COMP9444　PROJ 01

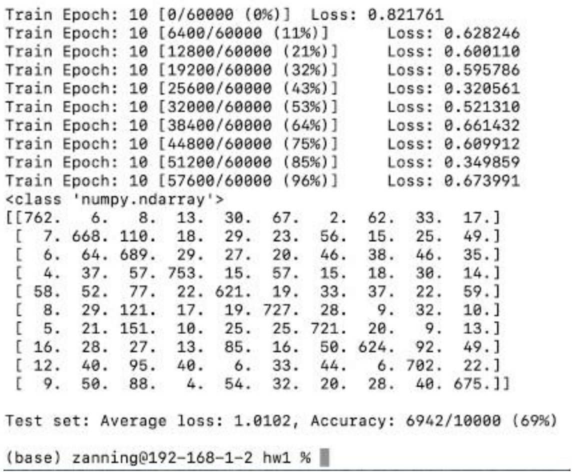
Zanning Wang　z5224151

Part1

Q1

1.

The confusion matrix show below:



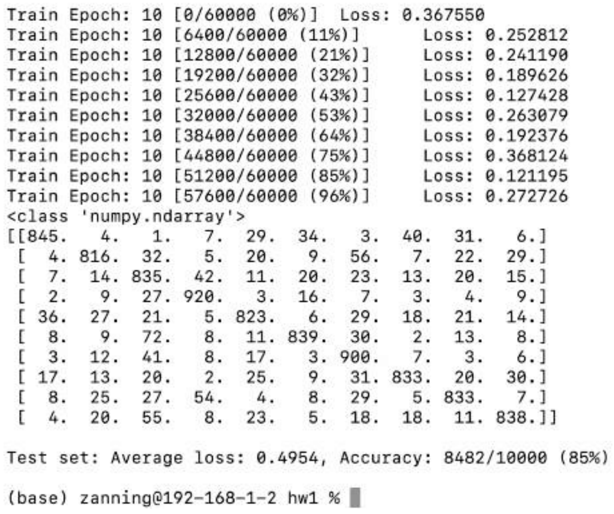
```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.821761
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.628246
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.600110
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.595786
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.320561
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.521310
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.661432
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.609912
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.349859
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.673991
<class 'numpy.ndarray'>
[[762.   6.   8.  13.  30.  67.   2.  62.  33.  17.]
 [  7. 668. 110.  18.  29.  23.  56.  15.  25.  49.]
 [  6.  64. 689.  29.  27.  20.  46.  38.  46.  35.]
 [  4.  37.  57. 753.  15.  57.  15.  18.  30.  14.]
 [ 58.  52.  77.  22. 621.  19.  33.  37.  22.  59.]
 [  8.  29. 121.  17.  19. 727.  28.   9.  32.  10.]
 [  5.  21. 151.  10.  25.  25. 721.  20.   9.  13.]
 [ 16.  28.  27.  13.  85.  16.  50. 624.  92.  49.]
 [ 12.  40.  95.  40.   6.  33.  44.   6. 702.  22.]
 [  9.  50.  88.   4.  54.  32.  20.  28.  40. 675.]]

Test set: Average loss: 1.0102, Accuracy: 6942/10000 (69%)

(base) zanning@192-168-1-2 hw1 %
```

The final accuracy is 69%.

Q2.



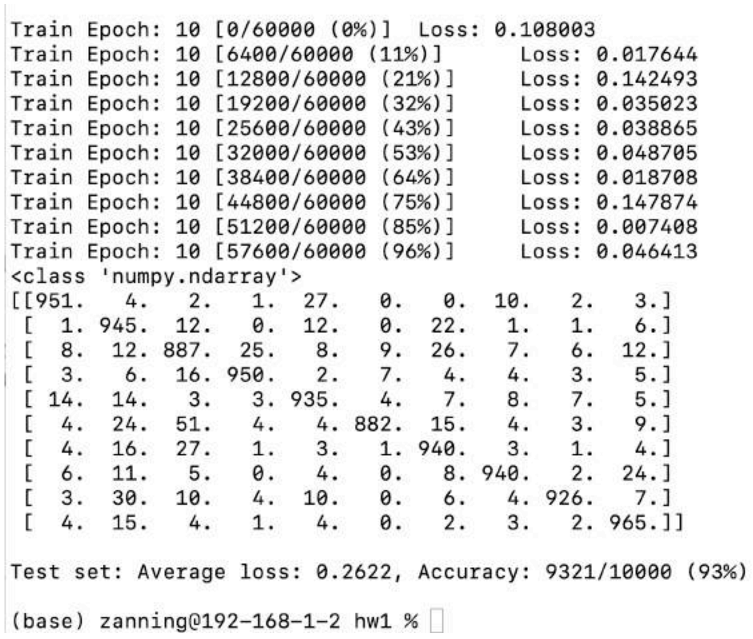
```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.367550
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.252812
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.241190
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.189626
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.127428
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.263079
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.192376
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.368124
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.121195
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.272726
<class 'numpy.ndarray'>
[[845.   4.   1.   7.  29.  34.   3.  40.  31.   6.]
 [  4. 816.  32.   5.  20.   9.  56.   7.  22.  29.]
 [  7.  14. 835.  42.  11.  20.  23.  13.  20.  15.]
 [  2.   9.  27. 920.   3.  16.   7.   3.   4.   9.]
 [ 36.  27.  21.   5. 823.   6.  29.  18.  21.  14.]
 [  8.   9.  72.   8.  11. 839.  30.   2.  13.   8.]
 [  3.  12.  41.   8.  17.   3. 900.   7.   3.   6.]
 [ 17.  13.  20.   2.  25.   9.  31. 833.  20.  30.]
 [  8.  25.  27.  54.   4.   8.  29.   5. 833.   7.]
 [  4.  20.  55.   8.  23.   5.  18.  18.  11. 838.]]

Test set: Average loss: 0.4954, Accuracy: 8482/10000 (85%)

(base) zanning@192-168-1-2 hw1 %
```

The final accuracy is 85%.

Q3.



```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.108003
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.017644
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.142493
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.035023
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.038865
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.048705
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.018708
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.147874
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.007408
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.046413
<class 'numpy.ndarray'>
[[951.   4.   2.   1.  27.   0.   0.  10.   2.   3.]
 [  1. 945.  12.   0.  12.   0.  22.   1.   1.   6.]
 [  8.  12. 887.  25.   8.   9.  26.   7.   6.  12.]
 [  3.   6.  16. 950.   2.   7.   4.   4.   3.   5.]
 [ 14.  14.   3.   3. 935.   4.   7.   8.   7.   5.]
 [  4.  24.  51.   4.   4. 882.  15.   4.   3.   9.]
 [  4.  16.  27.   1.   3.   1. 940.   3.   1.   4.]
 [  6.  11.   5.   0.   4.   0.   8. 940.   2.  24.]
 [  3.  30.  10.   4.  10.   0.   6.   4. 926.   7.]
 [  4.  15.   4.   1.   4.   0.   2.   3.   2. 965.]]

Test set: Average loss: 0.2622, Accuracy: 9321/10000 (93%)

(base) zanning@192-168-1-2 hw1 % 
```

The final accuracy is 93%.

Q4.

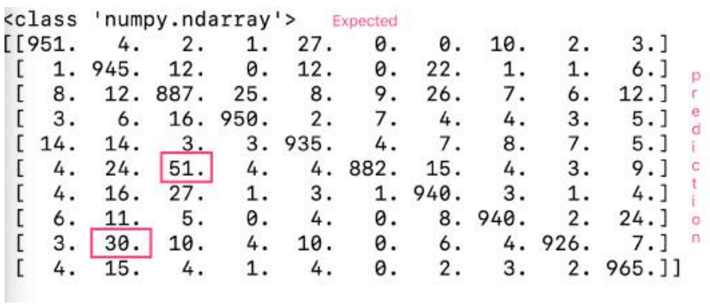
a).

In those three models, the accuracy of the first one called: “NeTLin” is the lowest because with only one layer linear function, the capacity of network is not enough to do some complex execution to inprove the efficiency and accuracy, therefore the accuracy can only reach about 70%, and not able to grow further.

For the second model, Comparing to the Netlin, the model can do more precisely separation to the classification with two layer fully connected network. By apply function such as Xor, the Janpanese characters can be classified into more proportion, which help the accuracy improve a lot, but still under fitting, the accuracy still not able to get over 85%。

For the third model, the accuracy reach about 93%, because in the convolution model, each neuron did not fully connect with the previous layer, whereas the neuron in NetLin and NetFull are fully connected with the previously layer which may contribute to the unnecessary noise from the weight between each neuron, and two convolutions layer is complex enough to tackle the complexity of graph problem, but the accuracy increase slower in the last three epoch, the accuracy may not get over 95% due to the hidden size of the model.

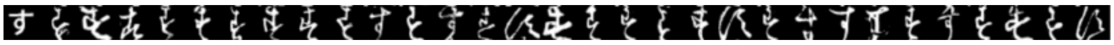
b).



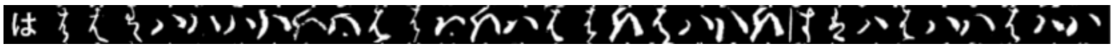
```
<class 'numpy.ndarray'>
[[951.   4.   2.   1.  27.   0.   0.  10.   2.   3.]
 [  1. 945.  12.   0.  12.   0.  22.   1.   1.   6.]
 [  8.  12. 887.  25.   8.   9.  26.   7.   6.  12.]
 [  3.   6.  16. 950.   2.   7.   4.   4.   3.   5.]
 [ 14.  14.   3.   3. 935.   4.   7.   8.   7.   5.]
 [  4.  24.  51.   4.   4. 882.  15.   4.   3.   9.]
 [  4.  16.  27.   1.   3.   1. 940.   3.   1.   4.]
 [  6.  11.   5.   0.   4.   0.   8. 940.   2.  24.]
 [  3.  30.  10.   4.  10.   0.   6.   4. 926.   7.]
 [  4.  15.   4.   1.   4.   0.   2.   3.   2. 965.]]
```

According to the confusion matrix, In “NetFull” and “NetConv”, the third category is most likely be predict as the sixth category wrongly, by check the photo of these two categories:

The third category:



The sixth category:



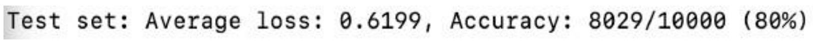
We can find for especially some of characters, these two category is look like really similar, we may need more than two convolutional network to fit the model.

c).

(1)

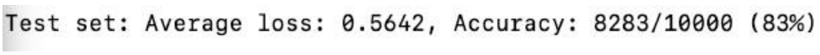
By changing the size of channels in linear function of NetFull model, if we change the channel between linear1 and linear2 as

50:



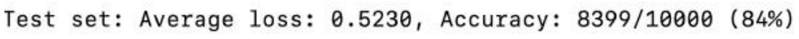
```
Test set: Average loss: 0.6199, Accuracy: 8029/10000 (80%)
```

75:



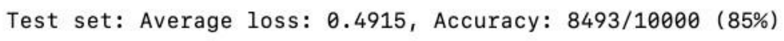
```
Test set: Average loss: 0.5642, Accuracy: 8283/10000 (83%)
```

100:



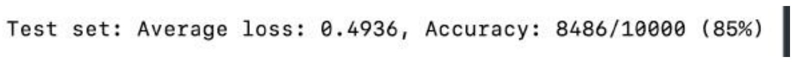
```
Test set: Average loss: 0.5230, Accuracy: 8399/10000 (84%)
```

300:



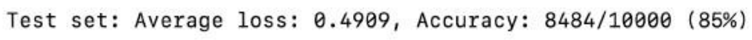
```
Test set: Average loss: 0.4915, Accuracy: 8493/10000 (85%)
```

400:



```
Test set: Average loss: 0.4936, Accuracy: 8486/10000 (85%)
```

500:



```
Test set: Average loss: 0.4909, Accuracy: 8484/10000 (85%)
```

From the figure, we can see with the increase of channel size between linear function, the accuracy keep increasing until about 200, after 200, the accuracy keep stable at 85%, by increasing the size of channel will be no help of the model, due to the complexity in this model.

(2)

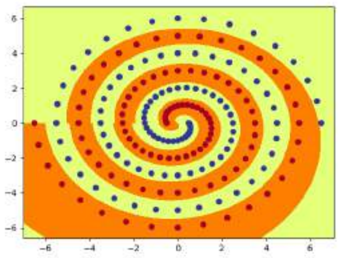
In the NetConv model, we use two convolutional layer, To prevent the model from overfitting, I introduce the maxpooling to Improve model generalization ability and Keep main features while reducing parameters. Before using the maxpooling, the accuracy show below:

```
Test set: Average loss: 0.2476, Accuracy: 9346/10000 (93%)
```

Although the maxpooling may only have no help to the accuracy, but it improved the generalization ability of the model.

Part 2

Q2.



By setting the hid from 15, the PolarNet can classify all of the training data within 1800 epochs

Hid:15

```
ep: 1600 loss: 0.0612 acc: 86.60
ep: 1700 loss: 0.0507 acc: 96.39
ep: 1800 loss: 0.0397 acc: 100.00
```

Hid:14

```
ep: 1600 loss: 0.0303 acc: 93.81
ep: 1700 loss: 0.0257 acc: 99.48
ep: 1800 loss: 0.0213 acc: 100.00
```

Hid 12

```
ep: 2600 loss: 0.0081 acc: 92.27
ep: 2700 loss: 0.0069 acc: 93.30
ep: 2800 loss: 0.0059 acc: 100.00
```

Hid10:

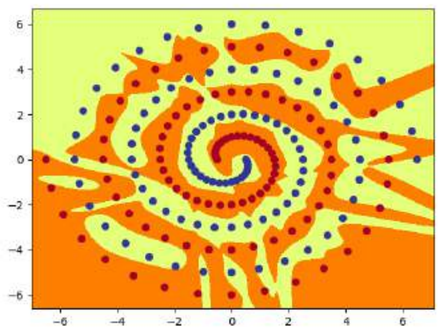
```
ep:10000 loss: 0.0357 acc: 82.47
ep:10100 loss: 0.0336 acc: 80.93
ep:10200 loss: 0.0318 acc: 81.44
ep:10300 loss: 0.0318 acc: 100.00
```

Hid 9

```
ep:19800 loss: 0.0841 acc: 76.80
ep:19900 loss: 0.0817 acc: 76.80
ep:20000 loss: 0.0795 acc: 77.84
```

The minimum number of hidden node in this model is 10.

Q4.



By setting the hid from 15 and init from 1.8, the RawNet can classify all of the training data within 1800 epochs on average.

Hid: 15, init: 0.21

```
ep: 1500 loss: 0.0393 acc: 96.39
ep: 1600 loss: 0.0291 acc: 99.48
ep: 1700 loss: 0.0204 acc: 100.00
```

Hid: 15, init: 0.11

```
ep: 4000 loss: 0.2152 acc: 96.91
ep: 4100 loss: 0.0909 acc: 98.97
ep: 4200 loss: 0.0460 acc: 100.00
```

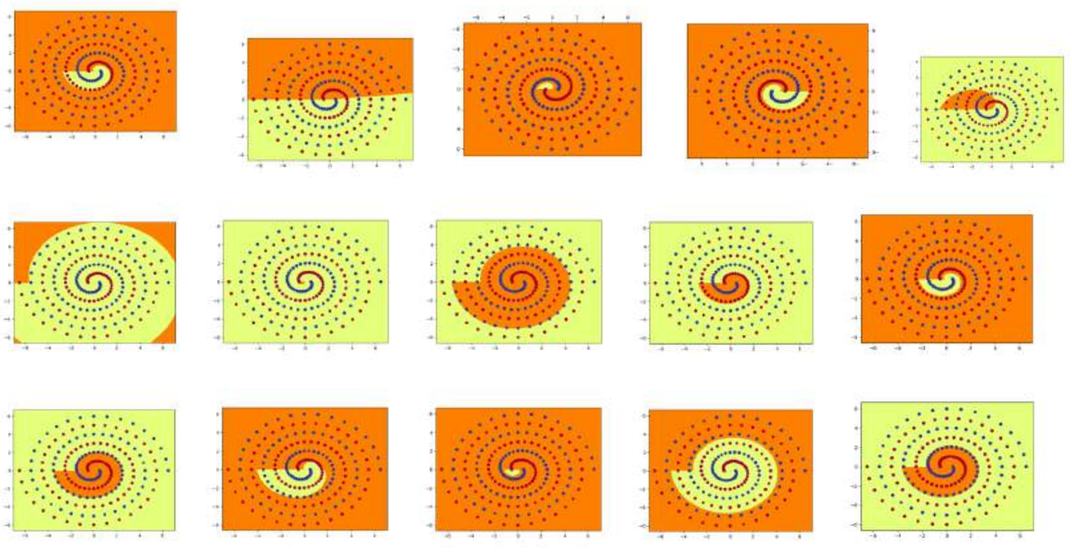
Hid: 11, init: 0.11

```
ep: 4700 loss: 0.0624 acc: 98.45
ep: 4800 loss: 0.0489 acc: 98.97
ep: 4900 loss: 0.0362 acc: 100.00
```

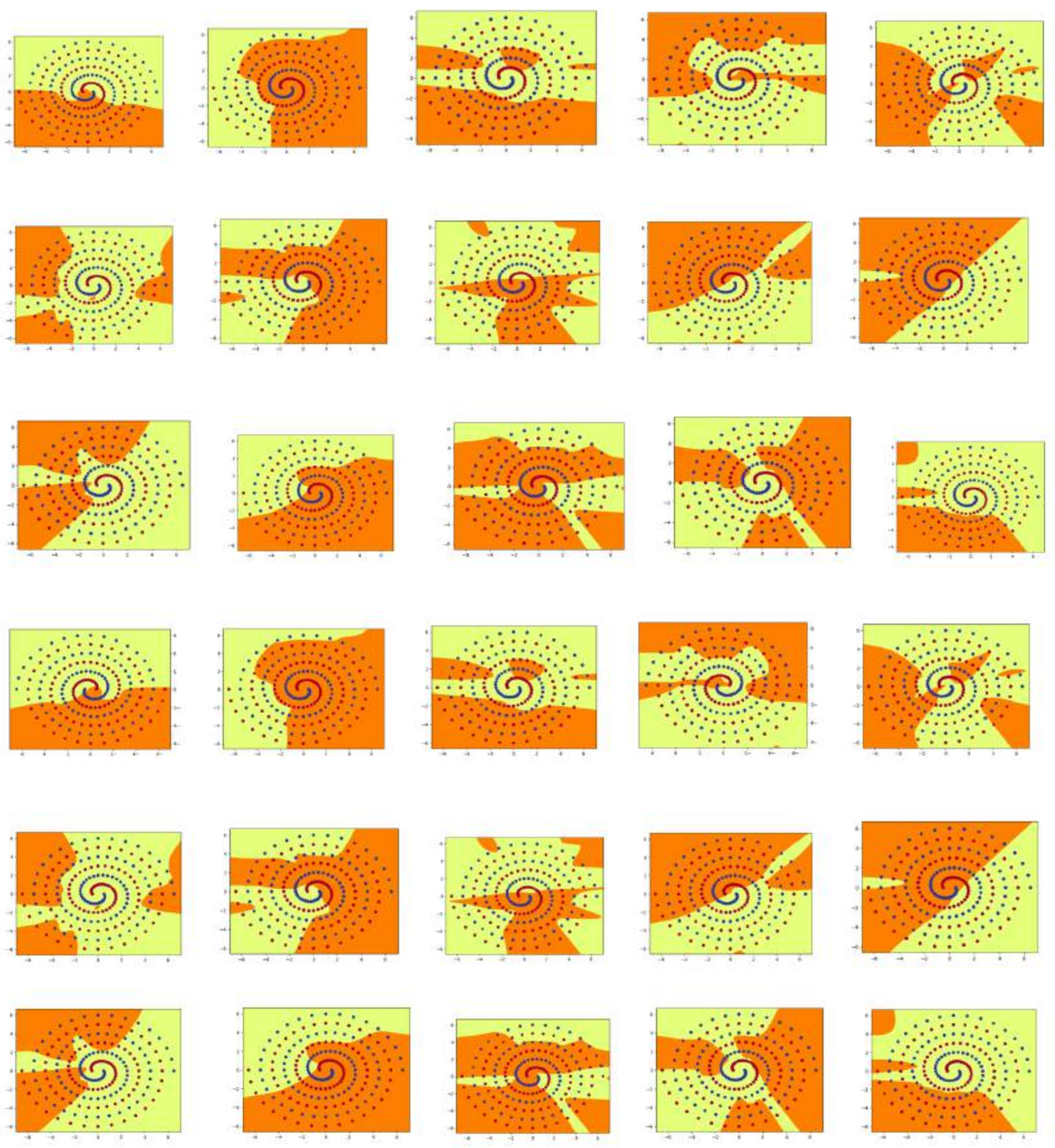
The minimum number of hidden node in this model is 11 and initial weight is 0.11.

Q5.

PolarNet:



RawNet:



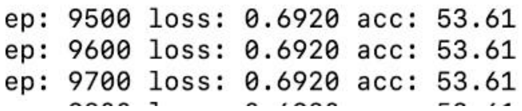
Q6.

a)

To take polar as an example, each picture show each node by hidden other node. The final output come from the sum of each weight from different node. For picture polar-1-0, the yellow part only cover a little size in the center, in this part, this node have a “heavier” weight in this part, compared to other nodes, after sum all the node’ s weight, we may get the final result of generalization.

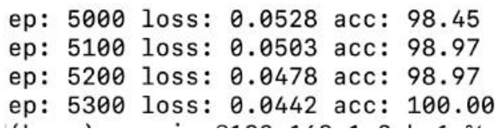
b)

for the init weight size, the minimal number is 0.11, if we set 0.10, the acc will keep stable at 53.61%:



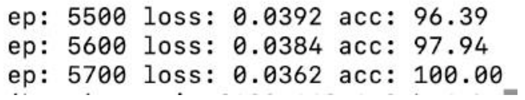
```
ep: 9500 loss: 0.6920 acc: 53.61
ep: 9600 loss: 0.6920 acc: 53.61
ep: 9700 loss: 0.6920 acc: 53.61
```

With the increase of initial weight, the epoch used to finish generalization decrease slightly, around 5500 epochs, however the size of initial weight should not over 0.25. the size of initial weight only has a little difference to the speed.
For init 0.15:



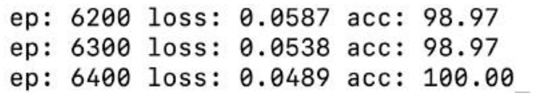
```
ep: 5000 loss: 0.0528 acc: 98.45
ep: 5100 loss: 0.0503 acc: 98.97
ep: 5200 loss: 0.0478 acc: 98.97
ep: 5300 loss: 0.0442 acc: 100.00
```

For init 0.18



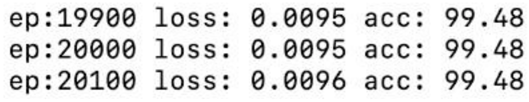
```
ep: 5500 loss: 0.0392 acc: 96.39
ep: 5600 loss: 0.0384 acc: 97.94
ep: 5700 loss: 0.0362 acc: 100.00
```

For init 0.22



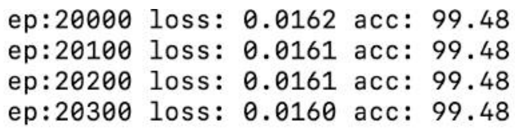
```
ep: 6200 loss: 0.0587 acc: 98.97
ep: 6300 loss: 0.0538 acc: 98.97
ep: 6400 loss: 0.0489 acc: 100.00
```

For init 0.25



```
ep:19900 loss: 0.0095 acc: 99.48
ep:20000 loss: 0.0095 acc: 99.48
ep:20100 loss: 0.0096 acc: 99.48
```

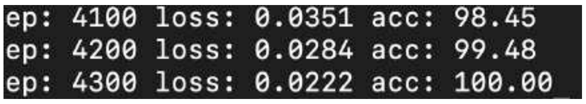
For init 0.30



```
ep:20000 loss: 0.0162 acc: 99.48
ep:20100 loss: 0.0161 acc: 99.48
ep:20200 loss: 0.0161 acc: 99.48
ep:20300 loss: 0.0160 acc: 99.48
```

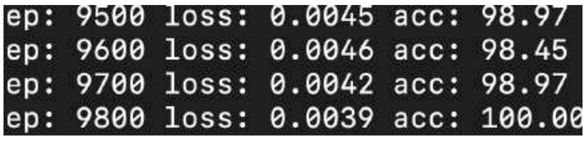
c)

(1) change tanh into relu
Under the condition: hid: 15, init: 0.15
Tanh

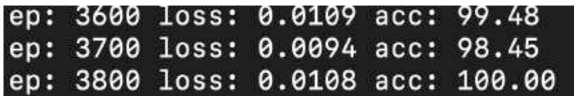


```
ep: 4100 loss: 0.0351 acc: 98.45
ep: 4200 loss: 0.0284 acc: 99.48
ep: 4300 loss: 0.0222 acc: 100.00
```

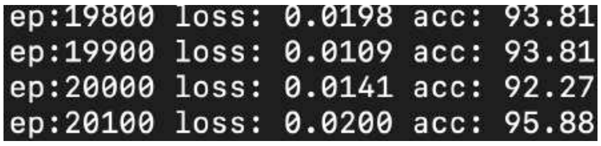
RELU



```
ep: 9500 loss: 0.0045 acc: 98.97
ep: 9600 loss: 0.0046 acc: 98.45
ep: 9700 loss: 0.0042 acc: 98.97
ep: 9800 loss: 0.0039 acc: 100.00
```

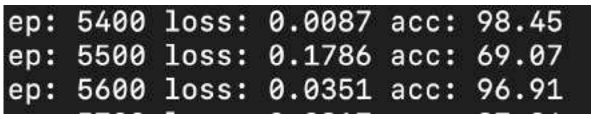


```
ep: 3600 loss: 0.0109 acc: 99.48
ep: 3700 loss: 0.0094 acc: 98.45
ep: 3800 loss: 0.0108 acc: 100.00
```



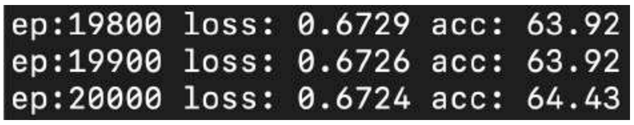
```
ep:19800 loss: 0.0198 acc: 93.81
ep:19900 loss: 0.0109 acc: 93.81
ep:20000 loss: 0.0141 acc: 92.27
ep:20100 loss: 0.0200 acc: 95.88
```

According to the figure above, RELU takes 3800-9800 epoch to finish the job, sometime the RELU may not finish the classification under 20000 epochs, whereas the Tanh is more stable to find the best classification around 5000 epoch, another strange thing is with the help of Relu, the accuracy of some epochs get decreased significantly, in the model which use tanh will not happen things like that.



```
ep: 5400 loss: 0.0087 acc: 98.45
ep: 5500 loss: 0.1786 acc: 69.07
ep: 5600 loss: 0.0351 acc: 96.91
```

(2) change Adam into SGD

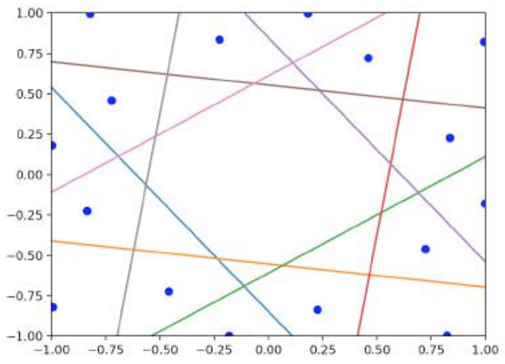


```
ep:19800 loss: 0.6729 acc: 63.92
ep:19900 loss: 0.6726 acc: 63.92
ep:20000 loss: 0.6724 acc: 64.43
```

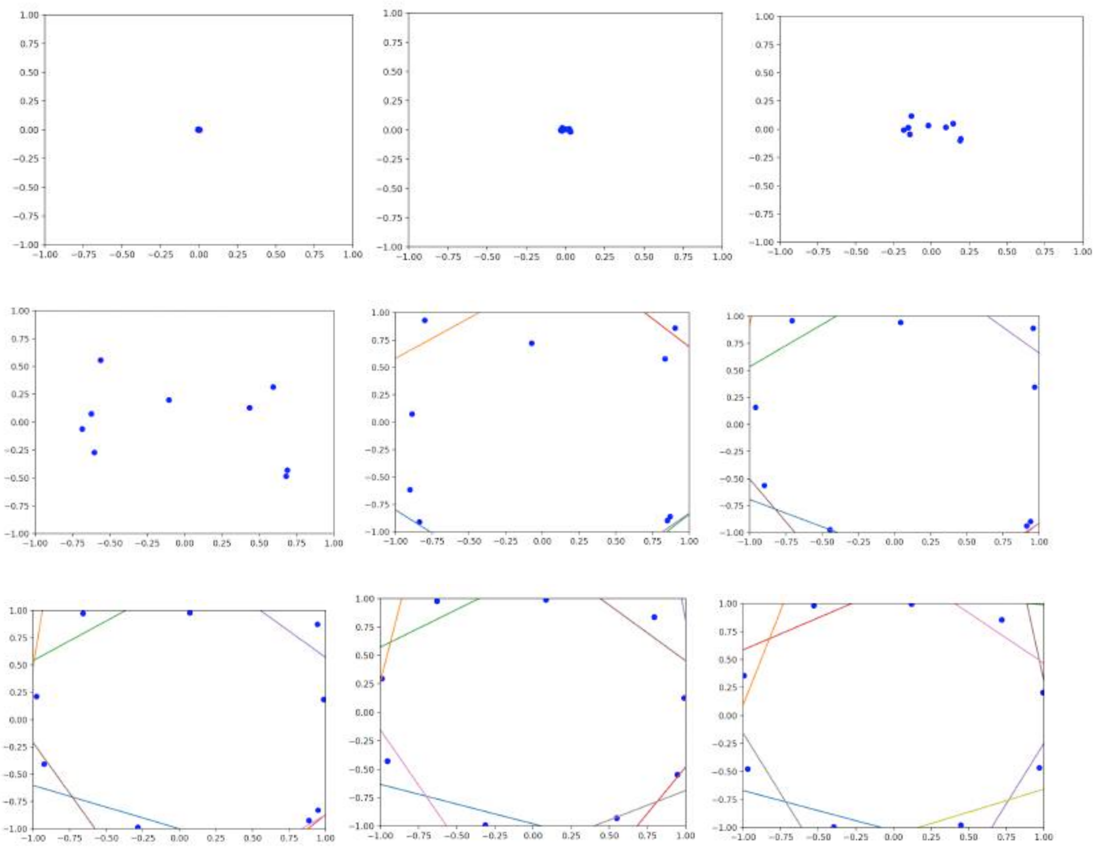
By using the SGD optimizer, under the condition: init 0.15, hid 15, the accuracy can only reach about 65.46%. though the adam is more suitable than SGD in separate the dots.

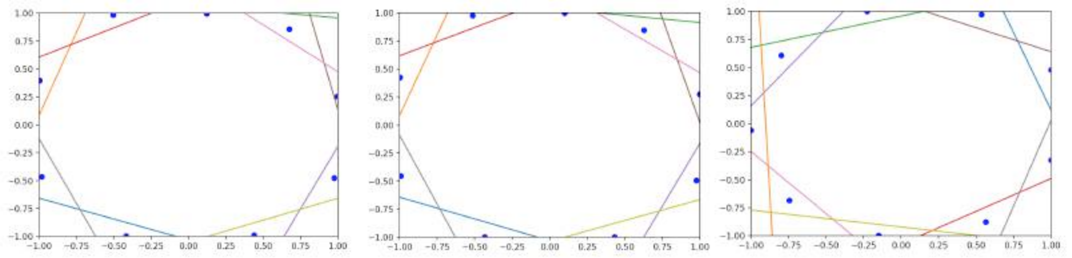
Part 3

Q1.



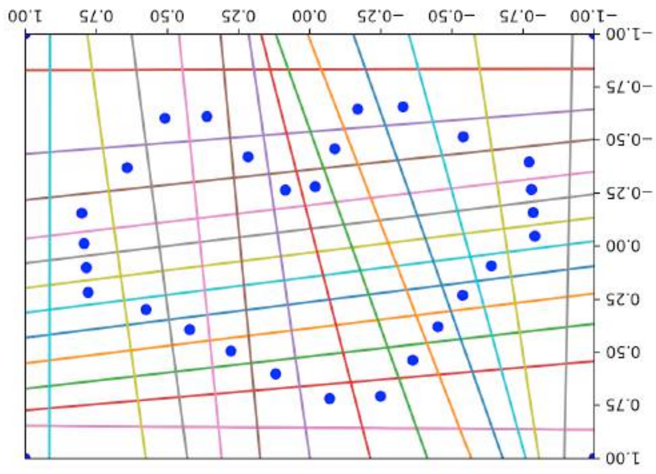
Q2.





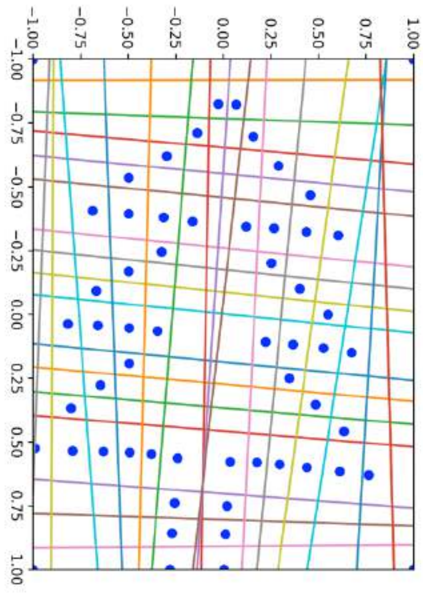
The hidden units move from the center to the border gradually, at the same time the boundary move outward with rotation, sometimes the boundary move to the opposite direction.

Q3.



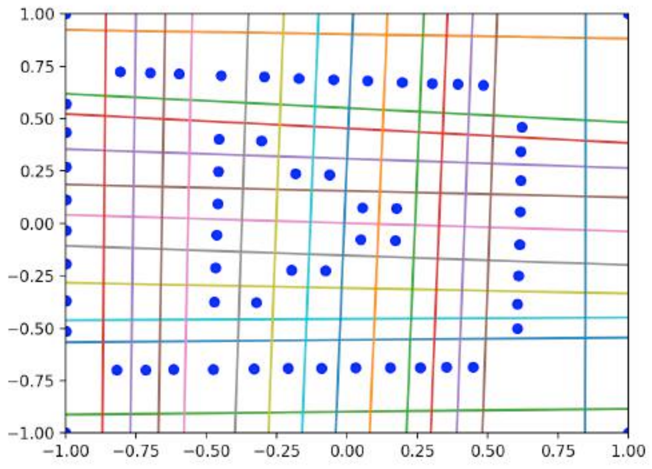
Q4.

Target1:



I paint a tree here, just in case, nobody can recognize it :)

Target2:



Here is a logo of Youtube , just in case, nobody can recognize it :)