ASSIGNMENT 1

COMP9444

Saurabh Jain – z5204308

Part 1

Q1.

```
[[769.   5.  10.  10.  31.  61.   2.  63.  29.  20.]
 [  7. 670. 111.  18.  28.  23.  56.  14.  25.  48.]
 [  7.  58. 696.  26.  25.  19.  45.  38.  46.  40.]
 [  4.  35.  63. 752.  14.  60.  14.  18.  27.  13.]
 [ 59.  52.  79.  21. 623.  21.  33.  35.  20.  57.]
 [  8.  29. 125.  17.  19. 727.  27.   7.  33.   8.]
 [  5.  24. 150.  10.  26.  24. 720.  20.   9.  12.]
 [ 17.  29.  28.  11.  84.  15.  54. 623.  90.  49.]
 [  9.  37.  98.  40.   9.  29.  43.   7. 706.  22.]
 [  9.  51.  91.   2.  49.  32.  19.  31.  38. 678.]]

Test set: Average loss: 1.0098, Accuracy: 6964/10000 (70%)
```

Total Independent Parameters = (28 x 28 + 1) x 10 = 7850

Q2.

With 10 hidden nodes:

```
[[756.   4.   5.  21.  24.  56.   2.  64.  53.  15.]
 [  4. 647. 143.  19.  23.  13.  66.   2.  30.  53.]
 [  6.  36. 755.  41.  21.  16.  40.  33.  35.  17.]
 [  9.  26.  81. 761.  10.  51.   4.  20.  23.  15.]
 [ 60.  41.  90.  25. 609.  23.  37.  20.  37.  58.]
 [  5.  25. 166.  15.  13. 699.  33.   9.  26.   9.]
 [  4.  23. 188.  15.  34.  20. 677.  10.  18.  11.]
 [ 33.  30.  91.  17. 123.  29.  33. 516.  95.  33.]
 [  8.  27.  82.  39.   8.  29.  38.   4. 754.  11.]
 [  3.  33. 141.   1.  50.  29.   8.  11.  33. 691.]]

Test set: Average loss: 1.0435, Accuracy: 6865/10000 (69%)
```

With 100 hidden nodes

```
<class 'numpy.ndarray'>
[[846.   3.   1.   5.  30.  35.   2.  40.  31.   7.]
 [  3. 804.  25.   6.  21.  18.  65.   4.  26.  28.]
 [  7.  20. 820.  44.   9.  17.  34.  12.  19.  18.]
 [  3.  11.  25. 918.   4.  13.   3.   5.   9.   9.]
 [ 41.  33.  13.   8. 807.  11.  28.  21.  19.  19.]
 [  6.  11.  69.  10.  11. 843.  26.   4.  13.   7.]
 [  4.  10.  58.   8.  20.   6. 878.   9.   2.   5.]
 [ 19.  11.  15.   5.  27.   8.  41. 807.  34.  33.]
 [ 10.  28.  21.  51.   6.   9.  33.   5. 829.   8.]
 [  3.  19.  44.   6.  23.   9.  21.  15.  16. 844.]]

Test set: Average loss: 0.5315, Accuracy: 8396/10000 (84%)
```

With 110 hidden nodes

```
[[863.   4.   2.   5.  22.  29.   2.  40.  30.   3.]
 [  5. 814.  35.   4.  17.  10.  60.   6.  23.  26.]
 [  9.  11. 845.  38.  10.  15.  23.   9.  28.  12.]
 [  4.  12.  43. 899.   1.  15.   3.   4.   8.  11.]
 [ 51.  31.  22.   8. 800.   8.  24.  18.  18.  20.]
 [  9.  12.  97.  14.  10. 811.  23.   2.  16.   6.]
 [  3.  18.  68.   8.  10.   7. 869.   5.   1.  11.]
 [ 22.  14.  27.   7.  24.   9.  34. 801.  34.  28.]
 [ 13.  30.  31.  44.   2.  12.  33.   3. 828.   4.]
 [  5.  21.  59.   4.  29.   9.  25.  15.  15. 818.]]

Test set: Average loss: 0.5316, Accuracy: 8348/10000 (83%)
```

With 150 hidden nodes

```
[[863.   3.   2.   6.  26.  28.   3.  35.  29.   5.]
 [  8. 818.  29.   4.  20.  10.  59.   5.  22.  25.]
 [  8.  12. 832.  48.  14.  17.  29.   9.  19.  12.]
 [  4.   8.  29. 917.   2.  14.   6.   3.   8.   9.]
 [ 40.  33.  21.  10. 812.   7.  30.  14.  19.  14.]
 [  8.  16.  70.  10.  10. 828.  30.   2.  18.   8.]
 [  3.   9.  40.   9.  18.   4. 903.   6.   2.   6.]
 [ 21.  13.  18.   3.  24.   8.  37. 825.  24.  27.]
 [ 11.  23.  22.  40.   4.   8.  30.   5. 849.   8.]
 [  5.  18.  48.   4.  34.   5.  28.  11.  13. 834.]]

Test set: Average loss: 0.5018, Accuracy: 8481/10000 (85%)
```

With a minimum of 150 hidden nodes, an accuracy of minimum 84% can be achieved

Total Number of Independent Parameters: [(28 x 28 + 1) x 150 + (150 + 1) x 10] = 117750 + 1510 = 119260

Q3.

```
[[970.   4.   0.   0.  17.   1.   0.   5.   1.   2.]
 [  0. 940.   4.   0.   9.   1.  35.   2.   3.   6.]
 [ 10.   7. 904.  20.   7.  16.  22.   4.   5.   5.]
 [  1.   1.  12. 961.   3.   5.  10.   2.   2.   3.]
 [ 15.   9.   2.   5. 944.   0.  11.   5.   8.   1.]
 [  5.  15.  31.   3.   4. 900.  22.   1.   6.  13.]
 [  2.   4.   6.   1.   4.   3. 976.   2.   0.   2.]
 [  4.   2.   5.   0.   4.   0.   5. 959.   4.  17.]
 [  3.  10.   3.   0.  12.   1.   6.   1. 963.   1.]
 [  3.   2.   4.   0.   9.   0.   4.   4.   5. 969.]]

Test set: Average loss: 0.2242, Accuracy: 9486/10000 (95%)
```

Total Number of Independent Parameters: [(4 x 4 x 1 +1) x 16 + (4 x 4 x 16 + 1) x 32 + (32 x 8 x 8 + 1) x 600 + (600 + 1) x 10] = 1243906

Q4.

a. The model is Q1, Q2 and Q3 have an accuracy of 70%, 85% and 955 respectively. The respective accuracy of the models can be observed to be dependent on the number of hidden levels and the kind of activation function which is used at each layer.

b.

- Model in Q1 = 7850
- Model in Q2 = 119260
- Model in Q3 = 1243906

c. From the three models below,

Model 1:

```
[[769.   5.  10.  10.  31.  61.   2.  63.  29.  20.]
 [  7. 670. 111.  18.  28.  23.  56.  14.  25.  48.]
 [  7.  58. 696.  26.  25.  19.  45.  38.  46.  40.]
 [  4.  35.  63. 752.  14.  60.  14.  18.  27.  13.]
 [ 59.  52.  79.  21. 623.  21.  33.  35.  20.  57.]
 [  8.  29. 125.  17.  19. 727.  27.   7.  33.   8.]
 [  5.  24. 150.  10.  26.  24. 720.  20.   9.  12.]
 [ 17.  29.  28.  11.  84.  15.  54. 623.  90.  49.]
 [  9.  37.  98.  40.   9.  29.  43.   7. 706.  22.]
 [  9.  51.  91.   2.  49.  32.  19.  31.  38. 678.]]

Test set: Average loss: 1.0098, Accuracy: 6964/10000 (70%)
```

Model 2:

```
[[863.   3.   2.   6.  26.  28.   3.  35.  29.   5.]
 [  8. 818.  29.   4.  20.  10.  59.   5.  22.  25.]
 [  8.  12. 832.  48.  14.  17.  29.   9.  19.  12.]
 [  4.   8.  29. 917.   2.  14.   6.   3.   8.   9.]
 [ 40.  33.  21.  10. 812.   7.  30.  14.  19.  14.]
 [  8.  16.  70.  10.  10. 828.  30.   2.  18.   8.]
 [  3.   9.  40.   9.  18.   4. 903.   6.   2.   6.]
 [ 21.  13.  18.   3.  24.   8.  37. 825.  24.  27.]
 [ 11.  23.  22.  40.   4.   8.  30.   5. 849.   8.]
 [  5.  18.  48.   4.  34.   5.  28.  11.  13. 834.]]

Test set: Average loss: 0.5018, Accuracy: 8481/10000 (85%)
```

Model 3:

```
[[970.   4.   0.   0.  17.   1.   0.   5.   1.   2.]
 [  0. 940.   4.   0.   9.   1.  35.   2.   3.   6.]
 [ 10.   7. 904.  20.   7.  16.  22.   4.   5.   5.]
 [  1.   1.  12. 961.   3.   5.  10.   2.   2.   3.]
 [ 15.   9.   2.   5. 944.   0.  11.   5.   8.   1.]
 [  5.  15.  31.   3.   4. 900.  22.   1.   6.  13.]
 [  2.   4.   6.   1.   4.   3. 976.   2.   0.   2.]
 [  4.   2.   5.   0.   4.   0.   5. 959.   4.  17.]
 [  3.  10.   3.   0.  12.   1.   6.   1. 963.   1.]
 [  3.   2.   4.   0.   9.   0.   4.   4.   5. 969.]]

Test set: Average loss: 0.2242, Accuracy: 9486/10000 (95%)
```

It can be seen that many incorrect characters are recognized in Model 1 and then gets better but still bad in Model 2. Model 3 has a good accuracy but still misidentifies the characters. So, all characters that are mis-identified in Model 3 are misidentified in model 2 and 1. And all misidentifications in Model 2 are also misidentified in model 1.

Going over what's common in all, examples for misidentifications are 1 is misidentified as 6, 4 is misidentified as 0. Looking at the table below it can be seen that the character for 0 & 4, and 1 & 6 has similar characteristics.

Hiragana	Unicode	Samples	Sample Images
お (o)	U+304A	7000	
き (ki)	U+304D	7000	
す (su)	U+3059	7000	
つ (tsu)	U+3064	7000	
な (na)	U+306A	7000	

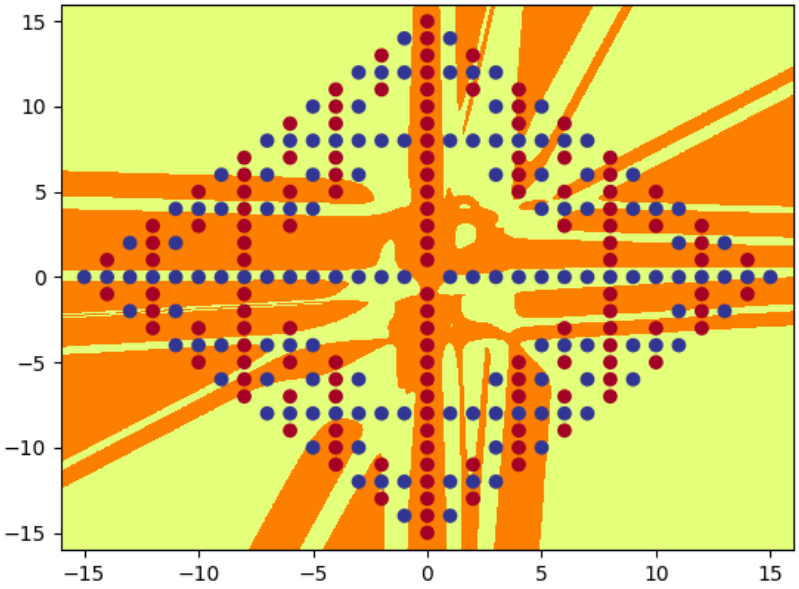
Hiragana	Unicode	Samples	Sample Images
は (ha)	U+306F	7000	
ま (ma)	U+307E	7000	
や (ya)	U+3084	7000	
れ (re)	U+308C	7000	
を (wo)	U+3092	7000	

Part 2

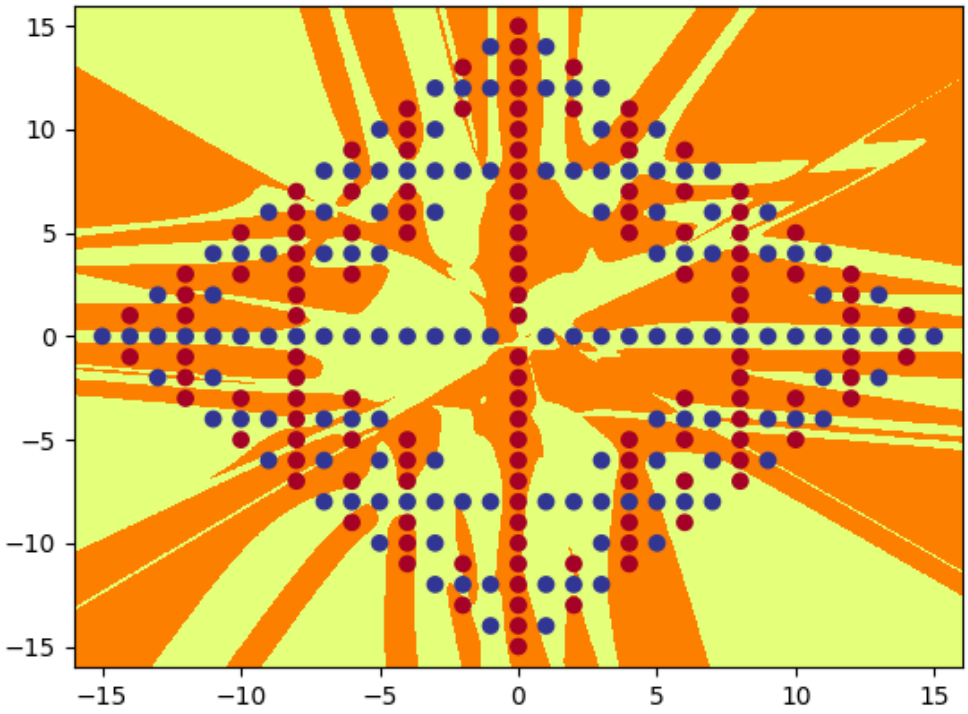
Question 2

After trying 100s of permutation of initial weights and lr value, I decided to use an init value of 0.575 and lr value of 0.005. All hids ran with an epoch of 20000 due to CPU limitations

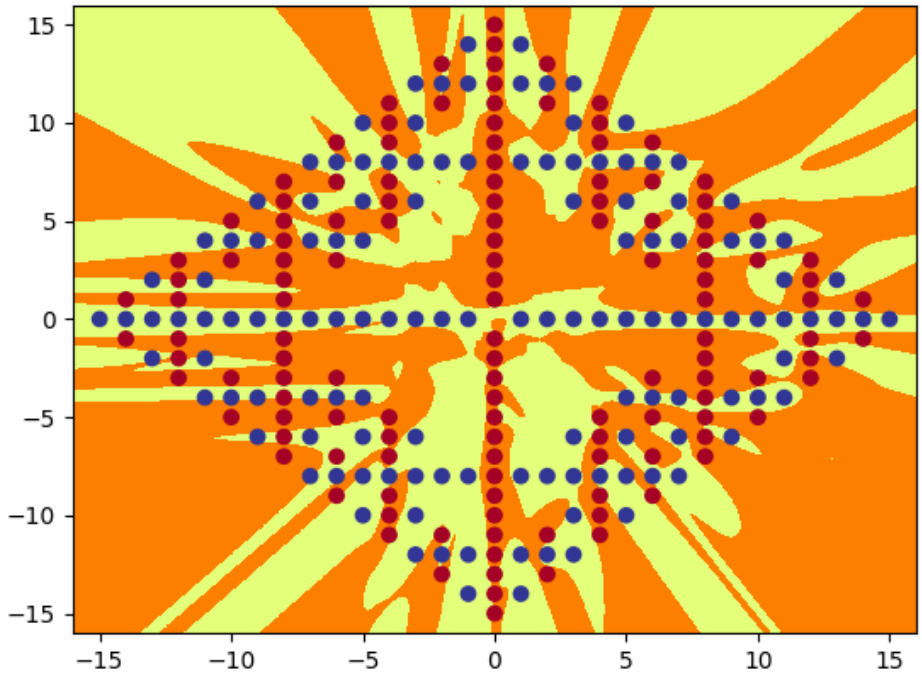
Starting of with the default hid of 10, the model ended up with an accuracy of 84.23% with a loss of 0.2461. The output looks acceptable, with a few cases where it'd get incorrect output.



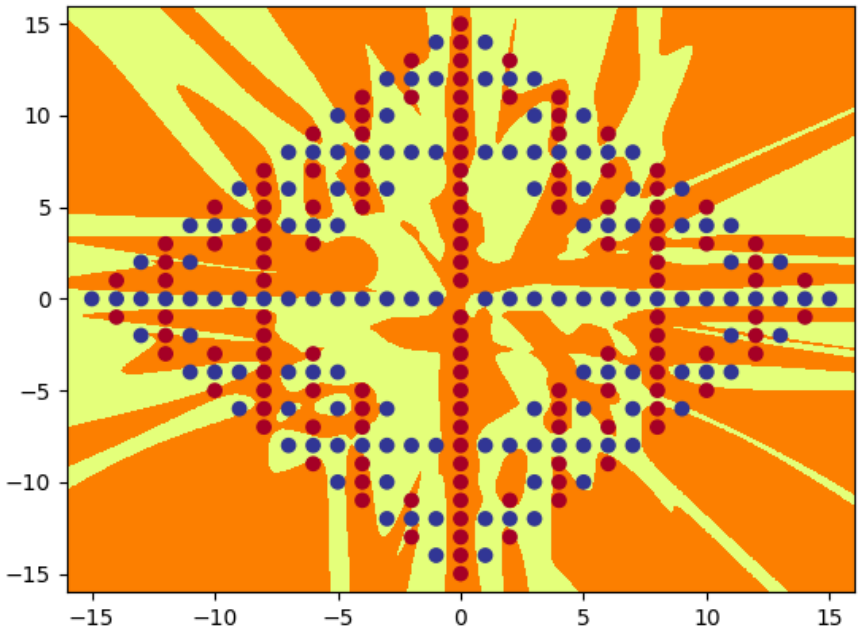
Hid of 15 and got acc of 96.15 and loss of 0.0710. There is a noticeable difference in the plot of hid 10 and 15.



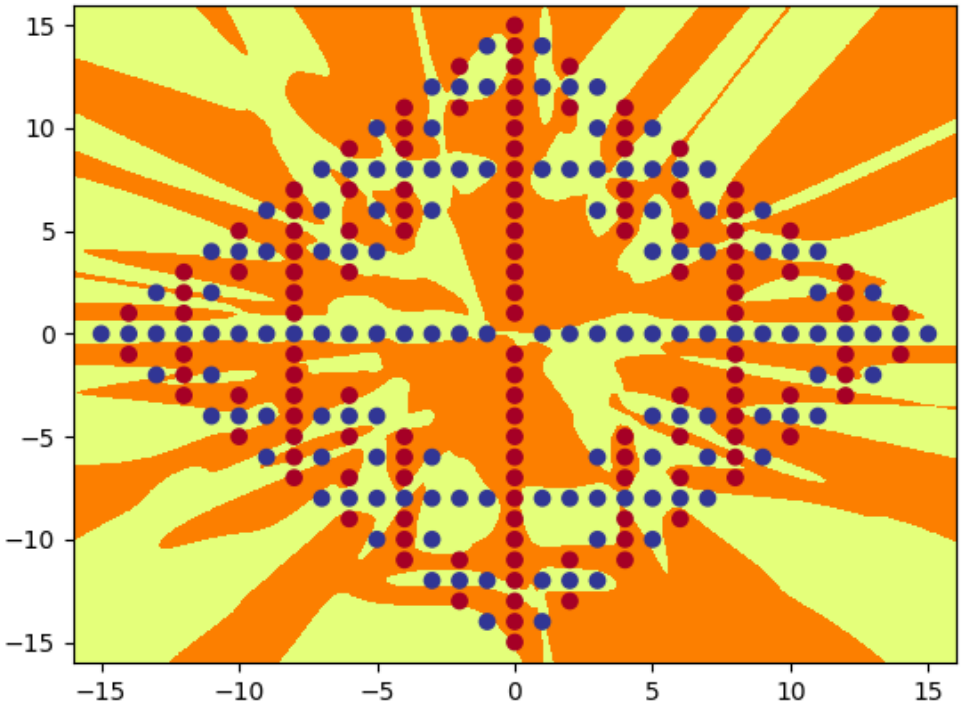
Hid of 20 gets diveged towards an acc of 98.08% and loss of 0.0421. Here barely any dot is inccorectlt detected.



Hid of 23 get a acc of 99.62 and a loss of 0.0096. The plot here creats a good pattern which can distictly locate the correct output.



A minimum of 28 hidden nodes wiill be required to acchieve high accuracy 100% and a loss of 0.0096

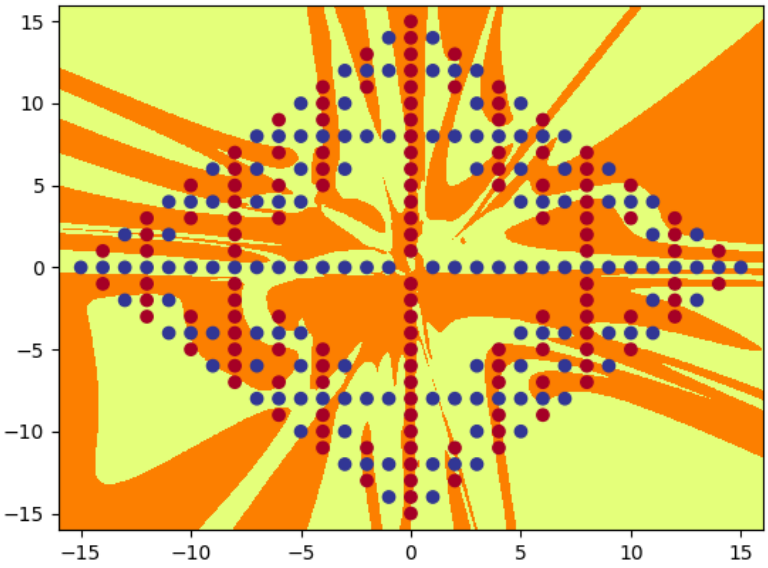


The total number of indpendent parameters = [(2+1)x28+(28+1)x28+(28+1)x1 = 925

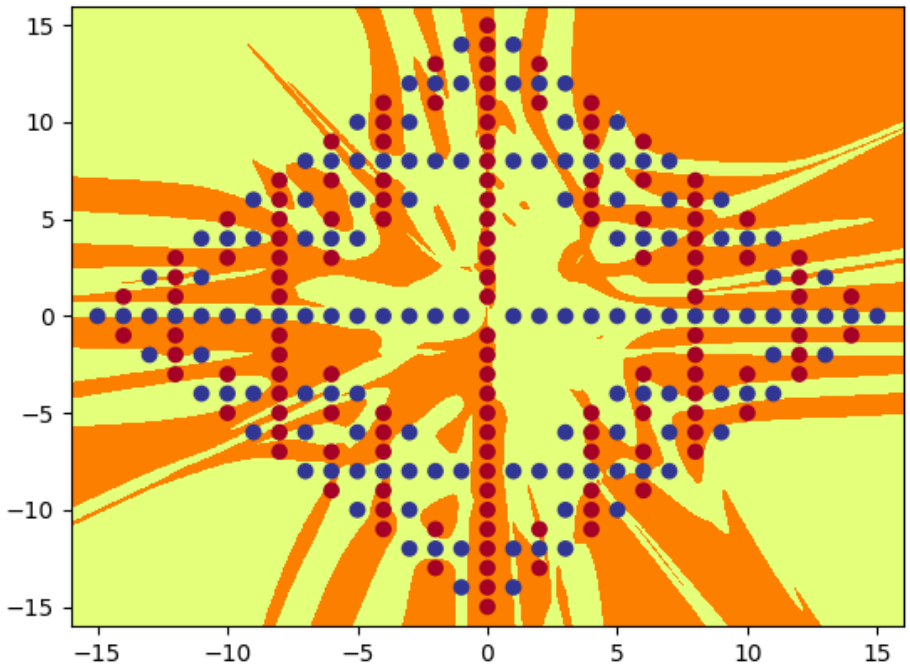
Question 4

All tests ran using an init value of 0.575 and lr value of 0.005. All hids ran with an epoch of 20000 due to CPU limitations

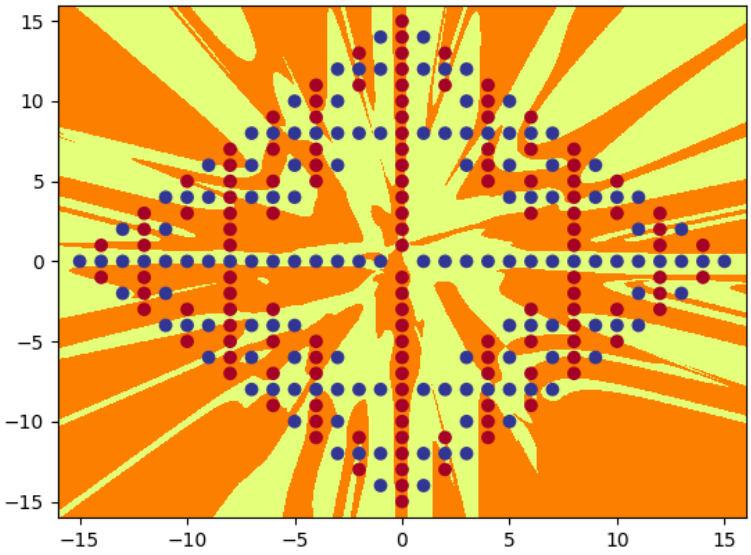
Hid of 10 produced an acc of 92.31% with a loss of 0.1392



Hid of 17 is the minimum required to get a high accuracy for the model. It gets an acc of 99.62% and loss of 0.0116



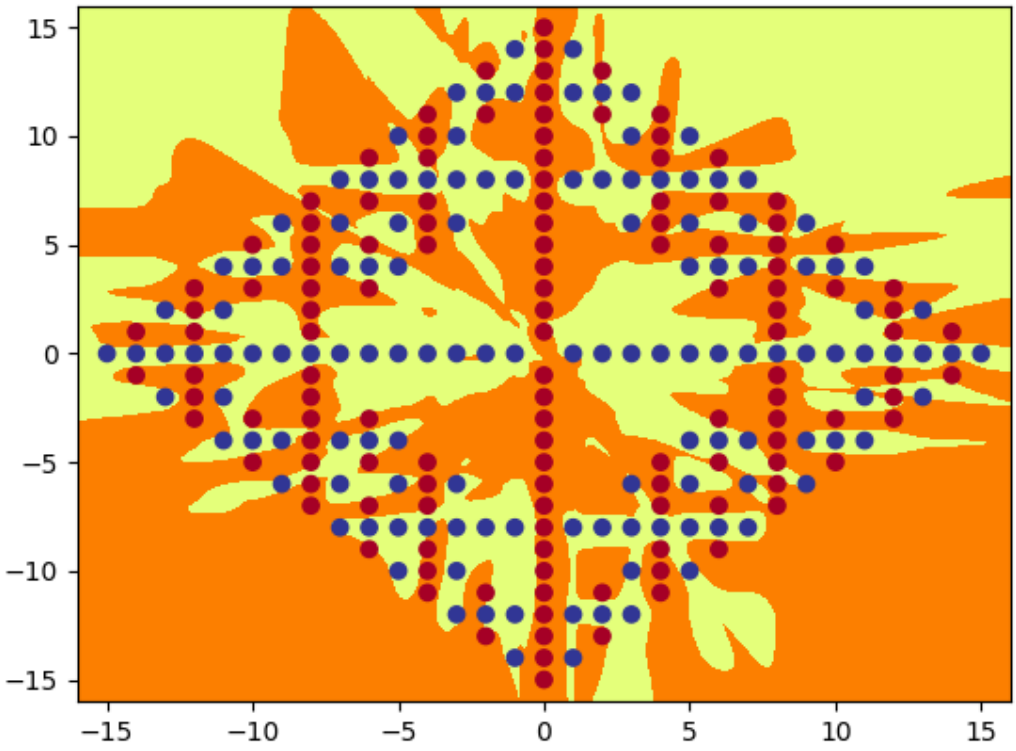
Trying out of curiosity, turns of using hid of 24 is the minimum number of hidden nodes that gives a model with 100% accuracy and a loss of 0.009. The plot it produces is below.



The total number of independent parameters = [(2+1)x24+(24+1)x24+(24+1)x24+(24+1)x1] = 1297

Question 6

Using the same lr and init values as before, it took 20 hidden nodes to reach 100% accuracy



The total number of independent parrameters = [(2+1)x20 + (22+1)x20 + (42+1)x1] = 563

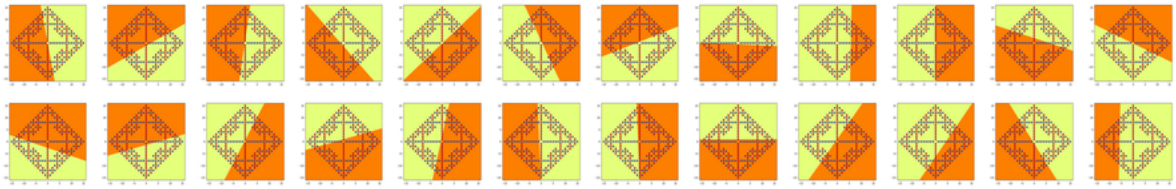
Question 7

a.

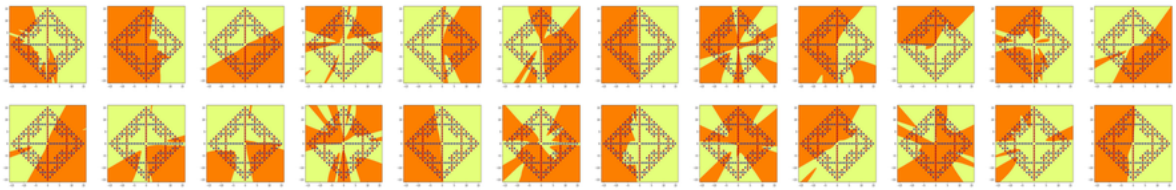
- Full2Net
  - HID: 28
  - Independent Parameters: 925
  - Epoch: 14800
- Full3Net
  - HID: 24
  - Independent Parameter: 1297
  - Epoch: 9700
- DenseNet
  - HID: 20
  - Independent Parameter: 563
  - Epoch: 12200

b.

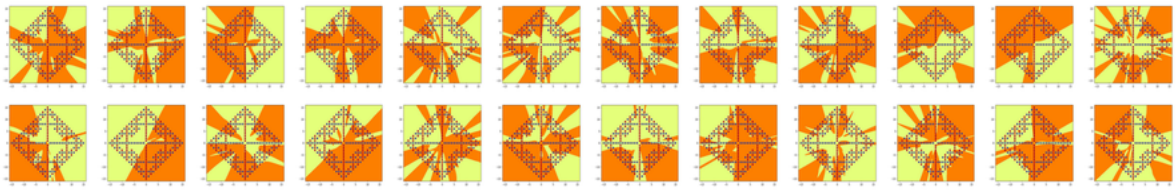
Full3Net layer 1:



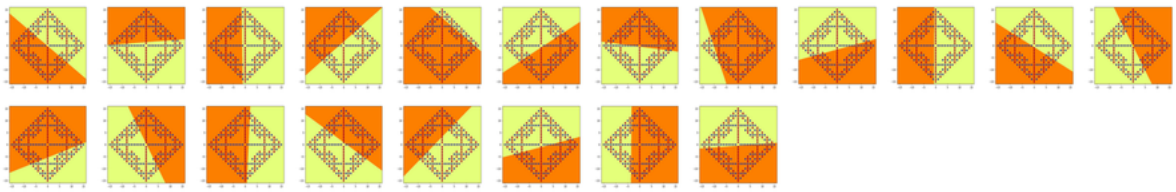
Full3Net layer 2:



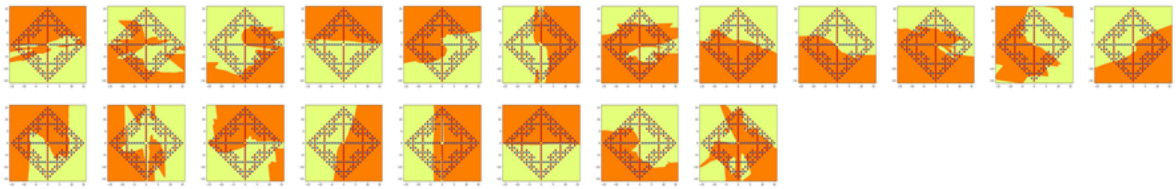
Full3Net Layer 3:



DenseNet Layer 1:

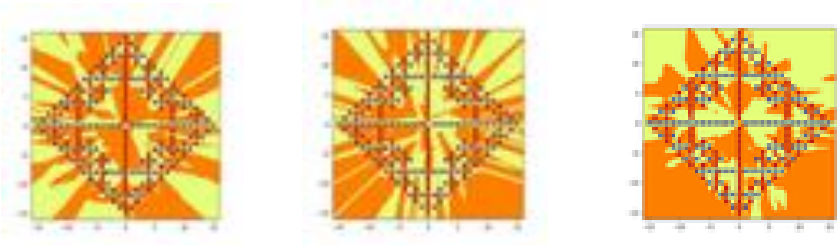


DenseNet Layer 2:



As we can see from the comparisons, layer 1 in both models look pretty much the same except that dense net model starts to show the orange color taking less than 50% of the plot towards the end. The layer 2 in Full3 starts with two continuous shapes but then starts to show abstract plot whereas densenet shows abstract plot from the beginning of layer 2. Layer 3 in the Full3Net looks different to the buildup of the Dense even tho Densenet finished up at layer 2, this produces more complex plots.

c. .

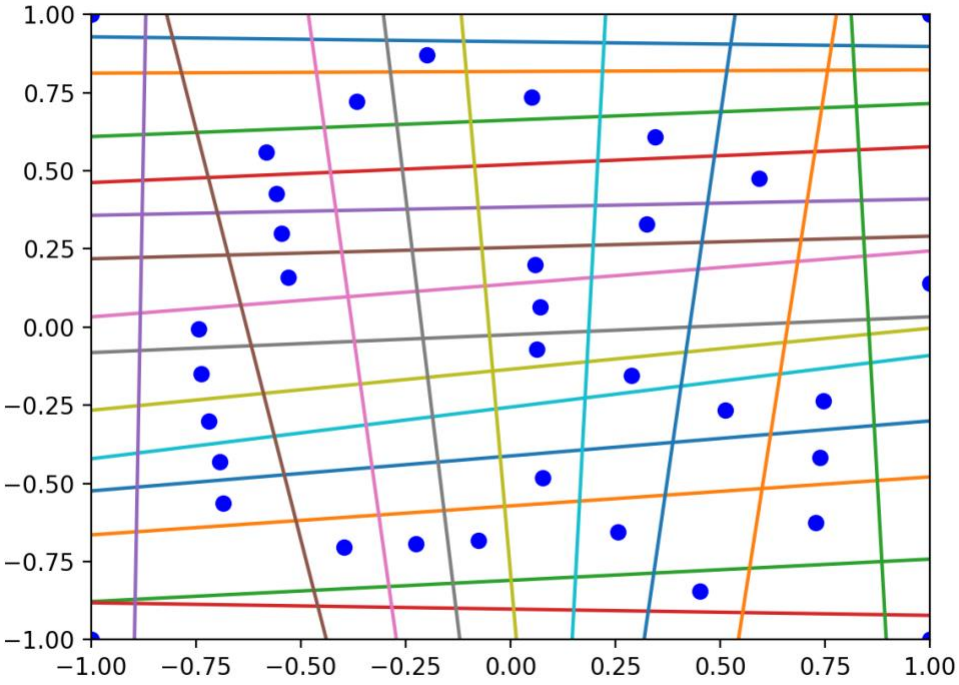


Full2Net Full3Net DenseNet

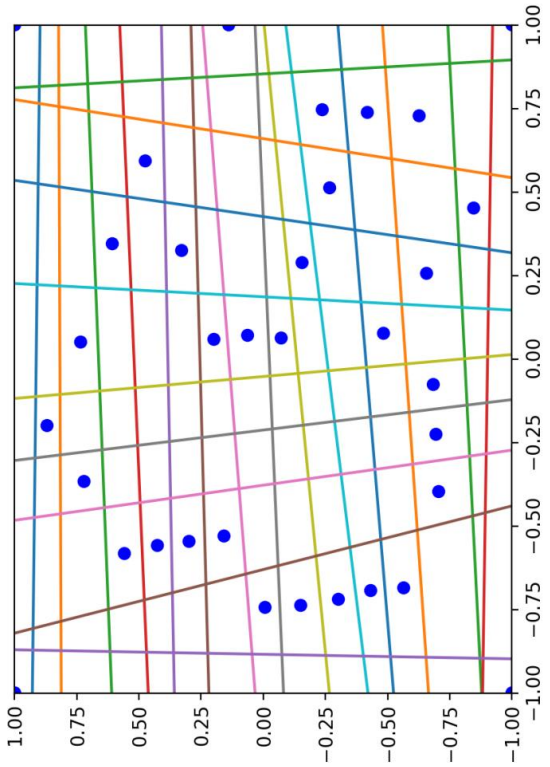
The outputs in all three models produced a 100% of accuracy. Although the Full3Net looks more sophisticated and looks like has not only gotten the accuracy of all points in dataset, but would also get a new point right if it was to be tested on this model. Followed by the Full2Net and then the DenseNet.

Part 3

The plot produces correctly the shape of mainland China except it is rotated right.



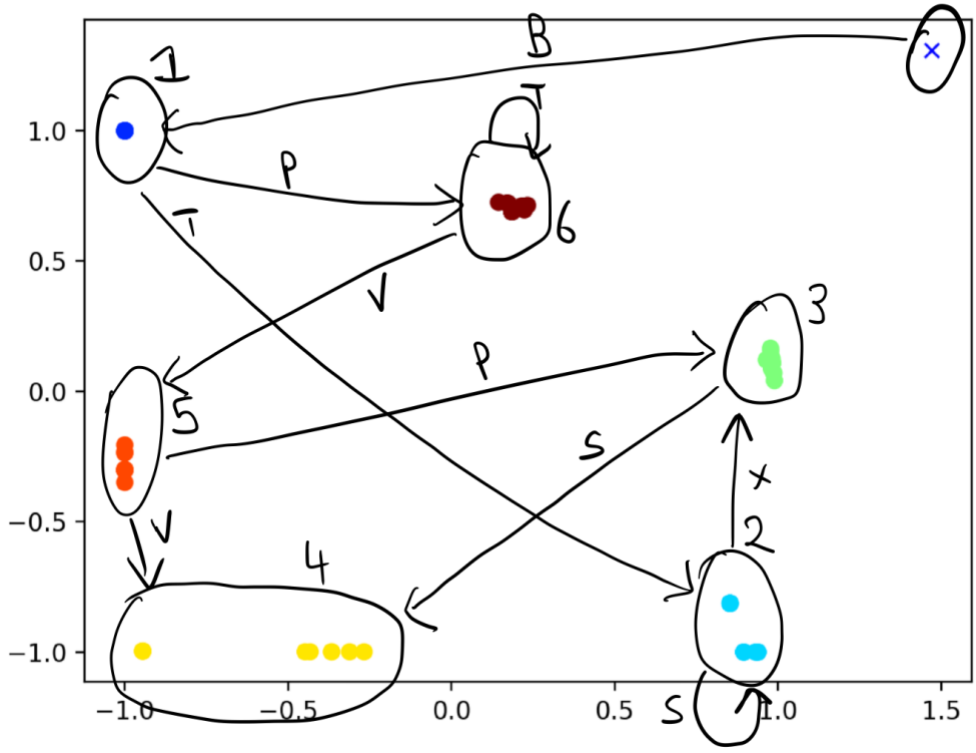
After rotating ^ to the left, it looks exactly like the expected output



Part 4

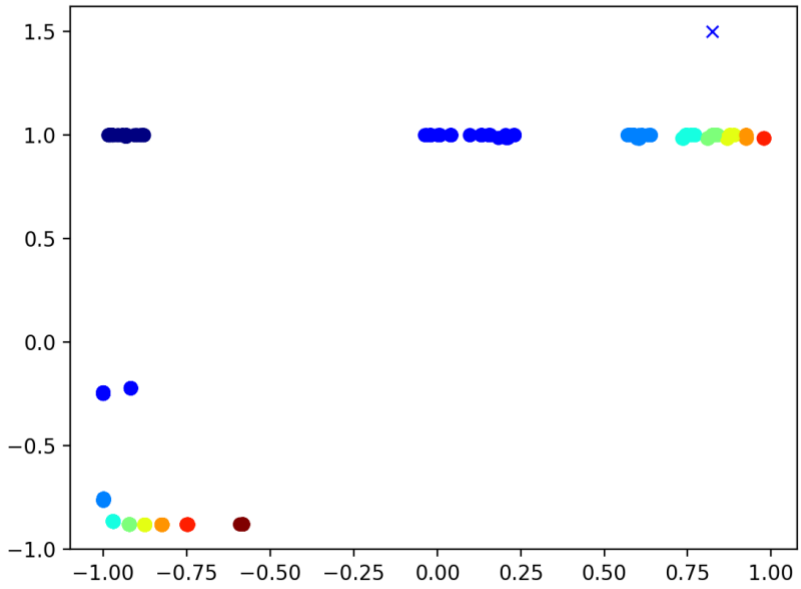
Question 1

After training the model with 50 epoch of Simple Recurrent Network (SRN) on the Reber Grammar prediction gives the following plot. On that the annotation shows that the input / start of the fsm is at x going to state 1, and then state 4 being the last state / outputs E.

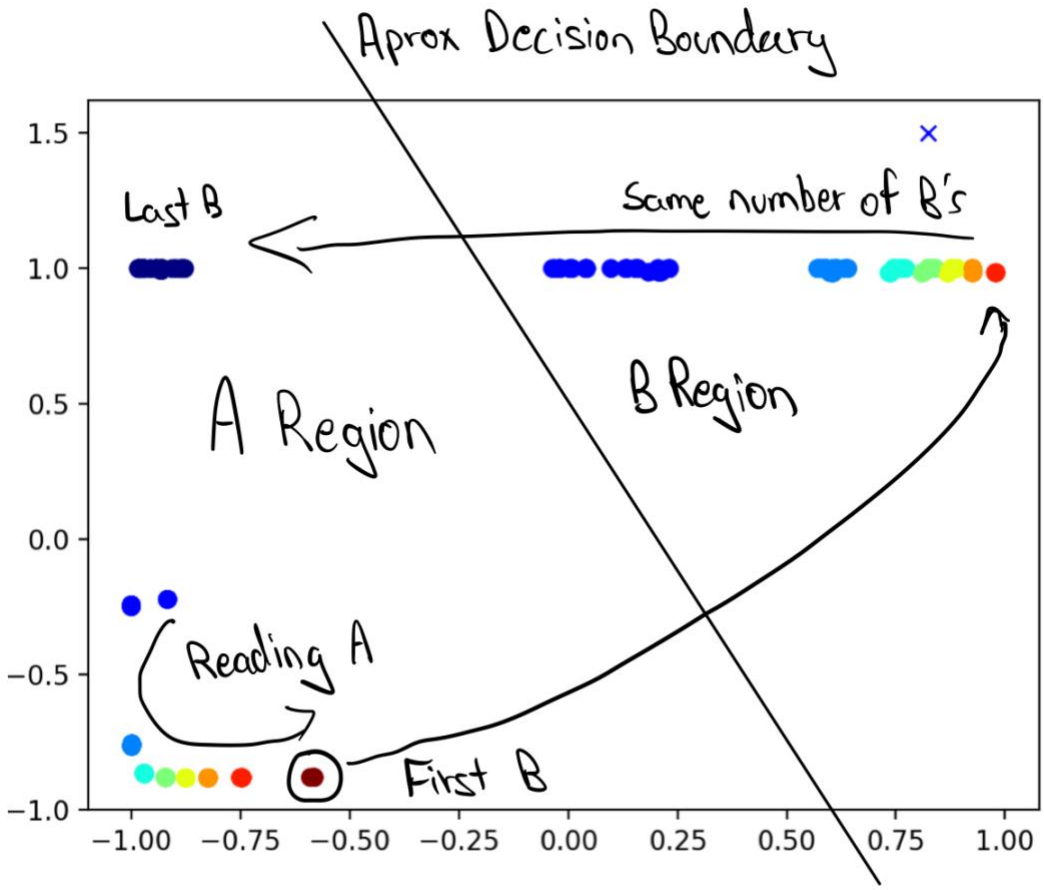


Question 2

After 40000 epochs, the plot produces :

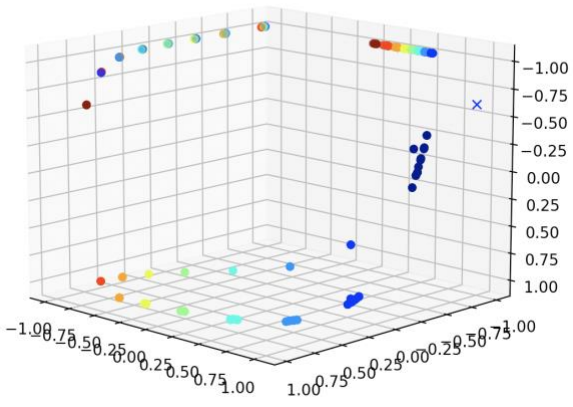


Question 3



The network is in a positive activation since it starts reading A's and until it receives the first B, after which the network goes into a negative activation and stays their for the equal number of B's as A is produced, until the last B which then goes back to the positive activation. Therefore, this model can correctly predict the number of B's (including the last but excluding the first) and finally predicts the correct subsequent A after the last B.

Question 4



Question 5

This network utilizes three dimensions to correctly predict the As, Bc and Cs. We can see that from the starting state A's are read and it stays in that region/plane until the first B is read. After which the dots start showing up on the far end on the B region/plane and get closer and closer towards the C region/plane until the first C is read. After which the same thing happens and the dots show up further from B plane and keeps getting closer to A plane and then the last C is read in the A plane. Therefore the model can correctly predict the Last B, First C, Last C and First A.

Question 6

```
state =  0 1 2 3 8 7 6 9 18
symbol= BTBPVVETE
label = 010455616
true probabilities:
     B    T    S    X    P    V    E
1 [ 0.   0.5  0.   0.   0.5  0.   0. ]
2 [ 1.  0.  0.  0.  0.  0.  0.]
3 [ 0.   0.5  0.   0.   0.5  0.   0. ]
8 [ 0.   0.5  0.   0.   0.   0.5  0. ]
7 [ 0.   0.   0.   0.   0.5  0.5  0. ]
6 [ 0.  0.  0.  0.  0.  0.  1.]
9 [ 0.  1.  0.  0.  0.  0.  0.]
18 [ 0.  0.  0.  0.  0.  0.  1.]
hidden activations and output probabilities [BTSXPVE]:
1 [-0.74  0.08 -0.75  0.55] [ 0.    0.51  0.    0.    0.49  0.    0.  ]
2 [ 0.06  0.75  0.7   0.88] [ 1.  0.  0.  0.  0.  0.  0.]
3 [-0.74  0.   -0.4  -0.04] [ 0.    0.5   0.    0.    0.49  0.01  0.  ]
8 [-0.39  0.72  0.69 -0.7 ] [ 0.    0.42  0.01  0.    0.    0.57  0.  ]
7 [-0.02 -0.71 -0.64 -0.95] [ 0.    0.01  0.    0.    0.43  0.56  0.  ]
6 [-0.01 -0.89  0.61 -0.53] [ 0.  0.  0.  0.  0.  0.  1.]
9 [-0.91 -0.01 -0.71 -0.05] [ 0.    0.48  0.    0.    0.52  0.    0.  ]
18 [ 0.13 -0.98  0.73 -0.92] [ 0.  0.  0.  0.  0.  0.  1.]
epoch: 50000
error: 0.0010
final: 0.0777
Context Unit:
 tensor([[[-0.9902,  0.9486, -0.9858,  0.6238],
          [ 0.0686,  1.0068,  0.8888,  1.3923],
          [-0.9507,  0.0396, -0.4366, -0.0355],
          [-1.1585,  0.9576,  0.8520, -0.8961],
          [-0.0970, -0.9043, -0.7624, -1.8692],
          [-0.8940, -1.5137,  0.7049, -2.1931],
          [-1.7339, -2.3357, -0.8939, -3.0770],
          [ 0.2409, -2.7531,  0.9340, -1.6152]]])
```

The symbol BTBPVVETE is inputted here. We can first see that the output probability matrix shows aprox 50% chance for both T and P. This also activates the context matrix updates the value going from -0.9858 to +0.8888, this update helps to remember the order and can be used for further predictions. This kind of pattern can be seen throughout and can remember the order of input. Therefore here the Context Unit plays a major role to remember the order which can be used to exit the state machine.