CS6910: Fundamentals of Deep Learning
Assignment 1, February 2020
Team 3

S Renganathan, CH16B058 S Nithya, CH16B113 Vasistha Singhal, CH16B119

Question 1: Function approximation task

Input data

The first two columns are extracted as the features and the last column is extracted as the target output.

Choice of Architecture

The following models were tried out in increasing order of complexity along with two different learning rates (10^{-4} and 10^{-5}) and the accuracy in the validation set was compared for model selection:

(2, 2), (2, 3), (3, 2), (3, 3), (2, 4), (4, 2), (3, 4), (4, 3), (4, 4)

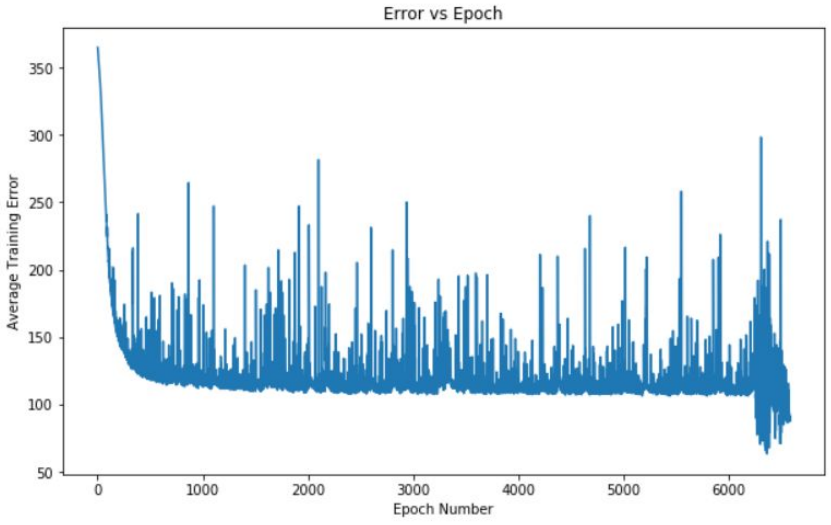
Model (3,3) with a learning rate of 10^{-4} was selected.

Modeling Challenges

The model faces the following challenges:

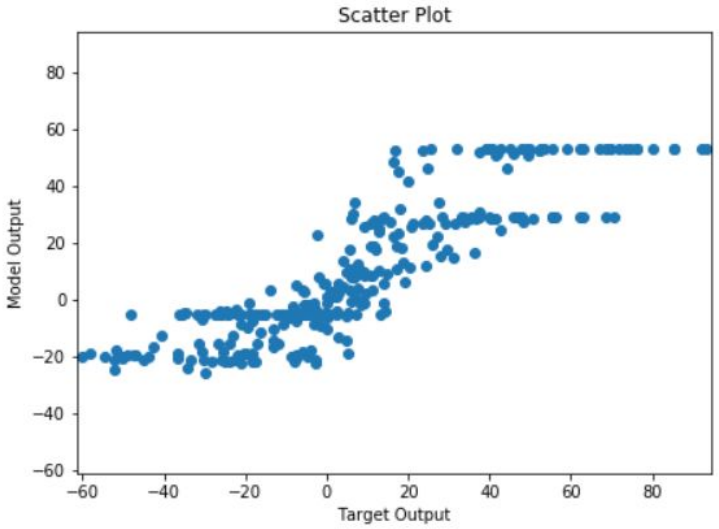
- Low Dataset size - 100 points. Thus, the model overfits if the number of parameters is high.
- Negative weight initialisation - since the function can output negative values as well, there need to be some negative weights initially too, otherwise, the model that is built will only predict positive values.

Plot of Average Error on training data vs Epoch:



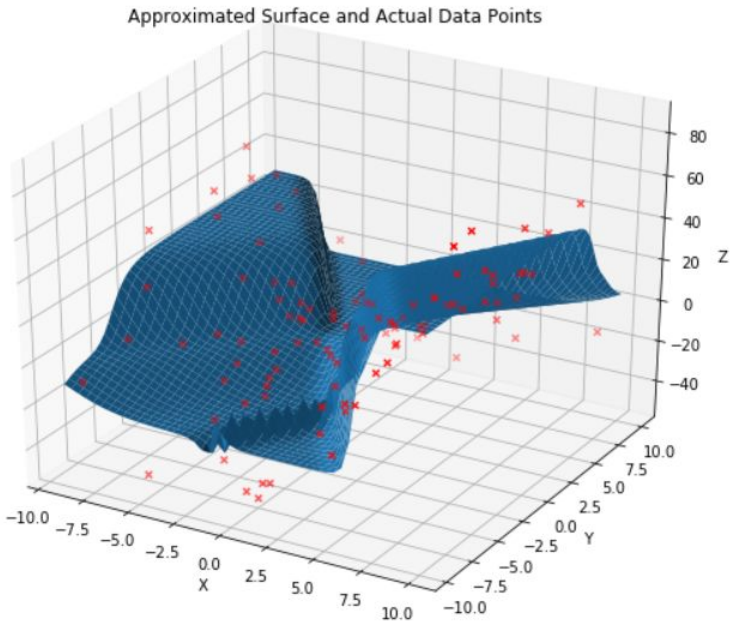
We see the expected trend of a **decrease in average training** error as more epochs are carried out. The slope of the error curve reaches a saturation after 2000 epochs, but due to the **stopping condition expecting a lower tolerable error**, the learning continues. We get another saturation at 7500 epochs with a lower error and the learning terminates. This type of wiggly error surface is **characteristic of pattern mode** where **every data point causes a weight update**.

Scatter Plot:



The model output has a **smaller range** than the target output. This could be because of the extreme target output values being **outliers** or a **imperfect choice of** β (activation function parameter). In the non-extremities, the scatter plot has a **majority of the points close to the y=x line**, and hence we can conclude that a satisfactory model has been built in this region.

Desired vs Approximated Function:



It is observed that the training data points lie very close to the approximated surface. The training points are the **farthest away from the surface in the extremities** (for e.g X=-10,Y=10). On the other hand, the **function surface is very close** to the points in the **non-extremities**, where most of the data lies. There seems to be an **inflection** near the region where X=0 and Y=0 intersect, and the **model is able to capture** this well as seen from the surface plot.

Question 2: Classification task for 2-D data

Input data

The weights are initially trained on the training dataset = 80% of the given dataset.

Choice of Architecture

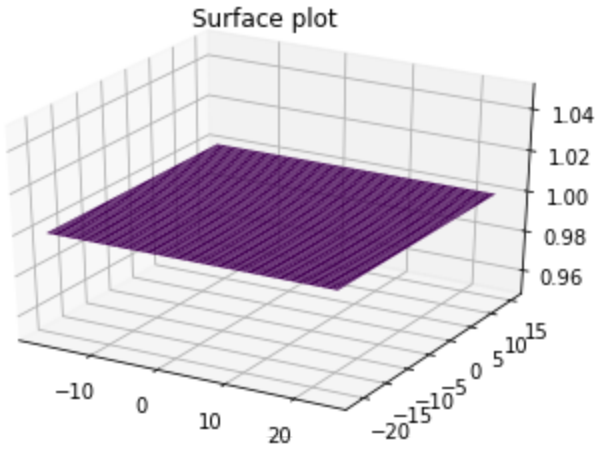
The following models were tried out in increasing order of complexity and the accuracy in the validation set was compared for model selection.

(2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9,9), (10,10), (11,11).

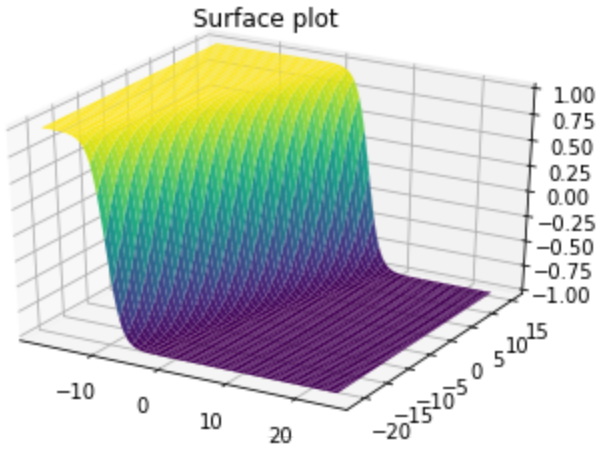
Model (10,10) was selected.

Surface Plots

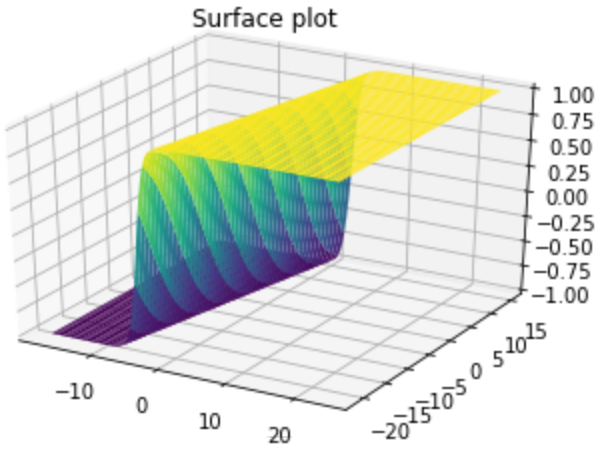
Epoch 1, Hidden Layer 1



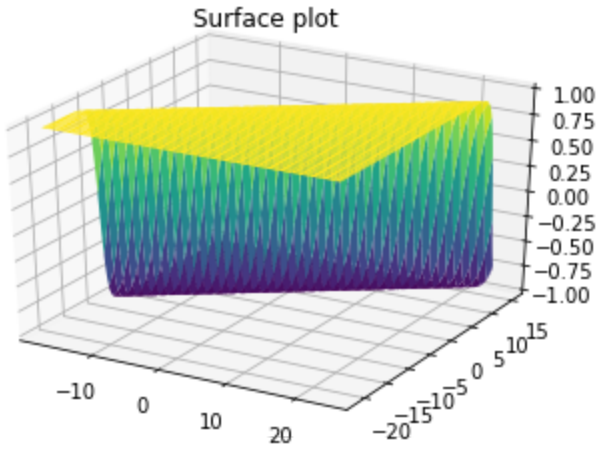
Node 1



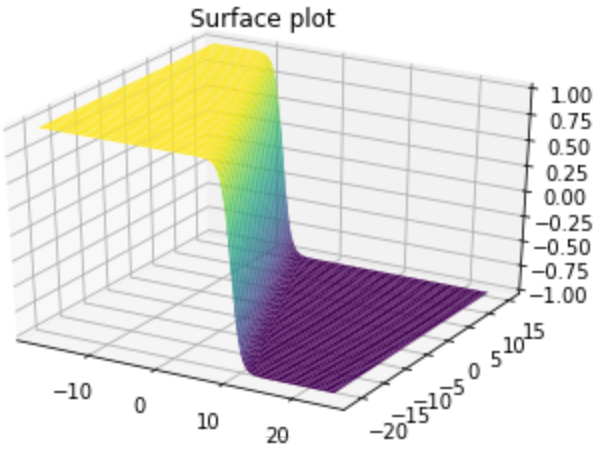
Node 2



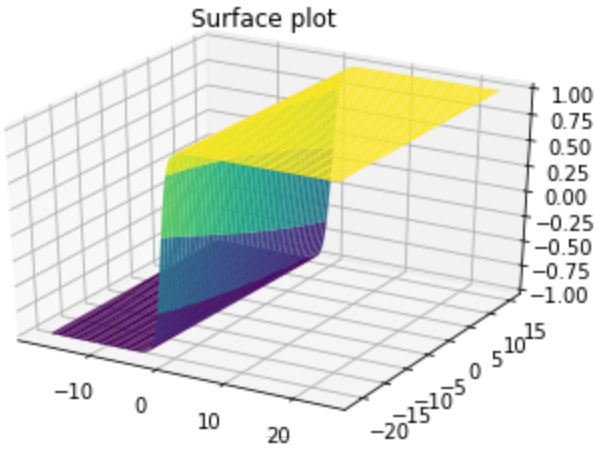
Node 3



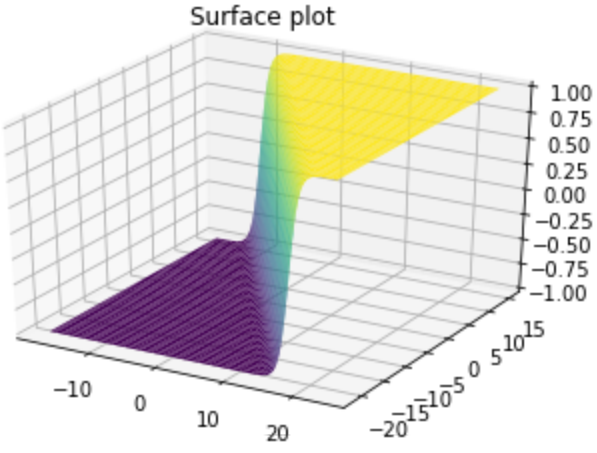
Node 4



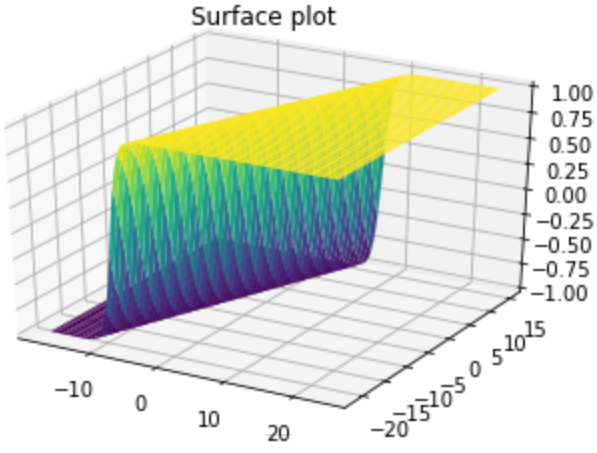
Node 5



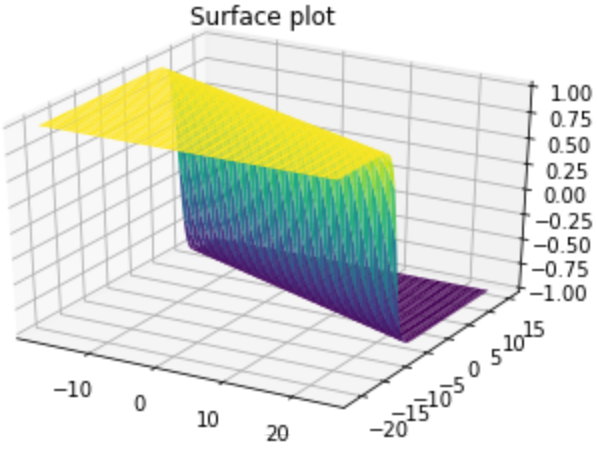
Node 6



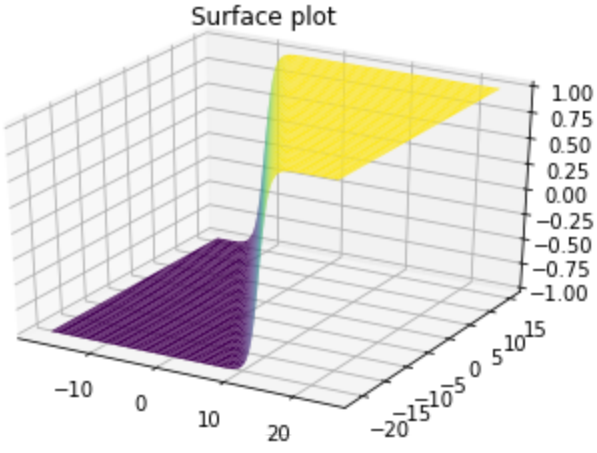
Node 7



Node 8

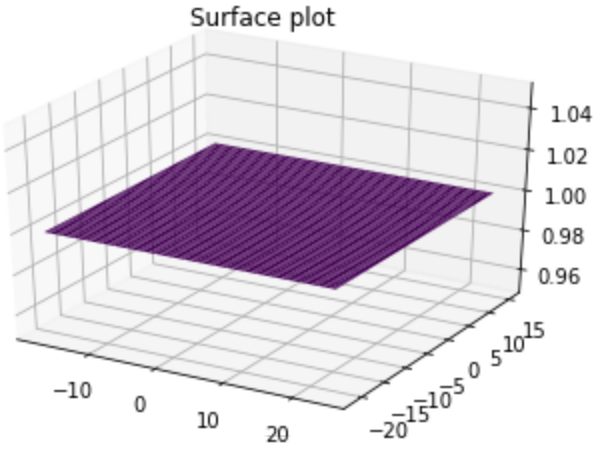


Node 9

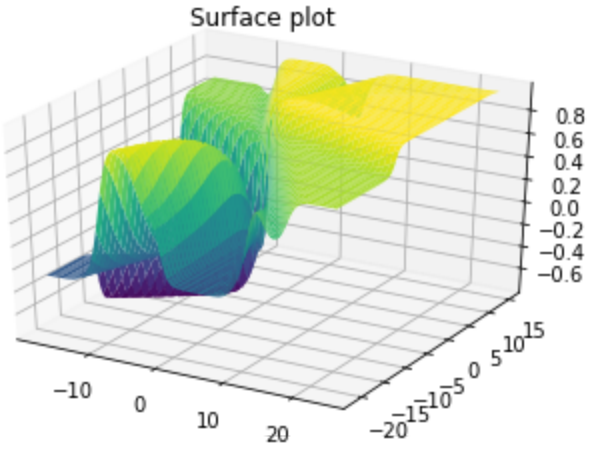


Node 10

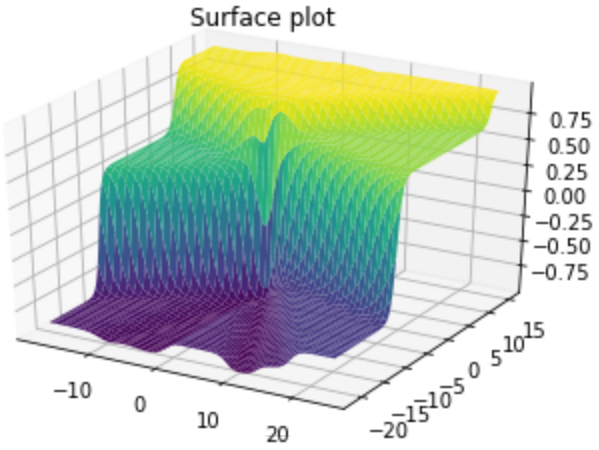
Epoch 1, Hidden Layer 2



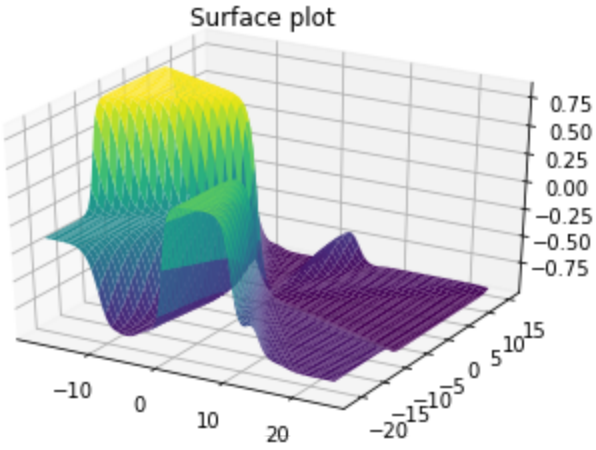
Node 1



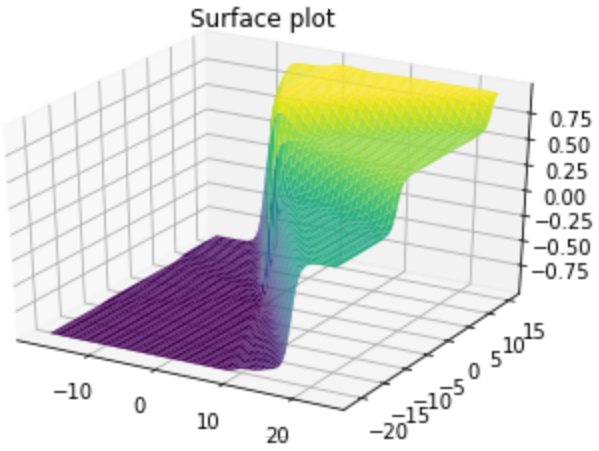
Node 2



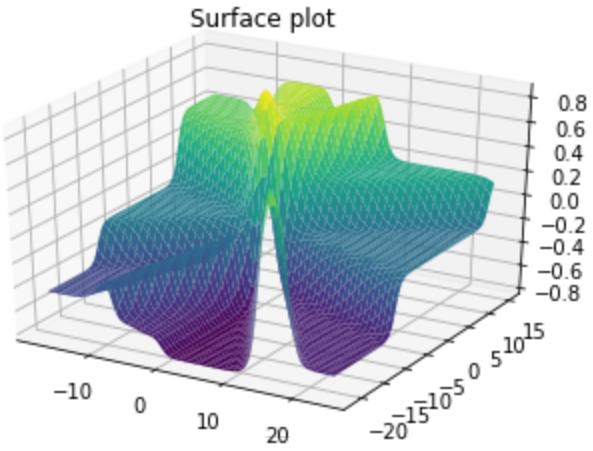
Node 3



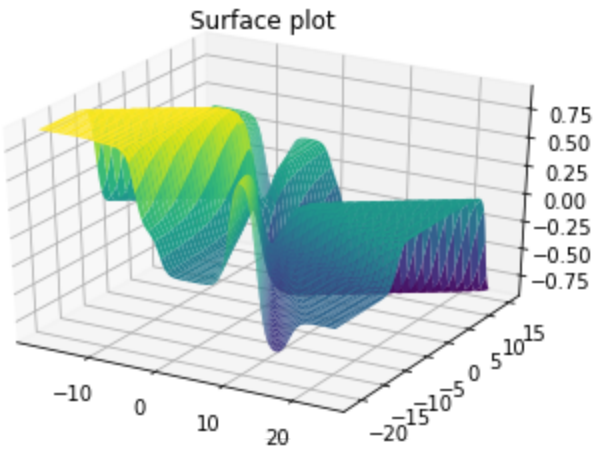
Node 4



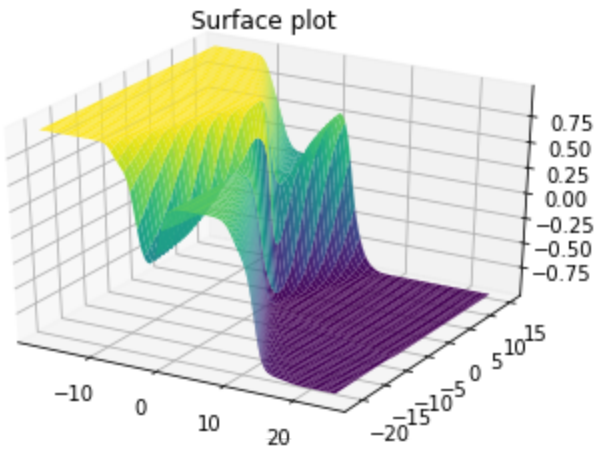
Node 5



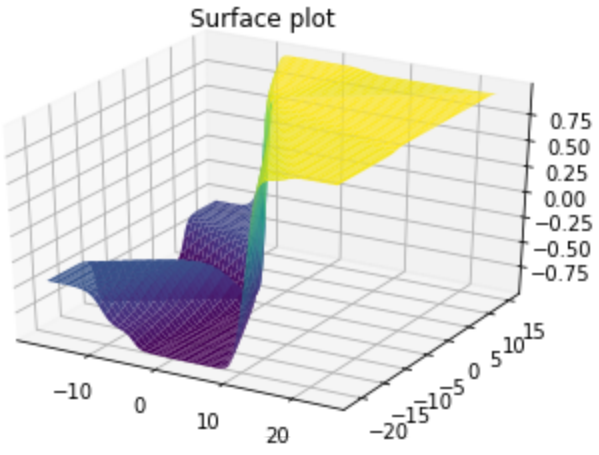
Node 6



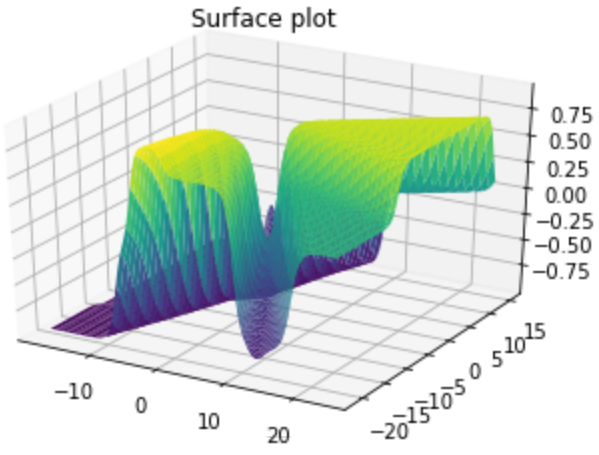
Node 7



Node 8

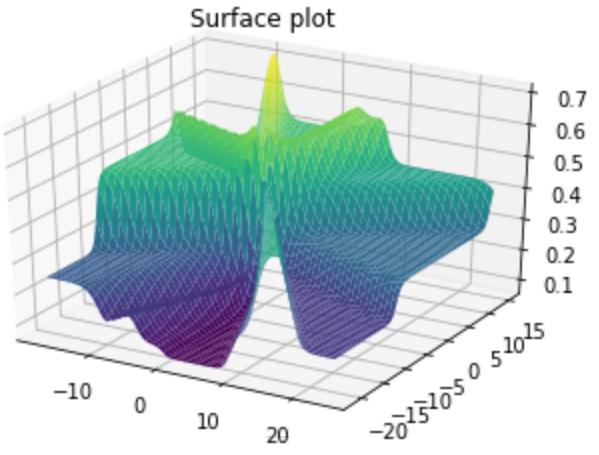


Node 9

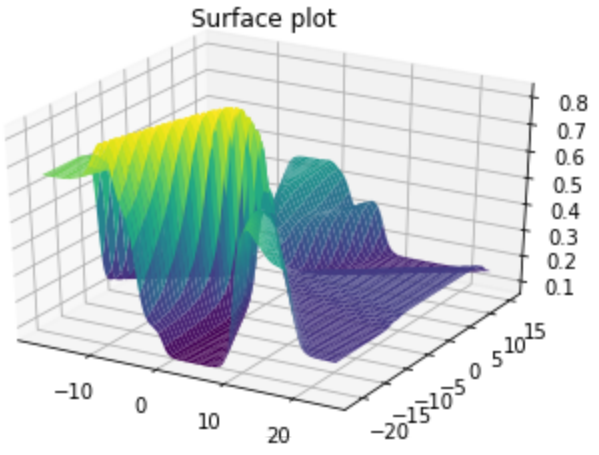


Node 10

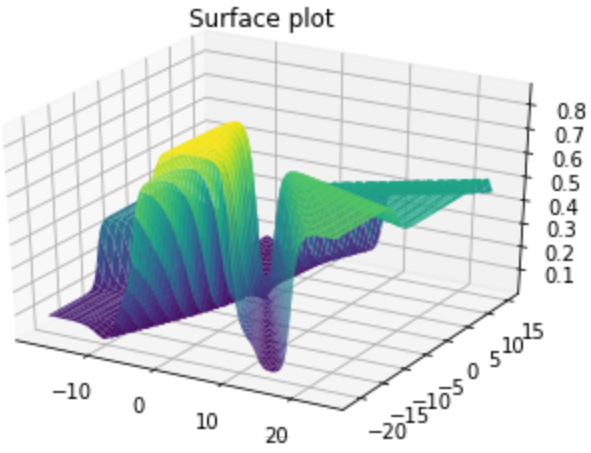
Epoch 1, Output layer



Node 1

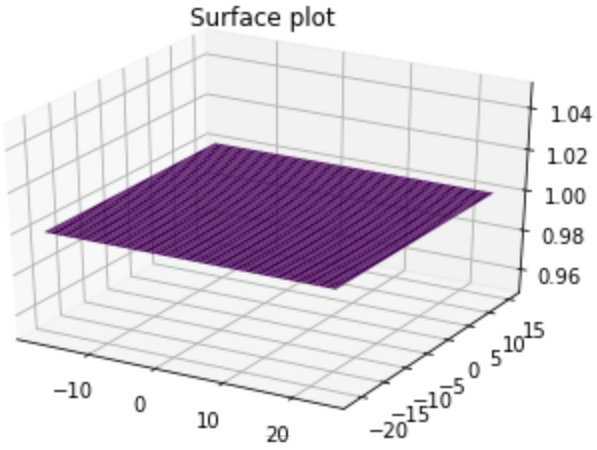


Node 2

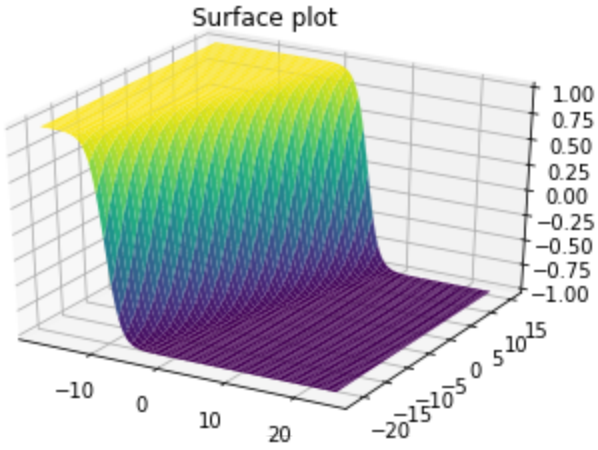


Node 3

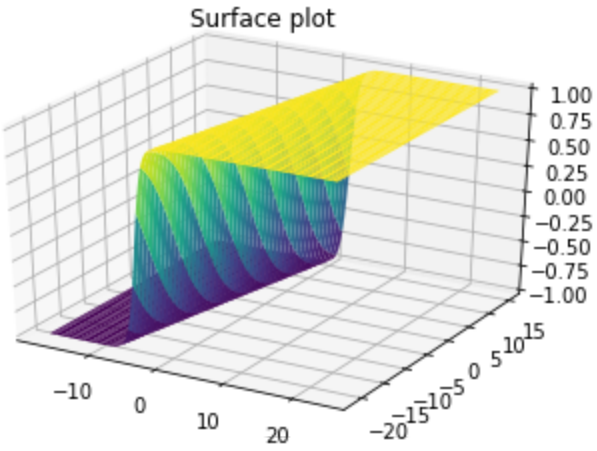
Epoch 2, Hidden Layer 1



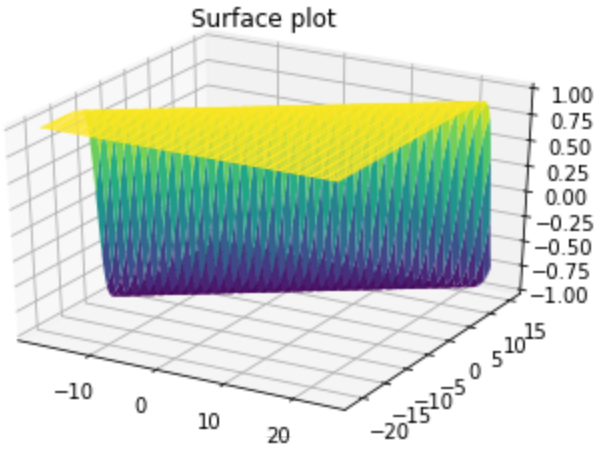
Node 1



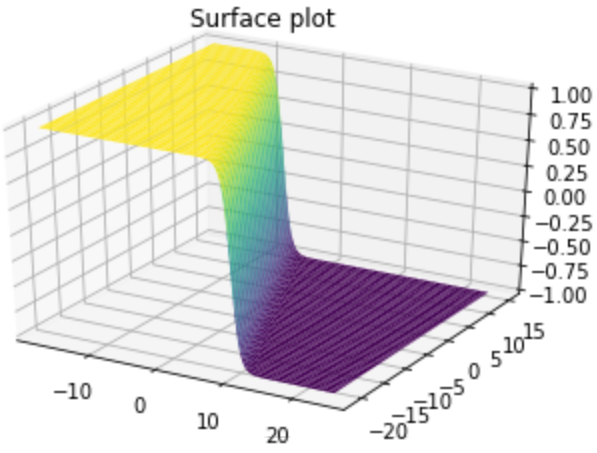
Node 2



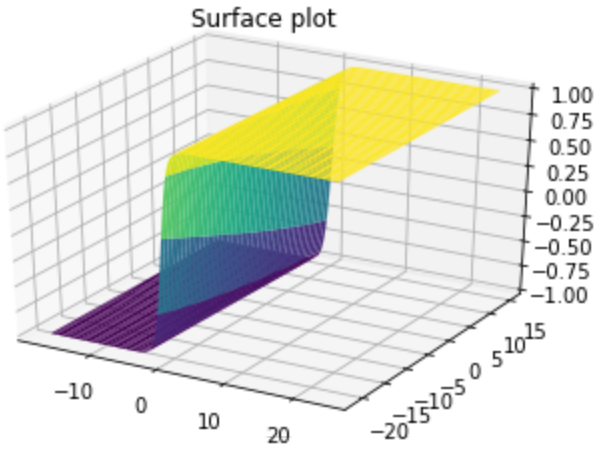
Node 3



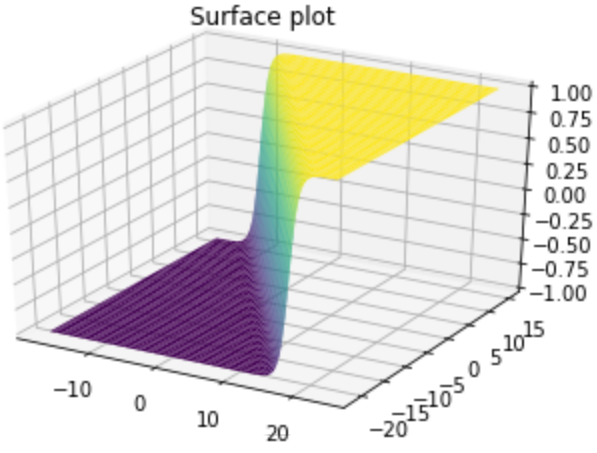
Node 4



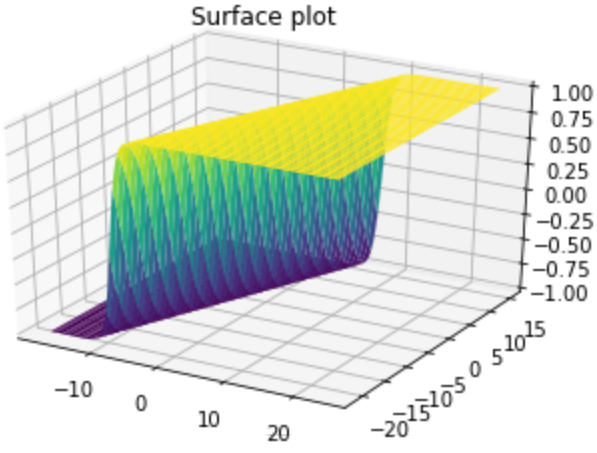
Node 5



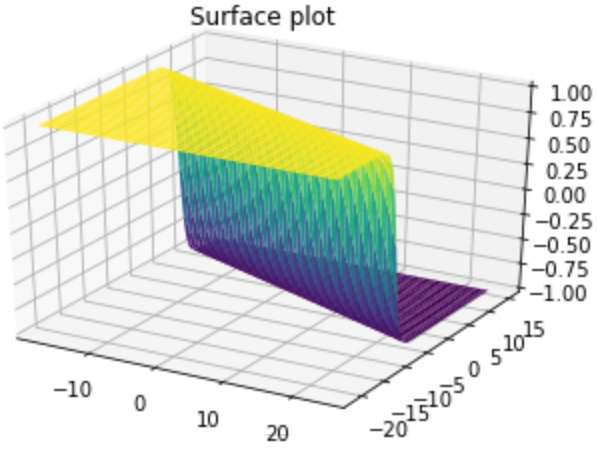
Node 6



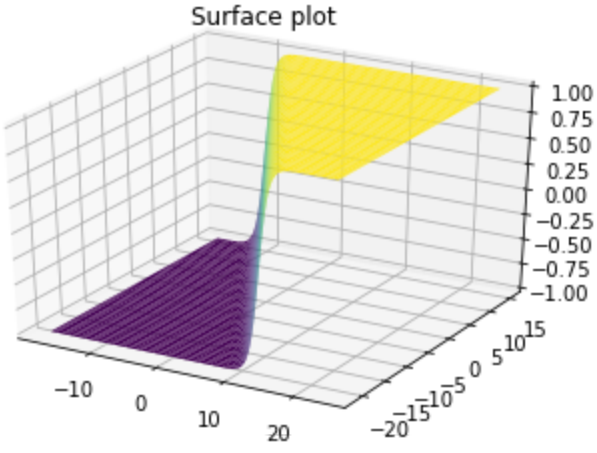
Node 7



Node 8

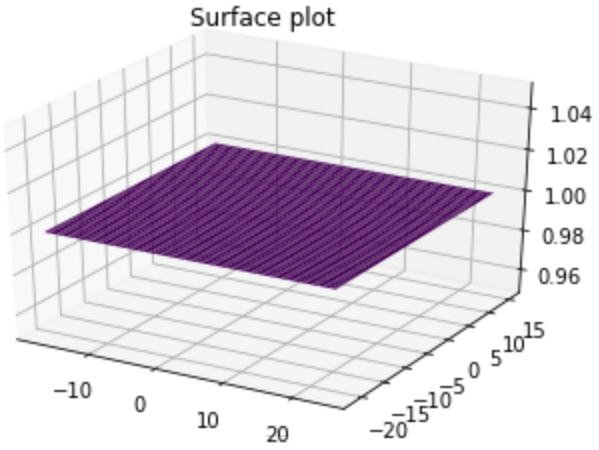


Node 9

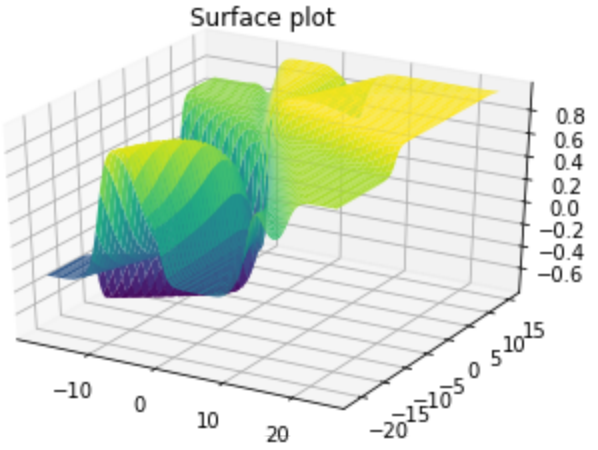


Node 10

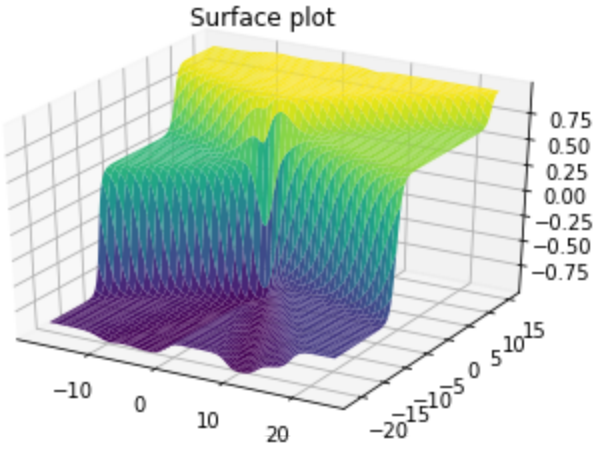
Epoch 2, Hidden Layer 2



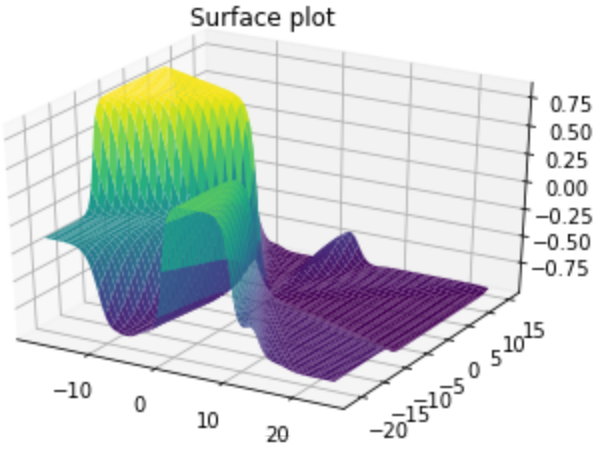
Node 1



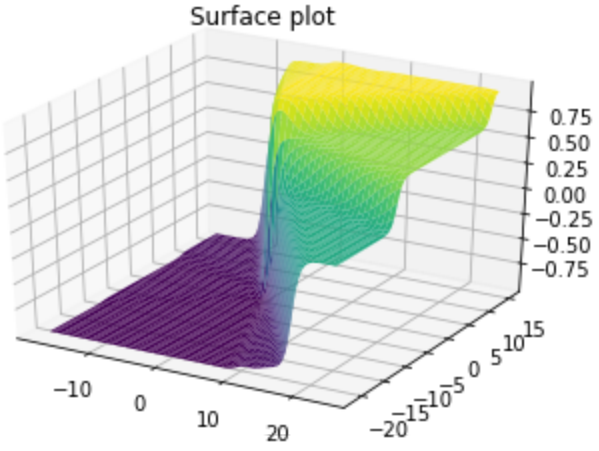
Node 2



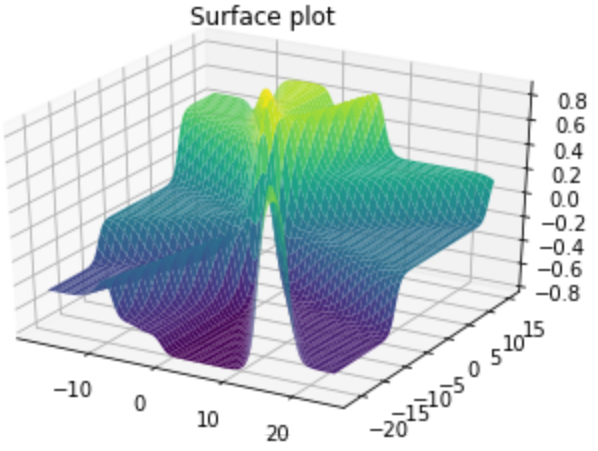
Node 3



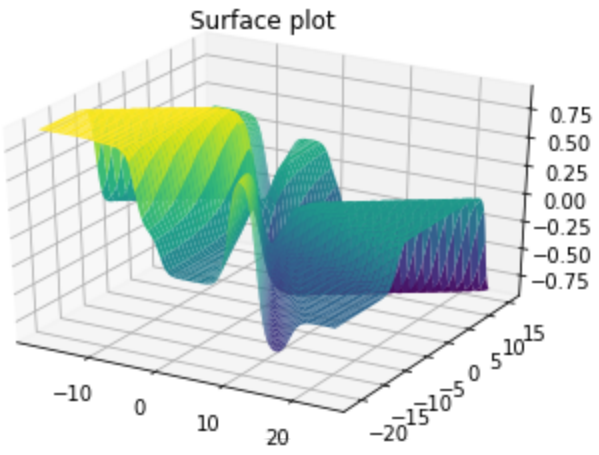
Node 4



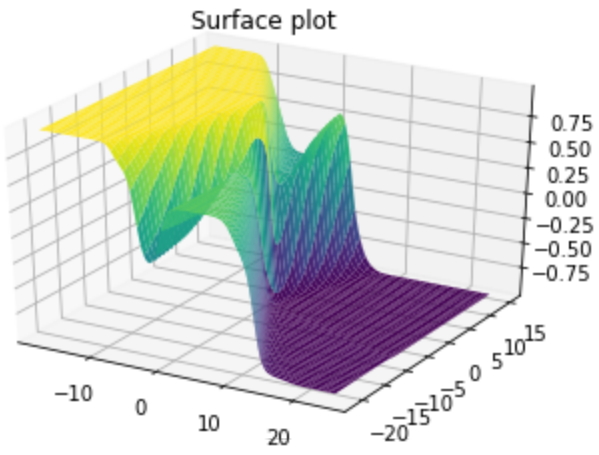
Node 5



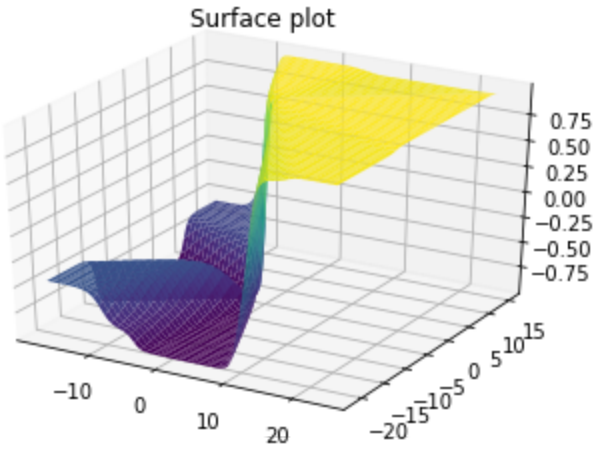
Node 6



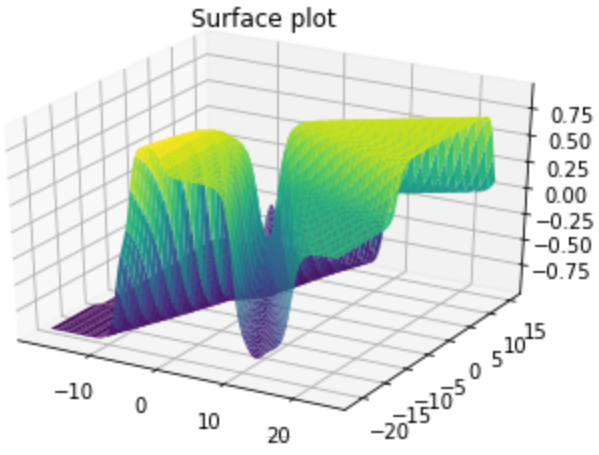
Node 7



Node 8

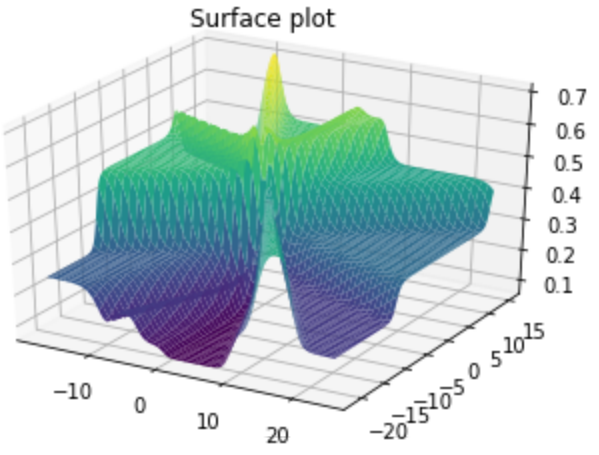


Node 9

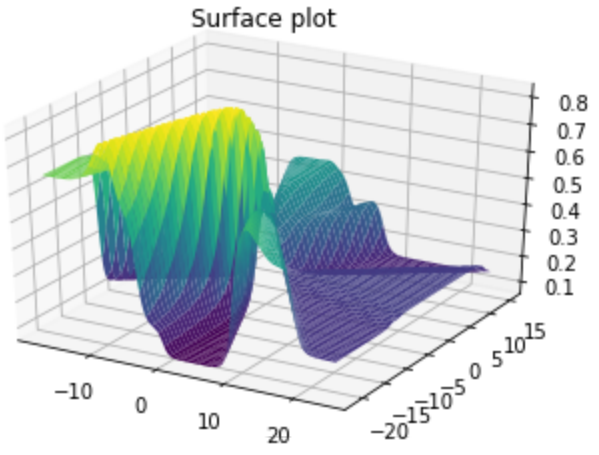


Node 10

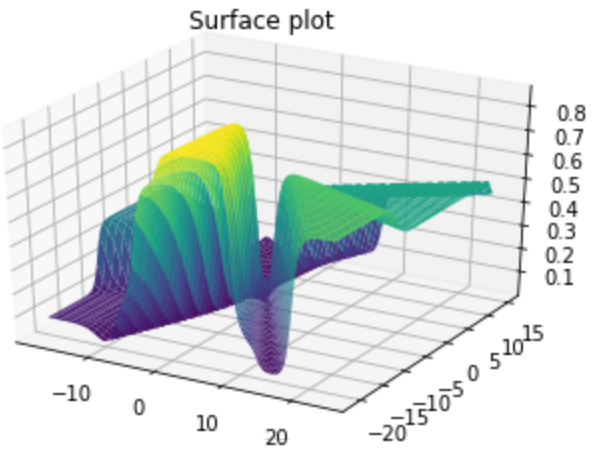
Epoch 2, Output layer



Node 1

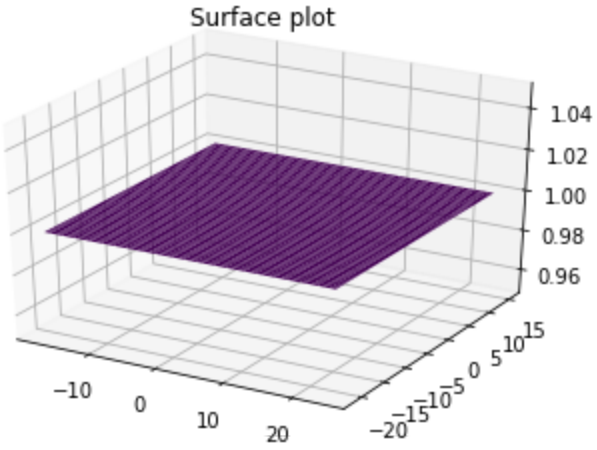


Node 2

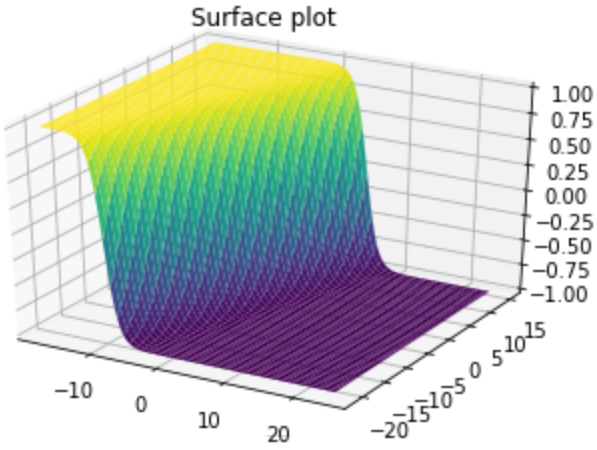


Node 3

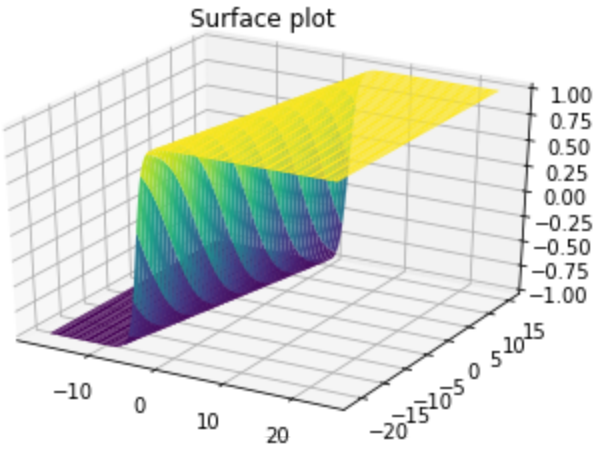
Epoch 10, Hidden Layer 1



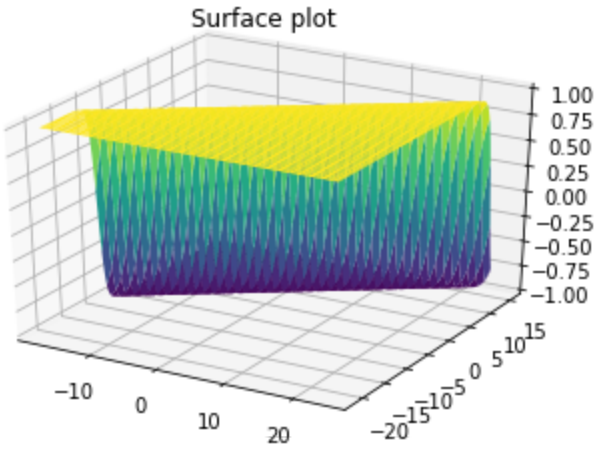
Node 1



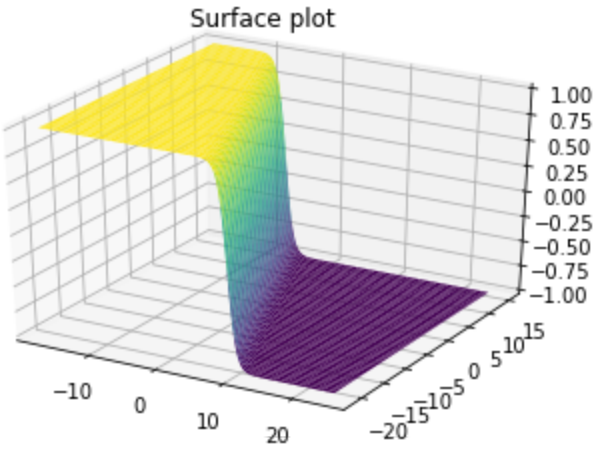
Node 2



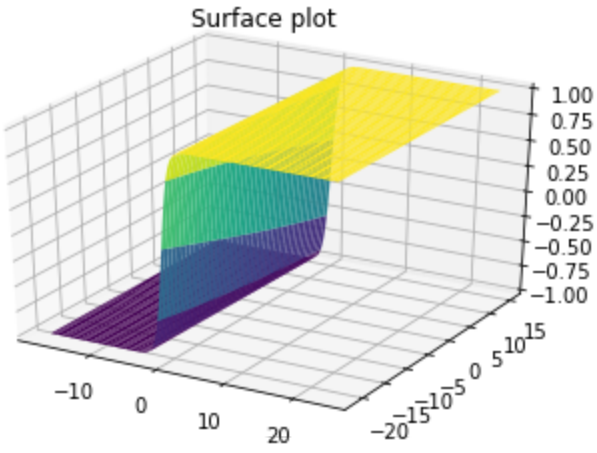
Node 3



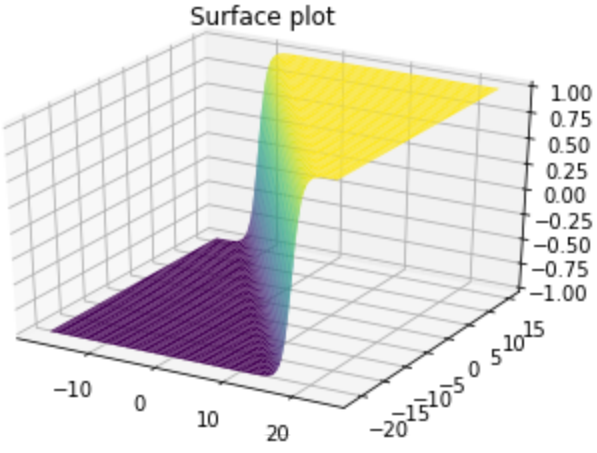
Node 4



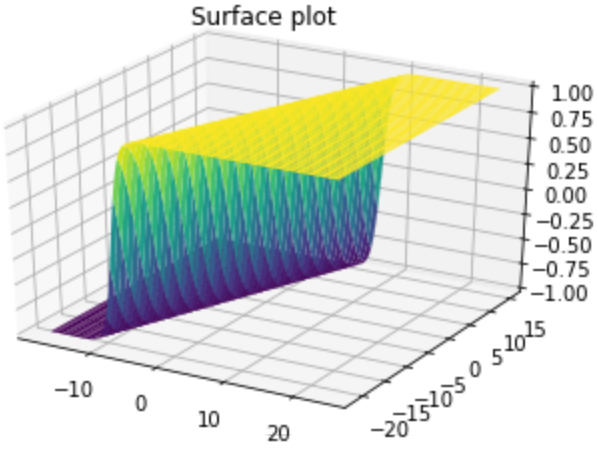
Node 5



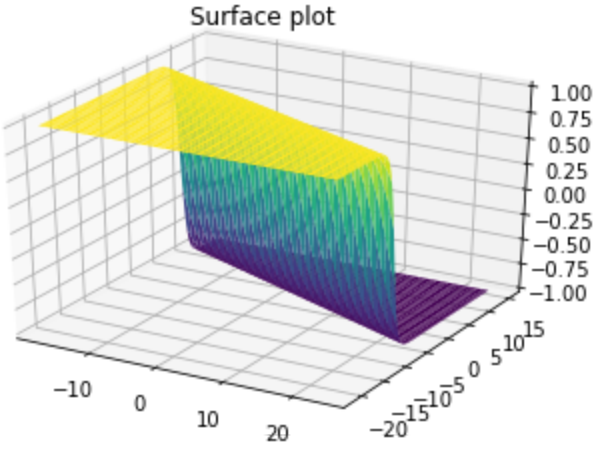
Node 6



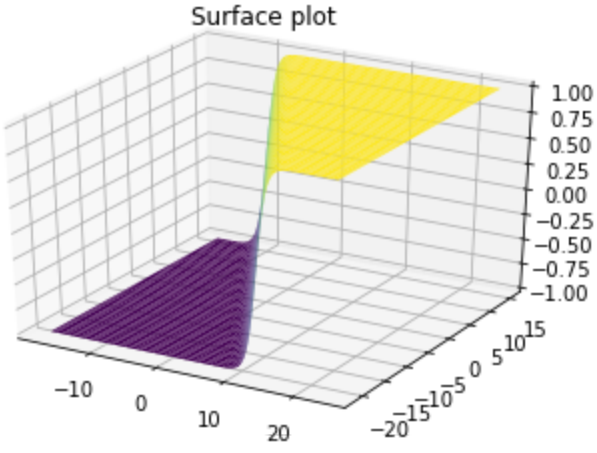
Node 7



Node 8

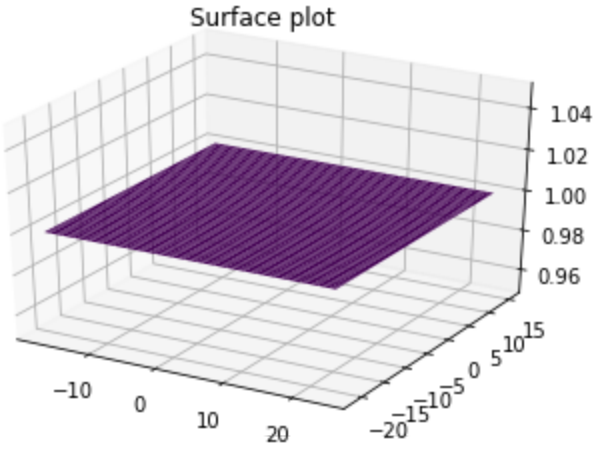


Node 9

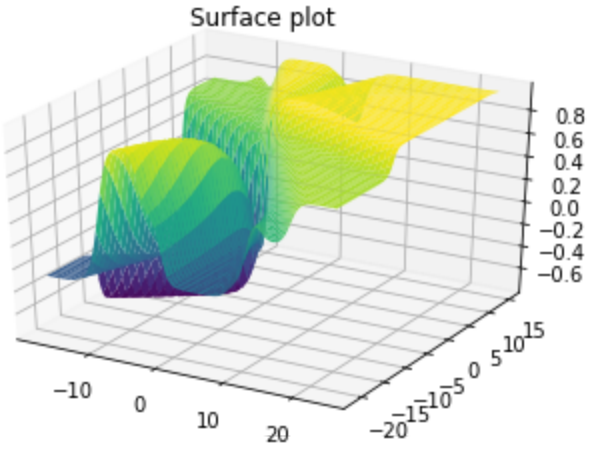


Node 10

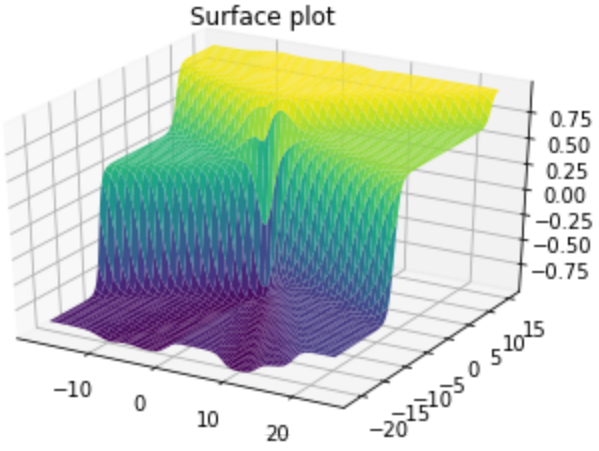
Epoch 10, Hidden Layer 2



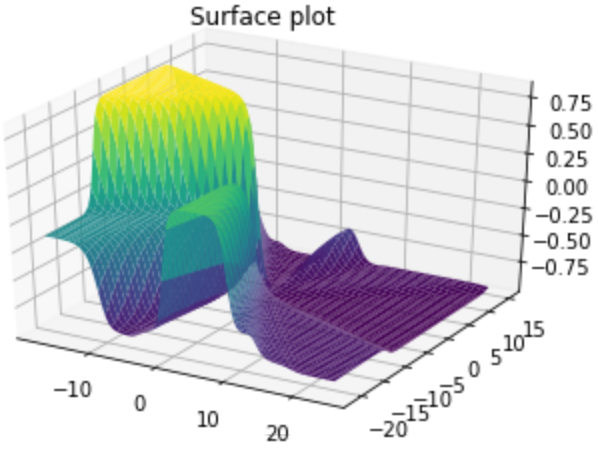
Node 1



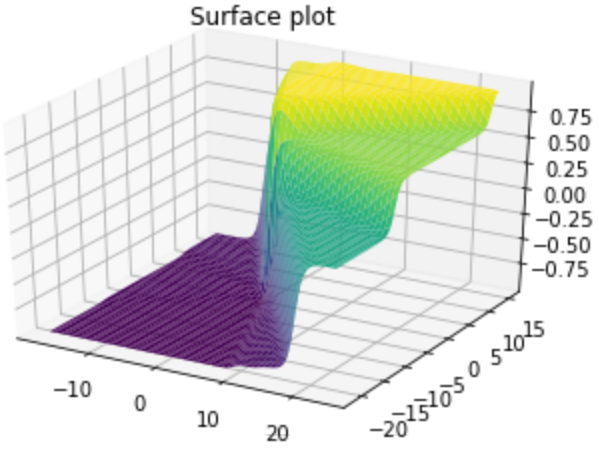
Node 2



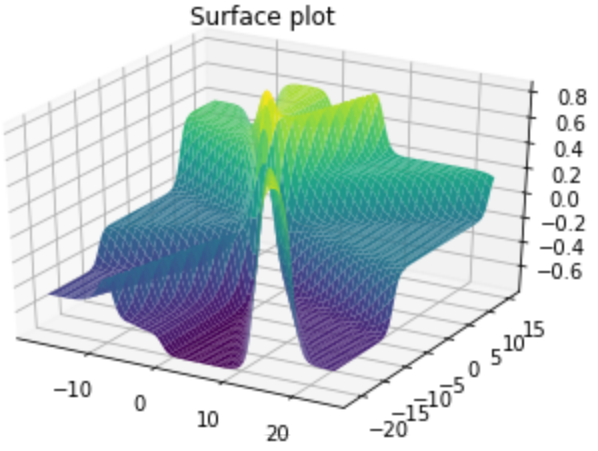
Node 3



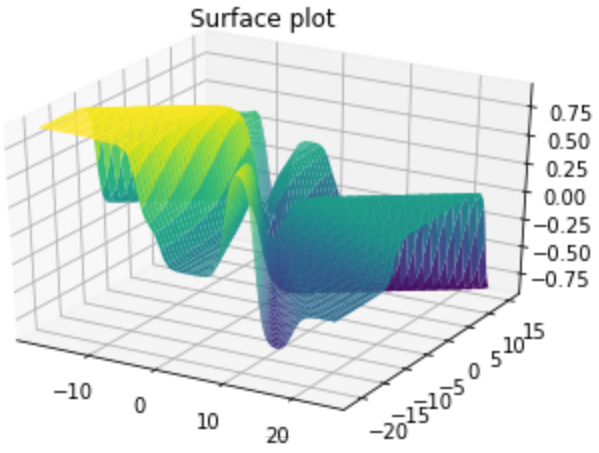
Node 4



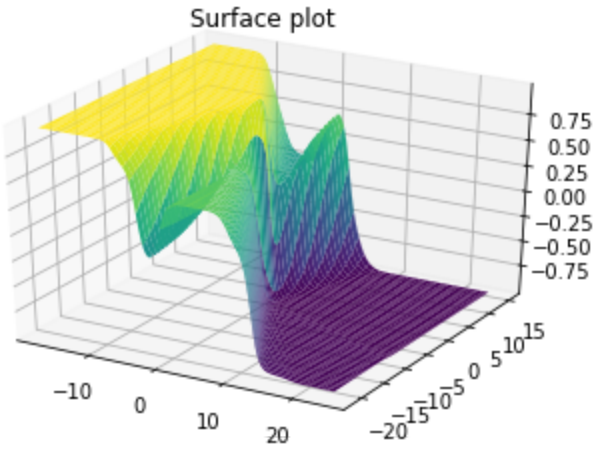
Node 5



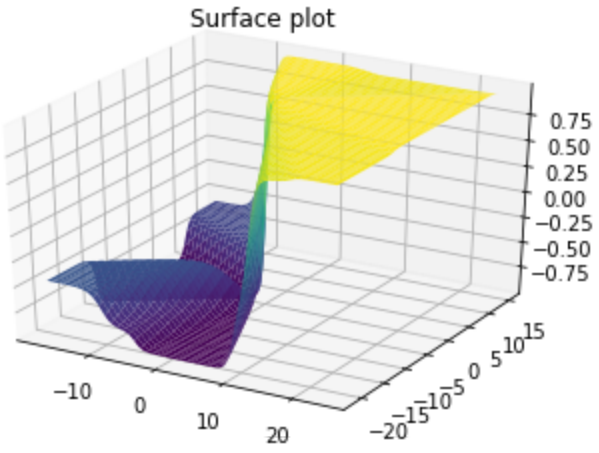
Node 6



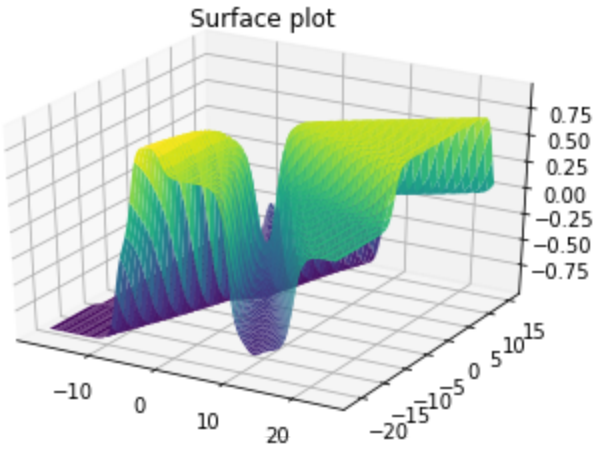
Node 7



Node 8

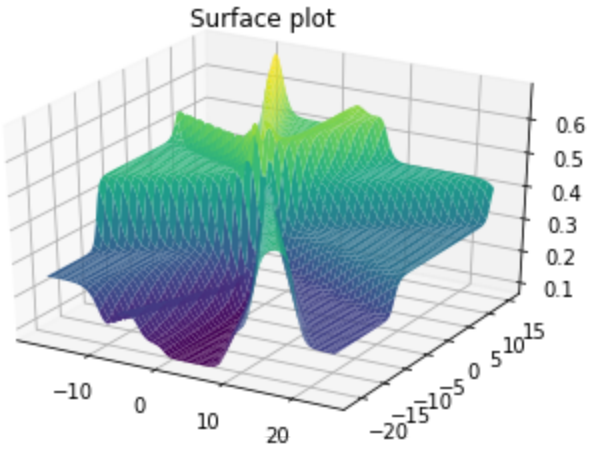


Node 9

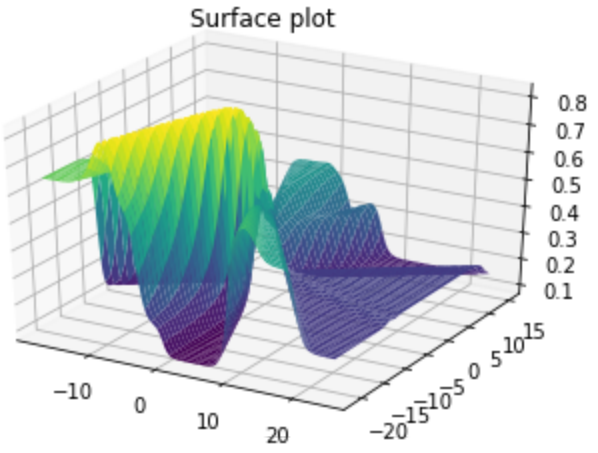


Node 10

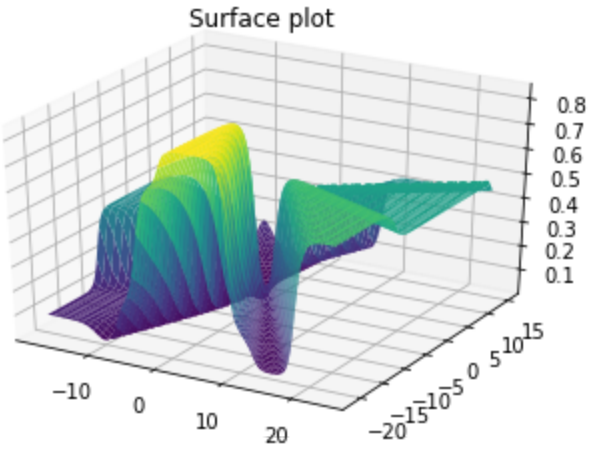
Epoch 10, Output Layer



Node 1

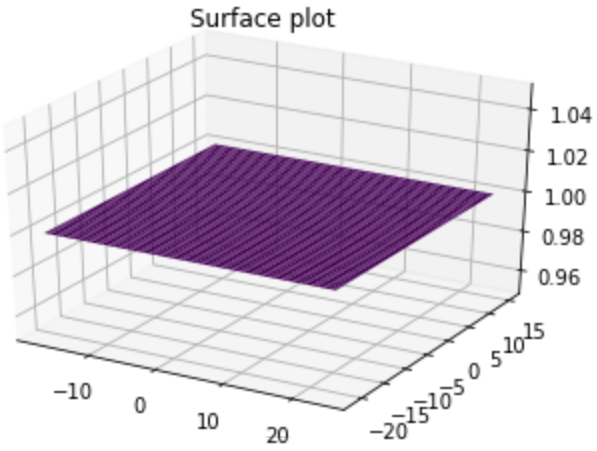


Node 2

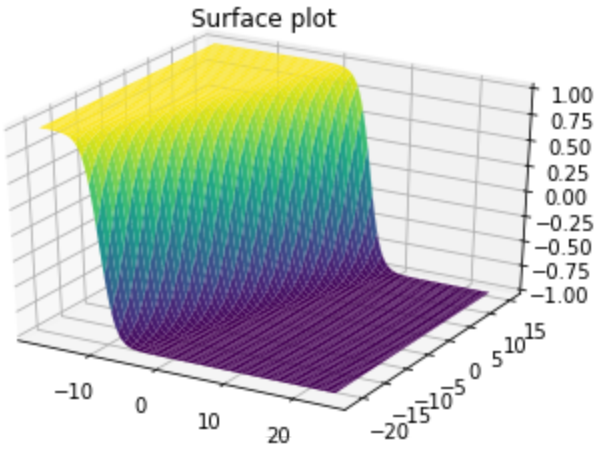


Node 3

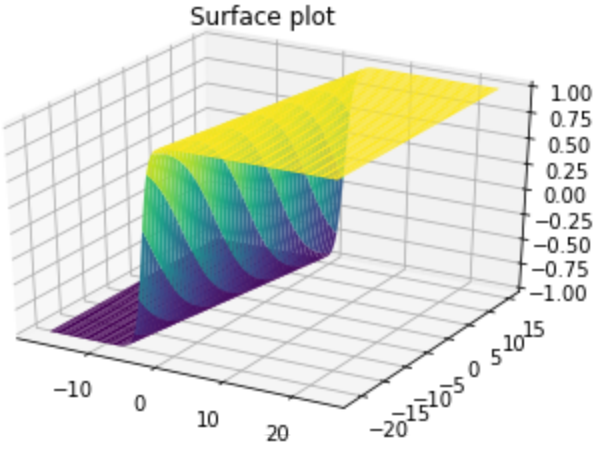
Epoch 50, Hidden Layer 1



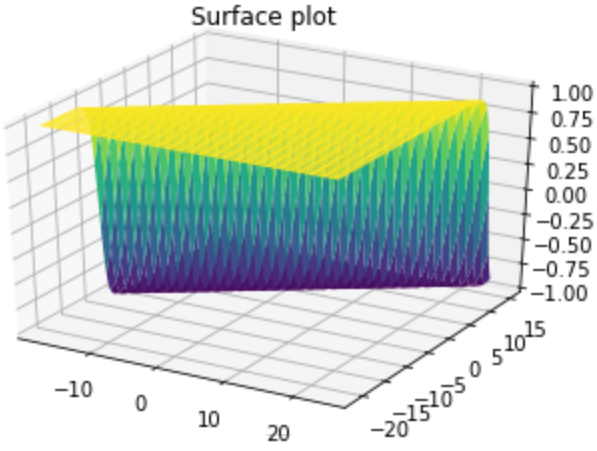
Node 1



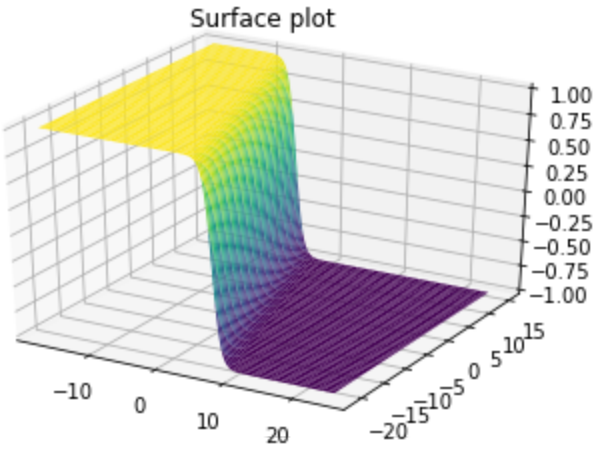
Node 2



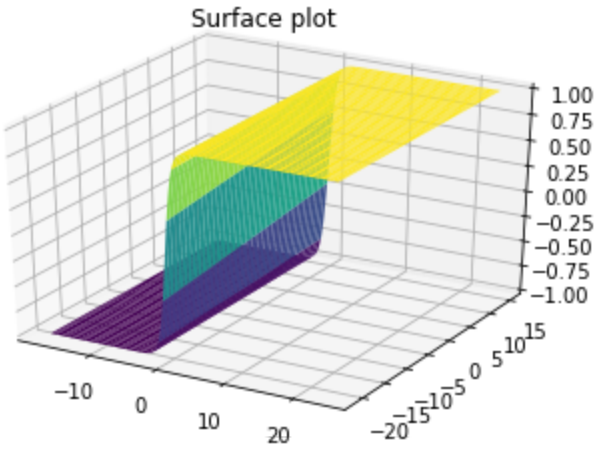
Node 3



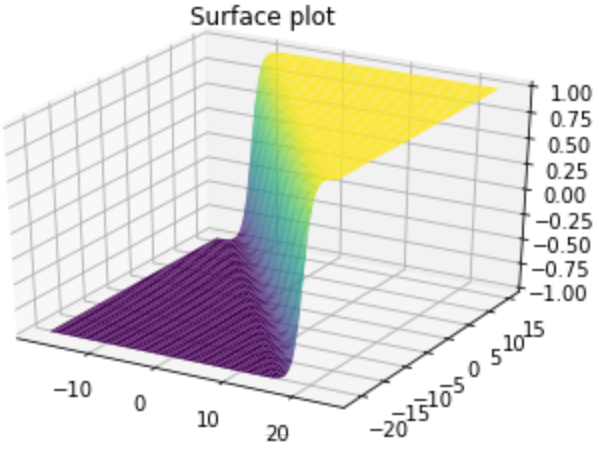
Node 4



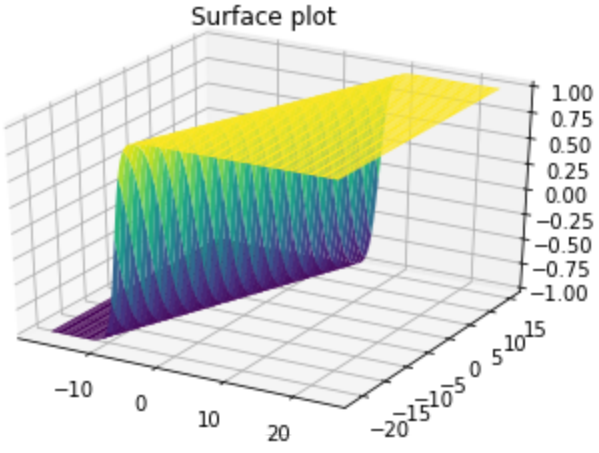
Node 5



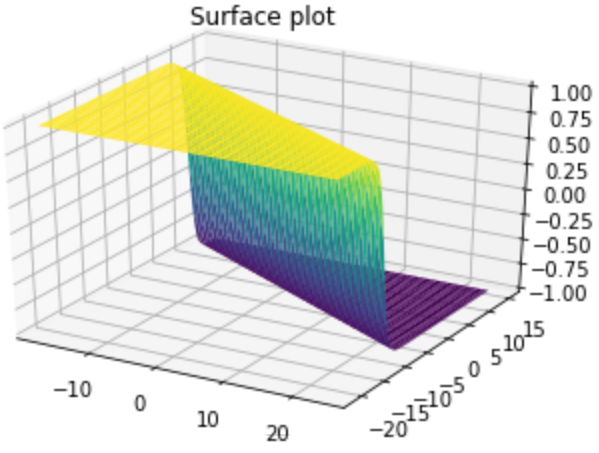
Node 6



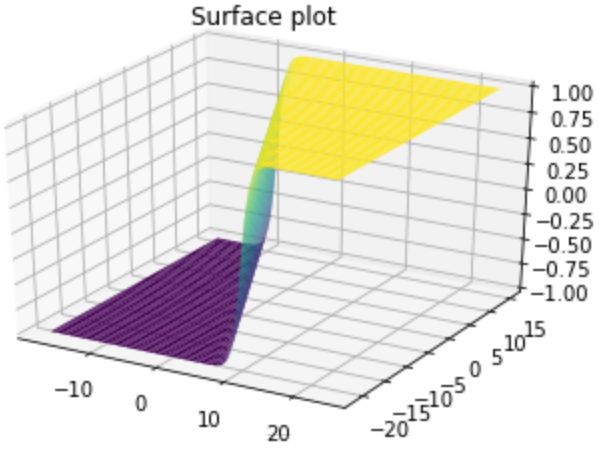
Node 7



Node 8

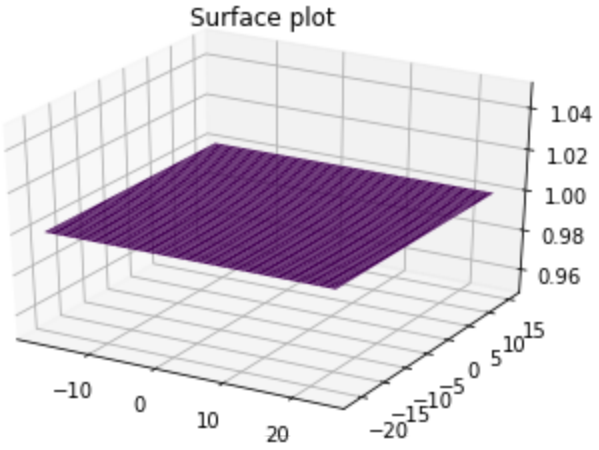


Node 9

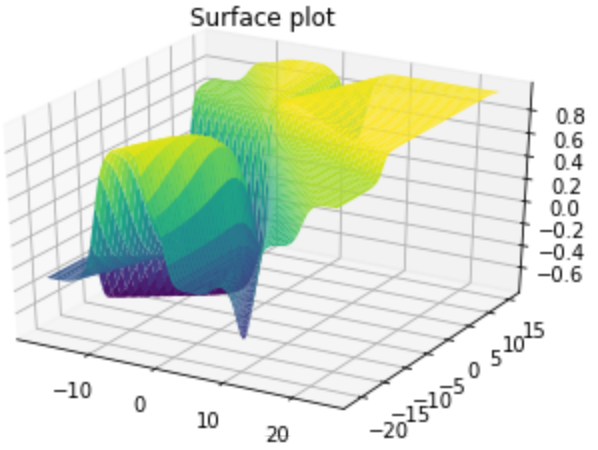


Node 10

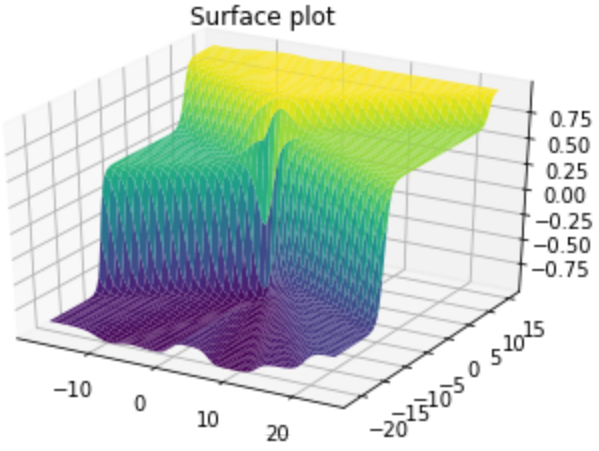
Epoch 50, Hidden Layer



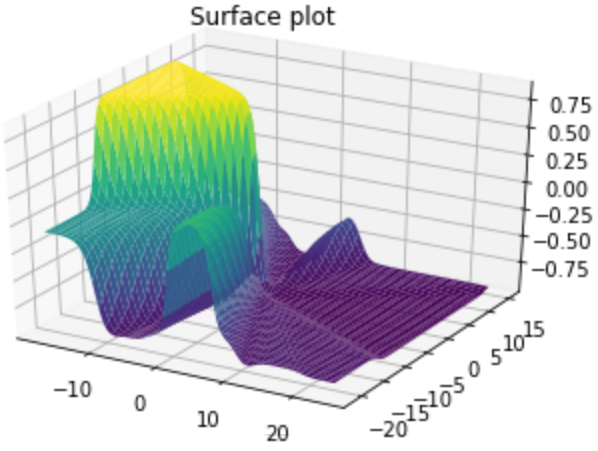
Node 1



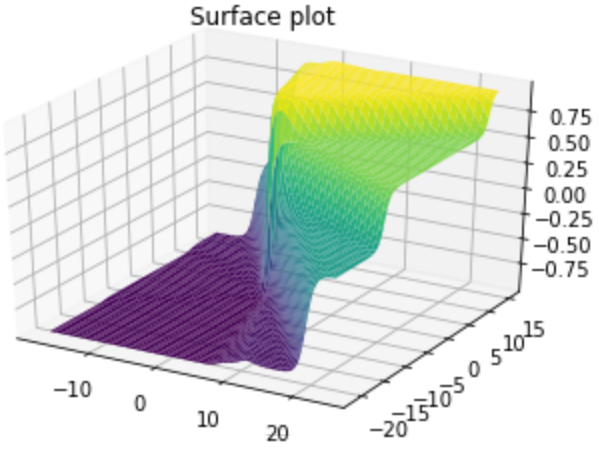
Node 2



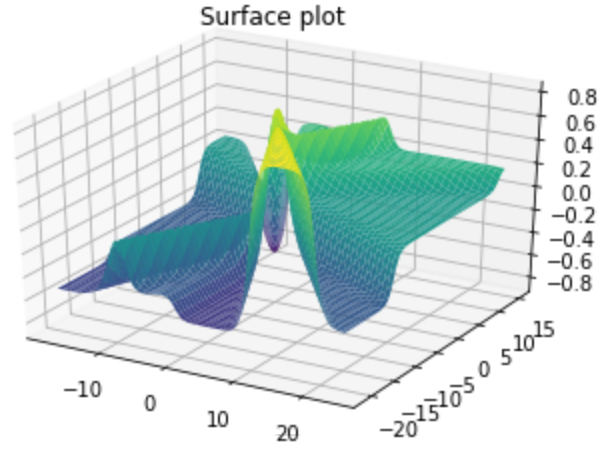
Node 3



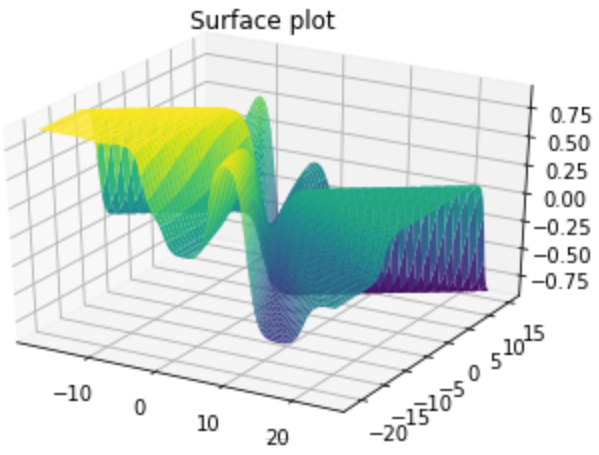
Node 4



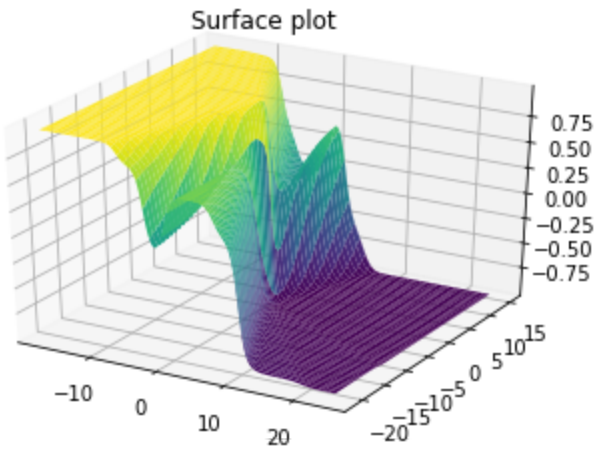
Node 5



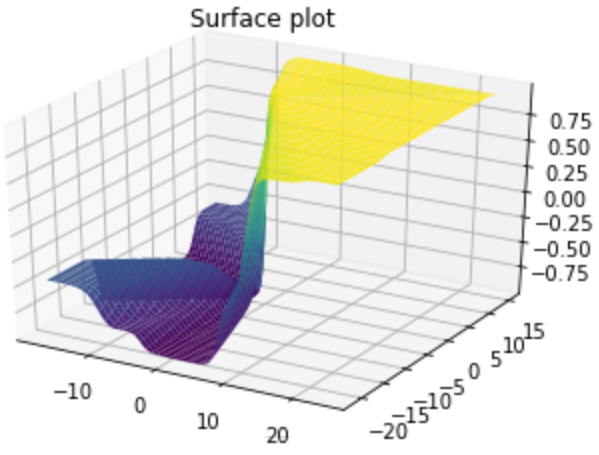
Node 6



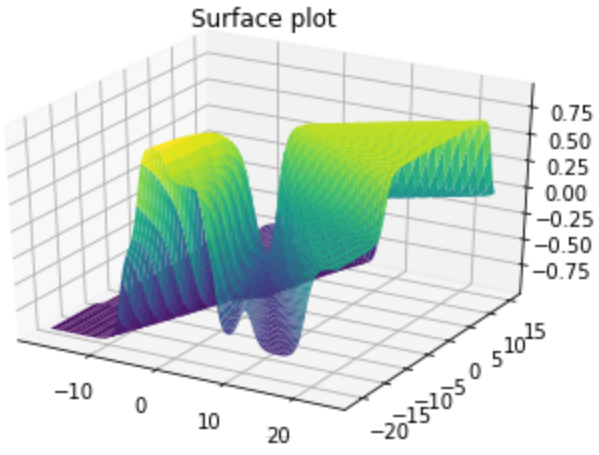
Node 7



Node 8

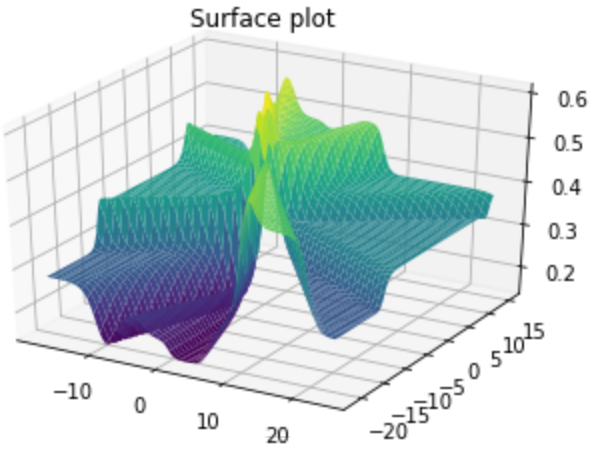


Node 9

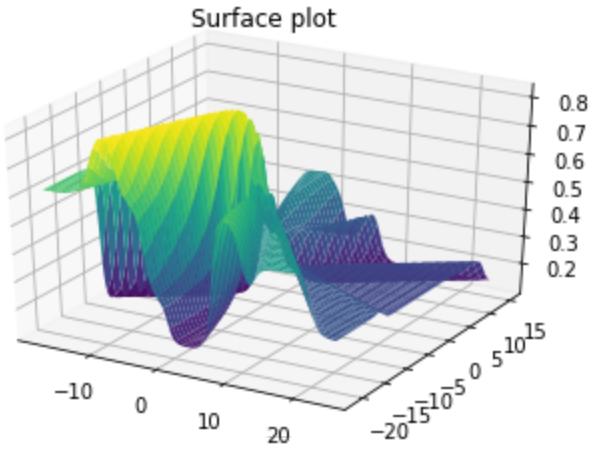


Node 10

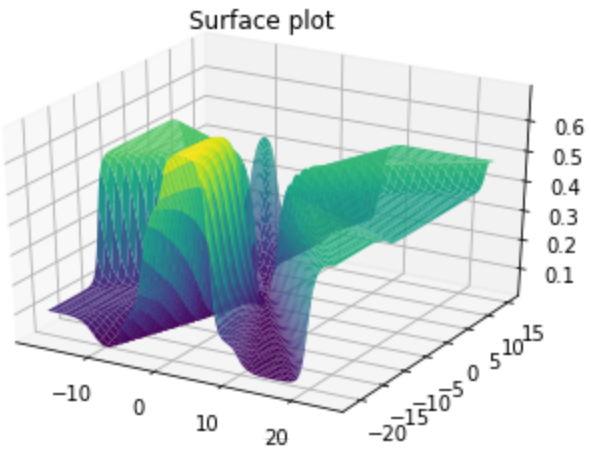
Epoch 50, Output Layer



Node 1

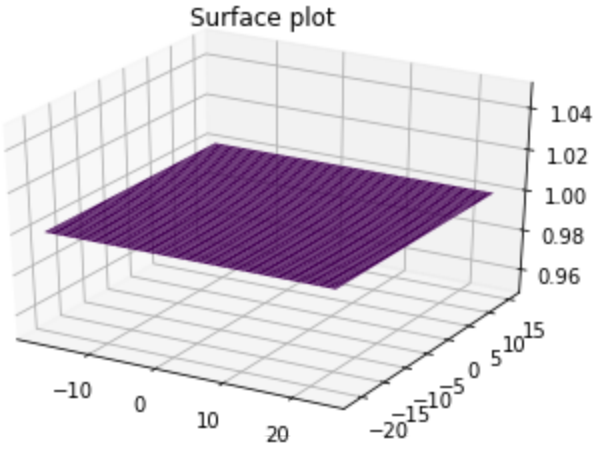


Node 2

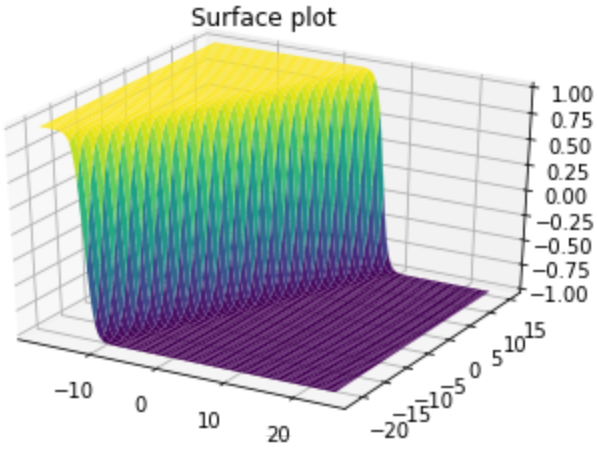


Node 3

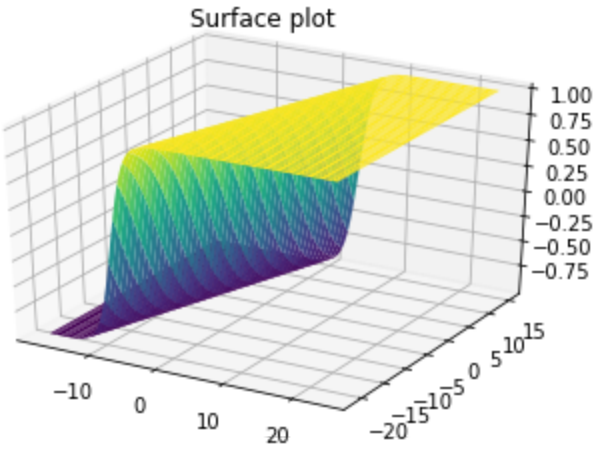
Final Epoch, Hidden Layer 1



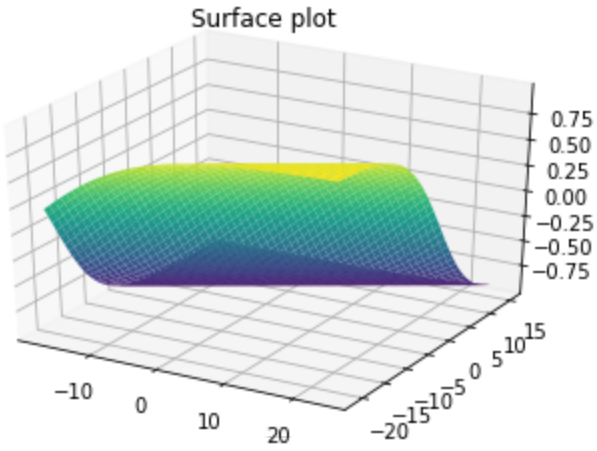
Node 1



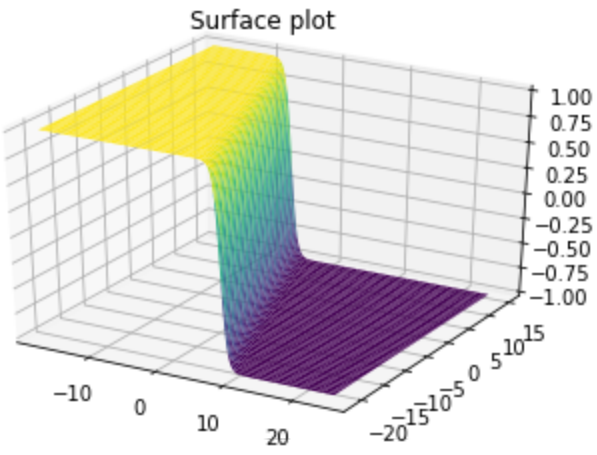
Node 2



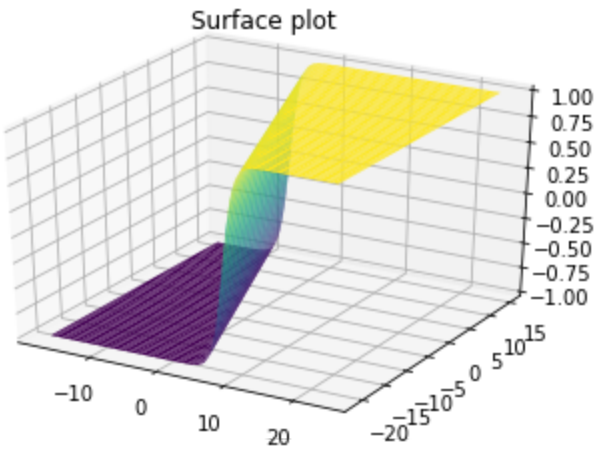
Node 3



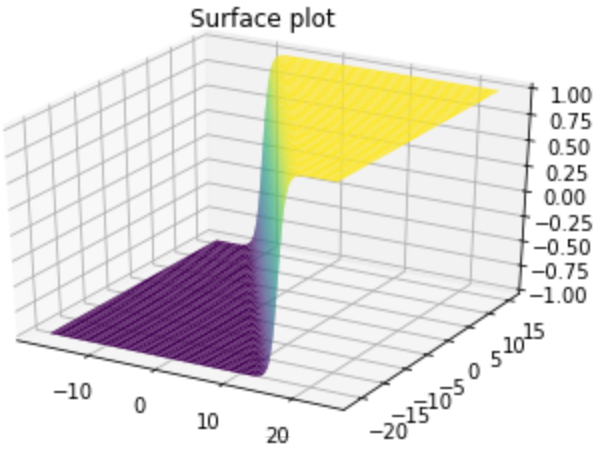
Node 4



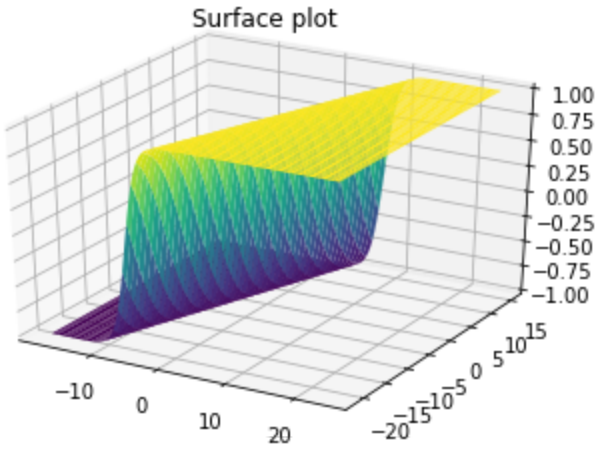
Node 5



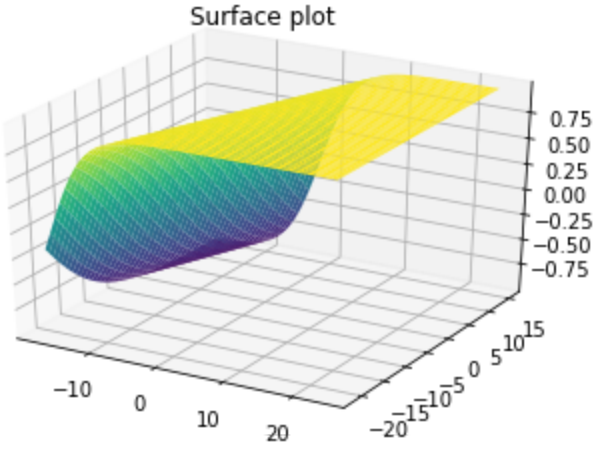
Node 6



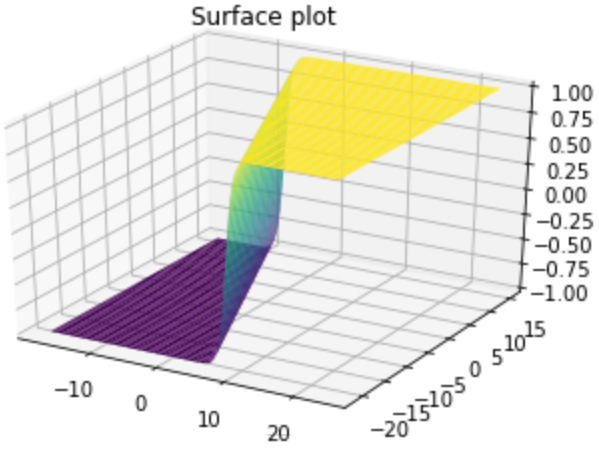
Node 7



Node 8

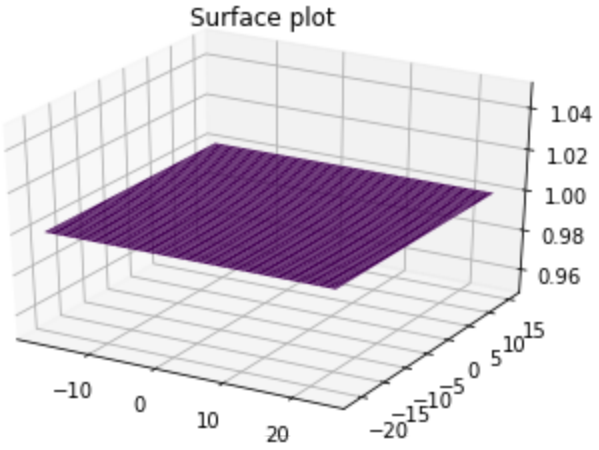


Node 9

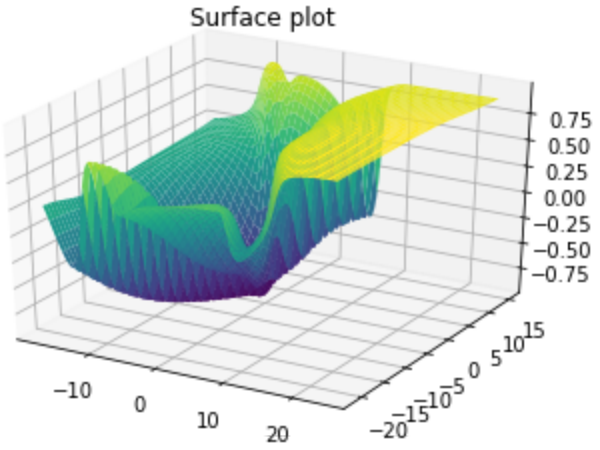


Node 10

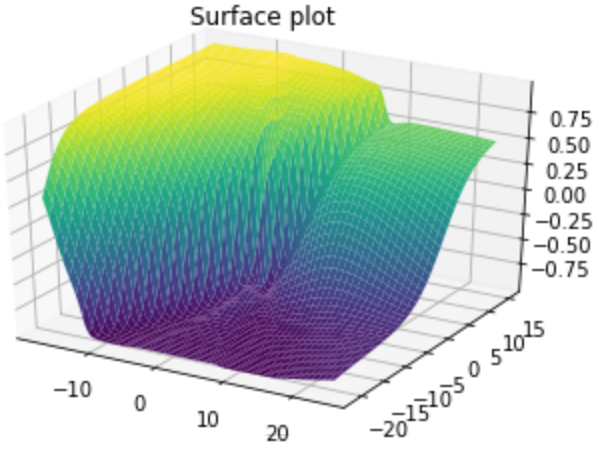
Final Epoch, Hidden Layer 2



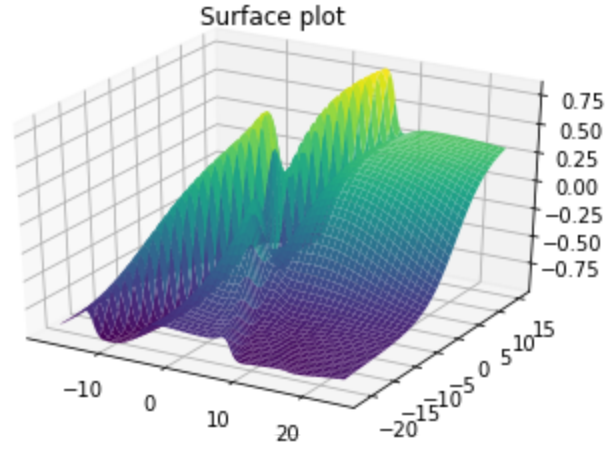
Node 1



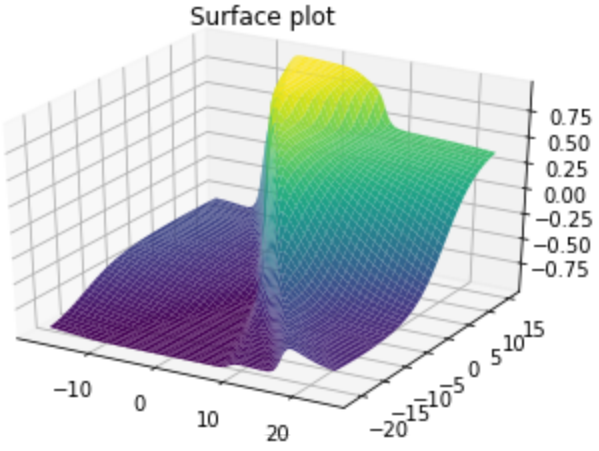
Node 2



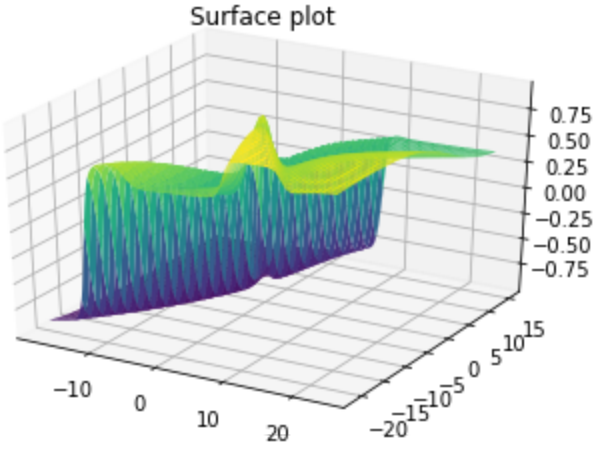
Node 3



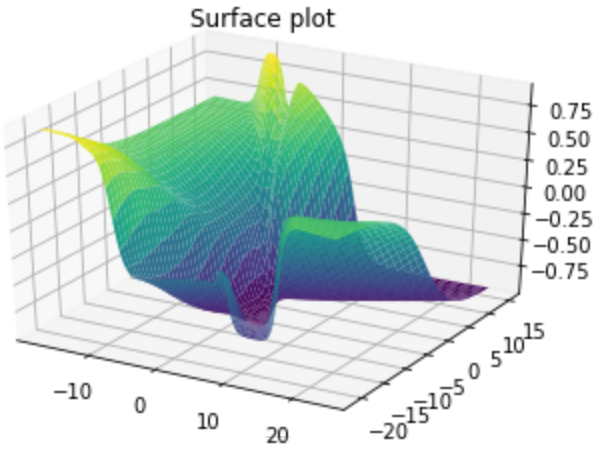
Node 4



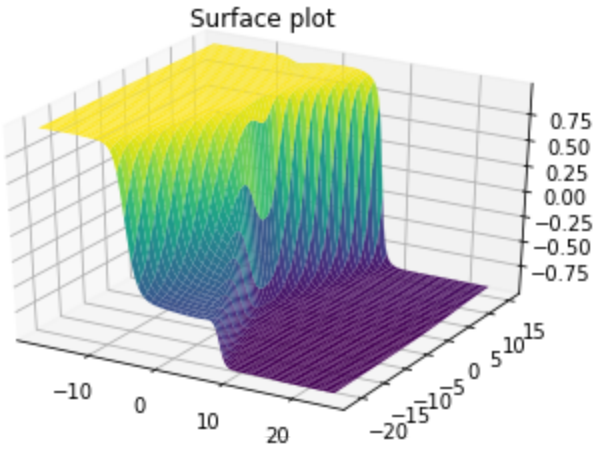
Node 5



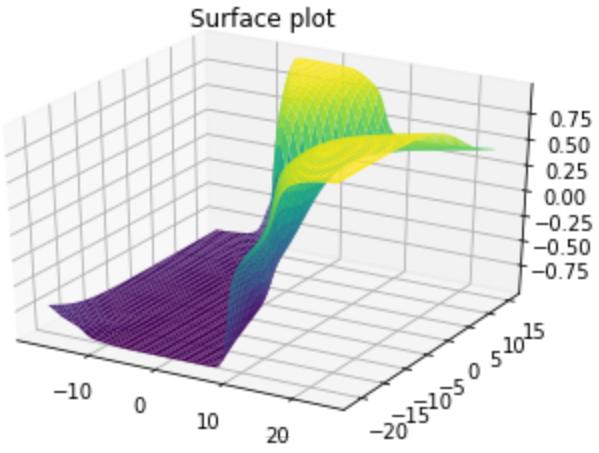
Node 6



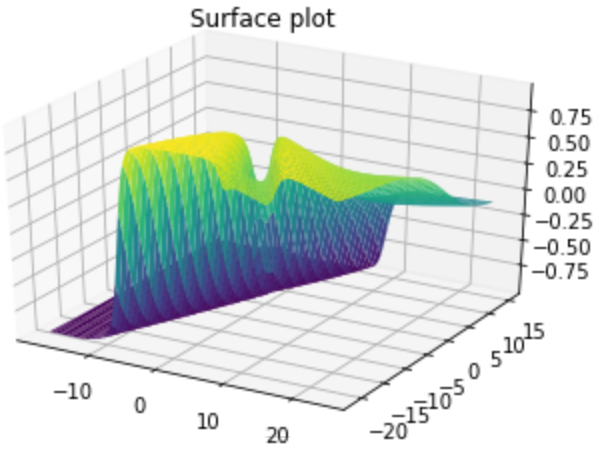
Node 7



Node 8

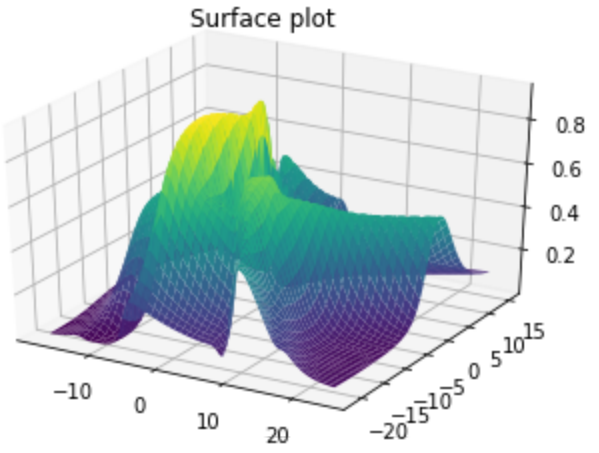


Node 9

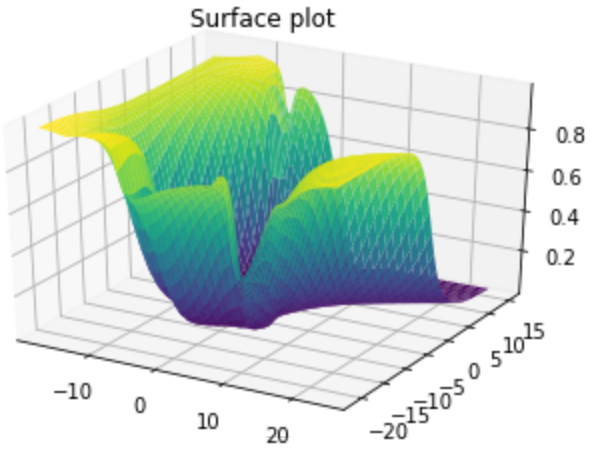


Node 10

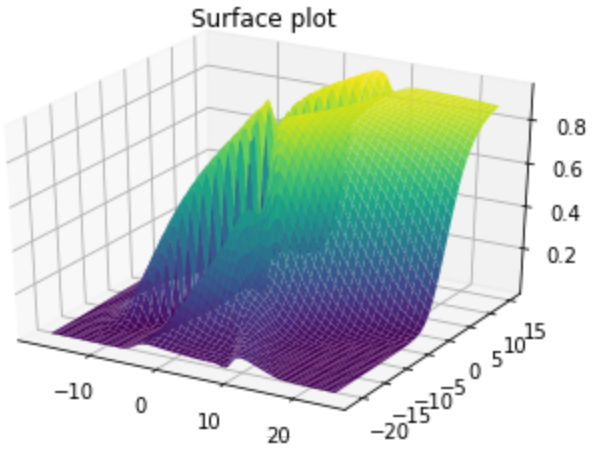
Final Epoch, Output Layer



Node 1



Node 2

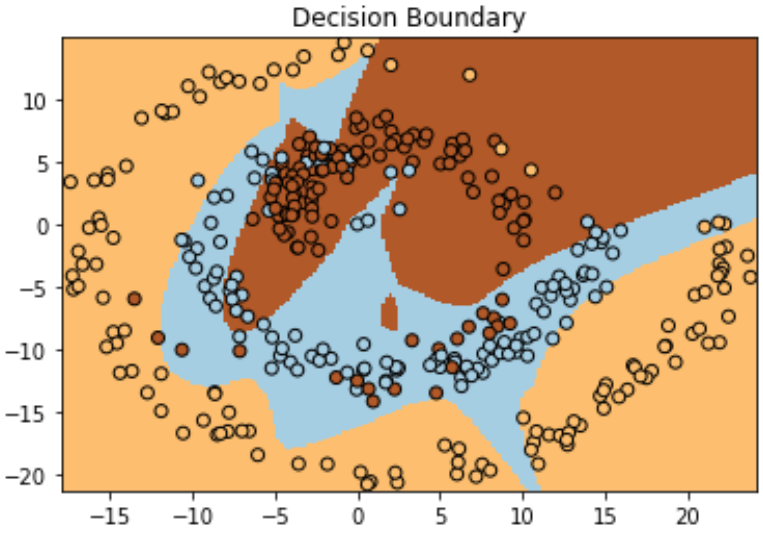


Node 3

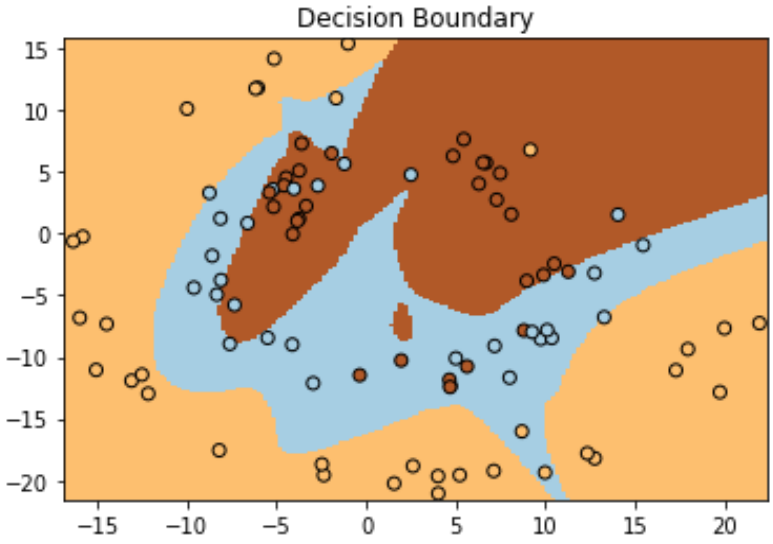
Observations

- The surface plot of the first node in both the hidden layers (in all epochs) is a linear plane. This is because the first node in a hidden layer corresponds to the bias node which is analogous to the intercept term in a linear model.
- It can be clearly seen that the surface plots of the nodes get increasingly complex (more non-linear) as we progress from the first hidden layer to the output layer. This gives us an idea of the features learned by the nodes in each hidden layer, i.e. the features learned by the hidden layers in a MLFFNN are progressively complex.
- There is barely any change in the surface plots as we go from epoch 1 to epoch 2. However a comparison of surface plots of epoch 1 and epoch 50 shows a clear difference. As the epochs increase, the surfaces of all the nodes become well defined.

Decision Regions



Training Data



Test Data

Blue colour - Class 0
Creme colour - Class 1
Brown colour - Class 2

Confusion matrices

```
Training Data

[[ 87.   6.  29.]
 [  3. 110.   4.]
 [ 20.   2.  99.]]
Train Accuracy = 0.8222222222222222
```

```
Test Data

[[19.  2.  8.]
 [ 3. 29.  1.]
 [ 8.  0. 21.]]
Test Accuracy = 0.7582417582417582
```

From the decision boundary plots and confusion matrices of the test and training data the following observations can be made:

- There seems to be a higher confusion between classes 0 and 2. This is because the data points these two classes are very closely spaced and some of them overlap.
- On the other hand, data points of class 1 are rarely mis-classified and their decision boundary is well defined.
- The chosen MLFFNN model does not overfit the training data and performs satisfactorily.

Question 3: Classification task for image data

Preprocessing

The input data has high dimensionality but insufficient data size to train a complex neural network. To overcome this, PCA was applied to retain 97% variance. It created a 400-dimensional representation of the 512-dimensional data. Any further reduction in dimension caused a loss of information.

Choice of Architecture

The following models were tried out in increasing order of complexity and the accuracy in the validation set was compared for model selection.

Model Architecture	**Validation Accuracy**	**Model Architecture**	**Validation Accuracy**
3, 3	0.227	4, 6	0.242
3, 4	0.228	6, 4	0.239
4, 3	0.229	**4, 7**	**0.257**
3, 5	0.236	7, 4	0.245
5, 3	0.214	5, 10	0.219
4, 4	0.218	10, 5	0.218
4, 5	0.230	10, 10	0.251
5, 4	0.233	10, 20	0.236
5, 5	0.224	20, 10	0.242
5, 6	0.229	15, 15	0.234
6, 5	0.252	20, 20	0.235

Training accuracy increases with model complexity (due to overfitting). The test accuracy increased with model complexity until a certain complexity and then decreased. Thus, architecture (4, 7) was selected.

Modeling Challenges

The model faces the following challenges:

- High feature dimensionality - 512 features
- High complexity requirement - Needs at least (5, 5) nodes in the hidden layer to get reasonable test accuracy.
- Low Dataset size - 1500 points (allows estimating approximately only 200 parameters accurately). Thus, the model overfits if the number of parameters is high.

These 3 factors force the model to have very low test accuracy.

Confusion Matrices

Delta Method

```
Training Data

[[222.   0.   0.   2.  20.]
 [  1. 239.   0.   4.   1.]
 [  1.   2. 223.   7.   2.]
 [  5.   1.  10. 216.   3.]
 [  8.   1.   2.   0. 230.]]
Train Accuracy = 0.9416666666666667

Test Data

[[ 8. 17.  9.  9. 13.]
 [12. 16. 10.  7. 10.]
 [10. 18. 12.  9. 16.]
 [13.  9. 13.  8. 22.]
 [16. 12.  7.  6. 18.]]
Test Accuracy = 0.20666666666666667
```

Generalized Delta Method

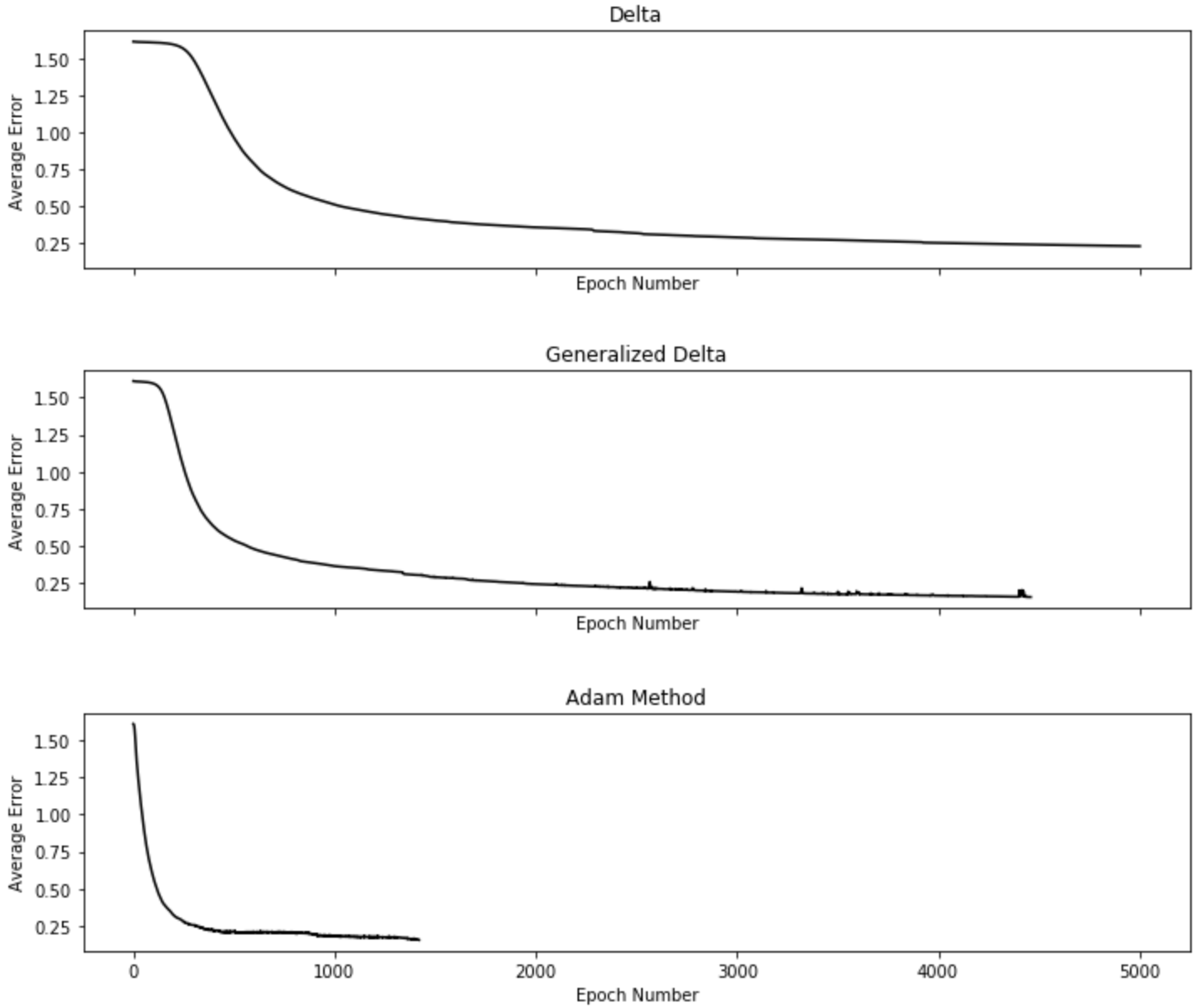
```
Training Data

[[232.   0.   0.   6.   6.]
 [  1. 239.   0.   4.   1.]
 [  1.   2. 221.   9.   2.]
 [  2.   1.   2. 228.   2.]
 [  8.   1.   0.   2. 230.]]
Train Accuracy = 0.9583333333333334

Test Data

[[10. 17.  7. 13.  9.]
 [10. 15.  7.  8. 15.]
 [13. 16. 12. 12. 12.]
 [11.  8. 10. 20. 16.]
 [14. 13.  6.  7. 19.]]
Test Accuracy = 0.25333333333333335
```

```
Training Data

 [[244.   0.   0.   0.   0.]
 [  3. 240.   0.   2.   0.]
 [  0.   3. 223.   9.   0.]
 [  4.   1.   3. 227.   0.]
 [  3.   0.   0.   2. 236.]]
Train Accuracy = 0.975

Test Data

 [[12. 11. 11. 10. 12.]
 [ 4. 20.  8. 11. 12.]
 [13.  7. 18. 10. 17.]
 [13. 13.  6. 19. 14.]
 [19. 13.  2. 12. 13.]]
Test Accuracy = 0.27666666666666667
```

We see that Adam's method has slightly better accuracy than Generalized Delta which is better than Delta. However, this trend **was not always observed in multiple runs**. Even though Delta performed the worst, in some cases Generalized Delta had better accuracy than Adam.

Convergence of different weight update rules

We see that the model converges faster for Adam's method followed by the Generalized Delta rule and slowest for Delta rule (for the same initial guess, seed and stopping condition). This observation was seen for multiple runs.



This is because the delta rule looks at the **momentary slope** of the cost function only. The generalized delta looks at the **momentum** (exponentially decaying weights to previous gradients) too to avoid spurious updates by accounting for **curvature**. Adam's rule uses **exponentially weighted moving averages** for both the gradient and the curvature to get better estimates for these terms. Thus it looks at the nature of the cost function as a whole and **not the local nature**.