COMP9444 Neural Networks and Deep Learning

Term 2, 2020

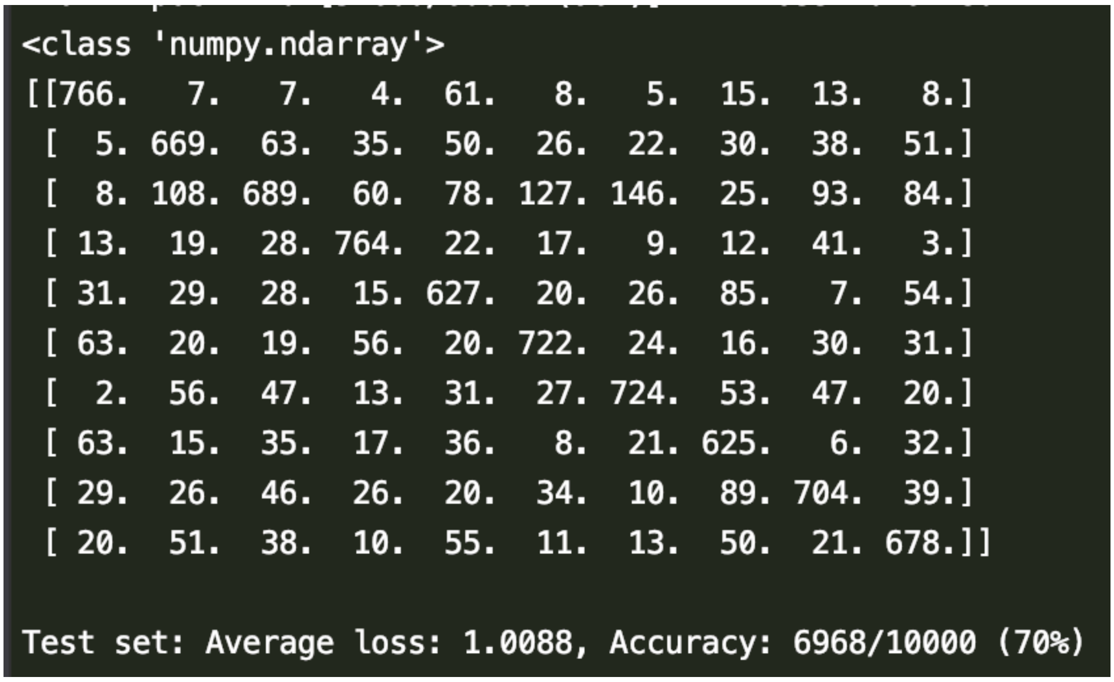
Project 1 - Japanese Characters and Intertwined Spirals

Yupeng Jiang

Z5237808

Part 1: Japanese Character Recognition

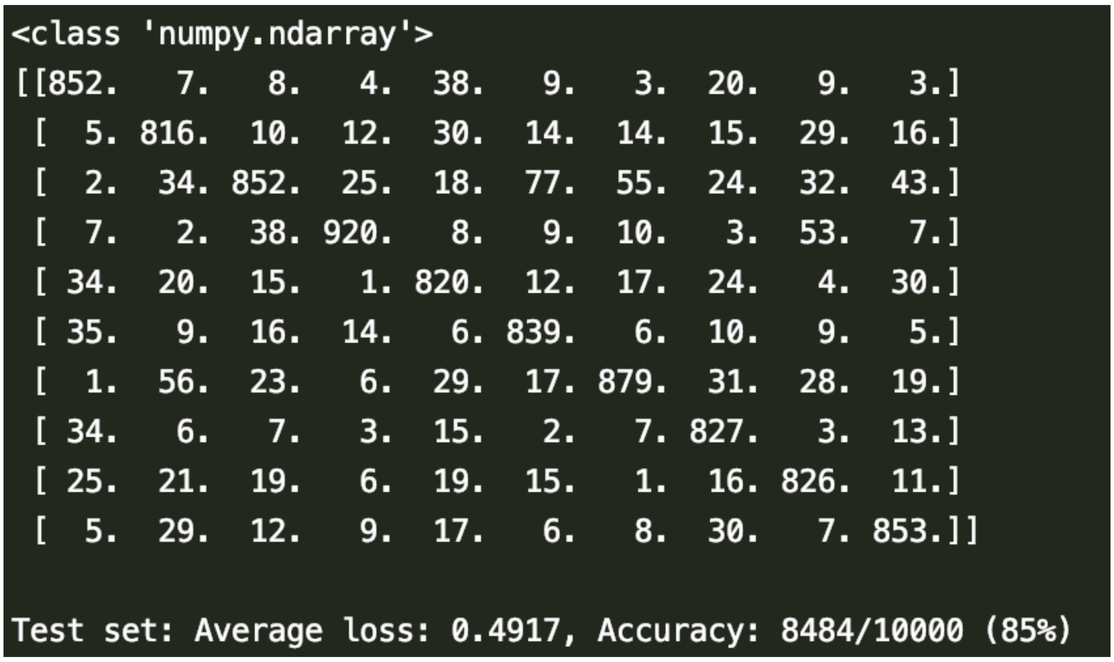
1. Following is my final accuracy and confusion matrix for model NetLin:



```
<class 'numpy.ndarray'>
[[766.   7.   7.   4.  61.   8.   5.  15.  13.   8.]
 [  5. 669.  63.  35.  50.  26.  22.  30.  38.  51.]
 [  8. 108. 689.  60.  78. 127. 146.  25.  93.  84.]
 [ 13.  19.  28. 764.  22.  17.   9.  12.  41.   3.]
 [ 31.  29.  28.  15. 627.  20.  26.  85.   7.  54.]
 [ 63.  20.  19.  56.  20. 722.  24.  16.  30.  31.]
 [  2.  56.  47.  13.  31.  27. 724.  53.  47.  20.]
 [ 63.  15.  35.  17.  36.   8.  21. 625.   6.  32.]
 [ 29.  26.  46.  26.  20.  34.  10.  89. 704.  39.]
 [ 20.  51.  38.  10.  55.  11.  13.  50.  21. 678.]]

Test set: Average loss: 1.0088, Accuracy: 6968/10000 (70%)
```

2. Following is my best final accuracy and confusion matrix for model NetFull with 300 hidden nodes:



```
<class 'numpy.ndarray'>
[[852.   7.   8.   4.  38.   9.   3.  20.   9.   3.]
 [  5. 816.  10.  12.  30.  14.  14.  15.  29.  16.]
 [  2.  34. 852.  25.  18.  77.  55.  24.  32.  43.]
 [  7.   2.  38. 920.   8.   9.  10.   3.  53.   7.]
 [ 34.  20.  15.   1. 820.  12.  17.  24.   4.  30.]
 [ 35.   9.  16.  14.   6. 839.   6.  10.   9.   5.]
 [  1.  56.  23.   6.  29.  17. 879.  31.  28.  19.]
 [ 34.   6.   7.   3.  15.   2.   7. 827.   3.  13.]
 [ 25.  21.  19.   6.  19.  15.   1.  16. 826.  11.]
 [  5.  29.  12.   9.  17.   6.   8.  30.   7. 853.]]

Test set: Average loss: 0.4917, Accuracy: 8484/10000 (85%)
```

For other number of hidden nodes:

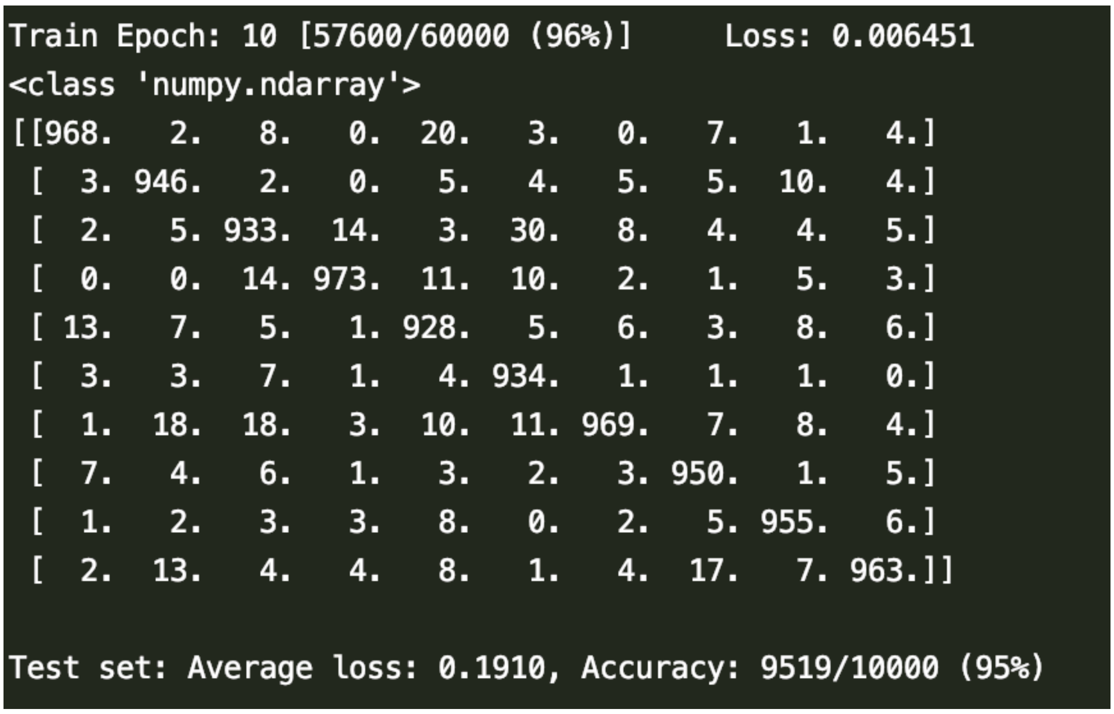
100 nodes, the final accuracy is 84%, average loss is 0.5331;

200 nodes, the final accuracy is 85%, average loss is 0.5034;

700 nodes, the final accuracy is 85%, average loss is 0.5042;

1000 nodes, the final accuracy is 84%, average loss is 0.5032;

3. Following is my final accuracy and confusion matrix for model NetConv with convolution kernel size 3, using max pooling with kernel size 2, 0.02 learning rate:



```
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.006451
<class 'numpy.ndarray'>
[[968.   2.   8.   0.  20.   3.   0.   7.   1.   4.]
 [  3. 946.   2.   0.   5.   4.   5.   5.  10.   4.]
 [  2.   5. 933.  14.   3.  30.   8.   4.   4.   5.]
 [  0.   0.  14. 973.  11.  10.   2.   1.   5.   3.]
 [ 13.   7.   5.   1. 928.   5.   6.   3.   8.   6.]
 [  3.   3.   7.   1.   4. 934.   1.   1.   1.   0.]
 [  1.  18.  18.   3.  10.  11. 969.   7.   8.   4.]
 [  7.   4.   6.   1.   3.   2.   3. 950.   1.   5.]
 [  1.   2.   3.   3.   8.   0.   2.   5. 955.   6.]
 [  2.  13.   4.   4.   8.   1.   4.  17.   7. 963.]]

Test set: Average loss: 0.1910, Accuracy: 9519/10000 (95%)
```

4. Compare the three models, Conv get the highest accuracy 95%. For fully connection model, two layer gets the better performance than one layer, which means accuracy will increase with the growth of the layer. However, comparing the convolution model (CNN) and fully connection model, CNN gets the better performance than linear model on image recognition task. As the linear model needs much more weight to be calculated than CNN on image processing, which is inefficient. Moreover, the number of layers of linear model will also limit the performance.

   As for confusion matrix:

   For NetLin model, according to the result above, character “su” is the most likely to be mistaken for other characters. For example, 146 “su” are recognized as “ma”, 127 “su” are recognized as “ha”.

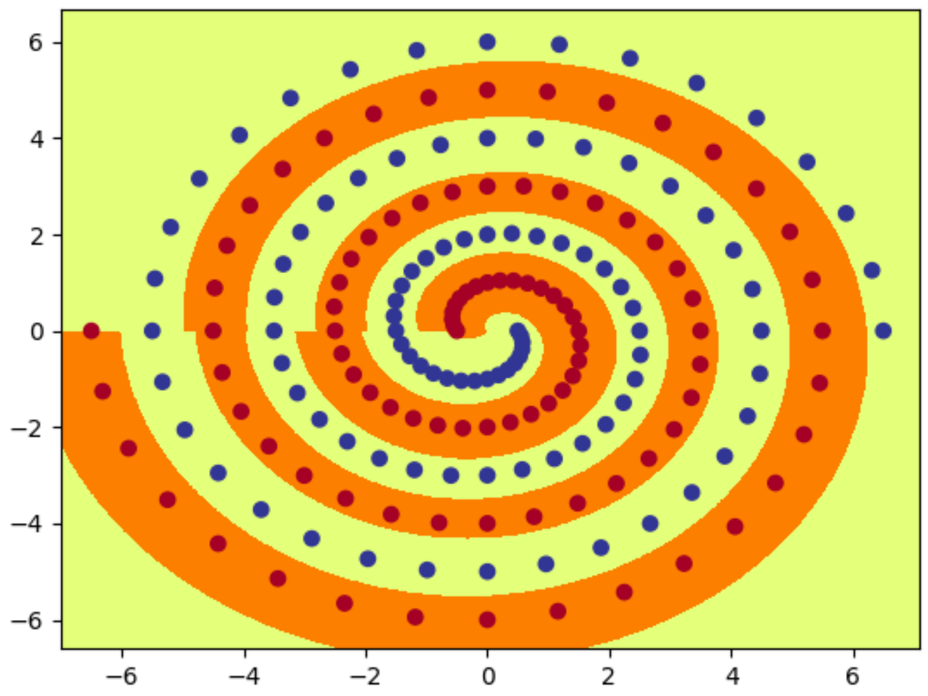
For Netful model, the character “su” is also the most likely to be mistaken, which is similar to the NetLin, but the number of character to be mistaken is smaller than NetLIn.

For Conv model, most of character are classified correctly. But there is still 30 “su” are recognized as “ha”.

I also tried other meta parameters in Conv model, such as learning rate, when change the lr from 0.01 to 0.02, the accuracy was increased from 94% to 95%. As for other parameters, I tried different size of kernel of convolution layer and different number of channels, but I got the similar or bad performance.

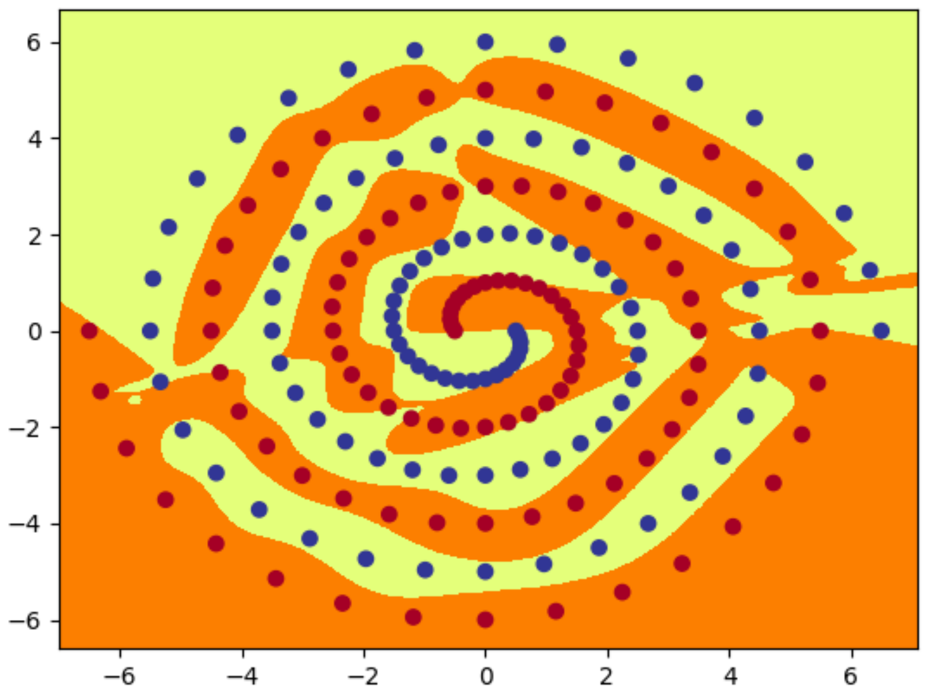
Part 2: Twin Spirals Task

2. Following is the result of PolarNet:



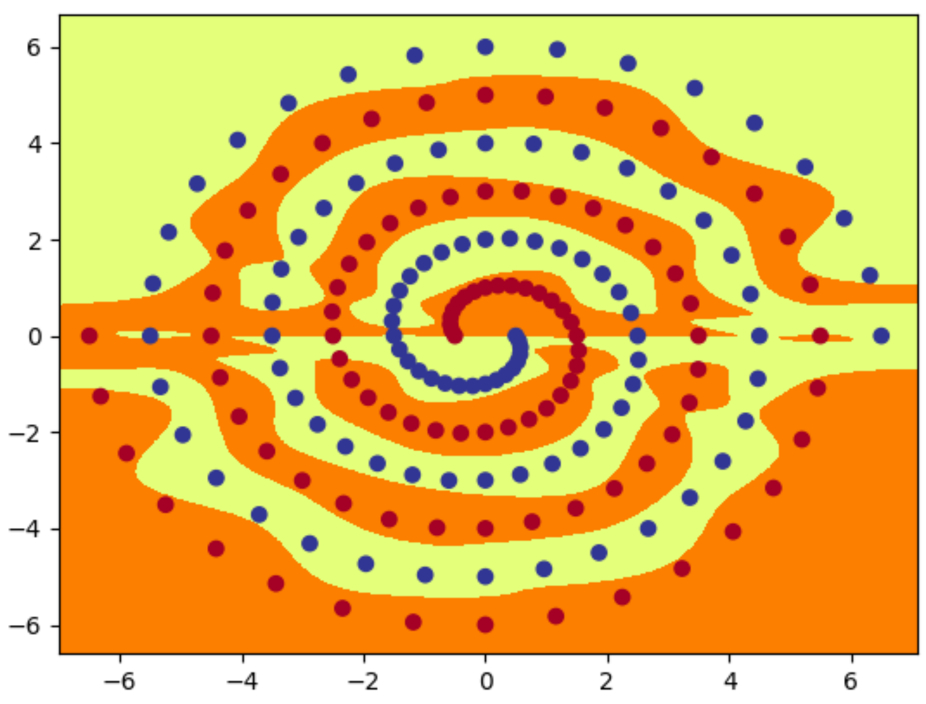
The minimum number of hidden nodes is 6.

4. Following is the result of RawNet:



This picture is the result of RawNet with initial weight 0.17 and 10 hidden numbers.

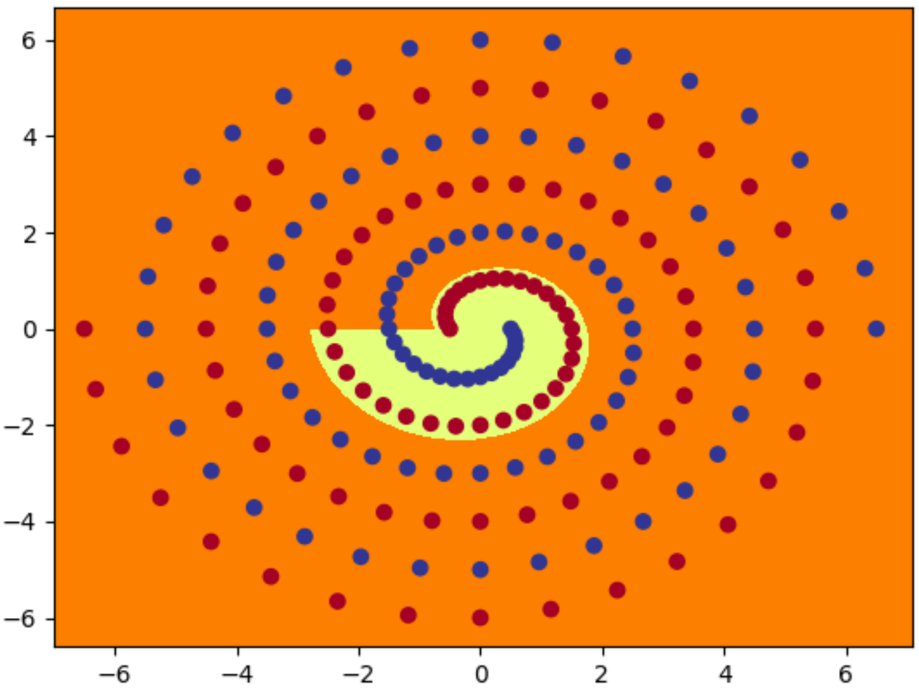
6. Following is the result of ShortNet



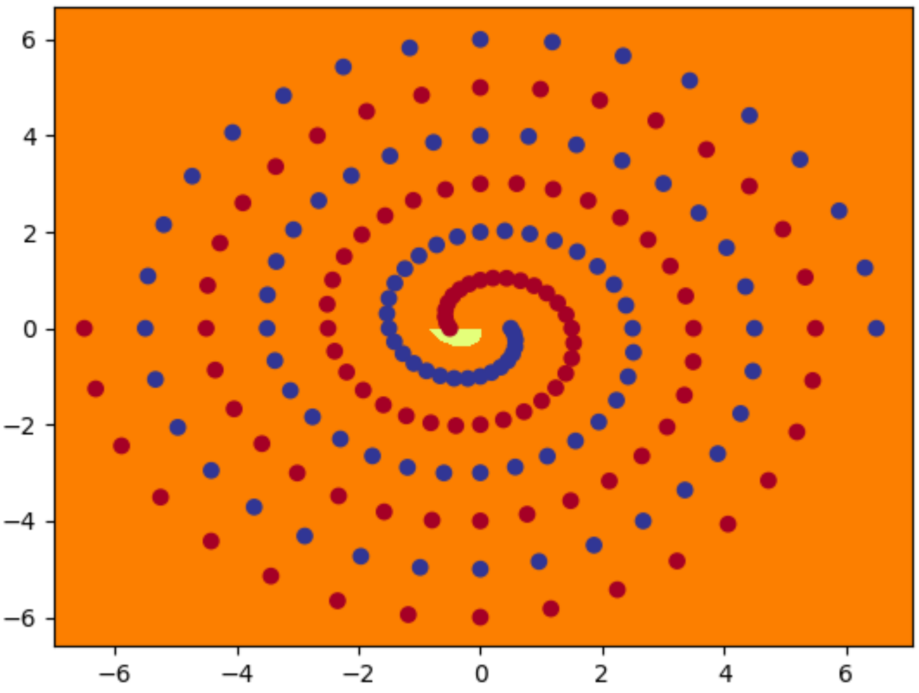
This picture is the result of ShortNet with initial weight 0.18 and 8 hidden numbers.

7. PolarNet hidden nodes:

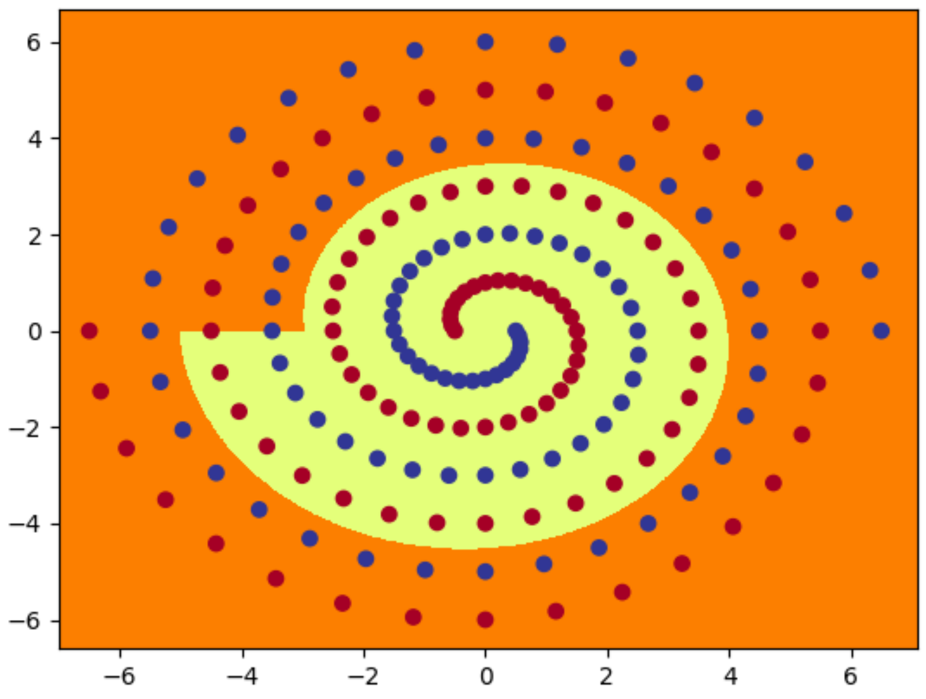
From node 0 to node 5 in layer 1:



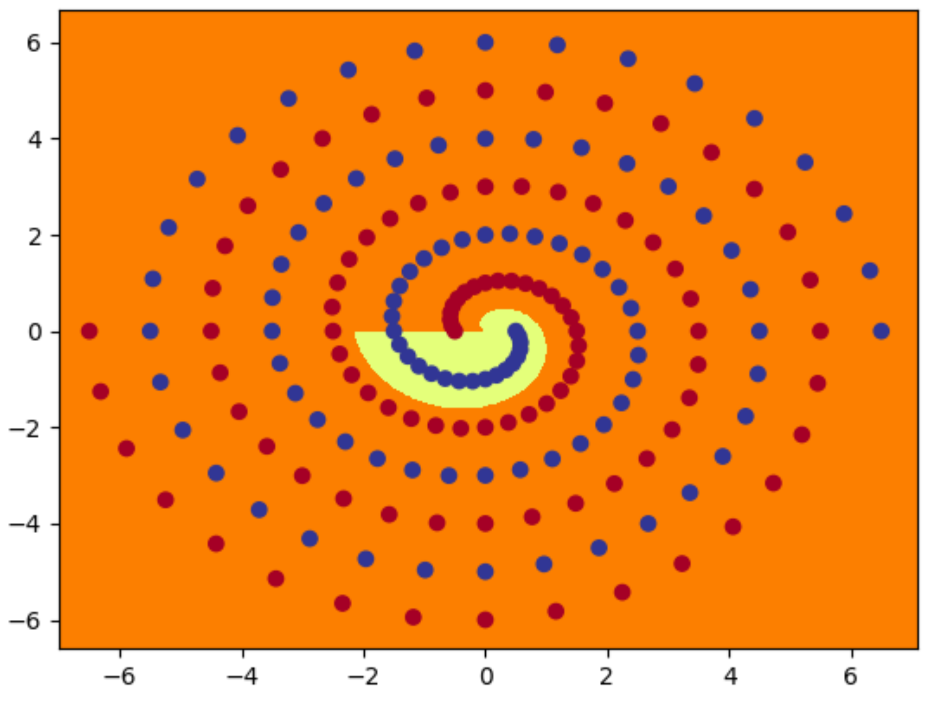
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



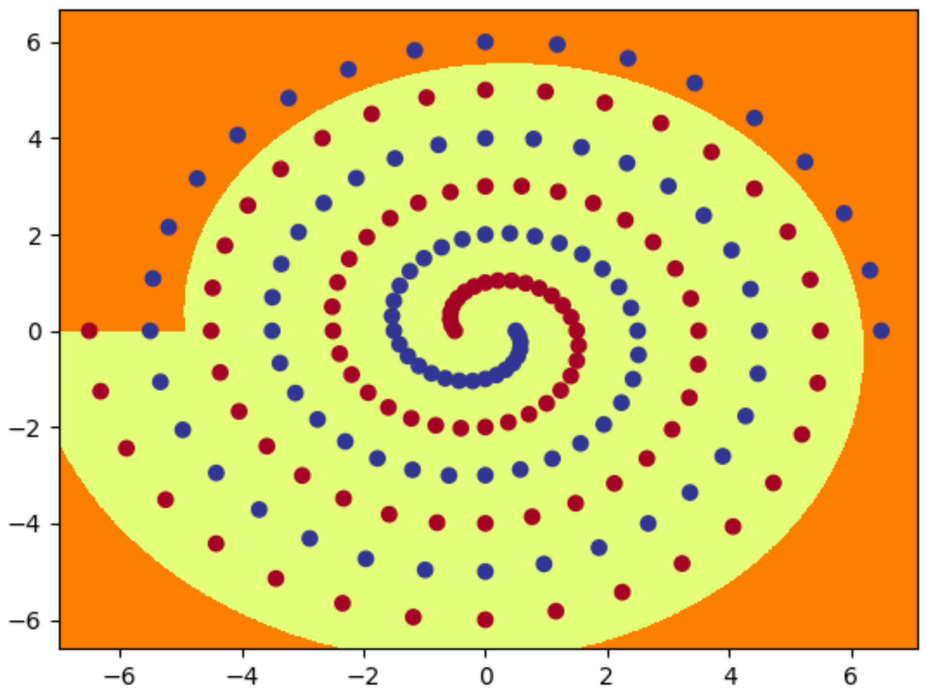
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

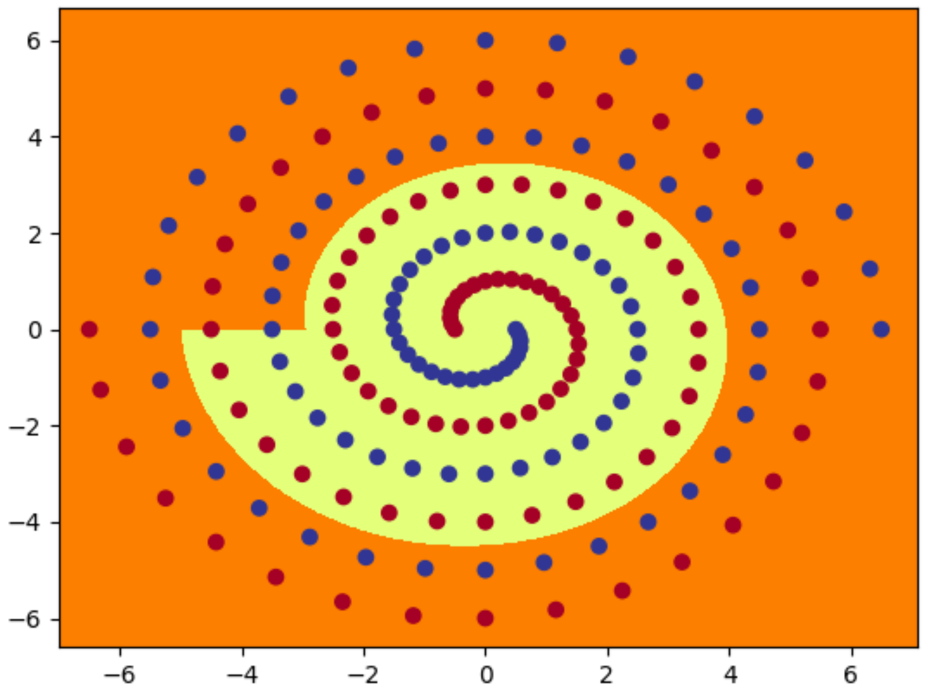


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



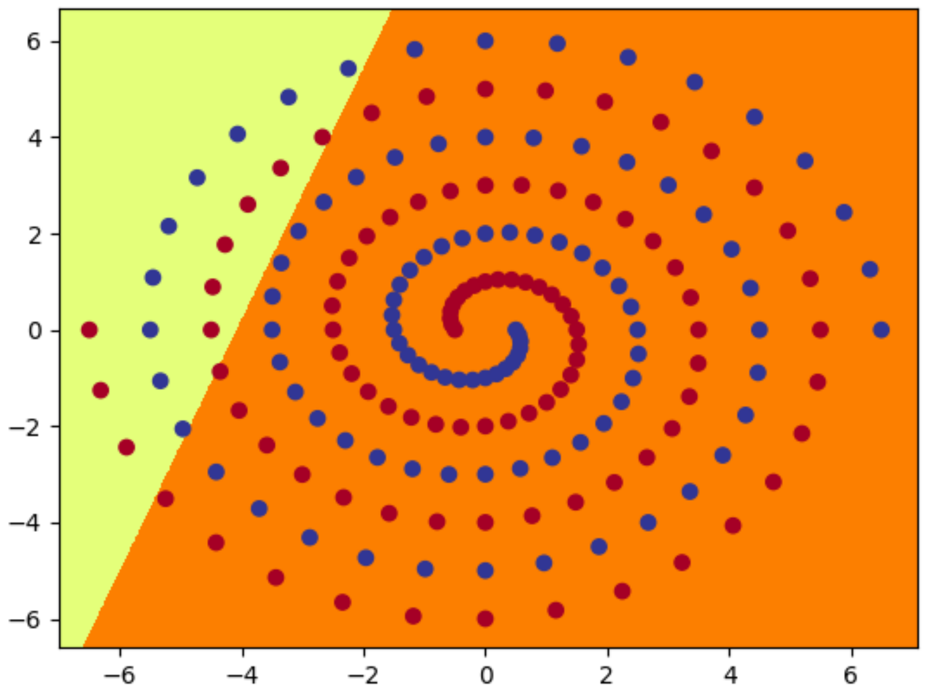
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



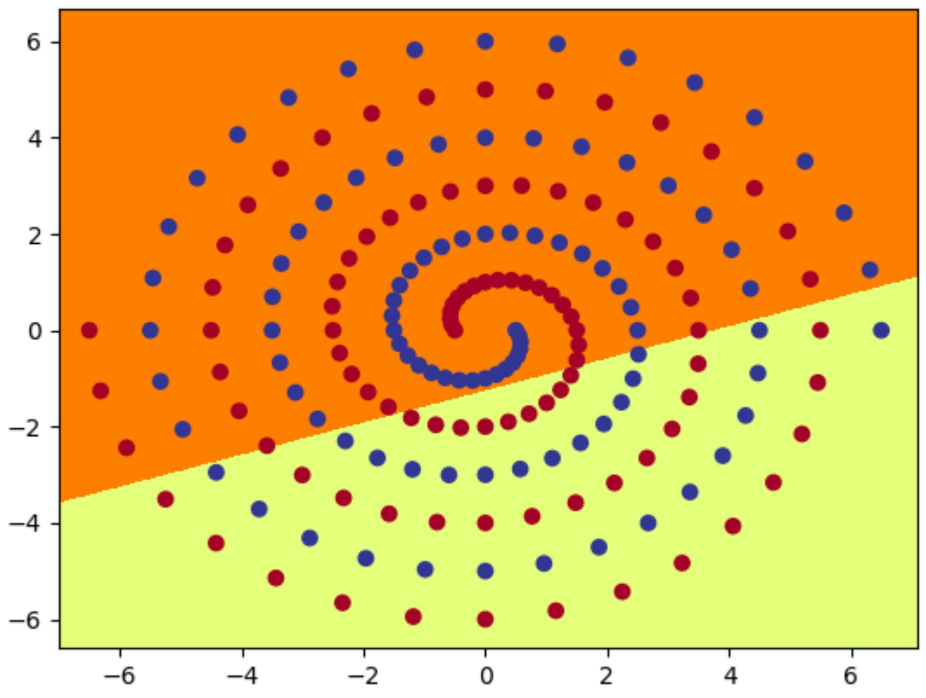


RawNet hidden nodes:

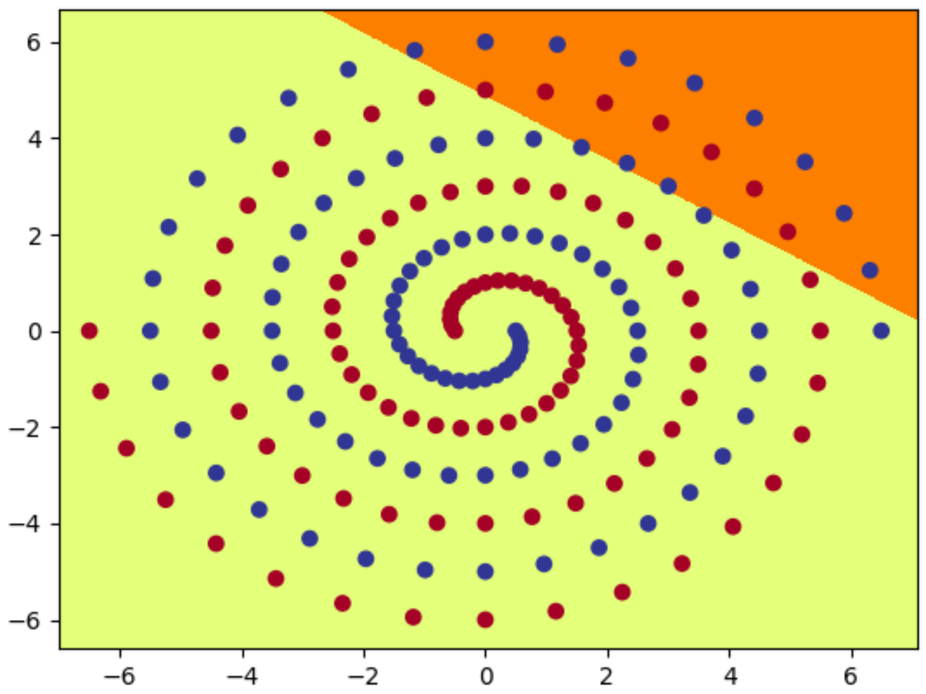
For layer1:

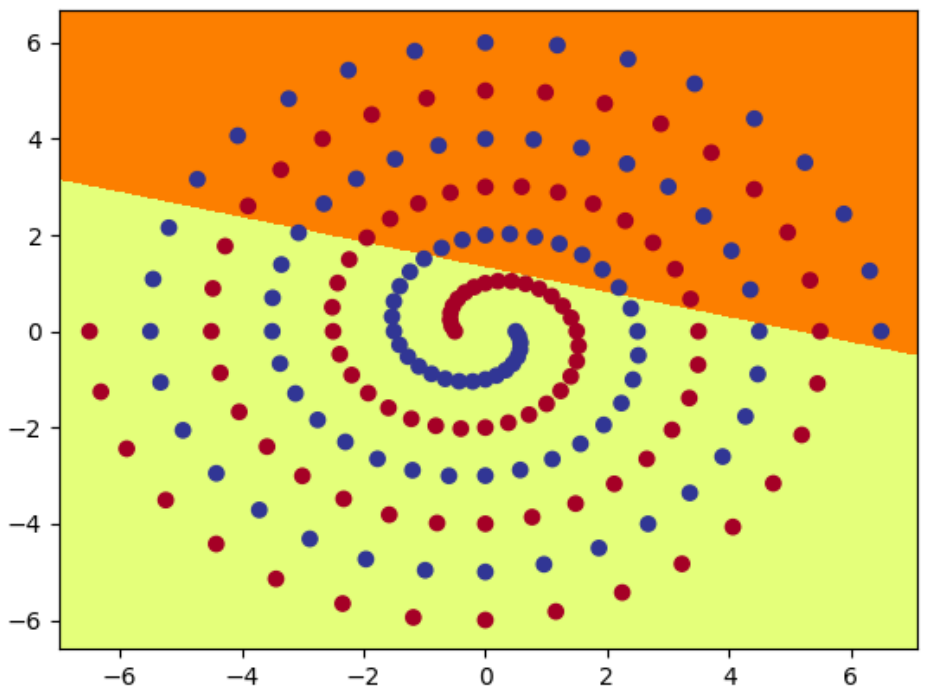


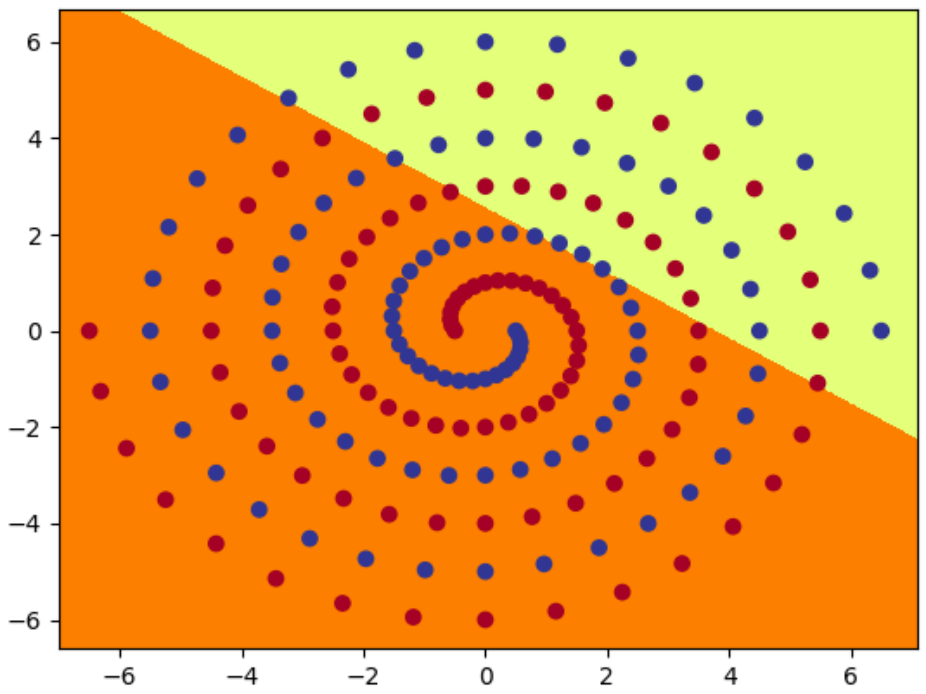
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

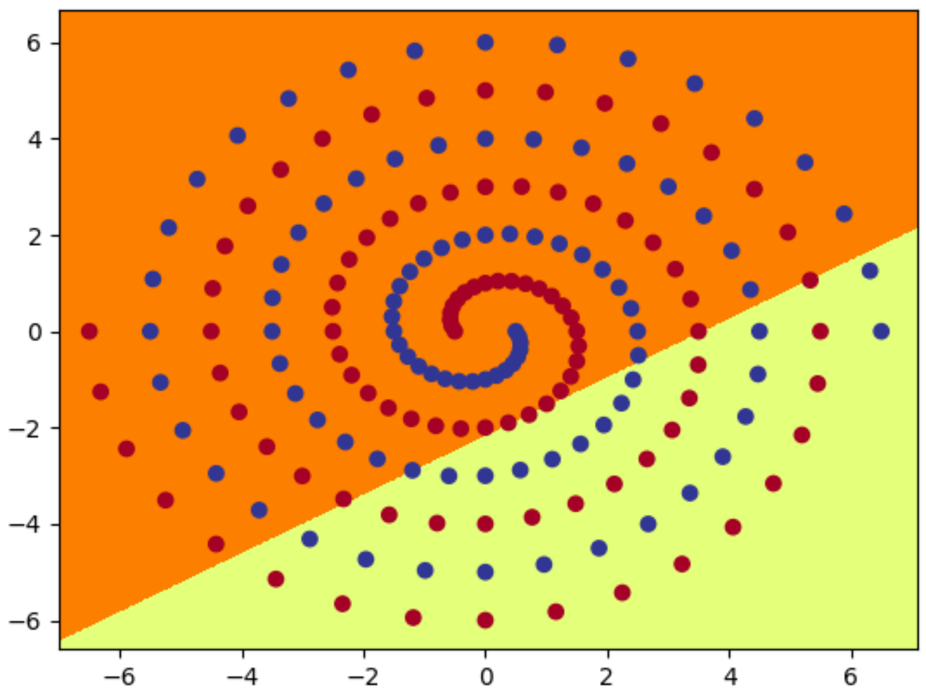


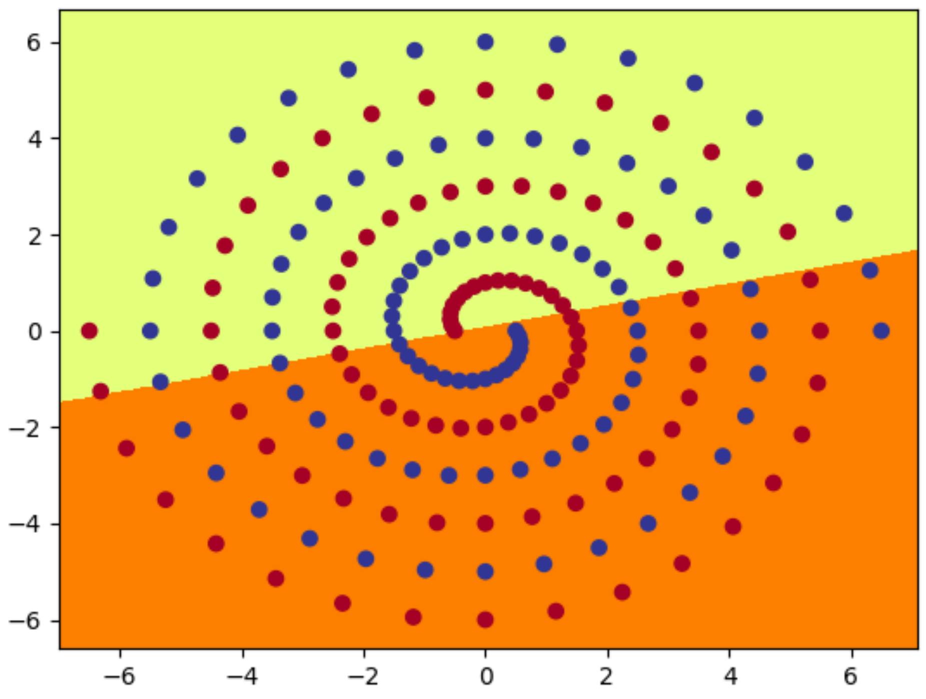
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

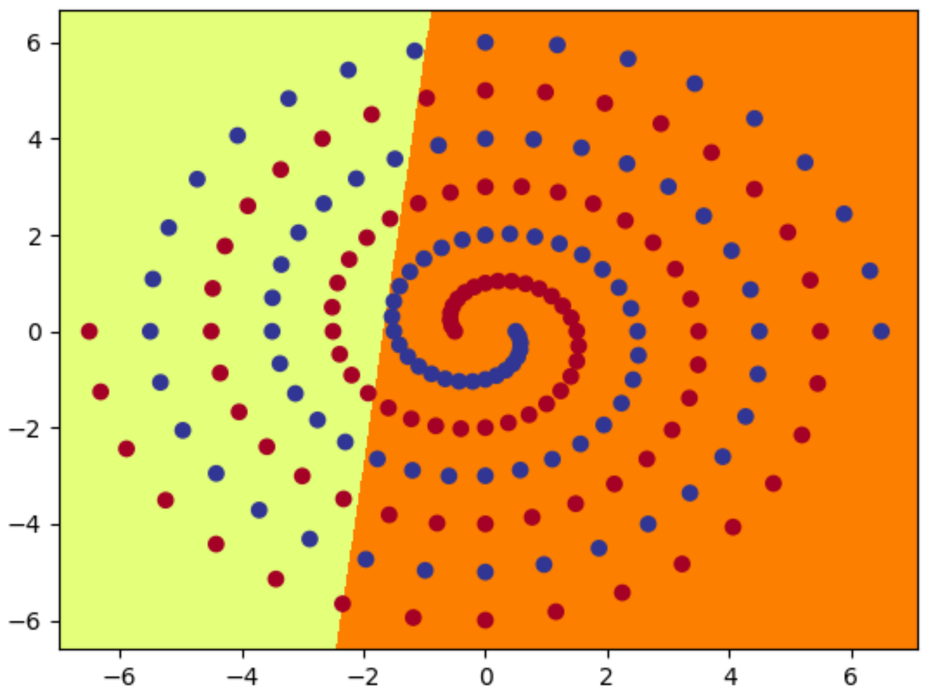

6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

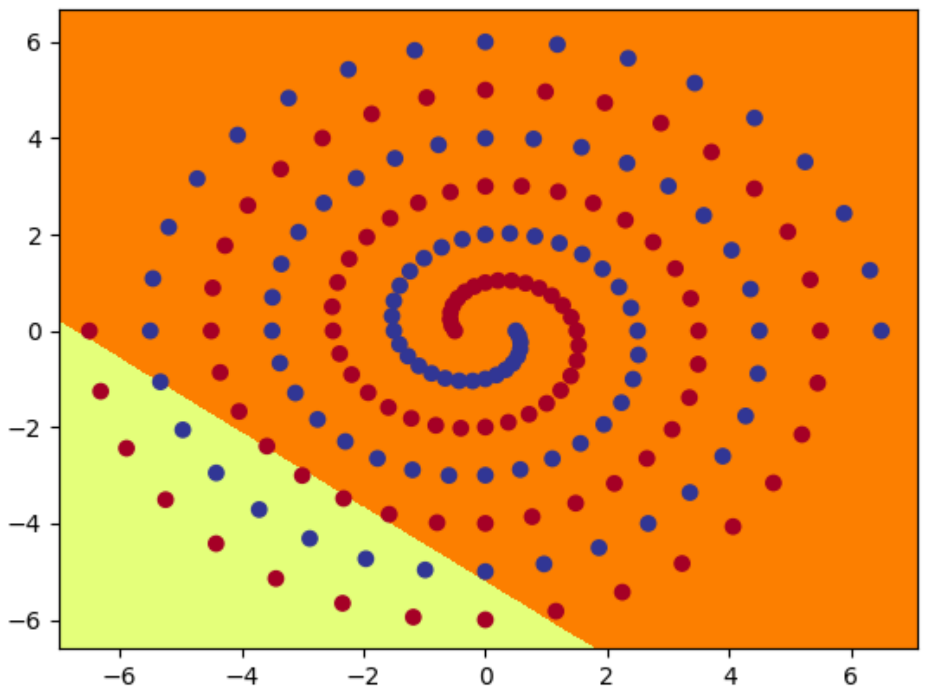

6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

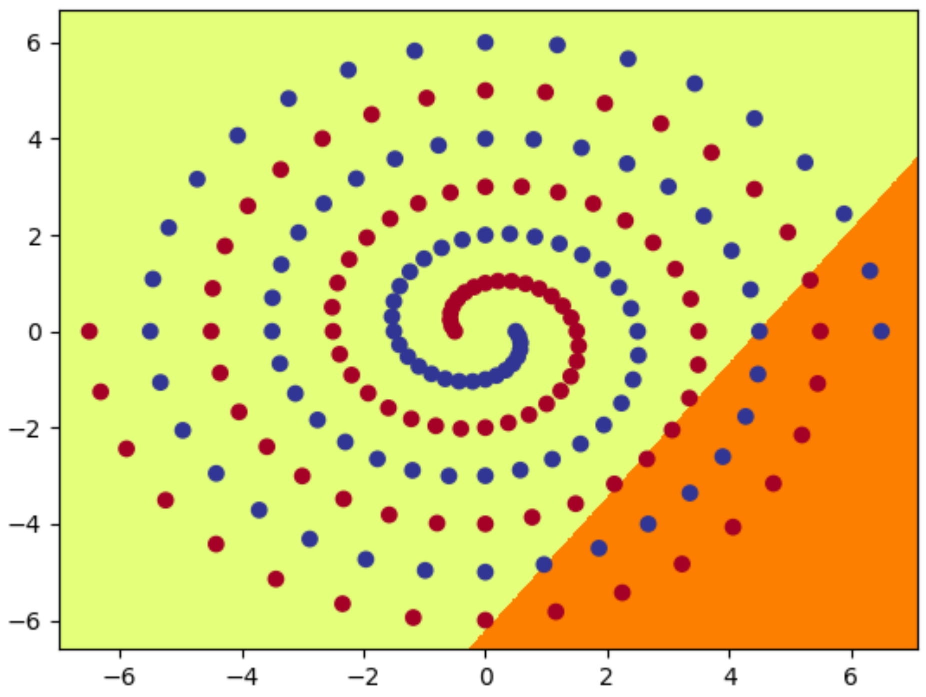


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

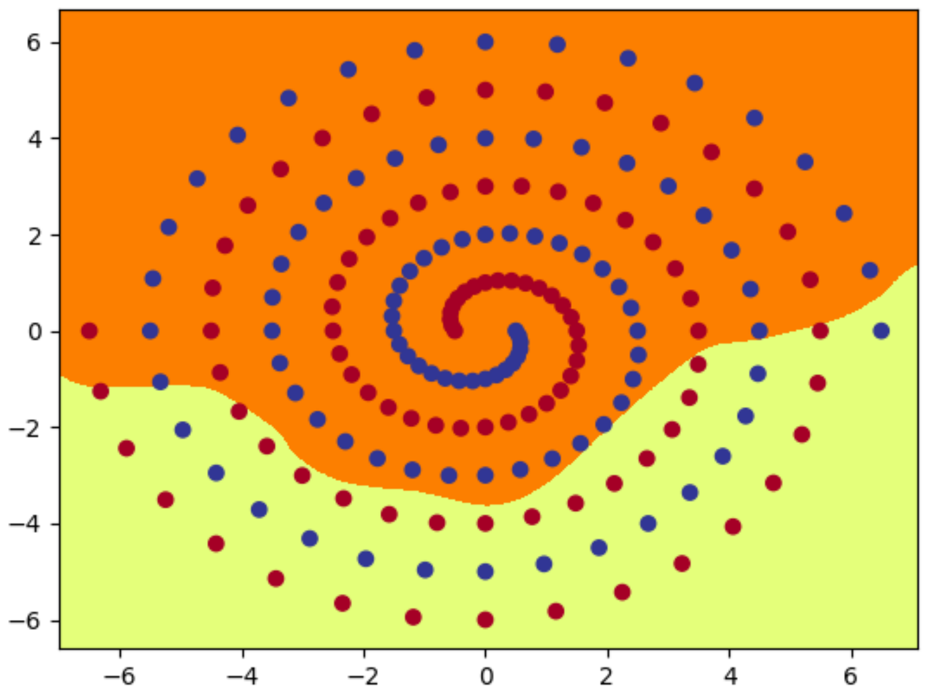


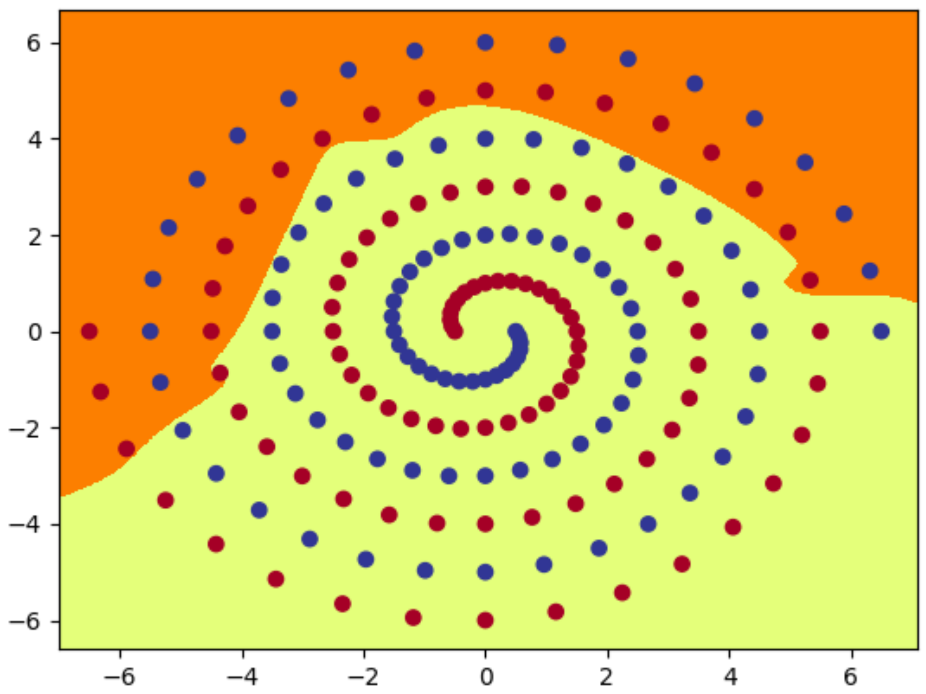
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

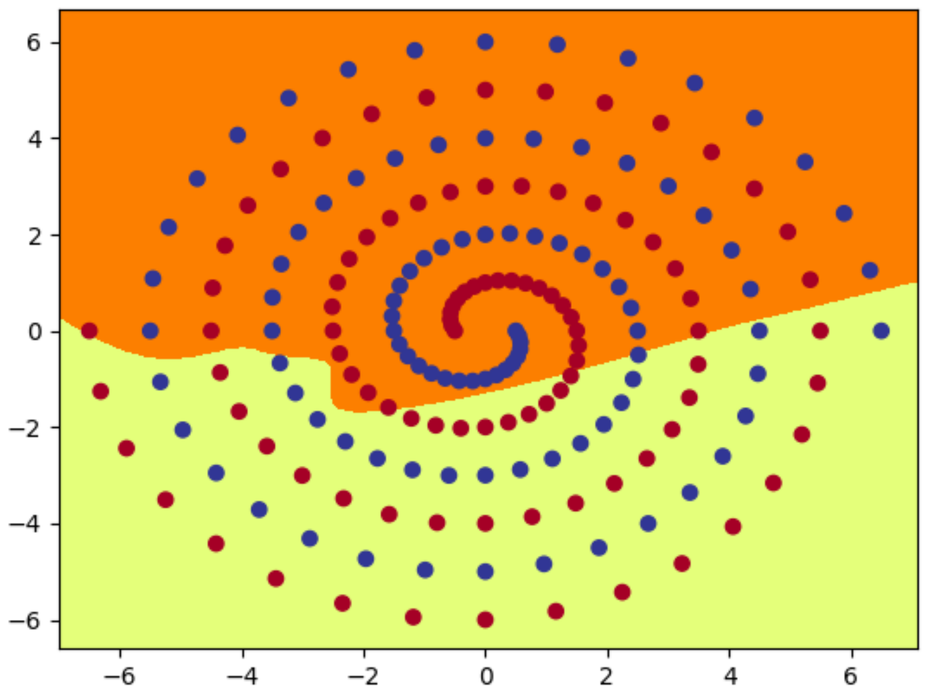




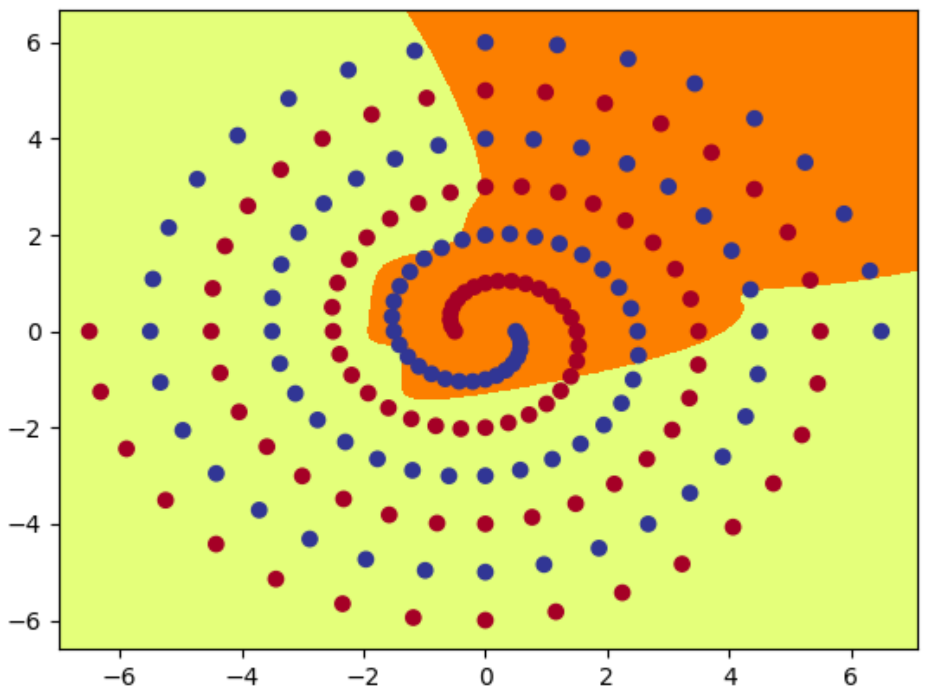
For layer2:


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

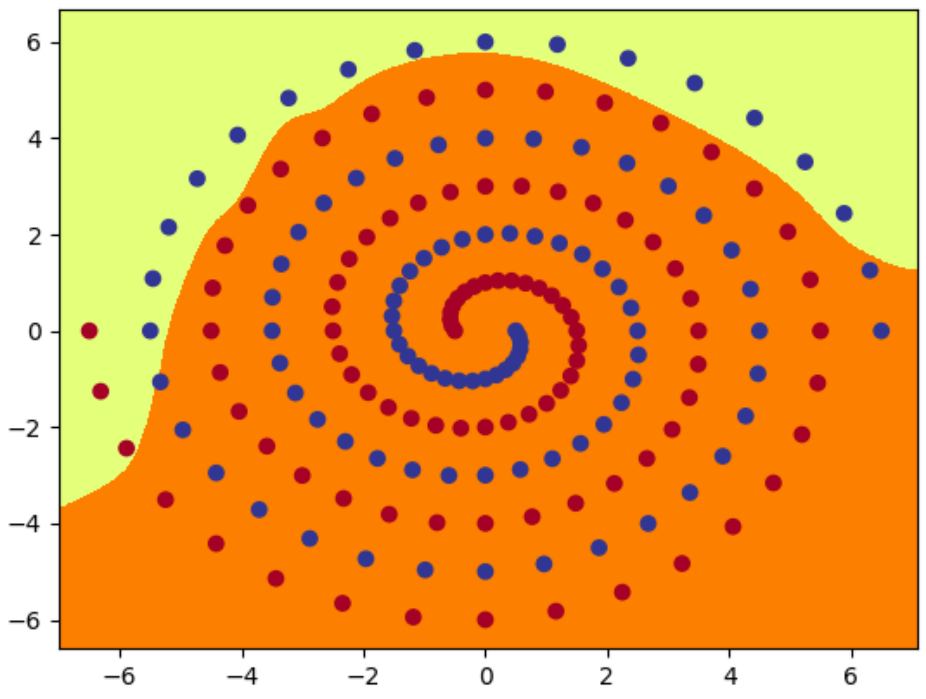

6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

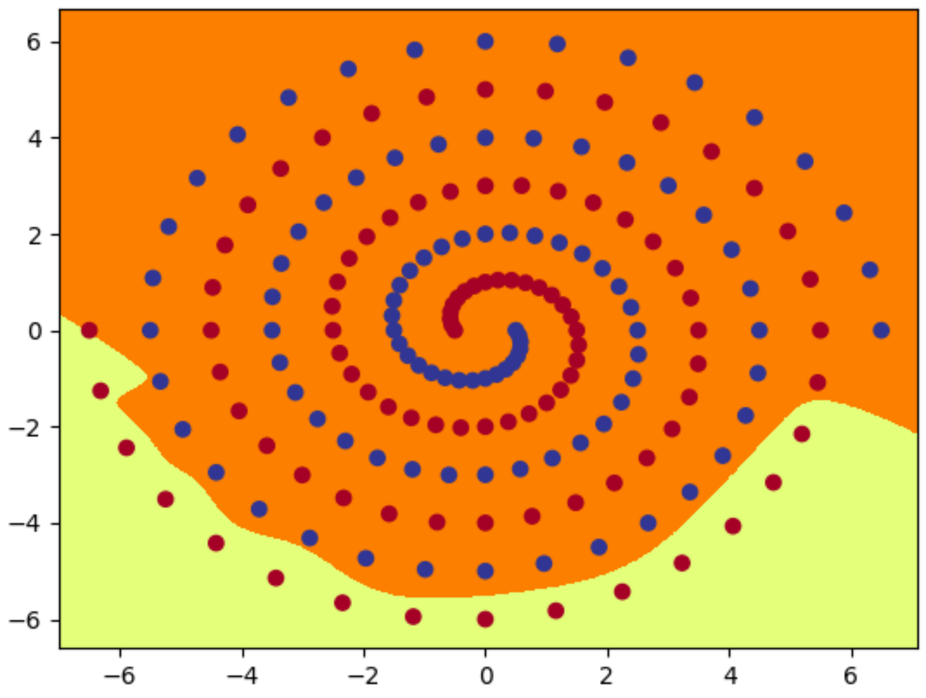


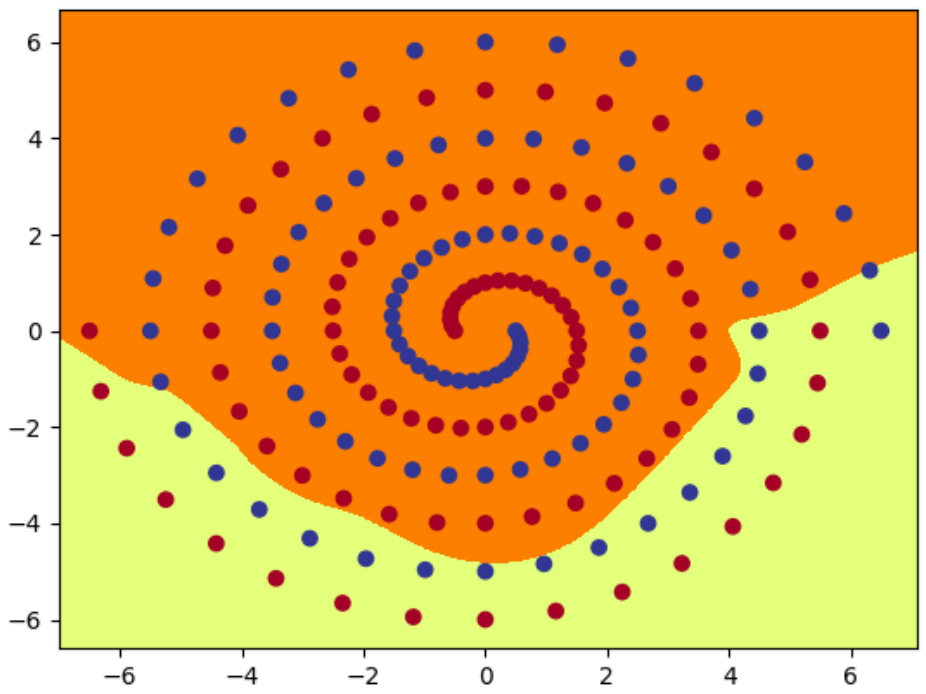
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



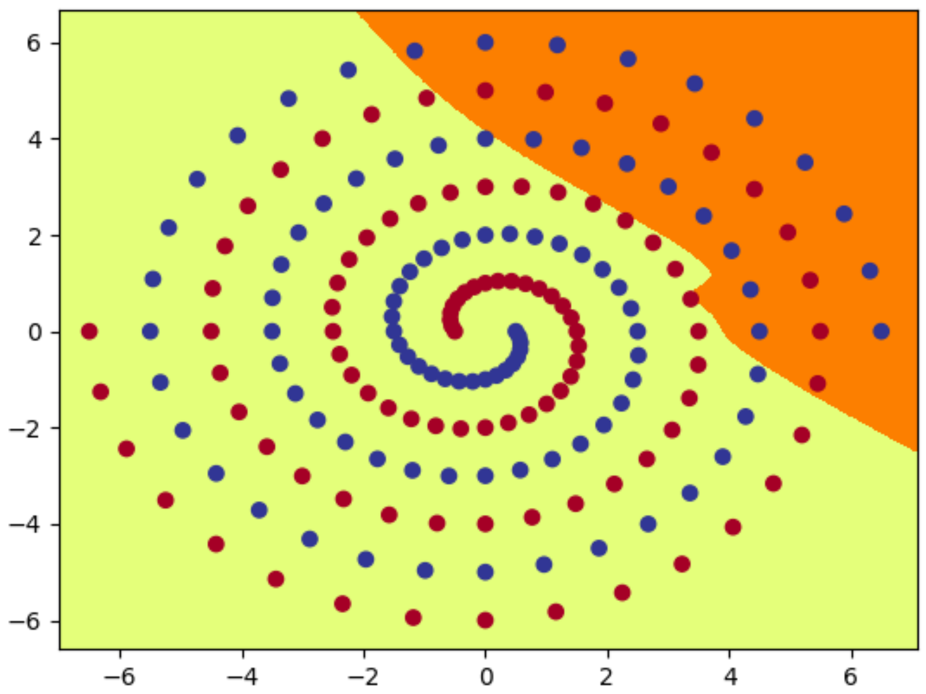
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

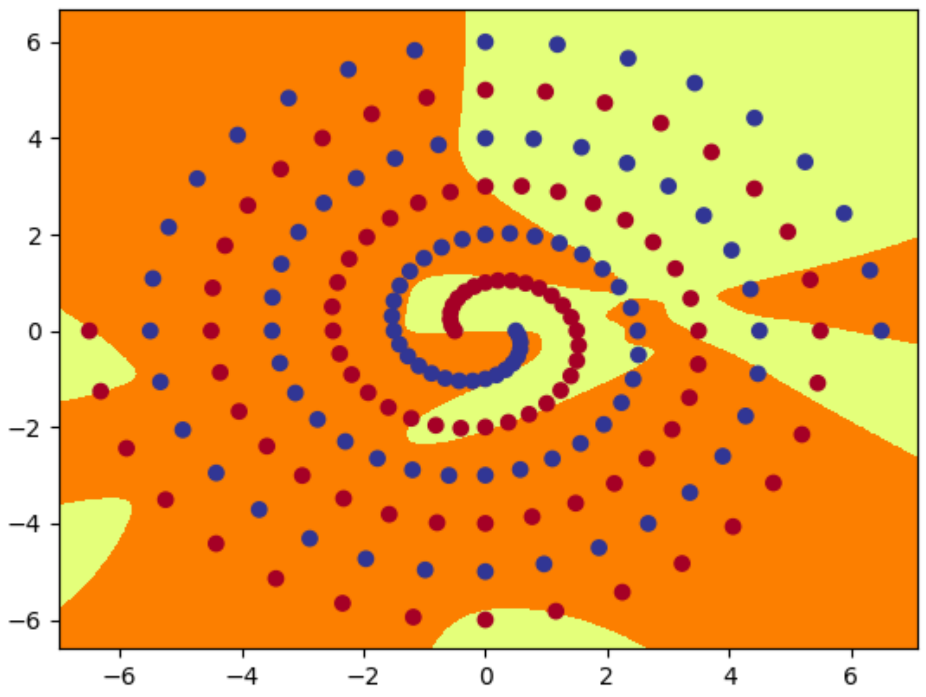

6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

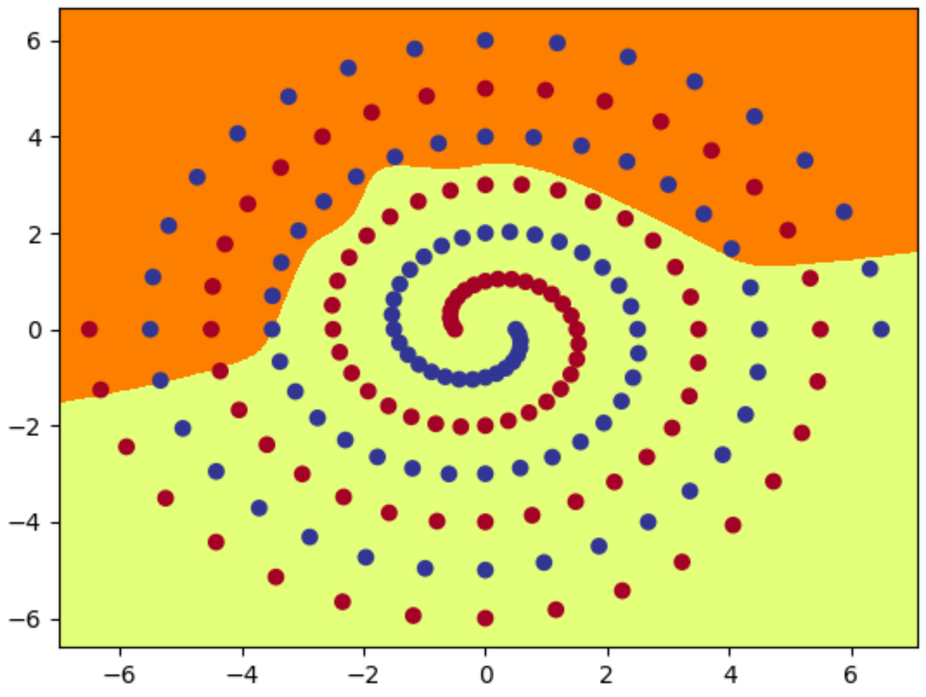


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



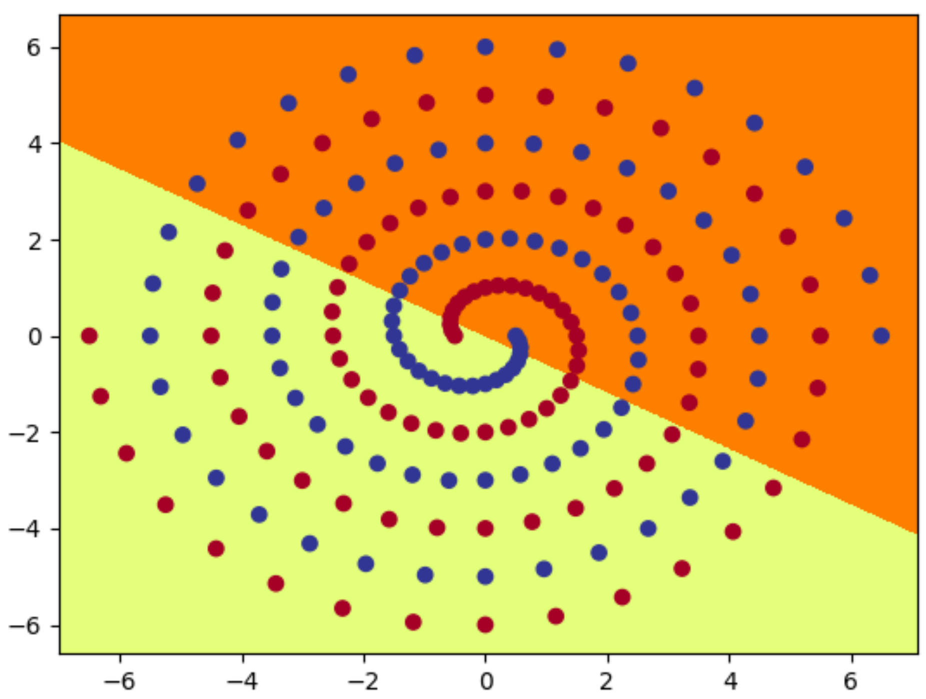
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

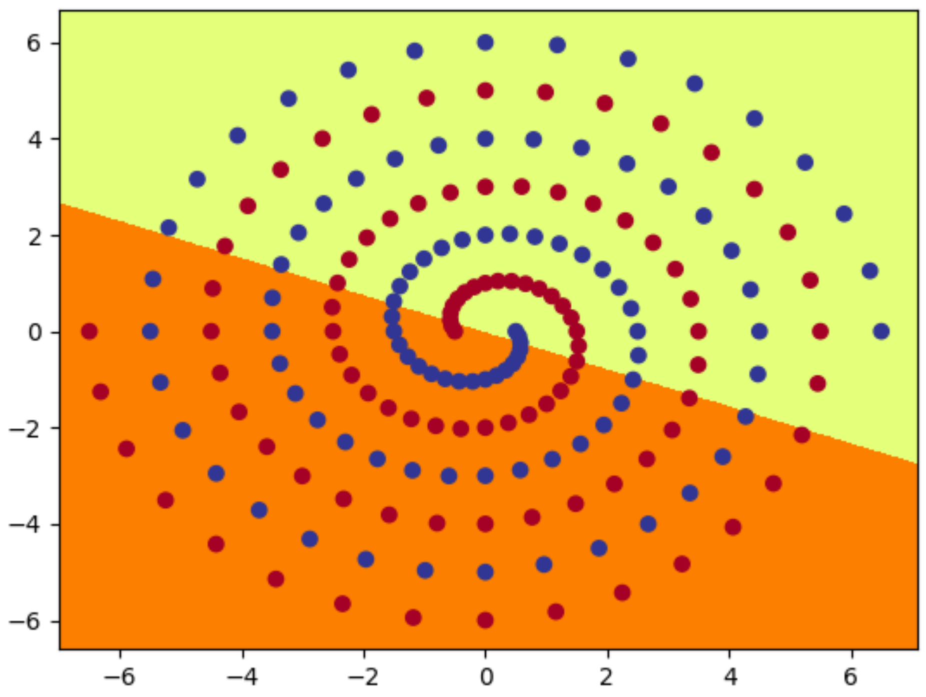


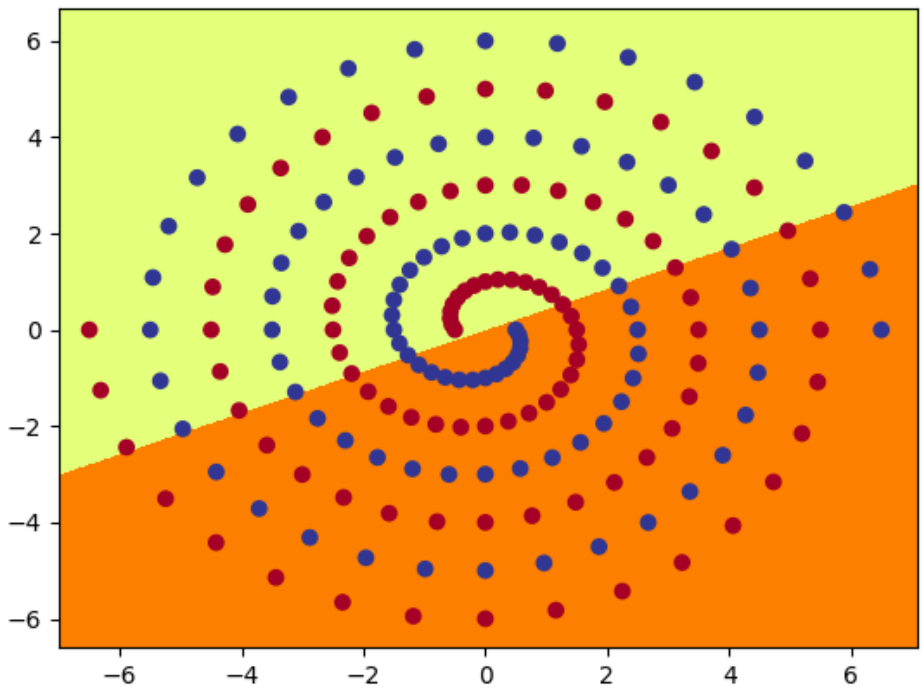


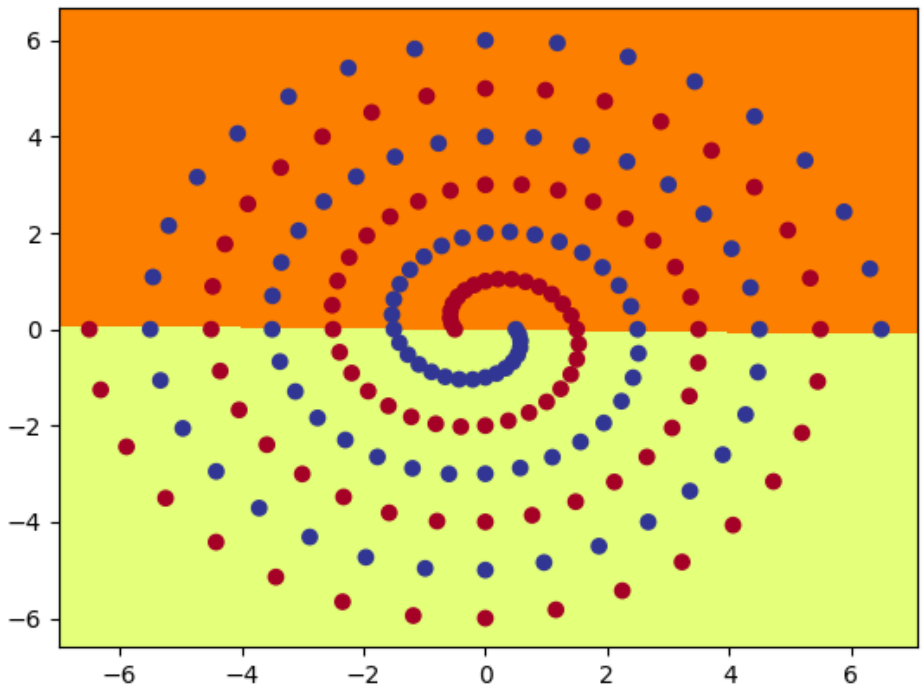
ShortNet hidden nodes:

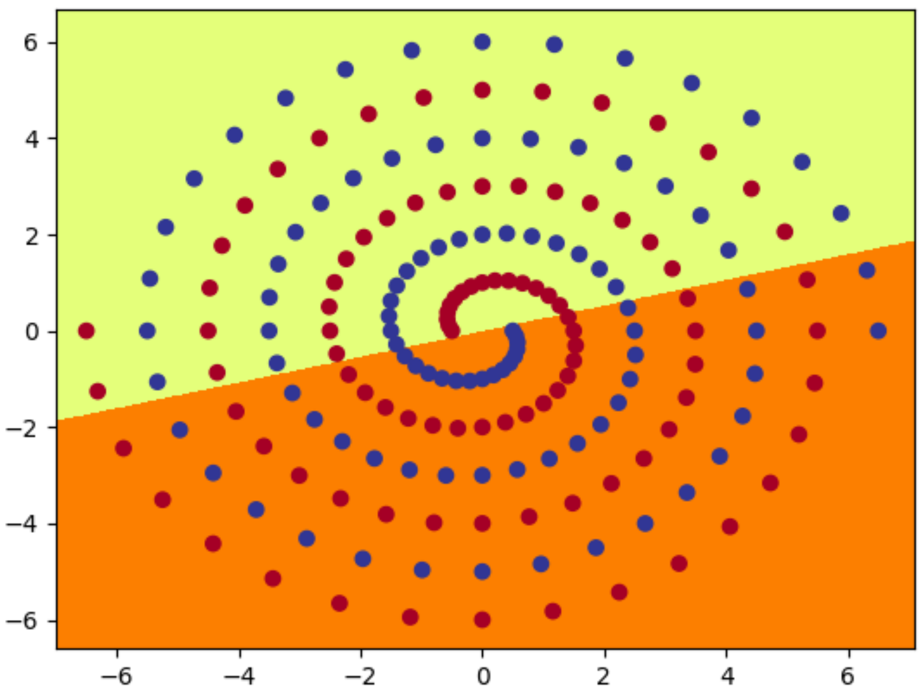
For layer1:


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

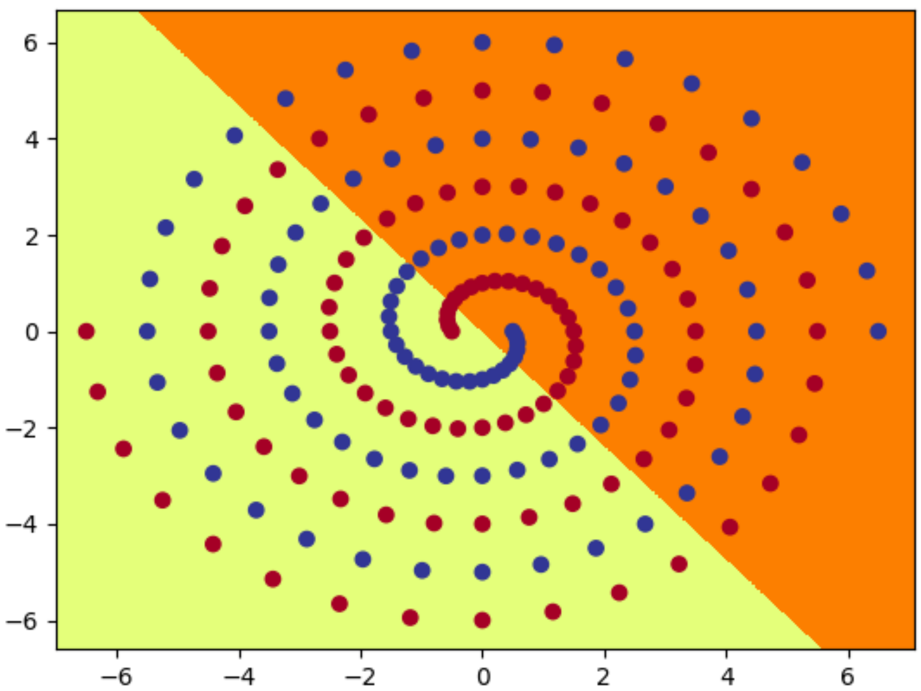

6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

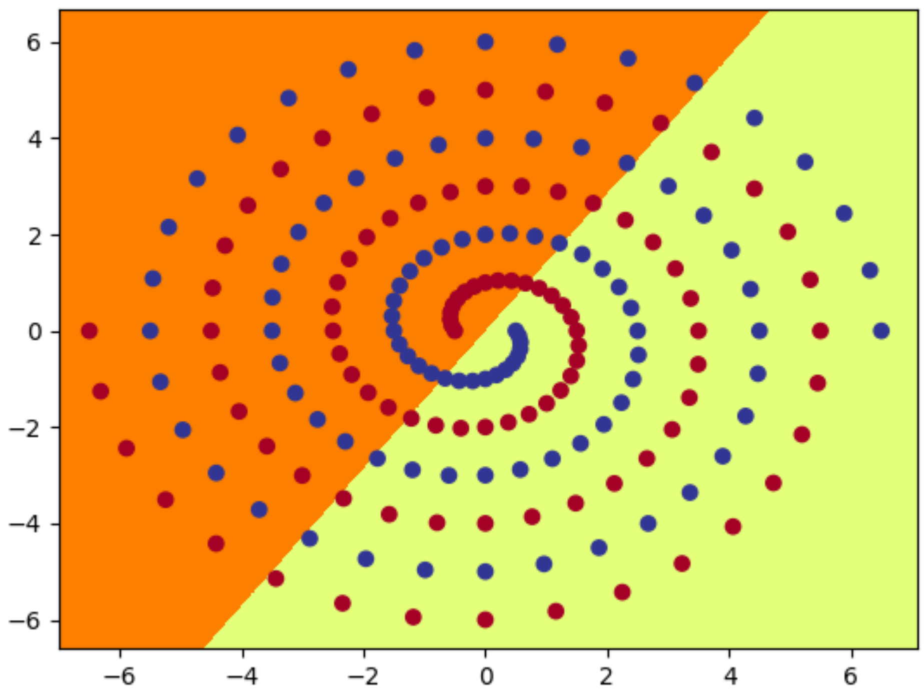

6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

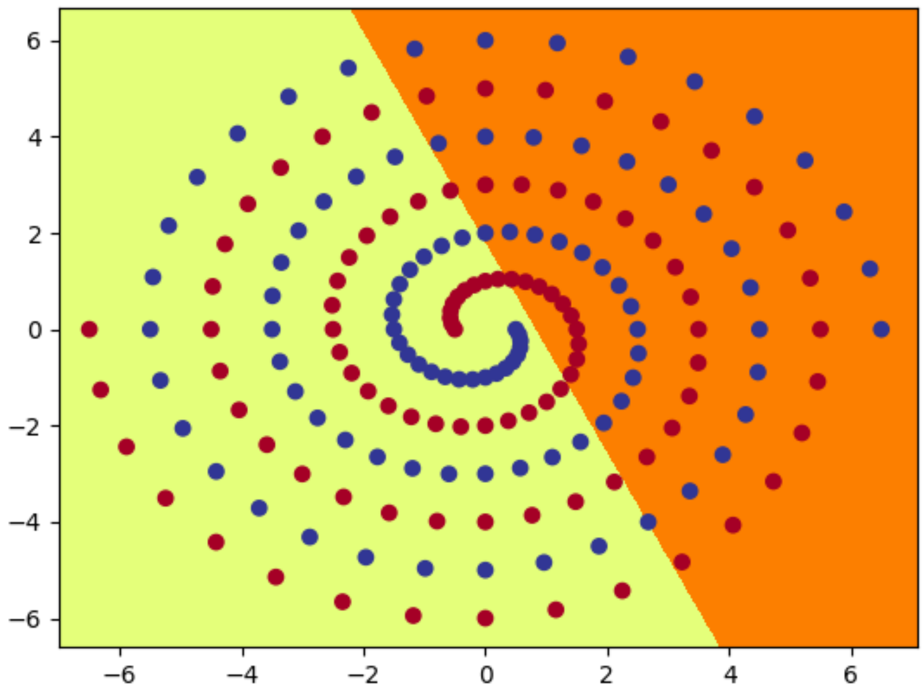


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

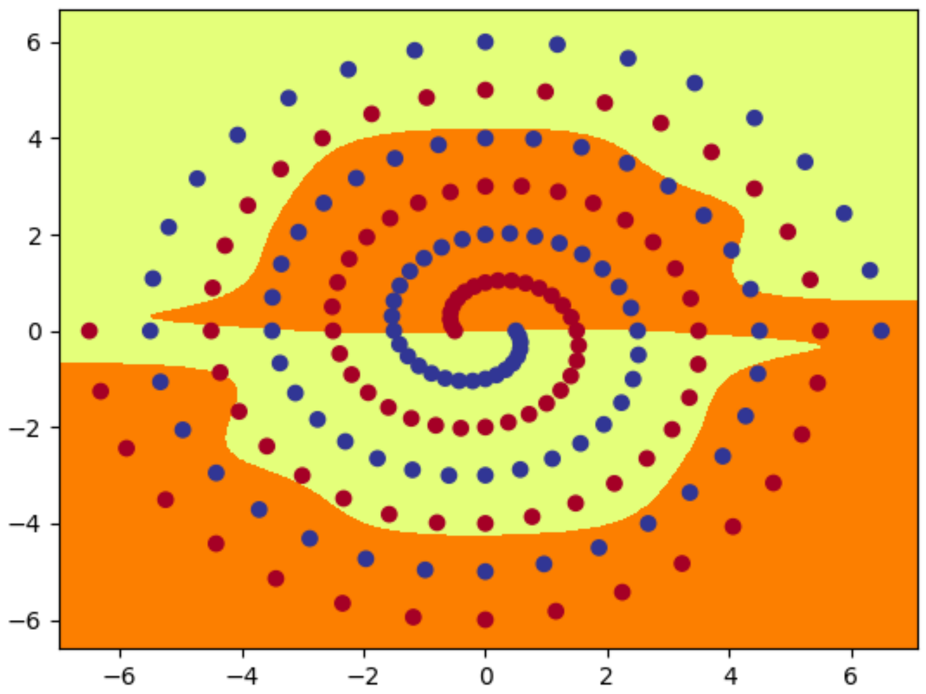


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

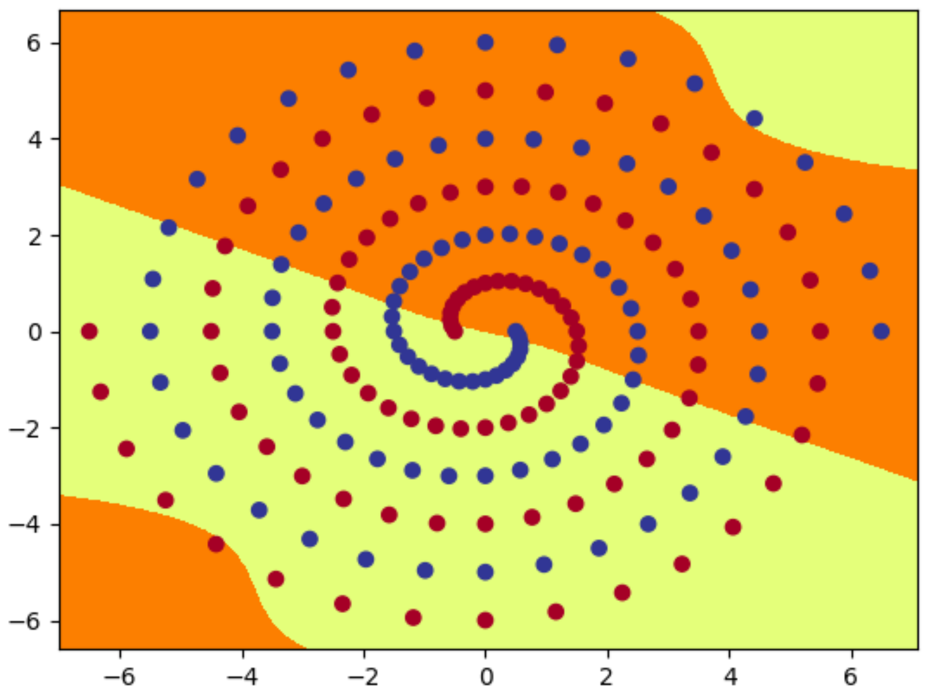




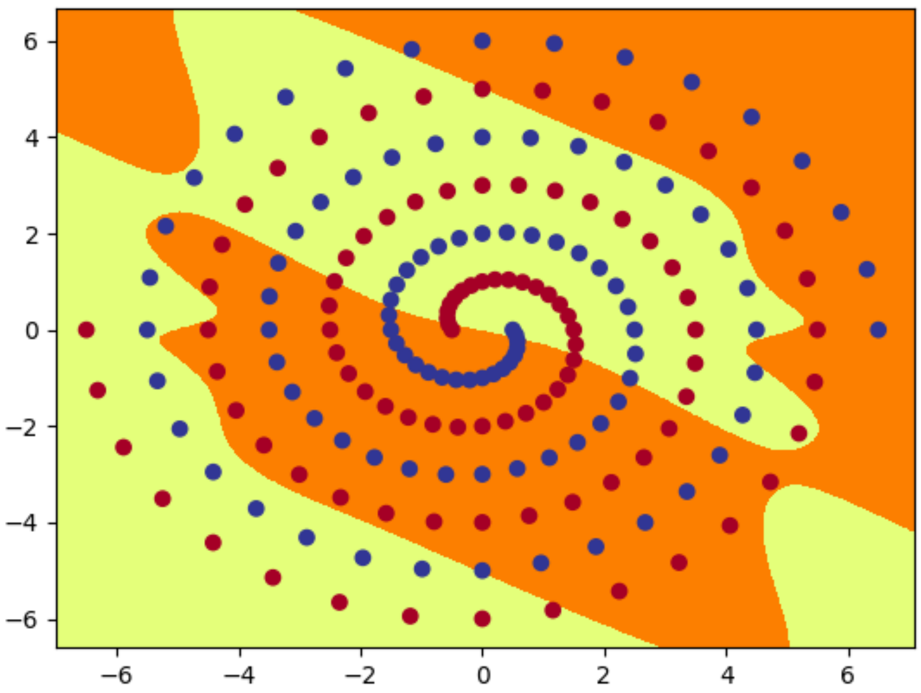
For layer2:



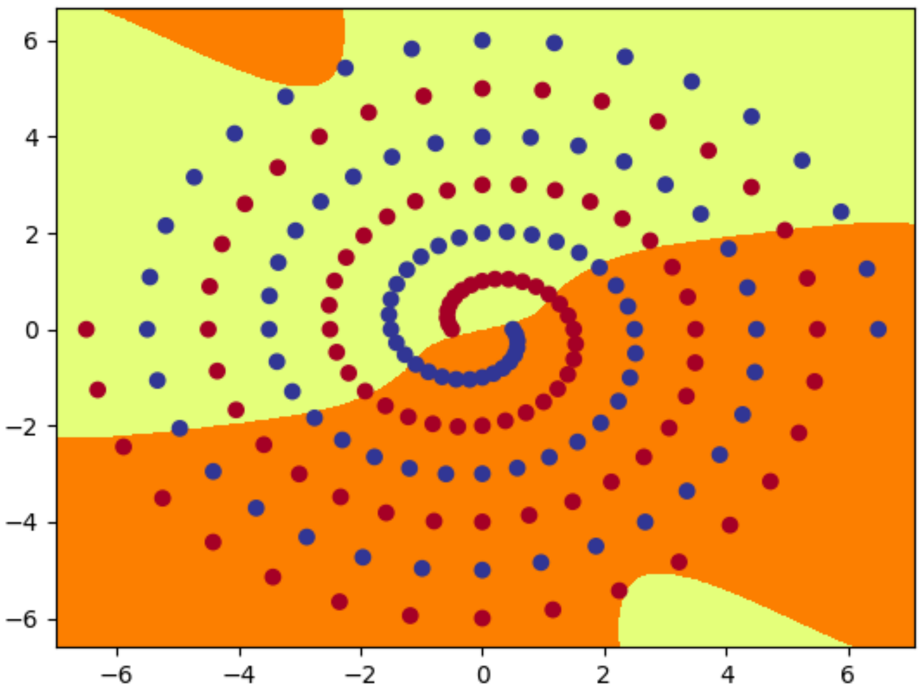
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



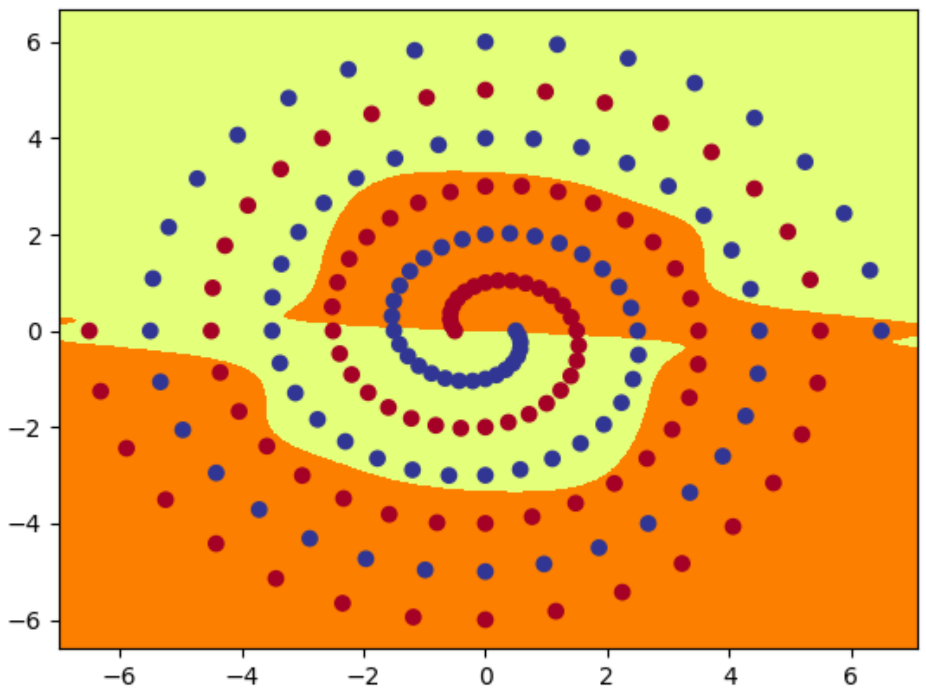
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



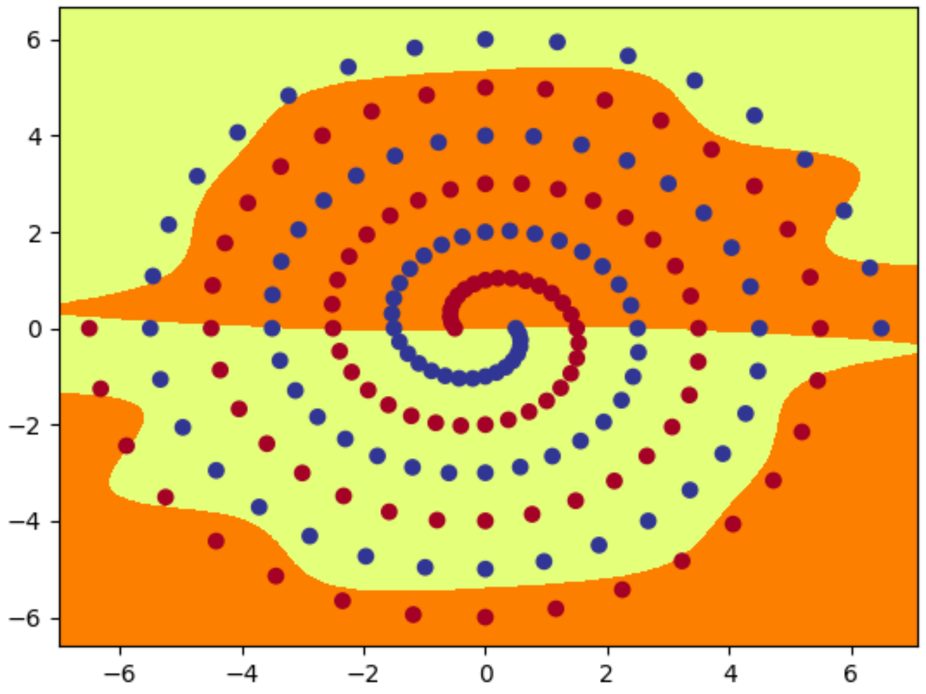
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



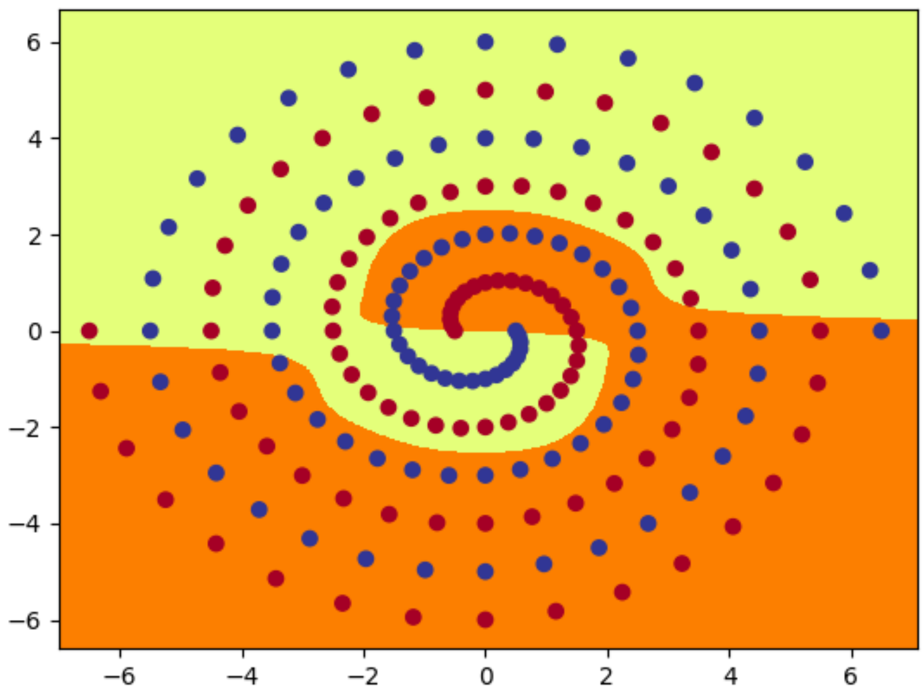
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

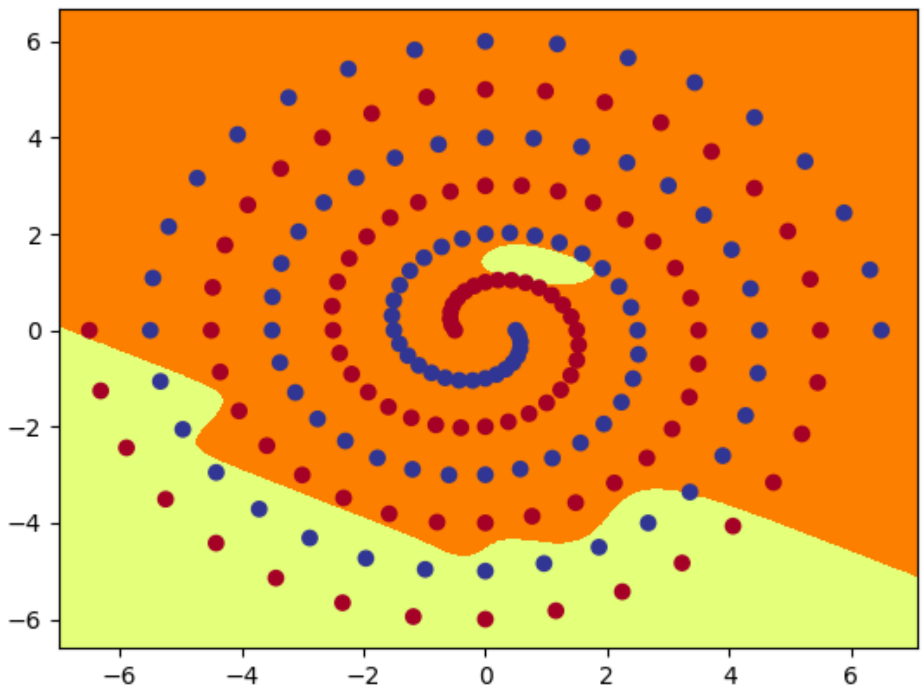


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6





8. According to the result computed by the hidden nodes, we can find that for layer 1, points are just classified by a simple line. After layer 2, points

are classified by different shapes or multiple lines. The result can be seen as a combination of the result of all these nodes.

As we just used Linear function to classify these points, the main purpose of linear function is to compute a line which can distinguish two different points. The progress of learning is to find the appropriate weight and bias. For RawNet and ShortNet, the value of initial weight size is the scale of the initial weight. A random weight will be choose from 0 to initial weight size, a suitable size can increase the speed and success of learning. For example, when initial weight size < 0.1 or > 0.25, the probability of success is quite low.

About the "naturalness" of the output function computed by the three networks. As this problem is a complex problem or this problem cannot be simply solved by a linear function, we need a non-linear function. As active function is a non-linear function, it can make our model to process more complex data, such as two spirals task.

I also tried the batch size of 194. With the same weight and hidden nodes, the accuracy of RawNet can reach 100% faster. Moreover, when I using SGD, the probability of success is quite low with the same parameters as using Adam.