

## Part 1: Japanese Character Recognition

### Question 1

```
Train Epoch: 10 [0/60000 (0%)] Loss: 0.832651
Train Epoch: 10 [6400/60000 (11%)] Loss: 0.577718
Train Epoch: 10 [12800/60000 (21%)] Loss: 0.598824
Train Epoch: 10 [19200/60000 (32%)] Loss: 0.590446
Train Epoch: 10 [25600/60000 (43%)] Loss: 0.314776
Train Epoch: 10 [32000/60000 (53%)] Loss: 0.509968
Train Epoch: 10 [38400/60000 (64%)] Loss: 0.542564
Train Epoch: 10 [44800/60000 (75%)] Loss: 0.595902
Train Epoch: 10 [51200/60000 (85%)] Loss: 0.371376
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.667455
<class 'numpy.ndarray'>
[[785.  5. 11. 11. 30. 42.  2. 66. 30. 18.]
 [ 8. 689. 104. 19. 29. 12. 57. 12. 23. 47.]
 [ 7. 62. 706. 27. 26.  7. 47. 33. 46. 39.]
 [ 5. 36. 74. 756. 17. 39. 13. 20. 30. 10.]
 [ 62. 51. 80. 19. 627. 11. 32. 37. 23. 58.]
 [ 8. 33. 143. 19. 21. 696. 27. 10. 33. 10.]
 [ 5. 27. 151. 12. 28. 12. 726. 19. 10. 10.]
 [19. 29. 26. 11. 88. 13. 54. 618. 99. 43.]
 [10. 33. 99. 43. 7. 23. 46. 9. 711. 19.]
 [10. 52. 92.  5. 60. 19. 23. 29. 40. 670.]]

Test set: Average loss: 0.9996, Accuracy: 6984/10000 (70%)
```

### Question 2

```
Train Epoch: 10 [0/60000 (0%)] Loss: 0.354667
Train Epoch: 10 [6400/60000 (11%)] Loss: 0.221446
Train Epoch: 10 [12800/60000 (21%)] Loss: 0.202414
Train Epoch: 10 [19200/60000 (32%)] Loss: 0.187982
Train Epoch: 10 [25600/60000 (43%)] Loss: 0.133946
Train Epoch: 10 [32000/60000 (53%)] Loss: 0.235989
Train Epoch: 10 [38400/60000 (64%)] Loss: 0.193504
Train Epoch: 10 [44800/60000 (75%)] Loss: 0.352763
Train Epoch: 10 [51200/60000 (85%)] Loss: 0.122110
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.257848
<class 'numpy.ndarray'>
[[851.  4.  1.  5. 27. 35.  3. 38. 29.  7.]
 [ 4. 810. 37.  4. 19.  9. 60.  4. 21. 32.]
 [ 7. 14. 844. 48. 12. 15. 22. 11. 14. 13.]
 [ 3. 10. 31. 923.  2. 10.  4.  2.  7.  8.]
 [36. 26. 21.  8. 828.  6. 27. 17. 19. 12.]
 [ 8. 12. 80. 14. 13. 829. 20.  1. 17.  6.]
 [ 3. 11. 53.  8. 15.  7. 886. 10.  1.  6.]
 [20. 13. 15.  6. 15.  9. 30. 838. 20. 34.]
 [ 8. 24. 29. 57.  6.  9. 31.  6. 821.  9.]
 [ 4. 16. 50.  4. 28.  7. 18. 14.  9. 850.]]

Test set: Average loss: 0.4963, Accuracy: 8480/10000 (85%)
```

### Question 3

```
Train Epoch: 10 [0/60000 (0%)] Loss: 0.170050
Train Epoch: 10 [6400/60000 (11%)] Loss: 0.157667
Train Epoch: 10 [12800/60000 (21%)] Loss: 0.143385
Train Epoch: 10 [19200/60000 (32%)] Loss: 0.197878
Train Epoch: 10 [25600/60000 (43%)] Loss: 0.081970
Train Epoch: 10 [32000/60000 (53%)] Loss: 0.168898
Train Epoch: 10 [38400/60000 (64%)] Loss: 0.182568
Train Epoch: 10 [44800/60000 (75%)] Loss: 0.259264
Train Epoch: 10 [51200/60000 (85%)] Loss: 0.106009
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.160994
<class 'numpy.ndarray'>
[[965.  3.  2.  2. 19.  0.  1.  7.  1.  0.]
 [ 4. 905. 11.  0. 12.  3. 46.  5.  5.  9.]
 [11.  0. 863. 42.  9. 10. 38.  8.  7. 12.]
 [ 1.  2. 13. 968.  1.  4.  7.  0.  1.  3.]
 [28.  6.  3.  6. 920.  5. 16.  6.  7.  3.]
 [ 3.  5. 36. 11.  1. 916. 20.  2.  3.  3.]
 [ 1.  3. 16.  2.  4.  3. 966.  3.  0.  2.]
 [11.  3.  3.  1. 10.  3. 15. 924.  5. 25.]
 [ 6. 15.  7. 23.  7.  6.  4.  5. 922.  5.]
 [ 6.  2. 13.  6. 15.  2. 10.  2.  7. 937.]]

Test set: Average loss: 0.2510, Accuracy: 9286/10000 (93%)
```

#### Question 4

A

In question 1, there is just one layer, the accuracy is around 70%. In question 2, there are two fully connected layers, the relate is 85%. In question 3, there are two convolution layers and a fully connected layer followed the output layer, during the process, max pooling function and dropout function are used, the accuracy is 93%. Obviously, the accuracy is increasing when the number of layers increased and if the model has a convolution layer the accuracy improved significantly.

B

In question 1, 'na' and 'ya' are more difficult to recognize than other words, and the recognize rates are below 65%. It means that 'na' and 'ya' are more easily to be recognized as other words. In addition, 'su' is the word which is more likely to be considered as another word. Especially, 'ki', 'ha' and 'ma' are likely to be considered as 'su' in the result. The reason is that due to the column data of the table, when identifying these three letters, there is a 10% probability that they will be considered 'Su'.

In question 2, 'ki' is the is the most difficult word to be recognized, which accuracy is 81%. And 'su' is more likely to mistake another word for this one. Especially 'ha' sometimes would be recognized as 'su'.

In question 3, 'su' and 'ma' are most likely to be mistaken for which other characters.

C

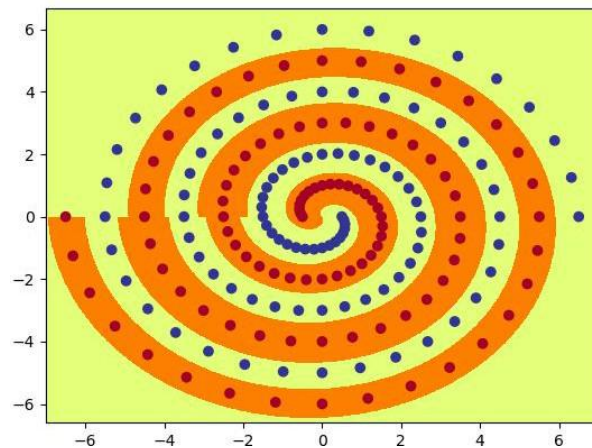
I used 3 convolution layers with 3 connected layers and relu during the process, but the accuracy did not increase compare with the result of question 3. May be it is not the deeper the network is, the better the network is. The simplest model maybe can not achieve a high accuracy, so use some method like dropout and maxplooing can increase the accuracy. But once the method was used, improve the number of layers or use the method again may not work. Changing the parameters of linear function may also has little influence.

In addition, if implement 2 layers with relu function and one layer with tahn function, it may not work.

## Part 2: Twin Spirals Task

### Question 1

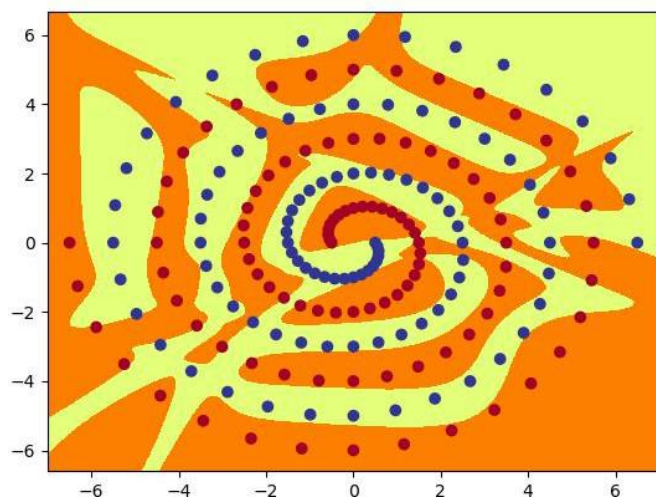
The minimum number of hidden nodes is 7.



### Question 4

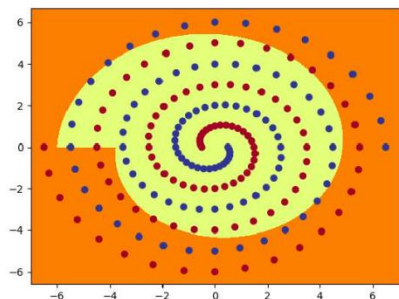
I tried several number of hidden and init, when hidden number  $\geq 9$  and init  $\geq 0.2$ , the data can be classified within 20000 epochs on almost all runs.

Hidden = 9 and init = 0.2

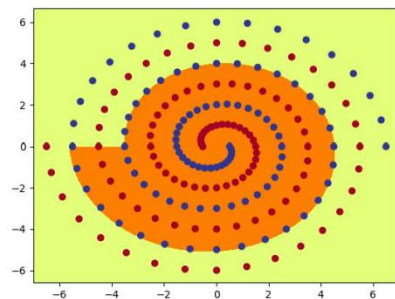


# Question 5

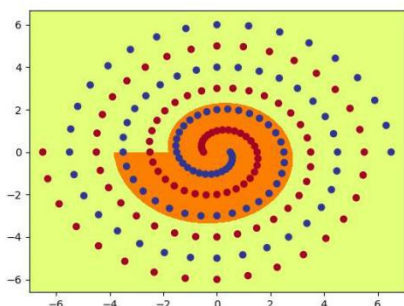
PolarNet hid = 7



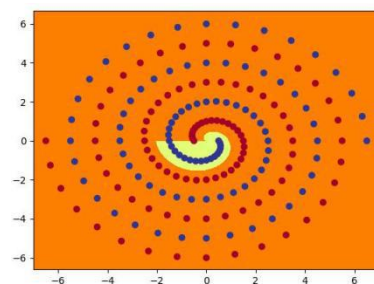
1



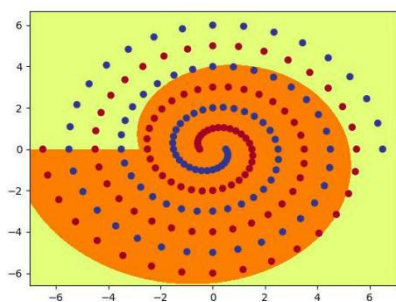
2



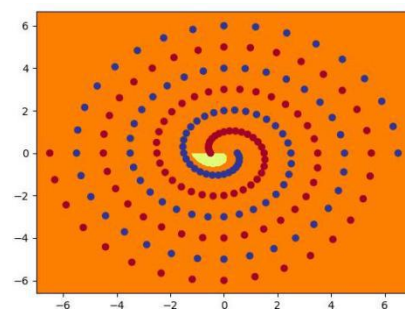
3



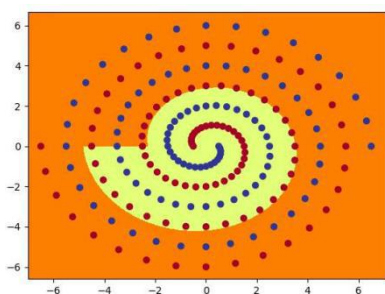
4



5



6

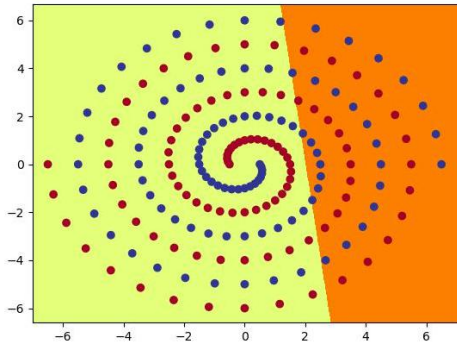


7

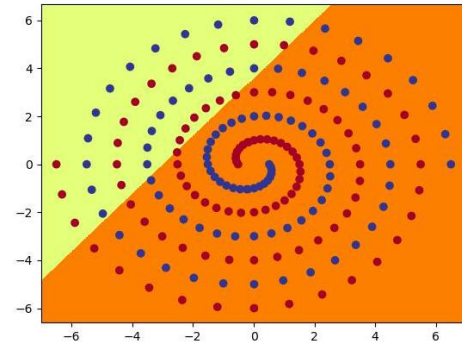


RawNet hid = 9, init = 0.2

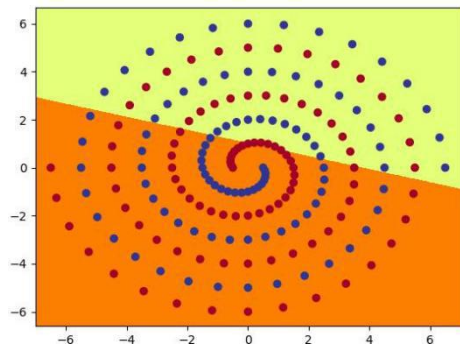
Layer1



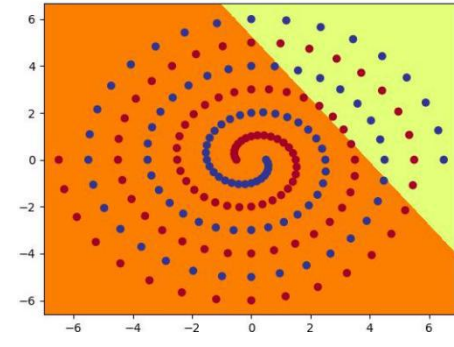
1



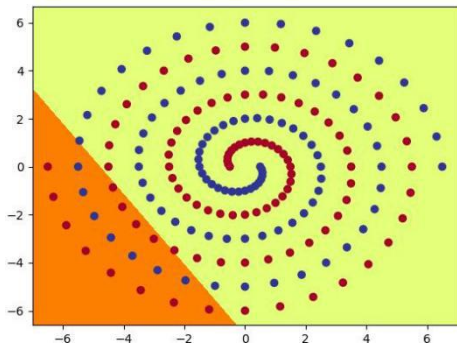
2



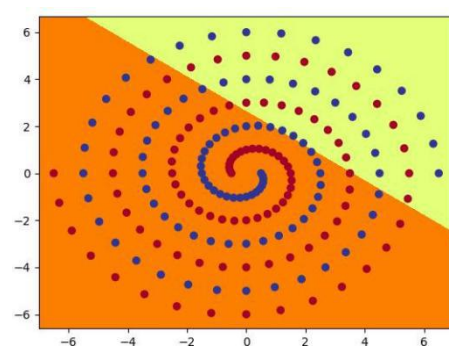
3



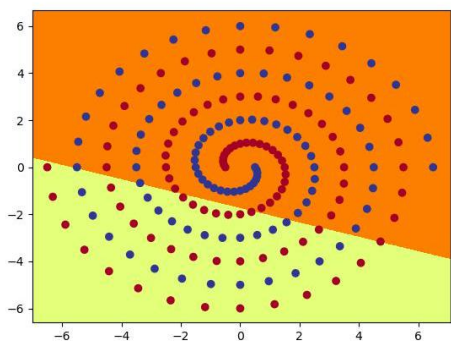
4



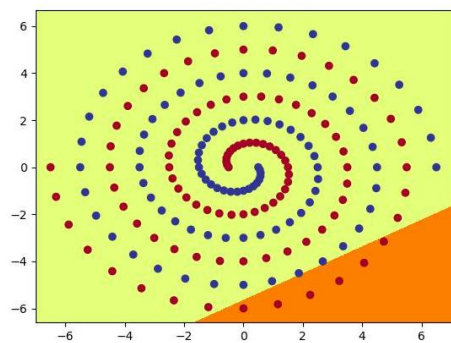
5



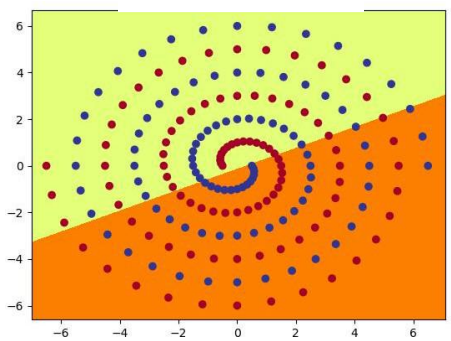
6



7

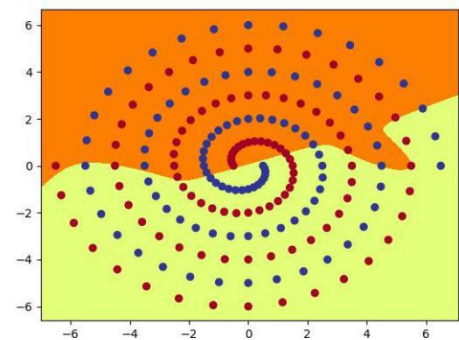


8

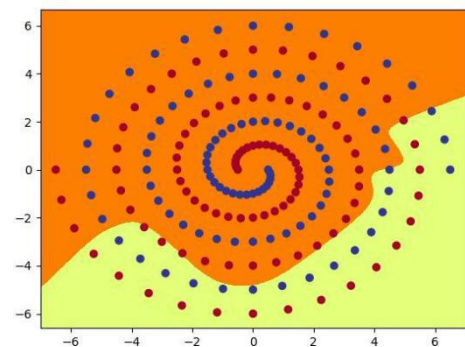


9

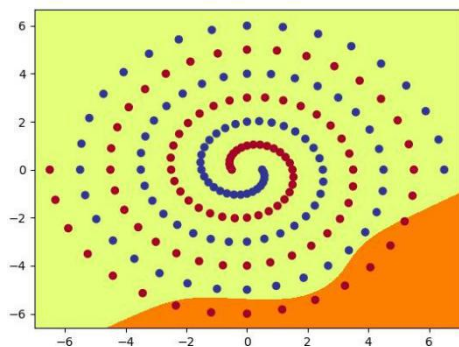
layer2



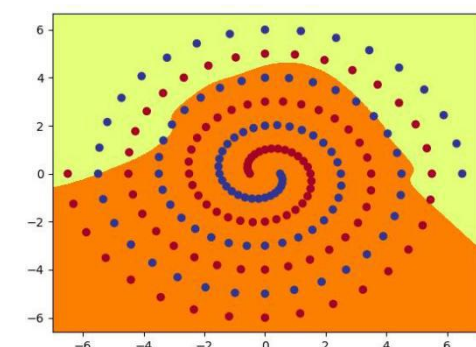
1



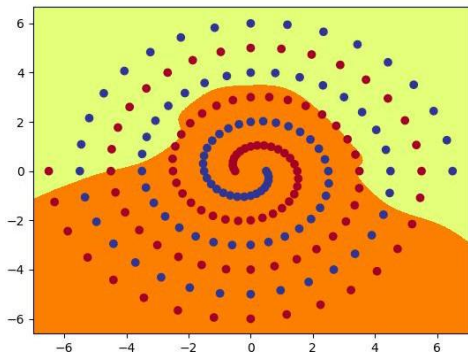
2



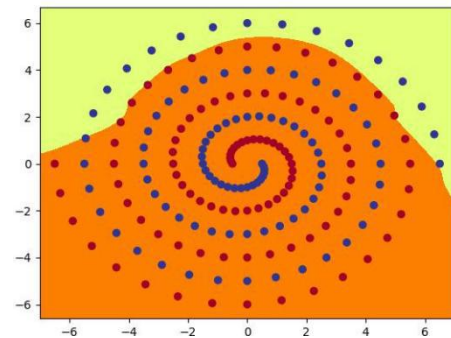
3



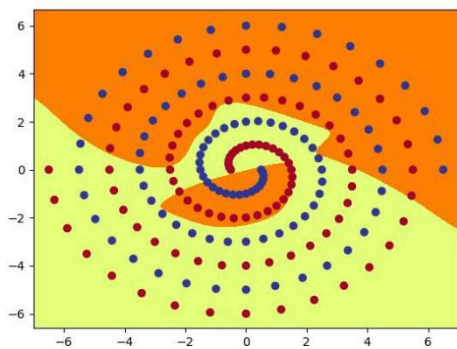
4



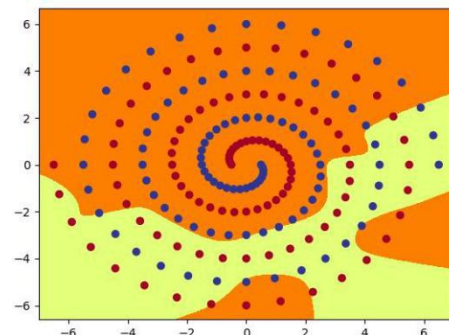
5



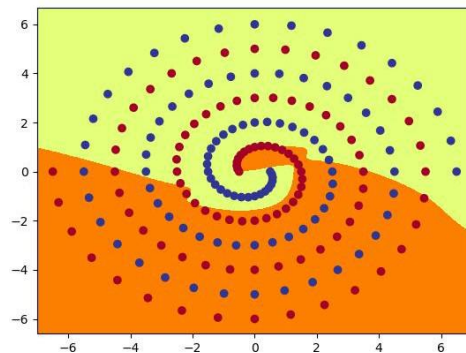
6



7



8



9

### Question 5

A

The images between PolarNet and RawNet are different, the images compute by PolarNet are more likely a circle, because it is polar coordinates and the 'line in Rectangular coordinate system' in polar coordinates is a circle. In the data set, the blue spot is class 0 and the red spot is class 1, the function is just write line to classify the spots. If the current parameters can not classify the points, the program will uses back-propagation method and optimize the weight. In addition the program tries to using adam function to update network weight. As a result, the points in two color will be divided into two groups.

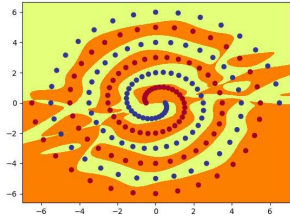
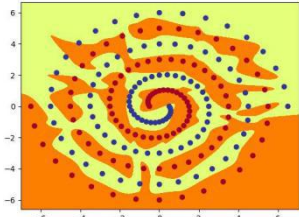
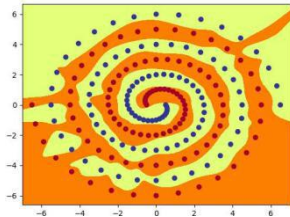
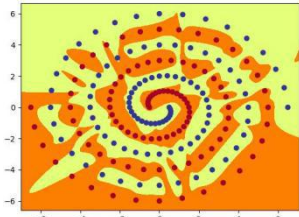
B

With a higher weight size, the speed of learning increased remarkable firstly and decreased continuously after that.

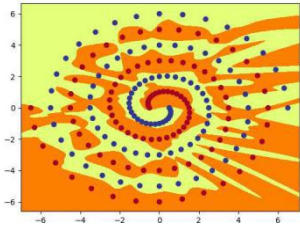
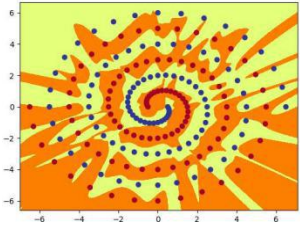
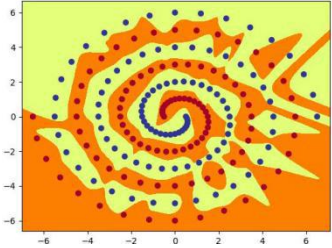
I tried weight = 9 and init from 0.01 to 5, it can not get the result when init = 0.01 and the accuracy is always around 50.52, and then I increase the value of init, the speed of learning increased and the test succeed when init = 0.2, but in the following process, the value of init increased to 5(include 0.3, 0.5, 1, 2), the test can not success in 20000 ep.

After that, I choose weight = 20 and init from 0.001 to 4, the result is in below

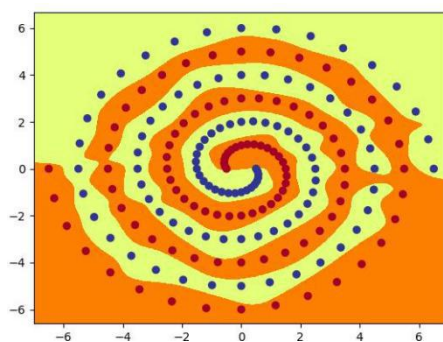
weight = 20, init from 0.001 to 4

Init	epoch	result
0.001	fail	Accuracy around 50.52 in 20000 epoch
0.01	fail	Accuracy around 50.52 in 20000 epoch
0.1	3900	
0.2	2300	
0.3	2100	
1	1200	



2	4400	
3	16700	
4	27100	

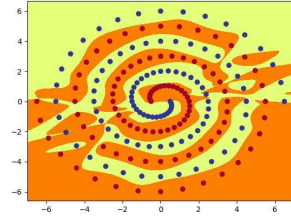
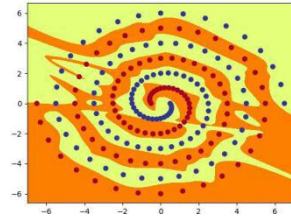
Obviously, when I change the value of init with weight=20, the result is same as weight=9. The output image maybe more chaos during the process of increasing the value init.



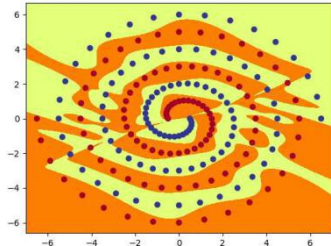
When weight = 30 and init = 0.1, the output picture is more natural.

C

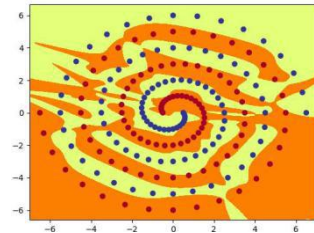
Weight = 20, init = 0.1

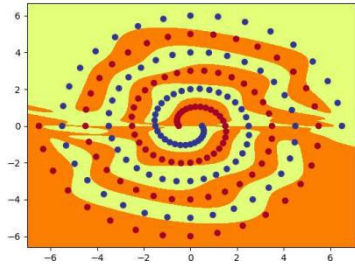
Batch size	epoch	result
96	3900	
174	4200	

Weight = 10 init = 0.1

Batch size	epoch	result
96	fail	Accuracy around 98.45 forever after 20000 epoch
174	1900	

Weight = 10 init = 0.2

Batch size	epoch	result
96	11300	

174	2700	
-----	------	--

Weight = 10 init = 1

Batch size	epoch	result
96	fail	Accuracy around 90 after 20000 epoch
174	fail	Accuracy around 95 after 20000 epoch

When batch size = 174 can also get a result, the tendency is same as batch size, but the most effective init may be different.

Use SGD or Adam

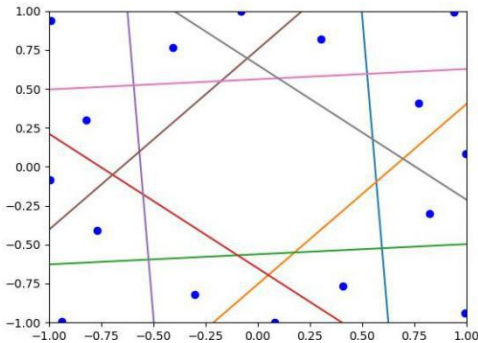
Polar, Hid = 10, momentum=0, use SGD function

hid	epoch	result
10	fail	Accuracy less than 60 after 20000 epoch
20	fail	Accuracy around 60 after 20000 epoch
50	fail	Accuracy around 65 after 20000 epoch

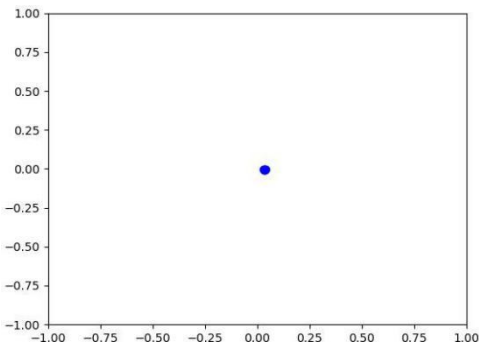
Using SGD function can not get a result within 20000 epoch, this function may not suit to this task and it is not effective enough. Change momentum to 1 can not get a result either within 20000 epoch.

Part 3: Hidden Unit Dynamics

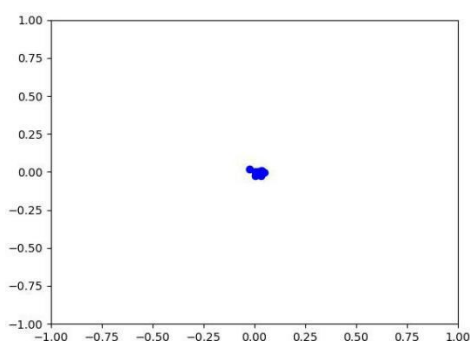
Question 1



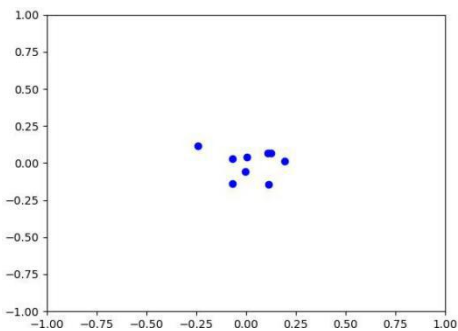
Question 2



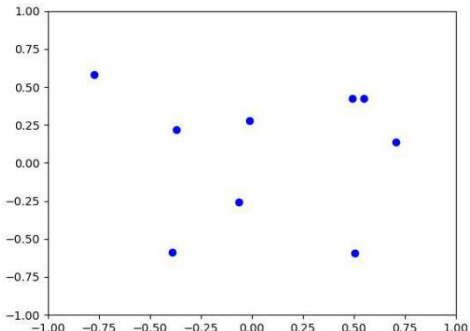
1



2

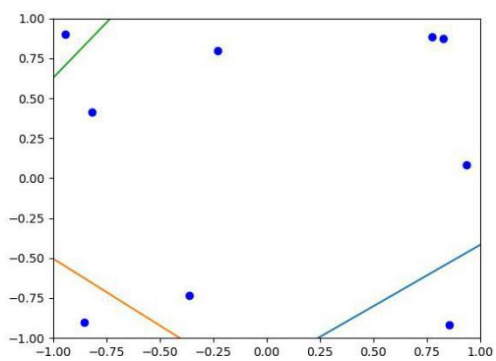


3

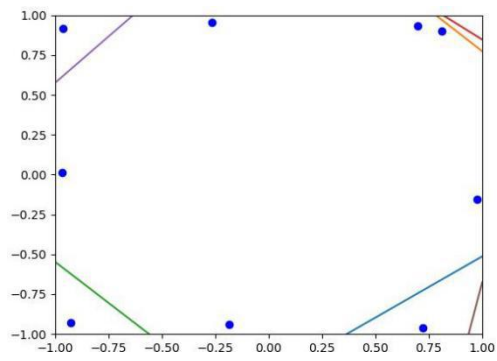


4

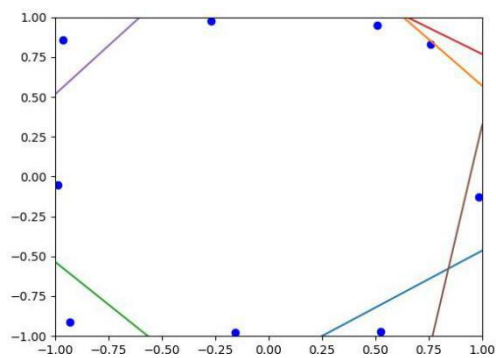




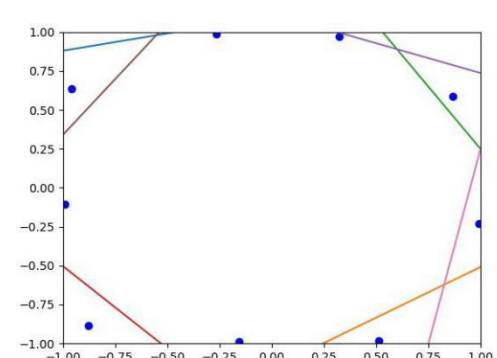
5



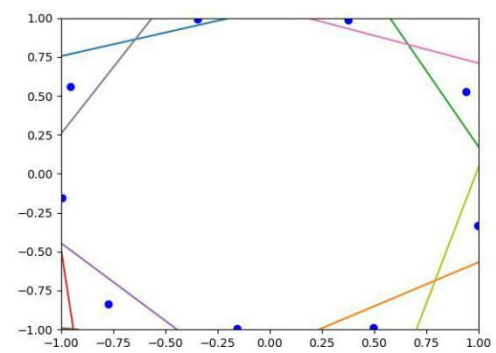
6



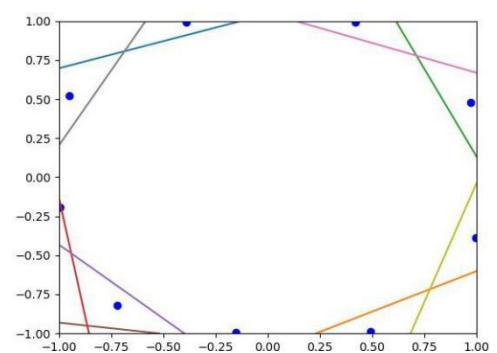
7



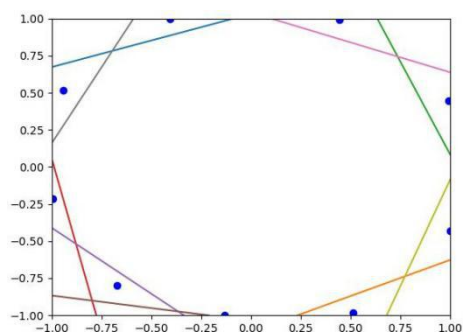
8



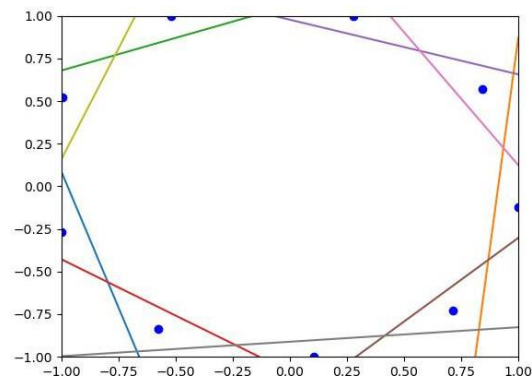
9



10



11

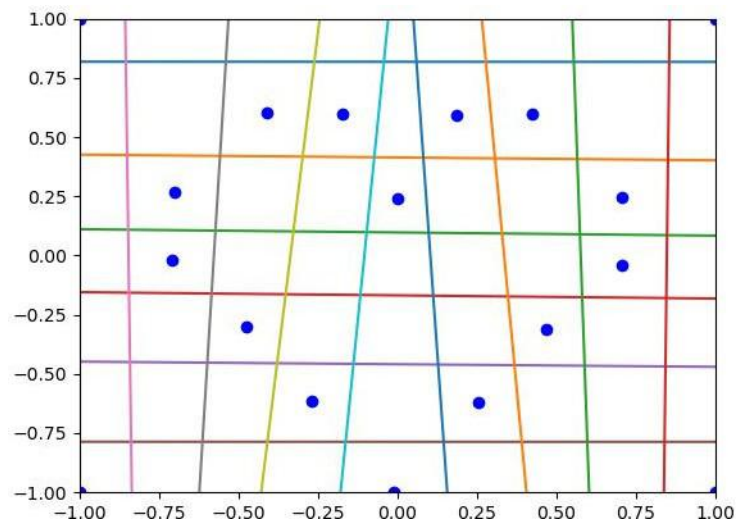


*Final image*

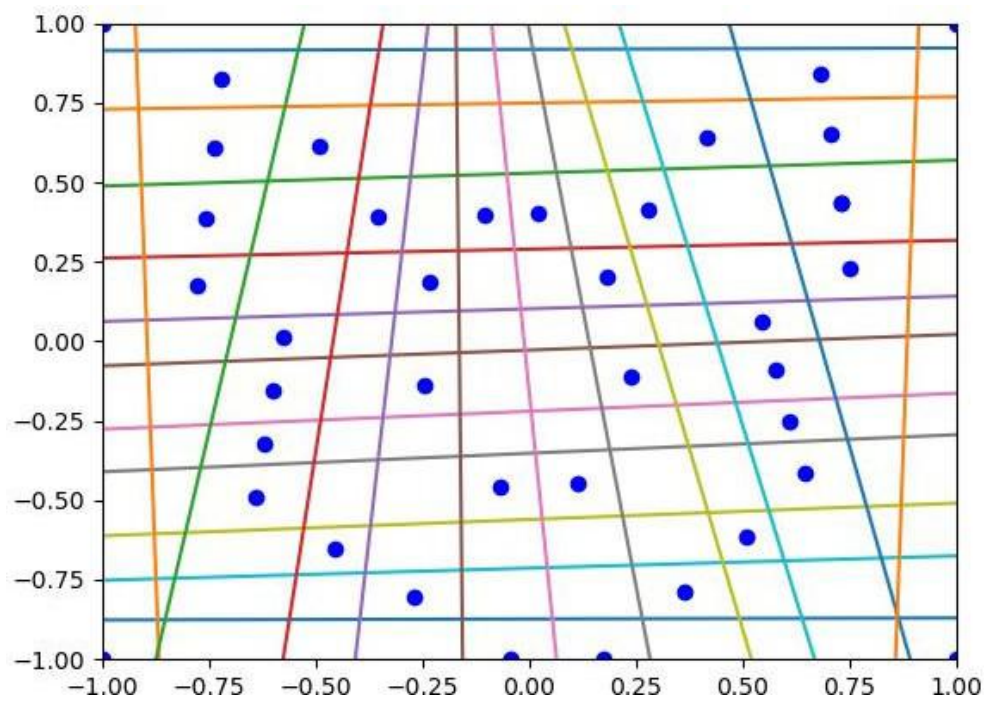
The target and input matrix are both a diagonal matrix which columns and rows are both 9.

The column of matrix means that there are 8 spots and the rows mean that there are 9 lines. The target of the program is to draw the output of each spot only in one the small part divided by a line. If class=1, the point is in the small part and if class=0, the point will in the big part. Reach a number of epoch, output a picture. In the first two pictures, there are no line, the program try to insert a line that can satisfy the target(a line divide the picture into two parts and only a point in the small part). If the spots not satisfy the target, then change both locations of lines and spots. So the spots should spread and after satisfy the target, then insert another line. As a result, the final image shows that a line dividing the hidden unit space into regions for which the value of that output is greater/less than one half.

### Question 3



Question 4  
Target - 1



Target - 2

