Part 1

1. Confusion matrix, average loss and accuracy for linear model after epoch 10:

```
[[764.   5.   7.  13.  30.  64.   2.  63.  31.  21.]
 [  7. 665. 106.  19.  31.  23.  59.  13.  25.  52.]
 [  9.  57. 692.  26.  25.  21.  47.  37.  47.  39.]
 [  5.  36.  56. 757.  15.  60.  13.  18.  29.  11.]
 [ 59.  55.  77.  22. 623.  20.  32.  35.  22.  55.]
 [  8.  28. 125.  17.  19. 725.  28.   8.  32.  10.]
 [  5.  20. 149.  10.  24.  24. 722.  20.  11.  15.]
 [ 14.  29.  30.  11.  85.  16.  52. 624.  90.  49.]
 [ 11.  36.  97.  41.   5.  31.  45.   7. 705.  22.]
 [  7.  49.  89.   3.  51.  32.  18.  31.  40. 680.]]
```

Average loss: 1.0092, Accuracy: 6957/10000 (70%)

2. Confusionmatrix, average loss and accuracy for fully connected 2 layer network after epoch 10:

```
[[837.   5.   1.   4.  30.  34.   2.  45.  37.   5.]
 [  4. 814.  33.   5.  19.  10.  62.   3.  18.  32.]
 [  8.  16. 831.  35.  10.  21.  26.  10.  24.  19.]
 [  4.   7.  33. 906.   4.  18.   8.   4.   8.   8.]
 [ 34.  24.  26.   8. 810.  11.  32.  19.  22.  14.]
 [  9.  12.  69.  12.  10. 834.  27.   4.  16.   7.]
 [  3.  16.  49.   5.  15.   6. 886.   8.   2.  10.]
 [ 15.  14.  20.   6.  32.  14.  34. 796.  32.  37.]
 [  9.  27.  24.  47.   5.   8.  29.   3. 837.  11.]
 [  2.  18.  45.   4.  31.   6.  26.  13.  14. 841.]]
```

Average loss: 0.5195, Accuracy: 8392/10000 (84%)

3. Confusion matrix, average loss and accuracy for a network with two convolutional layer plus one fully connected layer after 10 epoch:

```
[[942.   2.   1.   0.  30.   4.   1.  14.   2.   4.]
 [  4. 926.   9.   1.  14.   1.  32.   4.   3.   6.]
 [ 10.  15. 861.  41.  10.   9.  22.  20.   3.   9.]
 [  2.   3.  18. 947.   5.   3.   9.   7.   2.   4.]
 [ 19.   6.   6.   3. 912.   2.  17.  22.   7.   6.]
 [   5.   25.   32.    7.     4. 889.   19.   13.     3.     3.]
 [   4.   10.   14.    2.     7.     3. 953.     6.     0.     1.]
 [   8.     5.     2.    1.     4.     0.     5. 956.     2.   17.]
 [   5.   19.     6.    5.     9.     3.     4.     3. 943.     3.]
 [   4.   10.     5.    3.     6.     1.     2.     5.     3. 961.]]
```

Average loss: 0.2615, Accuracy: 9290/10000 (93%)

4.

a. From the results for the three network after 10 epoches, the accuaracy of multi-layer fully connected work is much higher than a simple linear function network, and the network with covolutional layer is better than the one only contains fully connected layer.

The hidden layer in the fully connected network helps to recognize some small features and the use them to recognize the whole image, therefore it’s much better than the linear function which tries to use all pixels to recognize the whole image at once.

However, there are too many connections and too many parameters in the fully connected layer. The connections in convolutional network are much more sparsy, but it still capture the feature from its predecessors automatically. This makes the convolutional network does a better job.

b. For the linear model, column 2 (count column from left to right from 0), correpsond to “su”, is most likely to be recoginized as raw 5,6, (“ha”,”ma”), since the (2,5) and (2,6) entry has value (125 and 149) which are closeset to the number correct prediction (692).

Observing from the other confusion matrix, for the 2 layer fully connected network, column 2 (“su”) is most likely to be predicted as raw 5 (“ha”), and for the convolutional network column 2 (“su”) is still most likely to be predicicted as raw 5 (“ha”).

c. Learning rate is an important metaparamter for nerual networks, it controls the speed of learning. First try different learning rate for three models:

Linear Network (Momentum set to 0.5)

Learning rate		0.001	0.01	0.1	0.5
Epoch, Accuracy	1	0.58	0.67	0.67	0.63
	5	0.66	0.69	0.68	0.62
	10	0.67	0.70	0.67	0.63
	15	0.68	0.70	0.67	0.63
	20	0.69	0.70	0.67	0.63

2-Layer fully connected network (Momentum set to 0.5)

Learning rate		0.001	0.01	0.1	0.5
Epoch, Accuracy	1	0.53	0.68	0.81	0.46
	5	0.65	0.80	0.86	0.63
	10	0.69	0.84	0.86	0.66
	15	0.71	0.86	0.87	0.71
	20	0.73	0.87	0.87	0.71

Convolutional network (Momentum set to 0.5)

Learning rate		0.001	0.01	0.1	0.3
Epoch, Accuracy	1	0.54	0.81	0.92	0.91
	5	0.76	0.92	0.95	0.93
	10	0.83	0.93	0.95	0.92
	15	0.86	0.94	0.95	0.90
	20	0.88	0.94	0.96	0.10

It is obverous that the learning rate will affect the learning speed. The network quickly converge with a relaive high learning rate. For example, in the 2 fully connected layer network, 0.1 learning rate approaches 86% accuacry after epoch 10, while the 0.01 and 0.001 only got 84% and 69% respectively.

But if the learning rate is too large, the model reaches a sub-optimal set of weight, or even osicallte between epoches, results the accuaracy siginificantly lower. For example, in the convolutional layer network, 0.3 learning rate achieved 93% accuaracy at epoch 5, but back to 90% accuaracy at epoch 15, and completely mess up at epoch 20!

Momentum can help the gradient vector accelerates in the right direction. Try different momentum for the models:

Linear Network (Learning rate set to 0.01)

Momentum		0.2	0.5	0.7	0.9
Epoch, Accuracy	1	0.66	0.67	0.68	0.69
	5	0.69	0.69	0.70	0.69
	10	0.70	0.70	0.70	0.69
	15	0.70	0.70	0.70	0.69
	20	0.70	0.70	0.70	0.69

2-Layer fully connected network (Learning rate set to 0.01)

Momentum		0.2	0.5	0.7	0.9
Epoch, Accuracy	1	0.66	0.68	0.71	0.78
	5	0.76	0.80	0.83	0.86
	10	0.81	0.84	0.86	0.87
	15	0.84	0.86	0.87	0.87

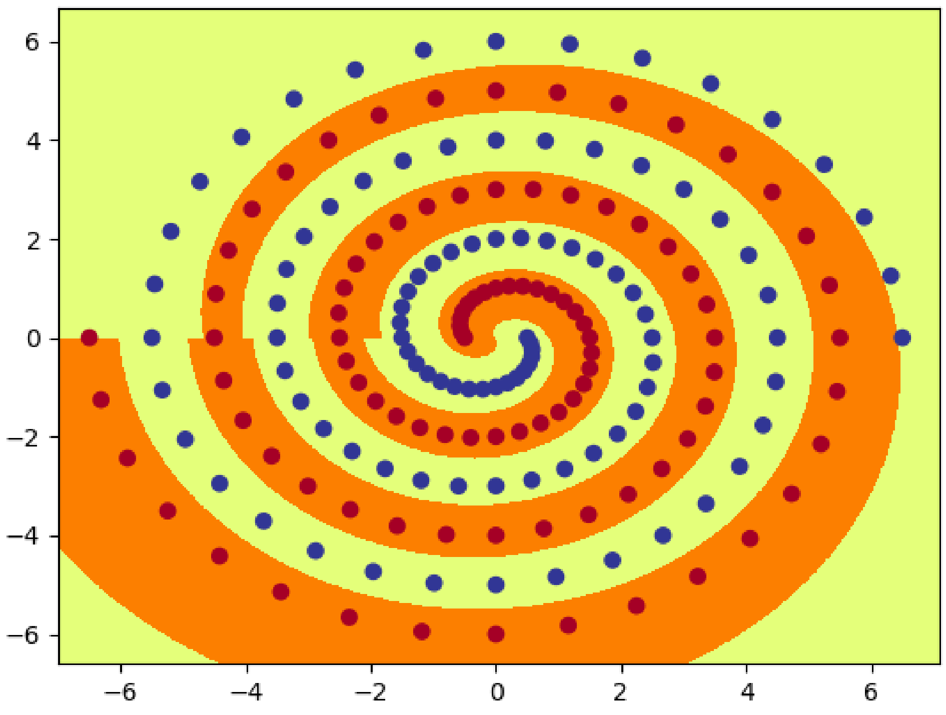
	20	0.85	0.87	0.88	0.87

From the table above, we can see the network perform slightly better with a moderate momentum.

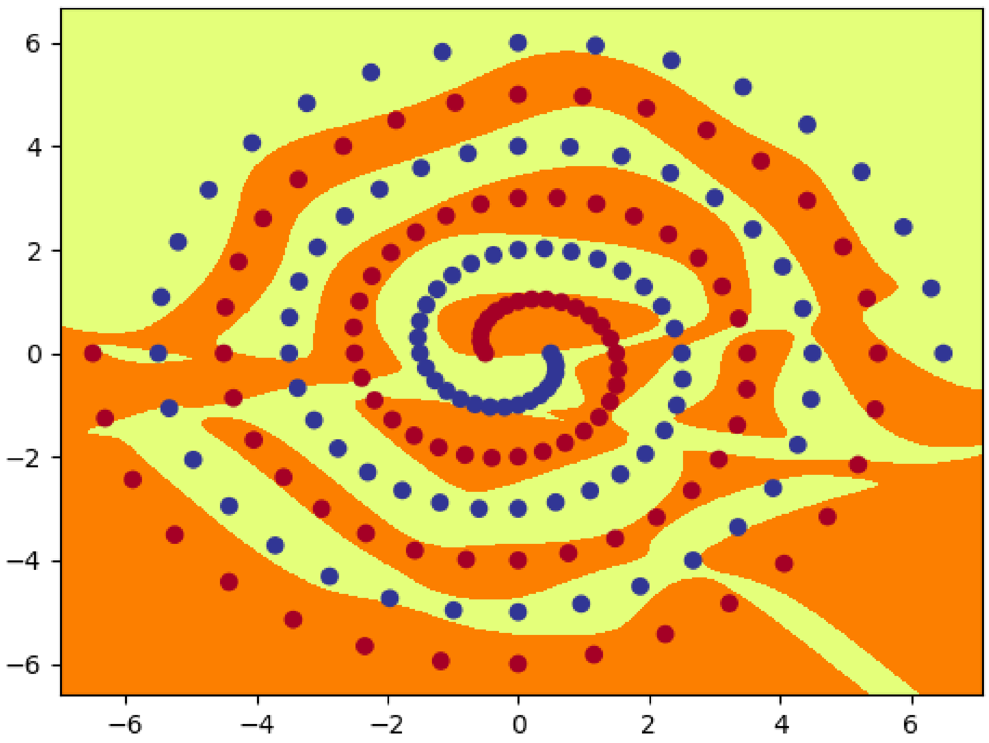
To constantly achieve 95% accuaracy, use my convolutional network with learning rate 0.1 and momentum 0.5.

Part 2

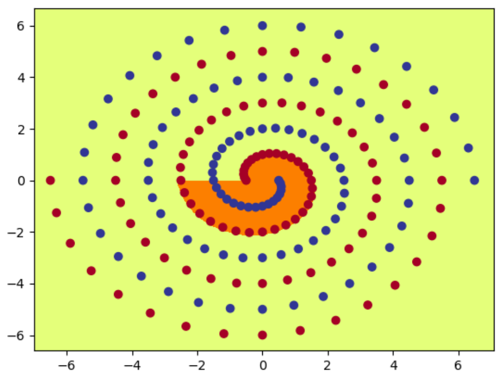
1. Code in spiray.py
2. polar_out.png

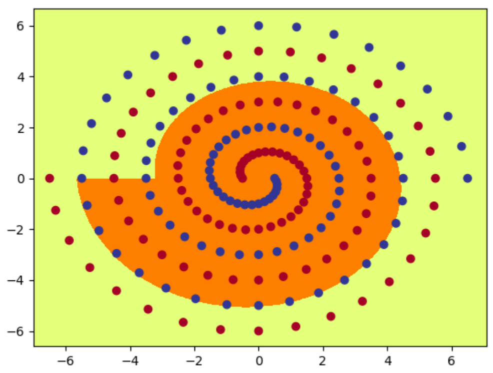


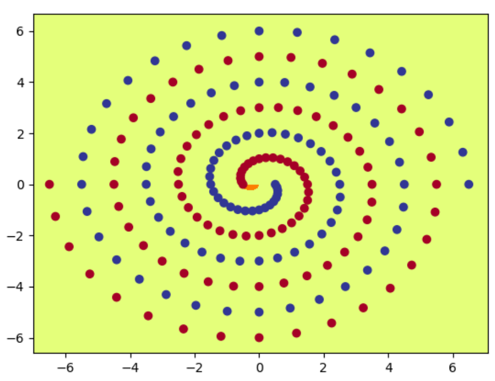
3. Code in spiray.py
4. Raw_out.png

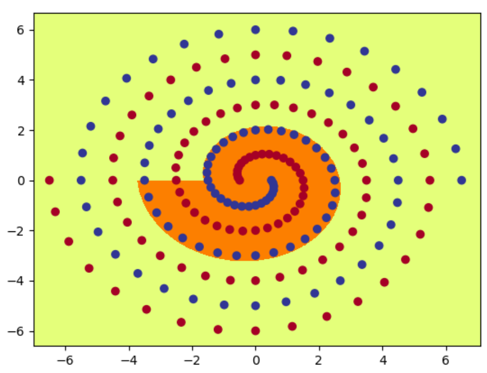


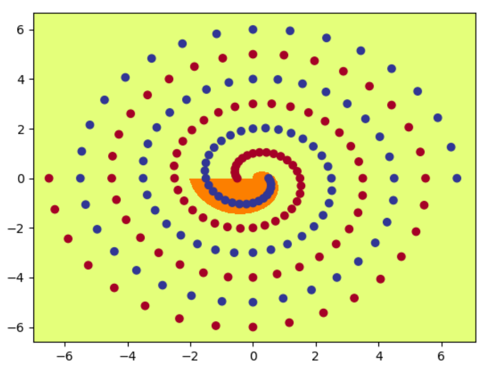
5. Hidden nodes: (from left to right, up to down, order from 0 to 9)
    a. network uses polar coordinate

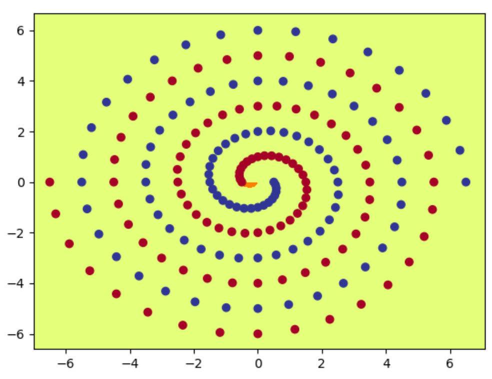


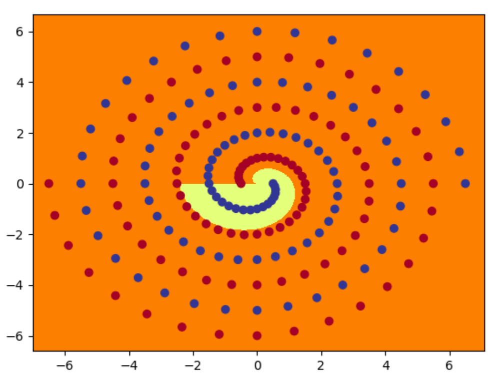


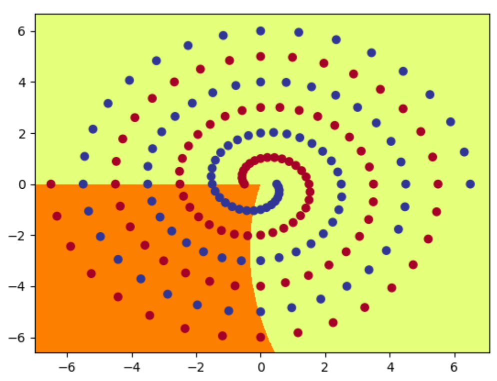

6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6

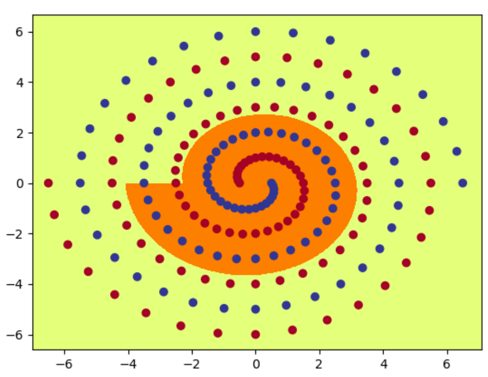

6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6

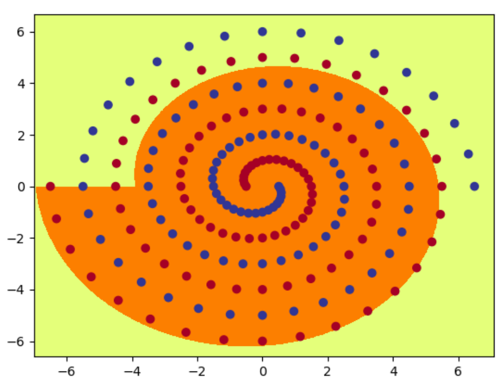

6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6


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

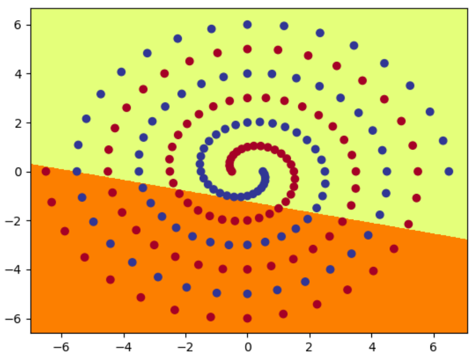
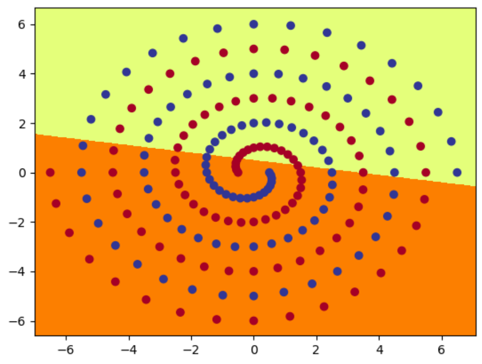
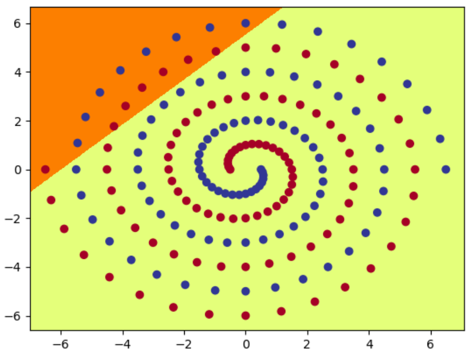
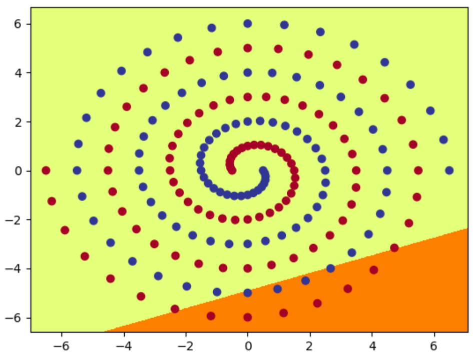
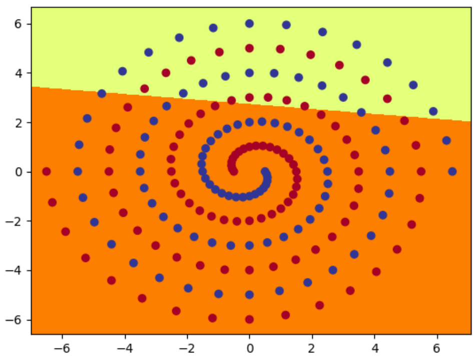
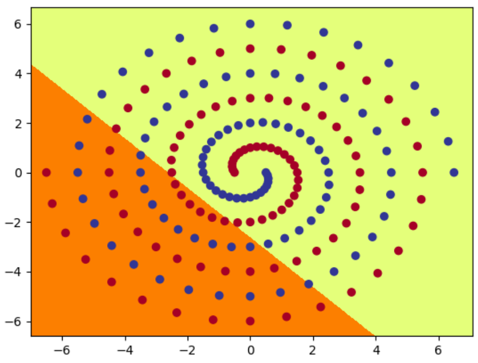

6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6

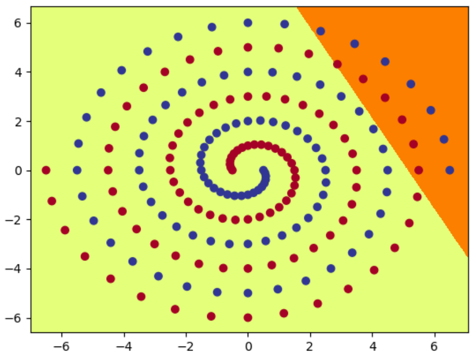

6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6

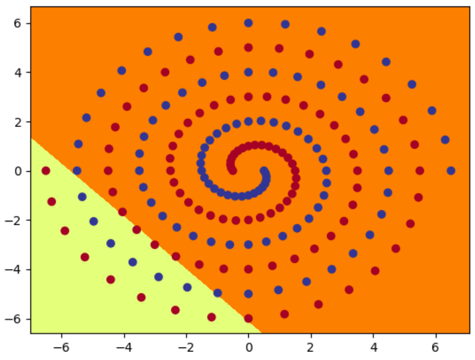

6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6

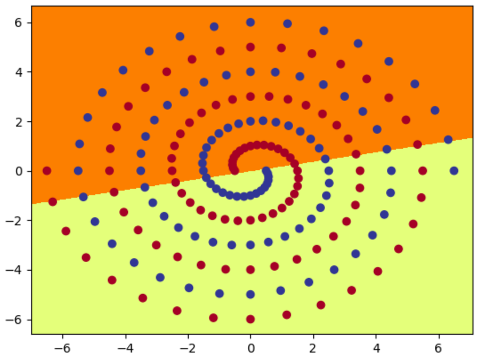
b. Network uses raw coordinates

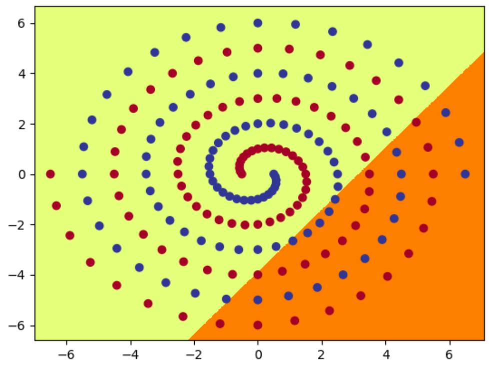
First layer

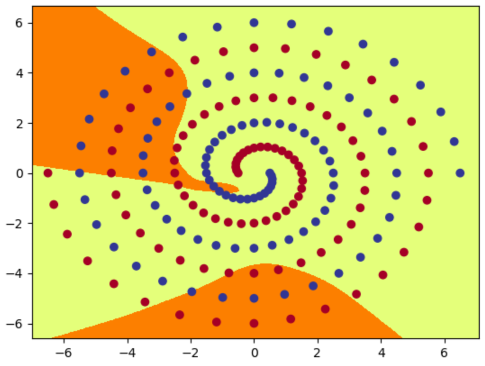


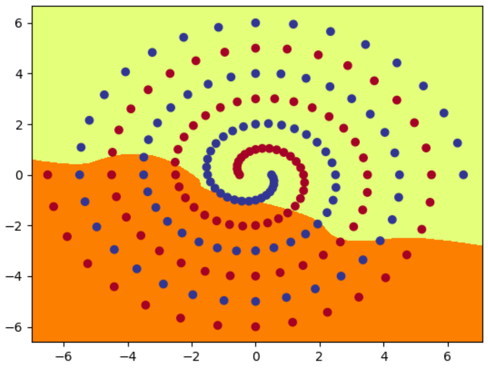


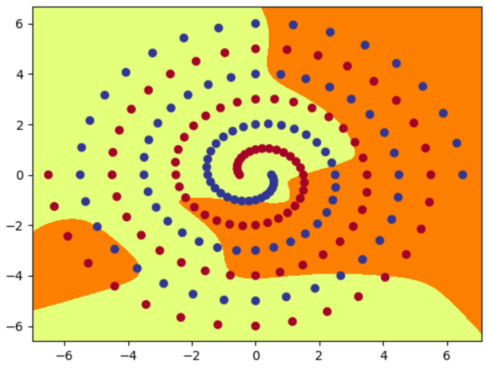


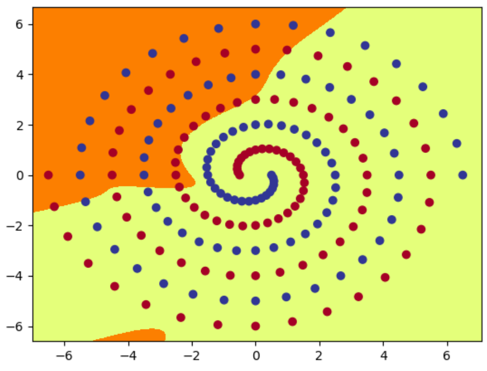


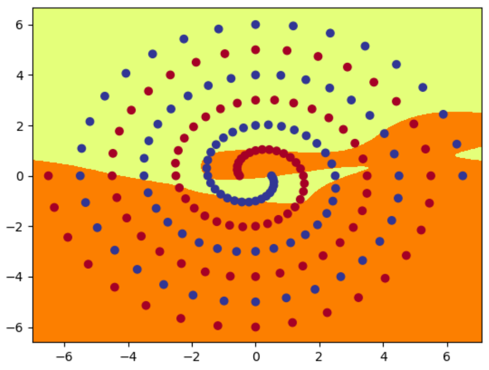
Second layer

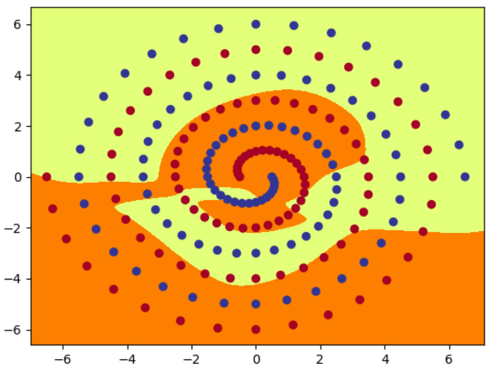


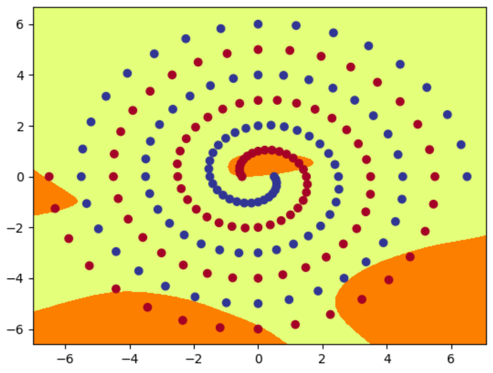


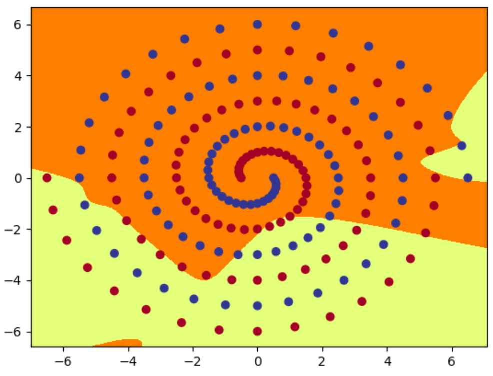


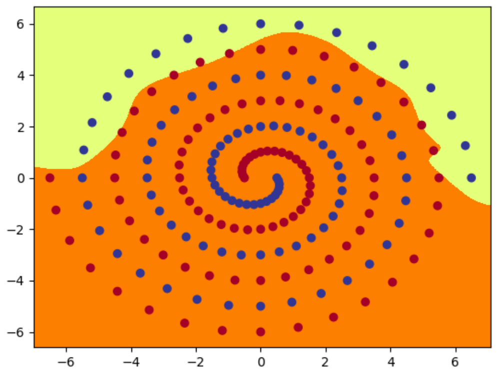


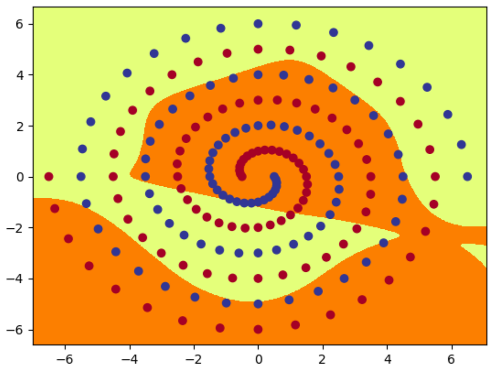












6. Discussion:
    a. The hidden nodes in polar net is non-linear.
    The hidden nodes in the first hidden layer of raw net is linear, and the nodes in the second hidden layer of raw net is non-linear.

    Each hidden nodes in the same layer works like a preceptron, which is linear separable from each other nodes in the same layer. The next layer takes the previous layer as input, aggregates the information from the previous layer into complex regions. The

b. For 10 hidden nodes, it won't success if the initial weights size smaller that 0.1, and the success speed reaches its maximum at weight size 0.1. When the weight size greater than 0.1, the success speed decreases as the weight size increases.

c. The epoch needed to achieve 100% accuaracy is different every time for both the rawand the polar net, even though all the metaparameters are the same. This makes sense because we use gradient descent, and the starting position each time is random, so the steps needed to apporach optimal are different.

However, the choice of batch size does effect the learning speed. The whole data set has a size of 194, try batch size 49, 97 and 194 for the raw net, some of the result is as following:

Epoches needed for raw net to achieve 100% accuarcy

Batch size	Trail 1	Trail 2	Trail 3	Trail 4	Trail 5	Trail 6
49	4000	7300	9400	7000	10200	>50000
97	4700	>50000	10800	9500	>50000	11900
194	>50000	11700	15400	>50000	>50000	>50000

For the raw net a relative small batch size generally have a better performance. Smaller batch size can quickly converge to an optimal, although it could be a local optimal and we stuck in there. That explains the epoches needed for a batch size 49 and 97 raw net are smaller on average, but still have some situation failed. The larger batch size have a greater chance to converge to the global optimal, but the speed is much slower than a smaller batch size.

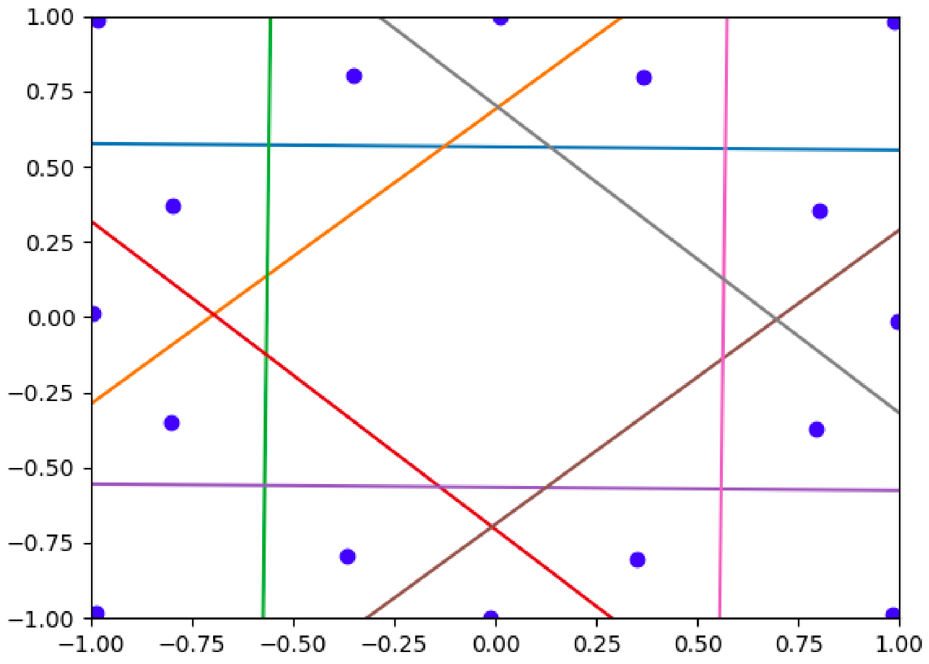
The features use raw data are much more complex than polarized data. For the polarized data, the feature is quite simple, so if we use a batch size of 194, the whole size of whole dataset, for polar net the it goes stright to global optimal and hence has a better performance. Some results for polar net with different batch size areas follow:

Epoches needed for polar net to achieve 100% accuarcy

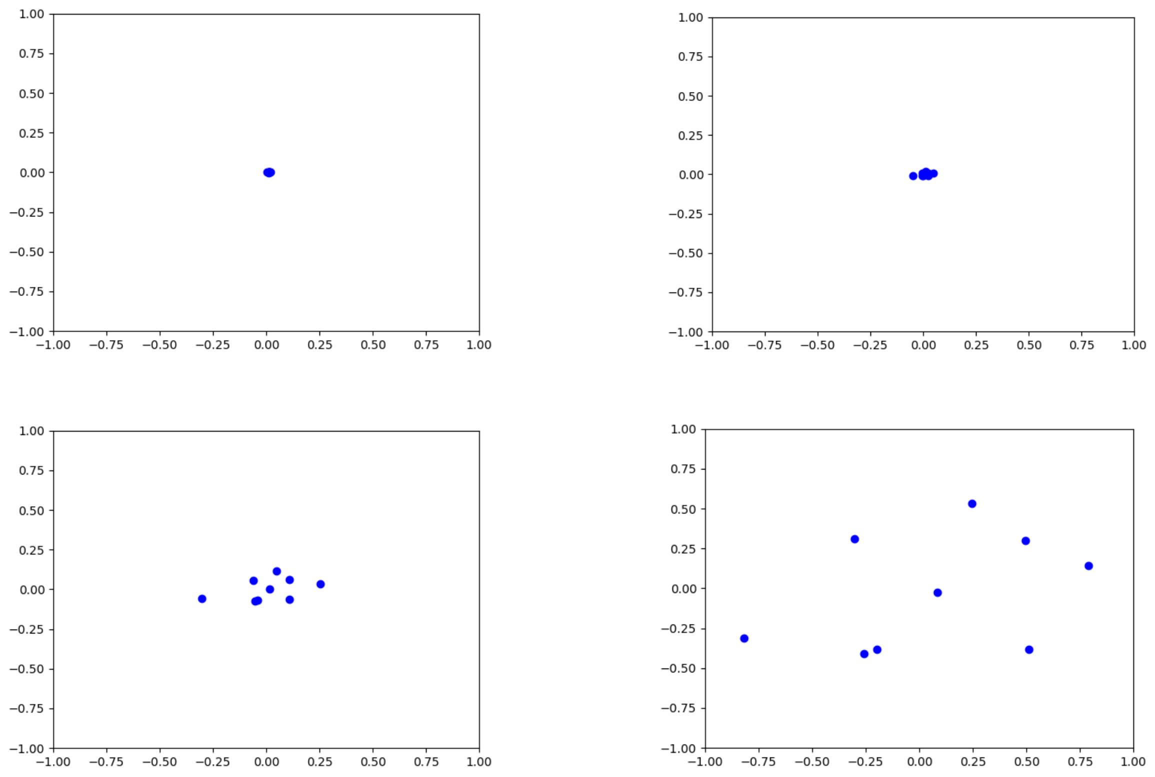
Batch size	Trail 1	Trail 2	Trail 3	Trail 4	Trail 5	Trail 6
49	2100	9600	3300	4800	2700	6300
97	4200	3500	2900	10700	3200	1600
194	800	1100	1200	1100	800	800

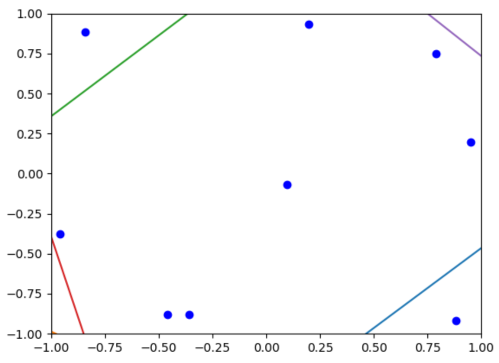
Part 3

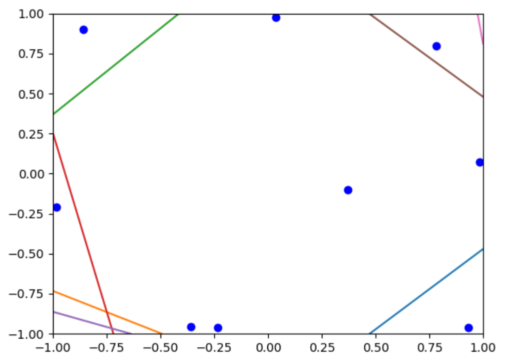
1. Star16:

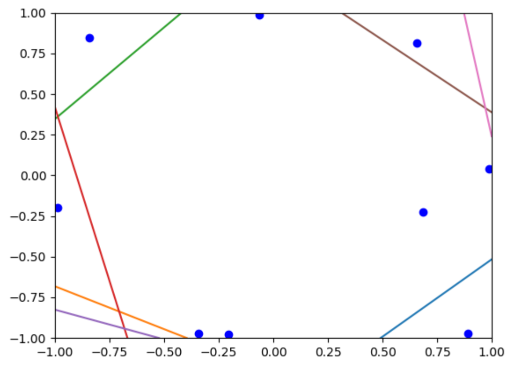


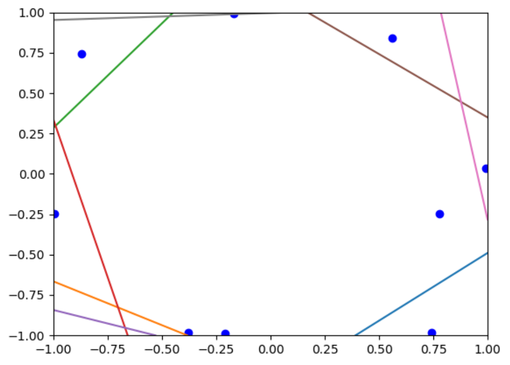
2. The dots gradually separate from each other and the lines appears trying to separate the dots. Each line is trying to separate one dots from all the other dots.

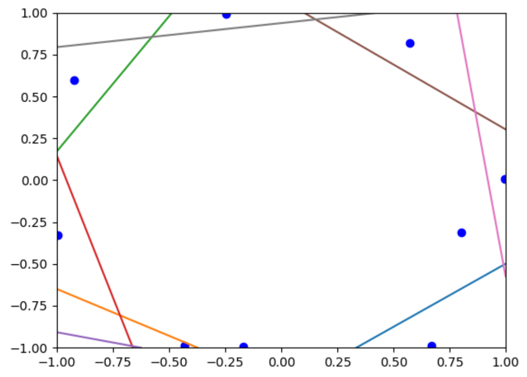


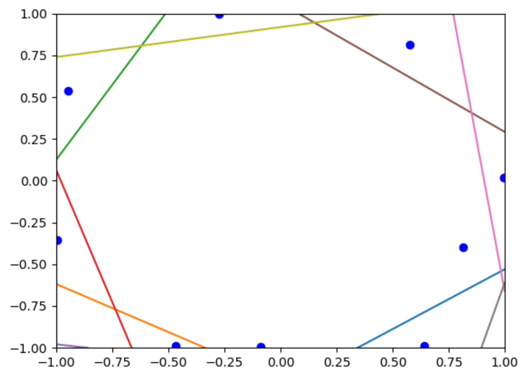

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

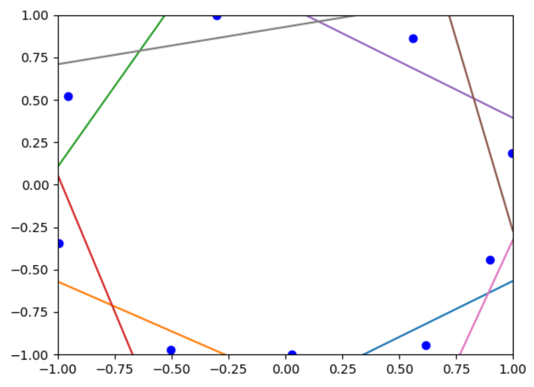

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

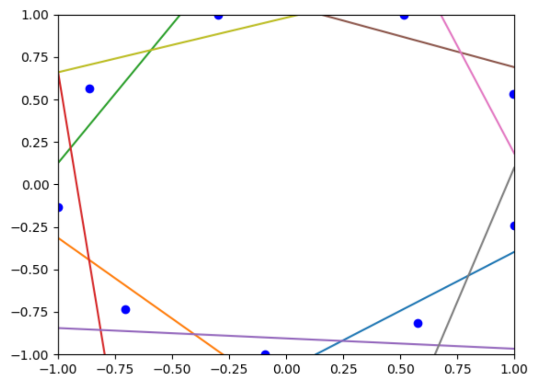

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


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

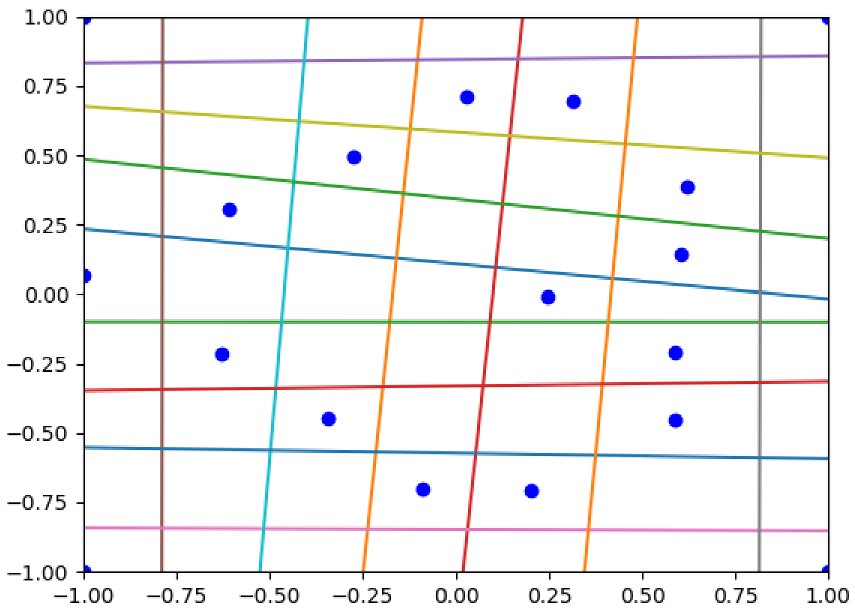

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


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

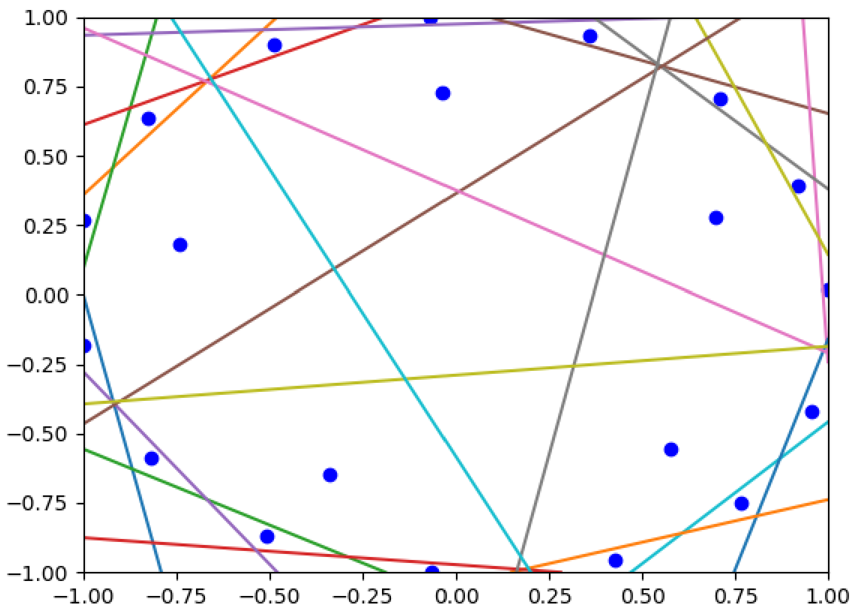

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


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

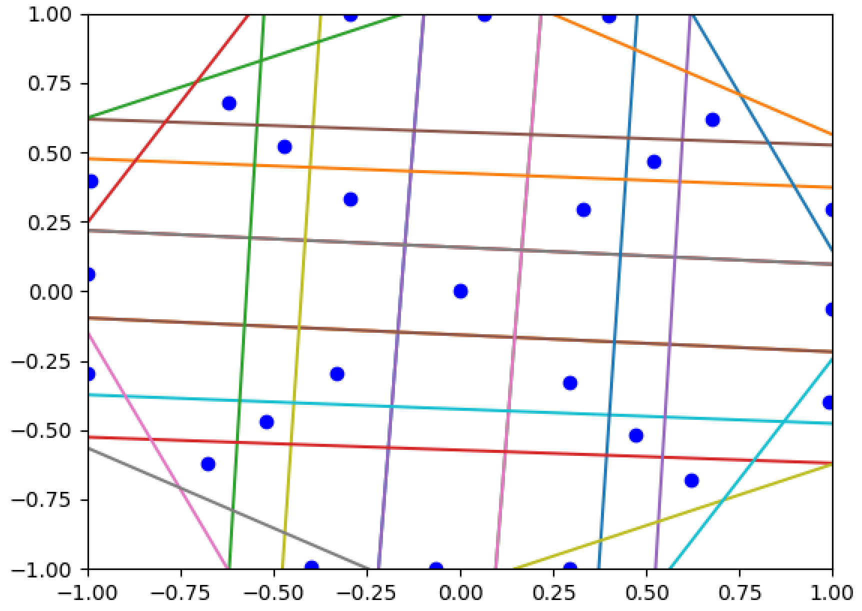
3. Heart18 :



4. Target1:



Target2:


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