Name: Junyu Ren zid: z5195715

COMP9444 Neural Networks and Deep Learning

Assignment 1 - Japanese Characters and Intertwined Spirals

Part 1: Japanese Character Recognition

1. The screenshot below is final confusion matrix and accuracy of NetLin.

Anaconda Prompt (anaconda)

```
<class 'numpy.ndarray'>
[[766.   7.   7.   3.  61.   8.   5.  16.  11.   8.]
 [  6. 672.  59.  37.  50.  27.  23.  27.  37.  53.]
 [  9. 106. 696.  62.  80. 125. 146.  27.  99.  87.]
 [ 14.  17.  25. 753.  20.  17.  10.  10.  39.   3.]
 [ 30.  30.  27.  15. 623.  19.  25.  86.   7.  53.]
 [ 64.  23.  18.  57.  21. 725.  24.  18.  30.  31.]
 [  2.  56.  47.  13.  32.  27. 723.  55.  44.  18.]
 [ 62.  14.  36.  18.  35.   9.  21. 625.   6.  29.]
 [ 30.  26.  47.  29.  20.  32.   9.  89. 707.  41.]
 [ 17.  49.  38.  13.  58.  11.  14.  47.  20. 677.]]

Test set: Average loss: 1.0104, Accuracy: 6967/10000 (70%)
```

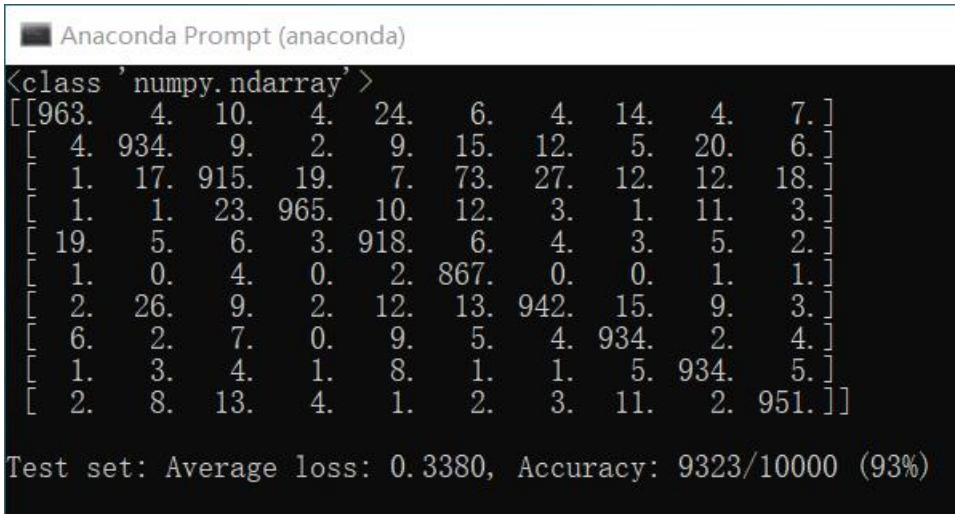
2. The screenshot below is final confusion matrix and accuracy of NetFull when hidden nodes number is 200. I tried the number of hidden nodes in [30, 50, 80, 100, 150, 200, 300, 500, 600, 800, 1000] and found when the number of hidden nodes >= 200, the accuracy goes stably (85%).

Anaconda Prompt (anaconda)

```
<class 'numpy.ndarray'>
[[868.   5.   7.   3.  39.  10.   3.  20.   9.   5.]
 [  3. 825.  15.   5.  31.  12.  10.  11.  30.  15.]
 [  1.  28. 841.  30.  15.  68.  48.  17.  26.  50.]
 [  6.   5.  37. 924.   8.  11.   8.   3.  47.   2.]
 [ 28.  26.  11.   2. 813.  12.  13.  22.   4.  29.]
 [ 28.   9.  15.  11.  10. 840.   7.  11.   9.   4.]
 [  2.  54.  28.   4.  29.  22. 892.  32.  33.  26.]
 [ 36.   6.  11.   3.  18.   1.   8. 830.   4.  12.]
 [ 24.  15.  19.   8.  20.  18.   2.  21. 830.  11.]
 [  4.  27.  16.  10.  17.   6.   9.  33.   8. 846.]]

Test set: Average loss: 0.4932, Accuracy: 8509/10000 (85%)
```

3. The screenshot below is final confusion matrix and accuracy of NetConv (without max pooling layer and padding, more exploration and discussion in 4c(3)).



4.

a. From above we can know that the accuracy of single linear network is relatively low, a 2-layer fully-connected network is in the middle and a convolutional network achieves relatively high accuracy. It can be concluded that pure linear or fully-connected network is not quite suitable for image recognition task because of inadequate fitting ability and convolutional network is the best choice when it comes to image processing.

b. We can look at each row of 3 confusion matrix above and find the second largest number. Their indexes (starts from 1 for descriptive convenience) indicate which character the current character is most likely to be mistaken as. We can draw a table to show it. Note: i-j means the ith character is mistaken as jth character. We can call it a confusion pair.

NetLin	NetFull	NetConv
1-5	1-5	1-5
2-3	2-5	2-9
3-7	3-6	3-6
4-9	4-9	4-3
5-8	5-10	5-1
6-1	6-1	6-3
7-2	7-2	7-2
8-1	8-1	8-5
9-8	9-1	9-5
10-5	10-8	10-3

From the table we can count frequency of each confusion pair and finally found the 1st character (o) is mostly likely to be mistaken as the 5th character (na) and the 7th character (ma) is also mostly likely to be mistaken as the 2nd character (ki). The reason is that they have similar visual structure and the handwriting are not as neat and distinguishable as printed ones, so sometimes networks may mistake them.

c. (1) I tried different command line parameters (i.e. learning rate) towards NetLin to explore whether a higher accuracy can be achieved. I tried values from [0.001, 0.005, 0.008, 0.02, 0.05,

0.1, 0.2]. I found that when learning rate is close to default value 0.01 (0.005,0.008), the network can achieve similar accuracy (70%) but when it is too large or too small, accuracy is even worse (67%). So, the low performance of NetLin is mainly comes from the architecture itself instead of other factors.

(2) I tried different values for momentum towards NetFull. Since the range of momentum is between 0 and 1, I selected values from [0.1, 0.2, 0.3, 0.7, 0.8, 0.9] (default one is 0.5). I found that when momentum is bigger, the higher accuracy this network can achieve, with peak at 88% when momentum is set to 0.9.

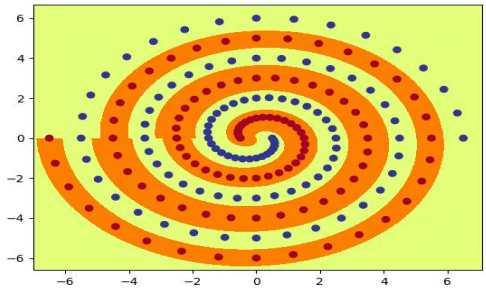
(3) I tried to add a max pooling layer to NetConv to see how it works although my NetConv can reach at least 93% accuracy after 10 epochs with basic structure conv1 -> relu -> conv2 -> relu -> linear1 -> relu -> linear2 -> log_softmax. After adding a pooling layer at the end of the convolution layer, the network structure is: conv1 -> relu -> conv2 -> relu -> pooling -> linear1 -> relu -> linear2 -> log_softmax. I set the kernel size of pooling layer is 5 and padding is 2. I found that this network can achieve a accuracy of more than 94% (almost 95%). Another thing I want to mention is the selection of size (in_channels, out_channels, kernel_size) of convolution layer. I once chosen (1, 8, 3) and (8, 16, 3) but the accuracy is below 93%, then I tried (1, 16, 3) and (16, 32, 3) and the accuracy is still slightly below 93%. This means it is still under-fitting. I tried kernel_size = 5 and it works. I also notice the 'padding' parameter, I found (1, 16, 5) and (16, 32, 5) without padding can achieve our goal but when I set padding=2, the accuracy is more than 94%.

Part 2: Twin Spirals Task

1. See codes in spiral.py in detail

2. Firstly I tried hidden node default number 10 and found it will 100% in all runs. Then I reduced number of hidden nodes by 1 at each attempt, and found when number of hidden nodes is 7, it will get 100% correct of training set within 20000 epochs in almost all runs. However, when I tried hidden node number = 6, sometimes it failed to reach the goal. So, the minimum hidden nodes number which satisfies the requirement is 7. Other meta-parameters such as --lr, --init are as default.

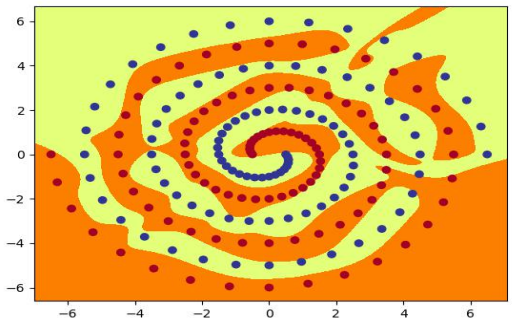
Below is polar_out.png when number of hidden nodes = 7:



3. See code in spiral.py in detail

4. The number of hidden nodes of each layer is fixed at 10 in this question. I firstly tried the default value of initial weights (0.1) and found that sometimes after 20000 epochs, the accuracy cannot reach 100%. Then I added 0.1 by 0.01 at each further attempt and found when initial weight is 0.12, the network can get 100% correct in training data within 20000 epochs at almost all runs. Other parameters like --lr are as default.

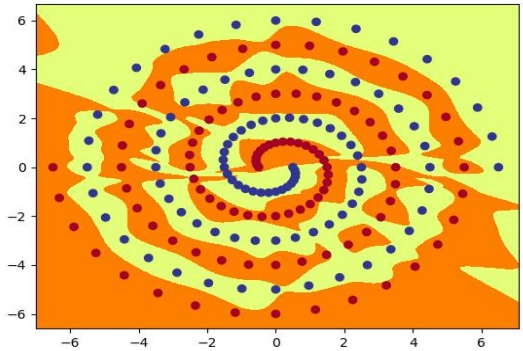
Below is raw_out.png when initial weight is 0.12:



5. See code in spiral.py in detail

6. This problem involves the determination of 2 parameters --init and --hid, my strategy is 'fix one and determine another'. I firstly tried the initial weight 0.12 which was suitable for RawNet and temporary fixed number of hidden nodes with default value 10. I found initial weight of 0.12 is acceptable as the network classified training set 100% correct in 6800 epochs. Afterwards, I reduced default hidden node number by 1 each time and found that when hidden node number is 8, our goal is reachable in almost all the time but the network cannot always get 100% accuracy within 20000 epochs when there are 7 hidden nodes. So, the minimum number of hidden nodes satisfies our requirement is 8. Other meta-parameters are as default.

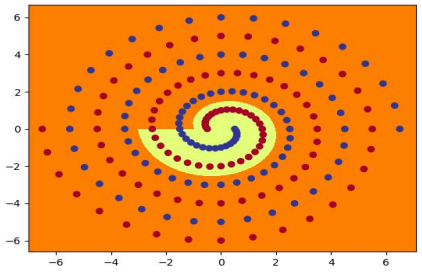
Below is short_out.png when initial weight is 0.12 and hidden node number is 8.



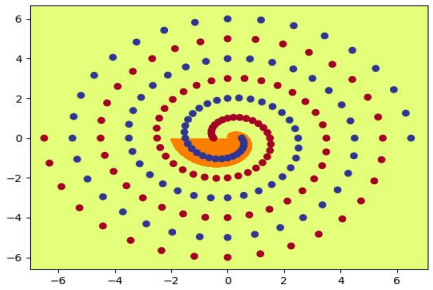
7. See code in spiral.py in detail and the images of hidden nodes generated of 3 networks are as follows (Parameters are as default if not additionally stated):

(1) PolarNet (when hidden node number = 7)

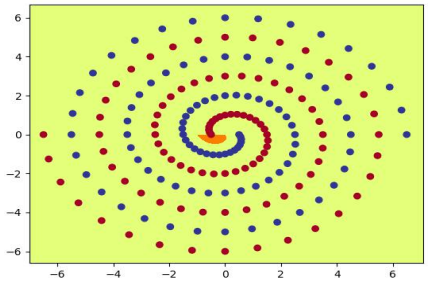
polar1-0.png:



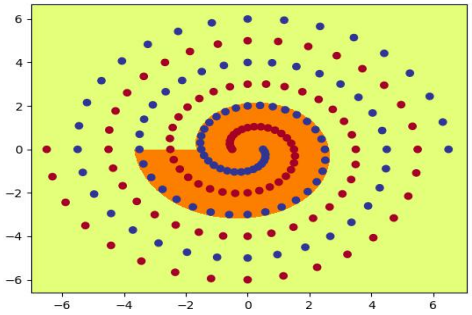
polar1-1.png:



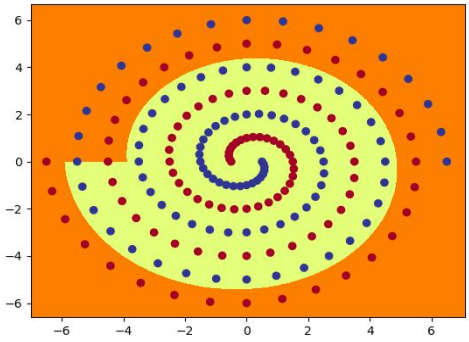
polar1-2.png:



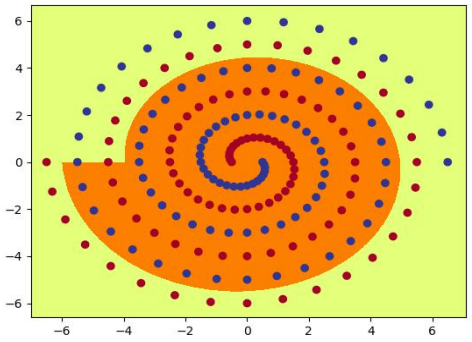
polar1-3.png:



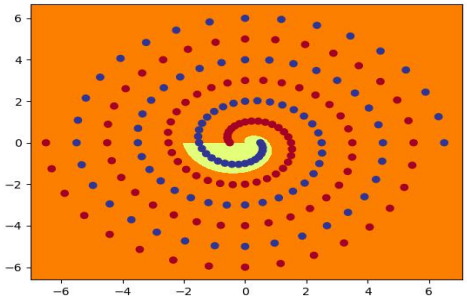
polar1-4.png:



polar1-5.png:

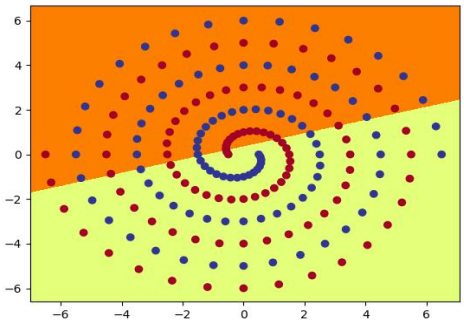


polar1-6.png:

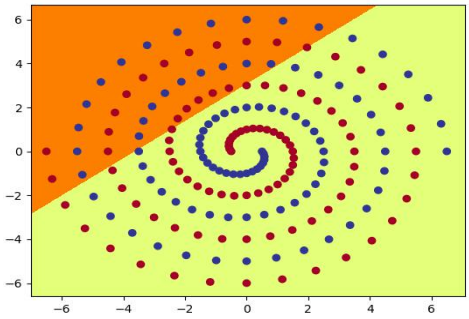


(2) RawNet (when initial weight is 0.12)

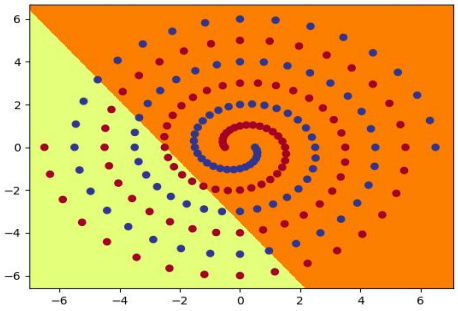
raw1-0.png:



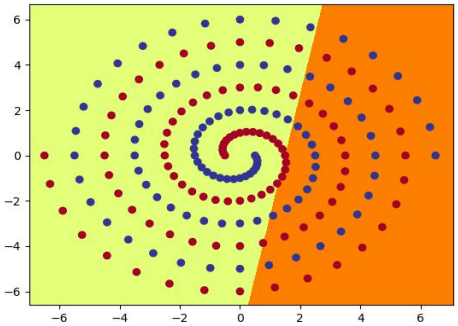
raw1-1.png:



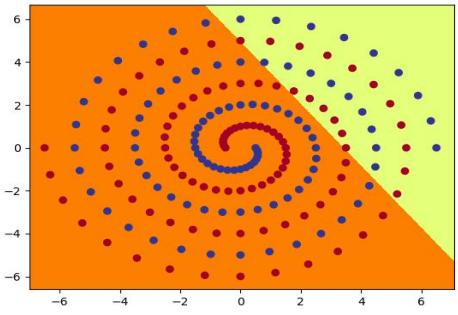
raw1-2.png:



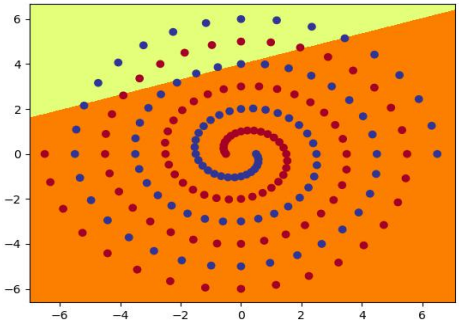
raw1-3.png:



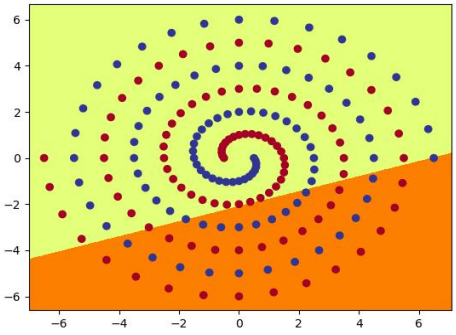
raw1-4.png:



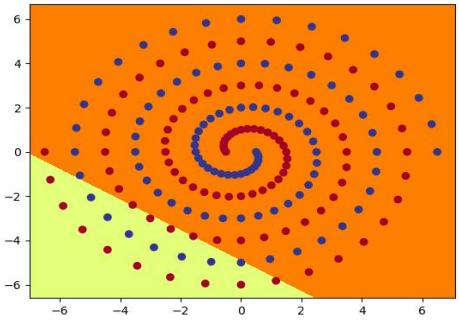
raw1-5.png:



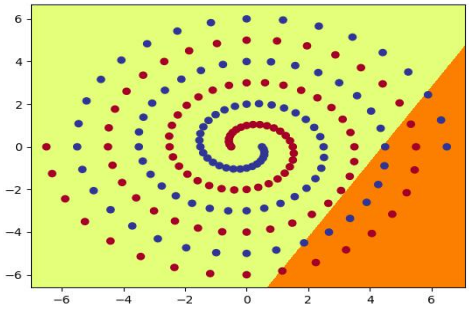
raw1-6.png:



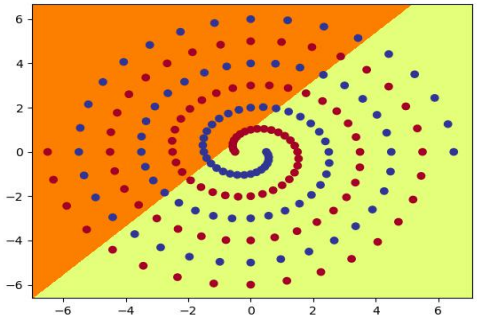
raw1-7.png:



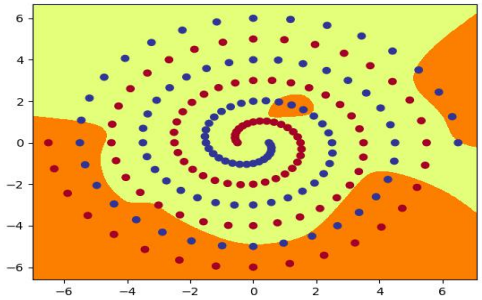
raw1-8.png:



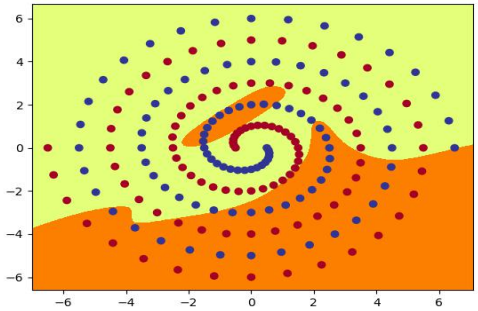
raw1-9.png:



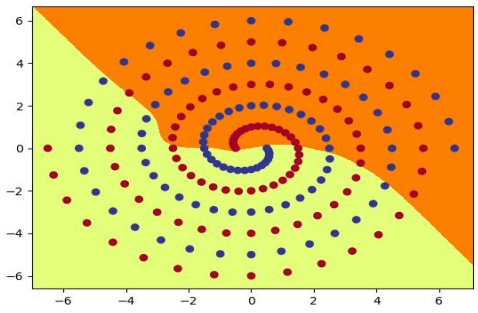
raw2-0.png:



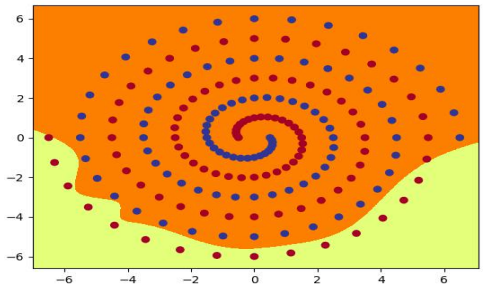
raw2-1.png:



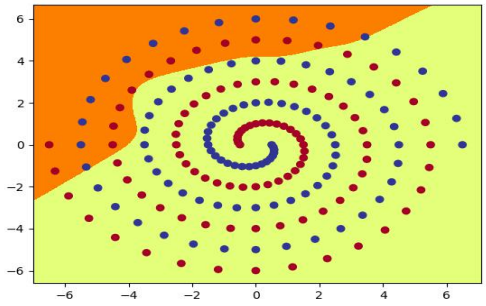
raw2-2.png:



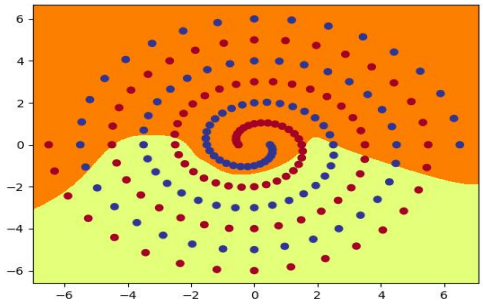
raw2-3.png:



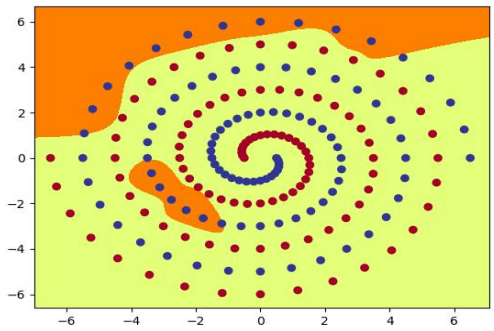
raw2-4.png:



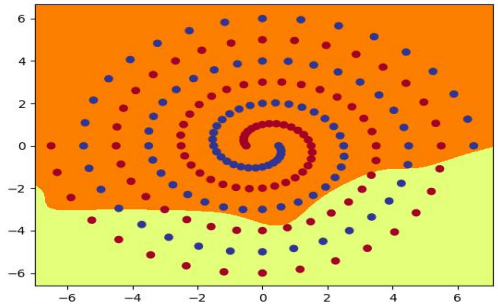
raw2-5.png:



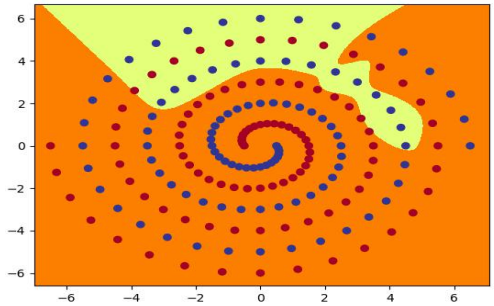
raw2-6.png:



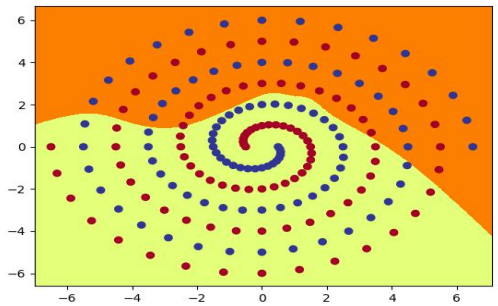
raw2-7.png:



raw2-8.png:

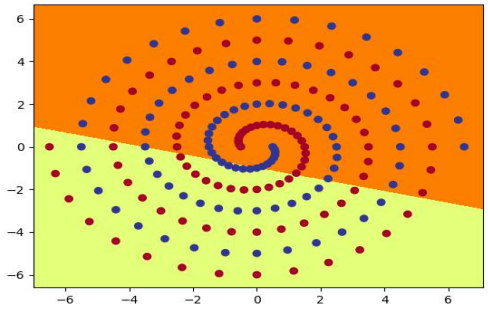


raw2-9.png:

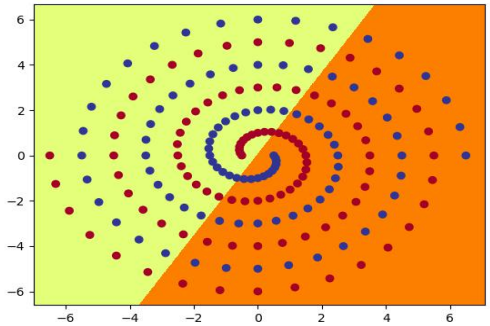


(3) ShortNet (when initial weight is 0.12 and hidden node number is 8)

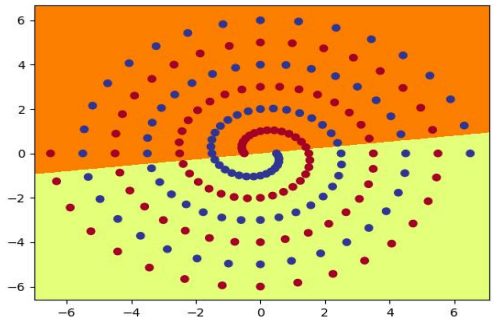
short1_0.png:



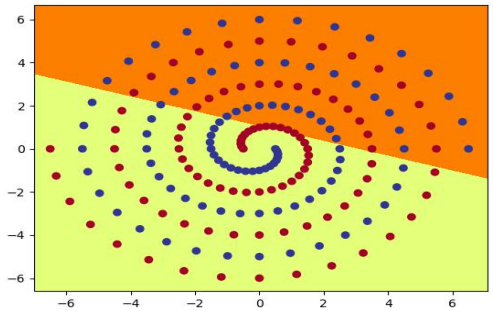
short1_1.png:



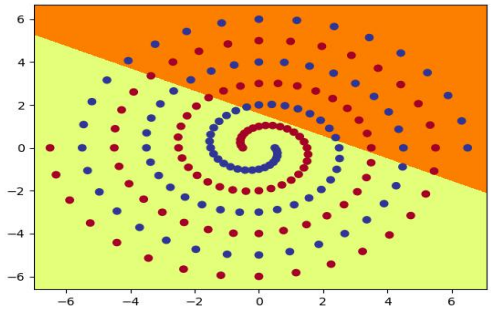
short1_2.png:



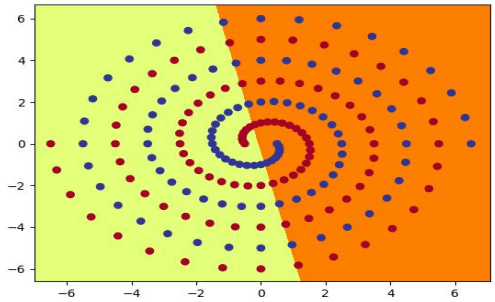
short1_3.png:



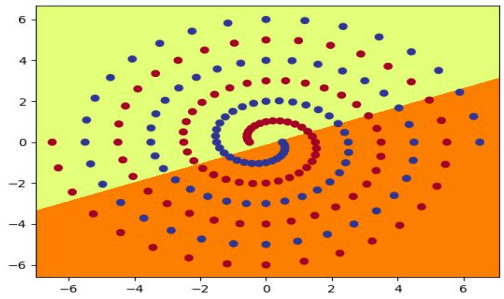
short1_4.png:



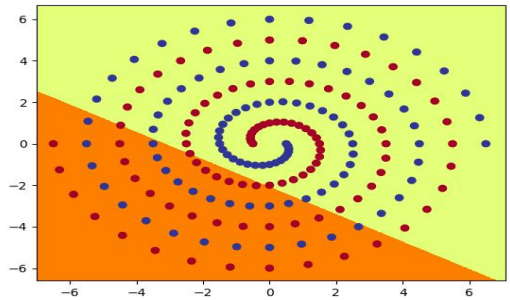
short1_5.png:



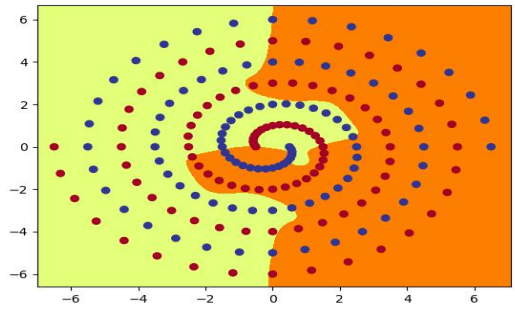
short1_6.png:



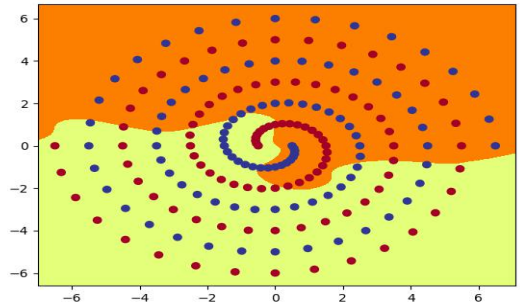
short1_7.png:



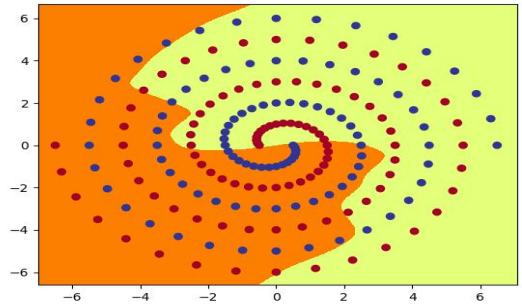
short2_0.png:



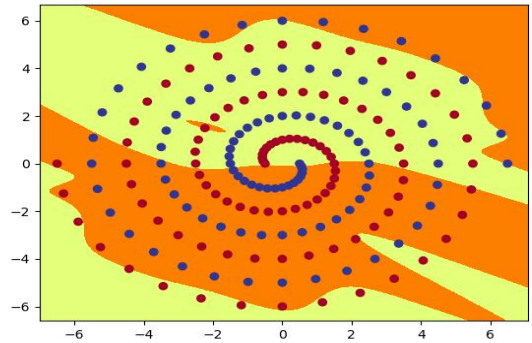
short2_1.png:



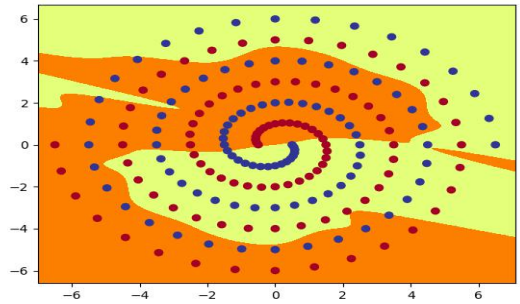
short2_2.png:



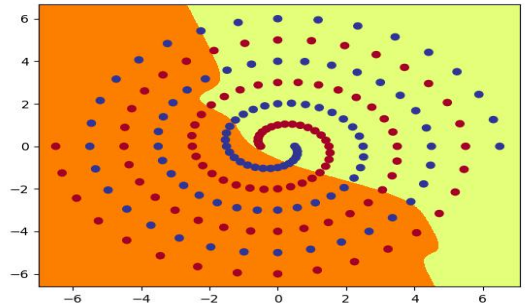
short2_3.png:



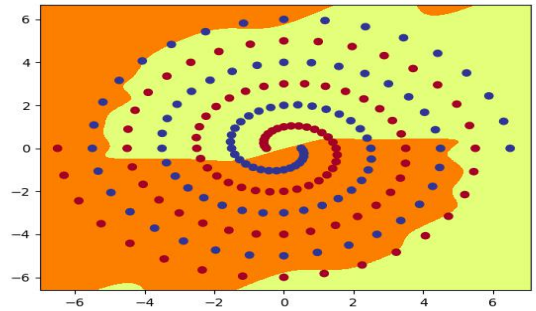
short2_4.png:



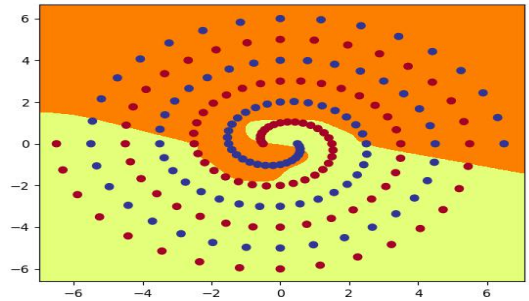
short2_5.png:



short2_6.png:



short2_7.png:



8.

a. For PolarNet, function of each hidden node is non-linear. For RawNet, first hidden layer of nodes give linear function decision boundary and the second hidden layer nodes learn non-linear function. The situation of ShortNet hidden nodes is similar as that of RawNet. No matter what

network is and no matter what kind of function hidden nodes learn, every hidden node can only learn a part of correct decision boundary. A weighted sum then applied to make the network learn the complete classification boundary.

b. I tried different values of initial weights in [0.001, 0.01, 0.1, 0.2, 0.3] for RawNet and ShortNet and found the most possible value for success is 0.1 and 0.2, too small (0.01) and too large (0.3) value could sometimes cause failure, and value 0.001 never success in my experiment. As for the speed of learning, it is not always true that the larger initial weight is, the faster (less epochs) it can learn. For example, when initial weight is 0.1, RawNet takes 6200 epochs to coverage to 100% but when initial weight is 0.2, it takes 13000 epochs to converge from my experiment result.

c. The classification boundary made by PolarNet is the most organized and smooth among the 3 networks. Although the output function of other 2 networks get 100% correct classification in the end, they are weaker than that of PolarNet in terms of generalization. The classification boundary of PolarNet is most close to that made by humans while others not. The activation of output nodes of PolarNet resemble the spiral pattern very much. I think this mainly because (x,y) is converted to polar counterparts in PolarNet only, which is a more suitable data representation method compared with no conversion. With this, even though the structure of PolarNet is simpler than other 2 networks, it can achieve good performance as well. So, the suitable representation for data is very important in deep learning tasks.

d. (1) I changed batch size value from 97 to 194 and keep other parameters unchanged. I run PolarNet and found the epochs needed reduced a lot from 4000 to 900. When I double it to 388, it needs 800 epochs, almost the same as batch size 194. I even tried larger value (485) and found it still needs 800 epochs to converge. So, I think in a certain range, increasing the batch size can accelerate learning.

(2) I used SGD instead of Adam for the optimizer and keep other parameters unchanged. I run RawNet with initial weight 0.15 and found it cannot converge to 100% within 20000 epochs. When using SGD, I also noticed that the accuracy do increase but increase slowly than using Adam. In fact, Adam can be viewed as a combination of RMSprop and SGD with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

(3) I changed tanh activation function to relu and keep other parameters unchanged. I run ShortNet with initial weight 0.15 and found when using relu activation function, the accuracy cannot converge to 100% within 20000 epochs and it fluctuate between 80% and 85%. I also tried the same thing in RawNet and got similar result. This maybe because relu function will make some neurons output as 0 and these neurons can never be activated by further layers. In our problem, it is a training process, relu function may cause under-fitting. So, it's not as good as tanh here.