COMP9444 Neural Networks & Deep Learning – Project 1

Name: Zicheng Qu

zID: z5092597

Table of Contents

Part 1: Japanese Character Recognition

Part 1, Q1

[[767. 5. 9. 12. 29. 67. 2. 61. 30. 18.]
 [7. 667. 108. 17. 31. 24. 58. 14. 24. 50.]
 [9. 57. 694. 26. 27. 21. 47. 35. 46. 38.]
 [3. 37. 57. 760. 16. 56. 14. 18. 27. 12.]
 [62. 52. 79. 21. 623. 19. 33. 37. 20. 54.]
 [8. 27. 125. 17. 19. 724. 28. 8. 33. 11.]
 [5. 21. 146. 10. 25. 24. 725. 20. 10. 14.]
 [16. 30. 27. 12. 86. 18. 55. 619. 89. 48.]
 [10. 35. 96. 42. 7. 31. 45. 6. 706. 22.]
 [8. 50. 90. 3. 54. 32. 17. 30. 42. 674.]]

Test set: Average loss: 1.0098, Accuracy: 6959/10000 (70%)

Part 1, Q2

[[845. 2. 3. 3. 29. 36. 2. 43. 31. 6.]
 [6. 811. 30. 2. 22. 8. 66. 7. 16. 32.]
 [9. 8. 854. 34. 12. 17. 21. 11. 21. 13.]
 [4. 9. 28. 923. 1. 16. 5. 1. 6. 7.]
 [40. 26. 16. 5. 815. 6. 28. 24. 22. 18.]
 [9. 14. 77. 10. 12. 831. 21. 2. 18. 6.]
 [3. 12. 45. 8. 18. 4. 897. 4. 2. 7.]
 [17. 12. 22. 4. 24. 7. 29. 835. 21. 29.]
 [10. 29. 31. 52. 3. 9. 29. 4. 827. 6.]
 [1. 18. 50. 2. 29. 4. 21. 18. 13. 844.]]

Test set: Average loss: 0.4966, Accuracy: 8482/10000 (85%)

Part 1, Q3

[[962. 4. 2. 0. 21. 1. 0. 4. 3. 3.]

[1. 924. 5. 1. 11. 1. 37. 4. 6. 10.]

[10. 9. 907. 24. 6. 9. 12. 9. 6. 8.]

[1. 2. 15. 964. 4. 6. 2. 1. 2. 3.]

[16. 11. 3. 5. 942. 1. 8. 4. 8. 2.]

[5. 9. 35. 5. 3. 915. 13. 2. 4. 9.]

[4. 1. 7. 1. 5. 5. 975. 1. 0. 1.]

[3. 1. 4. 0. 6. 0. 7. 956. 6. 17.]

[2. 9. 6. 0. 5. 1. 5. 2. 962. 8.]

[6. 6. 7. 1. 5. 0. 3. 4. 10. 958.]]

Test set: Average loss: 0.2174, Accuracy: 9465/10000 (95%)

Part 1, Q4

From this exercise (Part 1), I learned how to implement a handwriting recognition task with different network models, and learned more details about the performance and accuracy of different models for this task. Also, I noticed some rules or tricks. For example, the output and input between layers should match, and the last batch is not necessarily as set in advance, so the view(-1, pixels*pixels) should be used to complete the shape conversion and so on.

Q4.a

For this handwriting recognition task, my implements' accuracy is 95%, 85% and 70% with NetConv, NetFull and NetLin, respectively. Thus, the three models' relative accuracy from high to low is NetConv, NetFull and NetLin.

Q4.b

For the NetLin, the value in row 7 and column 3 is the largest non-diagonal value, and the value in row 6 and column 3 is the second-largest non-diagonal value. Therefore, in NetLin model, “ma” and “ha” are most likely to be incorrectly identified as “su”, mainly because non-linear features cannot be separated when only a linear function is available.

For the NetFull, the value in row 6 and column 3 is the largest non-diagonal value and the value in row 2 and column 7 is the second-largest non-diagonal value. So in NetFull models, "ha" and “ki” are more likely to be wrongly thought to be "su" and ”ma” respectively. The two fully connected linear layers with activation functions can not only separate linear features and can also take advantage of non-linear features to some extend like convex, but they cannot pull out more delicate features.

For the NetConv, the value in row 2 and column 7 is the largest non-diagonal value and the value in row 6 and column 3 is the second-largest non-diagonal value. Therefore, in the NetConv model, "ki" and “ha” are more likely to be mistakenly considered as "ma" and “tsu” respectively, mainly because the two-layer convolutional neural network can extract features of higher dimensions. Although the accuracy can be greatly improved, it is still unable to separate all data features due to the influence of the model choice, the number of layers or other parameters, etc.

Q4.c

For the architecture, I increased the number of fully connected hidden layers in the NetFull model and the number of fully connected layers after the two convolutional layers in the NetConv model. After trained, I found out that adding a small amount of new hidden layers, such as 1 or 2 layers, improved the two models' prediction accuracy slightly. However, they saturated and even fell down slightly. I think it should cause over-fitting for the training tests. Therefore, I reduced the hidden layers of the NetFull and found the accuracy back to the original level. Instead of reducing the hidden layers, I add nn.Dropout(0.5) to the convolutional layers, which improved the accuracy a little.

As for the meta-parameters, such as momentum (mom) and learning rate (lr), I raised the mom from the basic 0.5, but less than 1, its accuracy increased slightly, and the best prediction result of NetFull model reached 90%. Turning mom down reduces accuracy. The loss will explode to INF if mom is greater than 1. For the lr, I appropriately adjusted the lr from the default 0.01 to 0.1, and the accuracy increased to 89%. However, it decreased to 82% when lr is set to 0.2, mainly due to the high lr leading to fluctuations and high loss.

Part 2: Twin Spirals Task

Part 2, Q1

The conversion code from the (x,y) to coordinates (r,a) can be found in the forward method of the PolarNet module in the spiral.py.

Part 2, Q2

Run the command ’python3 spiral_main.py --net polar --hid 10’ as required, and the picture called polar_out.png is attached below.

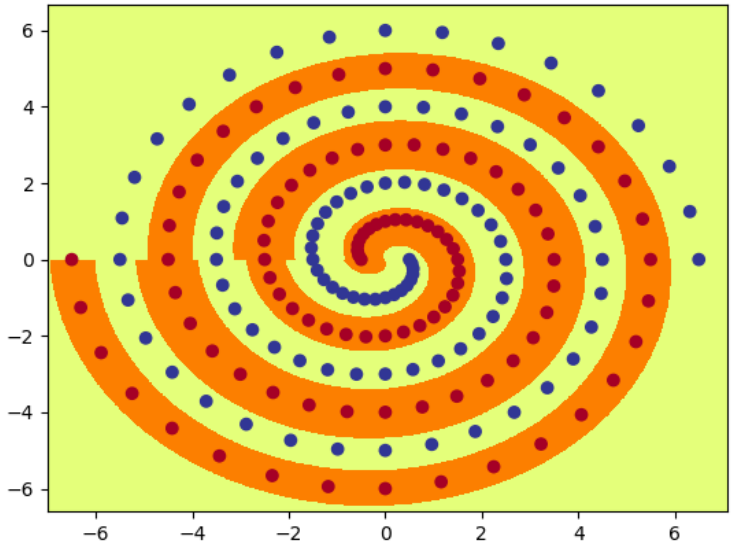

Figure 1:polar_out

Question: Try to find the minimum number of hidden nodes required so that this PolarNet learns to correctly classify all of the training data within 20000 epochs on almost all runs.

Answer: The number of the smallest hidden nodes I found is 7, which can be successfully trained in most trainings within 20000 epochs. The picture called polar_out.png for this ‘’python3 spiral_main.py --net polar --hid 7’ is attached below.

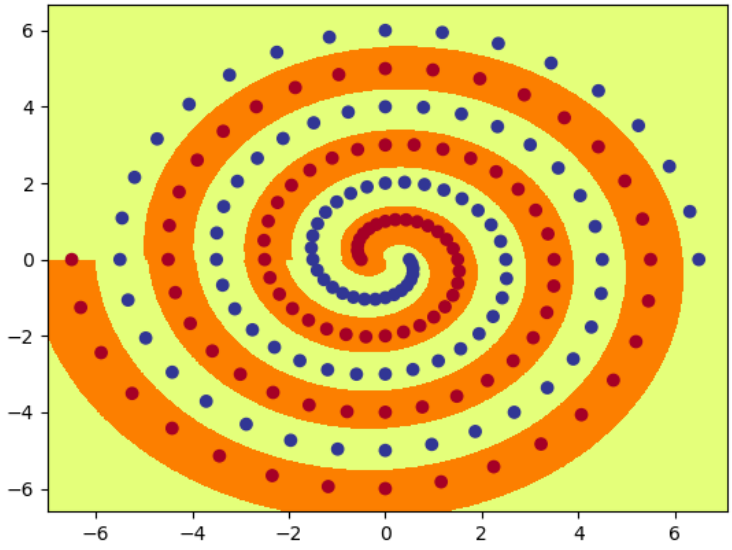

Figure 2: polar_out with hid 7

Part 2, Q3

The code can be found in the RawNet module in the spiral.py.

Part 2, Q4

The number of hidden nodes that I choose is 10 and the initial weights is 0.15. Run the command ‘python3 spiral_main.py –net raw --hid 10 --init 0.15’ and the picture for raw_out.png is below.

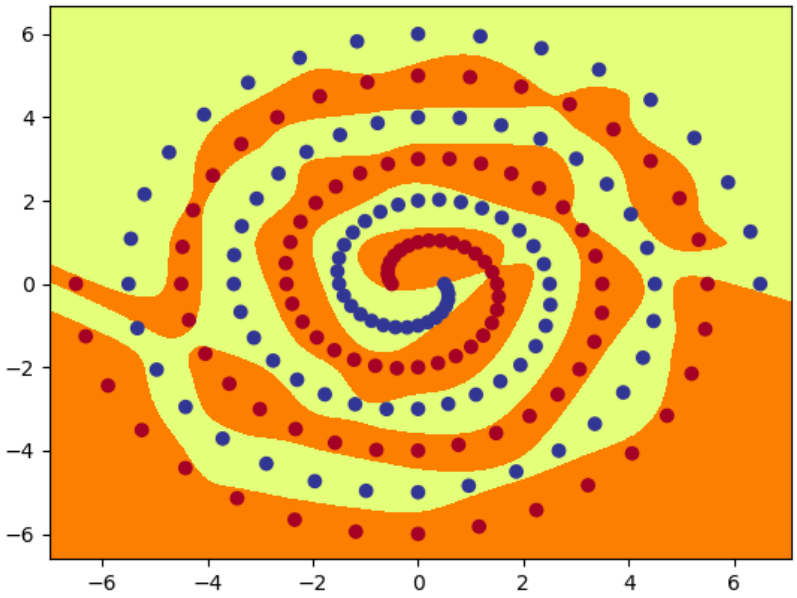

Figure 3: raw_out

Part 2, Q5

The code can be found in the graph_hidden(net,layer,node) in the spiral.py.

The pictures for all the hidden nodes in PolarNet are below:

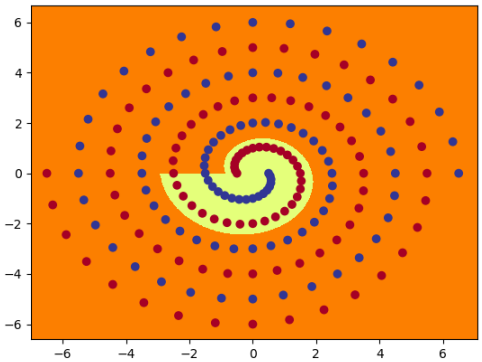

Figure 4: polar1_0

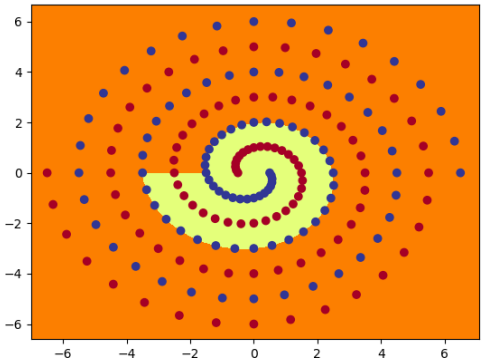

Figure 5: polar1_1

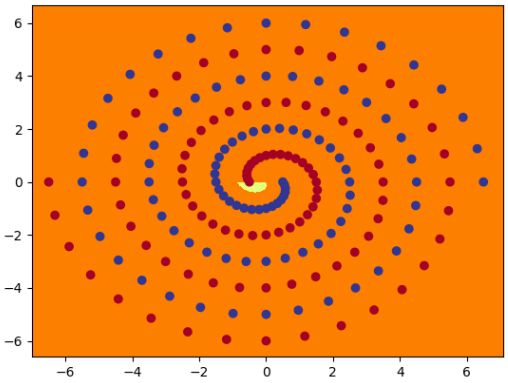

Figure 6:polar1_2

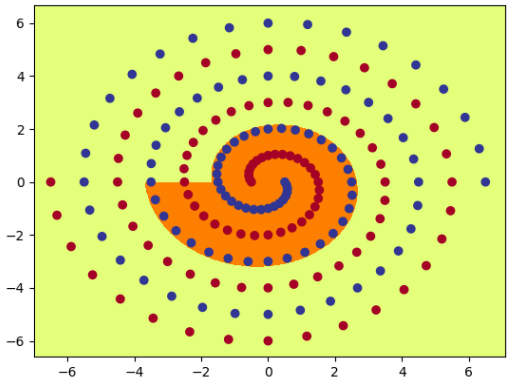

Figure 7: polar1_3

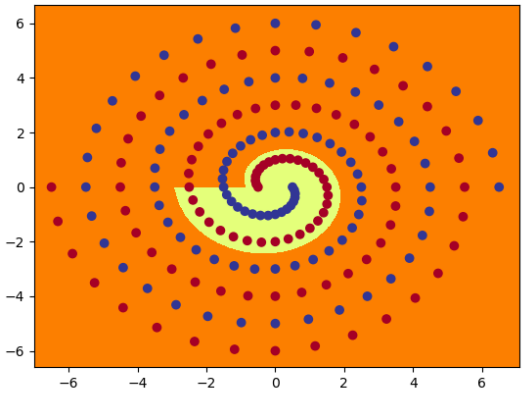

Figure 8: polar1_4

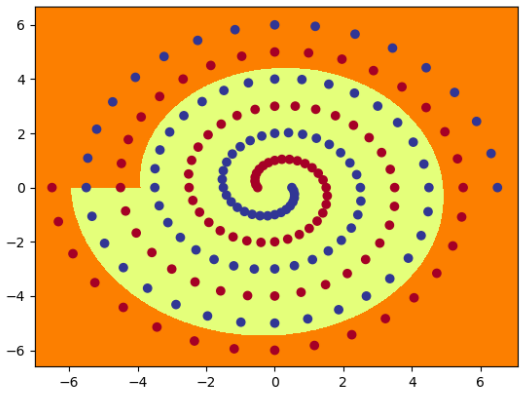

Figure 9: polar1_5

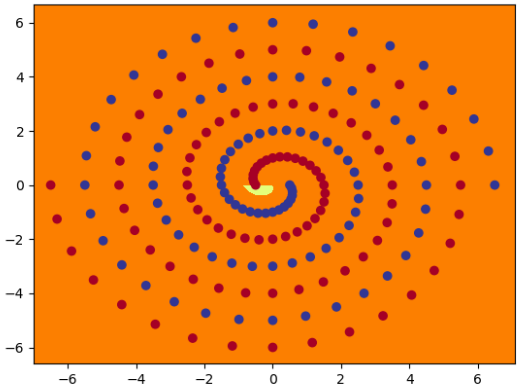

Figure 10: polar1_6

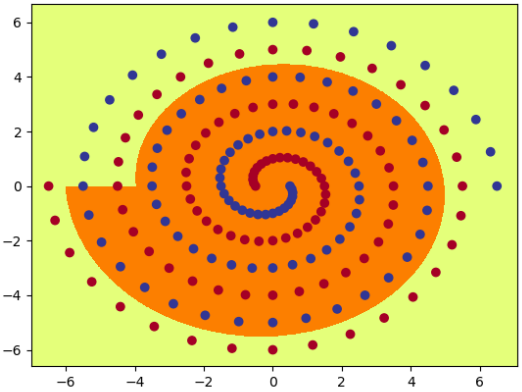

Figure 11: polar1_7

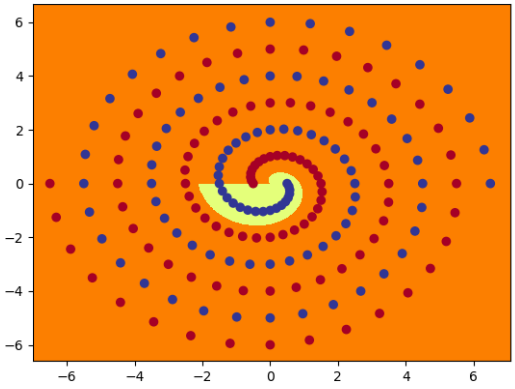

Figure 12: polar1_8

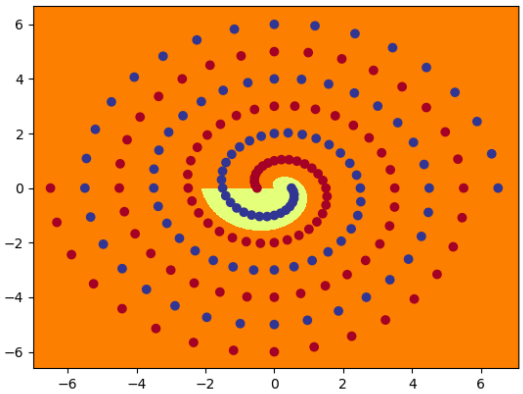

Figure 13: polar1_9

The pictures for all the hidden nodes in both layers of RawNet are below:

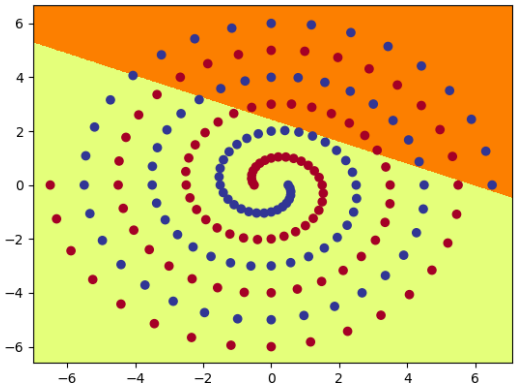

Figure 14: raw1_0

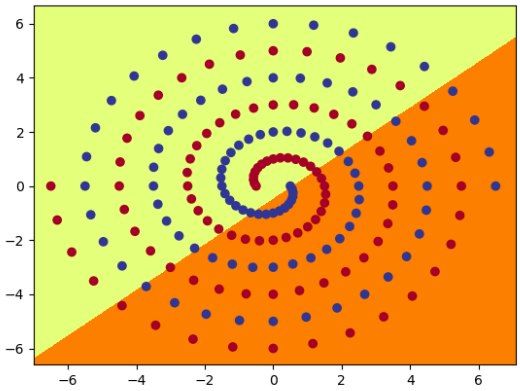

Figure 15: raw1_1

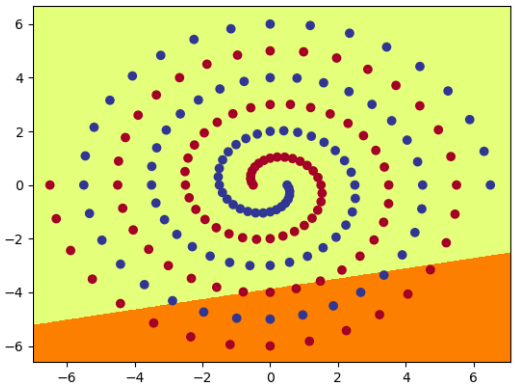

Figure 16: raw1_2

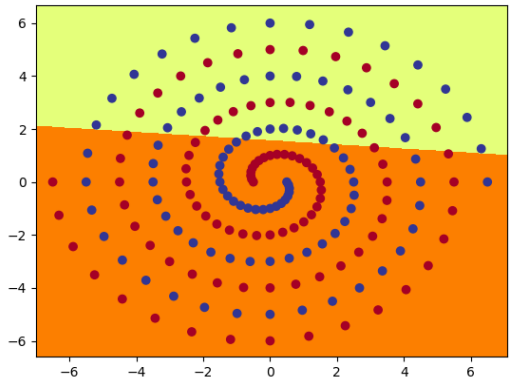

Figure 17: raaw1_3

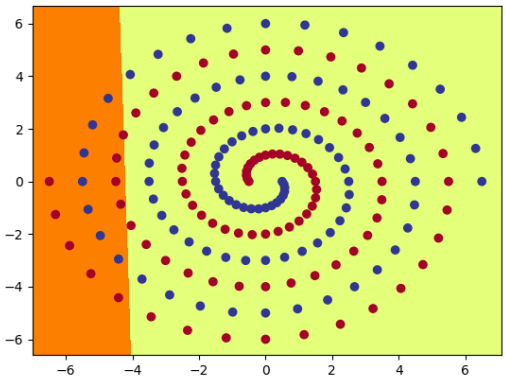

Figure 18: raw1_4

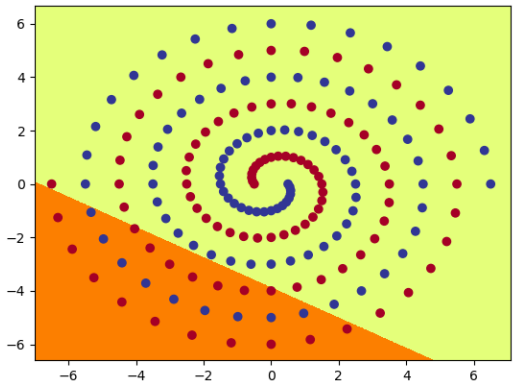

Figure 19: raw1_5

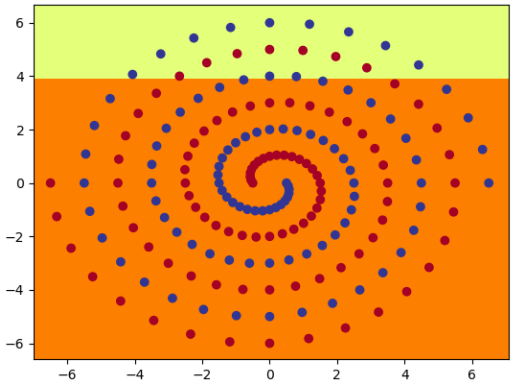

Figure 20: raw1_6

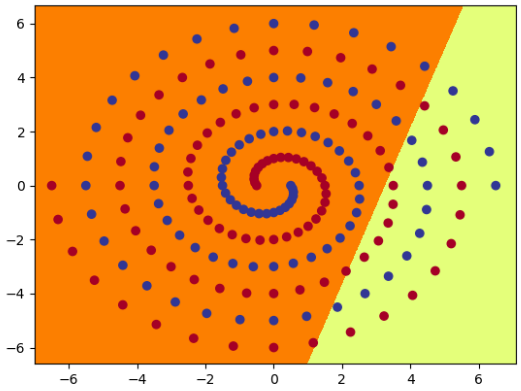

Figure 21: raw1_7

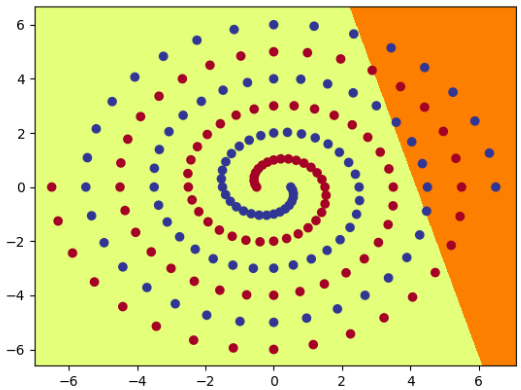

Figure 22: raw1_8

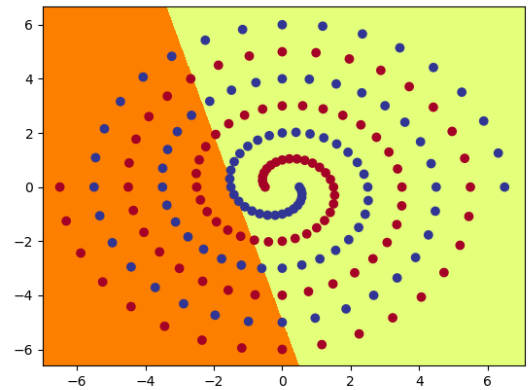

Figure 23: raw1_9

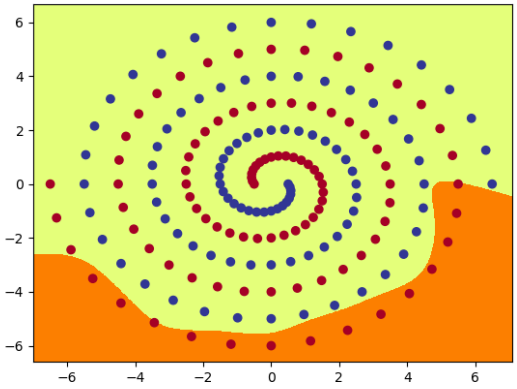

Figure 24: raw2_0

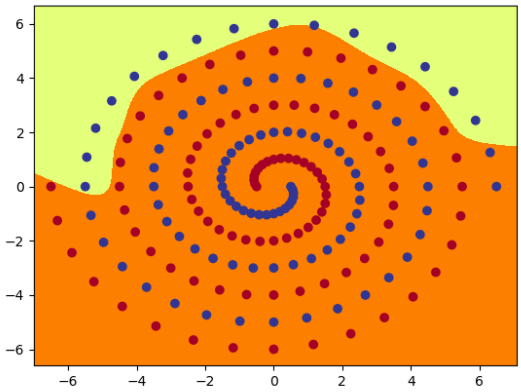

Figure 25: raw2_1

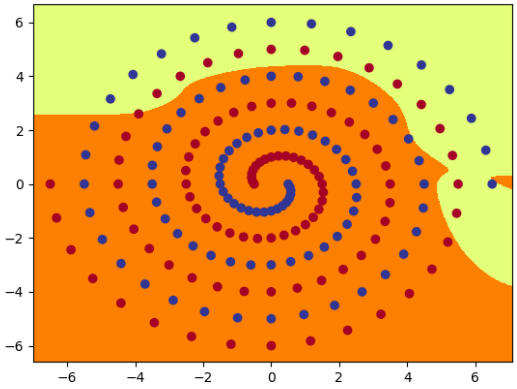

Figure 26: raw2_2

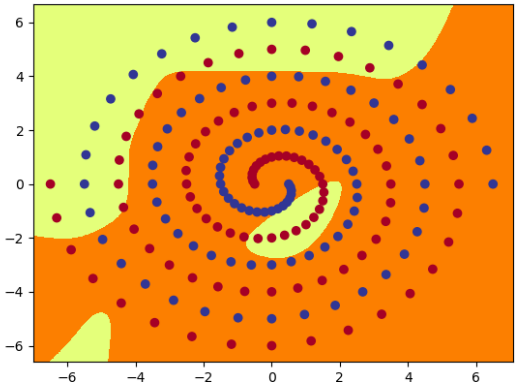

Figure 27: raw2_3

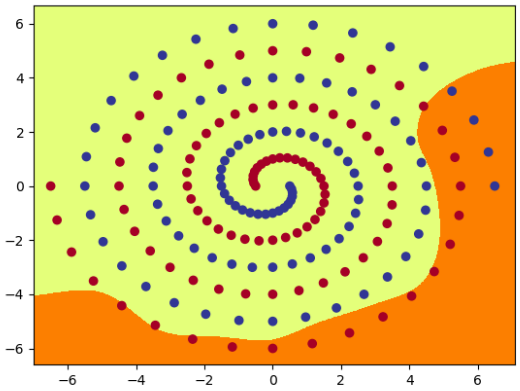

Figure 28: raw2_4

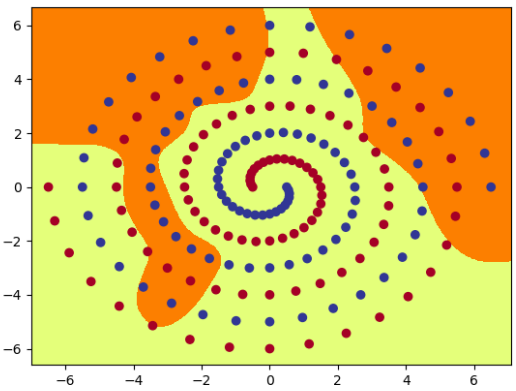

Figure 29: raw2_5

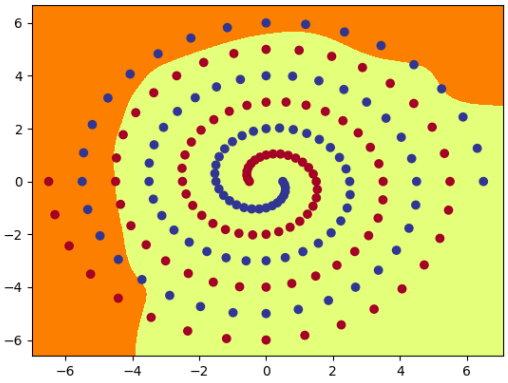

Figure 30: raw2_6

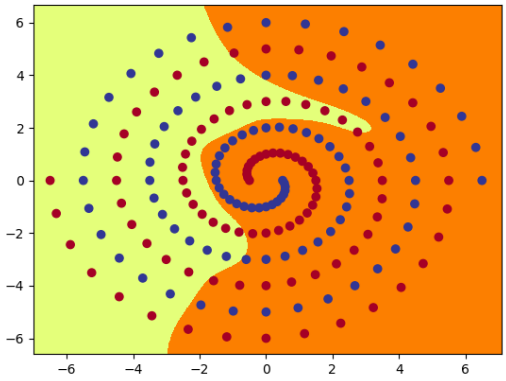

Figure 31: raw2_7

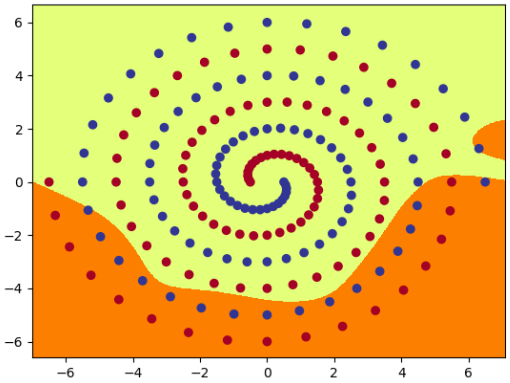

Figure 32: raw2_8

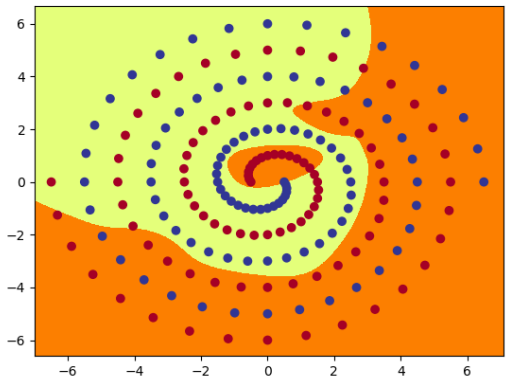

Figure 33: raw2_9

Part 2, Q6

From this exercise (Part 2), I have a better understanding of the difference between the one hidden and two hidden layers and a deeper understanding of the initial parameters such as weights. Also, I practiced more optimization mechanisms and batch choice.

Q6.a

The hidden layer of PolarNet is intended to do the job related to linearly separable features, since it has only one hidden layer. As for the RawNet, there are two hidden layers, so its first hidden layer is to solve the linearly separable problem, and its second hidden layer is to learn convex features.

As for how the network achieves the classification, there are some different components, including the first hidden, second hidden, and output layer. The first hidden layer mainly learns linearly separable functions from the inputs. The second hidden layer is mostly linear, but involves some degree of non-linearity for convex function. Thus, the boundary produced via the

second hidden layer is convex, and passing it to the next layer. The output layer (third layer) can get the concave function to make the classes separable.

Q6.b

When training the model RawNet, I chose 0.1 to initialize the weights as normal distributions in the code provided in the source file 'spiral_main.py'. If the initial weight is too large, the transfer function will approximate a step function and lead to digital computation and low learning speed. If the initial weight is too small, it will make each layer perform like a linear function, so it may not complete the non-linear separable tasks.

Q6.c

According to the recommendation, the batch size was increased from 97 to 194, and I found that the training speed was a little faster than before. After googled the reason, I know the increase of batch size is conducive to reducing the training time and improving the stability, but the substantial growth of batch size will reduce the generalization ability of the model. In this part 2, I tried to replace Adam with SGD, but it failed to converge in almost all the attempts, and I guessed that it might stop at some local optimal solution or saddle point. In some training, the pictures produced were not ideal when Relu replaced the activation function tanh. Besides, adding an extra layer doesn't necessarily lead to better results.

Part 3: Hidden Unit Dynamics

Part 3, Q1

Run the command 'Python3 encoder_main.py --target=star16' and the final image is attached below.

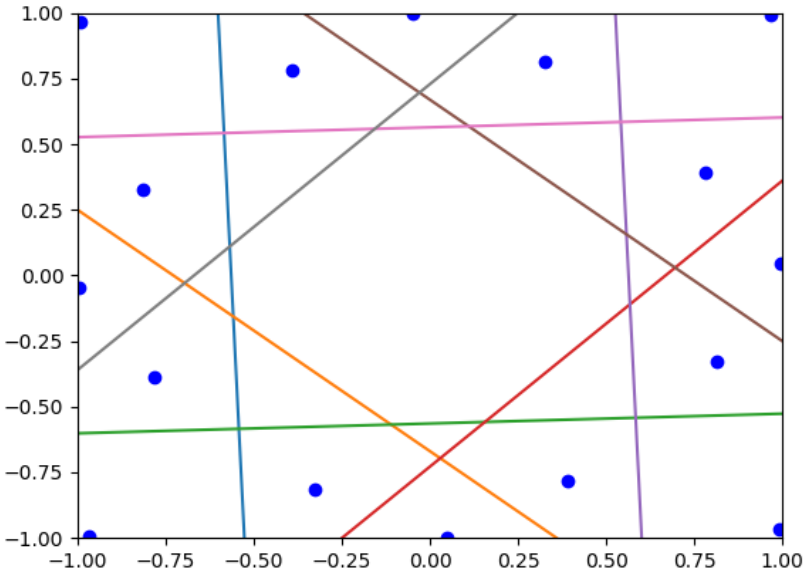

Figure 34: star16

Part 3, Q2

In this 9-2-9 encoder, each row represents a dot and each column represents a line (output boundary). Since the 9-2-9 encoder is the identity matrix generated using the torch.eye(args.eye), each dot is separated by a line. In the first few steps, generating 9 dots first and spreads out gradually. And then generate the corresponding lines around the dots. Gradually, the number of lines will reach 9 and will separate the 9 dots.

Run the command ‘python3 encoder_main.py --target=input --dim=9 --plot’, and the pictures are attached below.

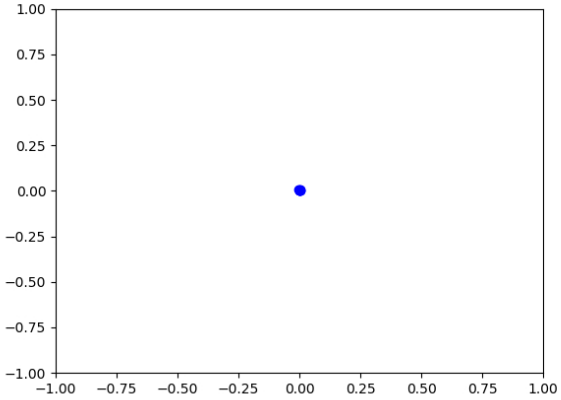

Figure 35: input_1

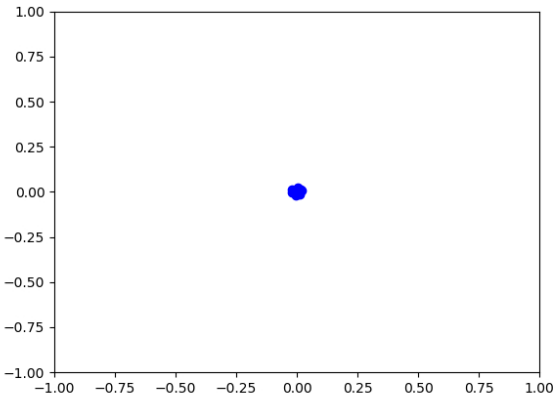

Figure 36: input_2

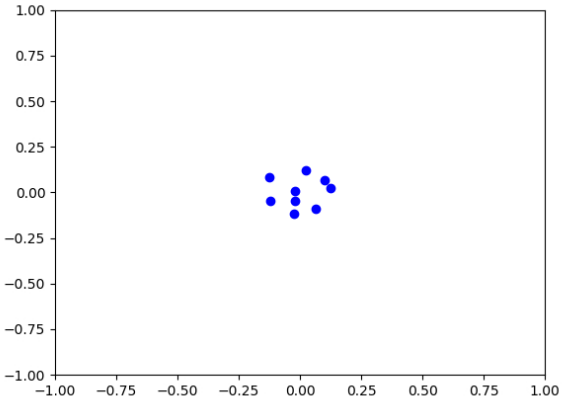

Figure 37: input_3

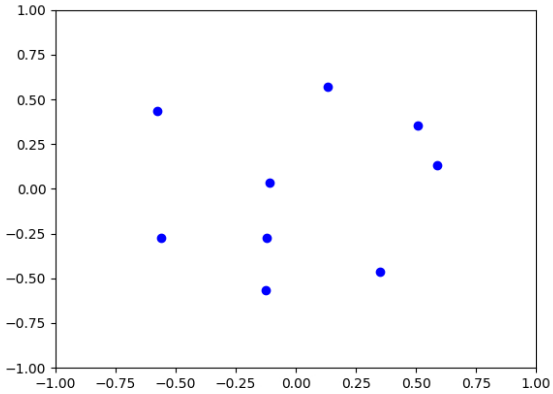

Figure 38: input_4

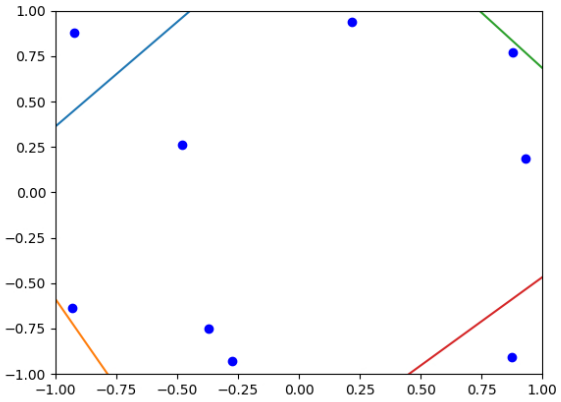

Figure 39: input_5

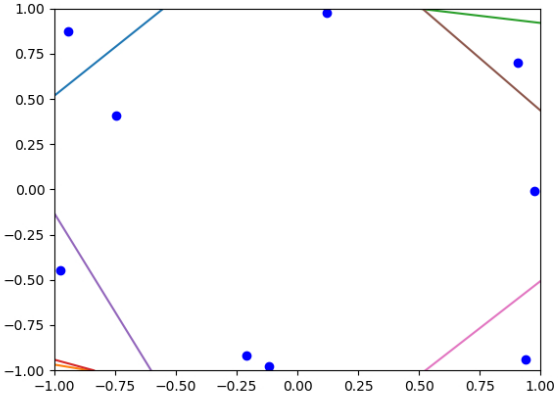

Figure 40: input_6

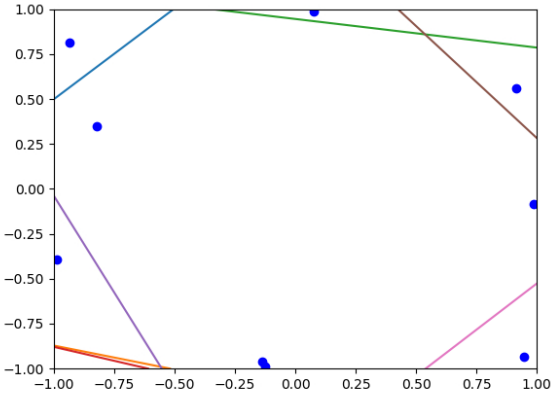

Figure 41: input_7

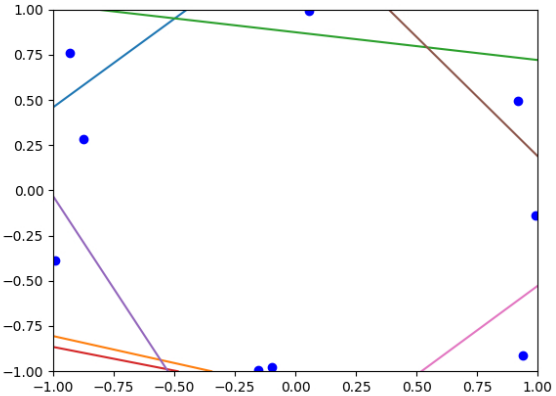

Figure 42: input_8

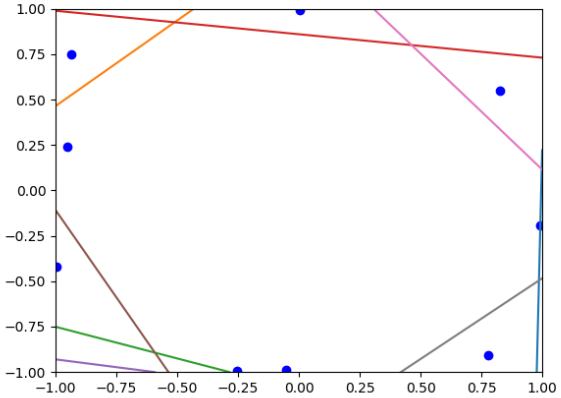

Figure 43: input_9

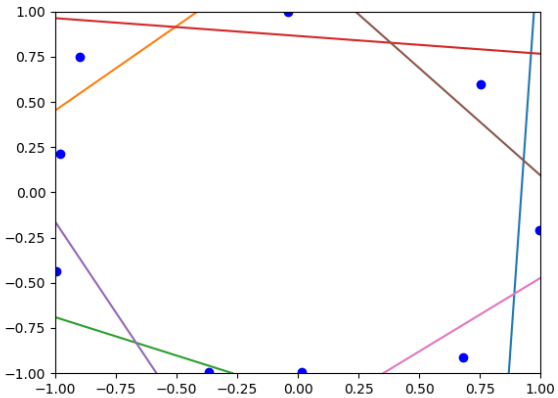

Figure 44: input_10

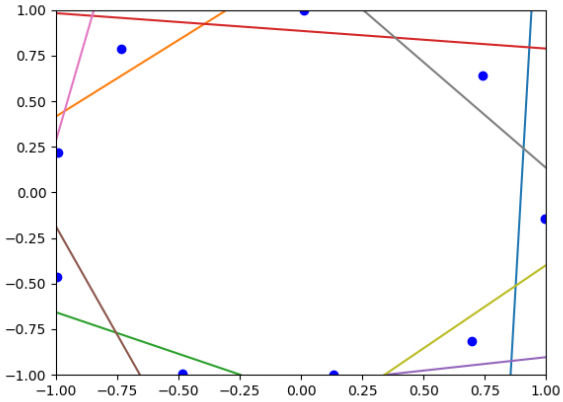

Figure 45: input_11

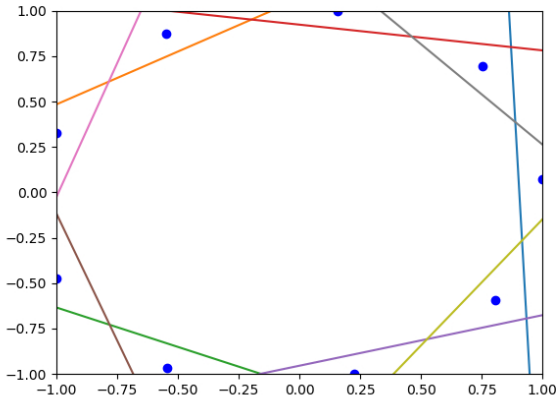

Figure 46: input_final_image

Part 3, Q3

The picture for heart18 is attached below and the tensor heart18 is in the encoder.py.

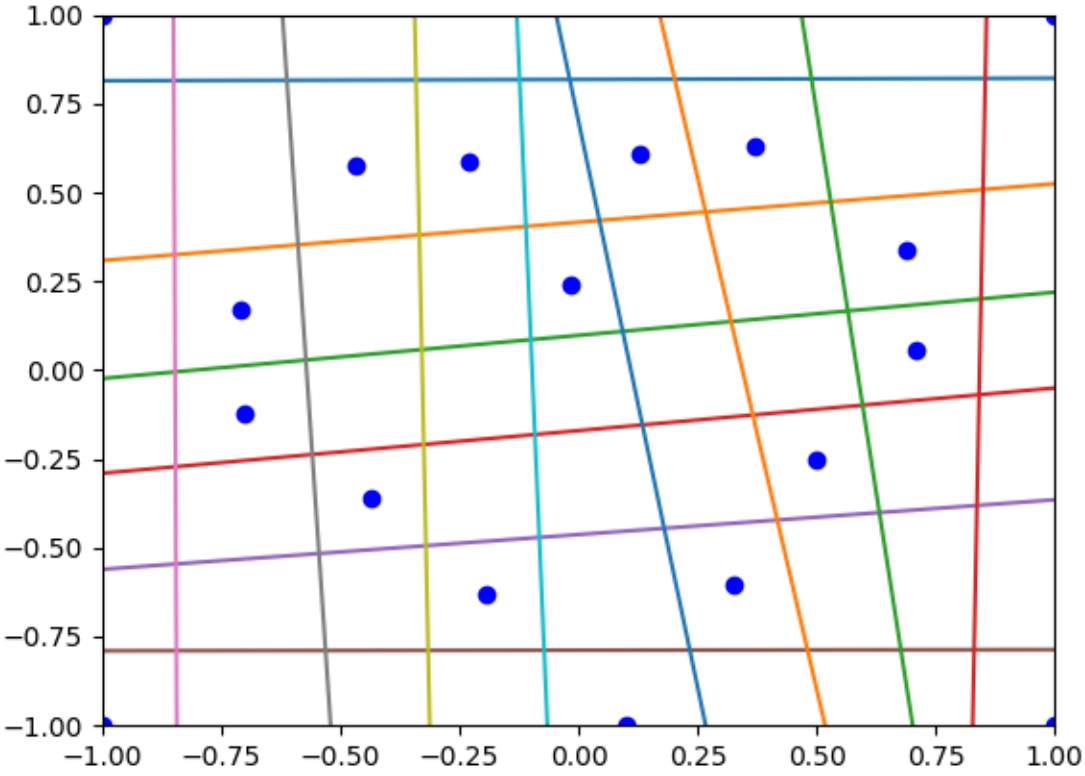

Figure 47: heart18

Part 3, Q4

Run the command ‘python3 encoder_main.py --target=target1’, generate a picture for Apple's logo, and the tensor target1 is in the encoder.py.

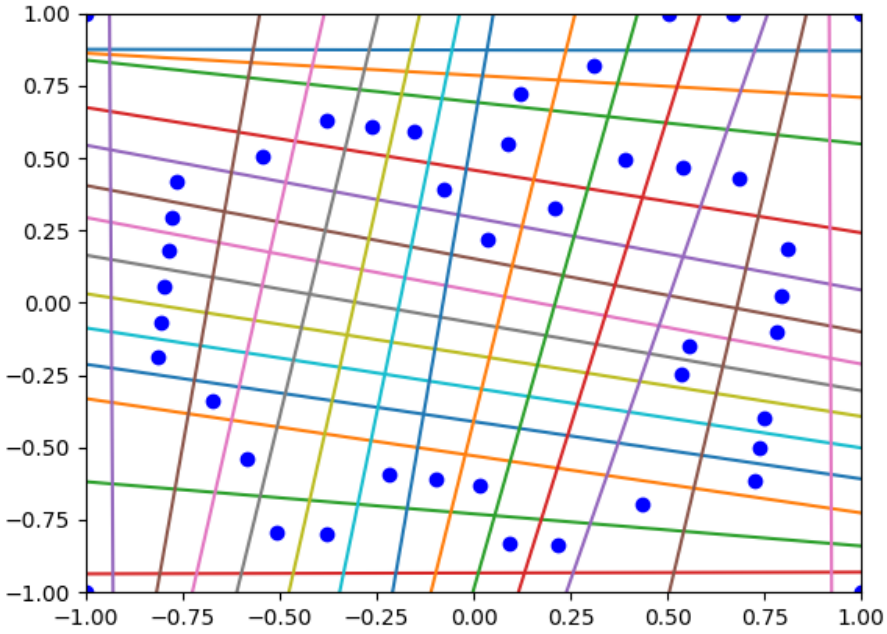

Figure 48: Apple logo

Run the command ‘python3 encoder_main.py --target=target2’, generate a picture for a simple tree, and the tensor target2 is in the encoder.py.

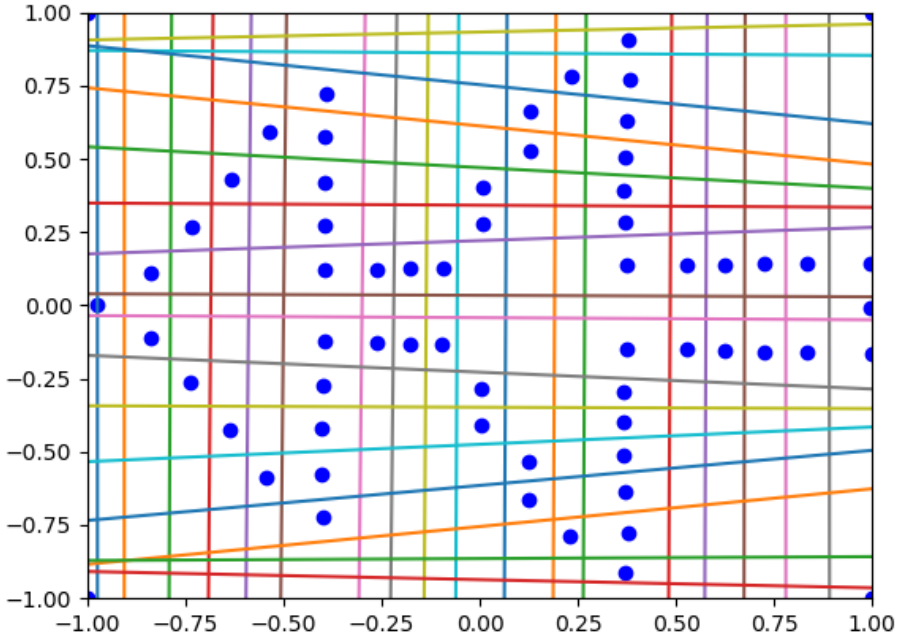

Figure 49: Tree