## 

## **COMP9444**

## Homework 1

### Usama Sadiq

### z5235652

## **Part 1 Question 1**

**Final Confusion matrix** =

[[765. 7. 7. 4. 59. 8. 5. 16. 11. 8.]

[ 5. 671. 65. 37. 51. 26. 23. 29. 37. 52.]

[ 8. 109. 689. 61. 84. 125. 150. 26. 94. 85.]

[ 12. 18. 26. 759. 21. 17. 10. 12. 42. 3.]

[ 31. 27. 27. 14. 623. 20. 27. 89. 6. 51.]

[ 65. 22. 21. 56. 20. 726. 25. 19. 32. 34.]

[ 2. 58. 45. 14. 33. 27. 719. 53. 45. 19.]

[ 61. 14. 37. 17. 34. 8. 21. 620. 6. 31.]

[ 32. 25. 44. 26. 19. 32. 9. 89. 704. 39.]

[ 19. 49. 39. 12. 56. 11. 11. 47. 23. 678.]]

**Final** 🡪 **Test set:** **Average loss:** 1.0107, **Accuracy:** 6954/10000 (70%)

# **Part 1 Question 2**

1. **Num\_id** = 500 **accuracy** = 85%
2. **Num\_id** = 1000 **accuracy** = 85%
3. **Num\_id** = 1500 **accuracy** = 84%
4. **Num\_id** = 2000 **accuracy** = 84%
5. **Num\_id** = 3000 **accuracy** = 84%

**Final Confusion matrix** =

[[850. 6. 7. 2. 37. 11. 3. 17. 11. 4.]

[ 5. 814. 13. 8. 31. 17. 14. 11. 28. 19.]

[ 2. 37. 831. 24. 16. 84. 40. 20. 29. 49.]

[ 6. 2. 46. 927. 7. 6. 10. 6. 55. 4.]

[ 26. 20. 10. 3. 823. 11. 16. 16. 4. 28.]

[ 36. 10. 19. 15. 6. 832. 6. 8. 8. 4.]

[ 2. 58. 26. 5. 28. 17. 894. 33. 28. 21.]

[ 42. 5. 13. 2. 14. 1. 7. 836. 2. 17.]

[ 28. 17. 20. 6. 23. 15. 2. 23. 830. 9.]

[ 3. 31. 15. 8. 15. 6. 8. 30. 5. 845.]]

**Final** -> **Test set:** **Average loss:** 0.4935, **Accuracy:** 8482/10000 (85%)

# **Part 1 Question 3**

**Training No Accuracy (%)**

1 93

2 93

3 93

4 93

5 93

**Final Confusion Matrix** =

[[939. 3. 11. 3. 19. 6. 5. 5. 3. 6.]

[ 7. 926. 11. 1. 14. 19. 10. 12. 22. 12.]

[ 3. 4. 894. 18. 4. 50. 21. 10. 15. 11.]

[ 1. 0. 36. 957. 3. 9. 3. 2. 11. 2.]

[ 28. 3. 2. 2. 912. 3. 7. 8. 5. 5.]

[ 3. 2. 10. 6. 4. 887. 3. 1. 5. 0.]

[ 2. 46. 19. 3. 17. 14. 946. 16. 10. 5.]

[ 11. 3. 3. 5. 10. 6. 2. 926. 4. 3.]

[ 2. 2. 7. 1. 9. 2. 1. 2. 922. 2.]

[ 4. 11. 7. 4. 8. 4. 2. 18. 3. 954.]]

**Test set:** **Average loss:** 0.2747, **Accuracy:** 9263/10000 (93%)

# Part 1 Question 4

My major takeaway from the above activity is to always start with the simplest model for the problem at hand, so that we can set a baseline (accuracy) which is also the suggested approach according to occam's razor. The simplest model can consist of a single linear layer along with a softmax activation function and mostly commonly used values for metaparameters. We can then start to add a layer and vary the number of hidden nodes at the hidden layer to see how well the model can fit the data before adding any more layers.

**Part a**

When talking about accuracy we can see from the above results that the CNN network performs best to perform the image classification task as the convolutional layers extract more features from the images which can then be learned in fully connected layers part of the network. While increasing the number of hidden nodes in the hidden layer of fully connected 2-layer network did not have any effect on the accuracy. CNN network hits 93% accuracy consistently.

**Part b**

If we do a comparison between the three confusion matrixes, we notice that the diagonal values are increasing as we move from NetLin towards NetConv indicating the increase in the performance of the model. As if we look at NetLin confusion matrix, we see that the confusion matrix has high values when it makes a prediction of 2 for target values of 1,3,4,5,6,8,9 characters and it is possible that the model will most likely make a wrong prediction for these characters.

If we look at the confusion matrix for NetFull model, the diagonal values have increased and the most likely character this model might classify wrong is 5 where the model is predicting 2.

**Part c**

**NetLin**

1. **Learning rate** =0.1 **num\_hid** = 10

**Test set:** **Average loss:** 1.3255, **Accuracy:** 6722/10000 (67%)

1. **Learning rate** = 0.001 **num\_hid** = 10

**Test set:** **Average loss:** 1.0467, **Accuracy:** 6725/10000 (67%)

1. **Num\_id** = 500

**Test set:** **Average loss:** 1.0096, **Accuracy:** 6972/10000 (70%)

1. **Num\_id** = 1000

Test set: Average loss: 1.0101, **Accuracy:** 6960/10000 (70%)

1. **Num\_id** = 1500

**Test set:** Average loss: 1.0096, **Accuracy:** 6971/10000 (70%)

First I changed the learning rate parameter while keeping the number of hidden nodes same to try to see if our model is not stuck in a local minima but as you can see from the above result that it decreased the accuracy rather than increasing it which means 0.01 learning rate is performing best for this model. Second parameter I changed for this model is number of hidden nodes while keeping learning rate to 0.01 and it turns out there is no change from my original result and therefore it could mean that our model is not complex enough to learn and generalize better.

**NetFull**

1. **Learning rate** =0.1 **num\_hid** = 500

**Test set:** **Average loss:** 0.4343, **Accuracy:** 8936/10000 (89%)

1. **Learning rate** = 0.001 **num\_id =** 500

**Test set: Average loss:** 0.4978, **Accuracy:** 8489/10000 (85%)

First I tried to change the number of hidden nodes as you can refer Part 1 Question 2 had no change in the accuracy of the model which could mean that the model might be getting stuck in a local minima and therefore I changed the learning rate to 0.1 which proved to improve the accuracy while changing the learning rate to 0.0001 proved to be slower when compared with other learning rates hence had less accuracy than before.

**NetConv**

1. **Conv2d in-channels** = 1 **Conv2d out-channels** = 4

**Conv2d in-channels** = 4 **Conv2d out-channels** = 6

**Test set: Average loss:** 0.4224, **Accuracy:** 8769/10000 (88%)

1. **Conv2d in-channels** = 1 **Conv2d out-channels** = 6

**Conv2d in-channels** = 6 **Conv2d out-channels** = 12

**Test set: Average loss:** 0.3663, **Accuracy:** 9023/10000 (90%)

While keeping kernel size of convolutional layers to 5 and max pooling of kernel size to 2, we can see as we increase the number of out-channels of the convolutional layer the accuracy of the model is increasing which is the result of more features for the fully connected layers to learn.

# **Part 2 Question 2**

**Num\_hid** = 6

**Training no Epoch No Percentage (%)**

1 14400 100

2 6500 100

3 17500 100

4 3400 100

5 20,000 98.48

6 6300 100

7 20,000 98.48

8 20,000 91.75

9 5900 100

10 20,000 99.48

**Num\_hid** = 7

**Training no Epoch No Percentage (%)**

1 20,000 91.75

2 12100 100

3 4100 100

4 5300 100

5 20,000 77.84

6 10400 100

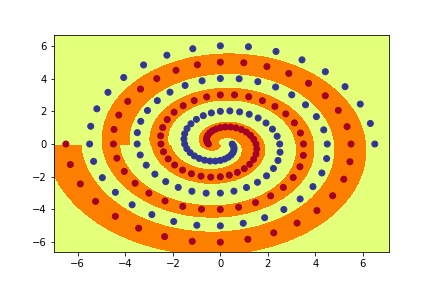
7 9200 100

8 5000 100

9 11800 100

10 20,000 91.75

The minimum number of hidden nodes in hidden layer of PolarNet which enables the model to learn to correctly classify all the training data within 20000 epochs, on almost all runs are 7. We can see that 7 number of hidden nodes model correctly classifies all the training data within 20000 epochs 7/10 times.



# **Part 2 Question 4**

**Learning Rate** = 0.01 **Number of hidden nodes** = 10 are constant for all runs in this section.

**Initial weight size** = 0.1

**Training No Epochs Percentage (%)**

1 20,000 92.27

2 20,000 50.52

3 20,000 53.61

4 20,000 53.61

5 20,000 53.61

6 4,600 100

7 9,300 100

8 17,000 100

9 6,800 100

10 20,000 50.52

**Initial weight size** = 0.2

**Training No Epochs Percentage (%)**

1 6700 100

2 20,000 97.94

3 20,000 98.94

4 9,800 100

5 20,000 97.94

6 6,800 100

7 11,700 100

8 9,100 100

9 5,700 100

10 10,100 100

**Initial weight size** = 0.3

**Training No Epochs Percentage (%)**

1 20,000 92.27

2 20,000 97.94

3 20,000 97.94

4 20,000 97.94

5 20,000 97.42

6 20,000 96.91

7 20,000 99.84

8 20,000 99.48

9 20,000 96.39

10 10,000 100

**Initial weight size** = 0.01

**Training No Epochs Percentage (%)**

1 20,000 50.52

2 20,000 50.52

3 20,000 50.52

4 20,000 50.52

5 20,000 50.52

6 20,000 50.52

7 20,000 50.52

8 20,000 50.52

9 20,000 50.52

10 20,000 50.52

**Initial weight size** = 0.001

**Training No Epochs Percentage (%)**

1 20,000 50.52

2 20,000 50.52

3 20,000 50.52

4 20,000 50.52

5 20,000 50.52

**Initial weight size** = 0.25

**Training No Epochs Percentage (%)**

1 20,000 96.39

2 20,000 98.45

3 9,700 100

4 20,000 94.85

5 20,000 89.69

6 8,300 100

7 20,000 99.48

8 20.000 94.85

**Initial weight size** = 0.21

**Training No Epochs Percentage (%)**

1 7,900 100

2 4,900 100

3 8,000 100

4 20,000 98.97

5 4,100 100

6 20,000 95.88

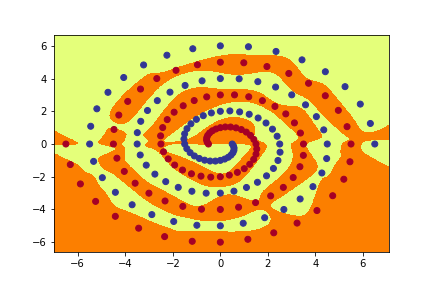
7 8,700 100

8 9,000 100

9 9,300 100

10 20,000 97.42

The minimum size of initial weights which enables the model to learn to correctly classify all the training data within 20000 epochs, on almost all runs are 0.21. We can see that 0.21 size of initial weight gives correctly classifies all the training examples within 20,000 epochs 7/10 times.



# Part 2 Question 6

**Num\_hid** = 5

**Initial weight** = 0.1

**Training No Epochs Percentage (%)**

1 20,000 94.85

2 20,000 99.85

3 20,000 88.66

4 20,000 87.63

5 20,000 76.80

6 20,000 87.63

7 20,000 90.72

8 20,000 83.51

9 20,000 83.51

10 20,000 83.51

**Num\_hid** = 5

**Initial weight** = 0.2

**Training No Epochs Percentage (%)**

1 20,000 91.24

2 20,000 92.78

3 20,000 75.77

4 20,000 83.51

5 20,000 78.35

6 20,000 94.85

7 20,000 93.30

8 20,000 77.32

9 20,000 94.85

10 20,000 86.60

**Num\_hid** = 5

**Initial weight** = 0.3

**Training No Epochs Percentage (%)**

1 20,000 82.47

2 20,000 73.20

3 20,000 82.47

4 20,000 83.51

5 20,000 84.02

6 6400 100

7 20,000 89.69

8 20,000 84.54

9 20,000 74.74

10 20,000 71.13

**Num\_hid** = 5

**Initial weight** = 0.01

**Training No Epochs Percentage (%)**

1 20,000 86.60

2 20,000 50.52

3 19,100 100

4 20,000 80.93

5 20,000 50.52

6 20,000 50.52

**Num\_hid** = 5

**Initial weight** = 0.001

**Training No Epochs Percentage (%)**

1 20,000 50.52

2 20,000 91.75

3 20,000 87.63

4 20,000 84.02

5 20,000 50.52

**Num\_hid** = 5

**Initial weight** = 0.21

**Training No Epochs Percentage (%)**

1 20,000 79.38

2 20,000 90.72

3 20,000 88.66

4 20,000 86.60

5 20,000 84.54

**Num\_hid** = 5

**Initial weight** = 0.25

**Training No Epochs Percentage (%)**

1 20,000 97.94

2 20,000 77.32

3 20,000 96.91

4 20,000 74.74

5 5,500 94.85

6 20,000 91.75

**Num\_hid** = 6

**Initial weight** = 0.1

**Training No Epochs Percentage (%)**

1 20,000 94.85

2 20,000 99.48

3 9,900 100

4 8,200 100

5 17,300 100

6 20,000 84.54

7 20,000 90.72

**Num\_hid** = 6

**Initial weight** = 0.2

**Training No Epochs Percentage (%)**

1 20,000 84.54

2 20,000 84.02

3 10,000 100

4 15,000 92.78

5 20,000 97.94

**Num\_hid** = 6

**Initial weight** = 0.21

**Training No Epochs Percentage (%)**

1 20,000 86.08

2 20,000 91.75

3 20,000 82.47

4 20,000 93.81

5 10,200 100

**Num\_hid** = 6

**Initial weight** = 0.25

**Training No Epochs Percentage (%)**

1 20,000 89.69

2 20,000 90.72

3 8,500 100

4 20,000 98.97

5 20,000 94.85

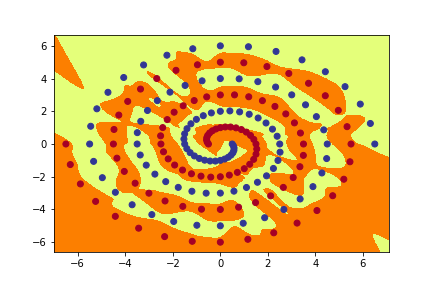
**Num\_hid** = 7

**Initial weight** = 0.20

**Training No Epochs Percentage (%)**

1. 11,900 100
2. 9,000 100
3. 12,700 100
4. 9,300 100
5. 20,000 92.78
6. 20,000 92.78
7. 8,300 100
8. 20,000 94.33
9. 3,900 100
10. 9,200 100

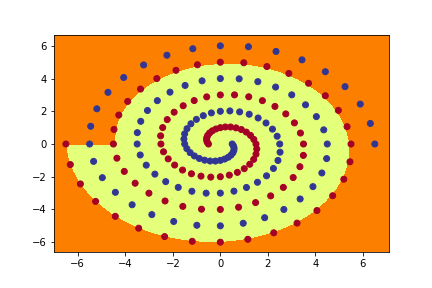
The minimum number of hidden nodes and value of initial weight size which enables the model to learn to correctly classify all the training data within 20000 epochs, on almost all runs are 7 and 0.20. We can see that 7 number of hidden nodes correctly classifies all the training data within 20,000 epochs 7/10 times.

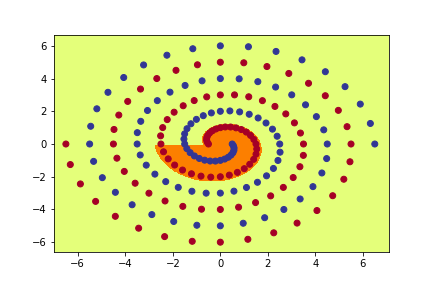


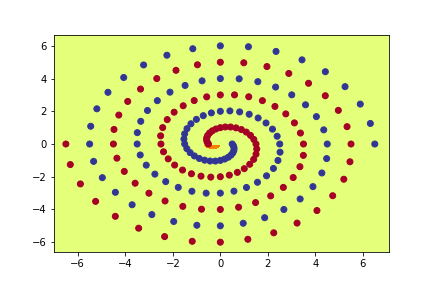
# Part 2 Question 7

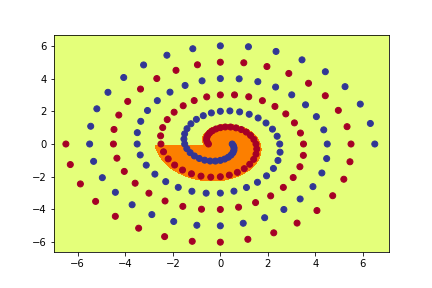
**PolarNet**

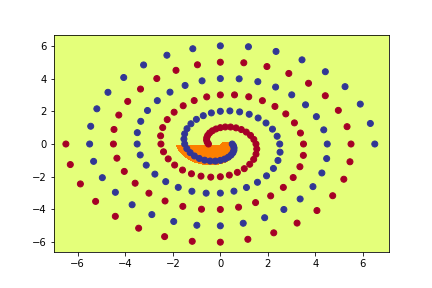
Layer 1 nodes in sequence

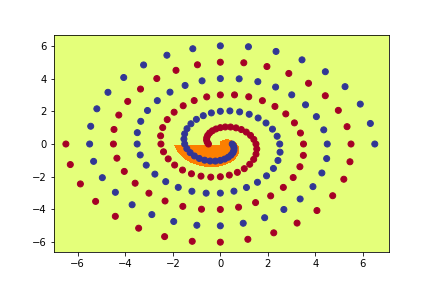


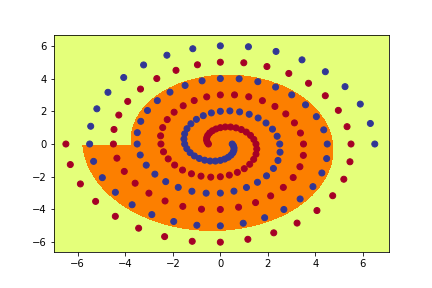


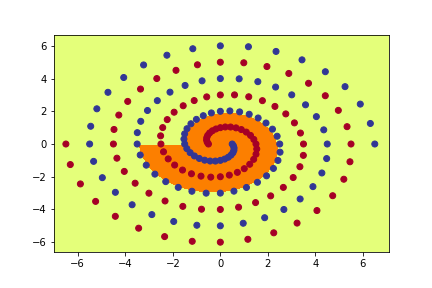


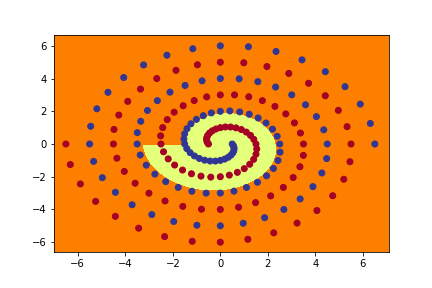


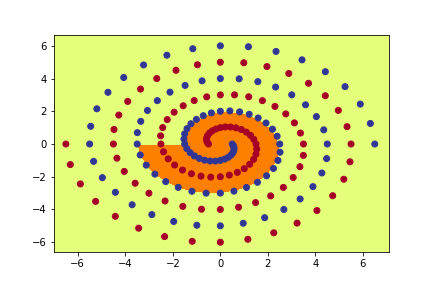






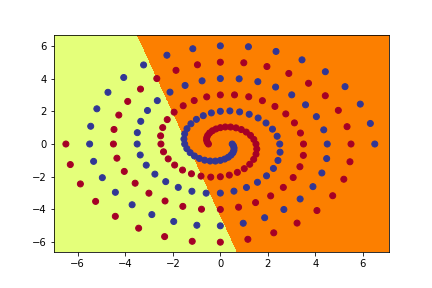
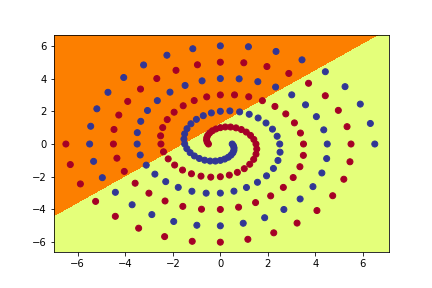
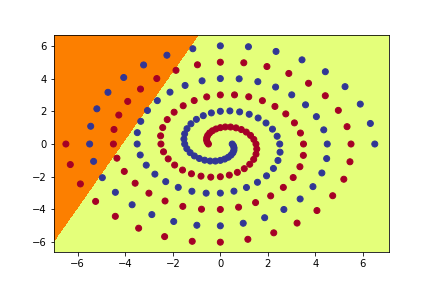
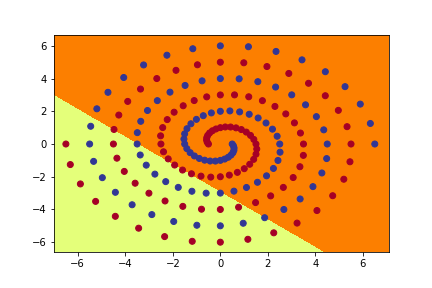
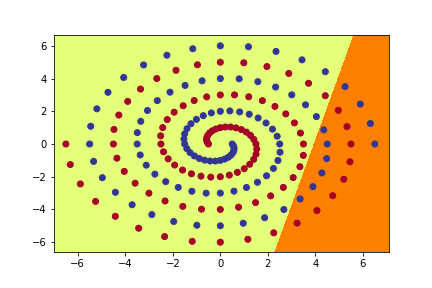
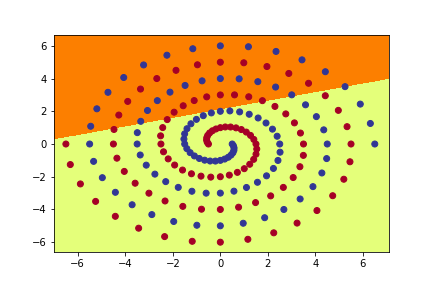
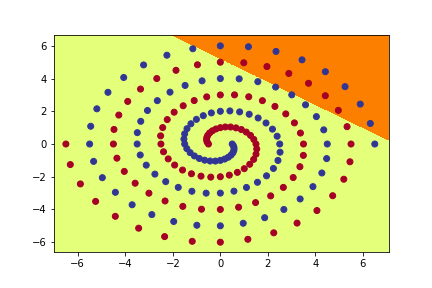
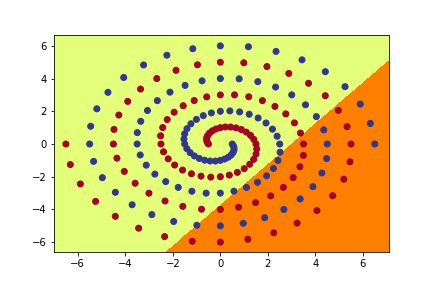
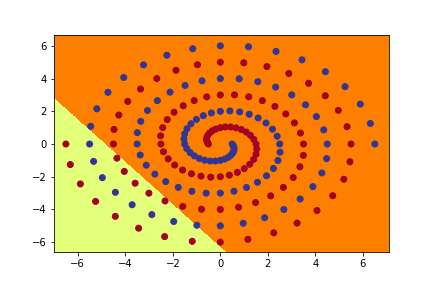
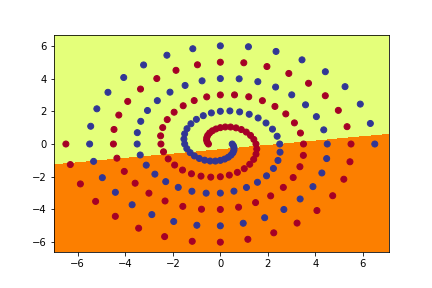




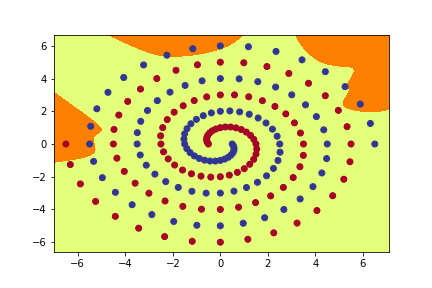
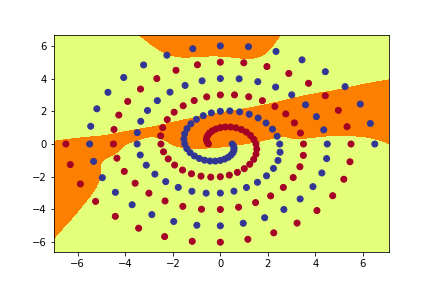
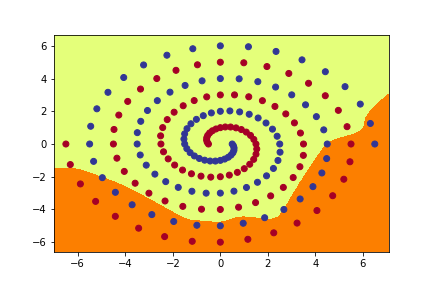
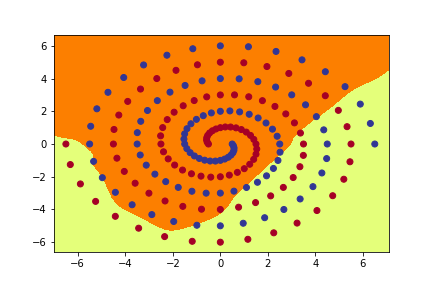
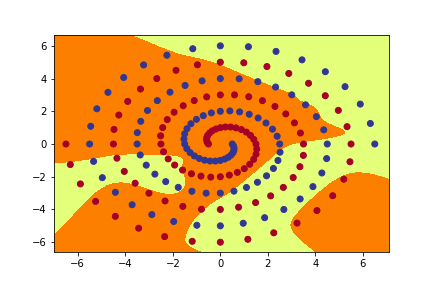
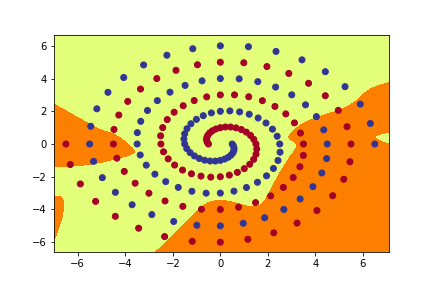
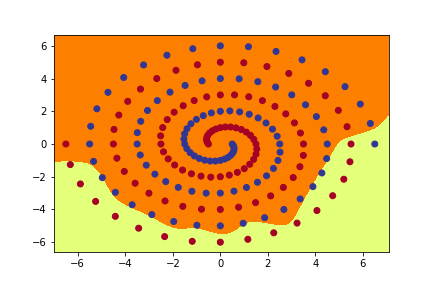
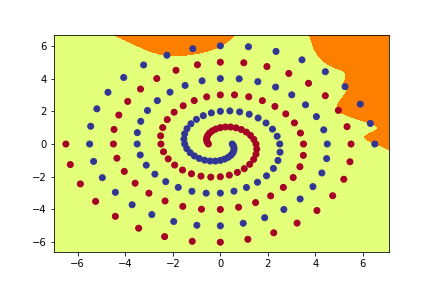
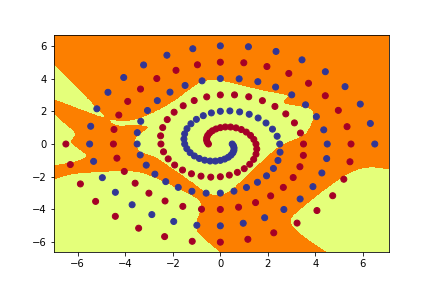
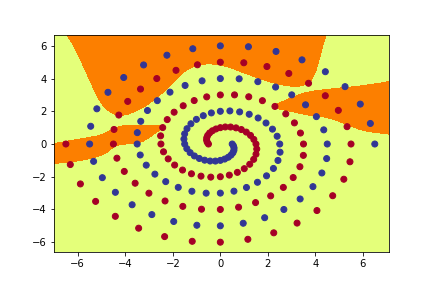


**RawNet:**

Layer 1 nodes in sequence

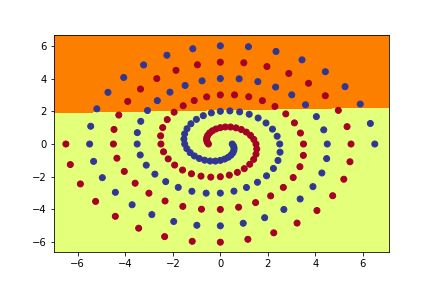
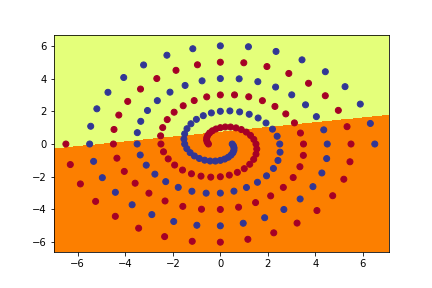
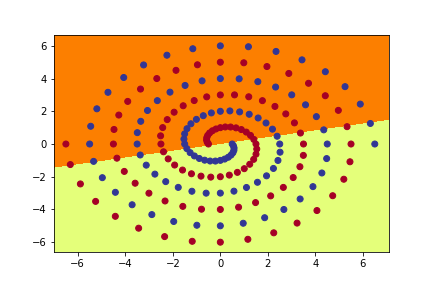
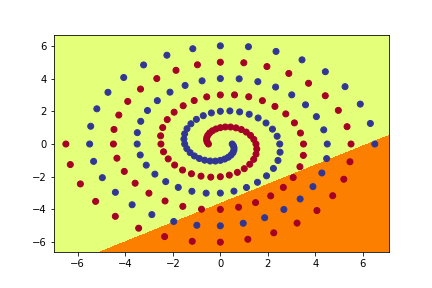
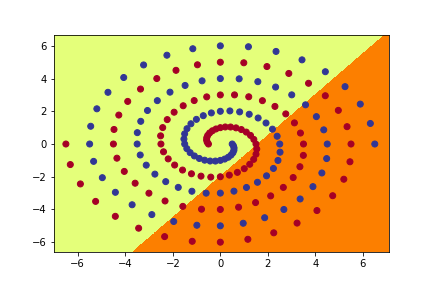
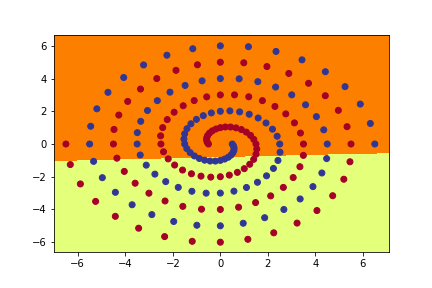
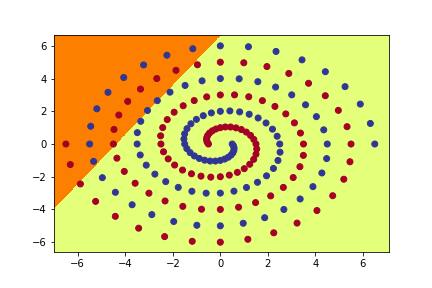
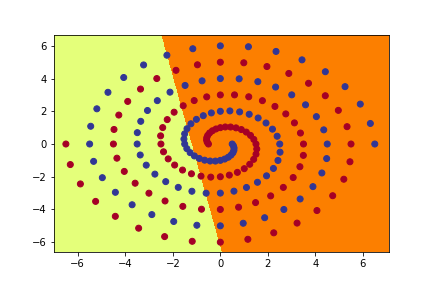
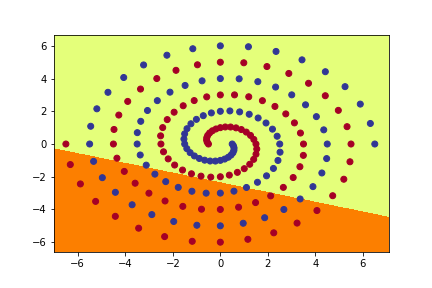
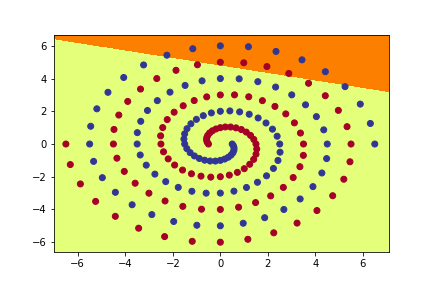
         

Layer 2 nodes in sequence

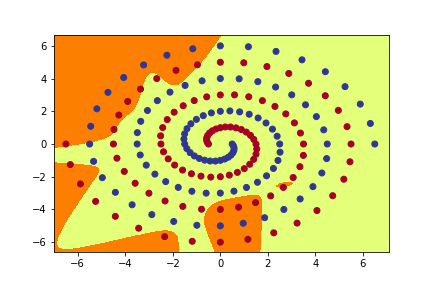
         

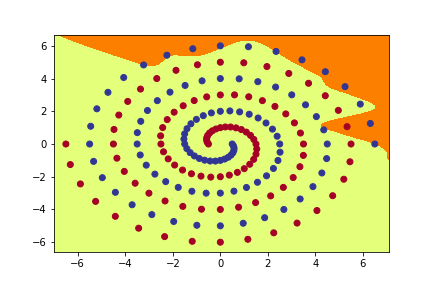
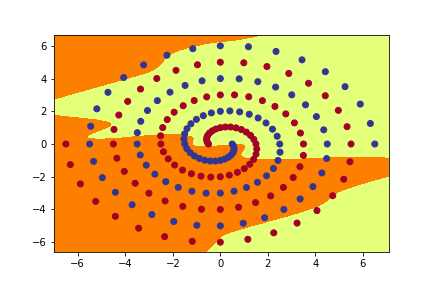
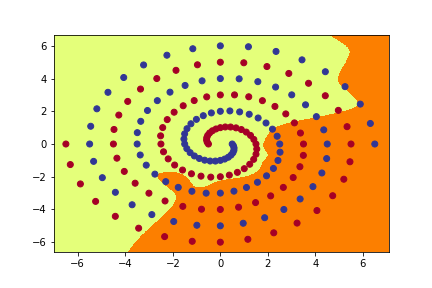
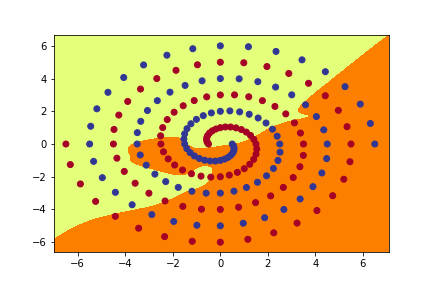
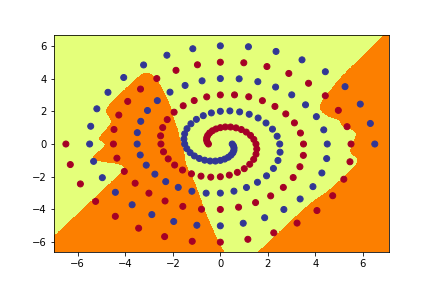
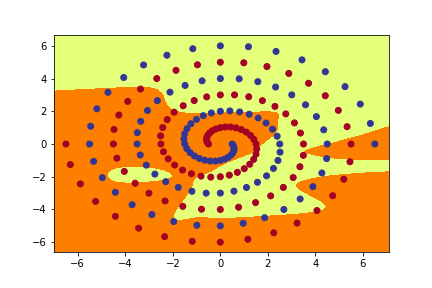
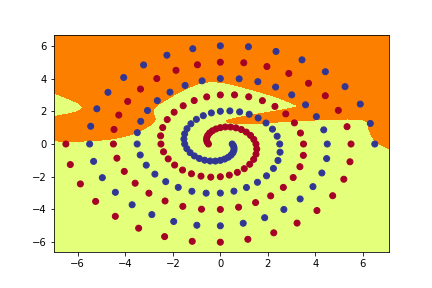
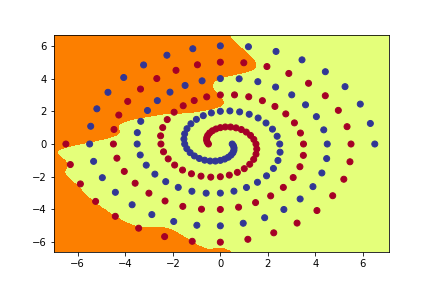
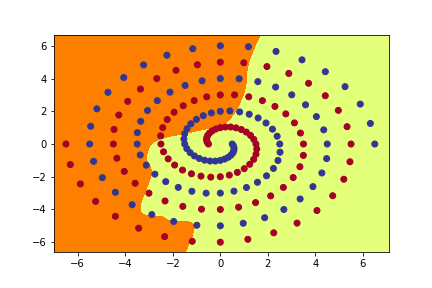
**ShortNet:**

Layer 1 nodes in sequence

Layer 2 nodes in sequence



# Part 2 Question 9 Part a

As we know that in PolarNet, first we are converting the coordinates to polar coordinates and then sending the input to the network to learn the two-spiral problem. The network is trying to learn different parts of the two spirals patterns by learning the polar coordinates and that’s why if we notice the images of the hidden layers we see the model learning parts of spiral pattern. These patterns are then combined in the output layer to produce the model which we can see that correctly classifies most training examples and close to the actual spiral pattern. While RawNet and ShortNet both in the first hidden layer are learning simple linear features of the spiral pattern and that is why we see straight lines. The second hidden layer of RawNet and ShortNet try to learn complex parts of the spiral pattern rather than straight lines. In the output layer of RawNet and ShortNet both combine the results of the hidden layers and produce a classifier for the spiral pattern.

# Part 2 Question 9 Part b

We see from the results in the previous questions that for RawNet if we consider initial weight size of 0.1 as our baseline, then smaller weights give us very low accuracy maybe because the weights are diminishing while if we are increase the weight size to 0.2, it gives us a better accuracy than 0.1 but setting to 0.3 gives us a lower accuracy than 0.2. Only when we set our weights to 0.21, we can get 100% accuracy in 7/10 runs. Similarly, if we see the previous results for ShortNet, while considering the values of 5,6 for number of hidden nodes we see a similar trend of 0.2 initial weight size performing better than 0.1 and 0.3 not performing better than 0.2. Only when we set the number of hidden nodes to 7 and initial size of 0.2 than ShortNet can correctly classify 7/10 times.

# Part 2 Question 9 Part C

Looking at the relative “naturalness” of the output function computed by the three networks, we see that PolarNet output function is very close to the actual function which generated the two-spiral pattern. While we look at the output function of RawNet and ShortNet, the output function not as good as PolarNet but if we compare RawNet and ShortNet, ShortNet does a better job than RawNet. The reason PolarNet’s output function is very close to the actual output function is the conversion of input to polar coordinates which is enables PolarNet model to learn the function correctly. The correct representation of input for deep learning task is important because if we represent the input correctly, the model will be able to extract more features and learn the function better. We can also see this from our two spiral pattern that when we converted the input to polar coordinates, the hidden layers in PolarNet were able to learn parts of spiral patterns and ultimately learn the function while this was not the case in RawNet and ShortNet

# Part 2 Question 9 Part D

**PolarNet batchsize** = 194

**Num\_id** = 7

**Training No Epochs Percentage (%)**

1 1700 100

2 3300 100

3 600 100

4 4200 100

5 5800 100

6 1600 100

7 20,000 68.40

8 2000 100

9 20,000 82.99

10 20,000 91.75

**RawNet batchsize** = 194

Initial weight size = 0.21

**Training No Epochs Percentage (%)**

1 20,000 97.94

2 2,400 100

3 20,000 98.97

4 4,100 100

5 1,600 100

6 20,000 98.45

7 20,000 99.48

8 5,500 100

9 12,900 100

10 20,000 97.94

**ShortNet batchsize** = 194

**Num\_id** = 7

**Initial weight size** = 0.20

**Training No Epochs Percentage (%)**

1 2,100 100

2 20,000 93.81

3 20,000 96.39

4 3,400 100

5 3,900 100

6 2,000 100

7 1,100 100

8 1,400 100

9 3,200 100

10 6,600 100

While observing the results of all the three models and changing batch size to 194 and keeping other parameters the same, we can observer that whenever the model achieves 100% accuracy, it does it in less epochs than previous results.

**Optimizer** = SGD

**Batchsize** = 97

**Num\_id** = 7

**PolarNet**

**Training No Epochs Percentage (%)**

1 20,000 73.20

2 20,000 69.07

3 20,000 71.65

4 19,300 70.10

5 20,000 67.01

6 20,000 67.01

7 20,000 66.49

8 20,000 66.49

9 20,000 63.92

10 20,000 68.04

**Optimizer** = SGD

**Batchsize** = 97

**Initial weight size** = 0.21

**RawNet**

**Training No Epochs Percentage (%)**

1 20,000 52.58

2 20,000 52.58

3 20,000 52.06

4 20,000 60.82

5 20,000 59.79

6 20,000 56.70

7 20,000 63.92

8 20,000 61.86

9 20,000 57.22

10 20,000 54.11

**Optimizer** = SGD

**Batchsize** = 97

**Initial weight size** = 0.20

**Num\_id** = 7

**ShortNet**

**Training No Epochs Percentage (%)**

1 20,000 60.31

2 20,000 64.95

3 20,000 64.43

4 20,000 56.70

5 20,000 62.37

6 20,000 58.25

7 20,000 61.34

8 20,000 60.31

9 20,000 56.70

10 20,000 79.38

From the above results of SGD and keeping all the other parameters same, we see that Adam performs much better than SGD on this set of data.