THE UNIVERSITY OF NEW SOUTH WALES

COMP 9444
Neural Networks and Deep Learning

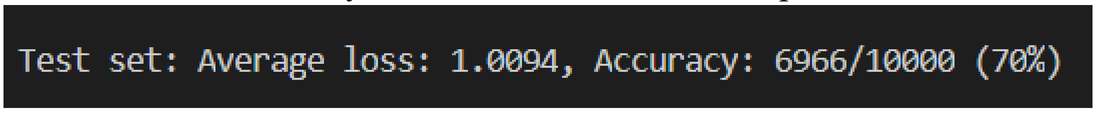
Assignment 1

Student Name	Hongyi Luo
Student Number	z5241868

Part1

Question1

Based on the question1, we should use the linear function and log softmax function to design the neural network. After running the source code, we can finally get the accuracy, which is 6966/10000. After doing the calculation, the accuracy is around 70%. Here is the picture of the result.



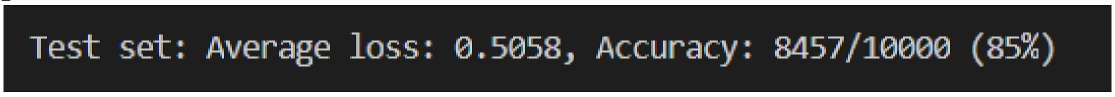
```
Test set: Average loss: 1.0094, Accuracy: 6966/10000 (70%)
```

Furthermore, we can also get the confusion matrix as showing below. Here is the result of the confusion matrix.

```
[[766.   5.   8.  13.  30.  64.   3.  62.  30.  19.]
 [  7. 671. 107.  18.  27.  23.  58.  13.  27.  49.]
 [  5.  58. 691.  27.  28.  21.  47.  37.  46.  40.]
 [  4.  33.  61. 759.  14.  58.  15.  19.  25.  12.]
 [ 58.  50.  82.  22. 626.  19.  32.  37.  20.  54.]
 [  8.  28. 128.  17.  20. 722.  26.   8.  33.  10.]
 [  4.  24. 147.  11.  24.  24. 722.  21.  10.  13.]
 [ 17.  29.  29.  11.  83.  16.  52. 624.  91.  48.]
 [  9.  35.  95.  41.   8.  31.  46.   7. 707.  21.]
 [  8.  50.  87.   2.  53.  31.  17.  33.  41. 678.]]
```

Question2

Based on the question2, we should use the linear function, tanh function and log softmax function to design a neural network with 2 layers. In this case, we can use a function called nn.sequential to wrap these functions up. This can make the designing process easier. After running the source code, we can easily find the accuracy is 85%, which is 8457/10000. Here is the picture of the result.



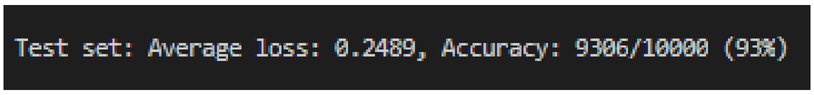
```
Test set: Average loss: 0.5058, Accuracy: 8457/10000 (85%)
```

Furthermore, we can also get the confusion matrix as showing below. Here is the result of the confusion matrix.

```
[[847.   7.   3.   6.  31.  34.   4.  39.  24.   5.]
 [  4. 824.  27.   2.  17.  15.  59.   7.  17.  28.]
 [  7.  16. 826.  50.  12.  19.  27.  12.  14.  17.]
 [  5.  12.  25. 919.   2.  18.   7.   2.   4.   6.]
 [ 35.  34.  21.   4. 818.  10.  27.  18.  19.  14.]
 [  9.  24.  76.   8.  10. 832.  18.   1.  16.   6.]
 [  3.  16.  42.   9.  13.   7. 891.   8.   1.  10.]
 [ 22.  13.  17.   3.  16.   9.  30. 841.  22.  27.]
 [ 12.  31.  29.  60.   4.   9.  30.   3. 816.   6.]
 [  4.  21.  44.   3.  32.   5.  19.  20.   9. 843.]]
```

Question3

In question3, we should design a 2-layers neural network by using the convolution function provided by nn. Based on the question2, we can also use the Sequential function to wrap up 2 convolution models. Here is the final accuracy of the question3, which is 9306/10000, which is 93%. Here is the picture of the result.



```
Test set: Average loss: 0.2489, Accuracy: 9306/10000 (93%)
```

Also, here is the result of the confusion matrix.

```
[[957.   2.   2.   0.  16.   8.   0.  12.   1.   2.]
 [  2. 910.   5.   0.  12.   2.  48.   6.   3.  12.]
 [  9.   6. 850.  43.   9.  19.  36.  15.   4.   9.]
 [  2.   1.  17. 954.   3.  12.   3.   4.   1.   3.]
 [ 25.   2.   3.   9. 911.   5.  18.   9.  14.   4.]
 [  3.  17.  30.   3.   5. 916.  19.   4.   2.   1.]
 [  5.   4.  11.   1.   7.   3. 964.   1.   2.   2.]
 [  8.   2.   1.   1.   8.   4.  12. 936.   8.  20.]
 [  7.  11.   3.   4.   4.   6.   9.   2. 953.   1.]
 [  5.   9.   2.   4.  12.   4.   0.   4.   5. 955.]]
```

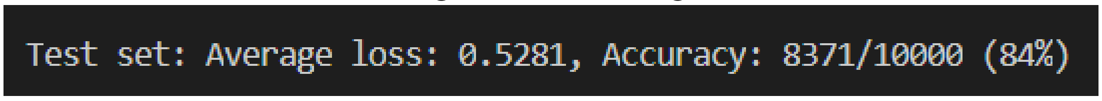
Question4

a. From the previous question, we can get 3 different models, which have their accuracy and confusion matrix. From the first question, we only use

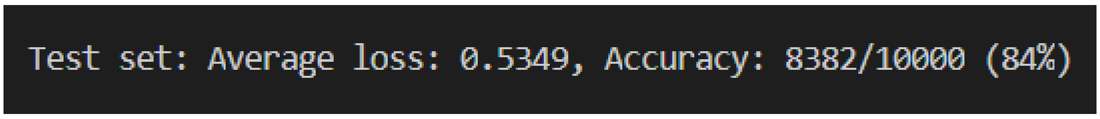
the linear function and softmax method. The accuracy is lower, which is around 70%.
In the second question, we still use the linear function. The difference is we increase the layer of the neural network, which is using the 2-layer linear function instead of using only one. Furthermore, we add an activation function which is Tanh in this model. The result is better than the first one, which accuracy is 85% (larger than the first question about 15%). This means we can add more layers and using the activation function to improve the accuracy of the training model.
In question3, we use the convolutional neural network, which is different from the linear function. Furthermore, using the ReLU activation function can help us avoid the appearance of the gradient vanishing problem. This activation function can also avoid the overfitting problem. The result of the third question can easily approve that using the convolutional neural network will give a good performance in image identification than using the linear function. The accuracy is 93% which is larger than the 2-layer linear function, which is larger 8%.

b. From the question1, we can get the information that the rows of the confusion matrix are the target character. Also, the columns indicate the one chosen by the network. The diagonal of the matrix is the number of the correct training samples of each character.
From the confusion matrix given by the question1, we can easily get the number of correct training sample is 624 and 626, which represent the "na" and "ya". This means the correct number of predictions will affect accuracy. Furthermore, the character "ki" which the number of correct sample is 671, performs a little bit worse but not the worst. Therefore, based on the confusion matrix in question1, character "na" and "ya" are most likely to be mistaken by other characters.
Using the same method, we can also find that character "na" and "re" in the question2(2-layer linear neural network) confusion matrix is most

likely to be mistaken.
From the question3 confusion matrix, we can also get that the character "na" is most likely to be mistaken.

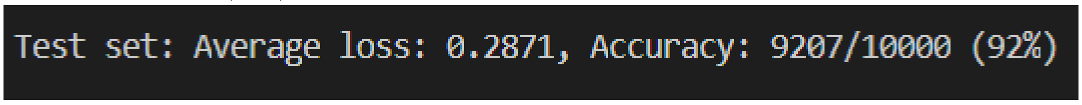
c. In the first part questions, I do several changes. In the question2, I changed the number of hidden nodes.
I set the number as 100, and get the following result as the show blew.


```
Test set: Average loss: 0.5281, Accuracy: 8371/10000 (84%)
```

I set the number as 10000, and get the following result as the show blew.


```
Test set: Average loss: 0.5349, Accuracy: 8382/10000 (84%)
```

The accuracy is 84%, but when I use 1000 hidden nodes the accuracy is 85%. This means the hidden node number can affect the accuracy. Furthermore, this can indicate that the more hidden node we use, we may not get higher accuracy.

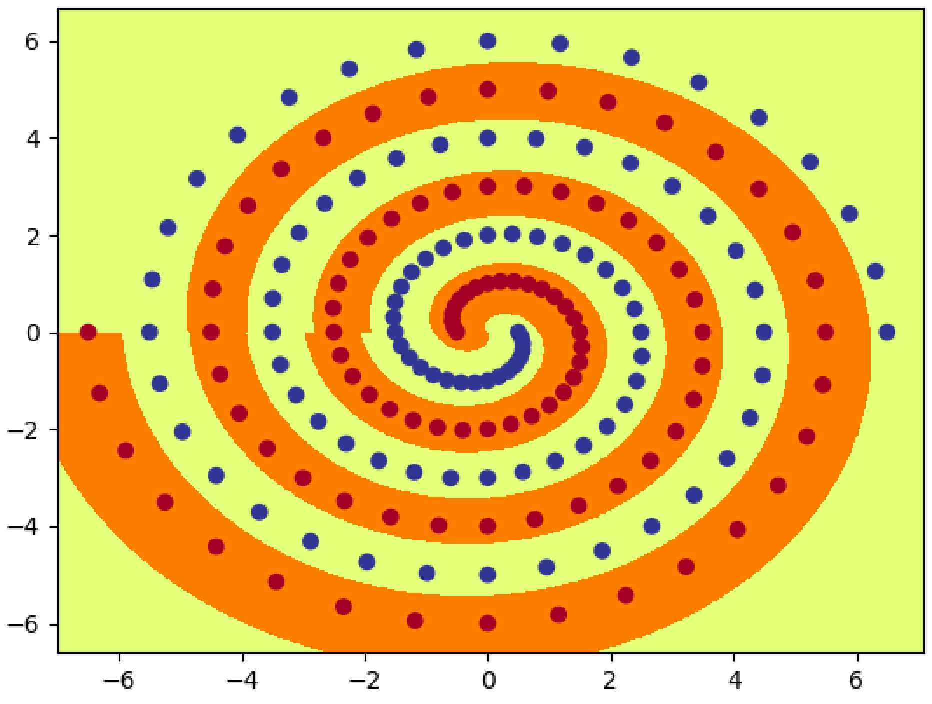
The most interesting point is in the question3. When I setting the parameter in the max pool function, I set the parameter as (2,2). After training, accuracy is always lower than 93%. Then changing the parameters to (3,2). The accuracy is 93%. This is because when using CNN to deal with the image. The size would not enough to get the information about the picture. When changing this parameter to (3,2), then doing the convolutional calculation, we can easily find that there is an overlap area. This can help us to improve the efficiency of getting details from the image by using CNN. Therefore, changing the size of the max pool can solve the problem of lower accuracy. In this way, we can get 93% accuracy. Here is the result of MaxPool2d(2,2).


```
Test set: Average loss: 0.2871, Accuracy: 9207/10000 (92%)
```

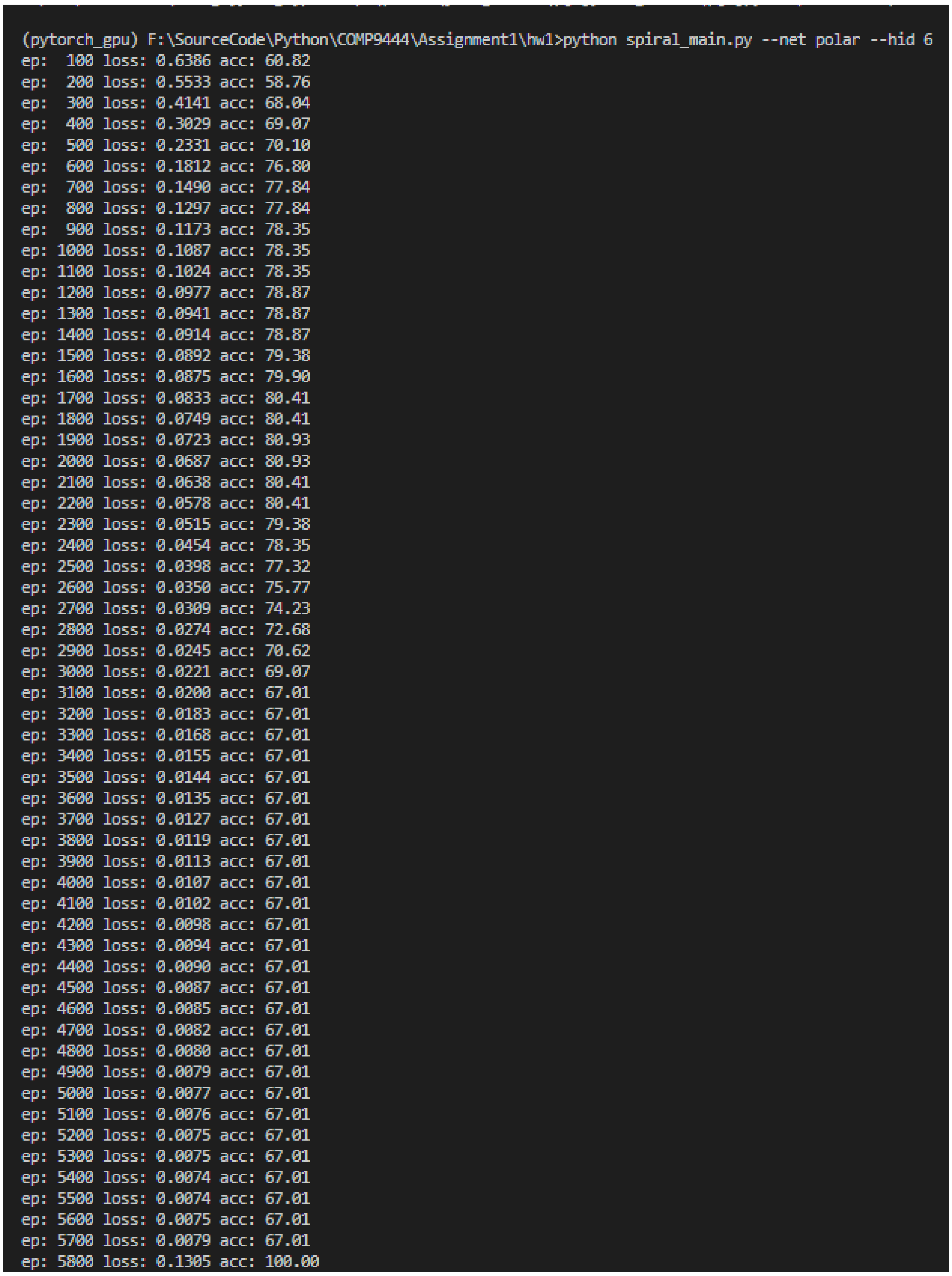
Furthermore, in the part1, we can also get the result that increasing the number of channels can also speed up training and increase accuracy.

Part2

Question2

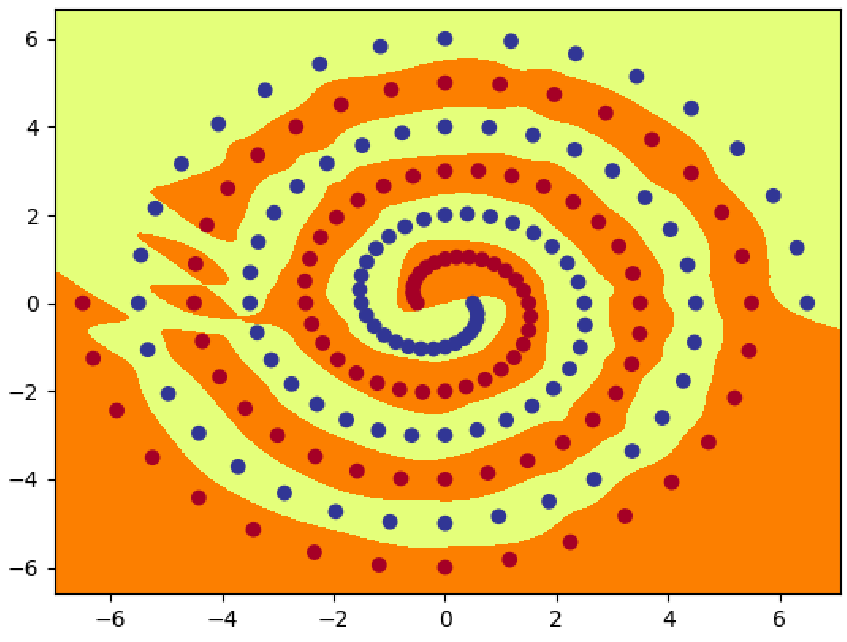


Here is the picture result of the question2. From this picture, we can find the process of convergence. The minimum number of hidden nodes 6. Here is the result picture.



```
(pytorch_gpu) F:\SourceCode\Python\COMP9444\Assignment1\hw1>python spiral_main.py --net polar --hid 6
ep:  100 loss: 0.6386 acc: 60.82
ep:  200 loss: 0.5533 acc: 58.76
ep:  300 loss: 0.4141 acc: 68.04
ep:  400 loss: 0.3029 acc: 69.07
ep:  500 loss: 0.2331 acc: 70.10
ep:  600 loss: 0.1812 acc: 76.80
ep:  700 loss: 0.1490 acc: 77.84
ep:  800 loss: 0.1297 acc: 77.84
ep:  900 loss: 0.1173 acc: 78.35
ep: 1000 loss: 0.1087 acc: 78.35
ep: 1100 loss: 0.1024 acc: 78.35
ep: 1200 loss: 0.0977 acc: 78.87
ep: 1300 loss: 0.0941 acc: 78.87
ep: 1400 loss: 0.0914 acc: 78.87
ep: 1500 loss: 0.0892 acc: 79.38
ep: 1600 loss: 0.0875 acc: 79.90
ep: 1700 loss: 0.0833 acc: 80.41
ep: 1800 loss: 0.0749 acc: 80.41
ep: 1900 loss: 0.0723 acc: 80.93
ep: 2000 loss: 0.0687 acc: 80.93
ep: 2100 loss: 0.0638 acc: 80.41
ep: 2200 loss: 0.0578 acc: 80.41
ep: 2300 loss: 0.0515 acc: 79.38
ep: 2400 loss: 0.0454 acc: 78.35
ep: 2500 loss: 0.0398 acc: 77.32
ep: 2600 loss: 0.0350 acc: 75.77
ep: 2700 loss: 0.0309 acc: 74.23
ep: 2800 loss: 0.0274 acc: 72.68
ep: 2900 loss: 0.0245 acc: 70.62
ep: 3000 loss: 0.0221 acc: 69.07
ep: 3100 loss: 0.0200 acc: 67.01
ep: 3200 loss: 0.0183 acc: 67.01
ep: 3300 loss: 0.0168 acc: 67.01
ep: 3400 loss: 0.0155 acc: 67.01
ep: 3500 loss: 0.0144 acc: 67.01
ep: 3600 loss: 0.0135 acc: 67.01
ep: 3700 loss: 0.0127 acc: 67.01
ep: 3800 loss: 0.0119 acc: 67.01
ep: 3900 loss: 0.0113 acc: 67.01
ep: 4000 loss: 0.0107 acc: 67.01
ep: 4100 loss: 0.0102 acc: 67.01
ep: 4200 loss: 0.0098 acc: 67.01
ep: 4300 loss: 0.0094 acc: 67.01
ep: 4400 loss: 0.0090 acc: 67.01
ep: 4500 loss: 0.0087 acc: 67.01
ep: 4600 loss: 0.0085 acc: 67.01
ep: 4700 loss: 0.0082 acc: 67.01
ep: 4800 loss: 0.0080 acc: 67.01
ep: 4900 loss: 0.0079 acc: 67.01
ep: 5000 loss: 0.0077 acc: 67.01
ep: 5100 loss: 0.0076 acc: 67.01
ep: 5200 loss: 0.0075 acc: 67.01
ep: 5300 loss: 0.0075 acc: 67.01
ep: 5400 loss: 0.0074 acc: 67.01
ep: 5500 loss: 0.0074 acc: 67.01
ep: 5600 loss: 0.0075 acc: 67.01
ep: 5700 loss: 0.0079 acc: 67.01
ep: 5800 loss: 0.1305 acc: 100.00
```

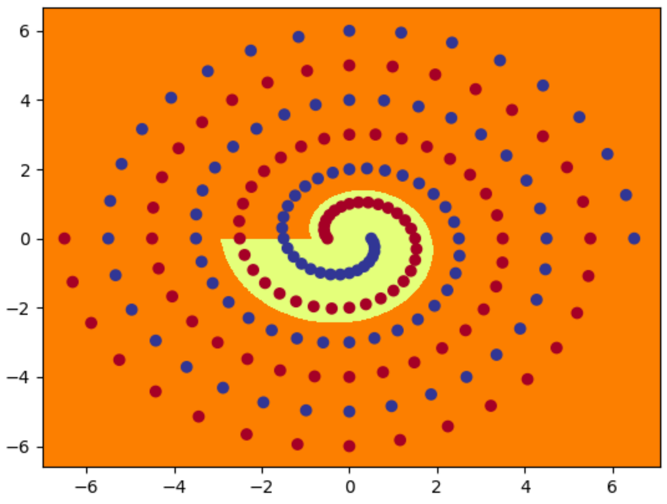
Question4

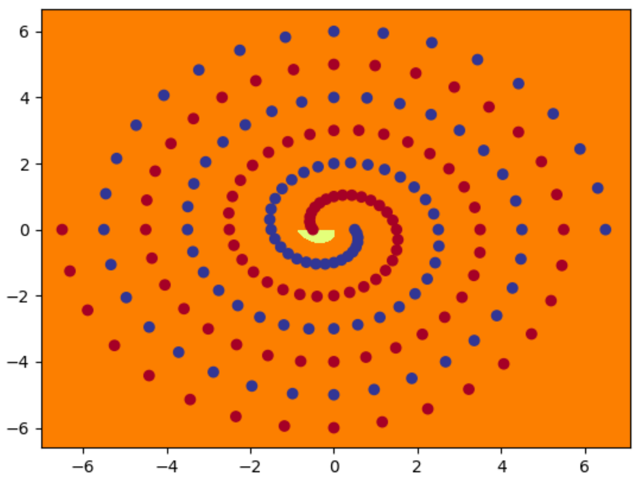


Here is the picture result of the question4. This picture is the convergence process of rawNet. Using the hidden node number as 100 can speed up the training. Also, setting the initial weight as 0.2 or larger can speed up training.

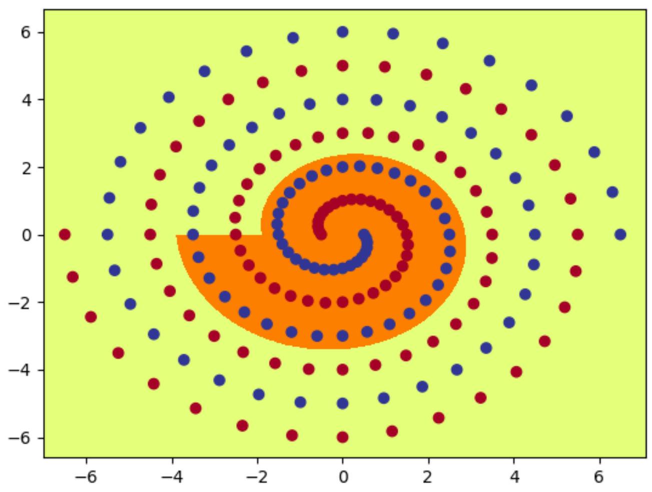
Question5

PolarNet result has 11 pictures.

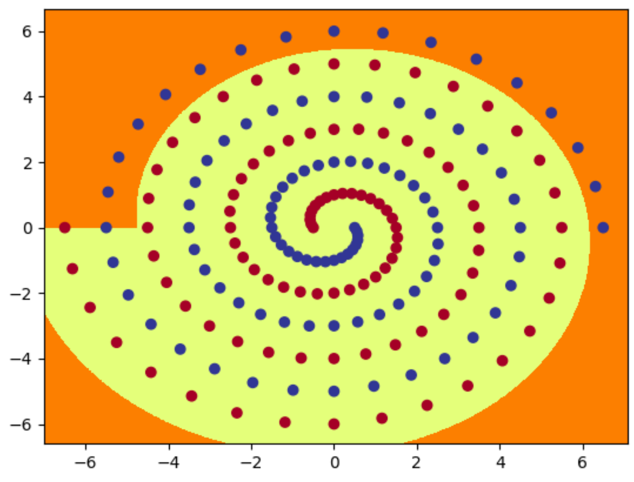




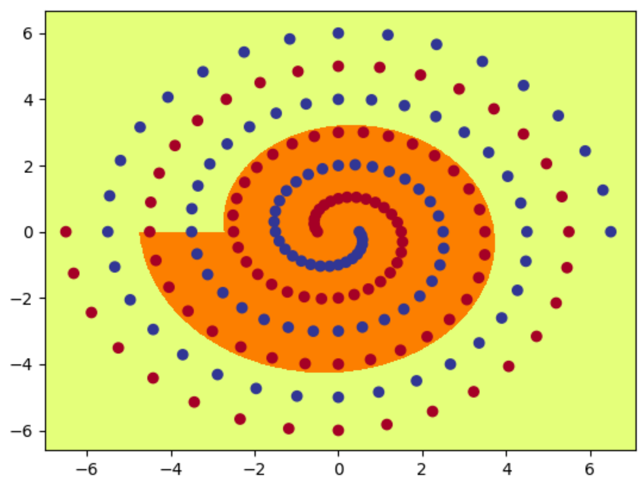
−6 −4 −2 0 2 4 6



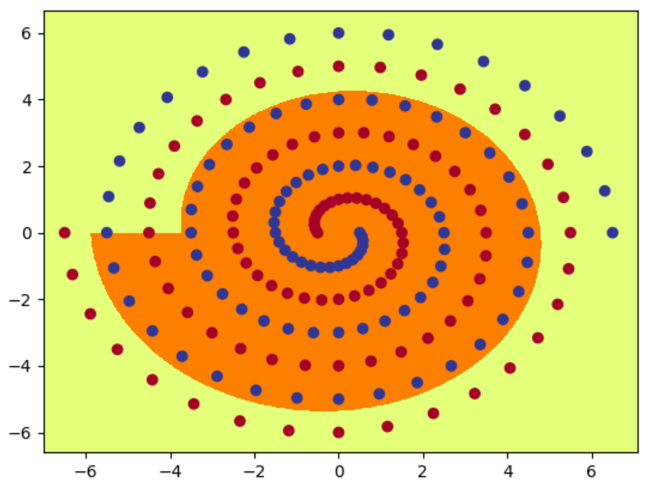
−6 −4 −2 0 2 4 6



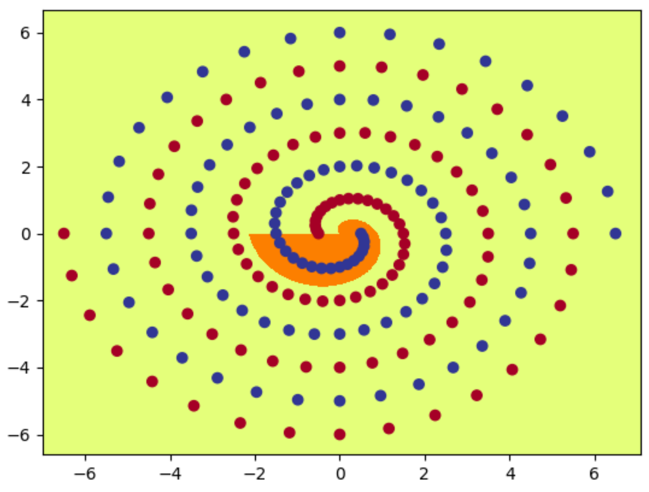
−6 −4 −2 0 2 4 6



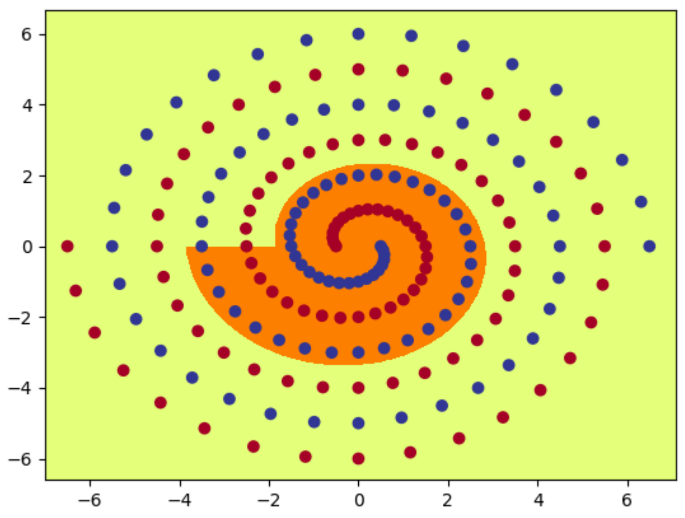
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



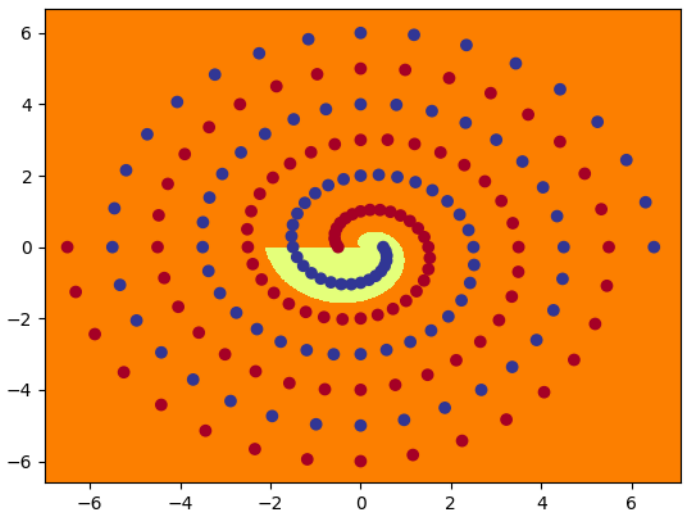
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



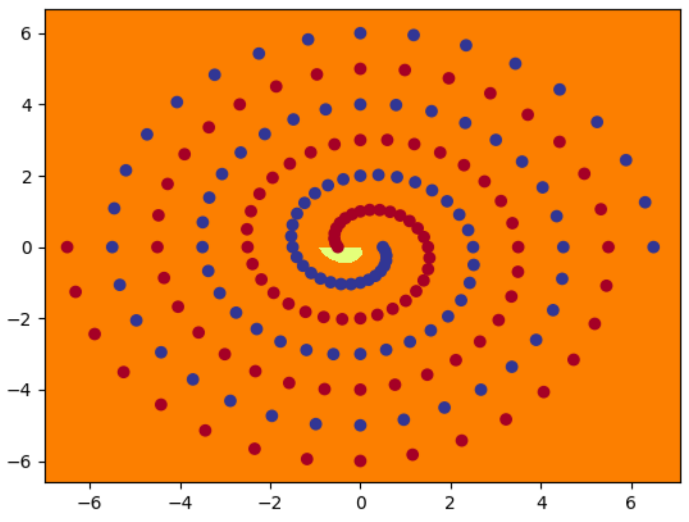
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



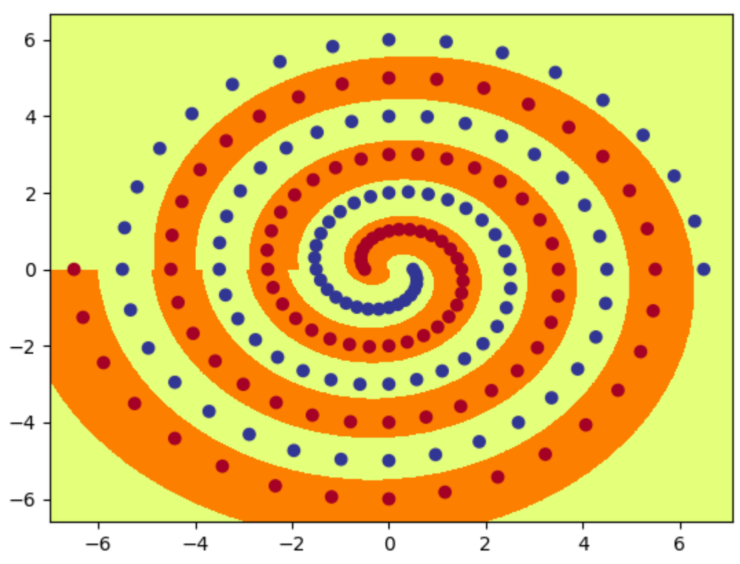
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



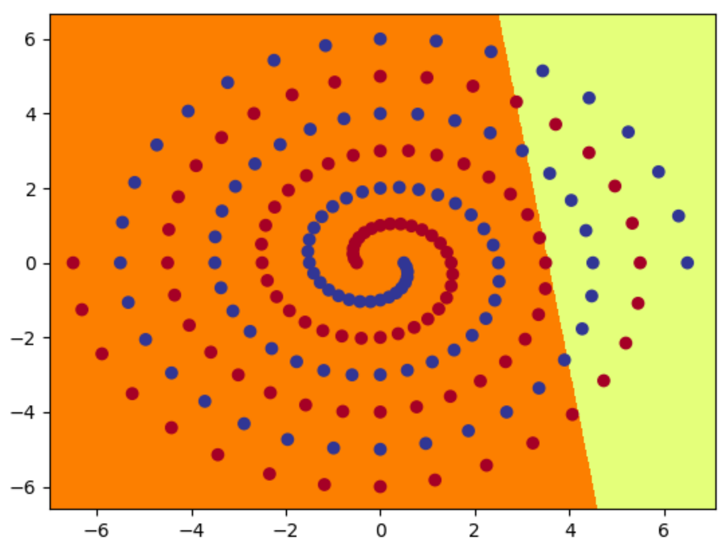
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

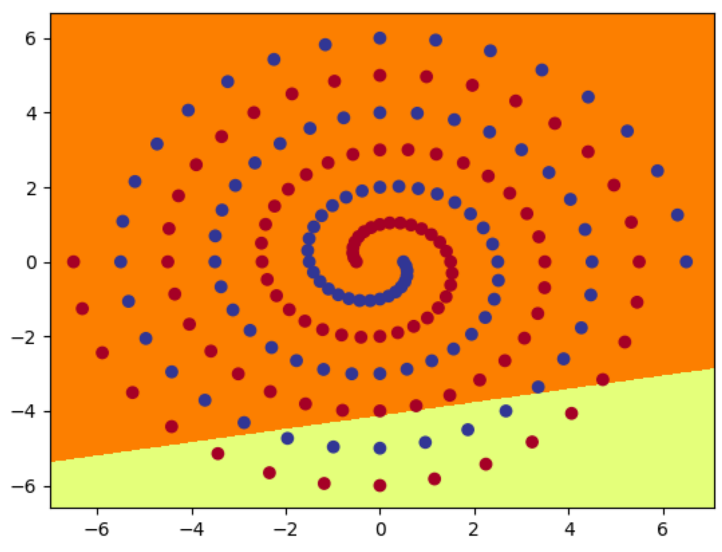


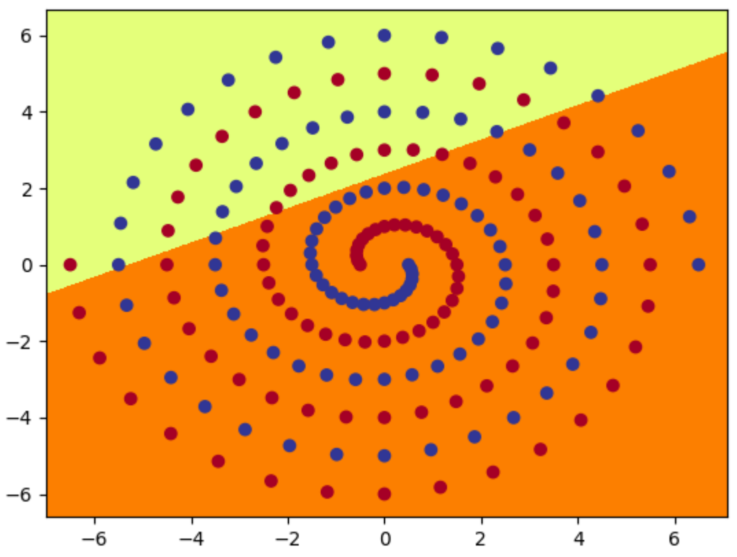
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



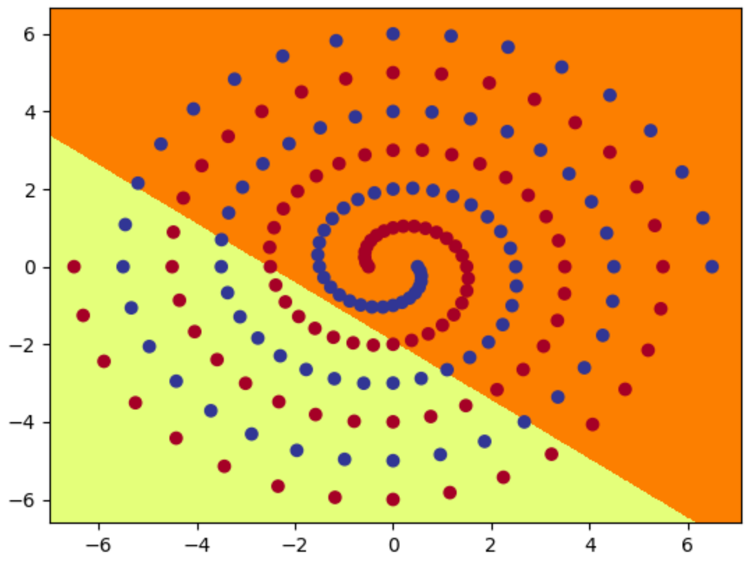
Here are the 21 pictures result of RawNet.



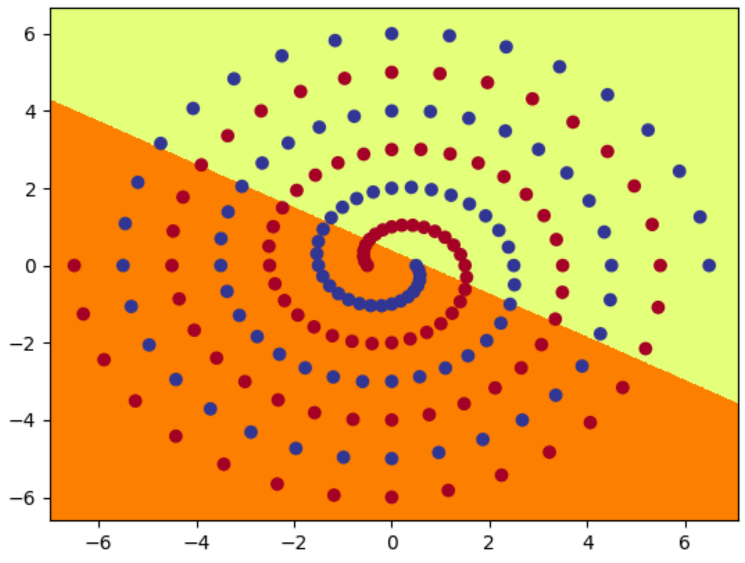




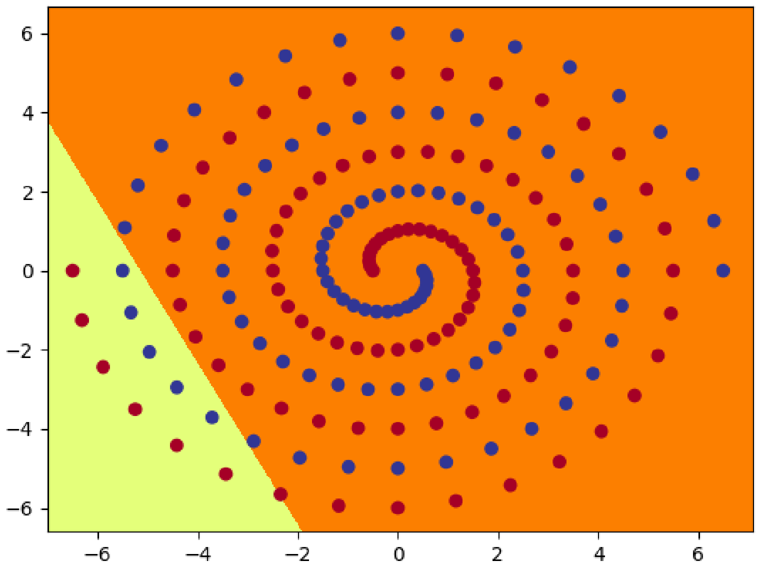
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



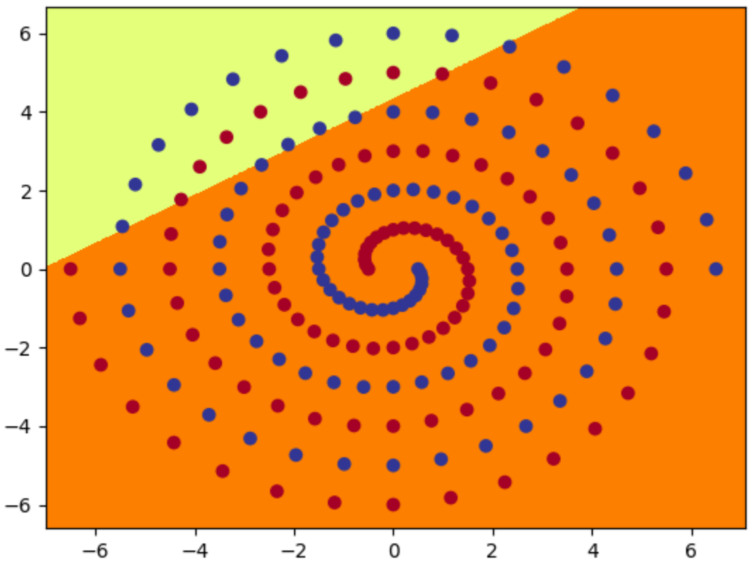
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



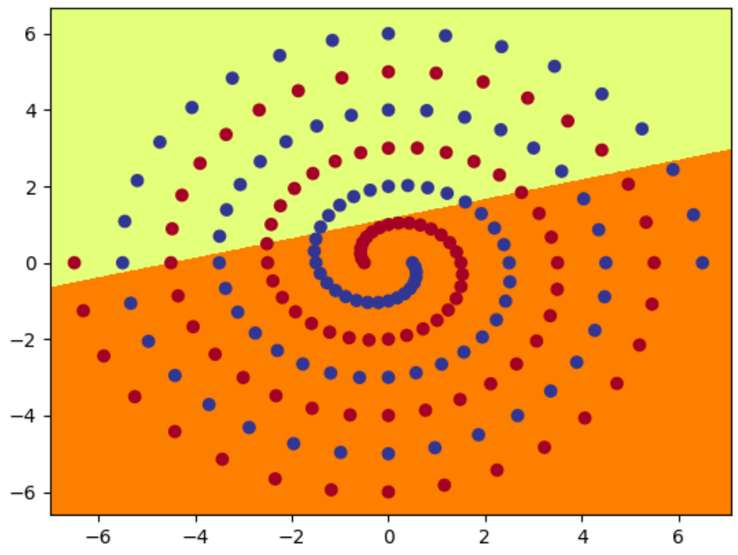
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



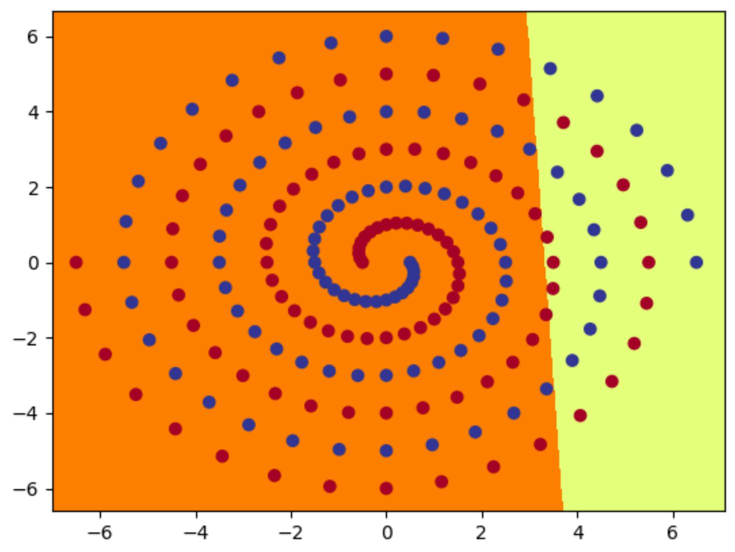
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

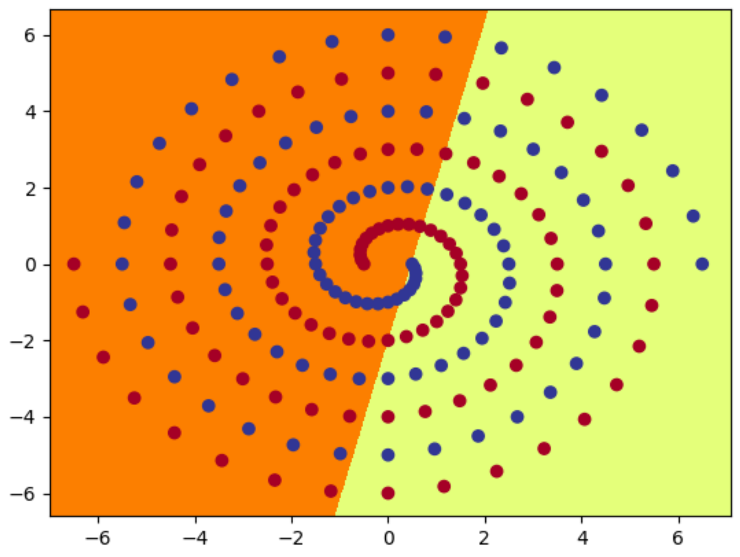


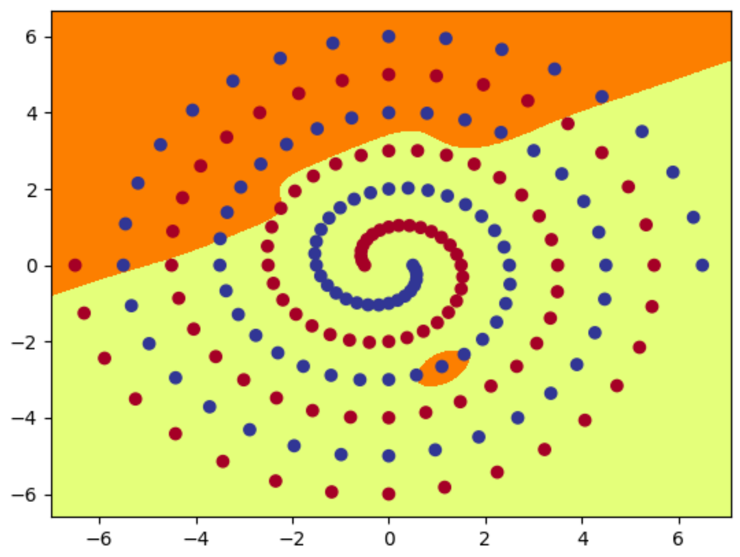
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

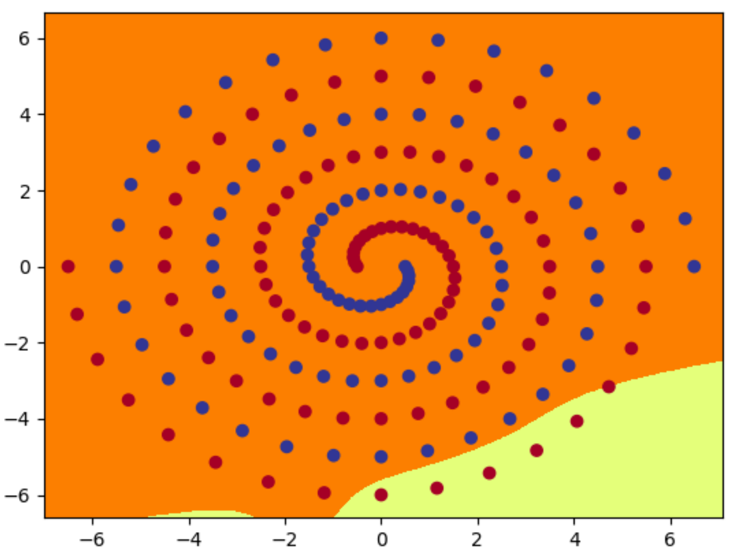


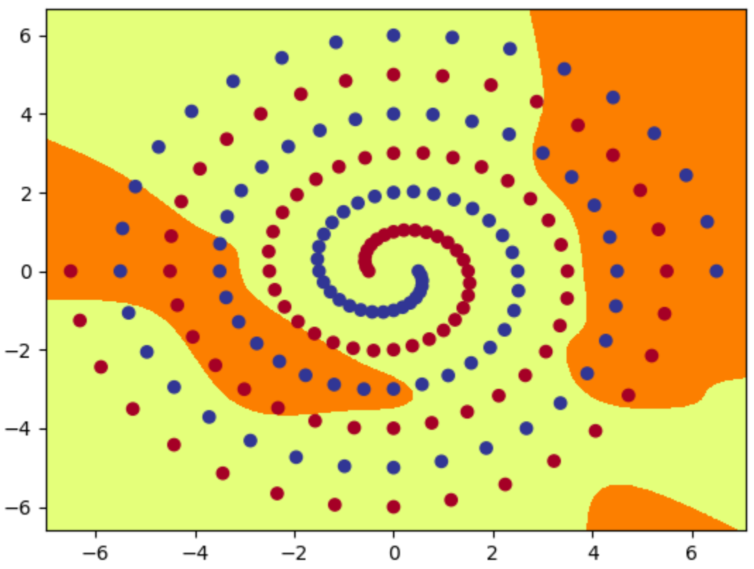
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

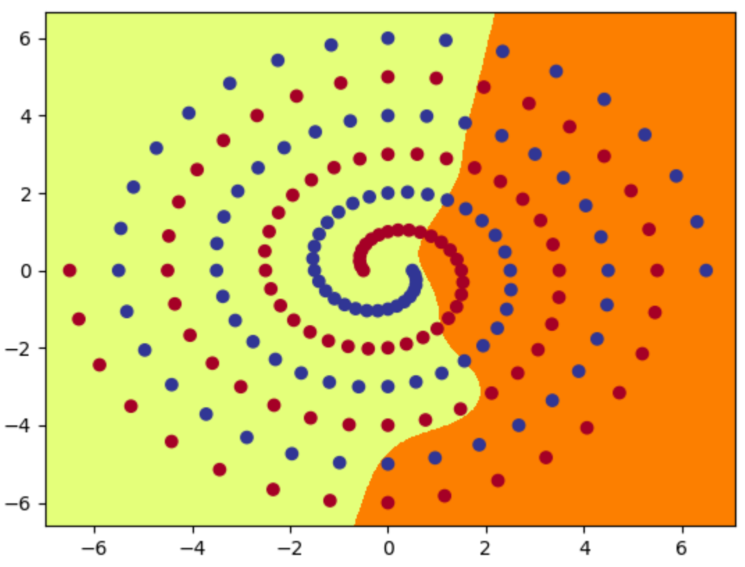

6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

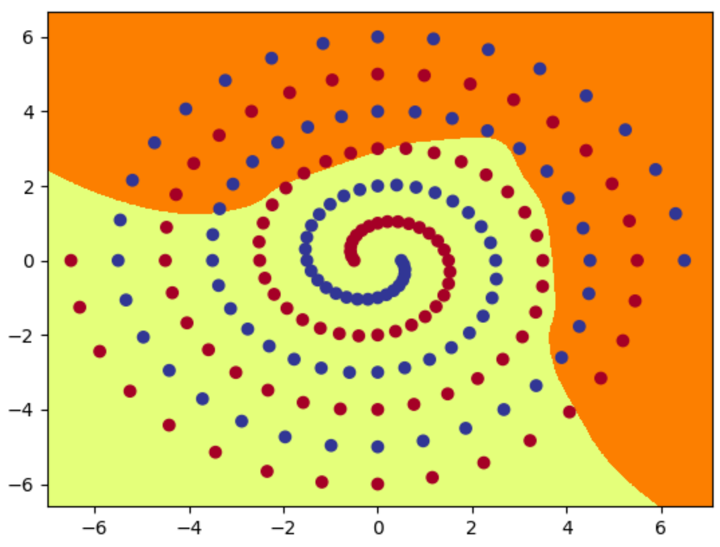

6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

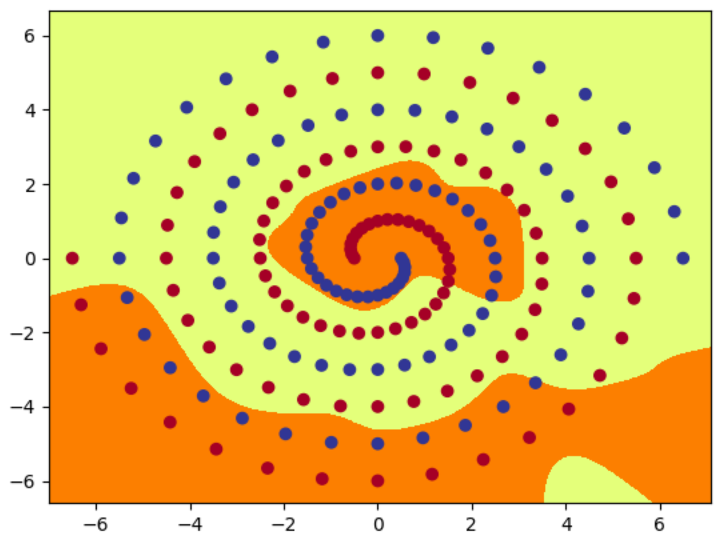




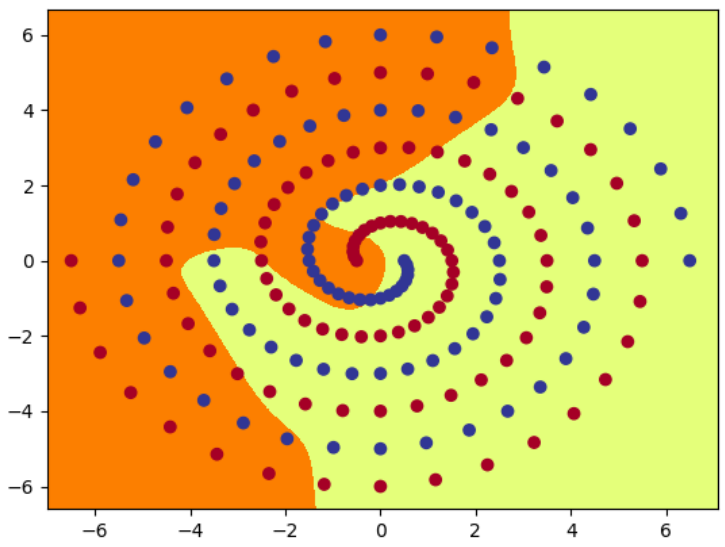




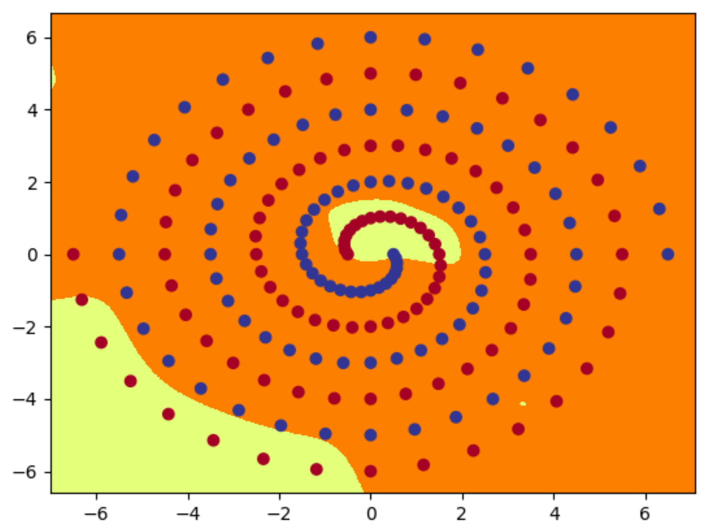
6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6

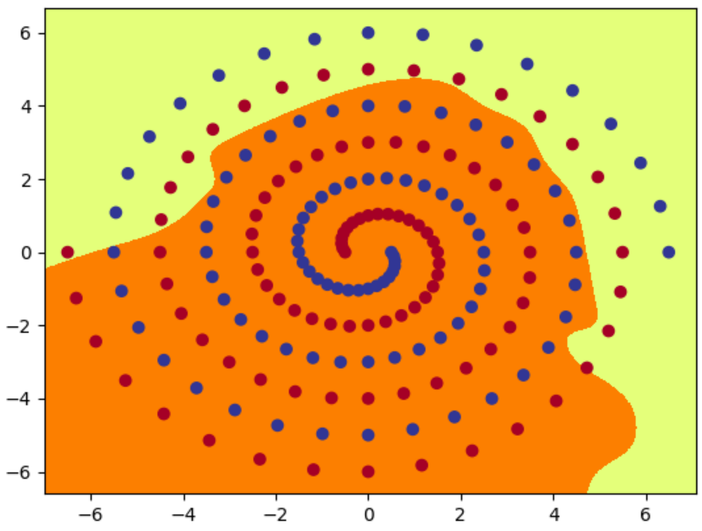


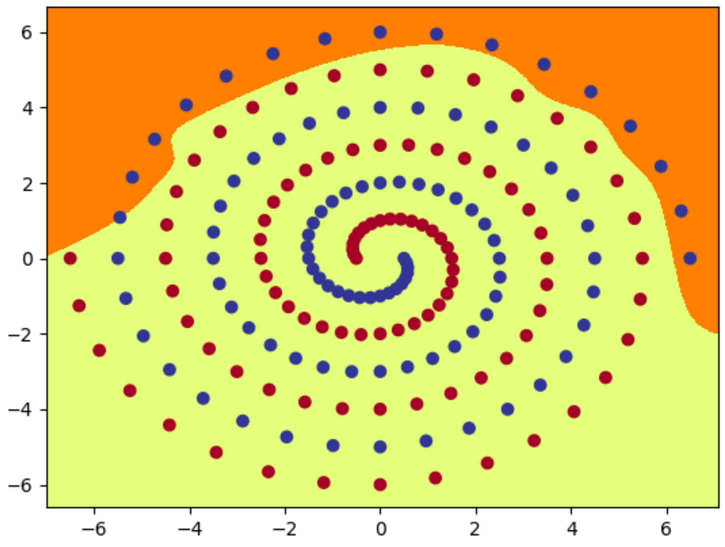
6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6

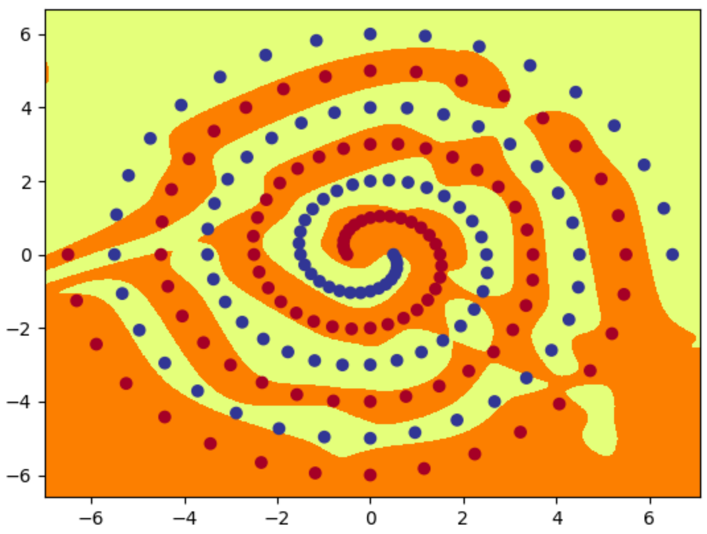


6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6


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



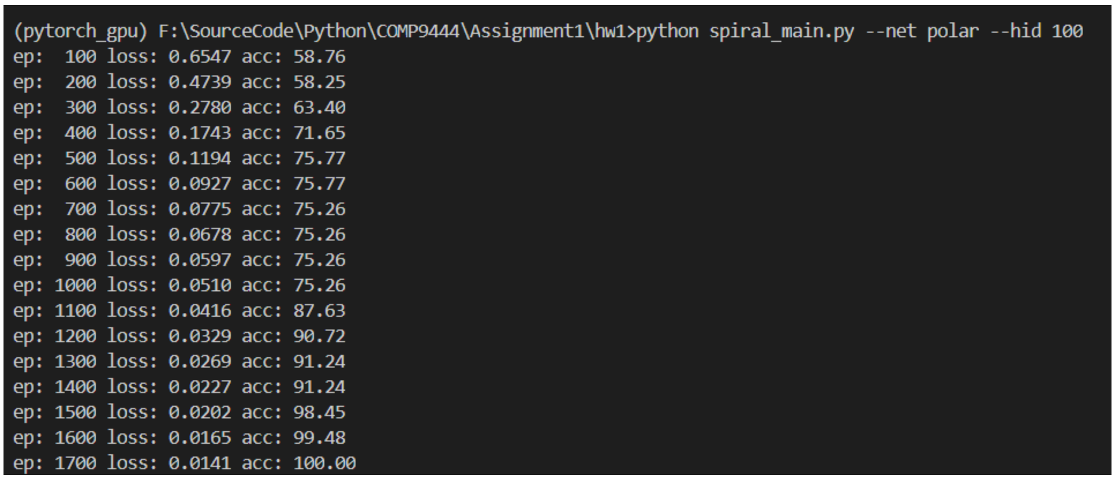
Question6

a. From the picture, we can get the differences between polarNet and rawNet. The first difference is the boundary line. From the picture, we can get that polarNet's boundary is a straight line. The boundary of rawNet is a curve line. The reason why the boundary of rawNet is a curve line is that this method tries to overfitting each point in the graph. In this way, rawNet can finally converge

b. The initial weight will affect the speed of training. For example, in part 2, if we set the hid_num as 10, it may cost a lot of time to convergence. Furthermore, if we set the hid_num as 100 even 1000, it may finish the training process faster than using a small number.

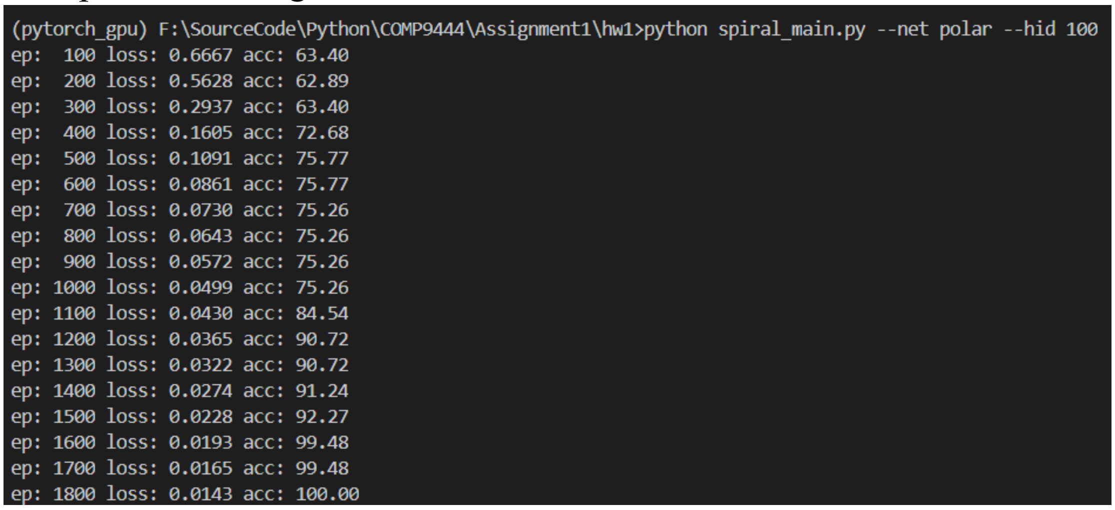
c. In the part2, if I change the activation function to ReLU, the epoch we need has increased. For example, when we change the activation function in polar net to ReLU, the epoch has increased from 1800(the result of using Tanh) to 2000(the result of using ReLU).

This picture is using the Tanh as the activation function.



```
(pytorch_gpu) F:\SourceCode\Python\COMP9444\Assignment1\hw1>python spiral_main.py --net polar --hid 100
ep:  100 loss: 0.6547 acc: 58.76
ep:  200 loss: 0.4739 acc: 58.25
ep:  300 loss: 0.2780 acc: 63.40
ep:  400 loss: 0.1743 acc: 71.65
ep:  500 loss: 0.1194 acc: 75.77
ep:  600 loss: 0.0927 acc: 75.77
ep:  700 loss: 0.0775 acc: 75.26
ep:  800 loss: 0.0678 acc: 75.26
ep:  900 loss: 0.0597 acc: 75.26
ep: 1000 loss: 0.0510 acc: 75.26
ep: 1100 loss: 0.0416 acc: 87.63
ep: 1200 loss: 0.0329 acc: 90.72
ep: 1300 loss: 0.0269 acc: 91.24
ep: 1400 loss: 0.0227 acc: 91.24
ep: 1500 loss: 0.0202 acc: 98.45
ep: 1600 loss: 0.0165 acc: 99.48
ep: 1700 loss: 0.0141 acc: 100.00
```

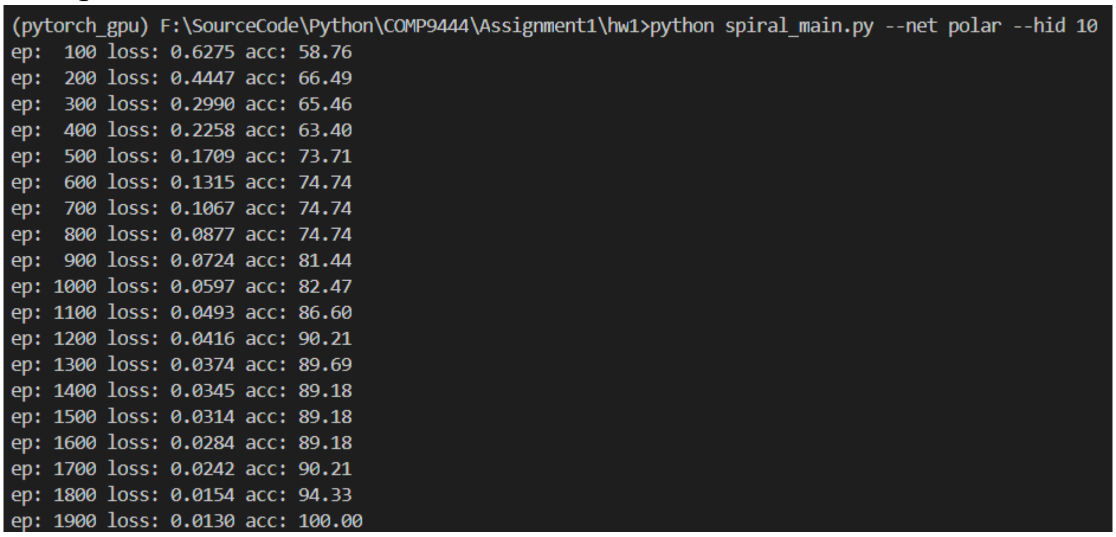
This picture is using the ReLU as the activation function.



```
(pytorch_gpu) F:\SourceCode\Python\COMP9444\Assignment1\hw1>python spiral_main.py --net polar --hid 100
ep:  100 loss: 0.6667 acc: 63.40
ep:  200 loss: 0.5628 acc: 62.89
ep:  300 loss: 0.2937 acc: 63.40
ep:  400 loss: 0.1605 acc: 72.68
ep:  500 loss: 0.1091 acc: 75.77
ep:  600 loss: 0.0861 acc: 75.77
ep:  700 loss: 0.0730 acc: 75.26
ep:  800 loss: 0.0643 acc: 75.26
ep:  900 loss: 0.0572 acc: 75.26
ep: 1000 loss: 0.0499 acc: 75.26
ep: 1100 loss: 0.0430 acc: 84.54
ep: 1200 loss: 0.0365 acc: 90.72
ep: 1300 loss: 0.0322 acc: 90.72
ep: 1400 loss: 0.0274 acc: 91.24
ep: 1500 loss: 0.0228 acc: 92.27
ep: 1600 loss: 0.0193 acc: 99.48
ep: 1700 loss: 0.0165 acc: 99.48
ep: 1800 loss: 0.0143 acc: 100.00
```

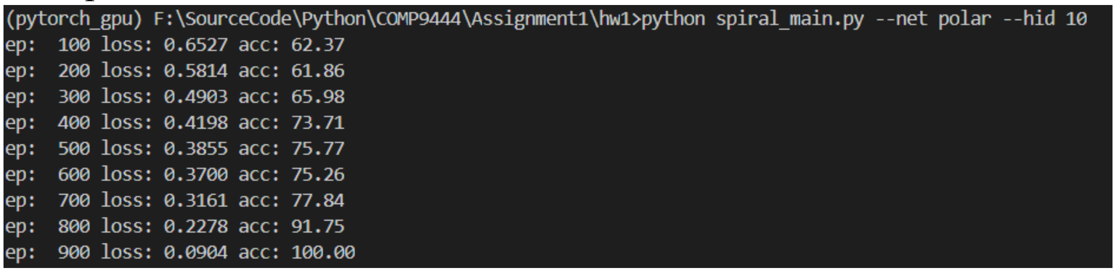
Furthermore, if I change the batch size to 194, the training speed has improved. This process needs 900 epochs. But using the batch size as 97, still needs 1900 epochs. Here is the picture of the result.

This picture is used batch size 97.



```
(pytorch_gpu) F:\SourceCode\Python\COMP9444\Assignment1\hw1>python spiral_main.py --net polar --hid 10
ep:  100 loss: 0.6275 acc: 58.76
ep:  200 loss: 0.4447 acc: 66.49
ep:  300 loss: 0.2990 acc: 65.46
ep:  400 loss: 0.2258 acc: 63.40
ep:  500 loss: 0.1709 acc: 73.71
ep:  600 loss: 0.1315 acc: 74.74
ep:  700 loss: 0.1067 acc: 74.74
ep:  800 loss: 0.0877 acc: 74.74
ep:  900 loss: 0.0724 acc: 81.44
ep: 1000 loss: 0.0597 acc: 82.47
ep: 1100 loss: 0.0493 acc: 86.60
ep: 1200 loss: 0.0416 acc: 90.21
ep: 1300 loss: 0.0374 acc: 89.69
ep: 1400 loss: 0.0345 acc: 89.18
ep: 1500 loss: 0.0314 acc: 89.18
ep: 1600 loss: 0.0284 acc: 89.18
ep: 1700 loss: 0.0242 acc: 90.21
ep: 1800 loss: 0.0154 acc: 94.33
ep: 1900 loss: 0.0130 acc: 100.00
```

This picture is used batch size 194.

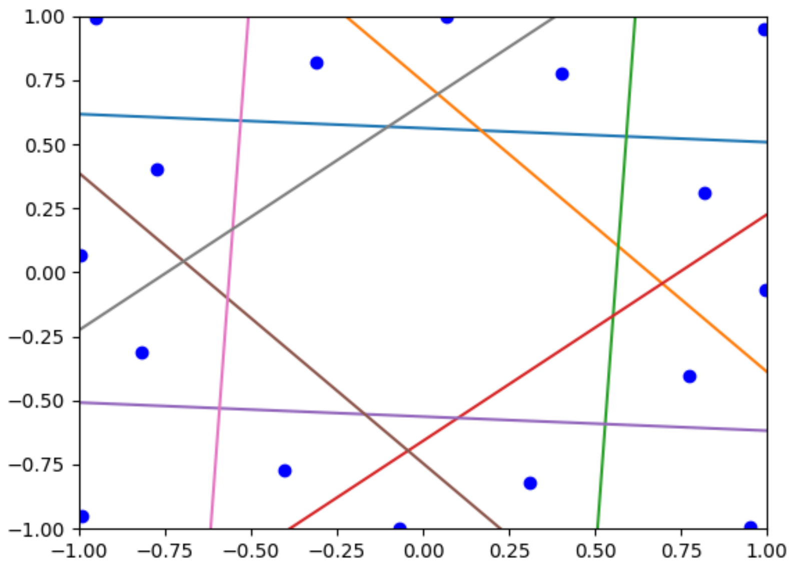


```
(pytorch_gpu) F:\SourceCode\Python\COMP9444\Assignment1\hw1>python spiral_main.py --net polar --hid 10
ep:  100 loss: 0.6527 acc: 62.37
ep:  200 loss: 0.5814 acc: 61.86
ep:  300 loss: 0.4903 acc: 65.98
ep:  400 loss: 0.4198 acc: 73.71
ep:  500 loss: 0.3855 acc: 75.77
ep:  600 loss: 0.3700 acc: 75.26
ep:  700 loss: 0.3161 acc: 77.84
ep:  800 loss: 0.2278 acc: 91.75
ep:  900 loss: 0.0904 acc: 100.00
```

Part3

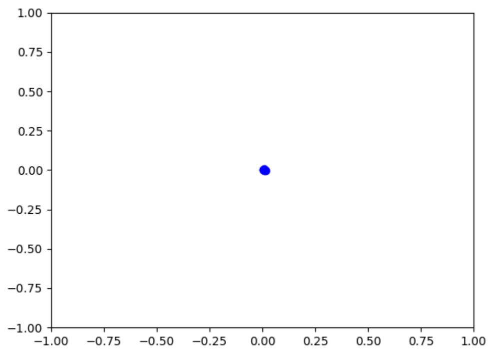
Question1

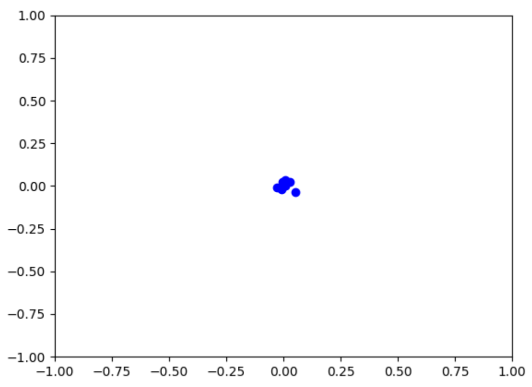
Here is the picture result of running the command in the terminal.

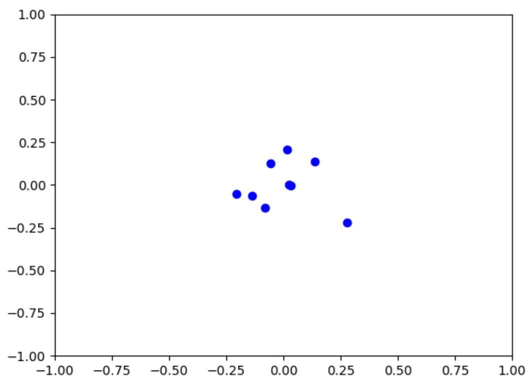


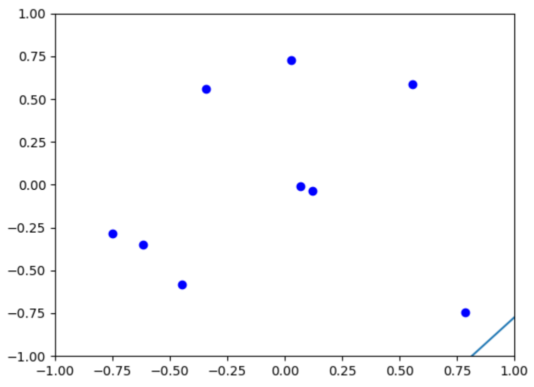
Question2

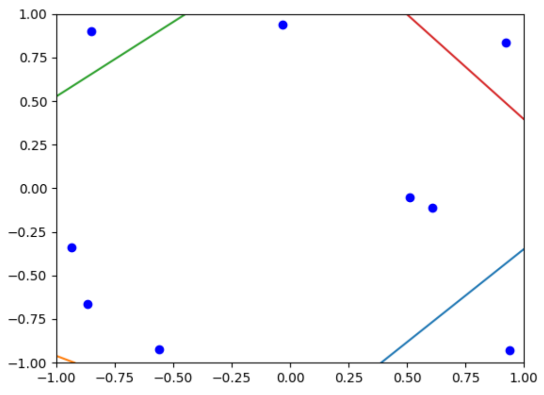
Here are the first 11 pictures from epoch 50 to 3000.

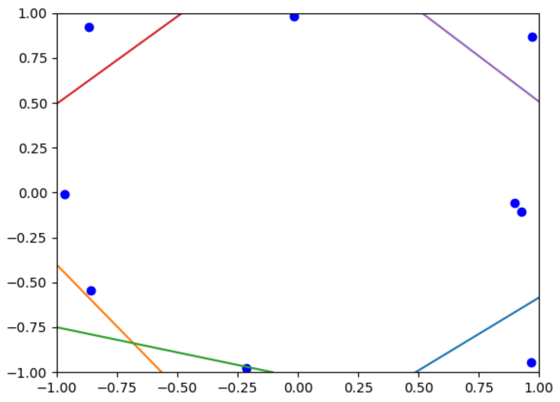


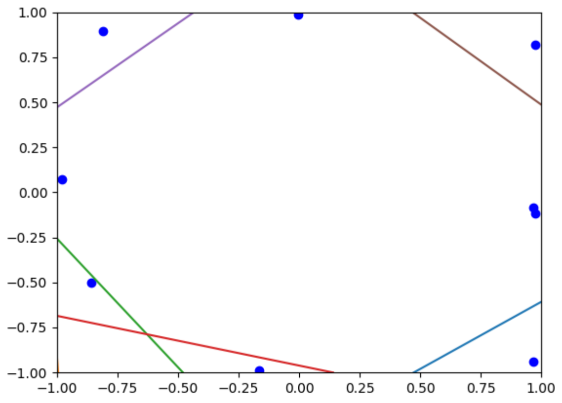

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

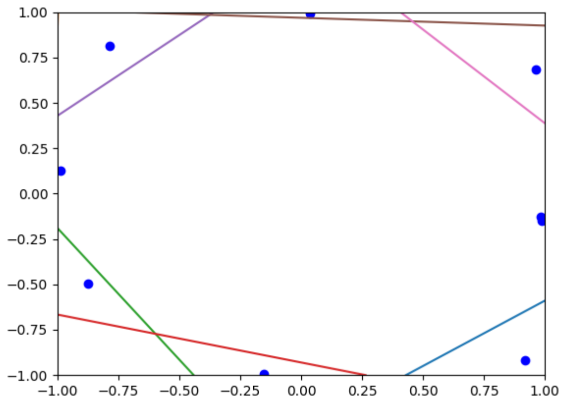

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

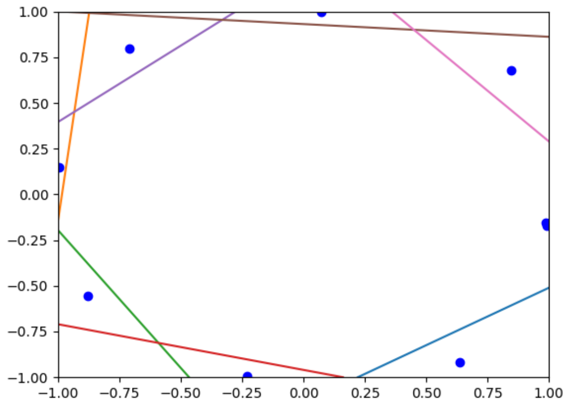

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

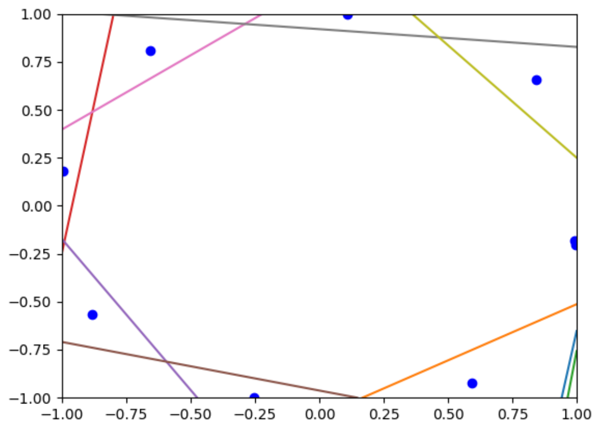

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

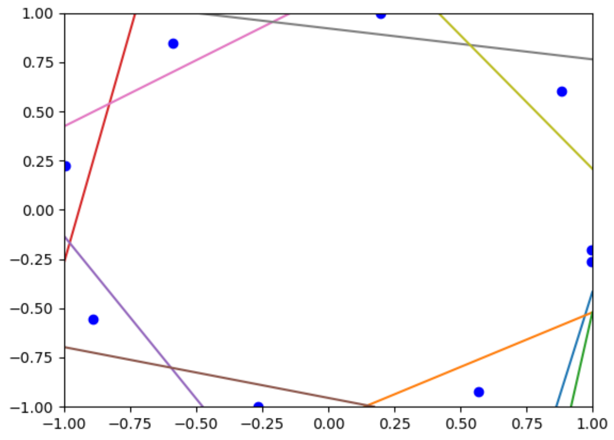

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

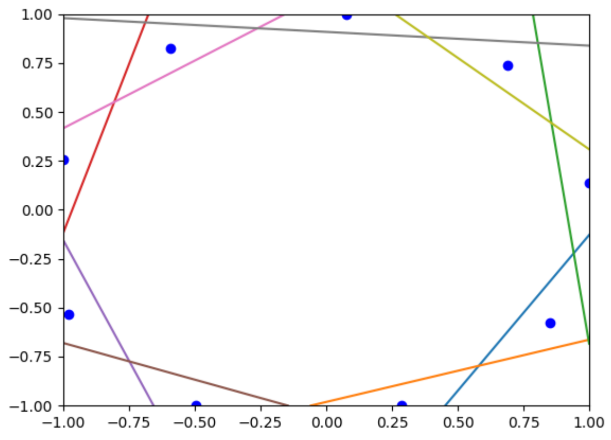

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00





This is the last picture of the question2.



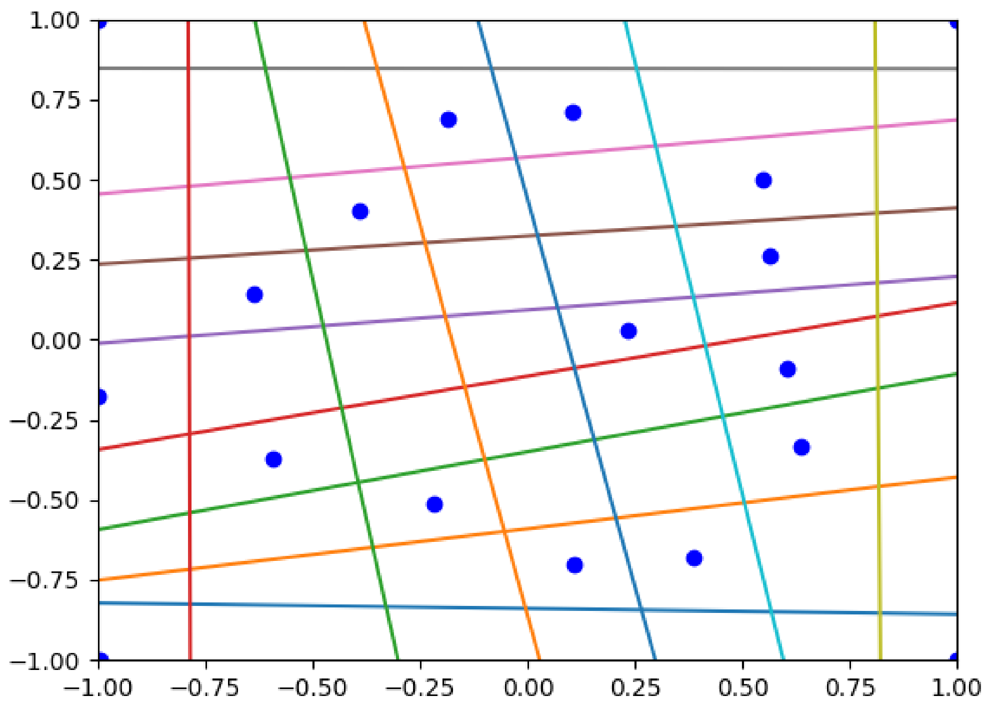
When the training start, the dots separate and go in different directions. When they near the boundary, the cut lines appear. These lines start from the boundary and then go to the middle of the graph. When these lines 'trap' these dots in different areas, the training process will stop.

Question3

First, we should change the graph to a matrix. Also, adding the node index into the table. The size of the table is due to the line in the graph, 9 columns which have 8 lines and 7 rows which have 6 lines. Therefore, we can get this table.

1								17
		5	7		11	13		
	3			9			15	
	4						16	
		6				14		
			8		12			
2				10				18

Then, change this table to the matrix, based on whether the node is on the left side of the column line or the downside of the row line. Then we can get the picture result as showing below.



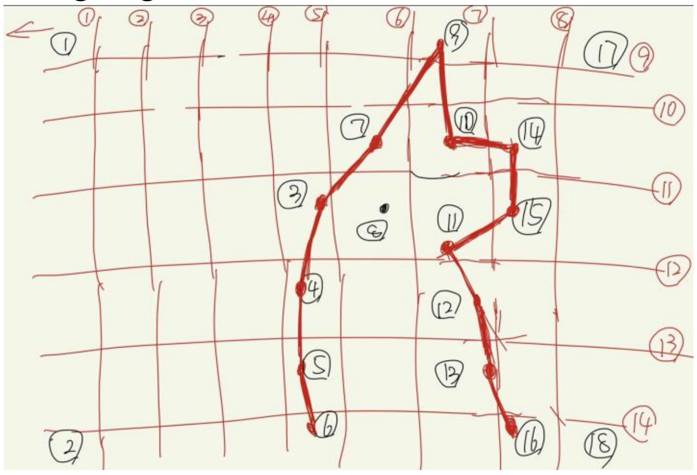
From this picture, we can find this heart has rotated left for 90 degree.

Question4

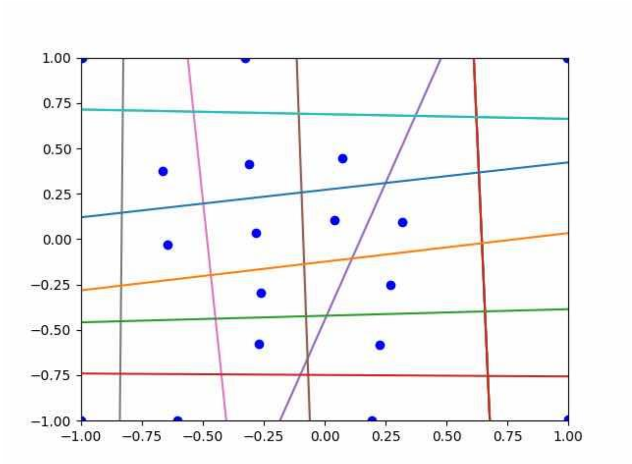
Target1

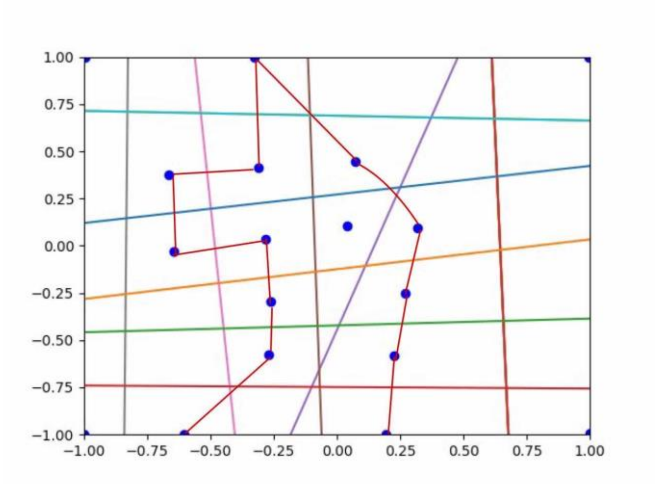
An Abstract Unicorn(Head only)

Designing Draft



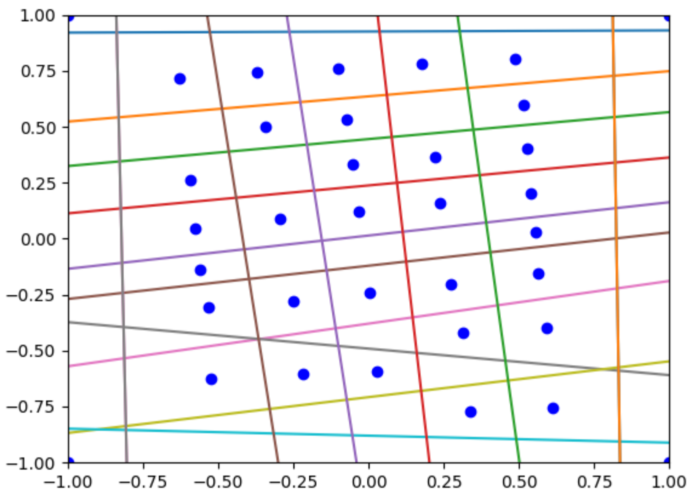
The Result

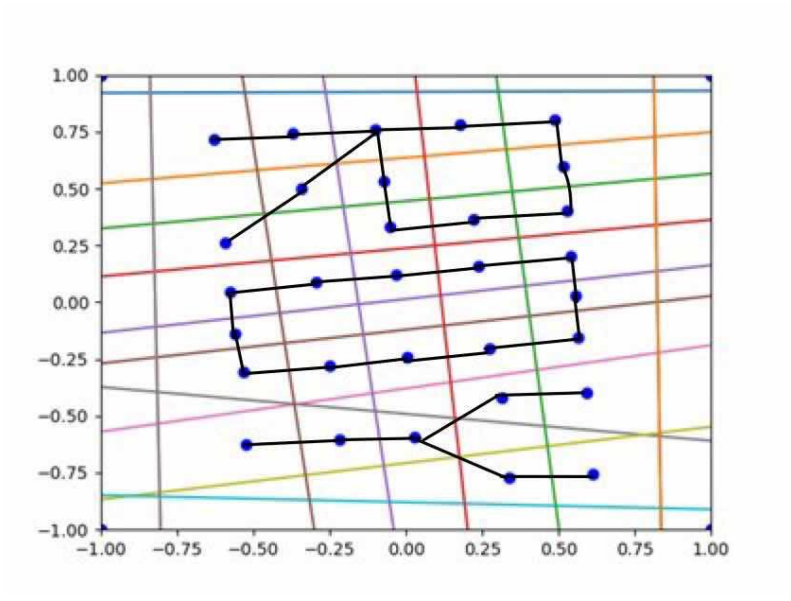




Target2 Roy(My English name)

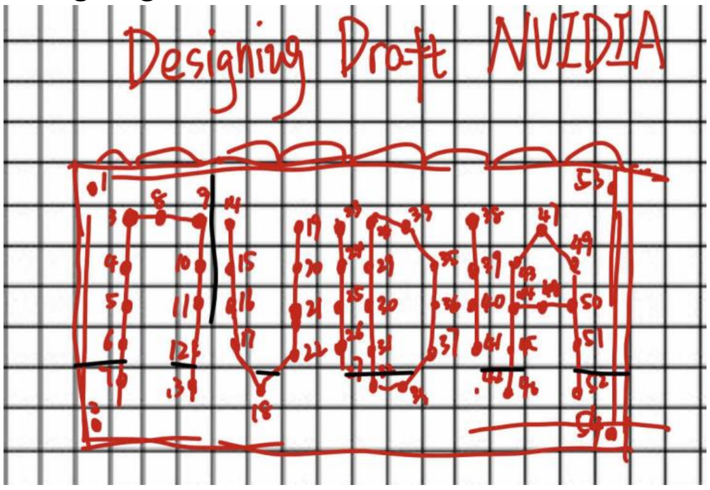
The picture result





Target3 nVIDIA

Designing Draft



Result picture