Neural Networks

COMP9444



Assignment 1

T3, 2020

Kovid Sharma

z5240067

[k.sharma.1@student.unsw.edu.au](mailto:k.sharma.1@student.unsw.edu.au)

**Table of Contents**

**Part 1 2**

Question 1 2

Question 2 2

Question 3 3

Question 4 4

**Part 2 6**

Question 1 6

Question 2 6

Question 3 8

Question 4 8

Question 5 10

Question 6 14

**Part 3 16**

Question 1 16

Question 2 17

Question 3 19

Question 4 20

**References 23**

**Part 1: Japanese Character Recognition**

**Question 1.** Experiments done below using various changes to lr and mom. **Confusion matrix is on page 4**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **# Run** | **lr** | **mom** | **Accuracy of Test Set** | | | | | | | | | | **Time**  **(mins)** |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 0.01 | 0.5 | 67 | 68 | 69 | 69 | 69 | 69 | 69 | 69 | 70 | 70 | 3.95 |
| 2 | 0.01 | 0.5 | 67 | 68 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 70 | 3.91 |
| 3 | 0.01 | 0.5 | 67 | 68 | 69 | 69 | 69 | 69 | 69 | 70 | 70 | 70 | 4.14 |
| 4 | 0.01 | 0.6 | 67 | 69 | 69 | 69 | 70 | 70 | 70 | 70 | 70 | 70 | 3.20 |
| 5 | 0.01 | 0.65 | 68 | 69 | 69 | 69 | 70 | 70 | 70 | 70 | 70 | 70 | 3.14 |
| 6 | 0.01 | 0.7 | 68 | 69 | 69 | 70 | 70 | 70 | 70 | 70 | 70 | 70 | 2.95 |
| 7 | **0.01** | **0.8** | 69 | 69 | 70 | 70 | 70 | 70 | 70 | 70 | 70 | 70 | 3.15 |
| 8 | 0.02 | 0.8 | 67 | 67 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 3.12 |
| 9 | 0.04 | 0.8 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 3.09 |
| 10 | 0.01 | 0.9 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 2.97 |
| 11 | 0.01 | 0.1 | 66 | 67 | 68 | 69 | 69 | 69 | 69 | 69 | 69 | 70 | 3.03 |
| 12 | 0.08 | 0.5 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 2.80 |
| 13 | 0.1 | 0.5 | 67 | 68 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 3.05 |
| 14 | 0.1 | 0.4 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 3.13 |
| 15 | 0.1 | 0.3 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 3.11 |
| 16 | 0.1 | 0.2 | 66 | 67 | 67 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 3.05 |
| 17 | 0.1 | 0.1 | 66 | 67 | 68 | 67 | 68 | 67 | 67 | 67 | 67 | 67 | 3.12 |

**Question 2.** Experiments done below using various changes to hid, lr and mom. **Confusion matrix is on page 4**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **# Run** | **# Hidden Layers** | **lr** | **mom** | **Accuracy of Test Set** | | | | | | | | | | **Time**  **(mins)** |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 30 | 0.01 | 0.5 | 67 | 71 | 73 | 74 | 76 | 77 | 78 | 78 | 79 | 79 | 4.29 |
| 2 | 60 | 0.01 | 0.5 | 69 | 72 | 75 | 77 | 78 | 80 | 81 | 81 | 82 | 82 | 4.25 |
| 3 | 90 | 0.01 | 0.5 | 68 | 72 | 75 | 77 | 79 | 80 | 81 | 82 | 83 | 84 | 4.26 |
| 4 | 150 | 0.01 | 0.1 | 66 | 69 | 72 | 73 | 75 | 77 | 78 | 79 | 80 | 81 | 3.93 |
| 5 | 150 | 0.01 | 0.5 | 69 | 73 | 76 | 78 | 80 | 81 | 82 | 83 | 84 | 85 | 4.28 |
| 6 | 150 | 0.01 | 0.6 | 69 | 74 | 77 | 80 | 81 | 82 | 84 | 84 | 85 | 85 | 3.40 |
| 7 | 150 | 0.01 | 0.7 | 71 | 76 | 79 | 81 | 83 | 84 | 85 | 86 | 86 | 86 | 3.34 |
| 8 | 150 | 0.01 | 0.8 | 74 | 79 | 83 | 84 | 86 | 86 | 87 | 87 | 88 | 88 | 3.92 |
| 9 | **150** | **0.02** | **0.8** | 79 | 83 | 85 | 86 | 87 | 87 | 88 | 88 | 88 | 88 | 3.19 |
| 10 | 150 | 0.04 | 0.8 | 81 | 85 | 86 | 86 | 86 | 86 | 87 | 87 | 87 | 88 | 3.33 |
| 11 | 200 | 0.01 | 0.5 | 69 | 73 | 76 | 78 | 80 | 81 | 82 | 83 | 84 | 85 | 4.25 |
| 12 | 300 | 0.01 | 0.5 | 68 | 72 | 75 | 78 | 80 | 82 | 83 | 84 | 85 | 85 | 4.65 |
| 13 | 500 | 0.01 | 0.5 | 68 | 72 | 75 | 78 | 80 | 81 | 82 | 83 | 84 | 85 | 4.93 |
| 14 | 600 | 0.01 | 0.5 | 68 | 72 | 75 | 77 | 79 | 81 | 82 | 83 | 84 | 85 | 5.03 |
| 15 | 600 | 0.01 | 0.8 | 73 | 79 | 82 | 84 | 86 | 87 | 87 | 88 | 88 | 89 | 3.91 |

**Question 3.** Experiments done below using various changes to c1,c2, l1,l2, lr and mom. **Confusion matrix is on page 4**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#Run** | **lr** | **mom** | **c1, c2** | **l1, l2** | **Accuracy of Test Set** | | | | | | | | | | **Time**  **(mins)** |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 0.01 | 0.5 | 10,20 | 320,106 | 77 | 85 | 88 | 90 | 92 | 92 | 93 | 93 | 93 | 93 | 20.87 |
| 2 | 0.1 | 0.5 | 10,20 | 320,256 | 89 | 91 | 92 | 93 | 93 | 94 | 94 | 94 | 94 | 94 | 19.29 |
| 3 | 0.1 | 0.5 | 10,20 | 320,420 | 88 | 91 | 92 | 93 | 93 | 94 | 94 | 95 | 95 | 94 | 17.64 |
| 4 | 0.1 | 0.5 | 10,20 | 320,650 | 89 | 91 | 92 | 92 | 93 | 93 | 93 | 94 | 94 | 93 | 16.38 |
| 5 | 0.1 | 0.5 | 10,50 | 800,256 | 90 | 94 | 94 | 95 | 95 | 95 | 96 | 96 | 95 | 95 | 23.37 |
| 6 | 0.1 | 0.5 | 10,70 | 1120,256 | 91 | 94 | 94 | 95 | 95 | 95 | 94 | 96 | 95 | 96 | 26.73 |
|  | | | | | | | | | | | | | | | |
| 7 | 0.1 | 0.5 | 12,24 | 384,128 | 88 | 91 | 93 | 93 | 93 | 94 | 94 | 94 | 94 | 93 | 21.87 |
| 8 | 0.1 | 0.1 | 12,24 | 384,128 | 90 | 93 | 94 | 94 | 94 | 95 | 95 | 95 | 95 | 95 | 22.04 |
|  | | | | | | | | | | | | | | | |
| 9 | 0.1 | 0.1 | 14,28 | 448,149 | 91 | 93 | 94 | 95 | 95 | 95 | 96 | 96 | 95 | 96 | 20.75 |
| 10 | 0.1 | 0.1 | 14,28 | 448,149 | 91 | 93 | 94 | 95 | 95 | 95 | 96 | 96 | 95 | 96 | 20.75 |
| 11 | 0.01 | 0.5 | 14,28 | 448,650 | 80 | 88 | 91 | 92 | 93 | 94 | 94 | 95 | 95 | 95 | 27.28 |
| 12 | 0.1 | 0.9 | 14,28 | 448,650 | 74 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 25.36 |
| 13 | 0.1 | 0.5 | 14,28 | 448,650 | 90 | 92 | 93 | 93 | 94 | 94 | 94 | 94 | 94 | 95 | 26.68 |
| 14 | 0.1 | 0.4 | 14,28 | 448,650 | 92 | 94 | 95 | 95 | 95 | 96 | 95 | 96 | 96 | 96 | 21.82 |
| 15 | 0.1 | 0.3 | 14,28 | 448,650 | 90 | 93 | 94 | 95 | 95 | 95 | 95 | 94 | 95 | 95 | 22.00 |
| 16 | 0.1 | 0.2 | 14,28 | 448,650 | 90 | 93 | 94 | 95 | 95 | 95 | 95 | 95 | 96 | 96 | 26.03 |
| 17 | 0.1 | 0.1 | 14,28 | 448,650 | 91 | 93 | 94 | 95 | 94 | 95 | 95 | 95 | 96 | 96 | 22.08 |
| 18 | 0.2 | 0.5 | 14,28 | 448,650 | 87 | 90 | 91 | 90 | 91 | 92 | 91 | 90 | 91 | 92 | 31.97 |
| 19 | 0.3 | 0.5 | 14,28 | 448,650 | 79 | 84 | 83 | 87 | 79 | 75 | 76 | 74 | 77 | 81 | 28.45 |
| 20 | 0.4 | 0.5 | 14,28 | 448,650 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 28.87 |
| 21 | 0.5 | 0.5 | 14,28 | 448,650 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 27.99 |
| 22 | 1 | 0.5 | 14,28 | 448,650 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 27.85 |
|  | | | | | | | | | | | | | | | |
| 23 | 0.01 | 0.5 | 16,32 | 512,650 | 82 | 89 | 91 | 93 | 93 | 94 | 94 | 95 | 95 | 95 | 30.52 |
| 24 | 0.1 | 0.5 | 16,32 | 512,650 | 90 | 93 | 95 | 93 | 95 | 95 | 96 | 94 | 96 | 96 | 26.10 |
| 25 | 0.1 | 0.4 | 16,32 | 512,650 | 92 | 94 | 94 | 94 | 95 | 95 | 95 | 95 | 96 | 96 | 23.82 |
| 26 | 0.1 | 0.3 | 16,32 | 512,650 | 90 | 92 | 94 | 95 | 96 | 95 | 96 | 96 | 96 | 96 | 23.75 |
| 27 | 0.1 | 0.2 | 16,32 | 512,650 | 89 | 93 | 94 | 95 | 95 | 95 | 95 | 96 | 96 | 96 | 24.31 |
| 28 | 0.1 | 0.1 | 16,32 | 512,650 | 90 | 93 | 94 | 95 | 95 | 96 | 96 | 96 | 96 | 96 | 24.30 |
| 29 | 0.1 | 0.1 | 16,32 | 512,170 | 89 | 93 | 93 | 94 | 94 | 95 | 95 | 95 | 95 | 95 | 23.25 |
| 30 | 0.1 | 0.09 | 16,32 | 512,650 | 91 | 94 | 94 | 95 | 95 | 95 | 96 | 96 | 95 | 96 | 24.65 |
| 31 | 0.1 | 0.08 | 16,32 | 512,650 | 88 | 93 | 94 | 94 | 94 | 95 | 96 | 95 | 95 | 96 | 25.64 |
| 32 | 0.1 | 0.05 | 16,32 | 512,650 | 90 | 93 | 95 | 95 | 95 | 95 | 96 | 96 | 96 | 96 | 25.85 |
| 33 | 0.2 | 0.5 | 16,32 | 512,650 | 90 | 91 | 92 | 92 | 94 | 94 | 94 | 92 | 94 | 92 | 25.70 |
|  | | | | | | | | | | | | | | | |
| 34 | 0.1 | 0.1 | 32,64 | 1024,341 | 91 | 94 | 95 | 96 | 96 | 96 | 96 | 96 | 97 | 96 | 47.71 |
| 35 | 0.1 | 0.1 | 32,64 | 1024,512 | 92 | 94 | 95 | 95 | 96 | 96 | 96 | 96 | 96 | 96 | 45.20 |
| 36 | 0.1 | 0.1 | 32,64 | 1024,1k | 92 | 94 | 95 | 96 | 96 | 96 | 96 | 96 | 96 | 96 | 45.94 |
| 37 | 0.1 | 0.3 | 32,64 | 1024,512 | 91 | 93 | 95 | 95 | 96 | 96 | 95 | 96 | 96 | 96 | 45.87 |
| 38 | 0.1 | 0.3 | 32,64 | 1024,1k | 92 | 94 | 94 | 95 | 96 | 95 | 96 | 96 | 96 | 96 | 46.27 |
| 39 | **0.1** | **0.1** | 32,64 | 1024,1k | 92 | 94 | 94 | 95 | 95 | 96 | 96 | 96 | 97 | 97 | 44.87 |

**Question 4a.**

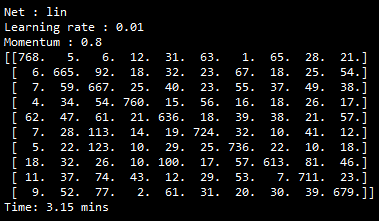
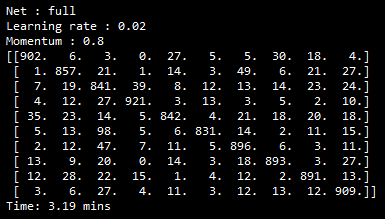
|  |  |  |  |
| --- | --- | --- | --- |
|  | **NetLin** | **NetFull** | **NetConv** |
| **Performance** | 70% | 88% | 97% |
| **Time** | 3.15 | 3.19 | 44.87 |

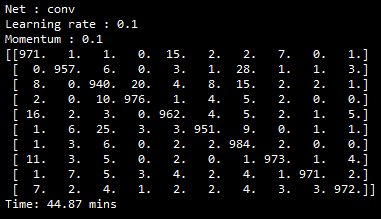
From the table above we can see that the accuracy of single linear network is relatively low, a 2-layer

fully-connected network performs better than the Linear network but a convolutional network achieves relatively the highest accuracy of all three networks. Ergo it’s clear that a pure linear or fully-connected network is not quite suitable for image/handwriting recognition because of inadequate fitting ability. Convolutional neural networks are the best choice for such tasks, they are also computationally expensive and take a lot of time to run.

**Question 4b.**

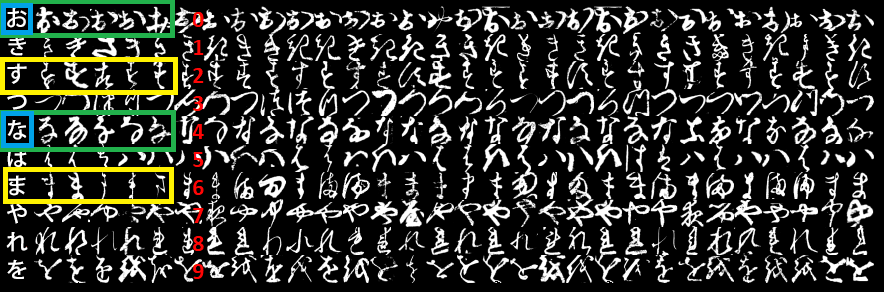
Confusion Matrix : to understand this better, I will make use of confusion pairs, i.e. what character was misinterpreted as another character. This confusion pair is simply calculated by looking at a row in a confusion matrix and comparing the largest number (right identification) with the second largest number in that row (character confused with). Indices start from 0 and end at 9 as displayed in the table below on the next page.



Confusion Matrices for Lin(top-left), Full(top-right), Conv(Bottom)

|  |  |  |  |
| --- | --- | --- | --- |
| **# Character** | **NetLin** | **NetFull** | **NetConv** |
| 0="o" | 0-7 | 0-7 | 0-4 |
| 1="ki" | 1-2 | 1-2 | 1-6 |
| 2="su" | 2-1 | 2-3 | 2-3 |
| 3="tsu" | 3-5 | 3-2 | 3-2 |
| 4="na" | 4-1 | 4-1 | 4-0 |
| 5="ha" | 5-2 | 5-2 | 5-0 |
| 6="ma" | 6-2 | 6-2 | 6-2 |
| 7="ya" | 7-4 | 7-9 | 7-0 |
| 8="re" | 8-2 | 8-1 | 8-1 |
| 9="wo" | 9-2 | 9-2 | 9-0 |



In this image we can see the 10 classes of Kuzushiji-MNIST, with the first column showing each character's modern hiragana counterpart.

The numbers in red represent the 10 classes of the dataset.

**Green and yellow** boxes are the ones that are confused quite often for all three models.

If you just focus on the **blue box**, one can see how similar those characters look like and how easy it would be to misinterpret one as the other.

**Question 4c.**

I’ve already added the tables with various experiments in questions 1,2 and 3 on pages 2 and 3.

For **NetLin**, I tried first changing the momentum and then the learning rate. It can be seen in the table on page 2 that the model performs best at default lr = 0.01 and mom = 0.8.

It can be concluded that the low performance of NetLin is mainly comes from the architecture itself instead of parameter turning.

Similarly, for **NetFull**, I found the best accuracy of 88% at lr = 0.02 and mom = 0.9. Experiments can be found in table on page 2.

For **NetConv**, I tried various nodes of 10x20, 10x50, 10x70, 12x24, 14x28, 16x32 and 32x64. I found that using more nodes leads to better overall accuracy and stability, while increasing the processing time. For each of them, I first increased the momentum from 0.5 to 0.6, 0.7, 0.8 and 0.9 and found that increasing the mom results in decreased accuracy. Then gradually lowered momentum with values 0.4, 0.3, 0.2, 0.1, 0.09, 0.08 and 0.05. It performs best in the 0.08-0.1 range, which is lower than the default setting. Table for these experiments can be found on page 3. In the tables on page 2 and 3, red colour rows mean worst performing, green the best performing & yellow good accuracy.

**Part 2: Twin Spirals Task**

**Question 1.**

See code in spiral.py

**Question 2.**

**PolarNet** - The ones highlighted in green are successful runs (100% accuracy)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#Run** | **init** | **#hidden** | **lr** | **Accuracy** | **Epochs** | **Time (mins)** |
| 1 | 0.001 | 5 | 0.01 | 49.48 | 100k | 9.8 |
| 2 | 0.01 | 5 | 0.01 | 76.80 | 100k | 9.77 |
| 3 | 0.1 | 5 | 0.01 | 77.48 | 100k | 10.24 |
| 4 | 0.2 | 5 | 0.01 | 77.84 | 100k | 10.14 |
| 5 | 0.3 | 5 | 0.01 | 59.28 | 100k | 10.19 |
|  | | | | | | |
| 6 | 0.001 | 6 | 0.01 | 49.48 | 100k | 11.06 |
| 7 | 0.01 | 6 | 0.01 | 85.05 | 100k | 11.08 |
| 8 | 0.1 | 6 | 0.01 | 100 | 8900 | 1.03 |
| 9 | 0.2 | 6 | 0.01 | 100 | 6700 | 0.82 |
| 10 | 0.3 | 6 | 0.01 | 91.75 | 100k | 11.01 |
|  | | | | | | |
| 11 | 0.001 | 7 | 0.01 | 49.48 | 100k | 10.21 |
| 12 | 0.01 | 7 | 0.01 | 100 | 4600 | 0.63 |
| 13 | 0.1 | 7 | 0.01 | 100 | 11500 | 1.59 |
| 14 | 0.2 | 7 | 0.01 | 100 | 3000 | 0.32 |
| 15 | 0.3 | 7 | 0.01 | 87.63 | 100k | 10.77 |
|  | | | | | | |
| 16 | 0.001 | 8 | 0.01 | 49.48 | 100k | 10.79 |
| 17 | 0.01 | 8 | 0.01 | 100 | 9300 | 1.06 |
| 18 | 0.1 | 8 | 0.01 | 100 | 3400 | 0.44 |
| 19 | 0.2 | 8 | 0.01 | 100 | 5100 | 0.51 |
| 20 | 0.3 | 8 | 0.01 | 100 | 2900 | 0.30 |
|  | | | | | | |
| 21 | 0.001 | 9 | 0.01 | 49.48 | 100k | 10.88 |
| 22 | 0.01 | 9 | 0.01 | 100 | 8900 | 1.03 |
| 23 | 0.1 | 9 | 0.01 | 100 | 3100 | 0.40 |
| 24 | 0.2 | 9 | 0.01 | 100 | 2300 | 0.25 |
| 25 | 0.3 | 9 | 0.01 | 100 | 5200 | 0.63 |
|  | | | | | | |
| 26 | 0.001 | 10 | 0.01 | 49.48 | 100k | 10.49 |
| 27 | 0.01 | 10 | 0.01 | 100 | 2600 | 0.34 |
| 28 | **0.1** | **10** | **0.01** | **100** | **2100** | 0.28 |
| 29 | 0.2 | 10 | 0.01 | 100 | 5200 | 0.6 |
| 30 | 0.3 | 10 | 0.01 | 100 | 2000 | 0.20 |

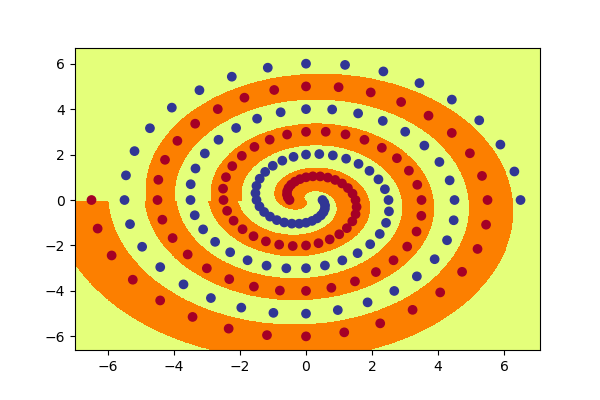
Then I did some experiments for PolarNet with the learning rate and found that using a slightly larger lr of 0.03 (default=0.01) results in faster learning.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#Run** | **init** | **#hidden** | **lr** | **Epochs** | **Time (mins)** |
| 31 | 0.1 | 10 | 0.01 | 3600 | 0.42 |
| 32 | 0.1 | 10 | 0.01 | 7700 | 0.95 |
| 33 | 0.1 | 10 | 0.01 | 2500 | 0.33 |
| 34 | 0.1 | 10 | 0.02 | 3900 | 0.44 |
| 35 | 0.1 | 10 | 0.02 | 1200 | 0.15 |
| 36 | 0.1 | 10 | 0.02 | 1200 | 0.13 |
| 37 | 0.1 | 10 | 0.03 | 1200 | 0.16 |
| 38 | **0.1** | **10** | **0.03** | **1000** | 0.13 |
| 39 | 0.1 | 10 | 0.03 | 1100 | 0.15 |
| 40 | 0.1 | 10 | 0.1 | 1000 | 0.18 |
| 41 | 0.1 | 10 | 0.1 | 3900 | 0.46 |
| 42 | 0.1 | 10 | 0.1 | 3600 | 0.42 |

Red = Worst performing; Green = Best runs, lowest epochs

Increasing the lr by a factor of 0.01 reduces the epochs to ~1000 for init=0.1 and hid=10. PolarNet starts giving 100% accuracy at around 8 hidden layers, below that, it’s dodgy and won’t classify properly. Increasing the layers has a positive effect on learning.

**Polar\_out.png :**



**Question 3.**

See code in spiral.py

**Question 4.**

**RawNet** - The ones highlighted in green are successful runs (100% accuracy)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#Run** | **init** | **#hidden** | **lr** | **Accuracy** | **Epochs** | **Time (mins)** |
| 1 | 0.001 | 5 | 0.01 | 50.52 | 100k | 14.32 |
| 2 | 0.01 | 5 | 0.01 | 50.52 | 100k | 14.38 |
| 3 | 0.1 | 5 | 0.01 | 72.68 | 100k | 14.36 |
| 4 | 0.2 | 5 | 0.01 | 73.20 | 100k | 14.16 |
| 5 | 0.3 | 5 | 0.01 | 68.04 | 100k | 13.97 |
|  | | | | | | |
| 6 | 0.001 | 6 | 0.01 | 50.52 | 100k | 14.52 |
| 14.7 | 0.01 | 6 | 0.01 | 50.52 | 100k | 14.87 |
| 8 | 0.1 | 6 | 0.01 | 70.62 | 100k | 14.83 |
| 9 | 0.2 | 6 | 0.01 | 82.47 | 100k | 14.94 |
| 10 | 0.3 | 6 | 0.01 | 74.74 | 100k | 14.68 |
|  | | | | | | |
| 11 | 0.001 | 7 | 0.01 | 50.52 | 100k | 14.32 |
| 12 | 0.01 | 7 | 0.01 | 50.52 | 100k | 14.25 |
| 13 | 0.1 | 7 | 0.01 | 77.32 | 100k | 15.42 |
| 14 | 0.2 | 7 | 0.01 | 82.99 | 100k | 15.25 |
| 15 | 0.3 | 7 | 0.01 | 88.66 | 100k | 15.37 |
|  | | | | | | |
| 16 | 0.001 | 8 | 0.01 | 50.52 | 100k | 13.96 |
| 17 | 0.01 | 8 | 0.01 | 71.28 | 100k | 15.03 |
| 18 | 0.1 | 8 | 0.01 | 97.94 | 100k | 14.92 |
| 19 | 0.2 | 8 | 0.01 | 97.42 | 100k | 14.85 |
| 20 | 0.3 | 8 | 0.01 | 94.85 | 100k | 14.94 |
|  | | | | | | |
| 21 | 0.001 | 9 | 0.01 | 50.52 | 100k | 15.15 |
| 22 | 0.01 | 9 | 0.01 | 50.52 | 100k | 14.67 |
| 23 | 0.1 | 9 | 0.01 | 53.61 | 100k | 15.02 |
| 24 | 0.2 | 9 | 0.01 | 100 | 10000 | 1.46 |
| 25 | 0.3 | 9 | 0.01 | 100 | 8000 | 1.16 |
|  | | | | | | |
| 26 | 0.001 | 10 | 0.01 | 50.52 | 100k | 15.39 |
| 27 | 0.01 | 10 | 0.01 | 50.52 | 100k | 15.76 |
| 28 | **0.1** | **10** | **0.01** | **100** | **7700** | 1.06 |
| 29 | 0.2 | 10 | 0.01 | 100 | 10100 | 1.11 |
| 30 | 0.3 | 10 | 0.01 | 100 | 17800 | 1.47 |

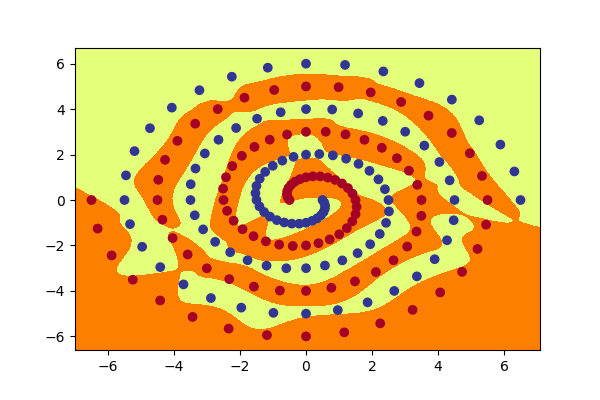
Red = Failed; Green = Successful learning (<20k epochs); Yellow = Best runs, lowest epochs

Just like PolarNet, once I found the init and hidden layer values, for RawNet I tweaked the learning rate and found that using a slightly larger lr of 0.02 (default=0.01) results in faster learning.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#Run** | **init** | **#hidden** | **lr** | **Epochs** | **Time (mins)** |
| 31 | 0.1 | 10 | 0.01 | 14200 | 1.58 |
| 32 | 0.1 | 10 | 0.01 | 12000 | 1.18 |
| 33 | 0.1 | 10 | 0.01 | Failed | 14.73 |
| 34 | **0.1** | **10** | **0.02** | 11900 | 1.12 |
| 35 | 0.1 | 10 | 0.02 | 22200 | 2.49 |
| 36 | 0.1 | 10 | 0.02 | 6900 | 0.82 |
| 36 | 0.1 | 10 | 0.02 | 5900 | 0.63 |
| 36 | 0.1 | 10 | 0.02 | 4500 | 0.48 |
| 36 | 0.1 | 10 | 0.02 | 4300 | 0.47 |
| 36 | 0.1 | 10 | 0.02 | 9100 | 1.03 |
| 36 | 0.1 | 10 | 0.02 | 8200 | 0.73 |
| 37 | 0.1 | 10 | 0.03 | 8500 | 0.98 |
| 38 | 0.1 | 10 | 0.03 | 16400 | 1.85 |
| 39 | 0.1 | 10 | 0.03 | 18200 | 2.04 |
| 40 | 0.1 | 10 | 0.03 | 9100 | 1.06 |
| 41 | 0.1 | 10 | 0.03 | 9000 | 1.26 |
| 42 | 0.1 | 10 | 0.03 | 34200 | 4.25 |
| 43 | 0.1 | 10 | 0.03 | 18800 | 2.13 |
| 44 | 0.1 | 10 | 0.04 | 4900 | 0.59 |
| 45 | 0.1 | 10 | 0.04 | 46400 | 5.84 |
| 46 | 0.1 | 10 | 0.04 | 20100 | 2.29 |
| 47 | 0.1 | 10 | 0.4 | 37700 | 4.27 |
| 48 | 0.1 | 10 | 0.1 | Failed | 11.98 |
| 49 | 0.1 | 10 | 0.1 | Failed | 12.06 |
| 50 | 0.1 | 10 | 0.1 | Failed | 12.08 |

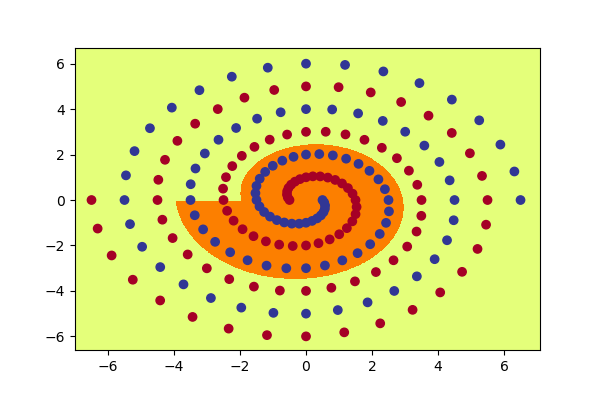
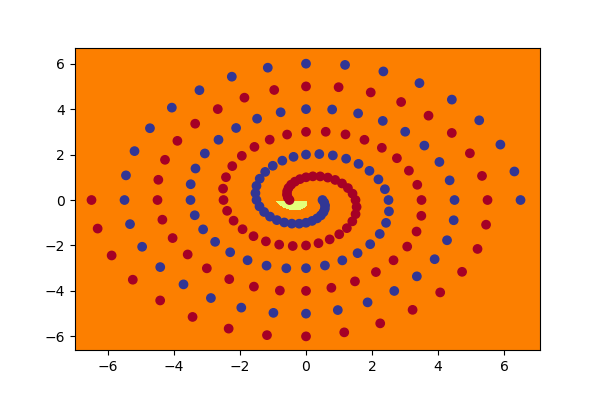
Here we can see that changing the lr to slightly more solves the problem 7 out of 8 times before 20k epochs. Going with hd=10, lr=0.02 and init=0.1.

**Raw\_out.png :**

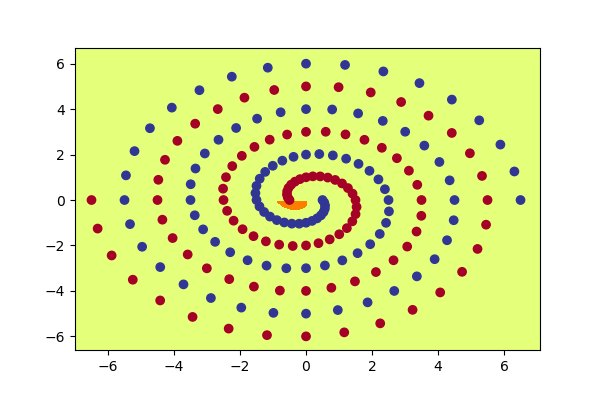
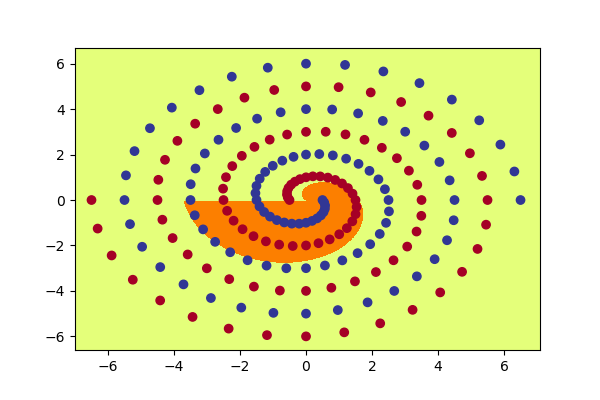


**Question 5.**

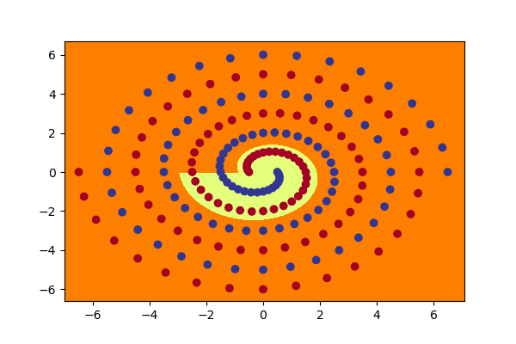
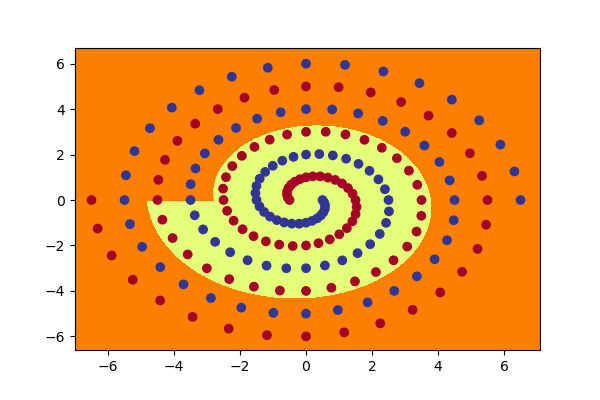
**Graphs for PolarNet:**

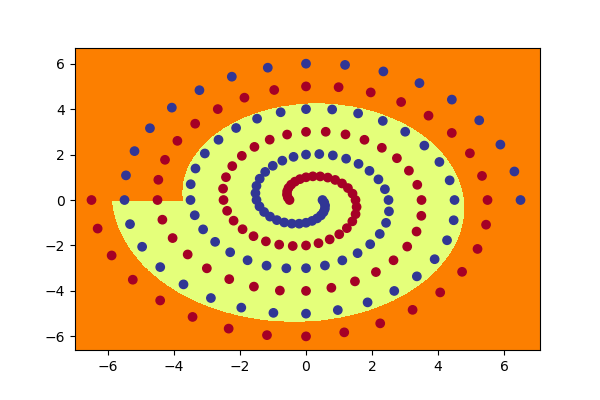
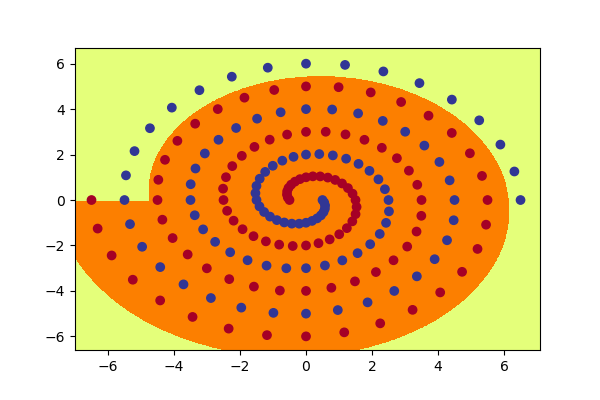
Polar1\_1.png Polar1\_2.png

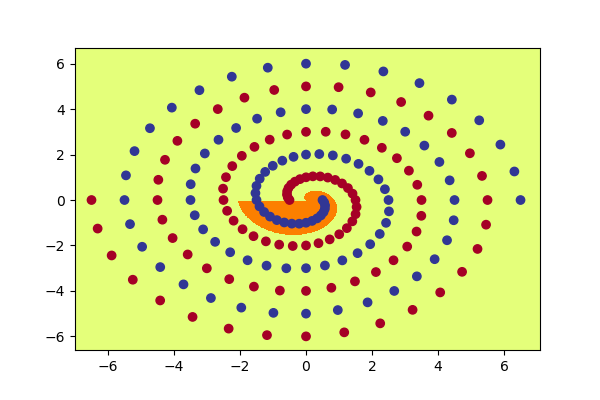
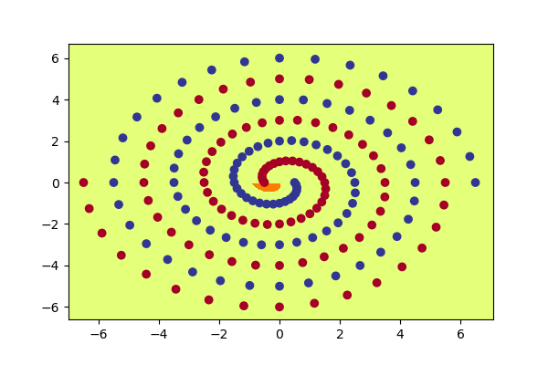
Polar1\_3.png Polar1\_4.png

Polar1\_5.png Polar1\_6.png



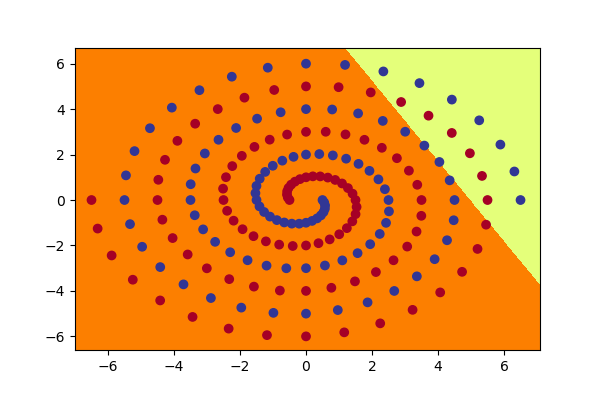
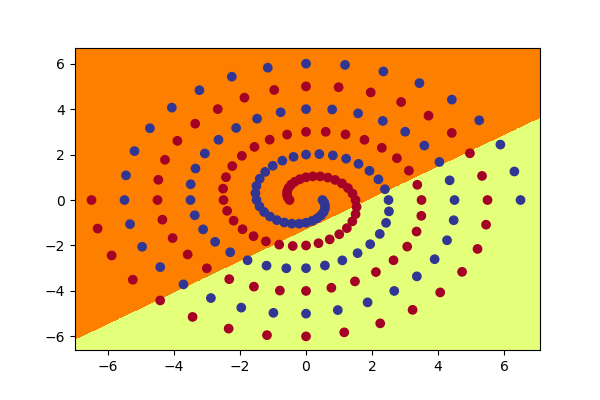
Polar1\_7.png Polar1\_8.png



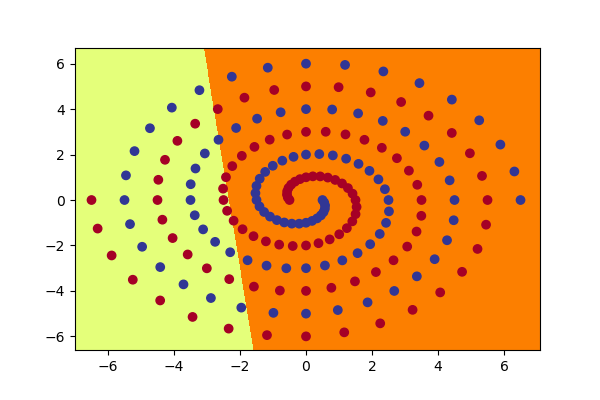
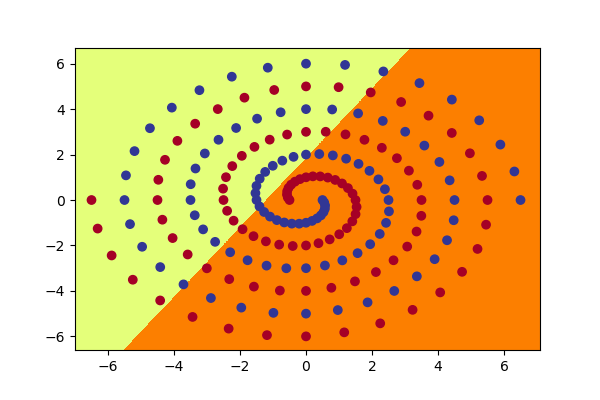
Polar1\_9.png Polar1\_10.png

**Graphs for RawNet:**

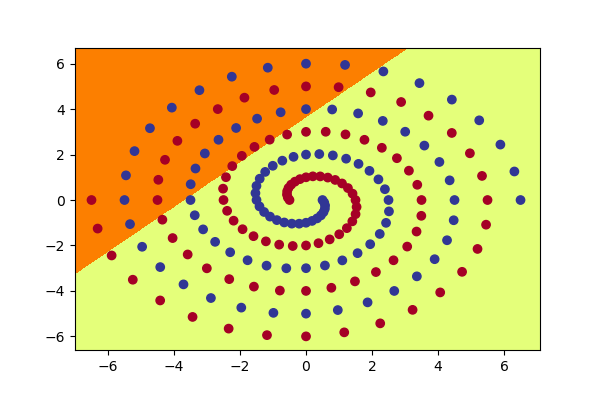
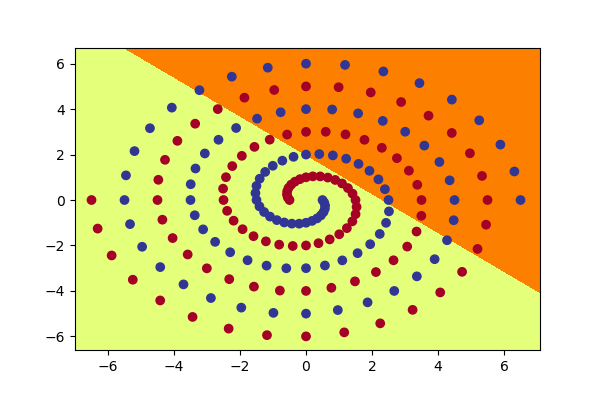
**RawNet Layer 1:**



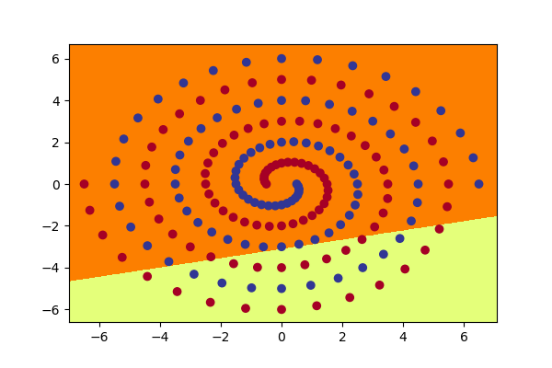
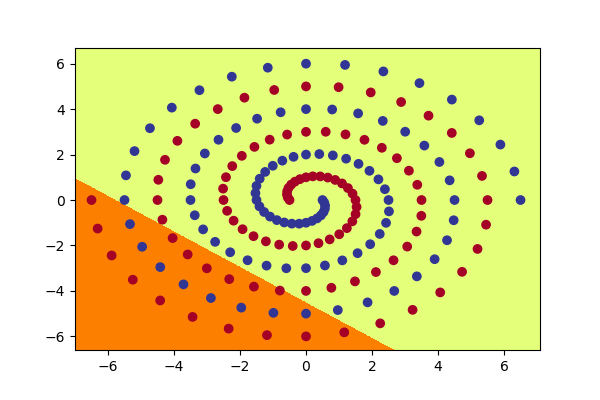
Raw1\_1.png Raw1\_2.png



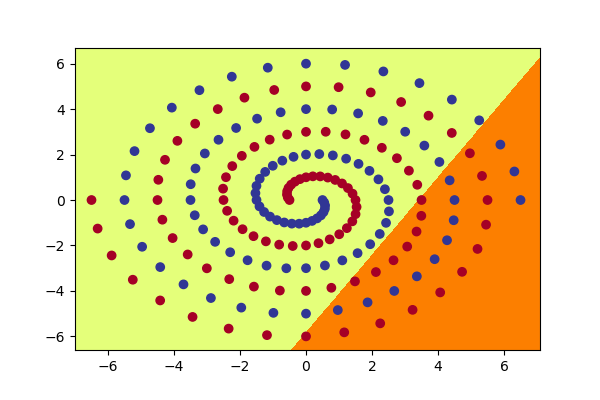
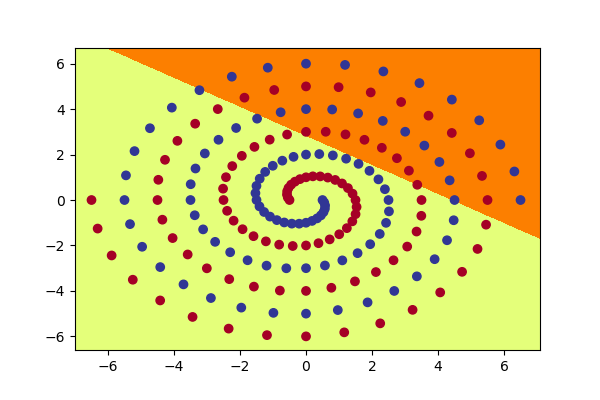
Raw1\_3.png Raw1\_4.png



Raw1\_5.png Raw1\_6.png

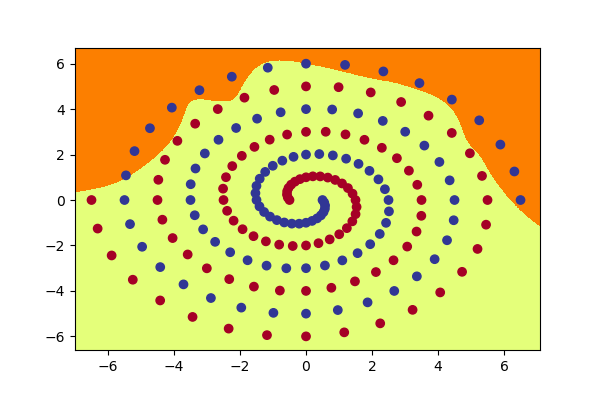
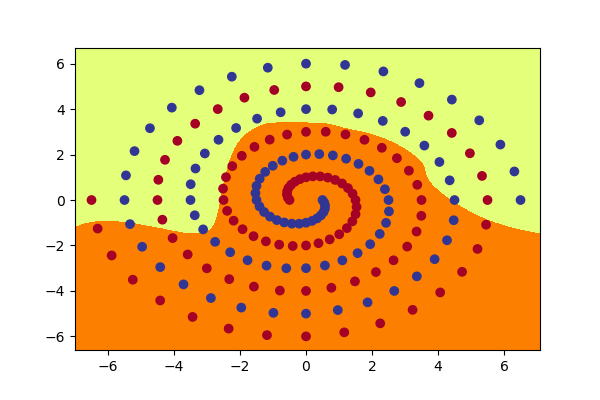
 

Raw1\_7.png Raw1\_8.png

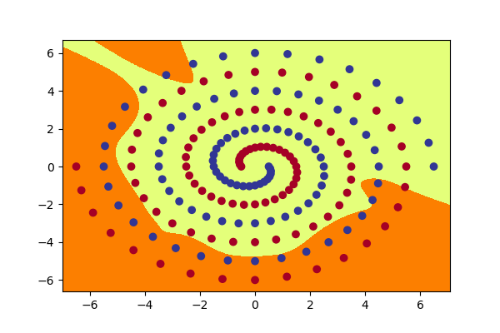
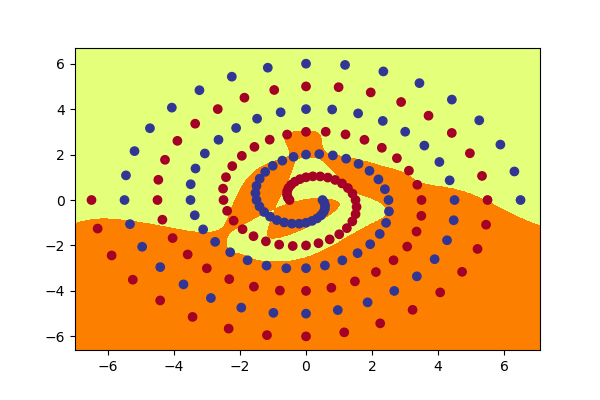


Raw1\_9.png Raw1\_10.png

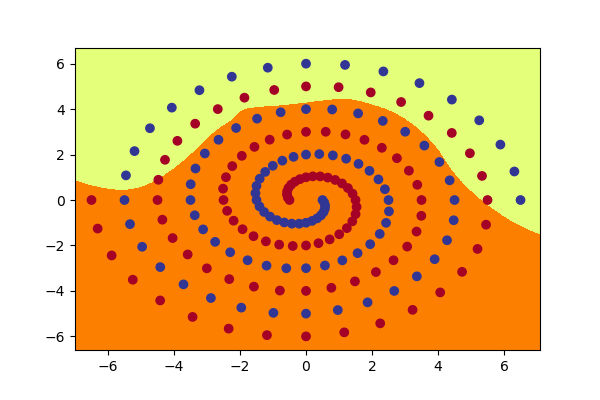
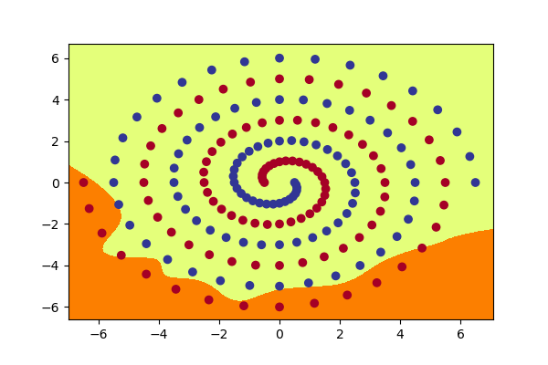
**RawNet Layer 2:**



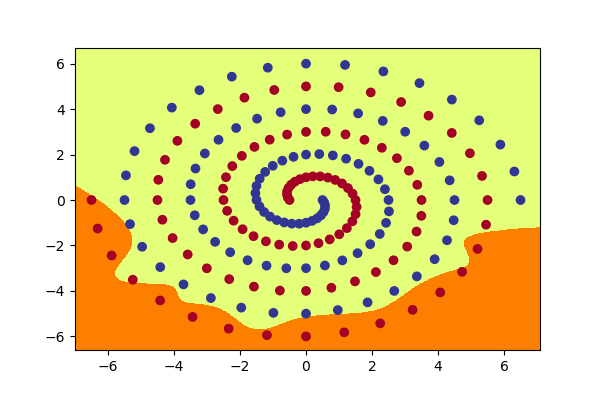
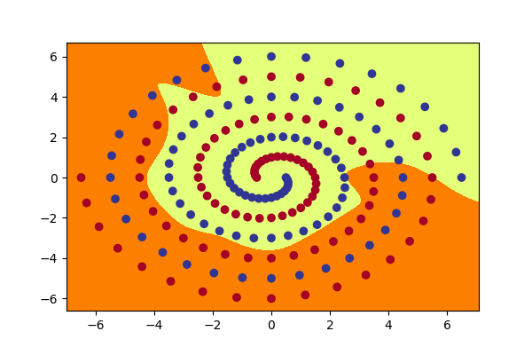
Raw2\_1.png Raw2\_2.png



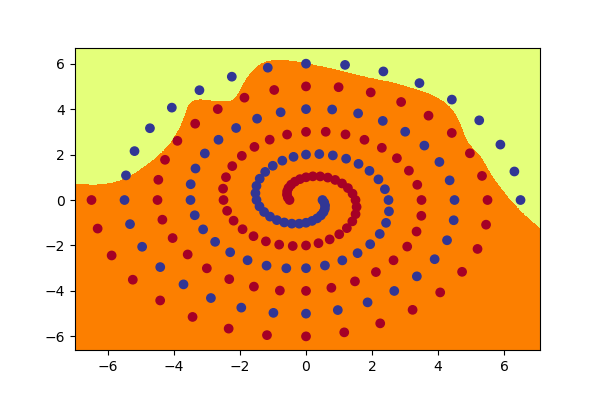
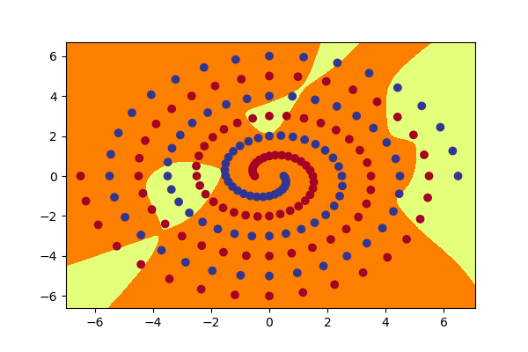
Raw2\_3.png Raw2\_4.png



Raw2\_5.png Raw2\_6.png



Raw2\_7.png Raw2\_8.png



Raw2\_9.png Raw2\_10.png

**Question 6a.**

In case of PolarNet, the function of each hidden node is non-linear. In RawNet, first hidden layer of

nodes produce a linear function decision boundary and the second hidden layer’s nodes produce a non-linear decision boundary. For both networks, regardless of the kind of function hidden nodes learn, every hidden node can only **learn a part of correct decision boundary**. Later, a weighted sum then applied to make the network learn the complete classification boundary.

**Question 6b.**

I tried different values of initial weights in [0.001, 0.01, 0.1, 0.2, 0.3] for RawNet as seen in table on page 8 and 9 and found the most suited value for 100% accuracy was 0.1 and 0.2.

A value too small (0.01) and too large (0.3) value could sometimes cause failure, and value 0.001 was never a success for any number of hidden layers in the range 5-10. As for the speed of learning, it is **not** always true that the larger initial weight is, the faster (less epochs) it can learn. For example, when initial weight is 0.1, RawNet takes 7700 epochs to coverage to 100% but when initial weight is 0.2, it takes 17000 epochs to converge (refer Table on page 8).

**Question 6c.**

**1.** Keeping other parameters unchanged (lr=0.03, init=0.1, hid=10), I changed batch size value from 97 to 194 to 388 . Ran multiple tests with PolarNet and found the epochs needed reduced a lot from 4000-8000 to 500-600. When I double it to 388, it reached 100% accuracy in roughly 500 epochs on all runs, which is almost the same as batch size 194. I even tried larger value (485) and found

it still needs 500 epochs to converge. In reality, for this data, we don’t really need a batch size larger than 194, maximum 388.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#Run** | **Batchsize** | **init** | **#hidden** | **lr** | **Epochs** | **Time (mins)** |
| 1 | 97 **(default)** | 0.1 | 10 | 0.03 | 800 | 1.58 |
| 2 | 194 | 0.1 | 10 | 0.03 | 500 | 0.03 |
| 3 | 194 | 0.1 | 10 | 0.03 | 600 | 0.03 |
| 4 | 194 | 0.1 | 10 | 0.03 | 600 | 0.03 |
| 5 | 388 | 0.1 | 10 | 0.03 | 500 | 0.4 |
| 6 | **388** | **0.1** | **10** | **0.03** | 500 | 0.03 |

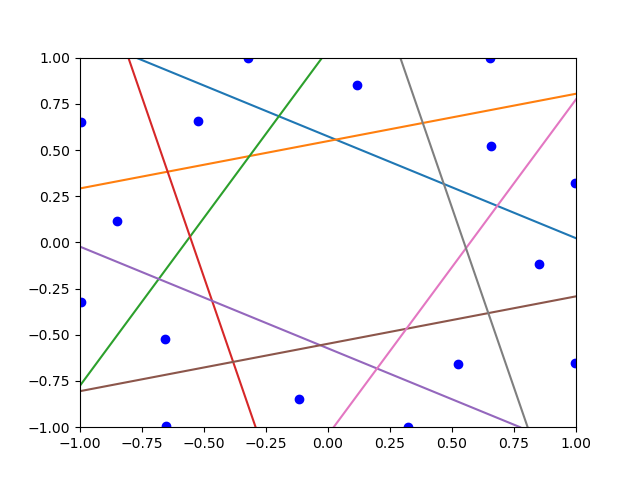
**2.** Another thing I tried was using SGD optimizer instead of Adam while keeping all other parameters unchanged. On running RawNet with initial weight 0.1 for multiple runs I found it cannot converge to 100% within 20000 epochs. When using SGD, I also noticed that the accuracy does increase but it increases rather slowly than using Adam. In fact, Adam can be viewed as a combination of RMSprop and SGD with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

**3.** I changed tanh activation function to RelU and keep other parameters unchanged. Again, on testing with RawNet with initial weight 0.1 on multiple runs I deduced that when using RelU activation function, the accuracy cannot converge to 100% within 20000 epochs and it fluctuates between 80% and 85%. This maybe because RelU function will make some neurons output as 0 and these neurons can never be activated by further layers. In this problem, it is a training process, RelU function may cause under-fitting. So, it’s probably not a good idea to use RelU in the network.

**Part 3: Hidden Unit Dynamics**

**Question 1.**

Presented here is the image formed by running encoder\_main.py passing --target=star16



The parameters for the shape are:

num\_in = 16

num\_out = 8

input = torch.eye(num\_in) makes a 16x16 diagonal matrix of 1's

net = EncModel(num\_in,2,num\_out).to(device) creates neural network of 16-2-8 encoder.

**Question 2.**

These figures are captured at epochs - 50, 100, 150, 200, 300, 500, 700, 1000, 1500, 2000, 3000 with the final image produced at 15655 epochs. Time taken is roughly 30 secs.

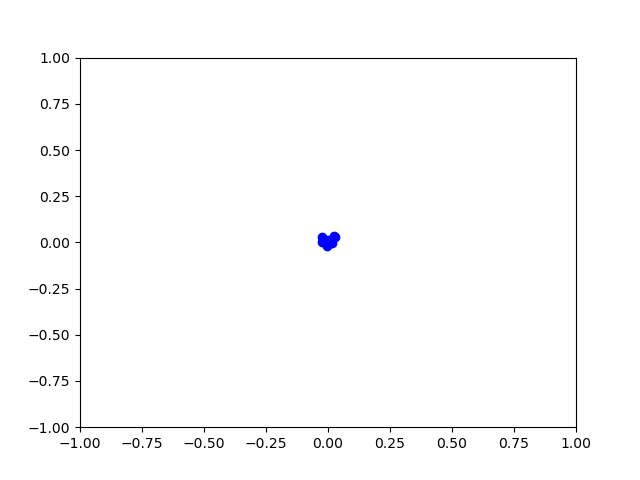
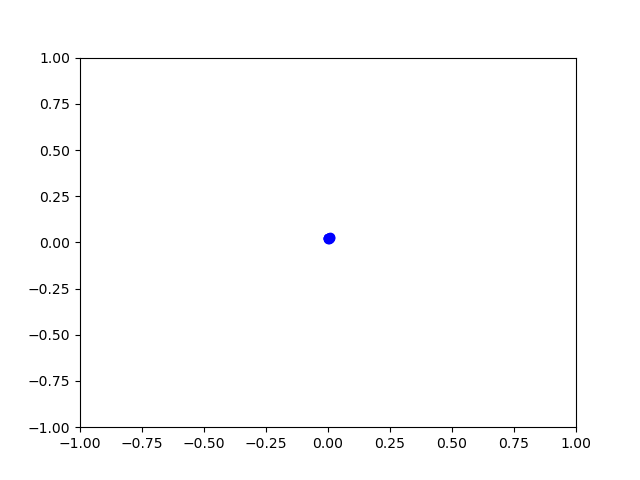


Figure 1 Figure 2

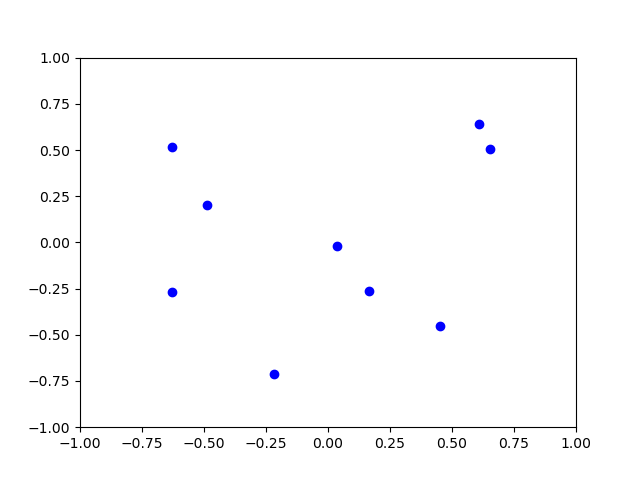
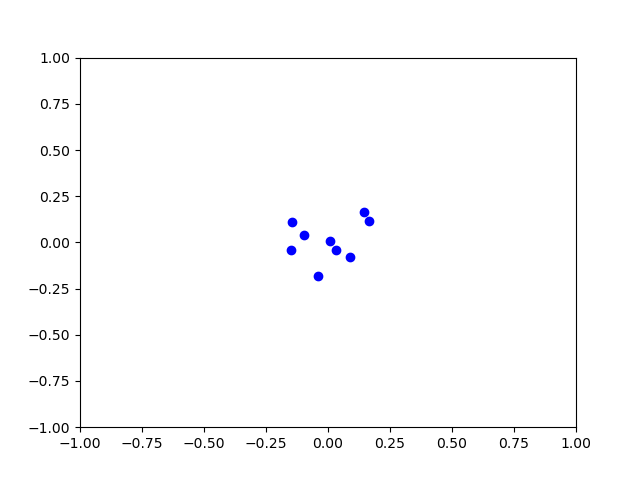


Figure 3 Figure 4

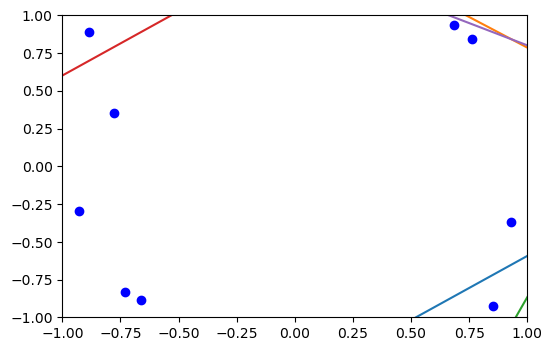
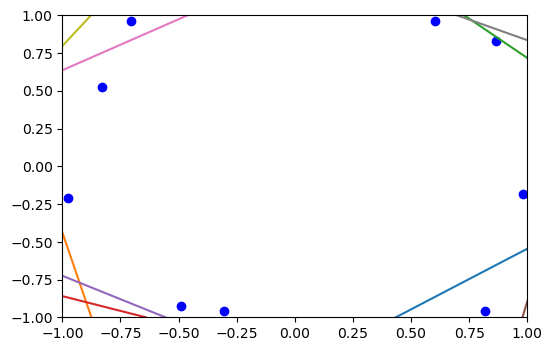
 

Figure 5 Figure 6

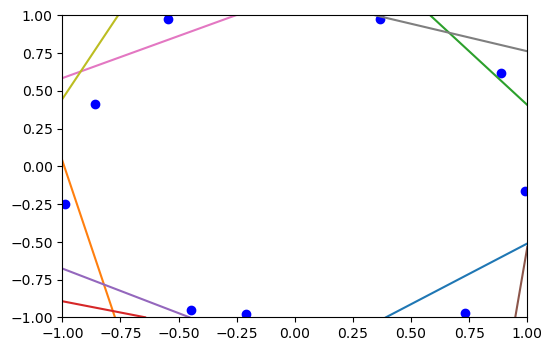
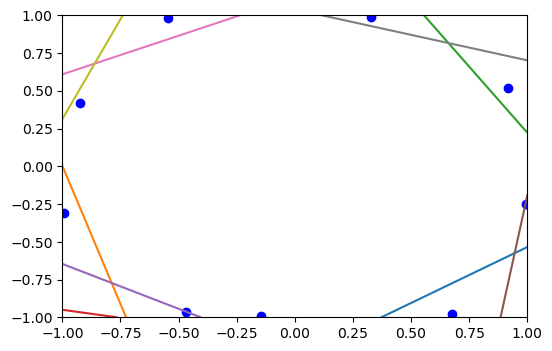
 

Figure 7 Figure 8

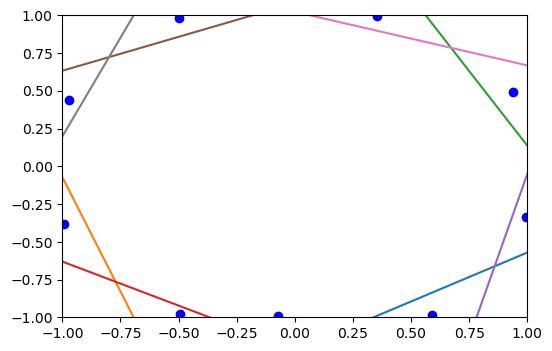
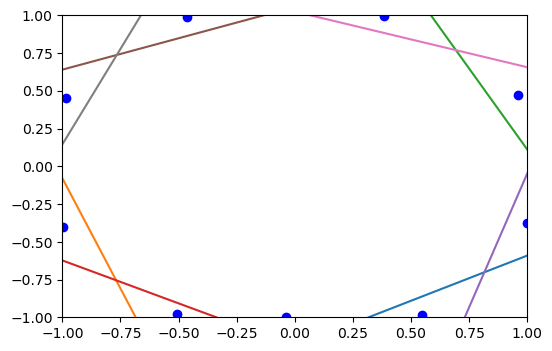
 

Figure 9 Figure 10

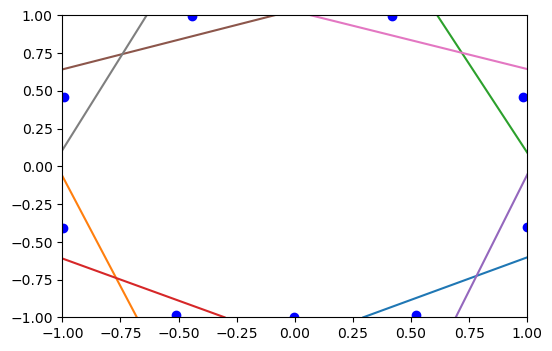
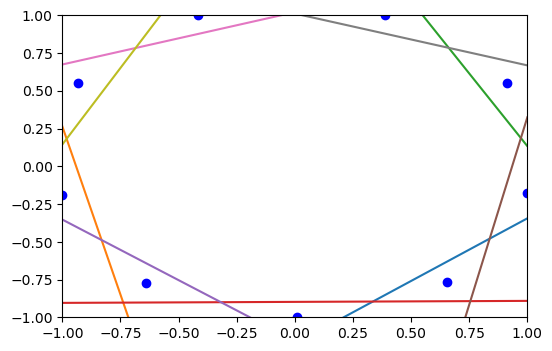
 

Figure 11 Final figure

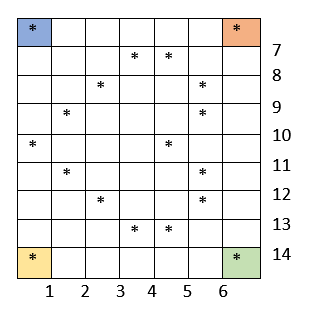
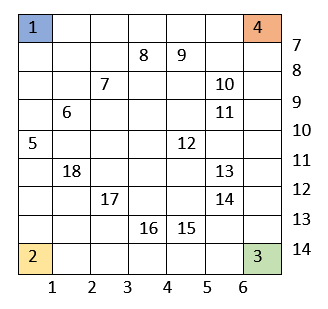
This 9-2-9 is simple form of what are called auto-encoder networks. This helps us understand the geometry of NN in fairly simple setting. We’ve got 9 inputs and 9 outputs with 2 nodes in the middle. Total weights are 9x2 + 9x2 + 2 = 38 weights. Since the activation is tanh, the graph is [-1,1].

Hidden to output weights are drawn as a line, that’s why the final image has 9 boundary lines. The network **slowly tries to achieve maximum separation.** Each line divides one point(input). The points are as far away from each other as possible.

In the final image we can see the network has successful learned this task, all points are separated from each other by a line.

**Question 3.**

Approached the problem by making a matrix first and the assigning each dot a numerical value.

**** 

Then, decided on 2 things:

1. Dots on the left of a vertical line is 0, right is 1
2. Dots on the top of a horizontal line is 0, below is 1

Then made the following tensor (matrix) of number of separating lines as columns and dots as rows.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 6 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 13 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 14 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 16 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 17 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 18 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

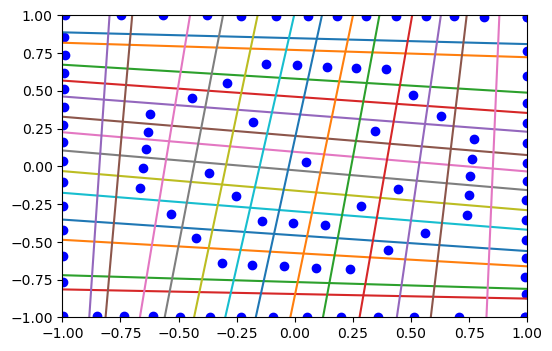
**Question 4.**

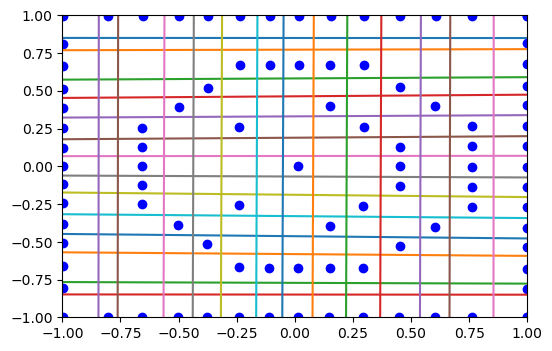
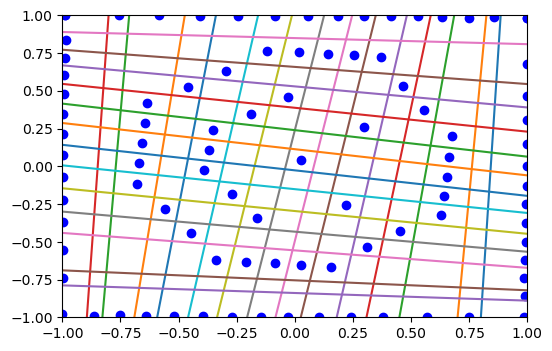
For my own figures, I tried a lot of shapes that would perform well and make a sensible shape for all almost all runs.

I’m going with a ‘**emoji-smile**’ and a ‘**boat**’.

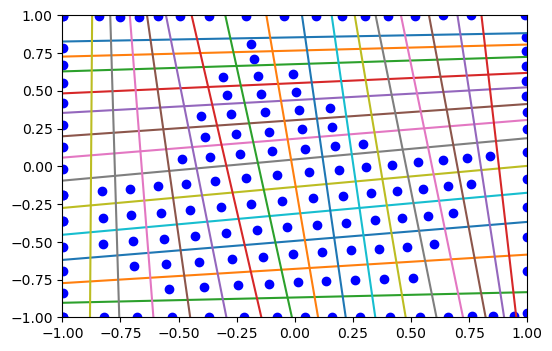
**I made a small script that I’ve added at the end of encoder.py file that generates a tensor for a given shape of 0’s and 1’s in a matrix form.**

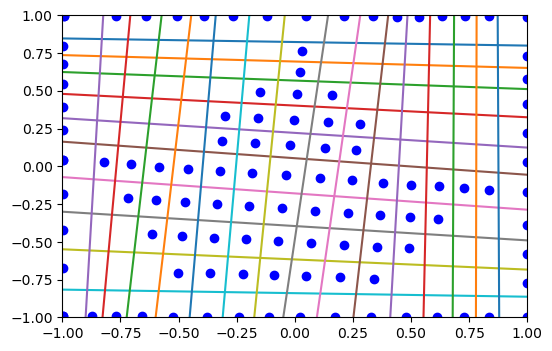
**Target1 – Smiling emoji**





**Target2 – Boat / Ship**

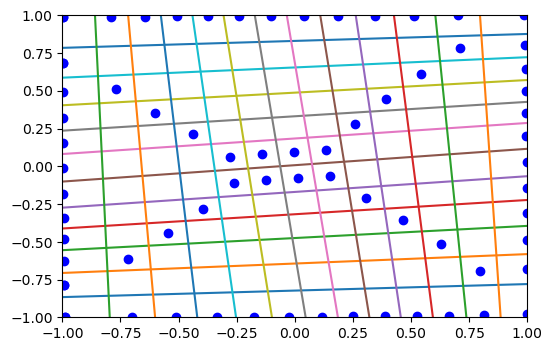




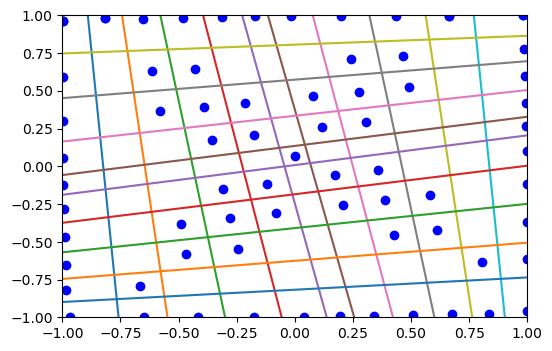
(Tried 2 different size boats (tensors) to make it clear that it’s a boat, encoder has the bottom boat tensor)

I also tried other shapes, such as :

1. **Envelope**



1. **Fan / Clover leaf**



**Note:** These tensors are quite large and take a lot of time to run

Set **lr=1.2** in encoder\_main for quick execution

**References**

1. Understanding the layers in-our features: <https://towardsdatascience.com/convolutional-neural-network-for-image-classification-with-implementation-on-python-using-pytorch-7b88342c9ca9>
2. Selecting parameters for a NN: <https://stackoverflow.com/questions/56660546/how-to-select-parameters-for-nn-linear-layer-while-training-a-cnn>