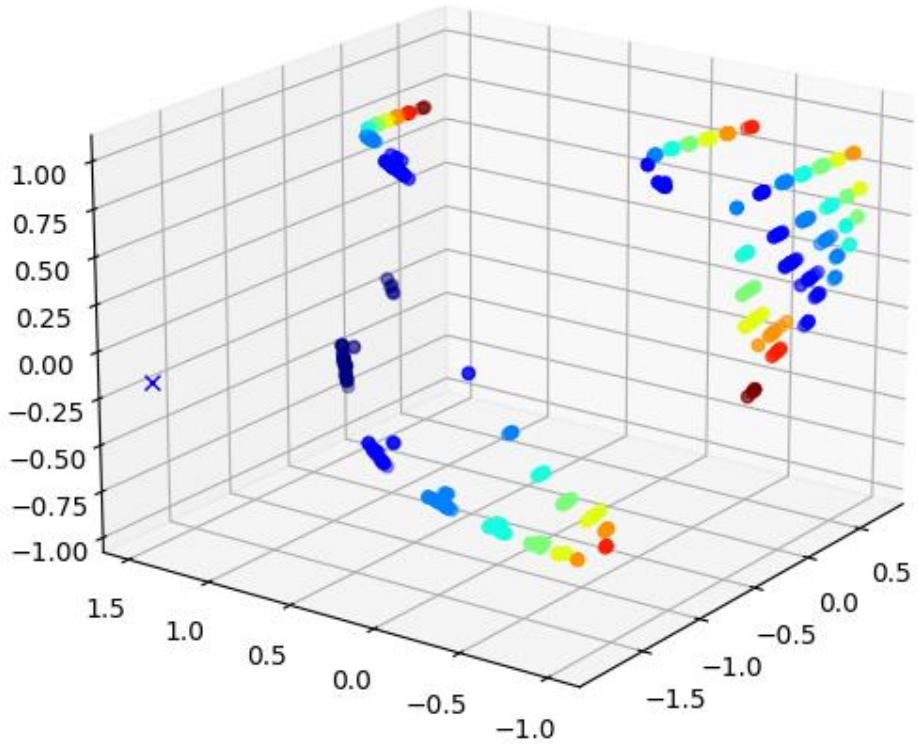






For example, the first 5 characters in the string are AABBA. According to the above 6 figures. The starting point is the cross, when we see the first A, we arrive at the deep blue point, then when we see the second A, we move away from the deep blue point to a light blue point. When we meet the first B, we jump to the B region, and count down the number of B and move away from this first B. When the count is 0, we have arrived at the dark blue purple point, we know the next character must be A, we would jump to a point in the A region that corresponds to the count of A is 1 and continue the process.

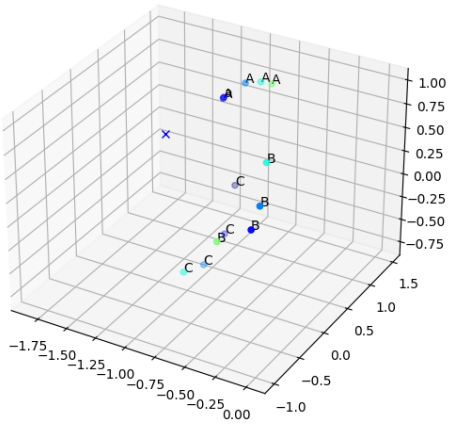
4.



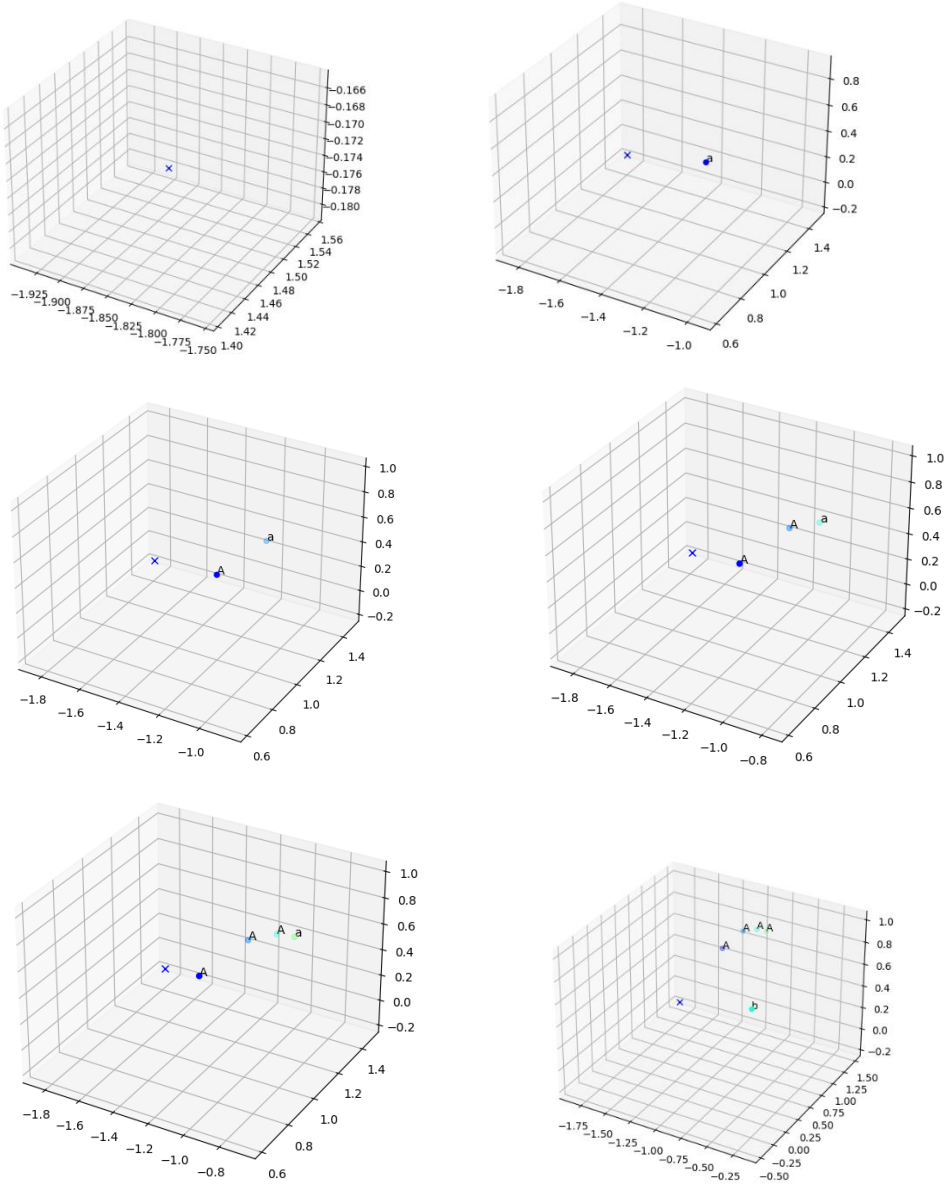
5.

For the AnBnCn prediction task, the first B is unpredictable, but the following B, following C and the first A after this AnBnCn are all predictable. Similar to question 3, we label the points in the hidden unit space and see if it is A or B or C (**this is a figure produced by different input to question 4**). The SRN must learn both the boundary of points and the transition of the linear bounded automaton. For this case, the hidden unit dynamics is as if there are 3 repellers.

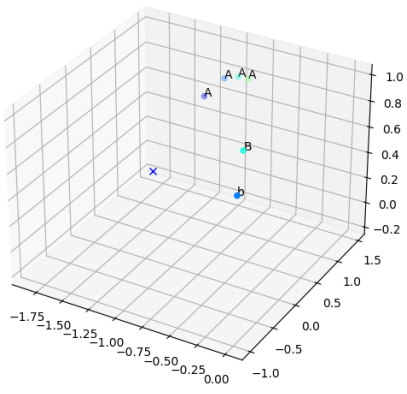
We only take the first group (A4B4C4A). We use lower case letter to denote the most recent character in the sequence.

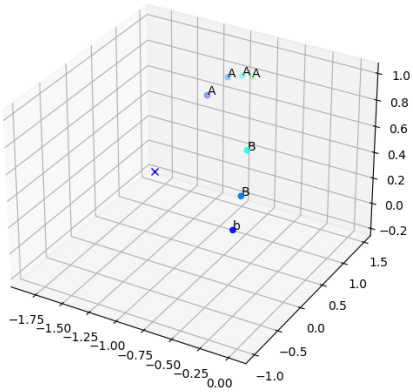


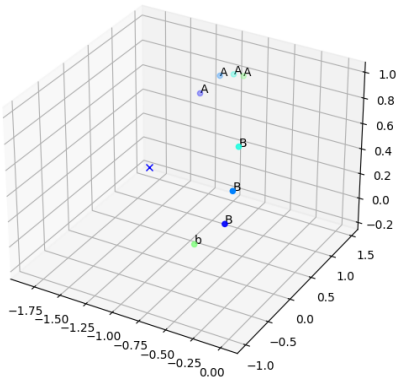
When the network see the first A, that is used as a fixed point (repeller #1). For each of the following As, the network would increment the count of A and move in the direction away from the fixed point towards the B region. Then, when the network see the first B, suppose at this point the network has seen n As, it would jumps to the point B in the B region corresponds to a count of n-1.

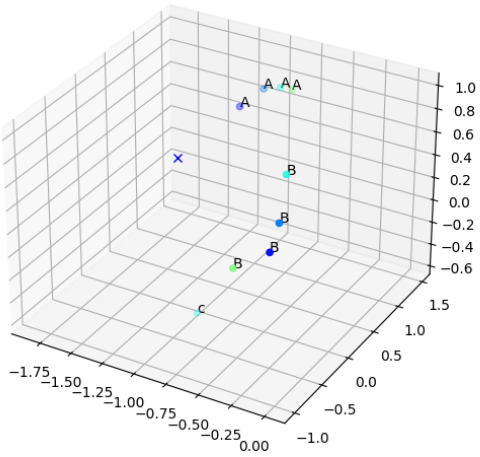


After that, since all the following Bs are predictable, it would decrement the count of expected B and move in the direction away the point corrsponds to the first B (repeller #2) towards the C region until the count of B reaches 1. After that it would predict the last B and increment the count of the expected number of following C to be n (this n must be learned by the network), and jump to a point in the C (repeller #3) region that corresponds to the expected number of C follows to be n-1.

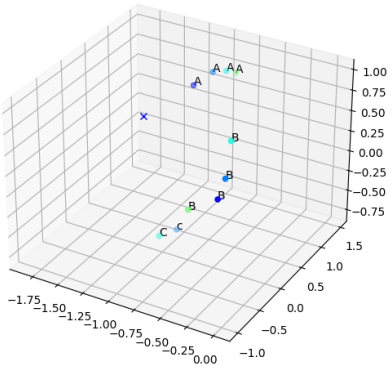


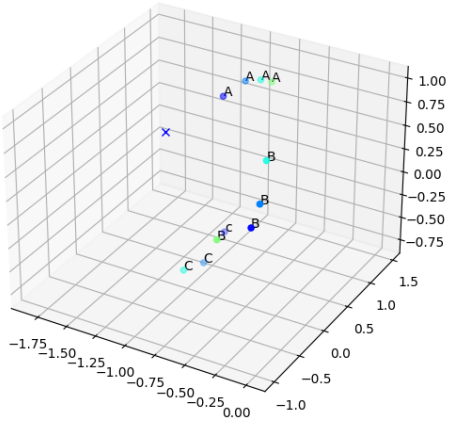


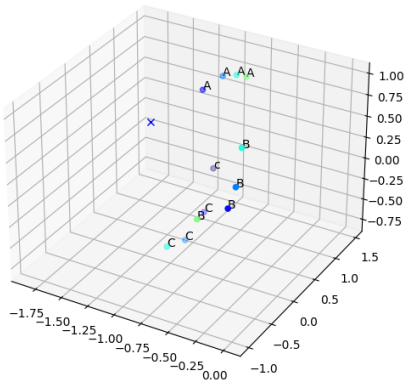


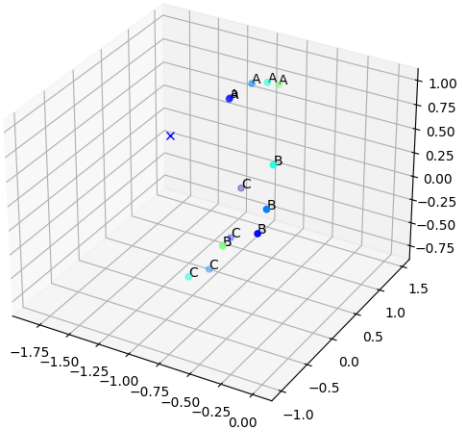


After that, the network would decrement count of expected number of C and moves away from the direction corrsponds to the first C towards the A region until the count reaches 0 (this corresponds to the last C). When the expected number of C reaches 0, it would jump to a point in the A region corresponds the number of A seen it 1 and predict A. This procedure is repeated for the entire string.

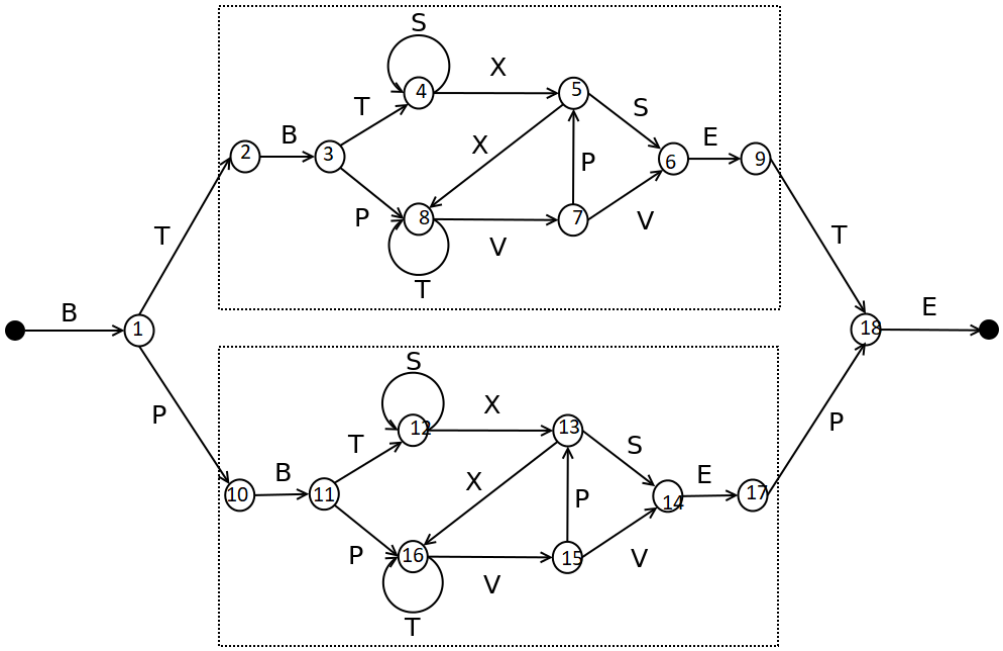








6.



The above image is referenced from [1]. The id of the nodes are given based on the code.

The minimum number of hidden nodes to make sure the training is successful is around 5-8 on my computer.

Since there are 5-8 hidden units and it is impossible to directly visualize the 5-8 dimensional hidden unit space, we attempt to compress it down to 3 dimensional space by applying PCA to the 5 to 8-d tensor. If we apply PCA, and compress the hidden unit space into a 3d space, the variance retained is approximately 70-80% which can maintain most of the essential information such as the distance between points.

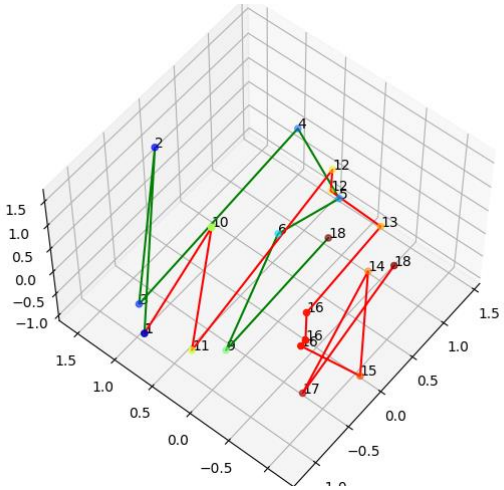


Figure 1: Hidden unit space 8d to 3d

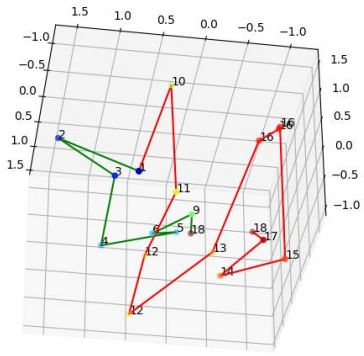


Figure 2: Hidden unit space 8d to 3d

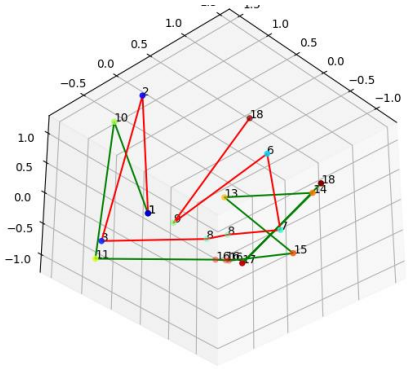
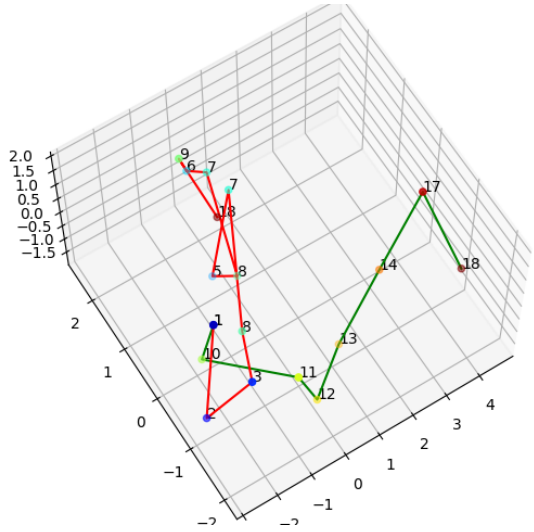


Figure 3: Hidden unit space 5d -> 3d

These above pictures are the hidden unit plots on different examples, all shows some similar features. We can see that for the embedded reber grammar case, the different nodes are also grouped into clusters. The entry node has id 1, the top part contains nodes 2-9, and the bottom parts contains nodes 10-17, the exit node is 18. For the top part, I plotted the edges between the

nodes in green, and for the bottom part, I plotted the edges between the nodes in red. The network must learn the clusters and the finite state machine.

We can see that the two parts form two relatively disjoint clusters. When we reach node 1, there might be 2 options, either enter the top part by moving to point 2 or enter the bottom part by moving to point 10. These points cannot be predicted accurately, the network would do the prediction probabilistically. After that, whatever point network enters, the situation is similar except the exit node. For all the nodes except the exit node, the network would do the prediction probabilistically, and transit to the corresponding cluster (see the lines between consecutive moves in the hidden unit space). The only thing LSTM can learn but SRN cannot is when we arrive at the exit node of the top half (i.e. node 9) or the exit node of the bottom half (i.e. 17) in the hidden unit space, the network is capable of predicting the symbol on the edge between node 9 and 18, and the symbol between node 17 and 18, this is achieved by memorizing whether we are on the red trajectory or the green trajectory (i.e. whether the node we reach is 2 or 10 after the starting node, or whether we are at node 17 or node 9 before node 18). This can be done with LSTM, because with simple SRN, the gradient might vanish during backpropagation, this long range dependency cannot be learned. The advantage of the LSTM is its context unit, it can be very specialized at a certain information. If we also plot the context unit activation (8d down to 3d by PCA), we can realize the context unit dynamics is like two clearly separated regions, one for the upper part of the state machine, one for the lower part of the state machine. Especially for the last several nodes, the distance between the two parts of the graph is very large, hence LSTM can easily identify the transition between 9 and 18 and the transition between 17 and 18.



[1]:-
https://github.com/DylanAuty/rebergen/blob/master/ReadmeGraphics/embeddedReberGrammar.png