COMP9444 Neural Networks and Deep Learning HOMEWORK 1 REPORT

Name : Vidyadhari Prerepa zID : z5284443

Part 1: Japanese Character Recognition

1. Linear Neural Network final accuracy and confusion matrix

```
Train Epoch: 10 [0/60000 (0%)] Loss: 0.817970
Train Epoch: 10 [6400/60000 (11%)]    Loss: 0.638252
Train Epoch: 10 [12800/60000 (21%)]   Loss: 0.594914
Train Epoch: 10 [19200/60000 (32%)]   Loss: 0.600996
Train Epoch: 10 [25600/60000 (43%)]   Loss: 0.321217
Train Epoch: 10 [32000/60000 (53%)]   Loss: 0.515749
Train Epoch: 10 [38400/60000 (64%)]   Loss: 0.656161
Train Epoch: 10 [44800/60000 (75%)]   Loss: 0.610840
Train Epoch: 10 [51200/60000 (85%)]   Loss: 0.347270
Train Epoch: 10 [57600/60000 (96%)]   Loss: 0.669514
<class 'numpy.ndarray'>
[[770.   5.   8.  12.  30.  63.   2.  61.  31.  18.]
 [  7. 672. 105.  18.  27.  24.  59.  14.  24.  50.]
 [  8.  62. 694.  26.  25.  21.  47.  37.  43.  37.]
 [  5.  37.  57. 759.  16.  56.  14.  17.  28.  11.]
 [ 61.  53.  81.  20. 621.  18.  32.  36.  21.  57.]
 [  8.  27. 123.  17.  19. 727.  28.   7.  32.  12.]
 [  5.  24. 148.  11.  24.  23. 722.  20.   9.  14.]
 [ 14.  28.  28.  13.  85.  16.  54. 624.  89.  49.]
 [ 11.  36.  95.  43.   7.  28.  44.   6. 707.  23.]
 [  9.  53.  85.   3.  49.  31.  19.  30.  41. 680.]]

  Test set: Average loss: 1.0098, Accuracy: 6976/10000 (70%)
```

2. Fully Connected 2 layer Neural Network final accuracy and confusion matrix

```
Train Epoch: 10 [0/60000 (0%)] Loss: 0.343665
Train Epoch: 10 [6400/60000 (11%)]    Loss: 0.235194
Train Epoch: 10 [12800/60000 (21%)]   Loss: 0.239950
Train Epoch: 10 [19200/60000 (32%)]   Loss: 0.193727
Train Epoch: 10 [25600/60000 (43%)]   Loss: 0.108565
Train Epoch: 10 [32000/60000 (53%)]   Loss: 0.221084
Train Epoch: 10 [38400/60000 (64%)]   Loss: 0.220378
Train Epoch: 10 [44800/60000 (75%)]   Loss: 0.321958
Train Epoch: 10 [51200/60000 (85%)]   Loss: 0.127583
Train Epoch: 10 [57600/60000 (96%)]   Loss: 0.277473
<class 'numpy.ndarray'>
[[849.   6.   2.   7.  25.  31.   3.  40.  31.   6.]
 [  5. 821.  29.   2.  16.  11.  67.   3.  18.  28.]
 [  9.  10. 841.  34.  10.  17.  28.  17.  20.  14.]
 [  4.   7.  32. 917.   1.  14.   8.   1.   8.   8.]
 [ 36.  24.  16.   5. 828.   8.  33.  18.  20.  12.]
 [  7.  11.  76.   8.  13. 836.  26.   2.  14.   7.]
 [  3.  14.  48.   8.  18.   6. 887.   4.   3.   9.]
 [ 23.  16.  13.   4.  21.   8.  34. 822.  24.  35.]
 [ 10.  25.  33.  46.   4.   9.  27.   4. 833.   9.]
 [  3.  20.  44.   6.  28.   5.  22.  12.  10. 850.]]

  Test set: Average loss: 0.4960, Accuracy: 8484/10000 (85%)
```

3. Convolutional Neural Network final accuracy and confusion matrix

```
Train Epoch: 10 [0/60000 (0%)] Loss: 0.016178
Train Epoch: 10 [6400/60000 (11%)]    Loss: 0.010429
Train Epoch: 10 [12800/60000 (21%)]   Loss: 0.067328
Train Epoch: 10 [19200/60000 (32%)]   Loss: 0.029859
Train Epoch: 10 [25600/60000 (43%)]   Loss: 0.022386
Train Epoch: 10 [32000/60000 (53%)]   Loss: 0.039307
Train Epoch: 10 [38400/60000 (64%)]   Loss: 0.017322
Train Epoch: 10 [44800/60000 (75%)]   Loss: 0.084175
Train Epoch: 10 [51200/60000 (85%)]   Loss: 0.015449
Train Epoch: 10 [57600/60000 (96%)]   Loss: 0.007064
<class 'numpy.ndarray'>
[[972.   2.   0.   0.  15.   1.   0.   6.   2.   2.]
 [  3. 919.   6.   0.  11.   2.  42.   5.   6.   6.]
 [ 12.   5. 916.  30.   4.   6.  10.   7.   4.   6.]
 [  1.   3.  15. 964.   0.   5.   9.   2.   1.   0.]
 [ 22.  11.   3.   6. 936.   1.   8.   3.   8.   2.]
 [  3.   5.  33.   1.   5. 921.  15.   4.   2.  11.]
 [  4.   2.  13.   1.   2.   5. 973.   0.   0.   0.]
 [  8.   0.   2.   1.   3.   0.   7. 962.   3.  14.]
 [  1.  14.   9.   2.   7.   2.   6.   3. 952.   4.]
 [  5.   5.   8.   0.   3.   0.   3.   4.   2. 970.]]

Test set: Average loss: 0.2146, Accuracy: 9485/10000 (95%)
```

4.

a. Linear Neural Network accuracy – 70%
Fully connected 2 layer Neural Network accuracy – 85%
Convolutional Neural Network accuracy – 95%
Upon observing the accuracy percentages and confusion matrices, we can conclude that simple linear neural network (that computes weighted sum of inputs with no activation function) does not perform well in recognizing input images and classifying them into 10 classes(0-9). On the other hand, Convolutional neural network gives the best performance by recognizing input characters and classifying them correctly up to 95%.With an accuracy of 85% a fully connected neural network shows average performance.

b. Let us consider the following confusion matrix of Convolutional Neural Network. The rows indicate the target character and columns indicate the character chosen by the network. Highlighted (in yellow) are the number of times a target character has been misclassified as some other character. The maximum number in each row is the number of times a character has been classified correctly, that is a *True Positive* classification.

Rows ⇩ Columns ⇨	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	972	2	0	0	15	1	0	6	2	2
1	3	919	6	0	11	2	42	5	6	6
2	12	5	916	30	4	6	10	7	4	6
3	1	3	15	964	0	5	9	2	1	0
4	22	11	3	6	936	1	8	3	8	2
5	3	5	33	1	5	921	15	4	2	11
6	4	2	13	1	2	5	973	0	0	0
7	8	0	2	1	3	0	7	962	3	14
8	1	14	9	2	7	2	6	3	952	4

9	5	5	8	0	3	0	3	4	2	970

Following contains highlighted misclassifications by the fully connected neural network.

Rows ⇩ Columns ⇨	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	849	6	2	7	25	31	3	40	31	6
1	5	821	29	2	16	11	67	3	18	28
2	9	10	841	34	10	17	28	17	20	14
3	4	7	32	917	1	14	8	1	8	8
4	36	24	16	5	828	8	33	18	20	12
5	7	11	76	8	13	836	26	2	14	7
6	3	14	48	8	18	6	887	4	3	9
7	23	16	13	4	21	8	34	822	24	35
8	10	25	33	46	4	9	27	4	833	9
9	3	20	44	6	28	5	22	12	10	850

Following contains highlighted misclassifications by the Linear neural network.

Rows ⇩ Columns ⇨	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	770	5	8	12	30	63	2	61	31	18
1	7	672	105	18	27	24	59	14	24	50
2	8	62	694	26	25	21	47	37	43	37
3	5	37	57	759	16	56	14	17	28	11
4	61	53	81	20	621	18	32	36	21	57
5	8	27	123	17	19	727	28	7	32	12
6	5	24	148	11	24	23	722	20	9	14
7	14	28	28	13	85	16	54	624	89	49
8	11	36	95	43	7	28	44	6	707	23
9	9	53	85	3	49	31	10	30	41	680

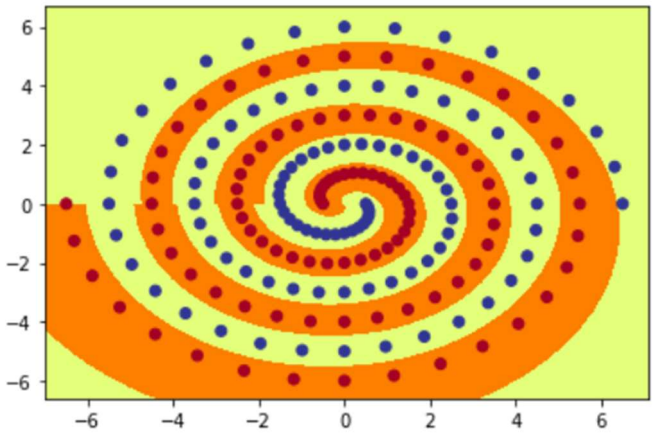
By observing the above 3 tables, it can be concluded that 1st (ki) character is mistaken as 6th (ma), 5th (ha) is mistaken as 2nd (su) and 6th is mistaken as 2nd. By looking at the following characters, we can understand that there is similarity between "ki" and "ma", "ha" and "su" due to which this misclassification is being done by the Neural Network.

c. In deep learning, to establish an optimal model using Neural Networks the meta parameters play a vital role(learning rate and momentum). Lower learning rate may slow down the learning process and higher learning rates may deviate towards overfitting. Momentum can help in accelerating the training process. An optimal combination of learning rate and momentum may lead to a better model.
I have experimented the linear model with learning rates (0.01, 0.03, 0.05, 0.07, 0.09). As the learning rate increases, it can be observed that the accuracy decreases from the optimal 70% (learning rate = 0.01).
For momentum(ranges from 0 to 1), I have tested the convolutional neural network for values 0.1, 0.3, 0.5,0.7,0.9. High momentum of 0.9, when applied to Convolution neural network gives an accuracy of 96 % ! The result is as shown below:

```
Train Epoch: 10 [0/60000 (0%)] Loss: 0.000411
Train Epoch: 10 [6400/60000 (11%)]    Loss: 0.000275
Train Epoch: 10 [12800/60000 (21%)]   Loss: 0.001194
Train Epoch: 10 [19200/60000 (32%)]   Loss: 0.003898
Train Epoch: 10 [25600/60000 (43%)]   Loss: 0.001685
Train Epoch: 10 [32000/60000 (53%)]   Loss: 0.000791
Train Epoch: 10 [38400/60000 (64%)]   Loss: 0.000173
Train Epoch: 10 [44800/60000 (75%)]   Loss: 0.003464
Train Epoch: 10 [51200/60000 (85%)]   Loss: 0.000166
Train Epoch: 10 [57600/60000 (96%)]   Loss: 0.000059
<class 'numpy.ndarray'>
[[962.   3.   2.   0.  19.   3.   0.   6.   3.   2.]
 [  3. 955.  10.   2.   3.   2.  17.   1.   4.   3.]
 [  9.   4. 944.  23.   4.   6.   4.   3.   1.   2.]
 [  0.   0.  13. 982.   0.   3.   2.   0.   0.   0.]
 [ 16.   4.   3.  14. 948.   3.   5.   1.   5.   1.]
 [  1.   4.  26.   5.   1. 952.   4.   1.   3.   3.]
 [  3.   6.  27.   3.   2.   5. 952.   0.   1.   1.]
 [  9.   4.   5.   1.   2.   0.   3. 965.   5.   6.]
 [  3.   4.   5.   3.   5.   0.   0.   0. 980.   0.]
 [  4.   3.  13.   1.   2.   0.   1.   3.   6. 967.]]

Test set: Average loss: 0.2210, Accuracy: 9607/10000 (96%)
```

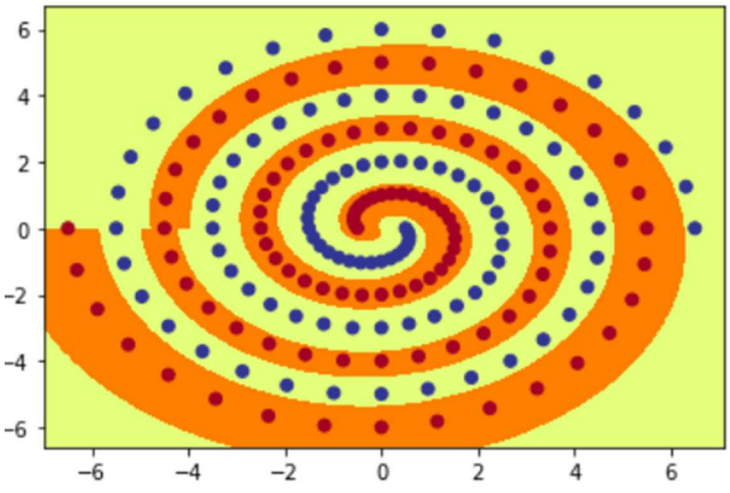
Part 2: Twin Spirals Task

1.A fully connected neural network with one hidden layer (tanh activation) has been implemented that produces a single output using sigmoid activation function. Below is the obtained output,

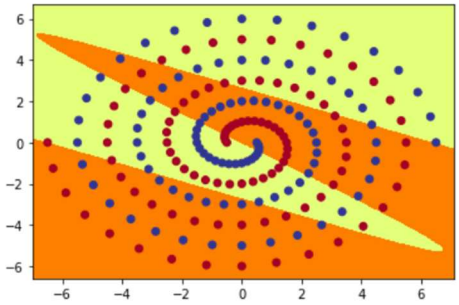


2.When I tried with number of hidden nodes = 5, the program never converges and gets stuck at an accuracy of 69.59% which is very low. When I increase the number of hidden nodes to 7, the program still gets stuck at an accuracy of 77.84% even after epoch 30800. Then I incremented the number to 8 and the program converged with 100% accuracy at epoch 6500 (<20000). Hence the ***minimum number of hidden nodes required for the PolarNet to correctly classify all training data within 20000 epochs is 8***. The accuracy output screenshot and classified image for number of hidden nodes = 8 is shown below:

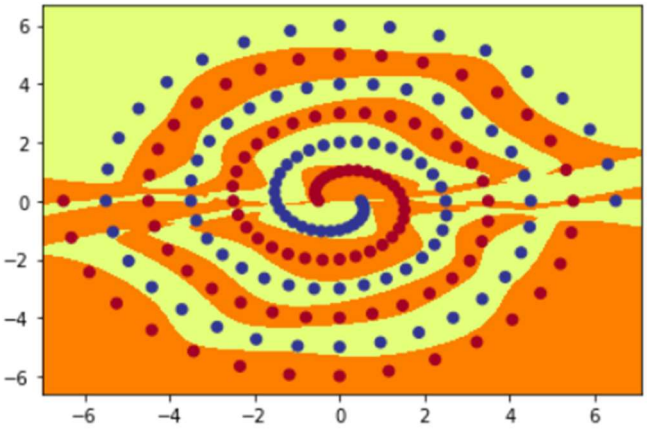
```
ep: 5300 loss: 0.0180 acc: 82.47
ep: 5400 loss: 0.0137 acc: 82.99
ep: 5500 loss: 0.0125 acc: 82.99
ep: 5600 loss: 0.0139 acc: 83.51
ep: 5700 loss: 0.0131 acc: 83.51
ep: 5800 loss: 0.0127 acc: 83.51
ep: 5900 loss: 0.0125 acc: 84.02
ep: 6000 loss: 0.0124 acc: 84.02
ep: 6100 loss: 0.0123 acc: 84.02
ep: 6200 loss: 0.0122 acc: 84.02
ep: 6300 loss: 0.0121 acc: 89.18
ep: 6400 loss: 0.0116 acc: 91.24
ep: 6500 loss: 0.0191 acc: 100.00
```



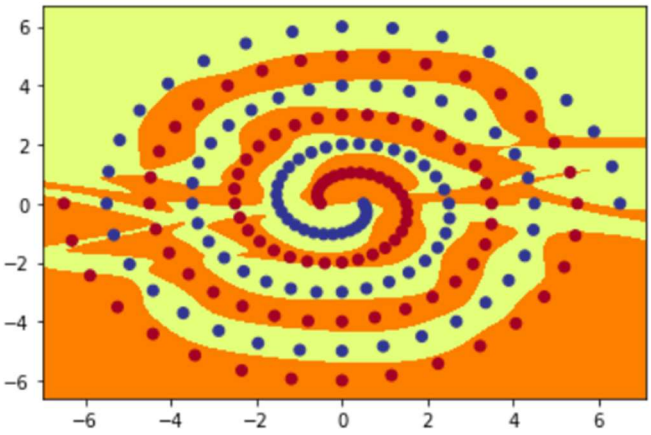
3. A fully connected neural network has been implemented with two hidden layers (tanh activation) and one output layer (sigmoid activation). Both the hidden layers were run with number of hidden nodes = 10 and the neural network produces an output with accuracy = 53.61 % which is very less. Output figure is shown below (after maximum number of epochs reached as in the given code the process is iterated till epochs = 100000) :



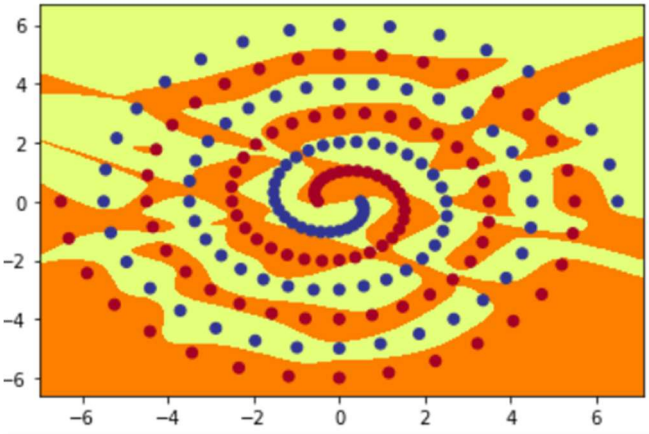
4. By setting the number of hidden nodes to 10 and initial weight = 0.2, RawNet does converge with 100% accuracy but the classification done by RawNet does not seem to be perfect. Then I have tried by increasing the hidden nodes from 10 to 13 and a similar observation can be made as with hidden nodes = 10 but this time the classification seems to be slightly better than earlier. When I fix the hidden nodes to 11 and change initial weights to 0.2, the result is still better. The classification seems to be satisfactory when hidden nodes = 11 and initial weights = 0.12. However, since we are told to choose the number of hidden nodes between 5 and 10, and initial weights as 0.001, 0.1, 0.2, 0.3, the only combination where the network seems to converge is when **hidden nodes = 10 and initial weight = 0.2**. All converged results are shown below (all obtained in epochs less than 20000):



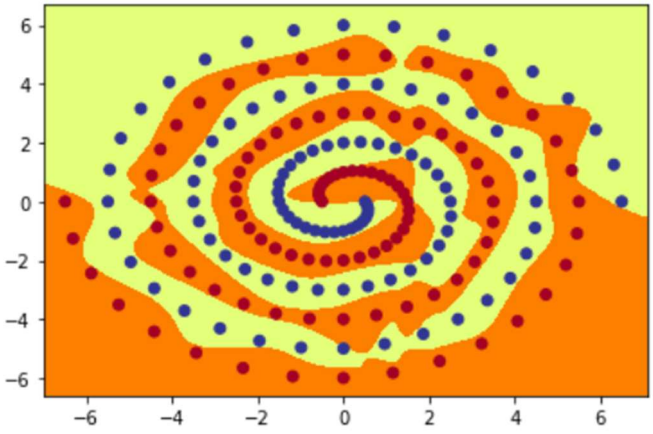
Hidden nodes = 10, initial weight = 0.2



Hidden nodes = 13



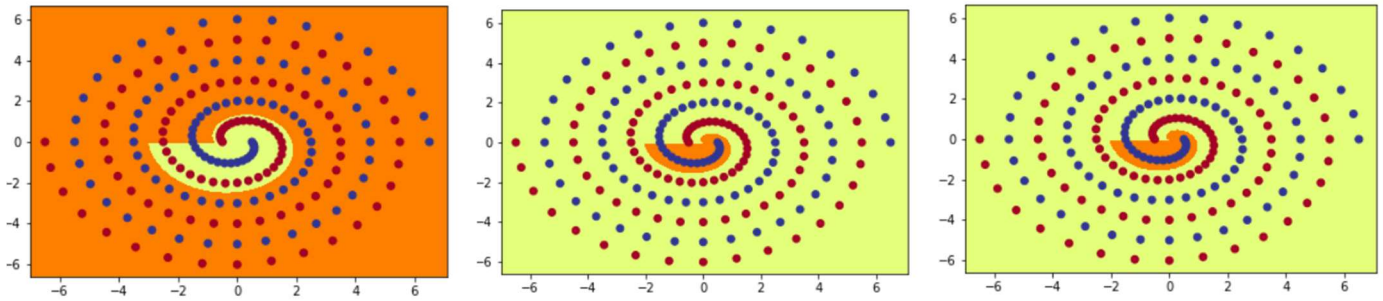
Hidden nodes = 11 and initial weights = 0.2

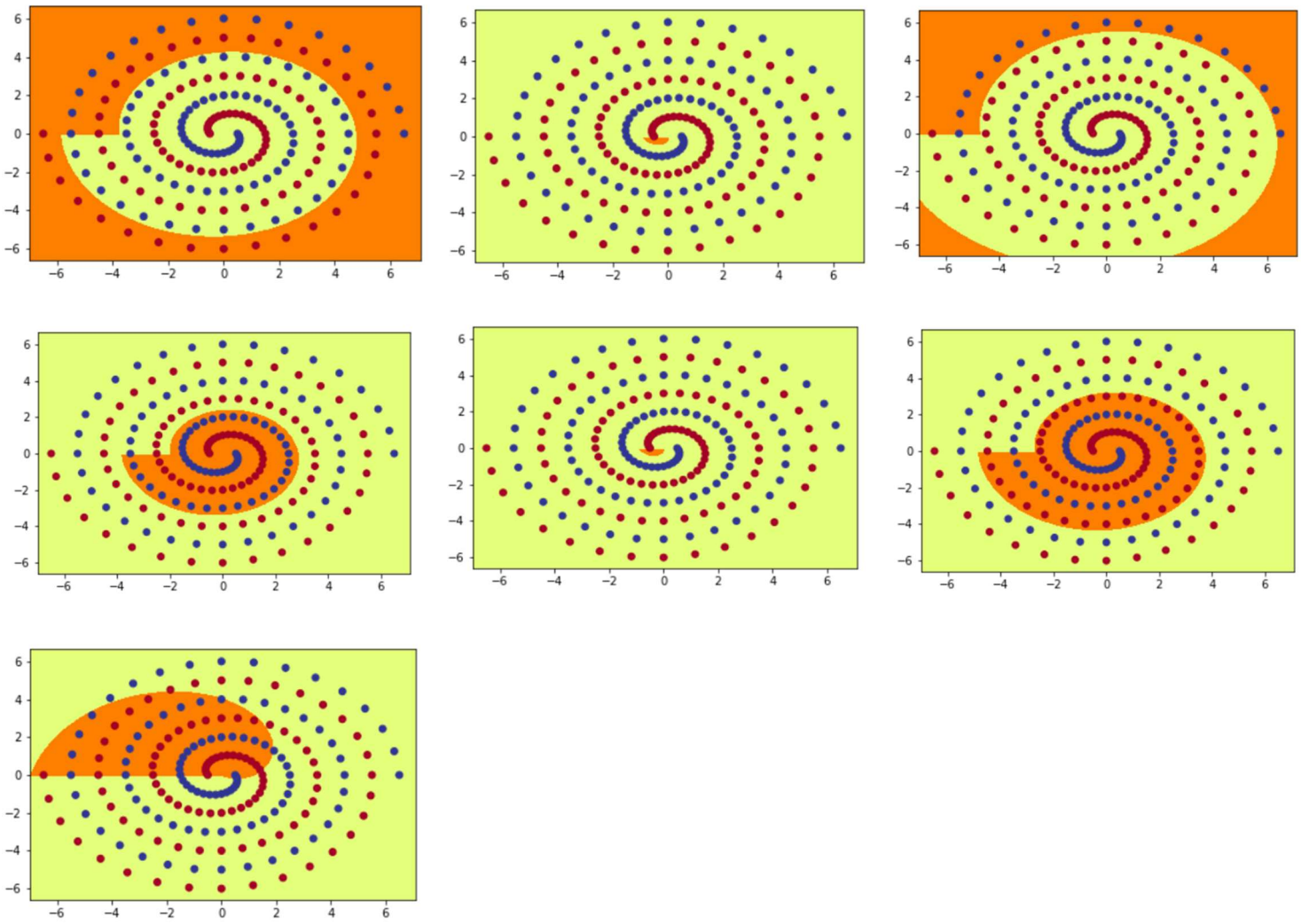


Hidden nodes = 11 and initial weights = 0.12

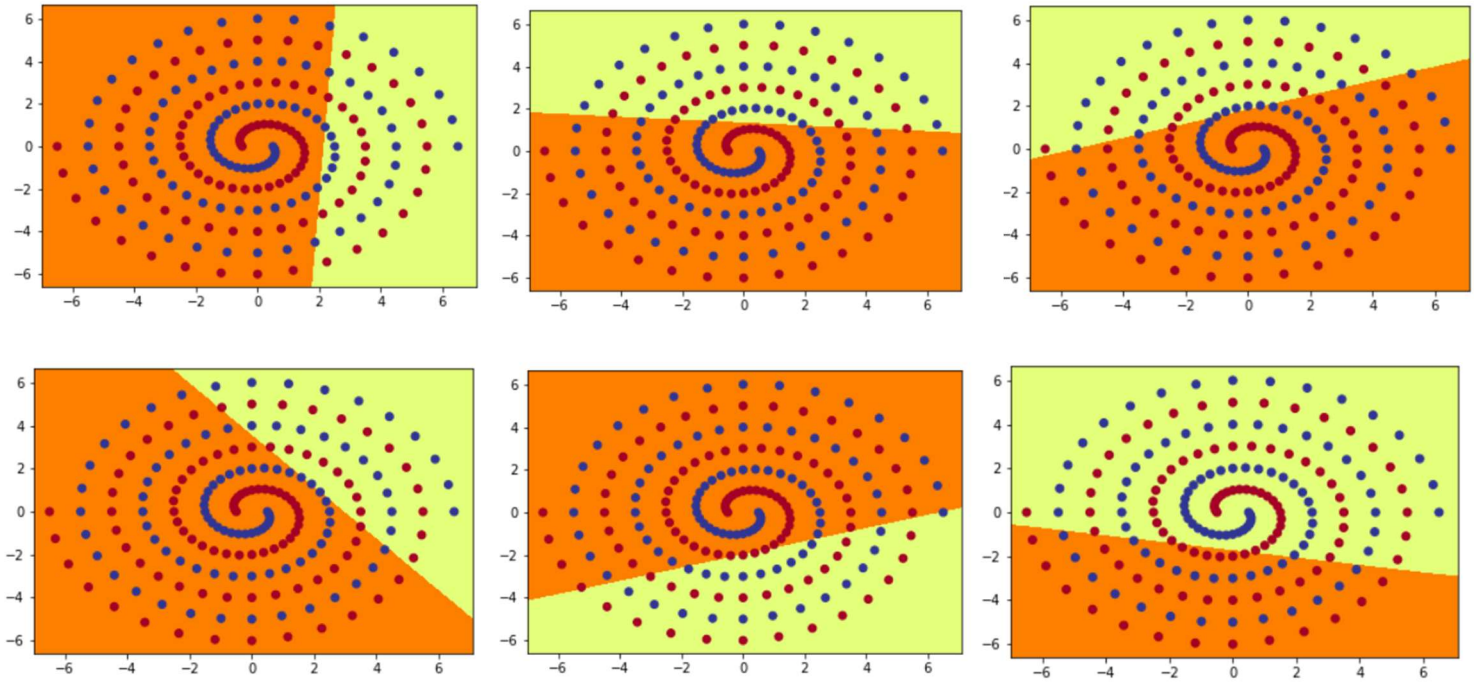
5. "graph_hidden" method has been implemented which runs separately for PolarNet and RawNet which have different number of hidden layers. Number of hidden nodes has been set the default value that is 10.

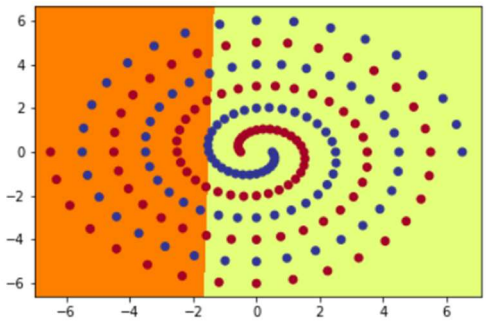
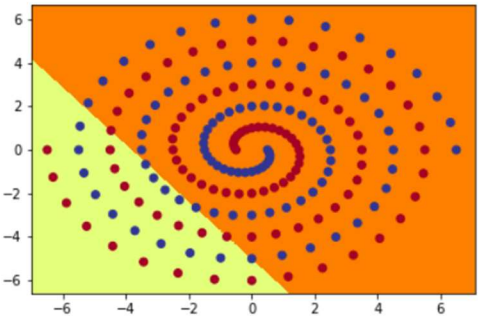
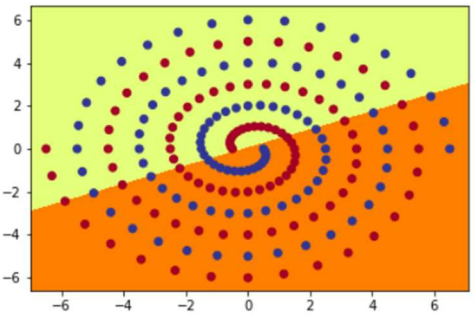
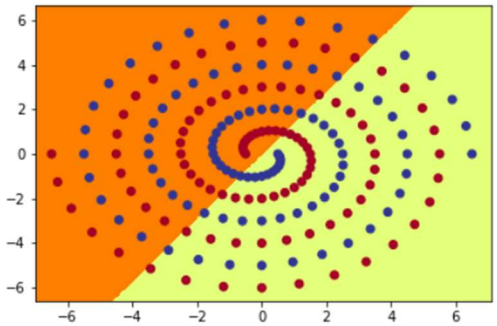
<u>PolarNet hidden nodes plot</u> –



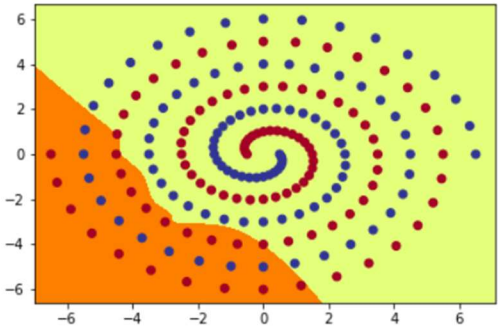
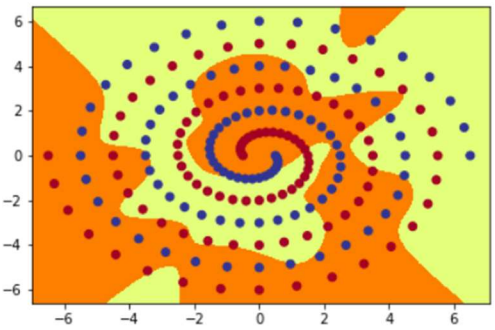
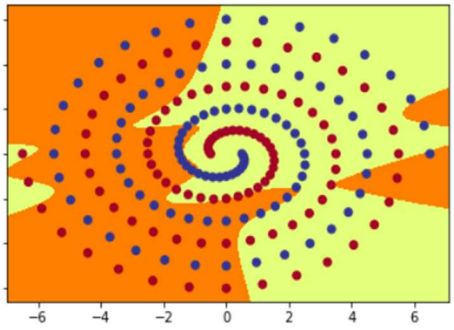
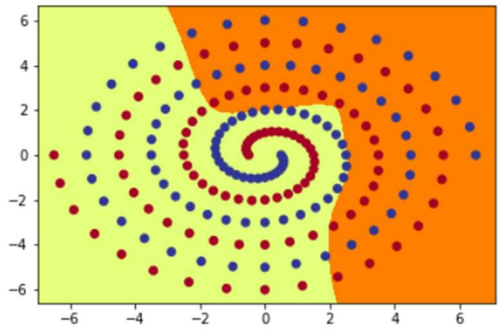
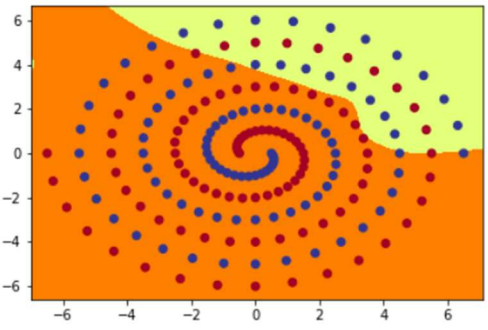
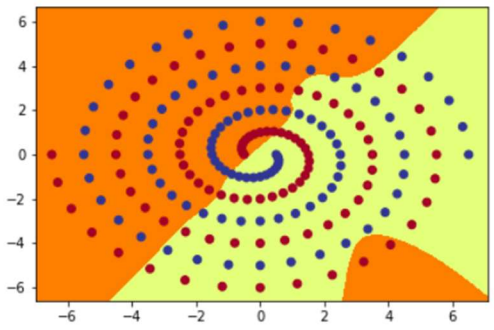
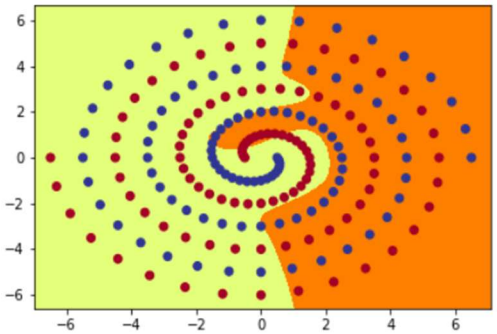
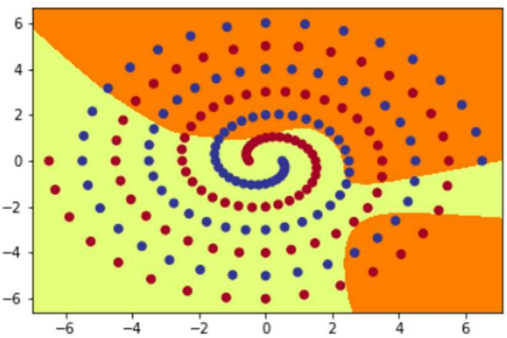
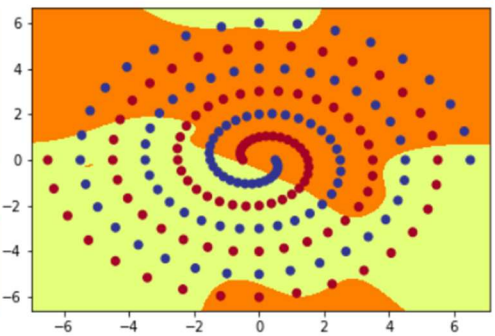


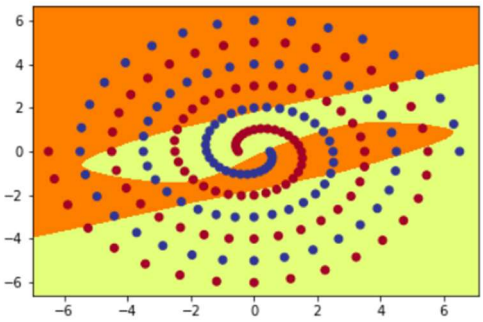
RawNet hidden nodes plot (hidden layer 1)–



RawNet hidden nodes plot (hidden layer 2) -



6.

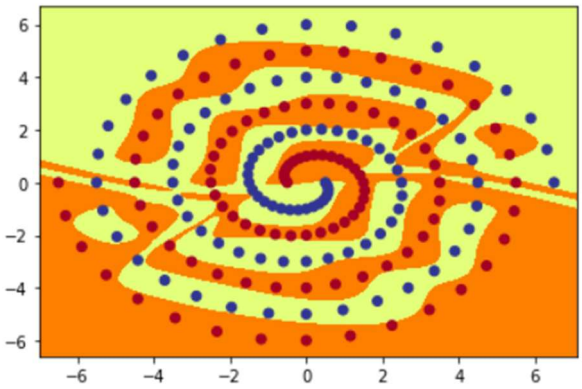
a. “PolarNet” has a single hidden layer with activation function tanh whereas “RawNet” has two hidden layers, both with tanh activation function.

The input is converted to polar coordinates before submitted to hidden layers in PolarNet. Hence the function is non-linear in nature where as in RawNet the function is linear in nature. Irrespective of the nature of the function, the summation is submitted to the activation function to train the classification.

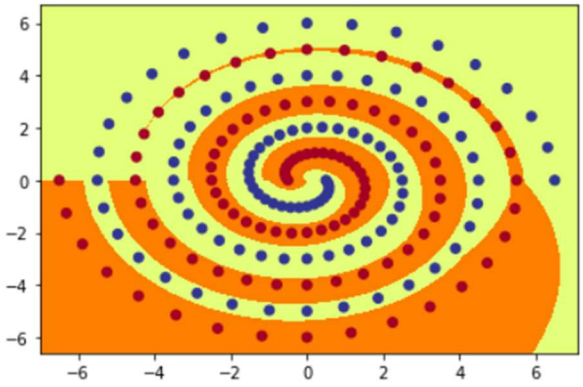
b. I have tried changing the initial weight value of RawNet to 0.2, 0.5, 0.003 and 0.4. A clear observation that can be made about the speed of RawNet is that the convergence takes much longer when the weight is set to 0.5 (converges after around 60000 epochs, sometimes even fails to converge) and takes much less time with smaller weights (converges after around 6000 epochs). The neural network doesn’t seem to converge (gets stuck at 98%) even for very low weights (0.003). Since it does not converge for higher values and very low values, no comment can be made about the success of training. One observation is that the success rate is good when weight = 0.1.

c.

I. **Changing batch size from 97 to 194**:
When I changed the batch size from 97 to 194 (other parameters remain same as set) for PolarNet, the neural network converges much faster than before as more amount of training data is being considered in one go. The network converges in 1100 (earlier 6000 epochs with batch size 97) epochs and the classification doesn’t seem to get affected and still looks perfect.
When run with RawNet, the classification varies and gives the following output:



II. **Changing tanh to relu and batch size = 194**:
The relu activation function for PolarNet seems to break the classification in between (output shown below) and is not perfect as with tanh. So relu is not as perfect as tanh for PolarNet.



When relu was used for RawNet, the convergence takes much longer than with tanh, sometimes even fails to converge (takes >20000 epochs and the accuracy is between 75%-76%).

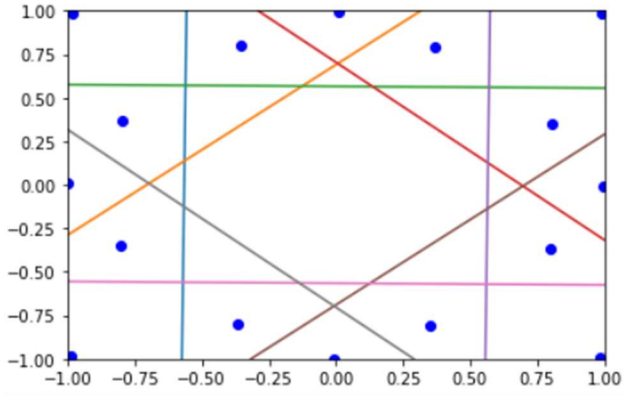
III. **Adding a third hidden layer to RawNet with batch size = 194**:

Adding a third hidden layer with tanh activations doesn't seem to add any benefit to the training as the learning is still slow and the accuracy fluctuates between 55% to 59% and doesn't seem to increase from there even after 20000 epochs. Without the third hidden layer the accuracy fluctuates between 51% to 54%. A snapshot of the forward() function with three hidden layers is shown below:

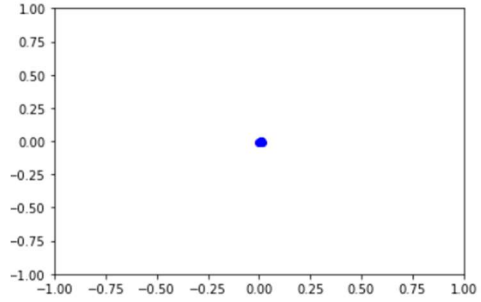
```
def forward(self, input):
    temp = self.linear1(input)
    self.hid1 = torch.tanh(temp) #hidden layer activation
    temp = self.linear2(self.hid1)
    self.hid2 = torch.tanh(temp)
    temp = self.linear3(self.hid2)
    self.hid3 = torch.tanh(temp)

    temp = self.linear4(self.hid3)
    output = torch.sigmoid(temp) #output layer activation
    #output = 0*input[:,0] # CHANGE CODE HERE
    return output
```

Part 3: Hidden Unit Dynamics

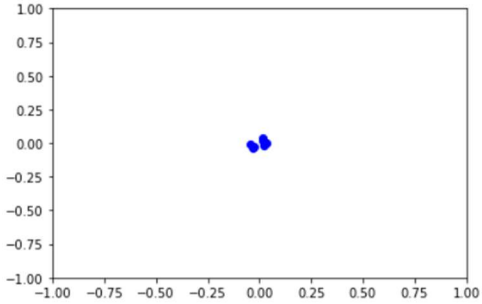
1. Output of encoder_main.py with target set to star16 is as shown below:



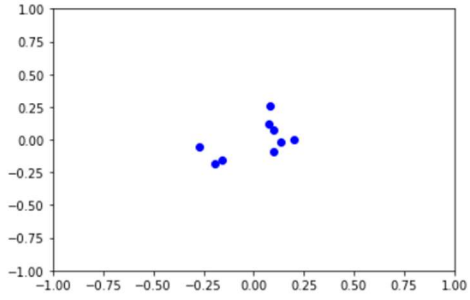
2. With target = 'input', input dimensions = 9, following are the first 11 plots generated :



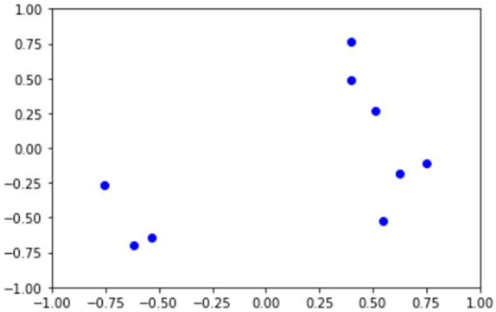
Epoch = 50



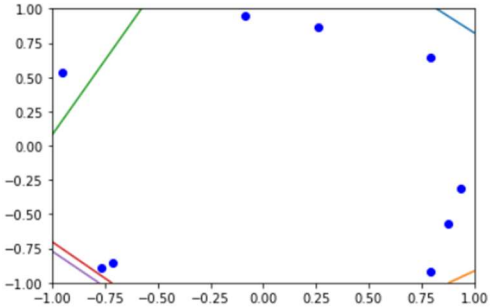
Epoch = 100



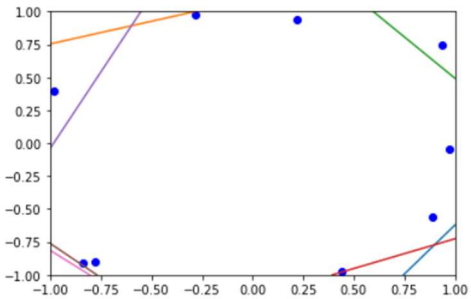
Epoch = 150



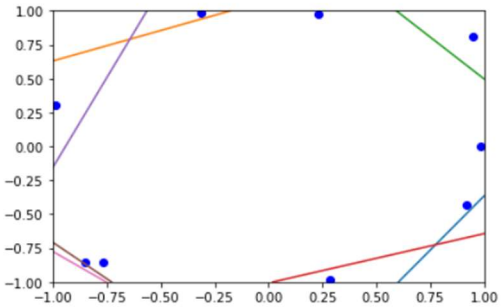
Epoch = 200



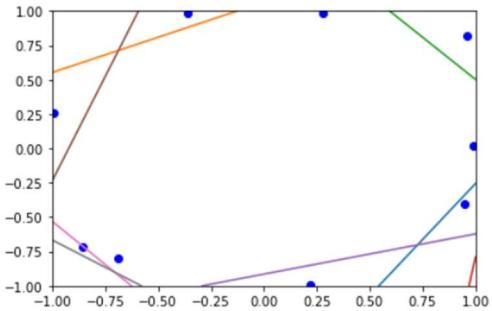
Epoch = 300



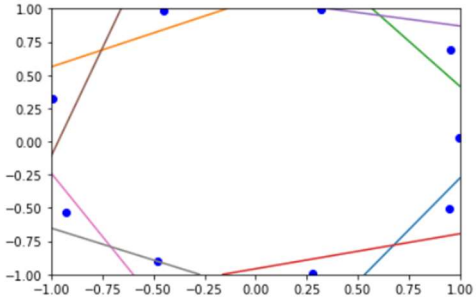
Epoch = 500



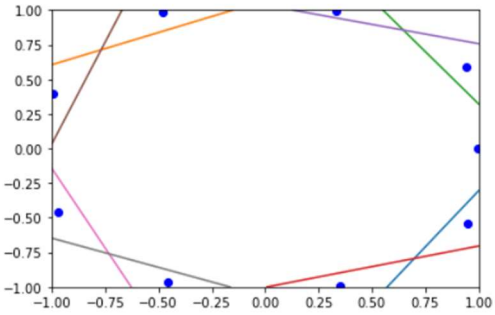
Epoch = 700



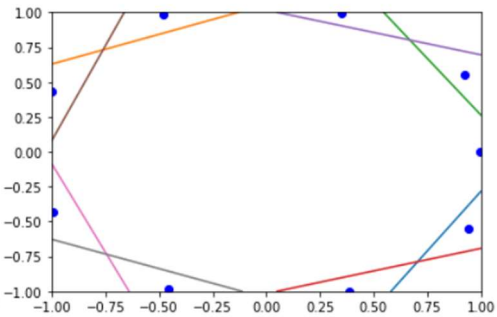
Epoch = 1000



Epoch = 1500

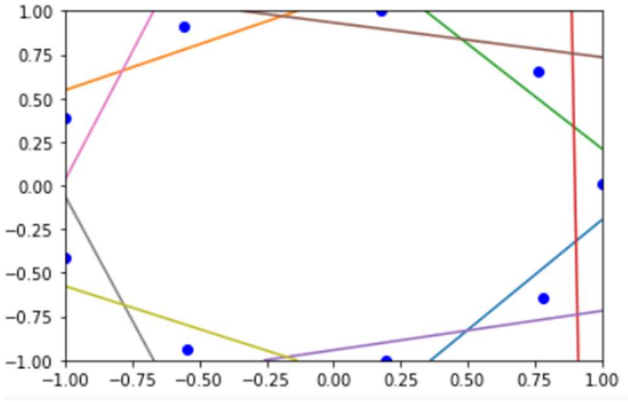


Epoch = 2000



Epoch = 3000

The final image is as shown below:

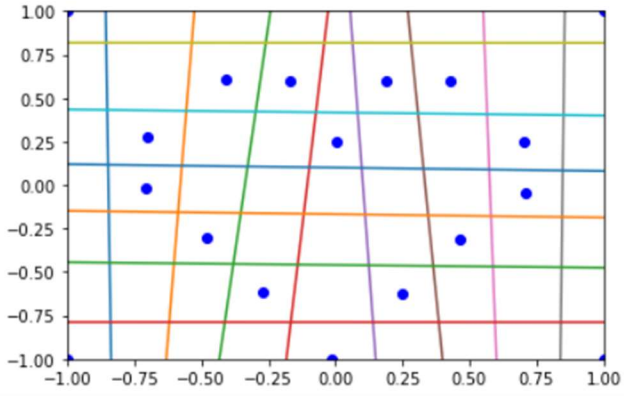


Epoch = 15120

As we observe the 11 plots, we can see that all the dots originate at the center and move towards the boundaries as the number of epochs increase from 50 to 3000 and above. The real bifurcation of dots and lines started after epoch = 1500. We can also observe that the dots seem to stabilize first rather than the lines.

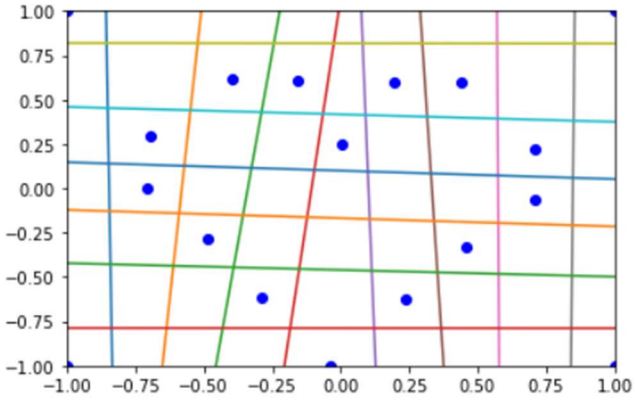
3.The dataset for tensor to create heart18 has been written in encoder.py and the output image is as follows:

```
ep9930: loss =  0.0200
ep9940: loss =  0.0200
```



Heart Shape

```
ep9970: loss =  0.0200
ep9980: loss =  0.0200
```



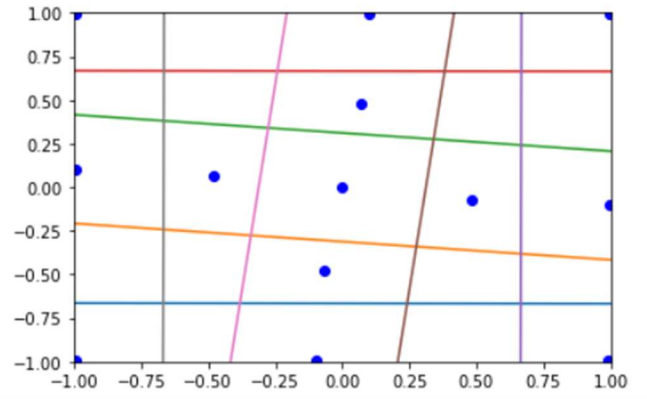
Heart Shape(slightly rotated)

4.

Target 1 -

I have thought of generating a '+' sign using a 13-by-8 tensor with 13 dots and 8 lines (or) 13 inputs and 8 outputs.

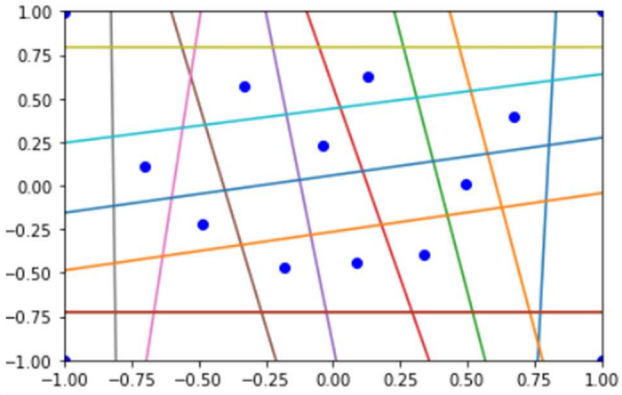
```
ep1770: loss =  0.0202
ep1780: loss =  0.0201
```



I have tried changing the learning rate from 0.4 to 0.9 and the output image shape seems to have changed drastically. The output seems to change each time I run the code with the same learning rate (0.4) and epochs (100000). I tried changing the learning rate to 0.8 and the program is faster and generates output within 900 epochs.

Target 2 -

The second design for the hidden unit geometry is a smiley ☺. It has been designed using a 14-by-14 tensor with 14 dots and 14 lines (or) 14 inputs and 14 outputs. Following is the output image generated. This has been generated using a learning rate = 0.8.



I have tried changing the learning rate to 0.5 and 0.7 and the output image seems to change its angle (smiley facing towards the right, left or upwards) but the shape remains undistorted. Similar observation is made when the program is run multiple times with the same learning rate of 0.8.