COMP9444 Assignment 1 - Network Structures and Hidden Unit Dynamics, by z5311922

Part 1: Fractal Classification Task

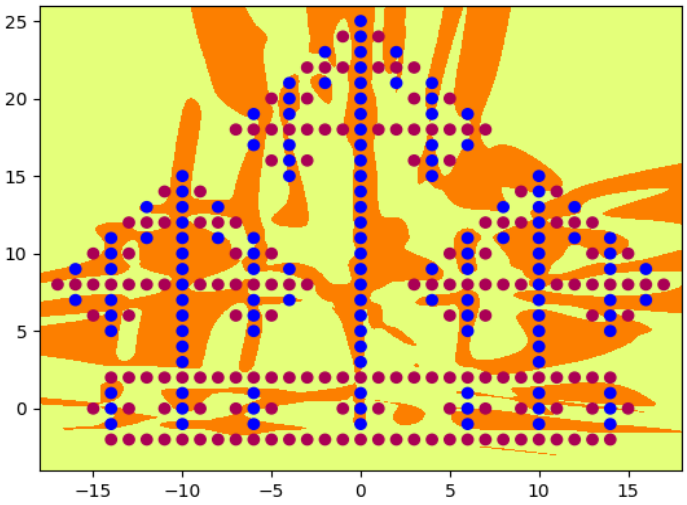
*Note that for Q2, Q4, Q6 I have outlined my process of determining the minimum hidden nodes required for each network to learn the task at the end of Part 1 Q7. **

Q1)

Check the attached crown.py as required.

Q2)

Graph output for Full3Net as generated by graph_output() method for the function computed by the network.



Full3Net out_full3_20.png

Full3Net was trained using 20 hidden nodes per layer with learning rate 0.005 and took 161,400 to learn the classification task with a consistent accuracy of 100. The number of parameters used for this network is given by the following calculation:

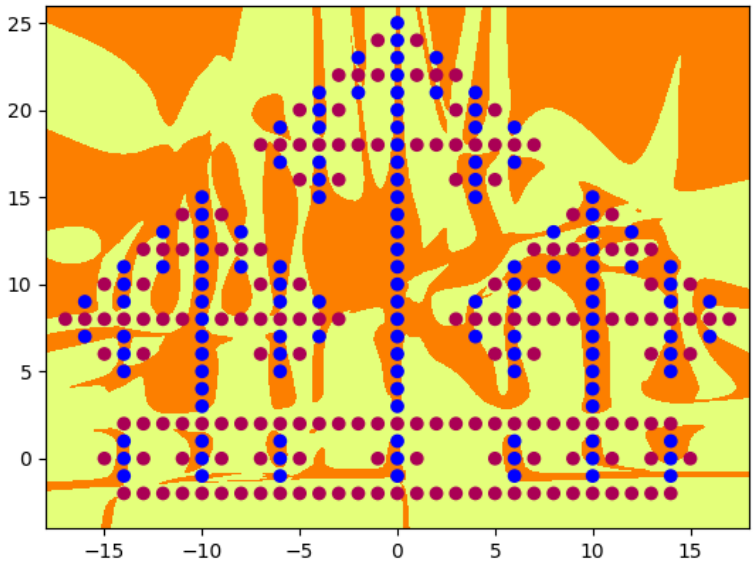
Num_inputs = 2
hid = 20
Hidden_layers = 2
Output_nodes = 1
*Total params = (2 * 20 + 20) + (20 * 20+ 20) + (20 * 1 + 1)*
= 501

Q3)

Check the attached crown.py as required.

Q4)

Graph Output for Full4Net as generated by graph_output().

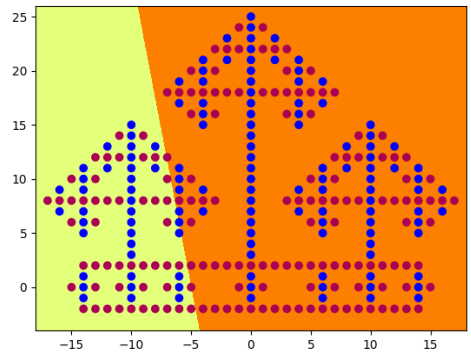


Full4Net out_full4_17.png

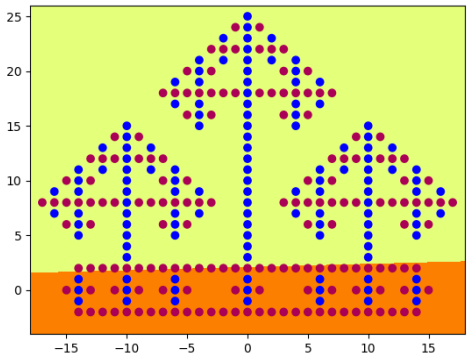
The Full4Net was trained using 17 hidden nodes per hidden layer with learning rate 0.005 and took 198,400 epochs to achieve consistent accuracy of 100. The initial weights were set as default to allow consistent comparison with other models. The number of total independent parameters for this network is given by the calculation below.

$$Num\_inputs = 2$$
$$hid = 17$$
$$Hidden\_layers = 3$$
$$Output\_nodes = 1$$
$$Total\ params = (2 * 17 + 17) + (17 * 17 + 17) + (17 * 17 + 17) + (17 * 1 + 1)$$
$$= 681$$

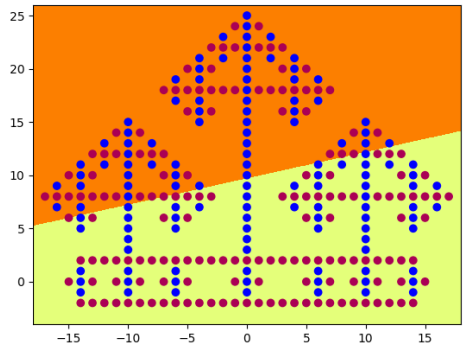
Plots of all hidden units in all 3 layers for network Full4Net.



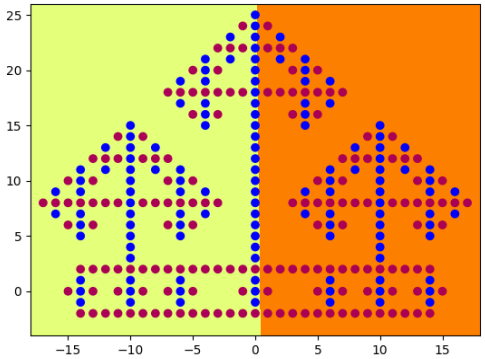
Full4Net Hidden Layer 1 Node 0



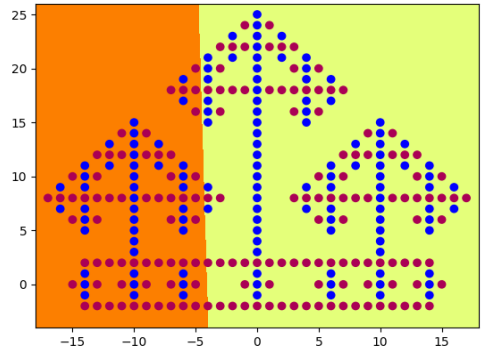
Full4Net Hidden Layer 1 Node 1



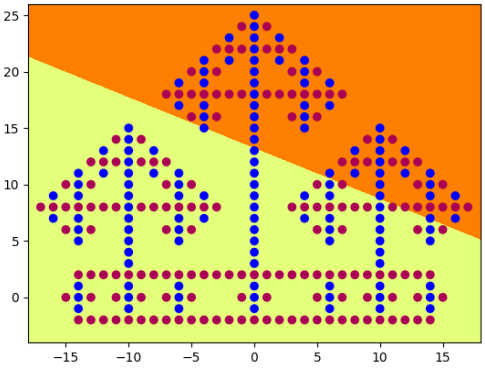
Full4Net Hidden Layer 1 Node 2



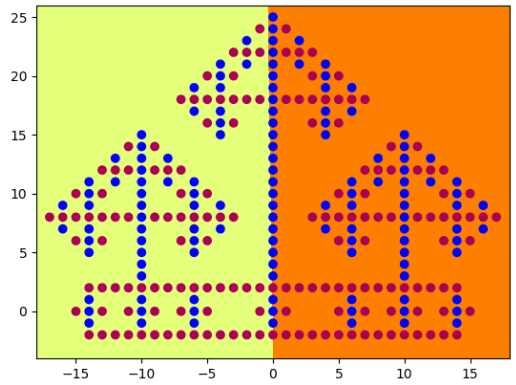
Full4Net Hidden Layer 1 Node 3



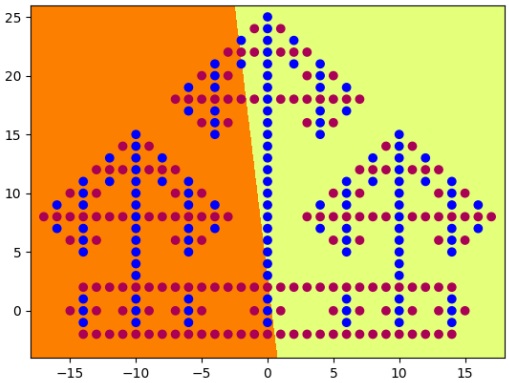
Full4Net Hidden Layer 1 Node 4



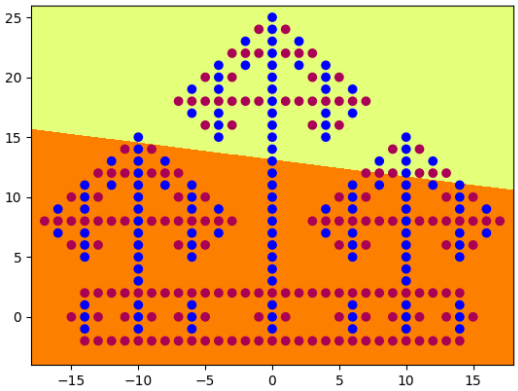
Full4Net Hidden Layer 1 Node 5



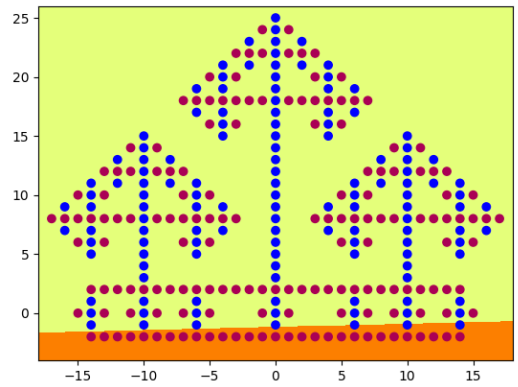
Full4Net Hidden Layer 1 Node 6



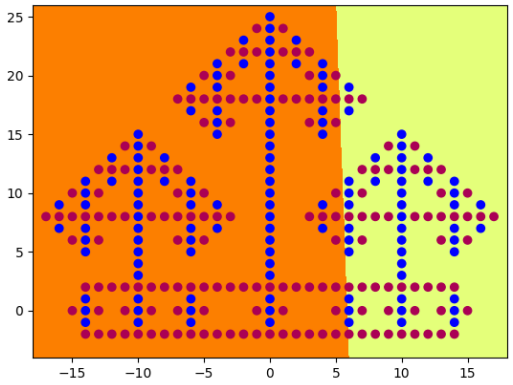
Full4Net Hidden Layer 1 Node 7



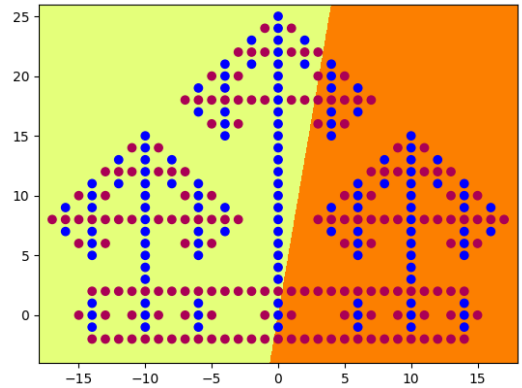
Full4Net Hidden Layer 1 Node 8



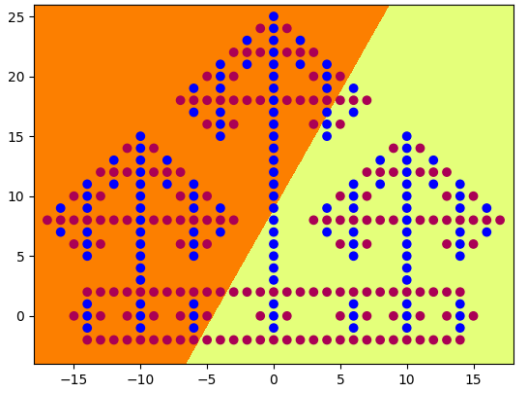
Full4Net Hidden Layer 1 Node 9



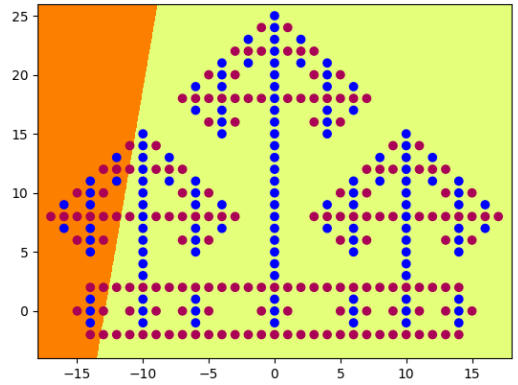
Full4Net Hidden Layer 1 Node 10



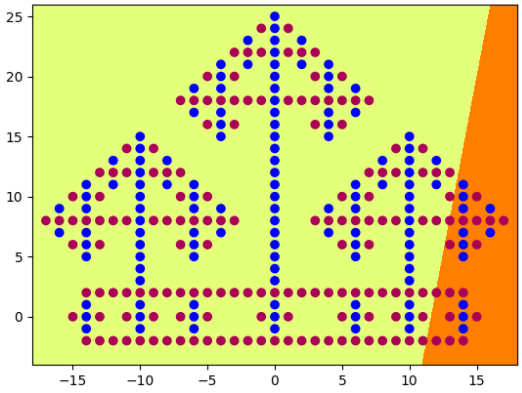
Full4Net Hidden Layer 1 Node 11



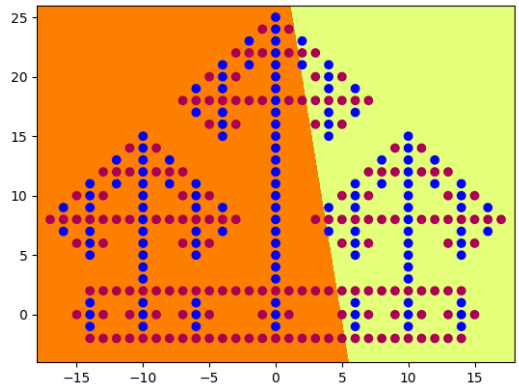
Full4Net Hidden Layer 1 Node 12



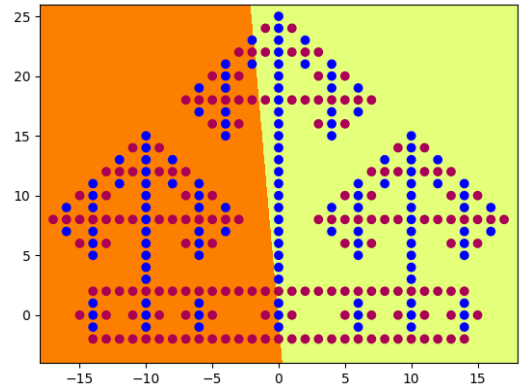
Full4Net Hidden Layer 1 Node 13



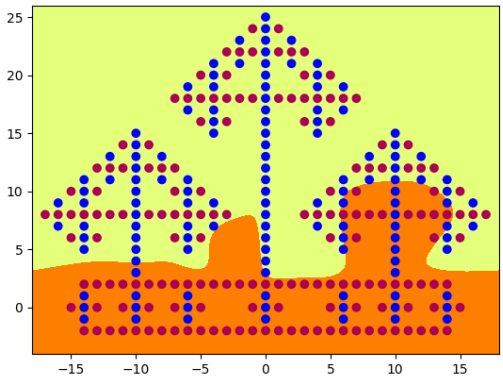
Full4Net Hidden Layer 1 Node 14



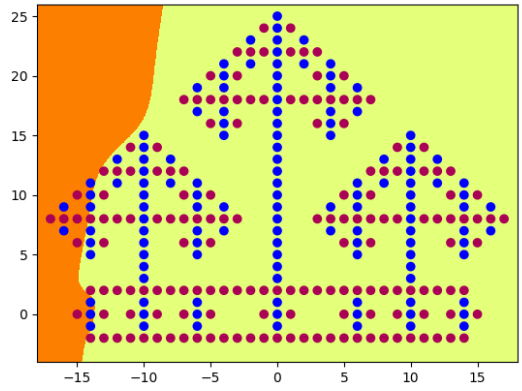
Full4Net Hidden Layer 1 Node 15



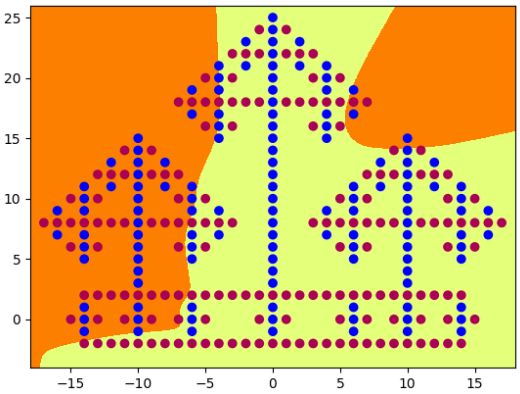
Full4Net Hidden Layer 1 Node 16



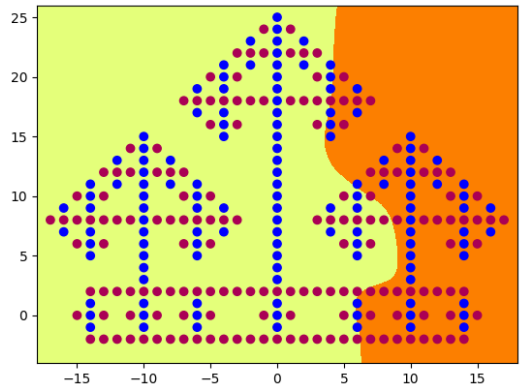
Full4Net Hidden Layer 2 Node 0



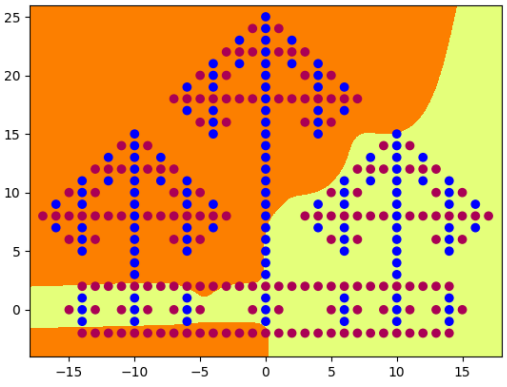
Full4Net Hidden Layer 2 Node 1



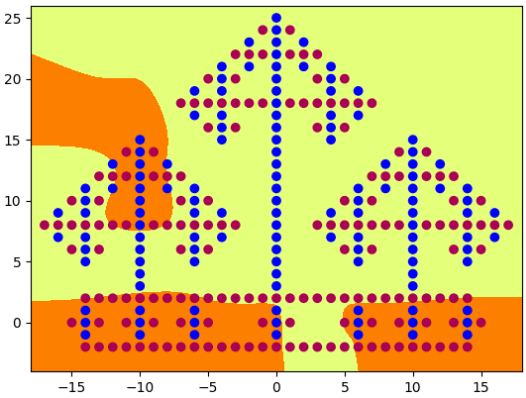
Full4Net Hidden Layer 2 Node 2



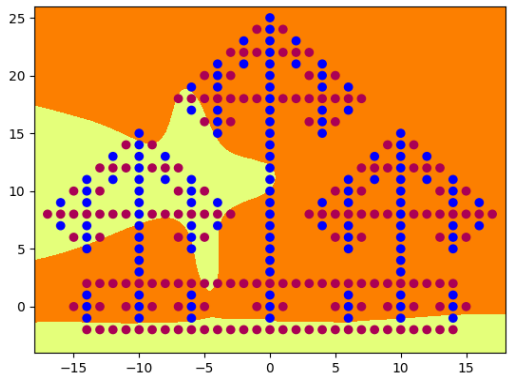
Full4Net Hidden Layer 2 Node 3



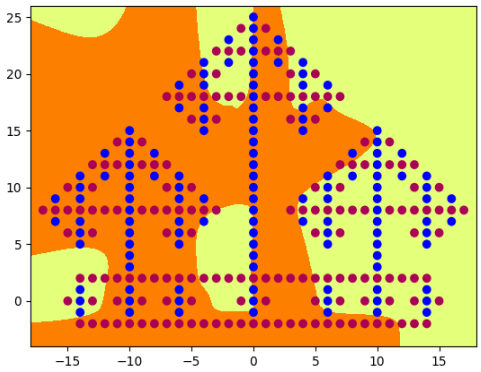
Full4Net Hidden Layer 2 Node 4



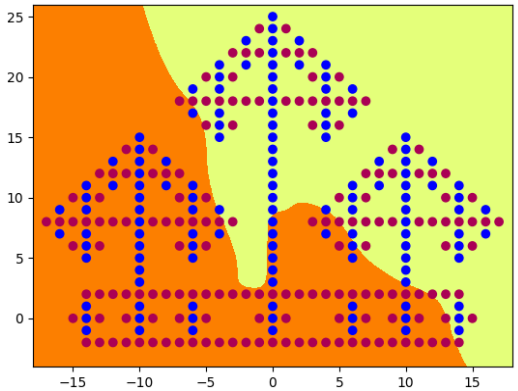
Full4Net Hidden Layer 2 Node 5



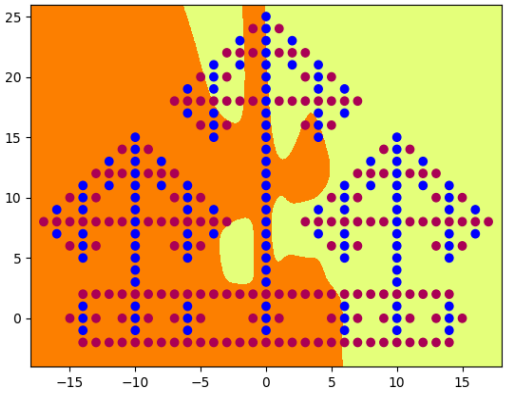
Full4Net Hidden Layer 2 Node 6



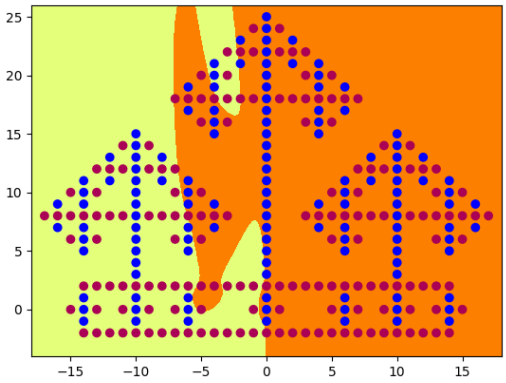
Full4Net Hidden Layer 2 Node 7



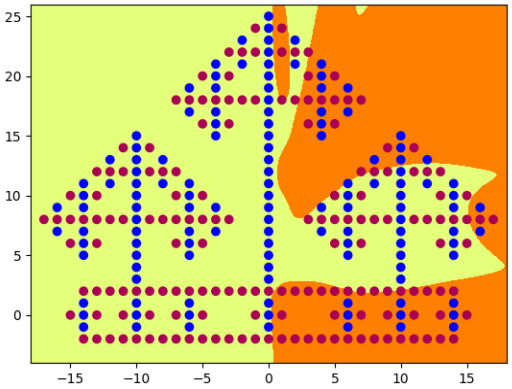
Full4Net Hidden Layer 2 Node 8



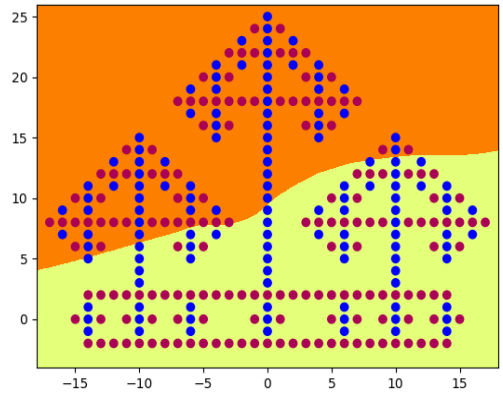
Full4Net Hidden Layer 2 Node 9



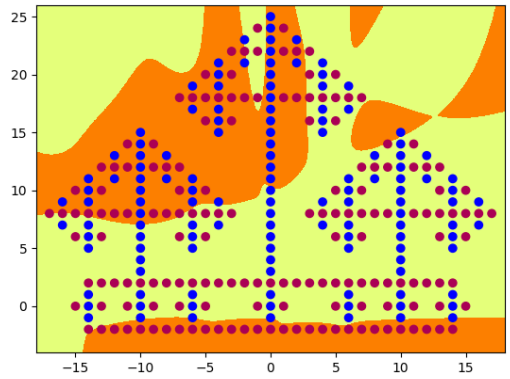
Full4Net Hidden Layer 2 Node 10



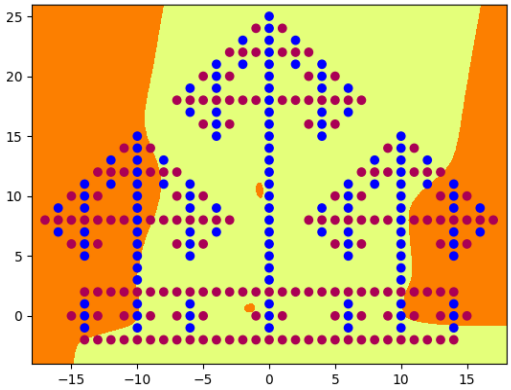
Full4Net Hidden Layer 2 Node 11



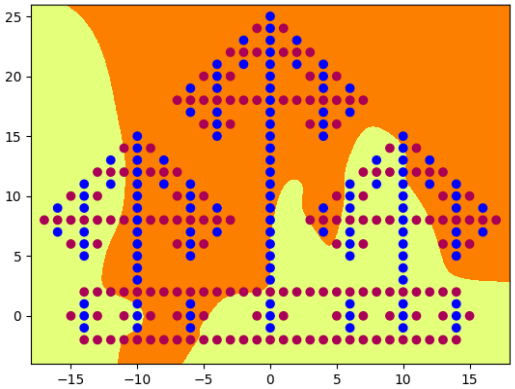
Full4Net Hidden Layer 2 Node 12



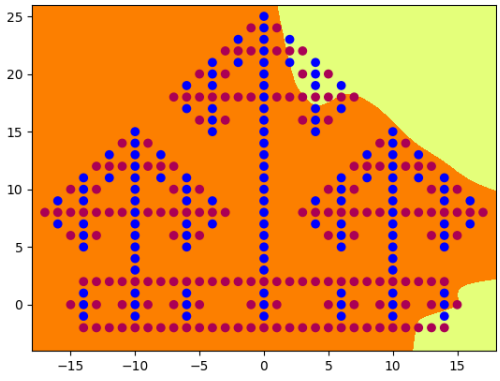
Full4Net Hidden Layer 2 Node 13



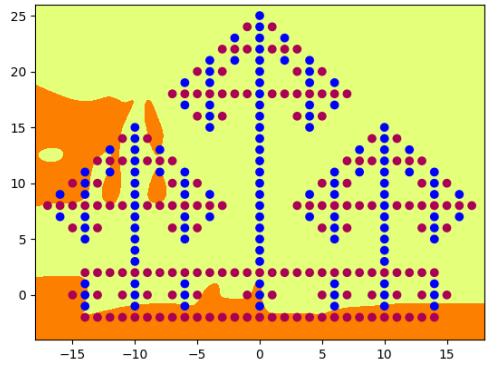
Full4Net Hidden Layer 2 Node 14



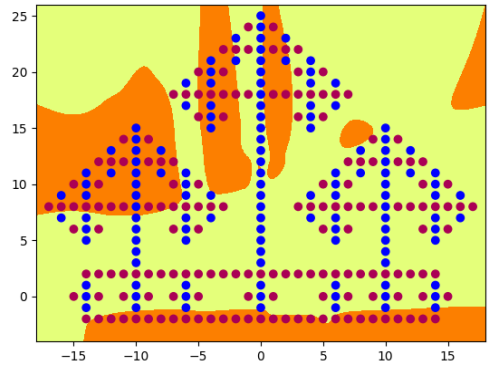
Full4Net Hidden Layer 2 Node 15



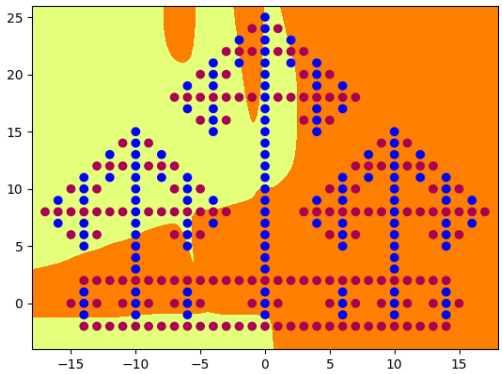
Full4Net Hidden Layer 2 Node 16



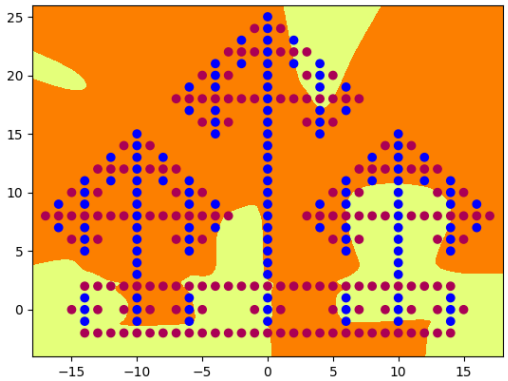
Full4Net Hidden Layer 3 Node 0



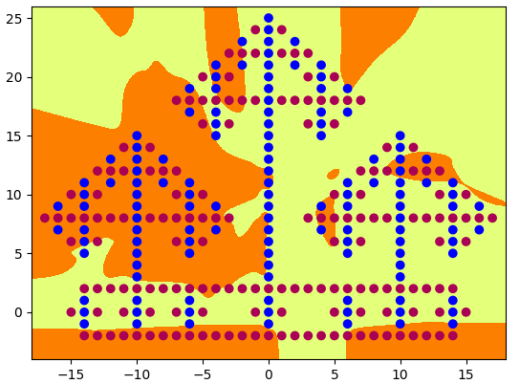
Full4Net Hidden Layer 3 Node 1



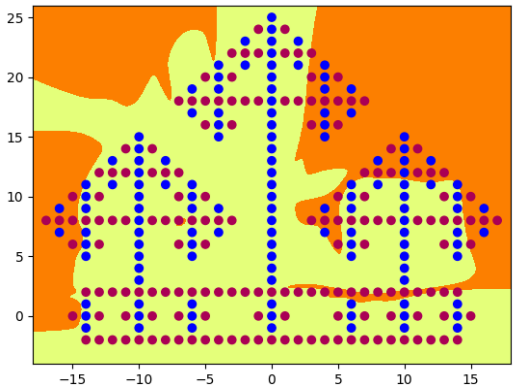
Full4Net Hidden Layer 3 Node 2



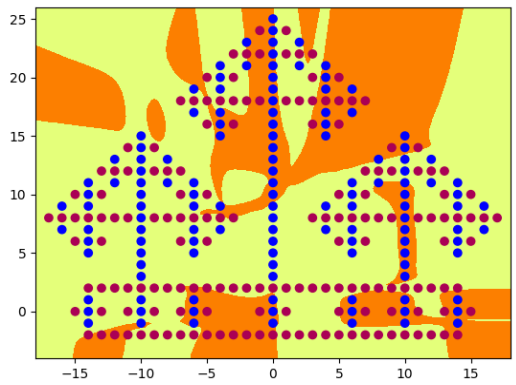
Full4Net Hidden Layer 3 Node 3



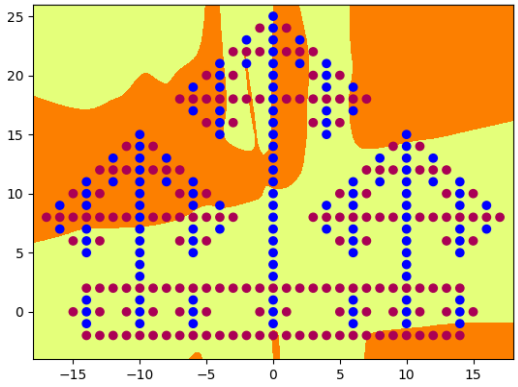
Full4Net Hidden Layer 3 Node 4



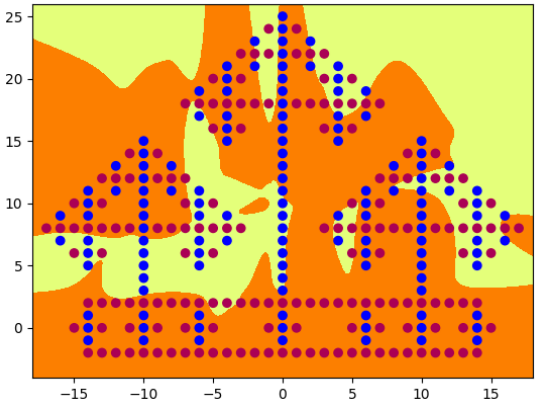
Full4Net Hidden Layer 2 Node 5



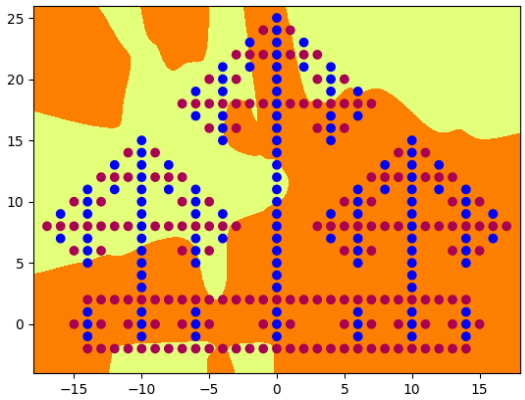
Full4Net Hidden Layer 3 Node 6



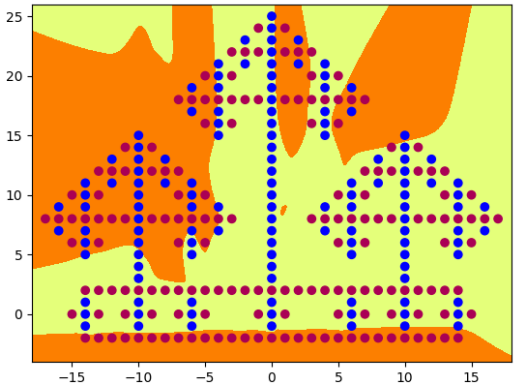
Full4Net Hidden Layer 3 Node 7



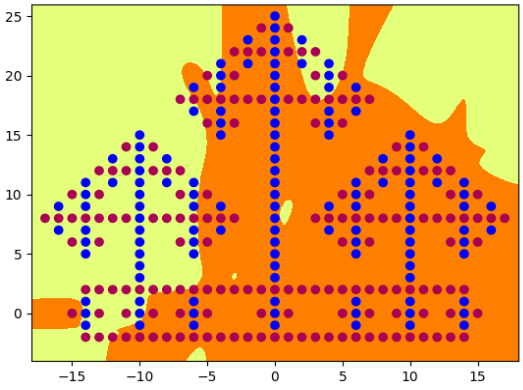
Full4Net Hidden Layer 3 Node 8



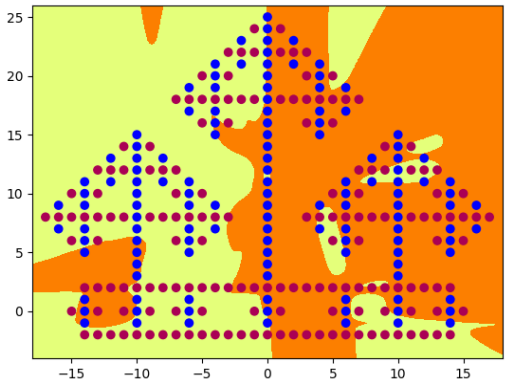
Full4Net Hidden Layer 3 Node 9



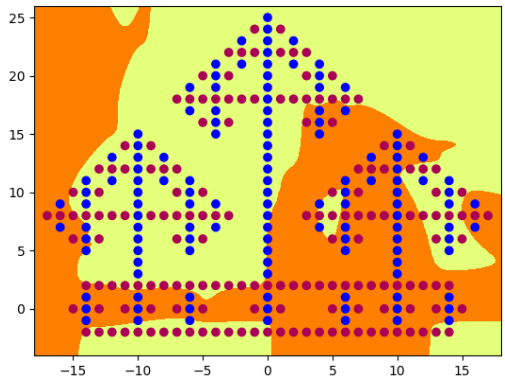
Full4Net Hidden Layer 3 Node 10



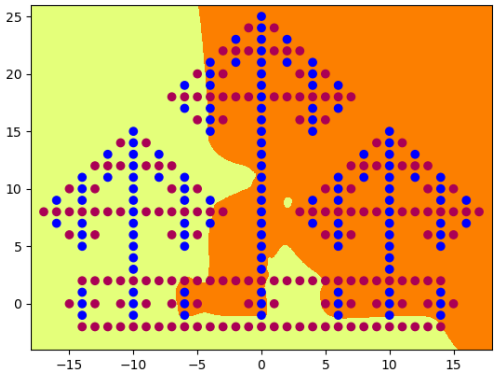
Full4Net Hidden Layer 2 Node 11



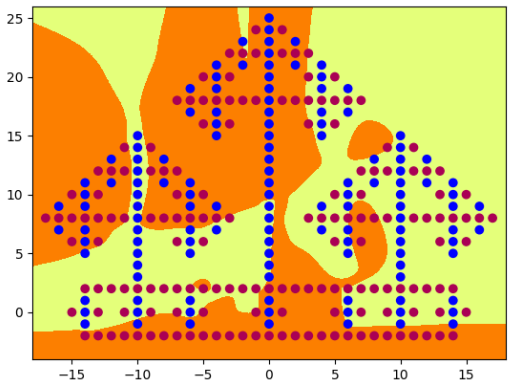
Full4Net Hidden Layer 3 Node 12



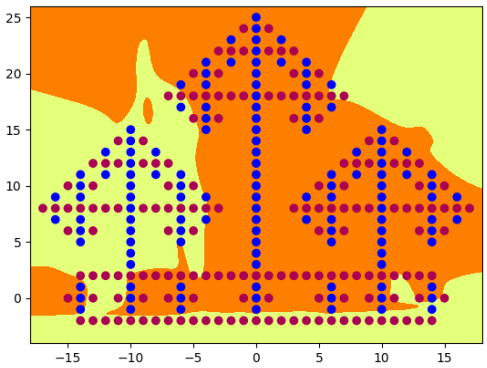
Full4Net Hidden Layer 3 Node 13



Full4Net Hidden Layer 3 Node 14



Full4Net Hidden Layer 3 Node 15



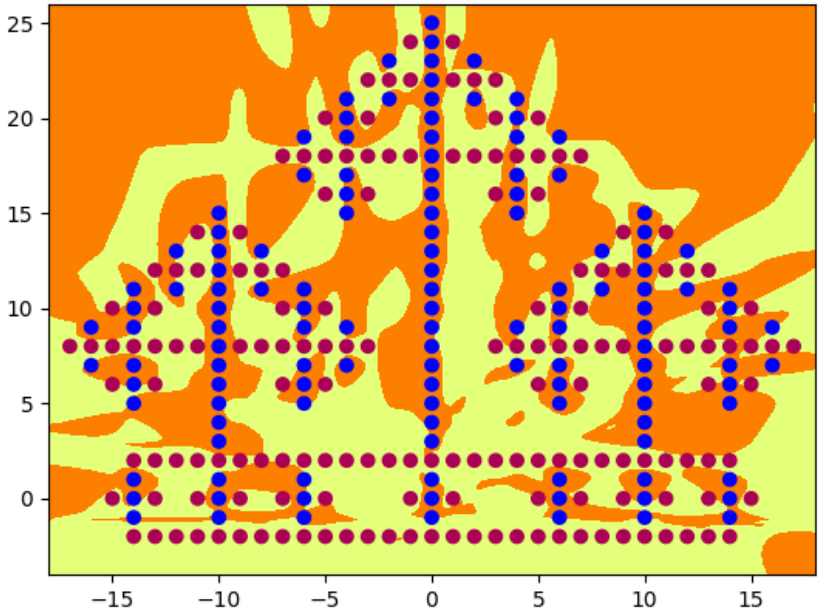
Full4Net Hidden Layer 3 Node 16

Q5)

Check the attached crown.py as required.

Q6)

Graph Output for DenseNet as generated by graph_output().



DenseNet out_dense_16.png

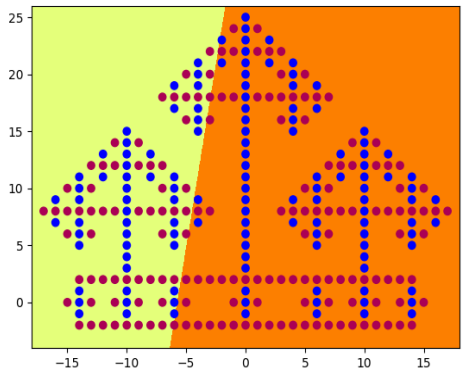
The DenseNet was trained using 16 hidden nodes per hidden layer. This network was trained using a learning rate of 0.005 to keep consistent with other models and initial weights of 0.15. This model took 108,100 epochs to consistently achieve an accuracy of 100.

With 16 hidden nodes as specified by the hid variable, the total number of independent parameters in the network is given by:

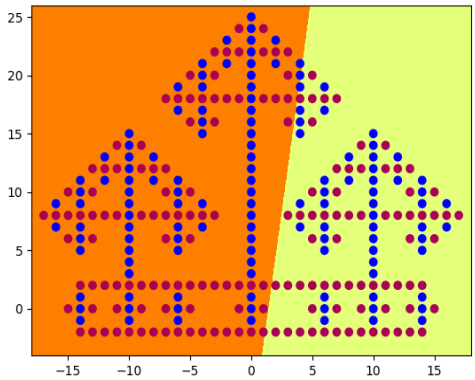
$$Input\_nodes = 2$$
$$Layer1\_nodes = 16$$
$$Layer2\_nodes = 16$$
$$Output\_layer = 1$$

$$Total\ parameters = (2 * 16 + 16) + (16 * 16 + 16) + (16*1 + 1) + (16*2 + 16) + (2*1 + 1) + (16*1 + 1)$$
$$= 405$$

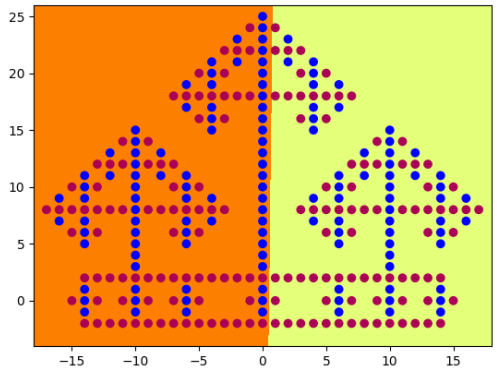
Plots of all hidden units in all 2 layers for network DenseNet.



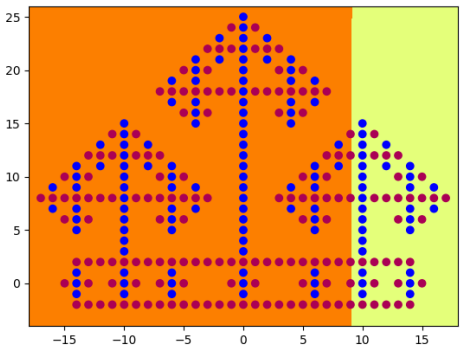
DenseNet Hidden Layer 1 Node 0



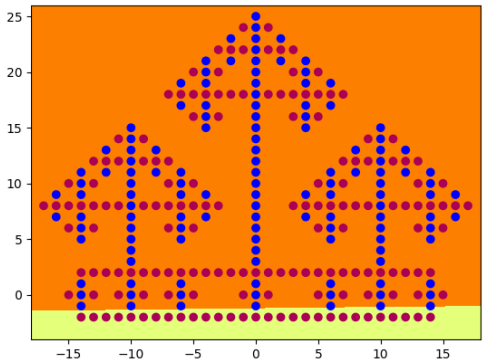
DenseNet Hidden Layer 1 Node 1



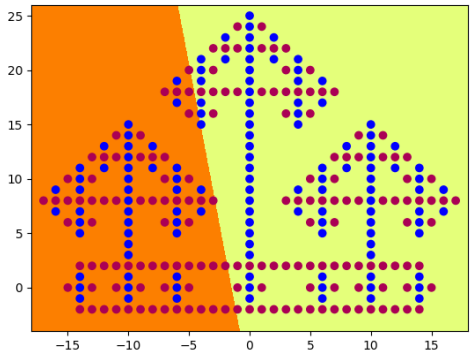
DenseNet Hidden Layer 1 Node 2



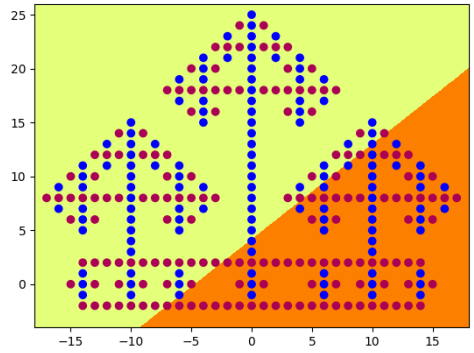
DenseNet Hidden Layer 1 Node 3



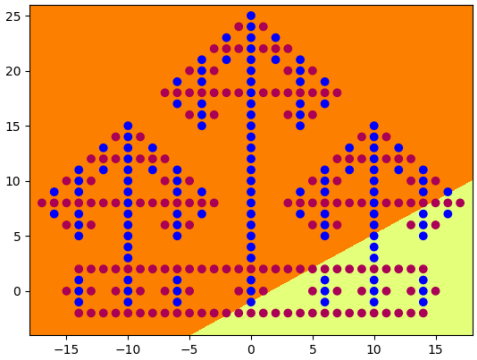
DenseNet Hidden Layer 1 Node 4



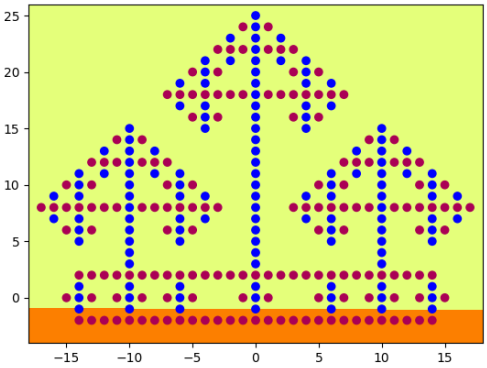
DenseNet Hidden Layer 1 Node 5



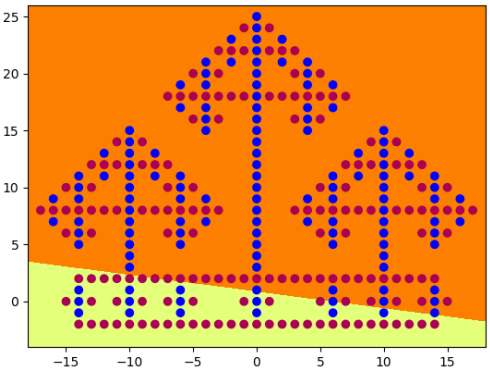
DenseNet Hidden Layer 1 Node 6



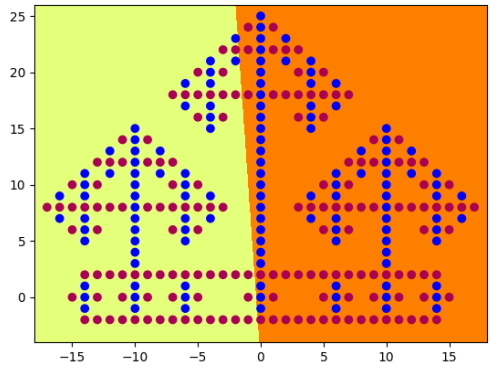
DenseNet Hidden Layer 1 Node 7



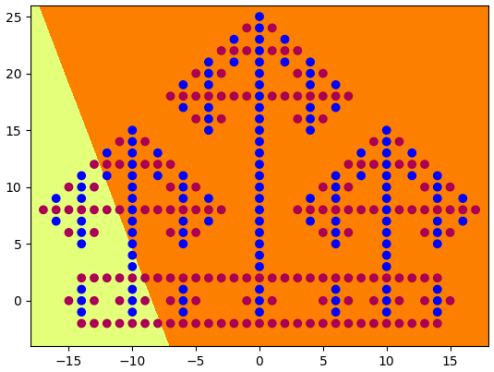
DenseNet Hidden Layer 1 Node 8



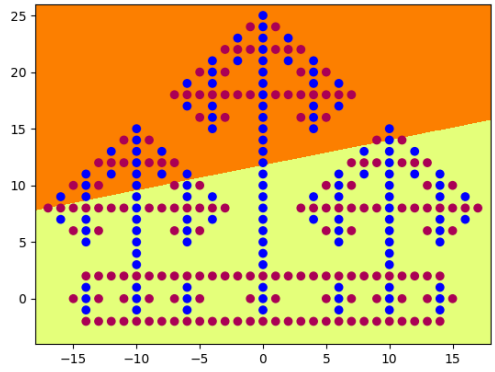
DenseNet Hidden Layer 1 Node 9



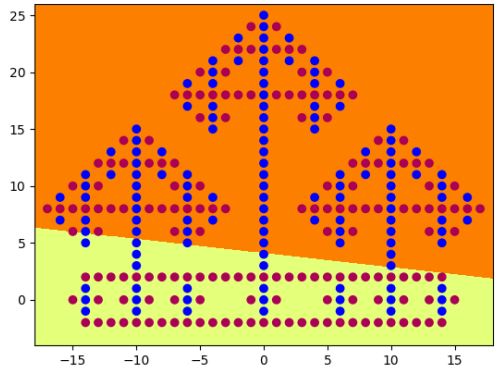
DenseNet Hidden Layer 1 Node 10



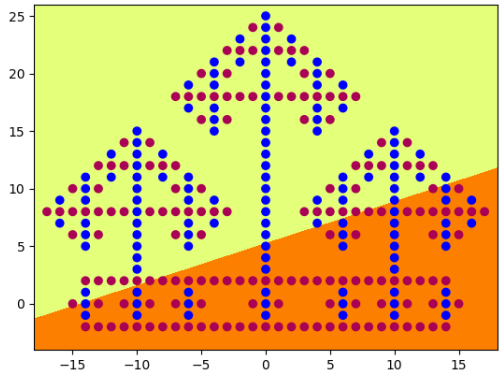
DenseNet Hidden Layer 1 Node 11



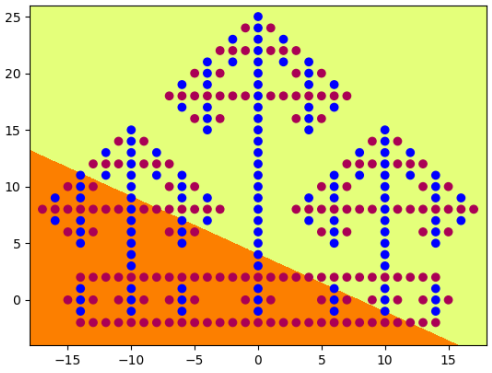
DenseNet Hidden Layer 1 Node 12



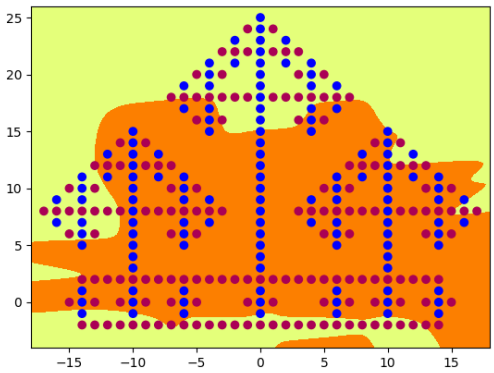
DenseNet Hidden Layer 1 Node 13



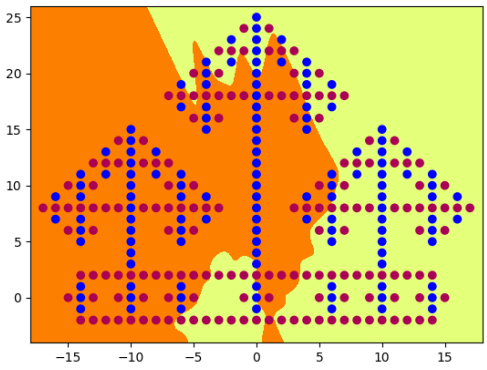
DenseNet Hidden Layer 1 Node 14



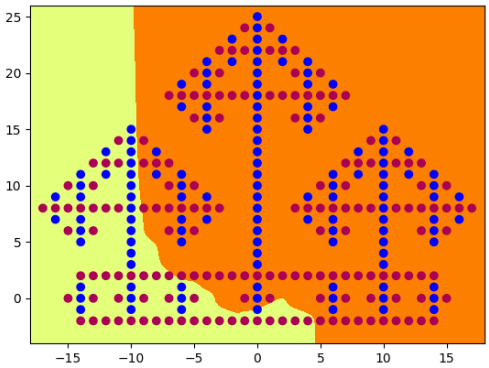
DenseNet Hidden Layer 1 Node 15



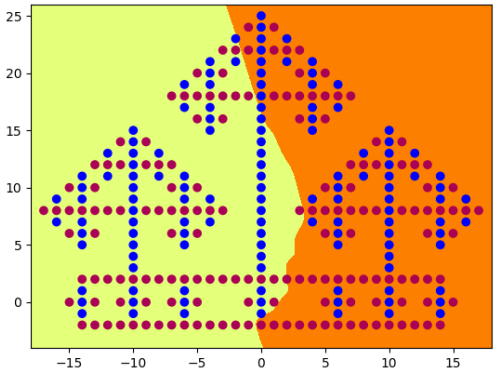
DenseNet Hidden Layer 2 Node 0



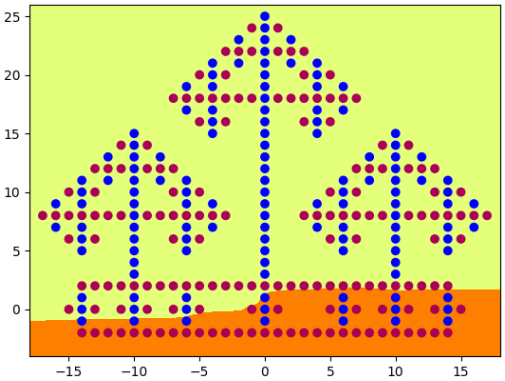
DenseNet Hidden Layer 2 Node 1



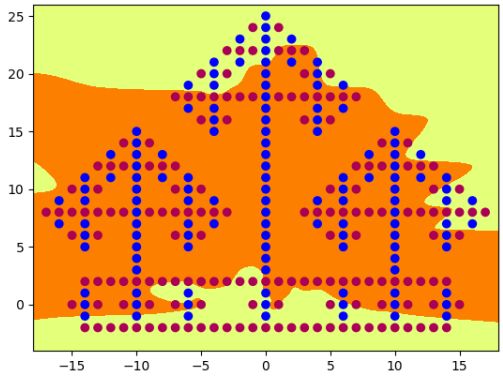
DenseNet Hidden Layer 2 Node 2



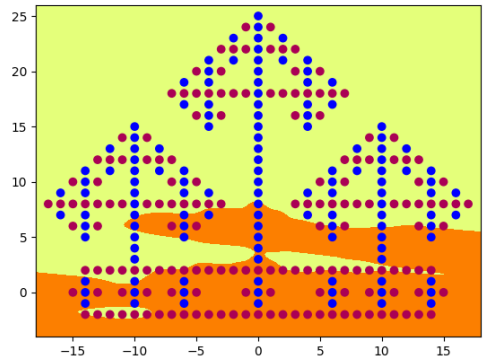
DenseNet Hidden Layer 2 Node 3



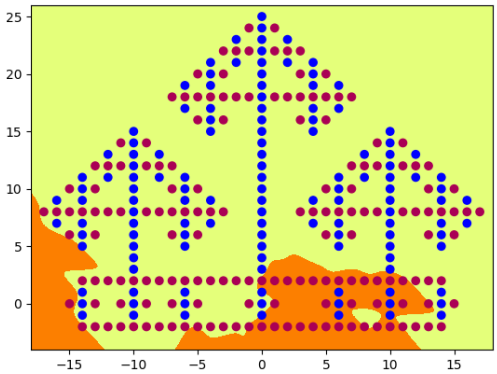
DenseNet Hidden Layer 2 Node 4



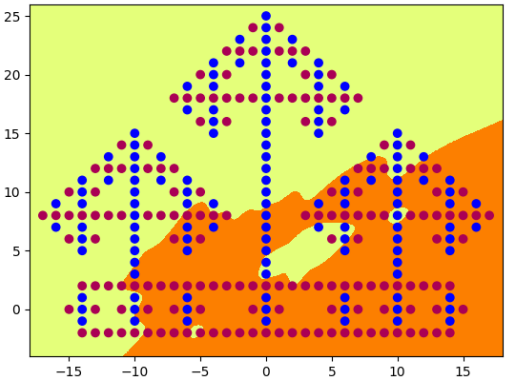
DenseNet Hidden Layer 2 Node 5



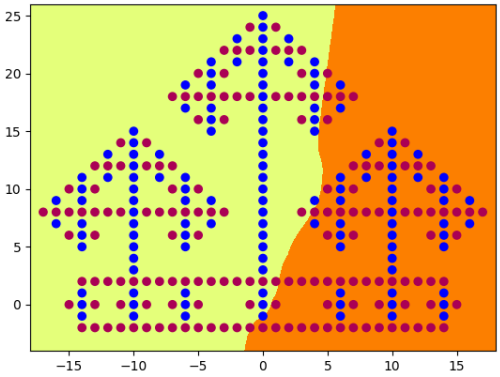
DenseNet Hidden Layer 2 Node 6



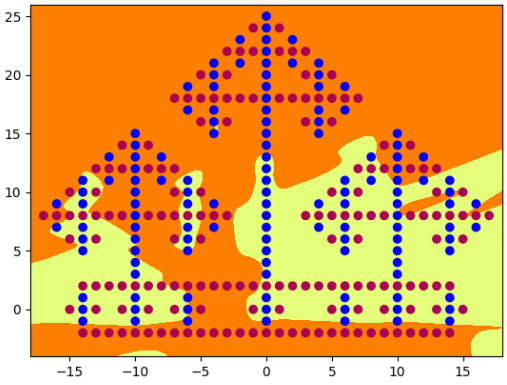
DenseNet Hidden Layer 2 Node 7



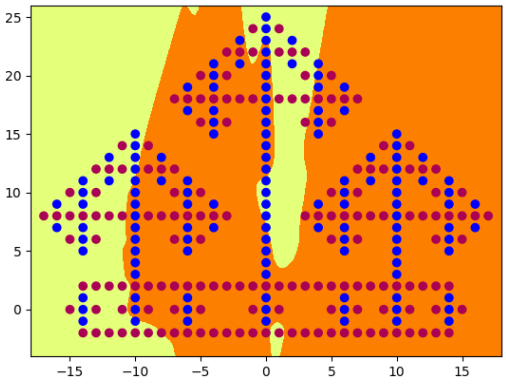
DenseNet Hidden Layer 2 Node 8



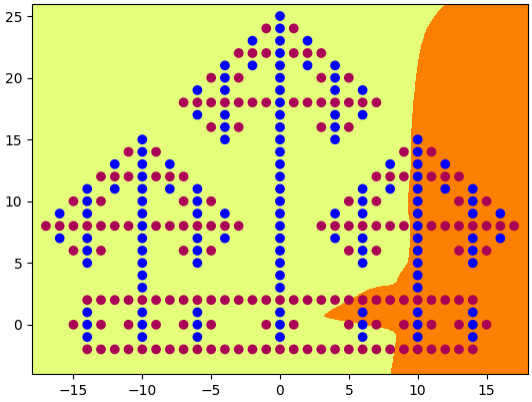
DenseNet Hidden Layer 2 Node 9



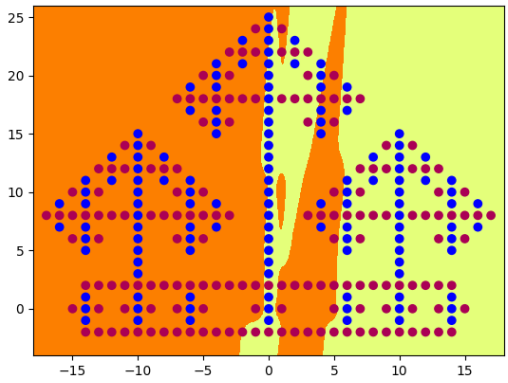
DenseNet Hidden Layer 2 Node 10



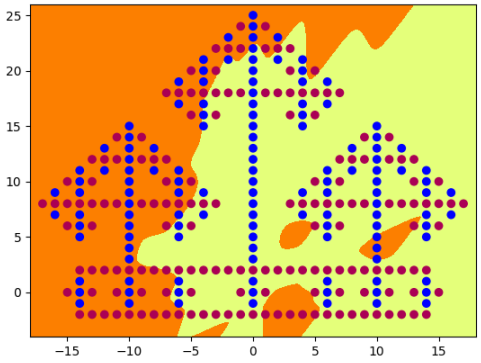
DenseNet Hidden Layer 2 Node 11



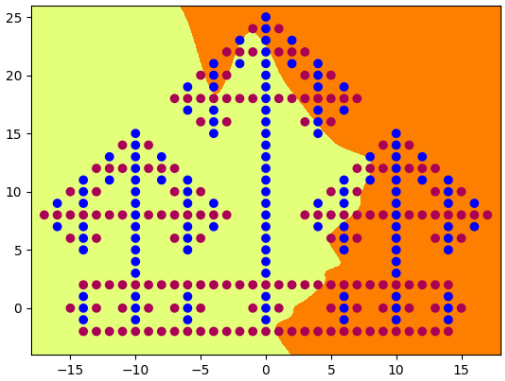
DenseNet Hidden Layer 2 Node 12



DenseNet Hidden Layer 2 Node 13



DenseNet Hidden Layer 2 Node 14



DenseNet Hidden Layer 2 Node 15

Q7)

Below is a table for each network trained including the number of parameters, the learning rate, initial weight, number of epochs taken to consistently achieve accuracy of 100 and the time it took to train the network to learn the task.

Model	Number of Parameters	Number of nodes per layer (hid)	Learning Rate	Initial Weight	Number of epochs	Time (seconds)
Full3Net	501	20	0.005	0.15	161,400	121.56
Full4Net	681	17	0.005	0.15	198,400	173.39
DenseNet	405	16	0.005	0.15	108,100	94.58

To compare between each network, I have kept the learning rate and initial weight consistent. We can see that Full4Net has significantly more parameters than Full3Net even with less nodes per layer because of the extra hidden layer and that the DenseNet has the least number of parameters because we only required 16 hidden nodes per layer to achieve an accuracy of 100. Since Full4Net has the largest total number of independent parameters, it also took the longest time to train. While DenseNet with the lowest number of independent parameters took significantly less time to train. Thus, we can see a correlation with the number of independent parameters and the time taken to train, where the larger the number of independent parameters the longer it took the model to train.

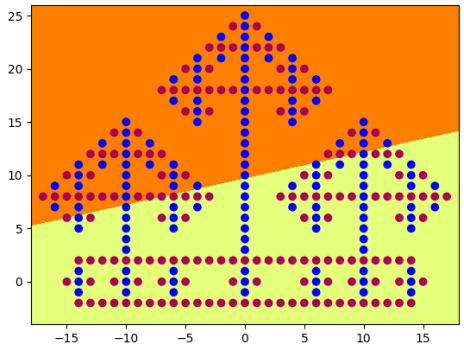
Full3Net with 2 hidden layers is good enough to solve this classification problem using 20 hidden nodes per layer, this wider neural network is useful in this instance where the problem is not too difficult. Although we see in research that a very wide and shallow network such as Full3Net are very good at memorization but not so good at generalization. For this problem of fractal classification and no test set, this shallow and wide network performs well enough. The wider network such as Full3Net will also have a tendency to overfit the data, but again this is not a problem for this particular task where there is no test data.

In comparison with Full4Net, adding an extra hidden layer allows a narrower neural network with 17 hidden nodes per layer. The advantage of this extra hidden layer adds an extra layer of non-linearity and allows the network to learn a more complex function. Having this deeper model instead of the wider Full3Net model also allows the network to generalize much better than the Full3Net wider network. As we can see from the results table above, the Full4Net deeper model contains many more parameters than Full3Net and thus the deeper model becomes more computationally expensive and takes longer to train. Additionally, going deeper allows the

models to capture richer structures that the wider model may not capture (*The Power of Depth for Feedforward Neural Networks, R. Eldon and O.Shamir, 2016, https://arxiv.org/pdf/1512.03965v4.pdf*).

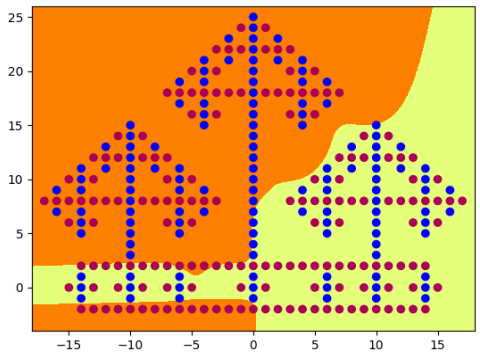
The best performing model in terms of minimum number of parameters and time to train is the DenseNet. Although the network only uses 2 hidden layers, the skip connections (shortcut connections) reduce the number of nodes required per layer and thus the total number of independent parameters for the network. This allows the network to be trained relatively much quicker than the other models, achieving an accuracy of 100 in only 94.58 seconds. Referring to *"Visualizing the Loss Landscape of Neural Networks" paper by H. Li et al. (https://arxiv.org/abs/1712.09913)* which provides visualizations of the loss surface of neural networks and shows how skip connections have a smoother loss surface, networks with skip connections lead to faster convergence than the network without any skip connections (Full3Net and Full4Net). Hence, we see that even with less total parameters, the DenseNet learns the classification task quicker and with less hidden nodes required than the two other models.

We can examine the functions computed by the Full4Net and DenseNet through the plots of the hidden unit activations. Simply, hidden layers are used to capture more complexity in the data. More hidden layers capture more complexity with every layer discovering different relationships in the data. In the Full4Net, we have the first layer of the network applying a linear transformation to the input and then being squashed by a non-linear hyperbolic tangent activation function. We can see the linear transformation of the inputs outputted by the hidden unit activation nodes of the first hidden layer, for example the layer 1 node 2.

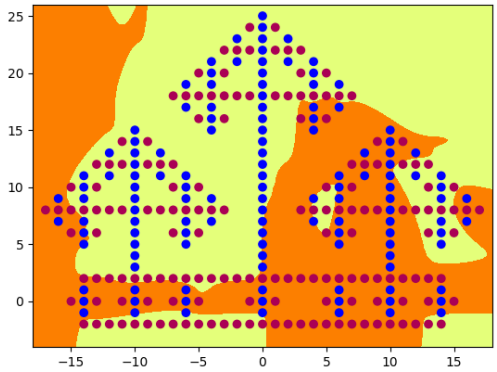


The first hidden layer of the Full4Net applies a linear transformation of the inputs and then applies the hyperbolic tangent activation function to squash these values. We can observe this for all outputs of all nodes of the first hidden layer of the Full4Net network. The second hidden layer again applies another transformation to all the hidden activation layer 1 outputs to alter the hidden unit space and extract more features from the input. We can see that the second hidden

layer nodes learn more of a non-linear function with more complexity, for example layer 2 node 4.

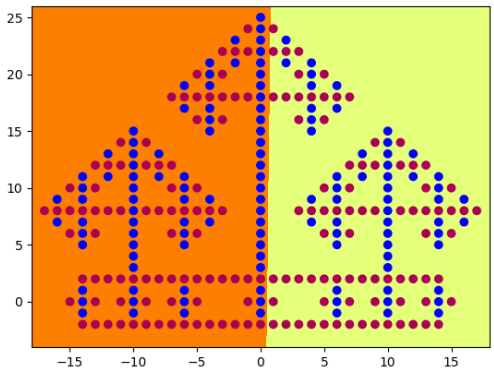


Finally, the third hidden layer further combines these outputs to extract more features and applies another linear function, transforming the hidden unit space once more. We can see this increased non-linearity function learned through the plot of the hidden activation nodes in the third layer, for example layer 3 node 13.

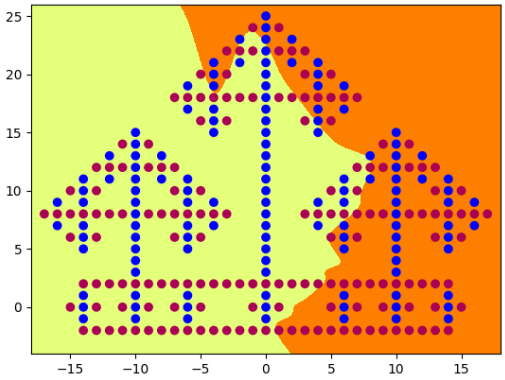


As we can see, the output plots of the hidden activation's nodes of Full4Net shows us the different complexity which increases as we apply further transformations to the hidden unit space in each hidden unit layer. We can therefore observe qualitatively the different functions computed by the different hidden layers of Full4Net through these plots and the increasingly non-linear function output computed by the network in each layer.

For the DenseNet we see that the first layer is a linear transformation of the input space and similarly to the Full4Net applies the hyperbolic tangent function to squash these values. Hence, similarly to Full4Net we see that the function computed by the first hidden layer is a linear function (transformation) of the input vector, for example in layer 1 node 2.

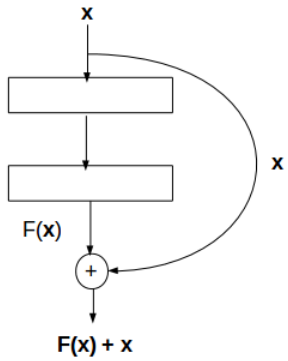


Different to Full4Net and Full3Net, the DenseNet computes a different function in the second hidden layer. The shortcut connection from the input layer to the second hidden layer make it easier for the gradient to flow from output layers to layers nearer the input. This shortcut connections shows that the function computed by the second hidden layer of DenseNet performs a linear transformation of the input vectors by a matrix W_1 with the outputs of the first hidden layer by another matrix W_2 and a bias term, then similarly applies a non-linear hyperbolic tangent function. We can observe the non-linear function output computed by the second hidden layer in DenseNet in for example the layer 2 node 15.



Finally, the final output layer applies a linear transformation of the inputs via a matrix W_1, with the outputs of the first hidden layer via a matrix W_2, the outputs of the second hidden layer via a matrix W_3 and finally a bias term. We can see that the functions computed by the DenseNet is similar to the Full4Net and Full3Net for the first hidden layer but is different for the second hidden layer and the output layer. The difference comes from the short cut connections and the similarity comes in that there is still being a linear transformation applied through matrix W, with a non-linear function being applied using the hyperbolic tangent function. To see the difference between the overall function computed by the DenseNet and the other networks, we

can refer to the figure below which shows how the shortcut is applied from the input to the output layer.



We can see that in the Full4Net and the Full3Net we would pass the learned mapping of *F(x)* into the output node and then to the sigmoid activation function. But, in the DenseNet we obtain a different mapping of *H(x)* which is equal to *F(x) + x*, which is then passed to the output node.

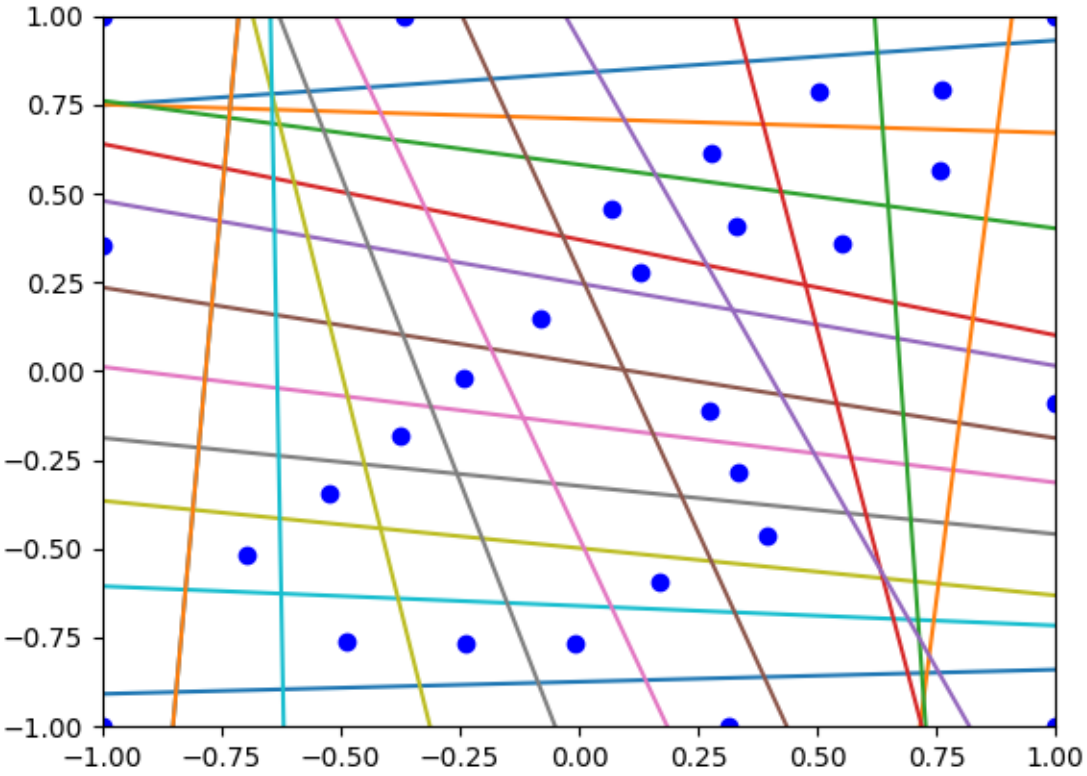
*To further optimize and try to find the minimum hidden nodes required for each network, we have collected the data of each network and presented them below. We will examine how the Full3Net performs with different learning rates and initial weight values.

Model	Number of hidden nodes	Learning Rate	Initial Weight	Number of epochs	Accuracy achieved in number of epochs	Time (seconds)
Full3Net	20	0.006	0.15	184,500	100	132.73
Full3Net	20	0.005	0.15	161,400	100	121.56
Full3Net	20	0.004	0.15	200,000	98.63	156.02
Full3Net	19	0.006	0.15	200,000	99.32	150.02
Full3Net	20	0.005	0.14	200,000	98.29	162.98
Full3Net	20	0.005	0.16	200,000	99.66	153.16

As we can see, with the Full3Net, we have tried to find the minimum number of hidden nodes required to achieve an accuracy of 100. Note that many times the network would achieve an accuracy of 99.32 or close to 100 but would not be able to achieve an accuracy of 100. For

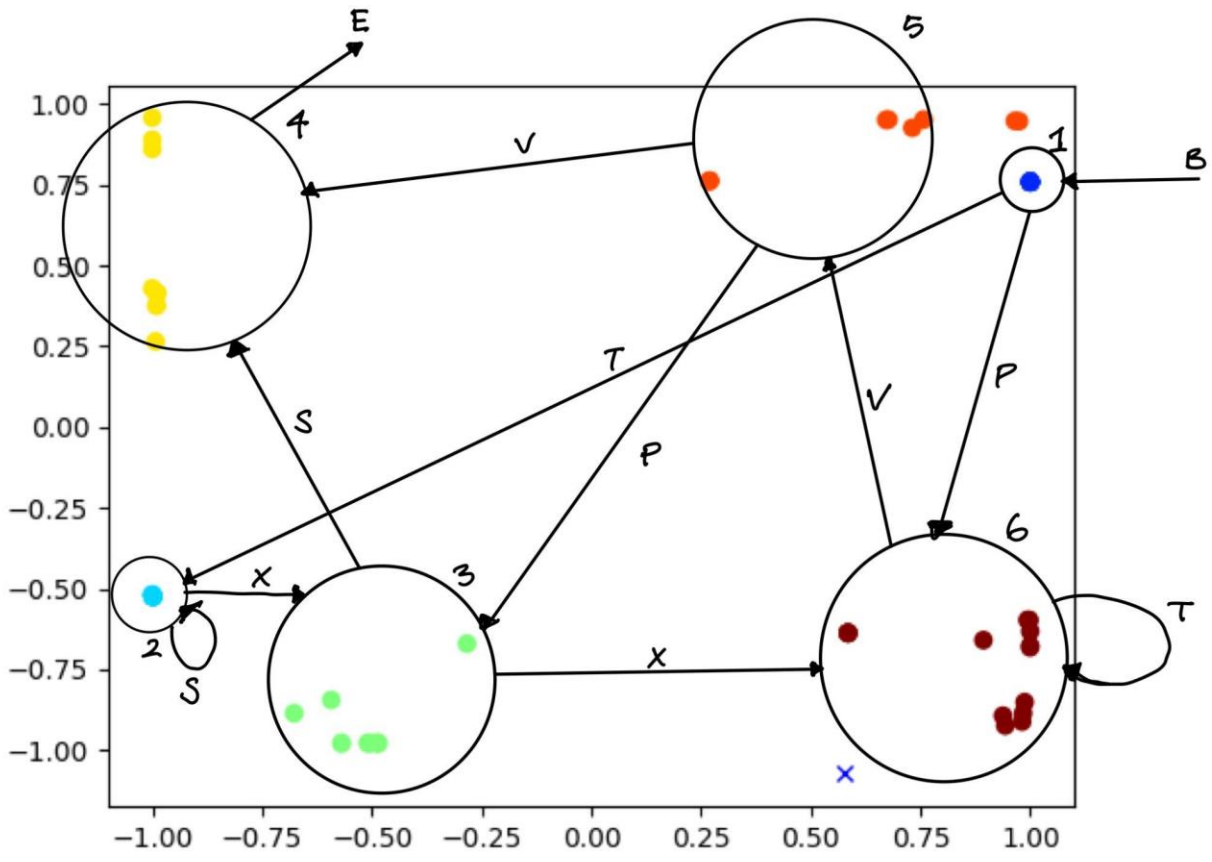
example, using only 19 hidden nodes per layer with learning rate of 0.006 and initial weight of 0.15 achieved a 99.32 accuracy within epoch 104,500 but was unable to reach an accuracy of 100. Similar analysis was done for the Full4Net and DenseNet networks to determine the optimal hidden nodes required for each network.

Part 2: Encoder Networks

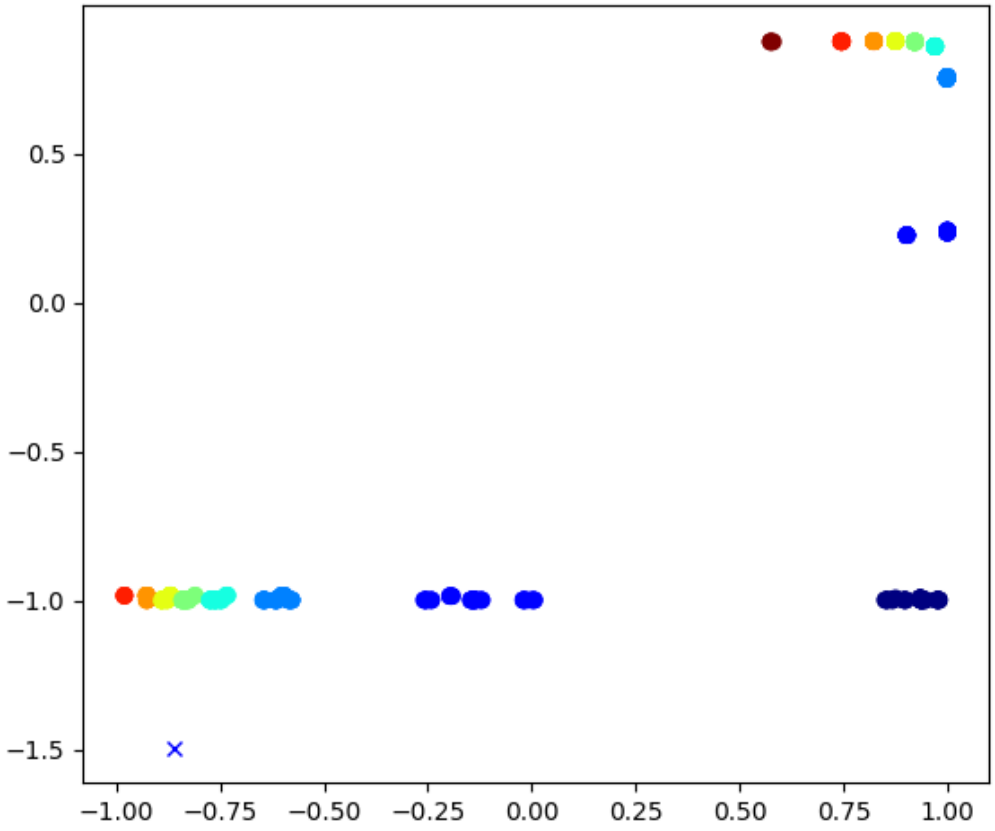


Part 3: Hidden Unit Dynamics for Recurrent Networks

Q1)



Q2)



Q3)

We can examine how the $a^n b^n$ language prediction task is achieved by the network by looking at the output of hidden unit activations in Q2, and by looking at how the output probabilities change as the string is processed. To see how our model performs we will look at the hidden unit activations and output probabilities of a particular $a^n b^n$ sequence. Without loss of generality, the sequence that we pick is AAAABBBBAABBAAAAAAABBBBBBBAABBAAAAAAABBBBBBB. The hidden activation and output probabilities given by our model (with error 0.0102) for this sequence is:

```
color = 012343210121012345676543210121012345676543210
symbol= AAAABBBBAABBAAAAAAABBBBBBBAABBAAAAAAABBBBBBBA
label = 000011110011000000011111110011000000011111110
hidden activations and output probabilities:
A [0.9  0.23] [0.85 0.15]
A [1.   0.75] [0.87 0.13]
A [0.97 0.86] [0.83 0.17]
B [0.92 0.88] [0.76 0.24]
B [-0.73 -0.98] [0. 1.]
B [-0.58 -1.  ] [0. 1.]
B [ 0. -1.] [0.02 0.98]
A [ 0.98 -1.  ] [0.98 0.02]
A [1.   0.24] [0.93 0.07]
B [1.   0.76] [0.87 0.13]
B [-0.2  -0.99] [0. 1.]
A [ 0.9 -1. ] [0.96 0.04]
A [1.   0.24] [0.93 0.07]
A [1.   0.76] [0.87 0.13]
A [0.97 0.86] [0.83 0.17]
A [0.92 0.88] [0.76 0.24]
A [0.88 0.88] [0.69 0.31]
A [0.82 0.88] [0.59 0.41]
B [0.74 0.88] [0.43 0.57]
B [-0.93 -0.98] [0. 1.]
B [-0.89 -1.  ] [0. 1.]
B [-0.84 -1.  ] [0. 1.]
B [-0.77 -1.  ] [0. 1.]
B [-0.65 -1.  ] [0. 1.]
B [-0.25 -1.  ] [0. 1.]
A [ 0.85 -1.  ] [0.94 0.06]
A [1.   0.24] [0.93 0.07]
B [1.   0.76] [0.87 0.13]
B [-0.19 -0.99] [0. 1.]
A [ 0.9 -1. ] [0.96 0.04]
A [1.   0.24] [0.93 0.07]
A [1.   0.76] [0.87 0.13]
A [0.97 0.86] [0.83 0.17]
A [0.92 0.88] [0.76 0.24]
A [0.88 0.88] [0.69 0.31]
A [0.82 0.88] [0.59 0.41]
B [0.74 0.88] [0.43 0.57]
B [-0.93 -0.98] [0. 1.]
B [-0.89 -1.  ] [0. 1.]
B [-0.84 -1.  ] [0. 1.]
B [-0.77 -1.  ] [0. 1.]
B [-0.65 -1.  ] [0. 1.]
B [-0.25 -1.  ] [0. 1.]
A [ 0.85 -1.  ] [0.94 0.06]
```

Now, we can see that as the string is processed, we expect that after the first A, all the A's after that are deterministic and can only be predicted in a probabilistic sense. We can observe this in the first 3 A's that are processed and see that the hidden unit activations are all relatively positioned at the top right of the grid. We can also observe that the probability of an A appearing reduces as we process more A's. We see this trend in the other A's that are processed as well. We also know that the first B in each sequence is not deterministic. We can observe this as the hidden activation for the first B is also in the top right and predicts that there will be an A rather than a B. We can see this for all the first B's in each sequence.

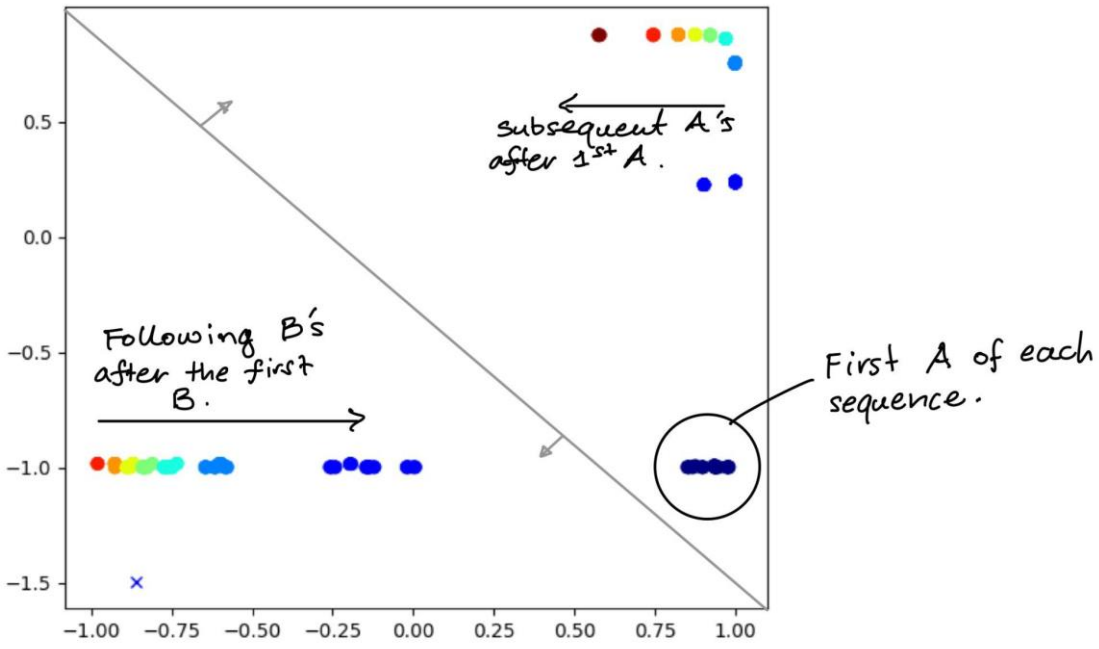
The moment that the hidden unit activations drastically changes is when it predicts the next B after the first B in each sequence (assuming a sequence of more than 1 B). We can observe this in the first sequence of the example where the hidden unit activation of the first B is [0.92, 0.88] with a prediction of [0.76, 0.24], compared to the hidden unit activation of the second B which is [-0.73, -0.98] with a prediction of [0.0, 1.0]. Here, we see that the hidden unit dynamics drastically changes and moves to the bottom left of the plot from the top right cluster, as well as predicting a B with near certainty. Continuing to look at this sequence, we see that each B after the first B in the sequence has hidden unit activations near the bottom left of the plot and with almost certain probability of expecting a B in the sequence. We also see the hidden unit activations of each B after the first B starts to move towards the blue cluster at the bottom middle of the plot. As the hidden unit activation's move towards this cluster from the bottom left to the bottom middle blue cluster, we always see a certainty in its prediction of a B appearing.

To summaries, we can observe that the model has learnt to correctly predict every B after the first B in each sequence (assuming a sequence greater than 1). And we have also seen that the first B cannot be predicted and is almost always predicted to be an A. An interesting observation though, is that as the sequence of A's gets larger, the hidden unit activation's move more towards the top center of the plot and outputs an increase in the probability of predicting a B. We can observe this in the last sequence of the example given above.

Now that we know how the network is able to correctly predict all B's except for the first B in each sequence through how the hidden unit dynamics changes as the sequence is processed, we want to also see how the network is able to predict the following A of each sequence as well. These are the two "expectations" for the network to learn the $a^n b^n$ language. We can observe the hidden unit activation of the first A in our example above. We see that the last B of the first sequence has a hidden unit activation value of [0.0, -1.0] (blue cluster at the middle of the plot) with an output probability of [0.02, 0.98], thus the network certainly predicts the last B of each sequence. We then see a significant change in the hidden unit activations when the string processes the first A of the sequence. The first A of the sequence has a hidden activation of [0.98, -1] (dark blue cluster at the bottom right of the plot) with an output prediction of [0.98,

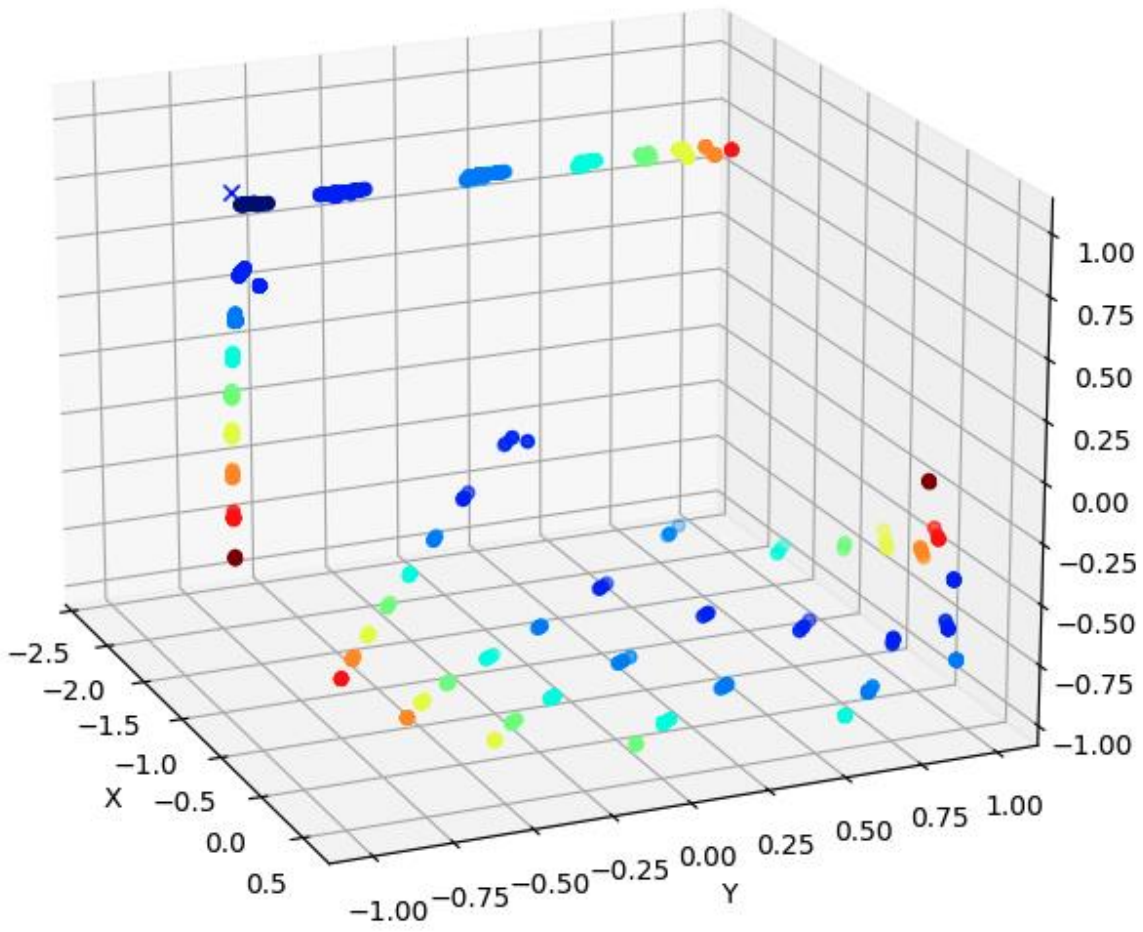
0.02], thus the network predicts with very high probability that there will be an A. We see this for each sequence that the first A is always predicted with almost certainty in every sequence. We see that for each of these predictions, the hidden activation lies on the bottom right cluster of dark blue values in the plot given in Q2. Although after this first A, the hidden unit activation moves upwards towards the top right of the plot, and then slowly left again as the sequence of A's gets larger, and the network increases its prediction of seeing a B appear next in the sequence.

In conclusion, we see that the network has achieved its prediction of the $a^n b^n$ language by being able to predict every B except for the first B with close certainty and the first A of every series with close certainty. We see that as the sequence is processed the hidden unit dynamics change significantly for these two cases. We see that for every A except for the first A, the hidden unit activation's starts around the top right cluster and moves left towards the top middle of the plot. Here we also see that the probability of expecting a B increases as the A sequence length gets larger. We see that after the first B, the hidden unit activations of the second B lies at the bottom left of the plot and then as we predict more B's the hidden unit activations move right towards the blue cluster at the bottom center of the plot. Finally, we see that the first A of each sequence is predicted with near certainty each time and has a hidden unit activation in the dark blue cluster in the bottom right of the graph. To further demonstrate how the hidden unit activation's change as the string is processed, I have annotated the plot showing this.



The dividing gray line shows the boundary between the final states for which the next letter is classified either A or B, where above the line is classified as A (A region) and the bottom half is classified as B (B region).

Q4)



Q5)

Similarly, to Q3, we can examine how the $a^n b^n c^n$ language prediction task is achieved by the network by looking at how the output of the hidden unit activation's change as the string is processed. Without loss of generality, we can look at the hidden unit activations and output probabilities of a particular $a^n b^n c^n$ sequence. Below shows the hidden unit activations and output probabilities for the given sequence and the first few activation and output probabilities:

```
-----
color = 01101234543215432101234567654321765432101232132101234543215432 10
symbol= ABCAAAAABBBBBCCCCCAAAAAAABBBBBBBCCCCCCCAAABBBCCCAAAAABBBBBCCCCCA
label = 01200000111112222200000001111111222222200011122200000111112222 20
hidden activations and output probabilities:
B [-0.95 -0.88  0.67] [0.89 0.11 0.  ]
C [-0.84 -0.05 -0.08] [0.   0.04 0.96]
A [-0.98 -0.84  1.  ] [0.85 0.05 0.1 ]
A [-0.96 -0.92  0.75] [0.92 0.07 0.  ]
A [-0.95 -0.95  0.58] [0.89 0.11 0.  ]
A [-0.94 -0.97  0.42] [0.85 0.15 0.  ]
A [-0.93 -0.97  0.27] [0.78 0.22 0.  ]
B [-0.92 -0.98  0.12] [0.69 0.31 0.  ]
B [-0.72 -0.53 -0.63] [0.01 0.99 0.  ]
B [-0.48 -0.26 -0.82] [0. 1. 0.]
B [-0.26  0.12 -0.84] [0. 1. 0.]
B [-0.15  0.68 -0.76] [0.   0.88 0.12]
C [-0.22  0.95 -0.46] [0. 0. 1.]
C [-0.94  0.39  1.  ] [0. 0. 1.]
C [-0.99  0.22  1.  ] [0. 0. 1.]
C [-0.99 -0.08  1.  ] [0. 0. 1.]
C [-0.99 -0.54  1.  ] [0. 0. 1.]
A [-0.99 -0.88  1.  ] [0.91 0.04 0.05]
A [-0.96 -0.93  0.74] [0.93 0.07 0.  ]
A [-0.95 -0.96  0.56] [0.89 0.11 0.  ]
A [-0.94 -0.97  0.41] [0.84 0.16 0.  ]
A [-0.93 -0.98  0.26] [0.78 0.22 0.  ]
A [-0.91 -0.98  0.1 ] [0.69 0.31 0.  ]
A [-0.9  -0.98 -0.06] [0.57 0.43 0.  ]
B [-0.88 -0.99 -0.22] [0.44 0.56 0.  ]
B [-0.62 -0.68 -0.8 ] [0.01 0.99 0.  ]
B [-0.3  -0.54 -0.91] [0. 1. 0.]
B [ 0.04 -0.34 -0.92] [0. 1. 0.]
B [ 0.31  0.07 -0.92] [0. 1. 0.]
B [ 0.45  0.69 -0.87] [0. 1. 0.]
B [ 0.4   0.96 -0.67] [0.   0.99 0.01]
C [ 0.21  0.99 -0.35] [0.   0.07 0.93]
C [-0.87  0.58  1.  ] [0. 0. 1.]
C [-0.99  0.51  1.  ] [0. 0. 1.]
C [-1.   0.4  1. ] [0. 0. 1.]
C [-0.99  0.22  1.  ] [0. 0. 1.]
C [-0.99 -0.09  1.  ] [0. 0. 1.]
C [-0.99 -0.54  1.  ] [0. 0. 1.]
A [-0.99 -0.88  1.  ] [0.92 0.04 0.04]
A [-0.96 -0.93  0.74] [0.93 0.07 0.  ]
A [-0.95 -0.96  0.56] [0.89 0.11 0.  ]
B [-0.94 -0.97  0.41] [0.84 0.16 0.  ]
```

We will first examine how the hidden activation's change as the string is processed and how it is able to correctly predict all the B's after the first B of each sequence. We can see the hidden activation of the first A's in the sequence have hidden unit activations near the top left blue cluster. As we process more A's the hidden unit activation 1 (HU1, x-axis) and hidden unit activation 2 (HU2, y-axis) stay relatively the same but the hidden unit activation 3 (HU3, z-axis) decreases as the prediction of a B appearing gets larger. As we know, it is only probabilistic and unlikely to predict the first B, we can see this as the first B has hidden unit activation of [-0.92, -0.98, 0.12] and the output probability is [0.69, 0.31, 0.0] and therefore the network cannot correctly predict the first B. Although, we see a significant change in the hidden unit activation value for the second B in the sequence which is [-0.72, -0.53, -0.63] with an output probability of [0.01, 0.99, 0.0], indicating that the network has successfully learnt to predict the second B after the first B. We see that there is an increase in all the hidden unit activation but most significantly in the value of the HU3. We see this pattern continue as the string processes more B's and the network correctly predicts every B for this sequence (as you can see in the figure above). As the string is processed for the B's we see the HU activations move from the left-hand side of the plot to the dark blue cluster at the bottom of the plot. We can see this through an increase in the HU1 and HU2 and a decrease in HU3.

We also observe from the HU values that there is a significant increase in all the hidden values for the first C in the sequence as well as a correct prediction of [0.0, 0.0, 1.0] as expected for a correctly trained network like ours. The HU activation's have a significant increase for all HU values, this is indicated by the movement of the values from the dark blue cluster at the bottom towards the higher and more rightwards redder values at the middle right of the plot. We then see this cluster continue as the values jump for the second C in the sequence to the top right red cluster of the top of the plot. As the C's get processed with all correct predictions of [0.0, 0.0, 1.0], the HU values move towards the top left of the plot as we can see in the change of color from the red cluster to the dark blue cluster. We finally see that the first A in the sequence has HU activation of [-0.99, -0.88, 1.00] which is situated at the top left of the dark blue cluster. The prediction of this first A is [0.91, 0.04, 0.05] which is correct as expected from the trained network. As the string gets processed for A's again then we see the same change in the HU values continue. Hence, we see how the network has learned to correctly predict the last B in each sequence as well as all of the C's and the following A through looking at how the hidden unit activation's change as the string is processed. We can also observe that as the string is processed from the first A to the last C and then the first A for another sequence, we move in a circular/spiral motion around the hidden unit space.

Q6)

The Embedded Reber Grammar (ERG) requires long range dependency as the second letter is always the same as the second last letter. Therefore, to predict the second last letter we need to remember which transition (T or P) occurred after the initial B, and retain this information while it is processing the transitions associated with the Reber Grammar in order to predict the T or P occurring before the final E.

With a SRN network, the vanishing gradient problem becomes prevalent with long-term dependencies. In the ERG problem, this long-term dependency becomes very important to remember the second character to successfully predict the second last character of the sequence. The problem is essentially that gradients propagated over many stages tend to either vanish (most of the time) or explode. To solve this problem, LSTM were introduced with the ability to learn long range dependencies using a combination of forget, input and output gates as required for the ERG problem.

LSTM is able to solve the ERG problem by using the combination of forget, input and output gates to retain the knowledge of the initial T or P, and to preserve that knowledge by assigning high values (close to 1) such that we remember the context of if either T or P was picked. Likewise, we assign values close to 0 for the gates if we do not need to retain the information for later. We can compare the performance of the SRN and the LSTM model on the ERG task to see how the LSTM is able to learn the task relatively much better. For the SRN model we achieved the following output at epoch 50,000 and with 4 hidden layers.

```
_____
state =  0 1 10 11 12 13 14 17 18
symbol= BPBTXSEPE
label = 040132646
true probabilities:
     B    T    S    X    P    V    E
1 [ 0.   0.5  0.   0.   0.5  0.   0. ]
10 [ 1.  0.  0.  0.  0.  0.  0.]
11 [ 0.   0.5  0.   0.   0.5  0.   0. ]
12 [ 0.   0.   0.5  0.5  0.   0.   0. ]
13 [ 0.   0.   0.5  0.5  0.   0.   0. ]
14 [ 0.  0.  0.  0.  0.  0.  1.]
17 [ 0.  0.  0.  0.  1.  0.  0.]
18 [ 0.  0.  0.  0.  0.  0.  1.]
hidden activations and output probabilities [BTSXPVE]:
1 [ 0.98  1.   -0.96  1.  ] [ 0.    0.53  0.    0.    0.46  0.    0.  ]
10 [-0.91 -0.65  0.21  0.38] [ 1.  0.  0.  0.  0.  0.  0.]
11 [ 1.    0.91 -0.3   0.47] [ 0.    0.5   0.    0.    0.49  0.    0.  ]
12 [ 0.06 -0.07  0.71 -0.73] [ 0.    0.    0.45  0.54  0.    0.    0.  ]
13 [-0.1   0.99  0.25 -0.91] [ 0.    0.    0.57  0.43  0.    0.    0.  ]
14 [ 0.99 -0.14  0.91 -0.17] [ 0.    0.    0.    0.01  0.    0.    0.99]
17 [ 1.   -0.02 -1.    0.44] [ 0.    0.28  0.    0.    0.72  0.    0.  ]
18 [ 0.26 -0.82  0.73 -0.41] [ 0.    0.    0.01  0.01  0.    0.    0.98]
epoch: 50000
error: 0.0007
final: 0.0223
```

For the LSTM model we achieved the following output at epoch 50,000 and with 4 hidden layers.

```
-----
state =  0 1 2 3 4 4 4 5 6 9 18
symbol= BTBTSSXSETE
label = 01012232616
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. ]
4 [ 0.   0.   0.5  0.5  0.   0.   0. ]
4 [ 0.   0.   0.5  0.5  0.   0.   0. ]
4 [ 0.   0.   0.5  0.5  0.   0.   0. ]
5 [ 0.   0.   0.5  0.5  0.   0.   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  0.53  0.15 -0.63] [ 0.    0.47  0.    0.    0.53  0.    0.  ]
2 [-0.5  -0.41 -0.62  0.74] [ 1.  0.  0.  0.  0.  0.  0.]
3 [-0.72 -0.03  0.06 -0.67] [ 0.    0.48  0.    0.    0.51  0.01  0.  ]
4 [-0.22 -0.41  0.9   0.68] [ 0.    0.01  0.47  0.52  0.    0.    0.  ]
4 [-0.19 -0.12  0.6   0.67] [ 0.    0.01  0.47  0.52  0.    0.    0.  ]
4 [-0.24 -0.05  0.64  0.67] [ 0.    0.01  0.46  0.53  0.    0.    0.  ]
5 [ 0.64 -0.54  0.93  0.33] [ 0.    0.01  0.57  0.41  0.    0.02  0.  ]
6 [ 0.85  0.62  0.66 -0.19] [ 0.    0.    0.    0.    0.    0.    0.99]
9 [ 0.02  0.51  0.87 -0.76] [ 0.    0.98  0.    0.    0.01  0.01  0.  ]
18 [ 0.83  0.78  0.42  0.74] [ 0.  0.  0.  0.  0.  0.  1.]
epoch: 50000
error: 0.0006
final: 0.0001
```

The LSTM model clearly performs much better in the ERG task as the final error outputted is 0.0001 in comparison to the SRN which outputted a final error of 0.0233. Furthermore, we are most interested in seeing how well each network predicted the second last character because the network which is able to predict the second last character better shows that it has better long-range dependency. With the SRN we have a prediction of 0.72 for the character to be P when the second character is P. With the LSTM we have a prediction of 0.98 for the character to be T when the second character that appeared was a T. Therefore, the LSTM has performed significantly better than the SRN in its ability to learn the task and specifically correctly predict the second last character.

To analyse the behavior of how the LSTM is able to successfully learn this task and retain information much better than the SRN model we can look at the LSTM cell state in which the information flows through the network compared to the SRN. In LSTM, cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. The gates are calculated through different equations which produce values between 0 and 1 via a sigmoid unit, for example the forget gate unit $f_i^{(t)}$ (for time step t and cell i) is calculated using the following equation:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

Where $x^{(t)}$ is the current input vector and $h^{(t)}$ is the current hidden layer vector, and b^f, U^f, W^f are respectively biases, input weights and recurrent weights for the forget gates.

In the code, we get that the forget gate f_t is stored in the variable *f_t* which is then used with the input gate i_t and update gate g_t to calculate the context state c_t as:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot g_t$$

We can see this in the code represented by line 91 in seq_models.py:

$$c\_t = f\_t * c\_t + i\_t * g\_t$$

Unfortunately, I was unable to print out the context layer values to further analyse the behavior of the LSTM model, but I can assume that we will find that the context layer would output values to indicate the network to "remember" the value of the second character with a high value of the forget gate.