Part 1: Japanese Character Recognition

1. Final accuracy and confusion matrix

```
 # 0   #1   #2   #3   #4 #5   #6   #7   #8   #9
[[764.  5.  9.  14.  31.  63.  2.  63.  31.  18.]
 [ 7. 671. 109.  18.  28.  22.  55.  15.  26.  49.]
 [ 9.  61. 692.  26.  24.  21.  46.  38.  46.  37.]
 [ 5.  37.  57. 759.  15.  55.  14.  19.  28.  11.]
 [ 61.  52.  79.  22. 623.  17.  33.  36.  21.  56.]
 [ 8.  29. 122.  17.  19. 727.  28.   7.  33.  10.]
 [ 5.  22. 149.  10.  26.  24. 721.  21.   9.  13.]
 [ 17.  31.  27.  11.  81.  17.  55. 621.  90.  50.]
 [ 12.  37.  96.  43.   6.  30.  46.   7. 703.  20.]
 [ 9.  54.  83.   3.  55.  33.  18.  31.  41. 673.]]
```

2. Fully connected 2-layer network `NetFull`, using tanh at the hidden nodes and log SoftMax at the output node. Final accuracy and confusion matrix.

```
# 0   #1   #2   #3   #4 #5   #6   #7   #8   #9
[[852.  5.  1.  6.  27.  33.  3.  39.  29.  5.]
 [  4. 792.  39.  5.  23.  13.  70.  8.  21.  25.]
 [  5.  15. 845.  30.  15.  14.  24.  13.  26.  13.]
 [  4.  10.  36. 912.  4.  11.  6.  3.  7.  7.]
 [ 49.  26.  14.  8. 806.  12.  30.  16.  17.  22.]
 [ 10.   8.  66.  16.  15. 830.  28.   2.  18.   7.]
 [  3.  11.  66.  13.  18.   4. 869.  10.   1.   5.]
 [ 18.  14.  20.   5.  27.   8.  41. 810.  21.  36.]
 [ 11.  34.  26.  50.   3.  10.  29.   4. 822.  11.]
 [  3.  21.  49.   3.  31.  10.  27.  12.  16. 828.]]
```

Test set: Average loss: 0.5345, Accuracy: 8366/10000 (84%)

3. A convolutional network called `NetConv`, with two convolutional layers plus one fully connected layer, all using relu activation function, followed by the output layer, using log SoftMax.

   Final accuracy, 93%. Momentum was used at .9

```
# 0   #1   #2   #3   #4    #5   #6  #7    #8   #9
[[941.  4.  2.  1.  27.  7.  1.  12.  2.  3.]
 [  1. 927. 16.  1.  4.  5.  29.  2.  4.  11.]
 [ 12.  5. 840.  53.  12.  9.  24.  29.  9.  7.]
 [  2.  2.  15. 952.  5.  6.  5.  5.  5.  3.]
 [ 19.  3.  7.  9. 921.  5.  8.  10.  13.  5.]
 [  4.  13.  32.  9.  3. 921.  10.  3.  4.  1.]
 [  5.  15.  13.  5.  11.  5. 933.  12.  0.  1.]
 [ 22.  13.  10.  2.  7.  2.  11. 912.  3.  18.]
 [ 11.  10.  5.  5.  3.  6.  5.  4. 950.  1.]
 [  5.  11.  13.  6.  10.  0.  8.  4.  5. 938.]]

Test set: Average loss: 0.3377, Accuracy: 9235/10000 (92%)
```

4. Discuss
   a. the relative accuracy of the three models

      Simpler architectures such as the Linear function tend to have lower accuracy levels than more complex architectures like convolutional neural networks and fully connected networks.

      This is due the non-linearity component captured by these types of networks.

      The fully connected has lower accuracy than the convolutional network as the fully connected neural network may fail to capture some of the spatial features the convolutional neural network can capture. It also experiences a well-known problem called exploding vanishing gradients. The problem affects the backpropagation algorithm. In this case, as the number of hidden neurons increases past 80 the gradient explodes.

   b. the confusion matrix for each model: which characters are most likely to be mistaken for which other characters, and why?

      In the three Models, character 2="su" is likely to be mistaken. Character 2="ma" is classified incorrectly consistently across all models.

      For lin, the character "su" is also mistaken with 1="ki" and 6="ma".

      For full, the character "su" is mistaken with 6="ma". Also, the character 6="ma" is mistaken with character "su".

Finally for conv, the character “su” is mistaken with 5="ha", but also the character 3=“tsu” is mistaken with “su”.

The reason for the misclassification is one of the challenges of the KMNIST dataset and the challenges representing the mapping to the one-character situation.

Hentaigana or variant kana, are Hiragana characters that have more than one form of writing, as they were derived from different Kanji. Therefore, one Hiragana class of Kuzushiji-MNIST may have many characters.

c. Other experiments

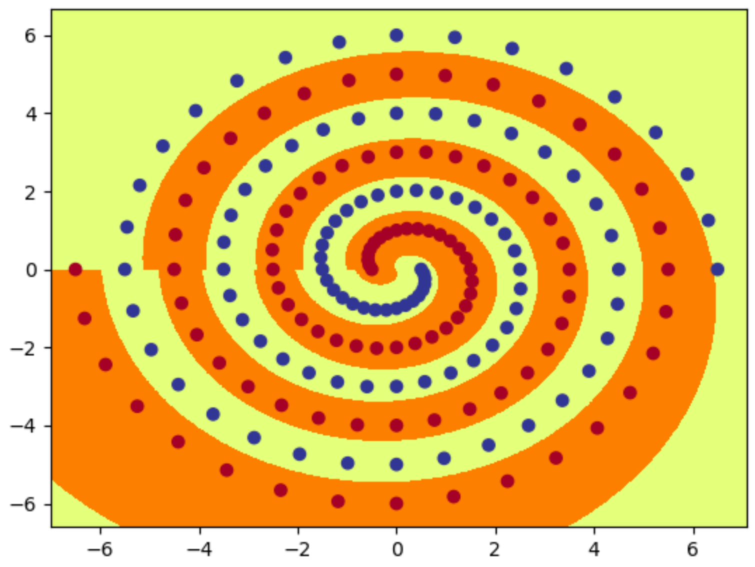
Part 2: Twin Spirals Task

1. Code

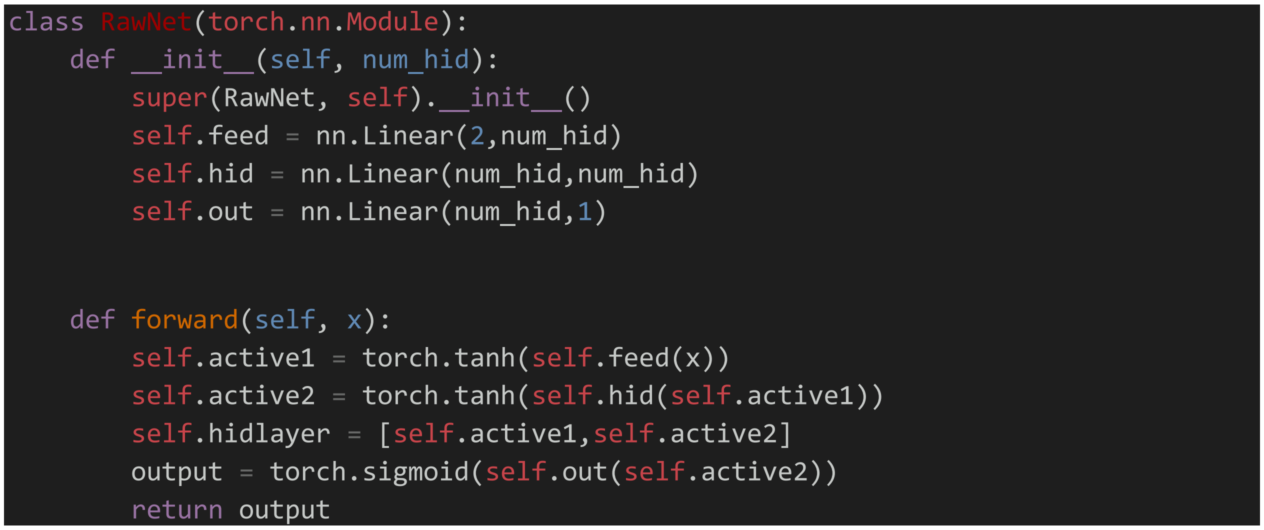
```python
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import math
import torch.nn.functional as F

class PolarNet(torch.nn.Module):
    def __init__(self, num_hid):
        super(PolarNet, self).__init__()
        self.feed = nn.Linear(2,num_hid)
        self.hid = nn.Linear(num_hid,1)

    def forward(self, input):
        x = input[:,0]
        y = input[:,1]
        out = torch.zeros(input.shape,dtype=torch.float32)
        out[:,0] = torch.sqrt(x*x + y*y) #tranform data
        out[:,1] = torch.atan2(y,x)
        #here we start the forward pass
        self.active1 = torch.tanh(self.feed(out))
        self.active2 = torch.tanh(self.hid(self.active1))
        self.hidlayer = [self.active1]
        return F.sigmoid(self.active2)
```

2. Minimum number of hidden nodes required so that this PolarNet learns to correctly classify all the training data within 20000 epochs, on almost all runs.

Minimum number of hidden nodes varies consistently between 15 to 18 hidden nodes.
Graph:



3. Two fully connected hidden layers with tanh activation, plus the output layer, with sigmoid activation. Code:



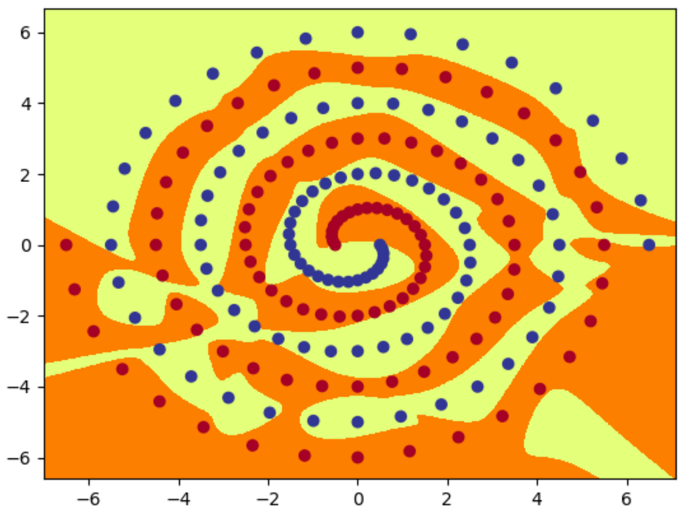
```
class RawNet(torch.nn.Module):
    def __init__(self, num_hid):
        super(RawNet, self).__init__()
        self.feed = nn.Linear(2,num_hid)
        self.hid = nn.Linear(num_hid,num_hid)
        self.out = nn.Linear(num_hid,1)


    def forward(self, x):
        self.active1 = torch.tanh(self.feed(x))
        self.active2 = torch.tanh(self.hid(self.active1))
        self.hidlayer = [self.active1,self.active2]
        output = torch.sigmoid(self.out(self.active2))
        return output
```

4. Value for the number of hidden nodes (--hid) and the size of the initial weights (--init) such that this RawNet learns to correctly classify all the training data within 20000 epochs, on almost all runs.

   Value of hidden nodes and other metaparameters:

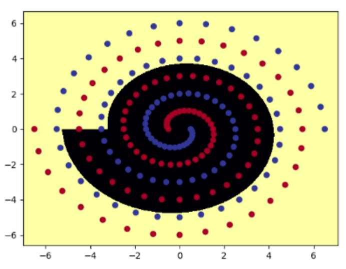
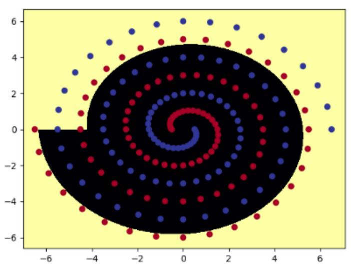
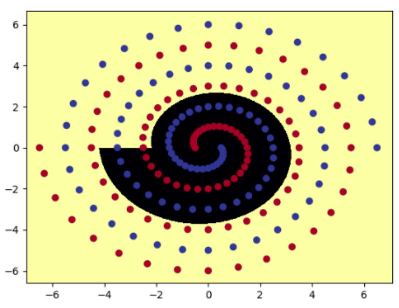
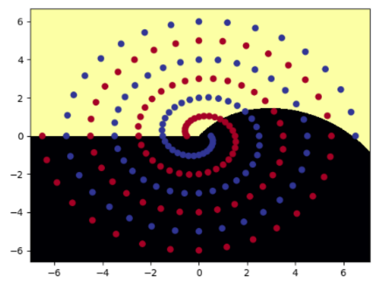
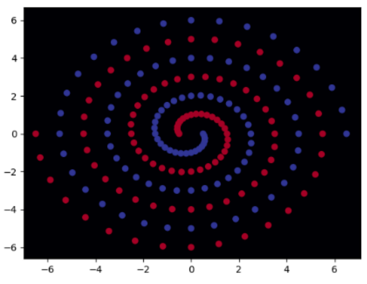
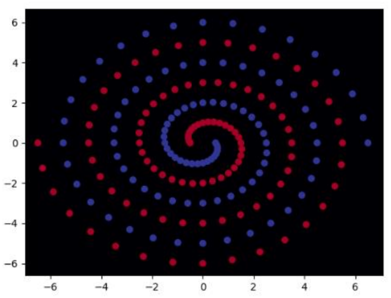
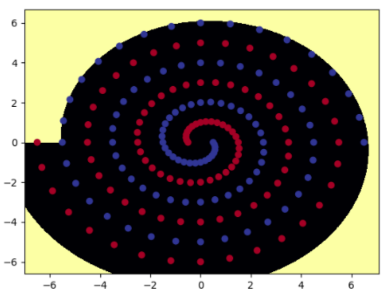
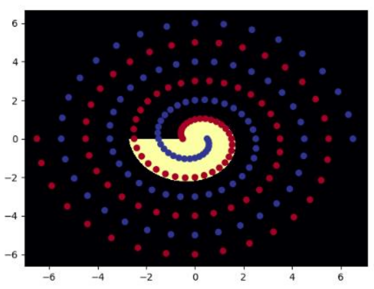
   Hidden: from 10+ hidden nodes 100% accuracy is achieved in less than 20,000 epochs. To limit the number of activation images number of hidden nodes = 10 was chosen with init=.9 to reduce the number of epochs.

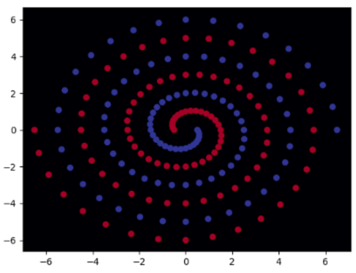
   Graph `raw_out.png`:



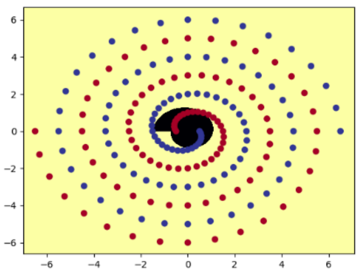
5. Plots of all the hidden nodes in PolarNet, and all the hidden nodes in both layers of RawNet:

PolarNet:

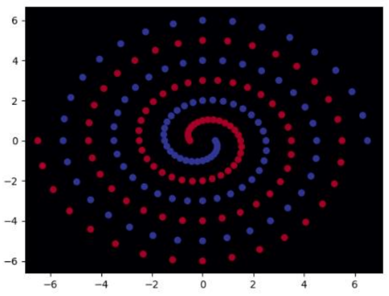











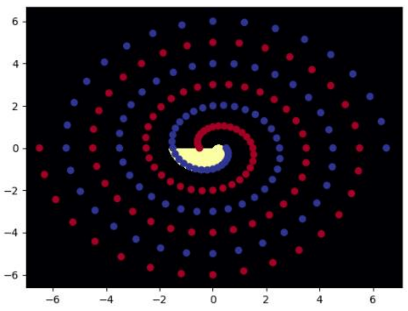
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6



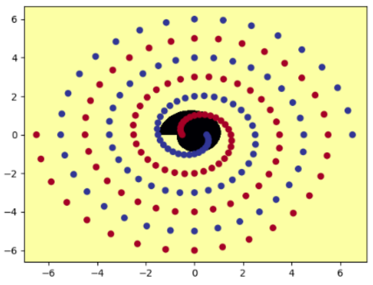
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6



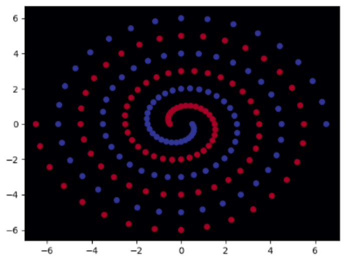
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6



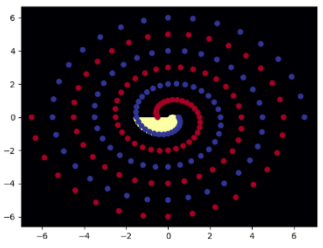
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6



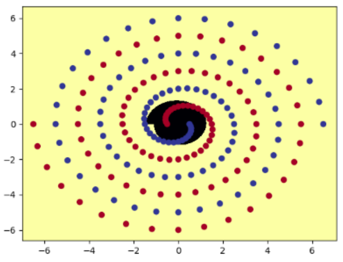
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6



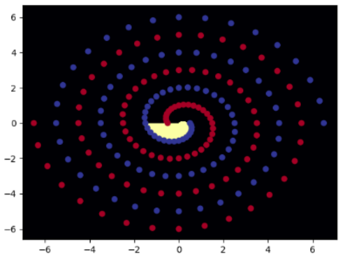
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6



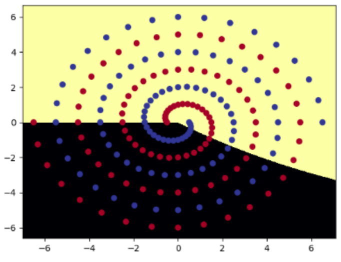
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6



6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6

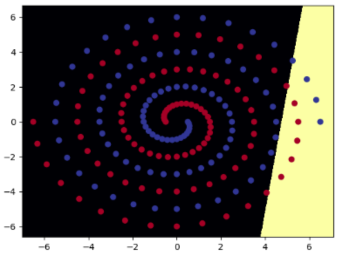


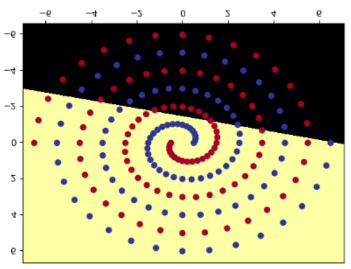
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6

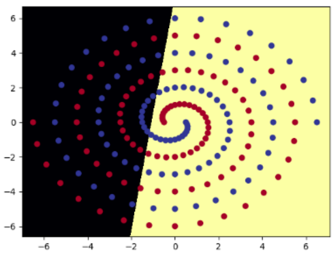


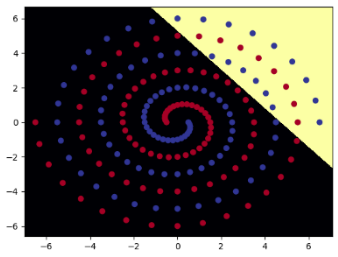
6 4 2 0 −2 −4 −6
−6 −4 −2 0 2 4 6

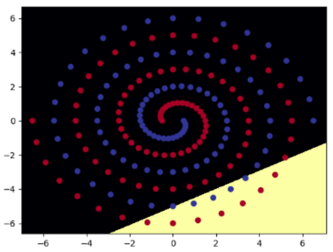
RawNet:

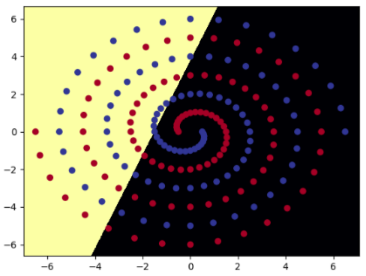


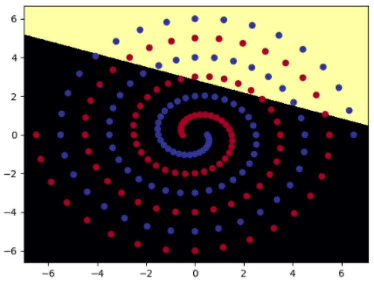


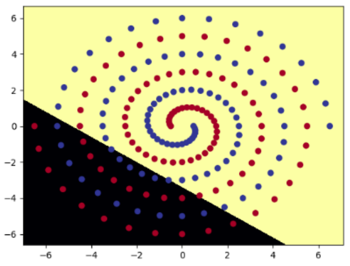


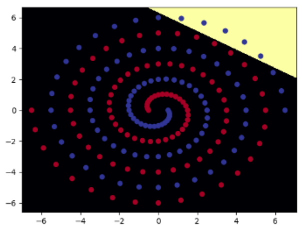


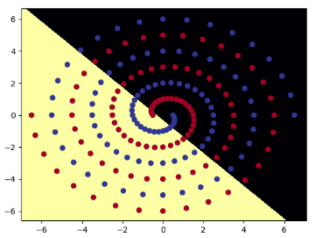


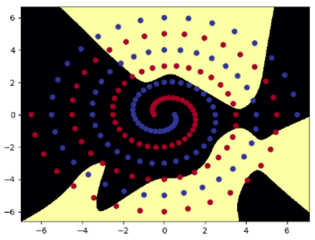


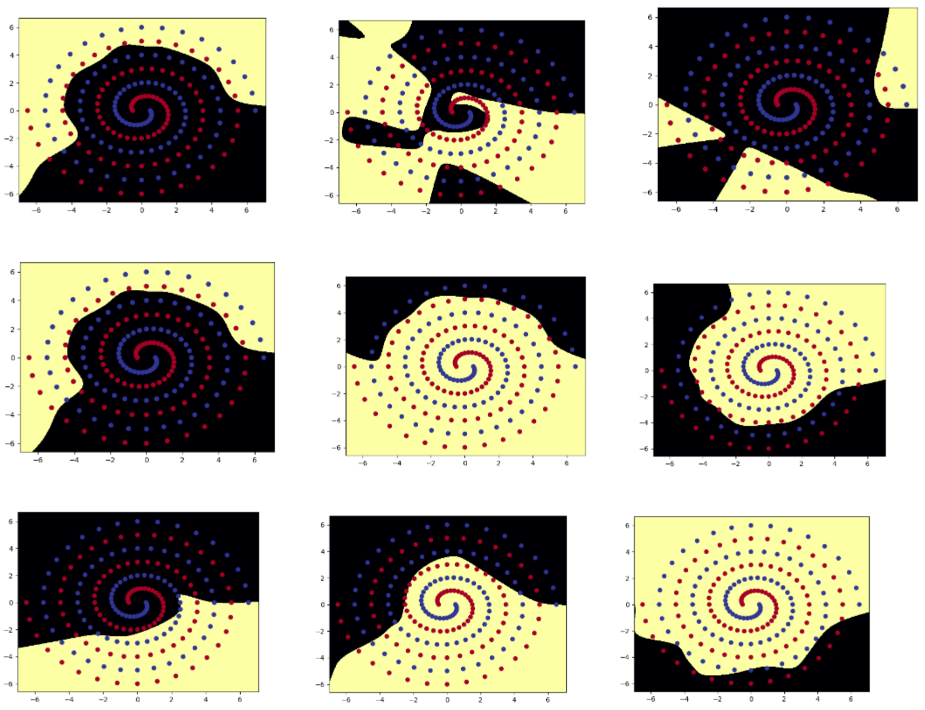












6. Discuss

a. the qualitative difference between the functions computed by the hidden layer nodes PolarNet and RawNet, and a brief description of how the network uses these functions to achieve the classification

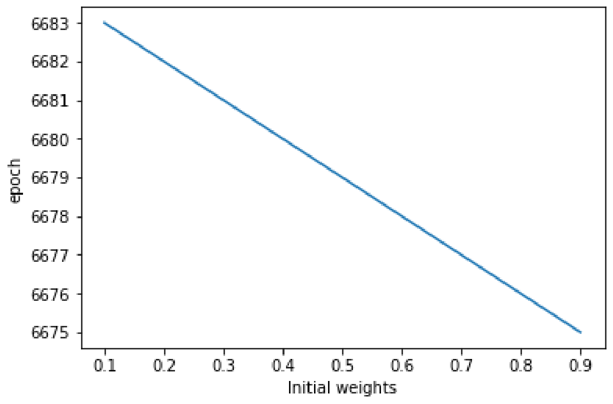
The difference between PolarNet and RawNet is that RawNet has an extra forward pass and backpropagates the network twice while PolarNet only backpropagates once.

In this case the transformation done to PolarNet helps enhance the classification of the x, y coordinates.

How it works is the activation function considers how different parameters interact with one another and does a transformation in this case tanh which gets to decide the neurons who go forward into the next layer.

The tanh activation function has the nice property that gets values of different signs the range is between 1 and -1 which is what is needed as the x, y values have negative values. While the sigmoid function is between one and zero which is what we need to classify the values as higher than .5 or lower than the threshold.

b. the effect of different values for initial weight size on the speed and success of learning for RawNet



As the initialization of the weights increases the number of epochs required to achieve 100% accuracy decreases. This is not surprising as the weights can determine how quickly the network converges or not at all. The distribution of the weights affects the gradients of the network and therefore how effective the training is.

c. Experiment with other changes and comment on the result

changing batch size from 97 to 194,

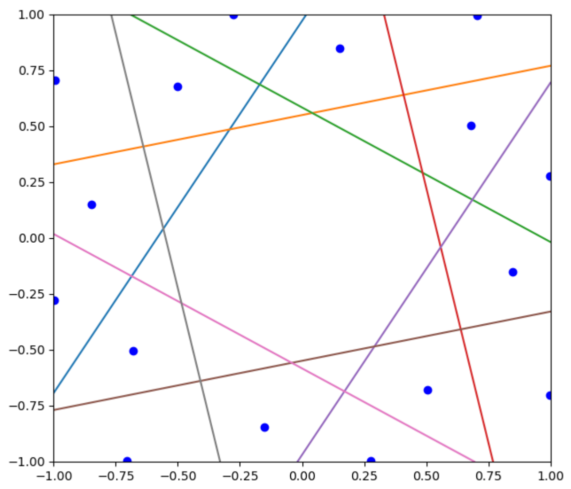
using SGD instead of Adam,

changing tanh to relu,

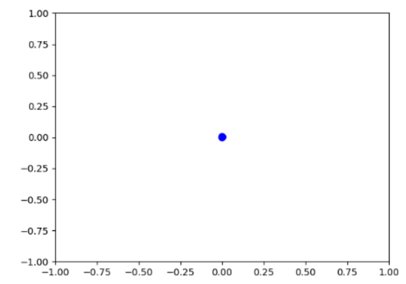
adding a third hidden layer

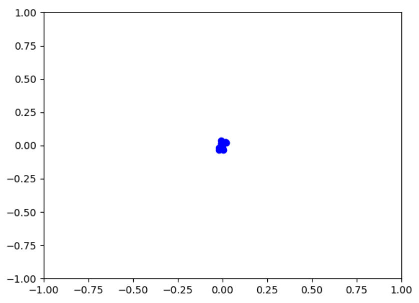
Part 3: Hidden Unit Dynamics

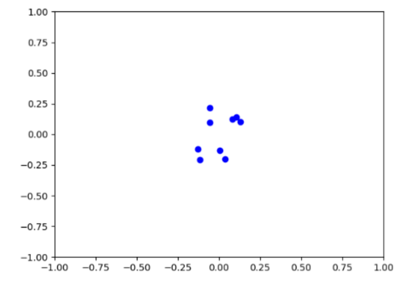
1. Image:

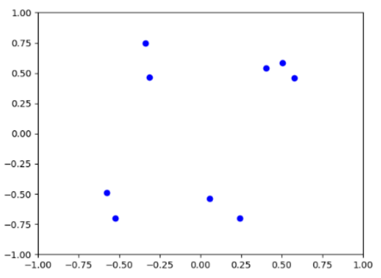


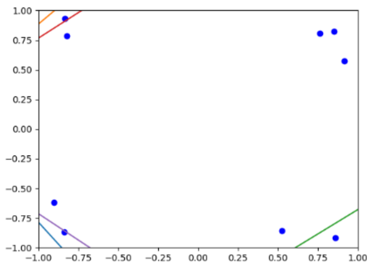
2. 9-2-9 encoder with both input and target determined by a one-hot encoding. First eleven images generated (epochs 50 to 3000), plus the final image.

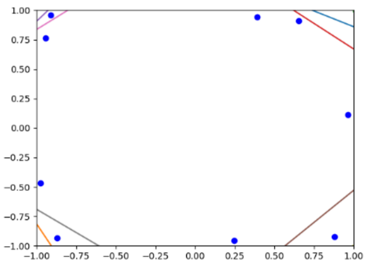


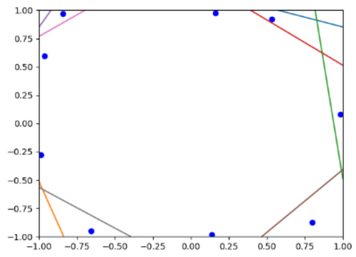


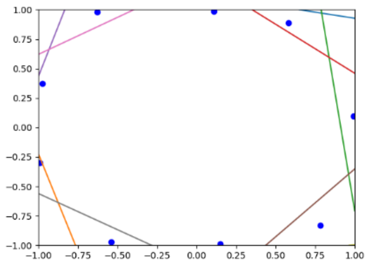


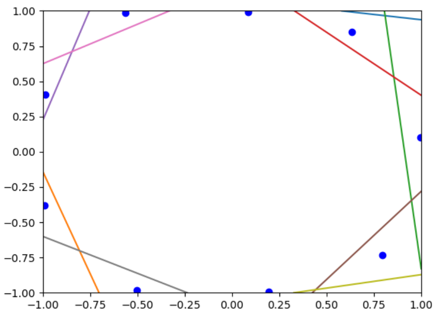


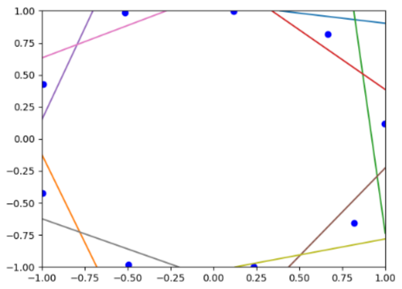

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

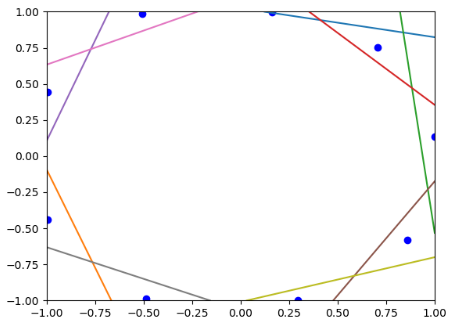

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


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

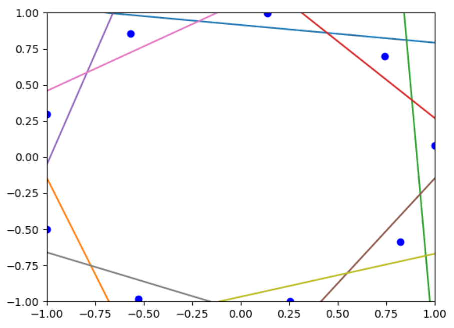

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


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

Final Image:



how the hidden unit activations (dots) and output boundaries (lines) move as the training progresses.

At every iteration, the input matrix is just an identity matrix with all diagonals equal to one and all other entries equal to zero. The target value is always the same desired output we are trying to match.

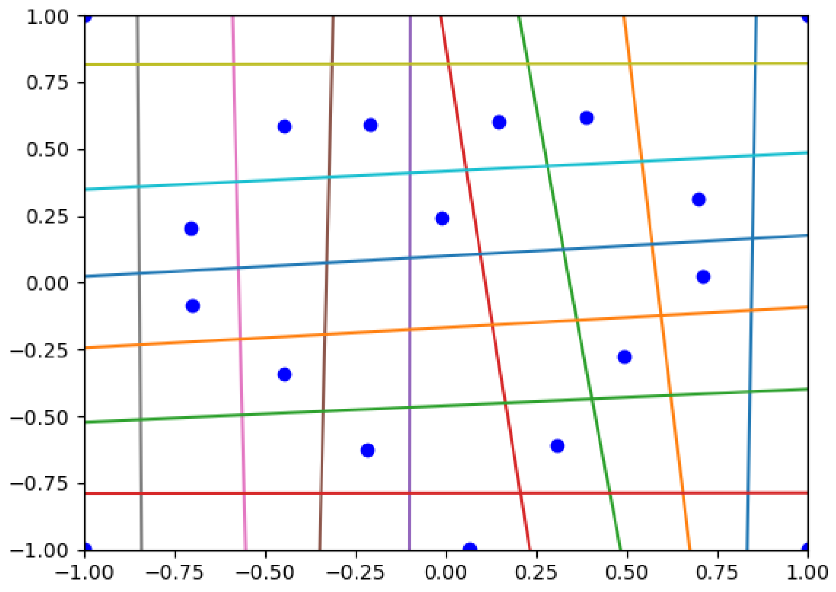
The network applies the forward step, which involves computing a single neuron at the time. The first row will always have one value at each row. Therefore, only the weight of that neuron will affect the network. It is then applied to the transformation function tanh it then activates that neuron and computes the sigmoid function. Then computes the loss and applies the backpropagation algorithm to the network.

For the first 11 images the epochs are
`50,100,150,200,300,500,700,1000,1500,2000,3000`
In any of those epochs the algorithm takes the input to hidden weights and biases and plots those weights in the range -1 to 1. Then, the algorithm takes the output hidden weights and draws a line interval to show the decision boundary of each output.

At the end the network will solve (if there is a solution) the decision boundaries of each output of the target. The columns of the target can be seen as decision boundaries while the rows can be seen as the blue dots.

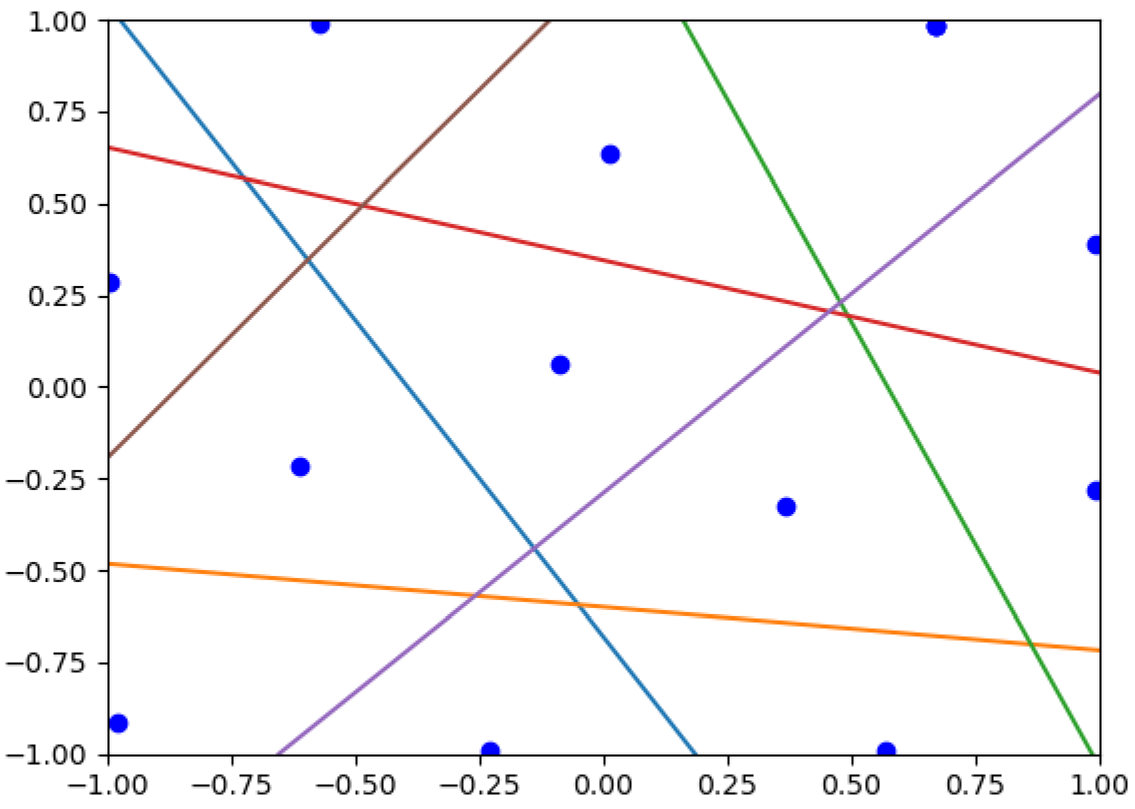
Sometimes there are multiple solutions, the image plotted is dependent on the initialization of the weights.

3. Final image in your report, and include the tensor heart18 in your file encoder.py

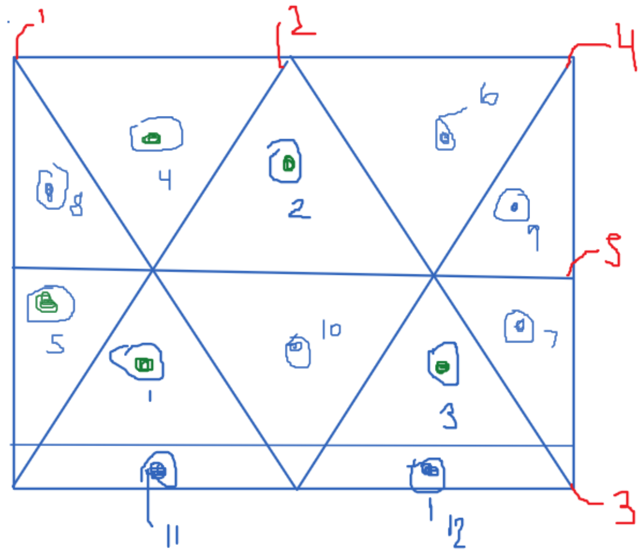


4. Create training data in tensors target1 and target2, which will generate two images of your own design.

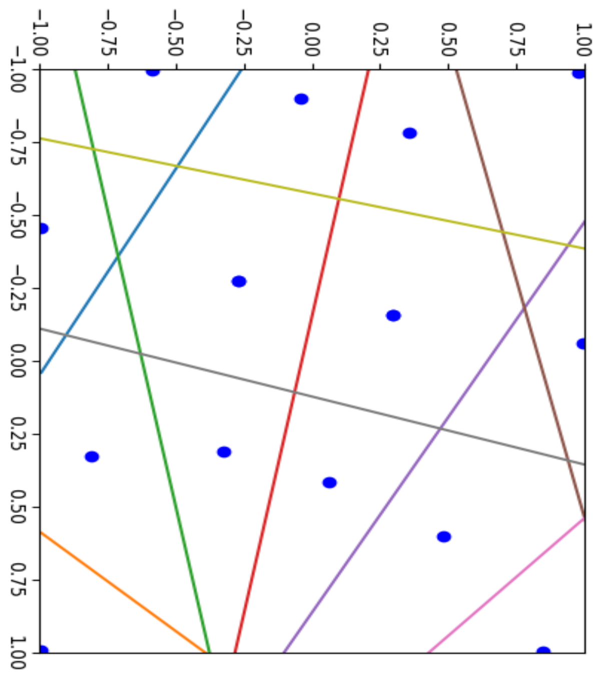
Target 1, Zelda's Triforce:

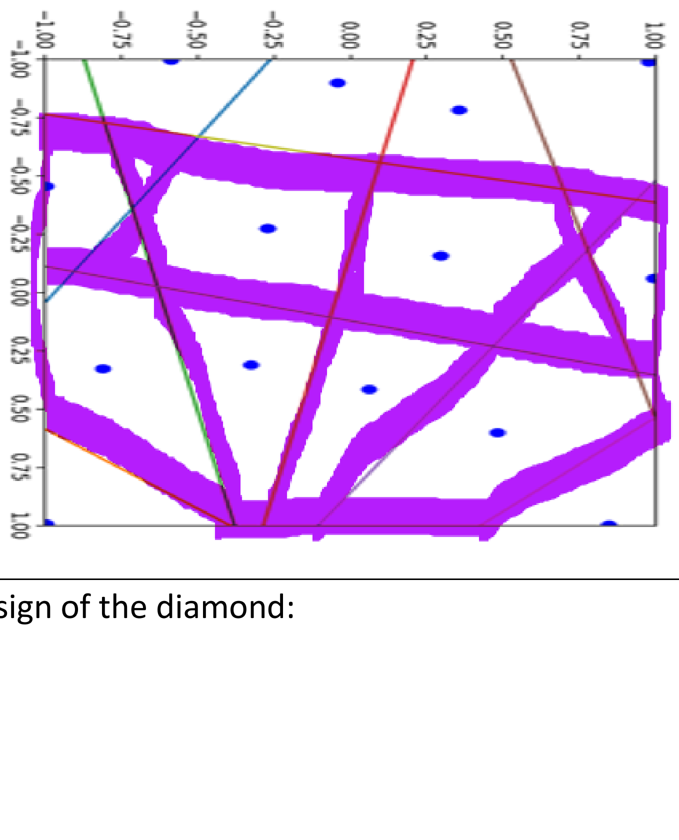


Design of Zelda’s Triforce:

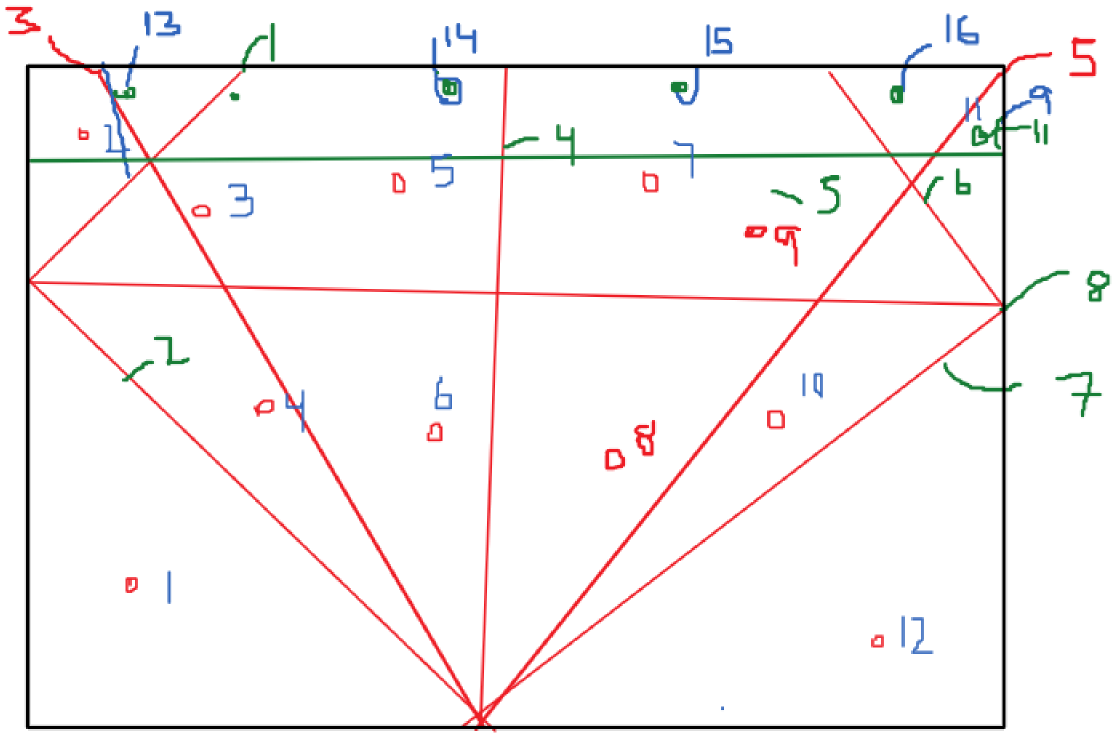


Target 2, Diamond:





Design of the diamond:


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