# Machine Learning

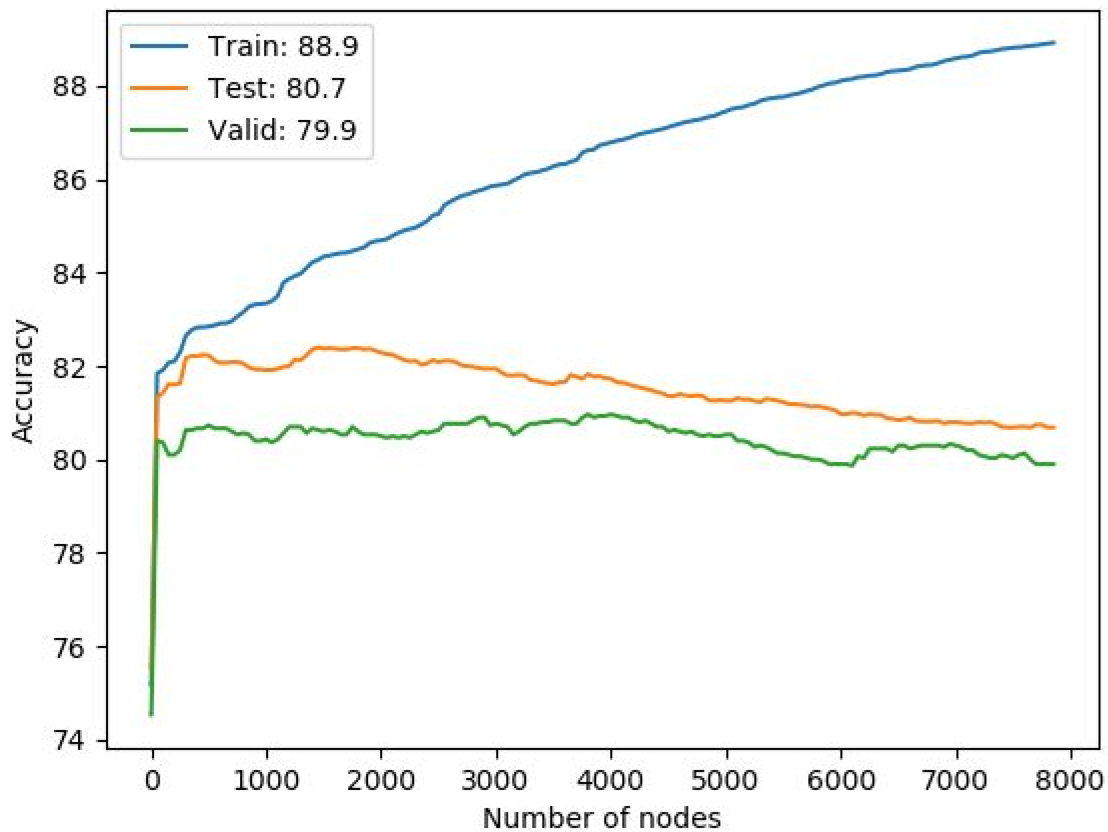
## Assignment 3

Shadab Zafar
2017MCS2076

# Q1: Decision Trees

**a.)**

Decision tree accuracy as number of nodes grow




The decision tree generated is of height 11 with total node count of 7902.

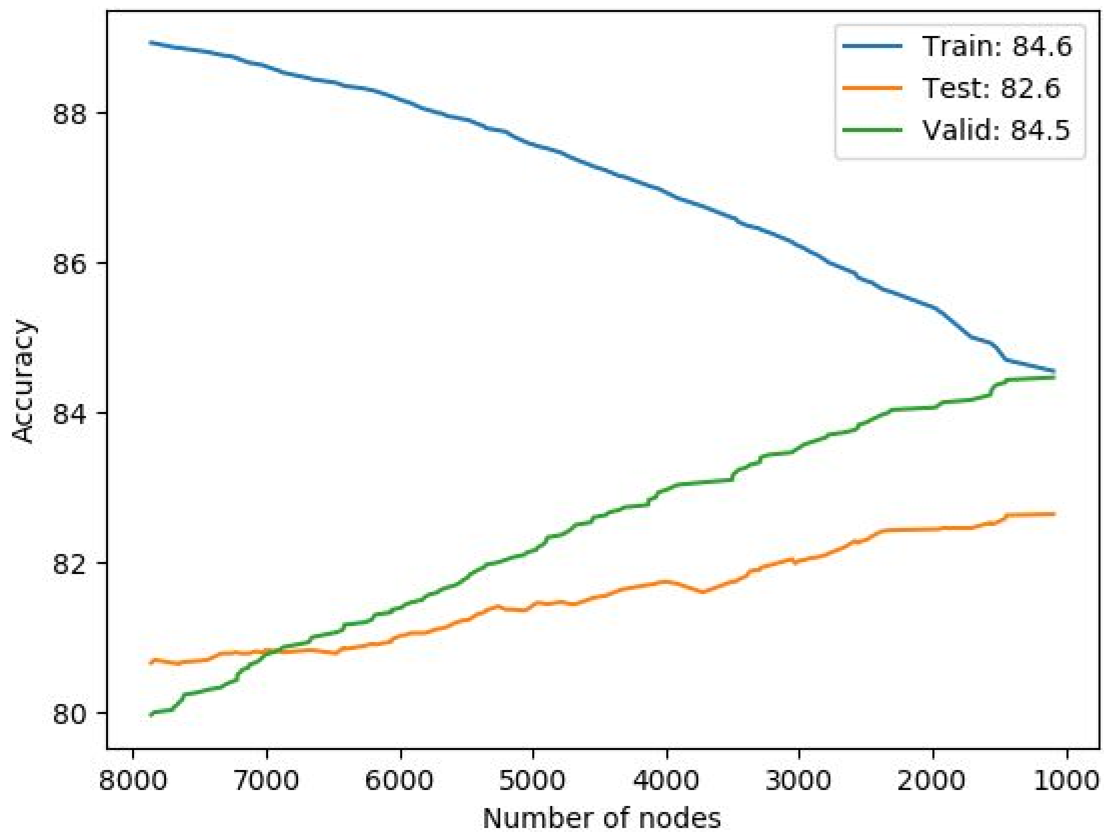
A tree with just a single decision node predicting majority class (0) results in an accuracy of ~75%.

As we can see from the graph, as number of nodes increase, Training accuracy increases but Test / Validation accuracies decrease - a typical case of Overfitting.

**b.)**

Post pruning the tree

We prune a node (and the entire subtree rooted at it) if pruning it increases the accuracy on validation data. While pruning, accuracies on all 3 datasets is calculated, resulting in the following plot:



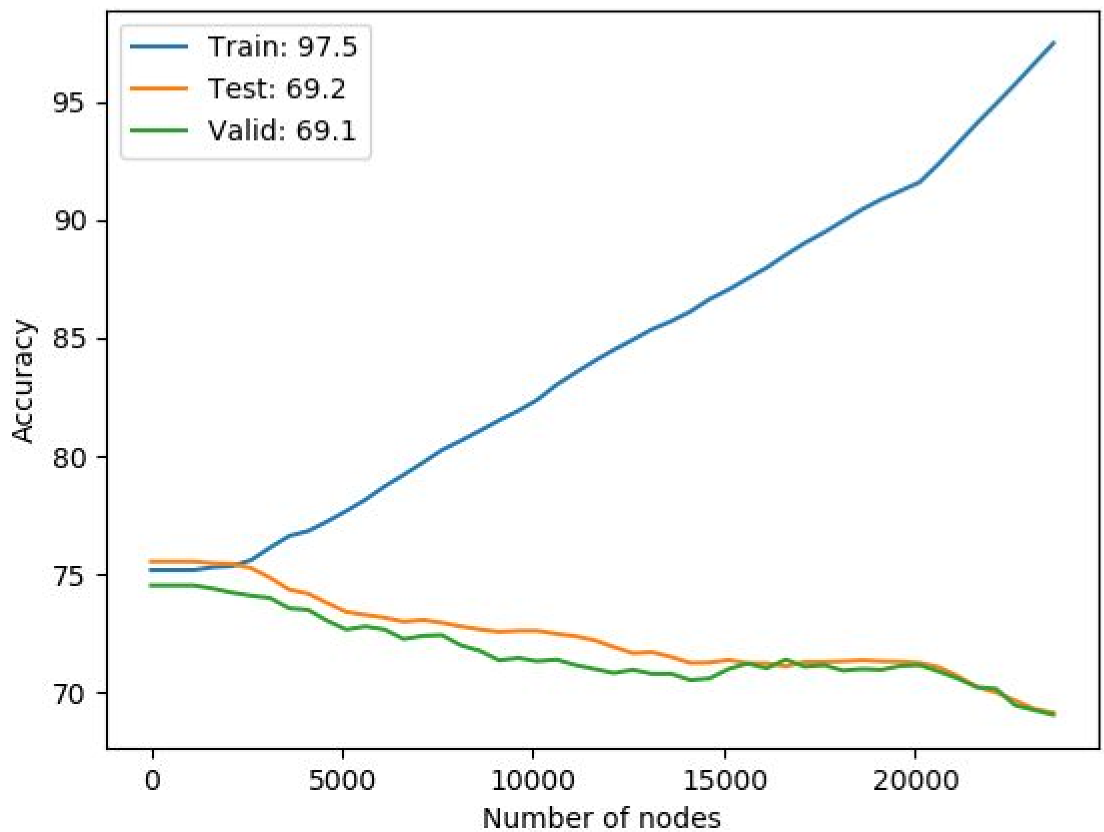

We can see how validation accuracy increases from 79.9% to 84.5%, and test accuracy increases from 80.7% to 82.6% while the training accuracy decreases. So, this type of pruning helps us reduce overfitting.

Pruning keeps the height of the tree same (11) while decreases the number of nodes from 7902 to 1069. The entire pruning process takes ~4 minutes of time on my machine.

**c.)**

Using dynamic medians (without pruning)

When computing the medians dynamically (of only the data that coming to a node) instead of computing them all at once, before, we find that the following plot is generated:




This depicts how badly the dynamic-median-tree-building algorithm overfits the data - we observe high accuracy on training data but very poor ones on test / validation data. This is due to the fact that it tries to fit to the idiosyncrasies of the data, and does not generalize well.

The observed tree is of height 16 and has 24116 nodes, which is quite bigger than what we saw in the previous case, this is due to the fact that same attributes split multiple times along a path, depending on the median value of the data coming in at that node.

**Numerical Attributes** that are split maximum times on some path, and their corresponding thresholds:

**age** 7 times: [38.0, 46.0, 56.0, 57.0, 62.0, 64.0, 65.0]

**fnlwgt** 15 times: [178615.0, 117681.0, 80665.5, 46729.0, 62176.0, 71770.0, 67248.5, 64922.0, 63577.0, 62898.5, 62485.0, 62346.0, 62374.0, 62385.0, 62438.0]

**edun** 3 times: [9.0, 9.5, 11.5]

**d.)**

scikit-learn's Decision Tree

With the default parameters:

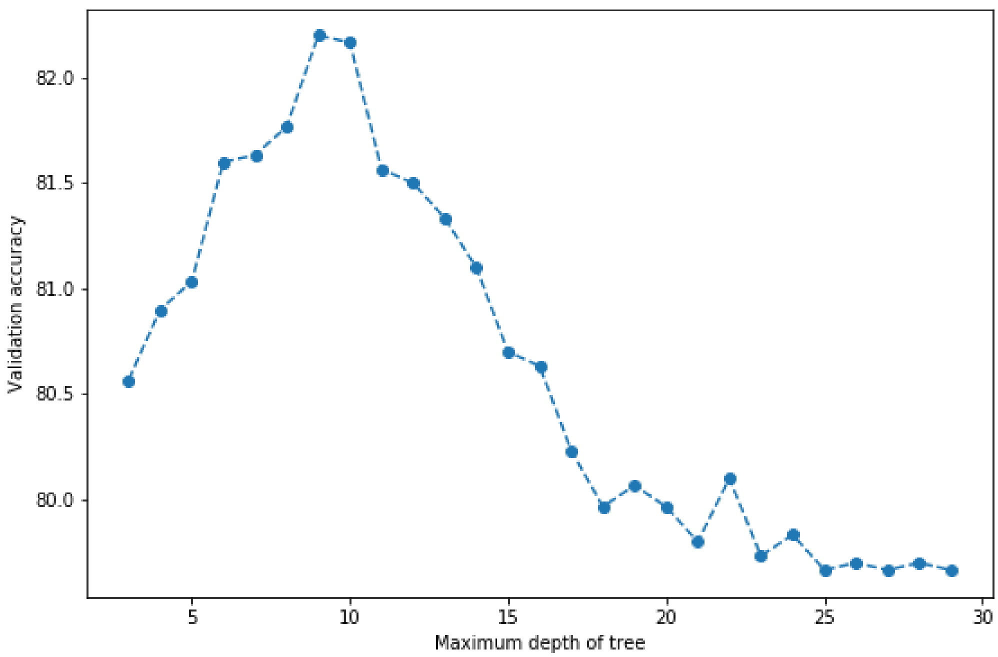
max_depth=None, min_samples_split=2, min_samples_leaf=1

Validation Acc: 79.6%, Tree Depth: 29, Number of Nodes: 7667

Explanation of parameters:

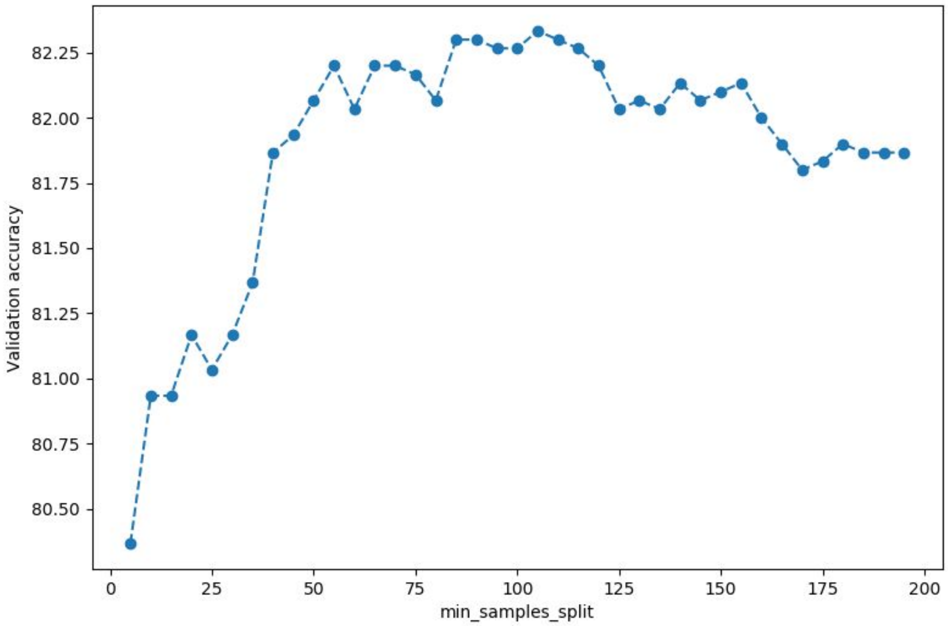
**max_depth**: Maximum depth of the tree.
**min_samples_split**: Minimum number of samples to split an internal node.
**min_samples_leaf**: Minimum number of samples to be at a leaf node.

Effect of varying **max_depth**:

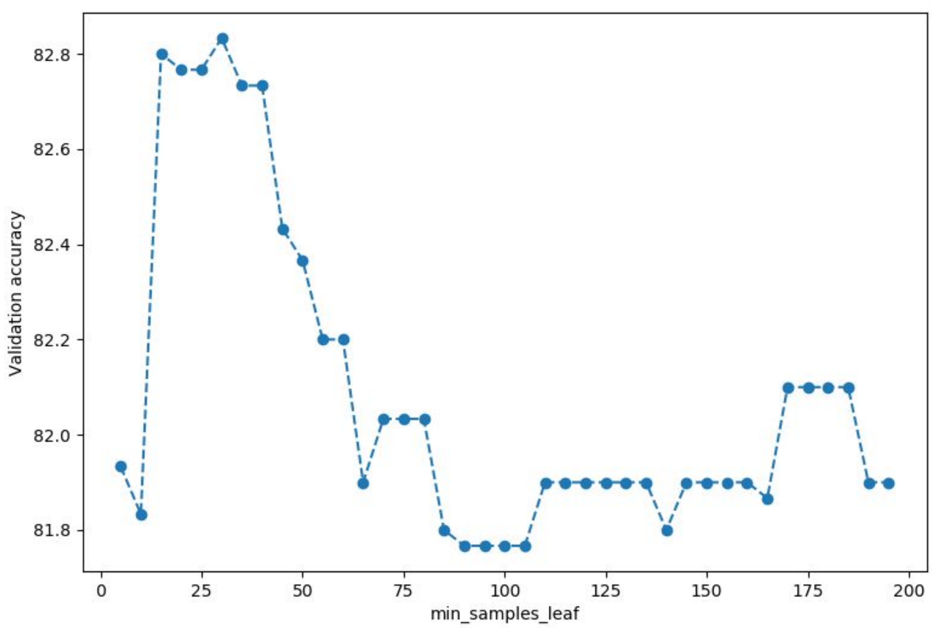



We can see how there is a sweet spot of maximum depth, beyond which the tree starts to overfit the data.

Effect of varying **min_samples_split**:




Effect of varying **min_samples_leaf:**




Running a grid search on the parameters to find the best ones, yields these:

max_depth = 12, min_samples_split = 60, min_samples_leaf = 10

Validation Acc: 83 %

**e.)**

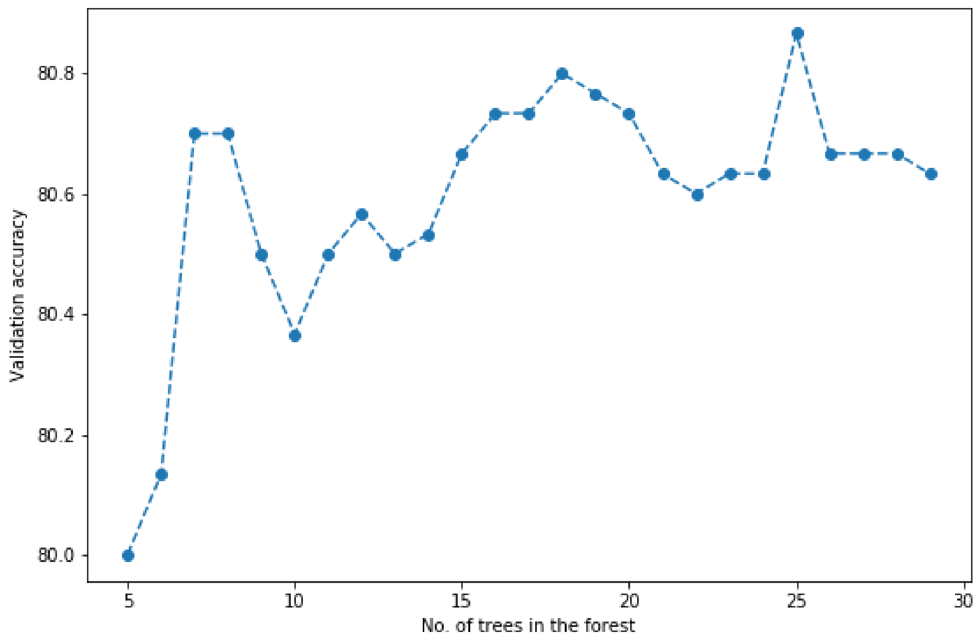
scikit-learn's Random Forests

With the default parameters:

n_estimators=10, Bootstrap=True, max_features=14

Validation Acc: 80.3%

Effect of varying **n_estimators:**




Running a grid search on the parameters to find the best ones, yields:

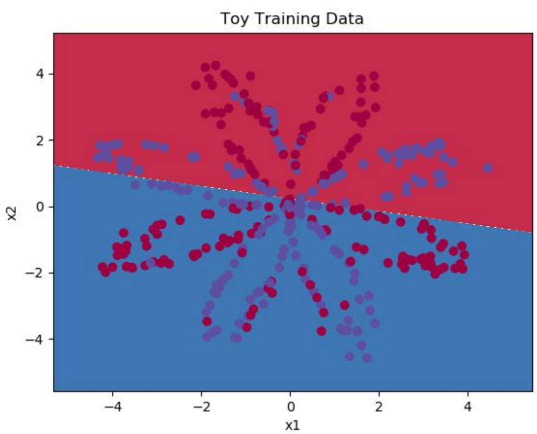
bootstrap = True, max_depth = 12, max_features = 4, n_estimators = 22

Which resulted in an accuracy of: 82.6 %

# Q2: Neural Networks

**b.)** **Toy Data**

**i.)** sklearn's Logistic Learner

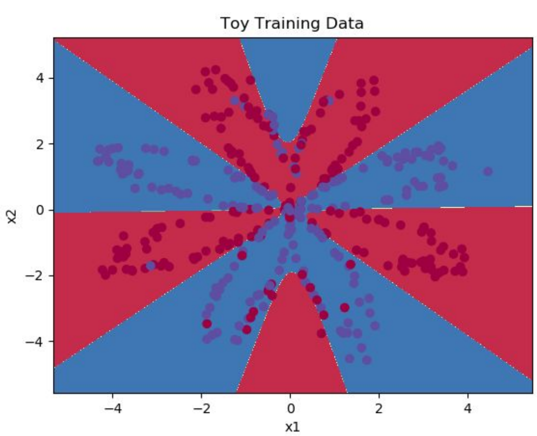




Training Accuracy: 45.8%

Testing Accuracy: 38.3%

**ii.)** Single Hidden Layer with 5 neurons





Training Accuracy: 90%

Testing Accuracy: 85.0%

We observe that, while logistic regression (which is a linear classifier) was unable to classify the data, because the data is inherently non linear. Neural Networks do a pretty good job of classifying the data correctly.

**iii.)** Effect of increasing neurons in hidden layer

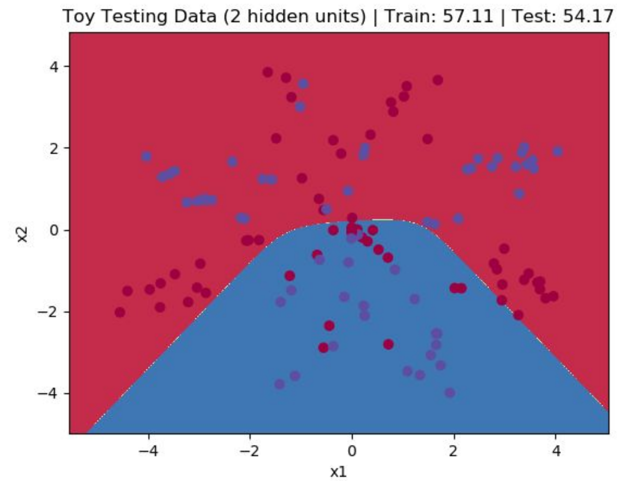
Stopping criteria was a maximum of 6000 epochs or an error threshold of 10**-6, whichever

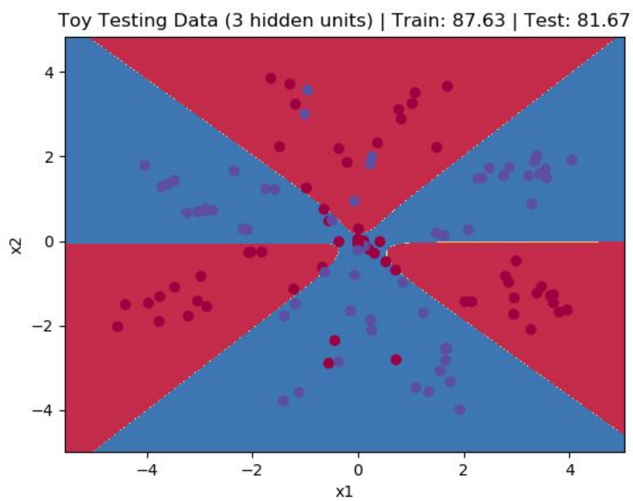
| Neurons | Training Accuracy | Testing Accuracy | Stopping |
|---|---|---|---|
| | | | |
| 1 | 57.36 | 61.66 | Epochs: 835 / 5000<br>Avg. Error: 0.12380 |
| 2 | 57.10 | 54.16 | Epoch: 2788 / 5000<br>Avg. Error: 0.11008 |
| 3 | 87.63 | 81.66 | Epoch: 3245 / 5000;<br>Avg. Error: 0.04650 |
| 5 | 88.68 | 85.0 | Epoch: 6000 / 6000<br>Avg. Error: 0.05003 |
| 10 | 88.42 | 82.50 | Epoch: 6000 / 6000<br>Error: 0.04949 |
| 20 | 87.89 | 82.50 | Epoch: 6000 / 6000<br>Error: 0.04913 |
| 40 | 88.15 | 82.50 | Epoch: 6000 / 6000<br>Error: 0.05053 |

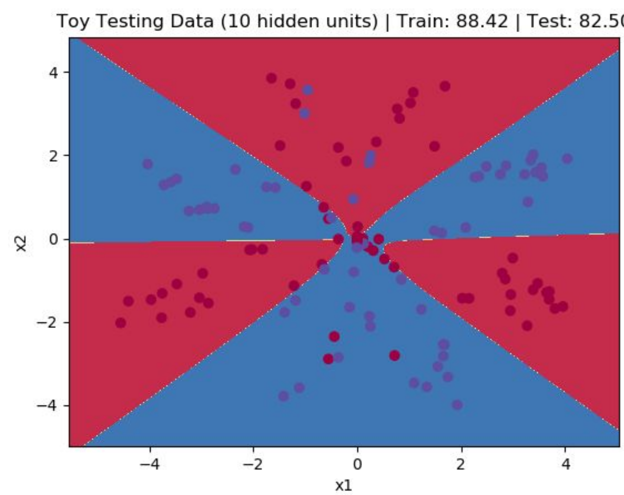
As the number of neurons in the hidden layers are increased, the network starts to learn the idiosyncrasies of the data and begins to overfit, as a result of which the decision boundaries become tighter and may not generalize well on future data.

The optimal number of neurons seem to be 5.

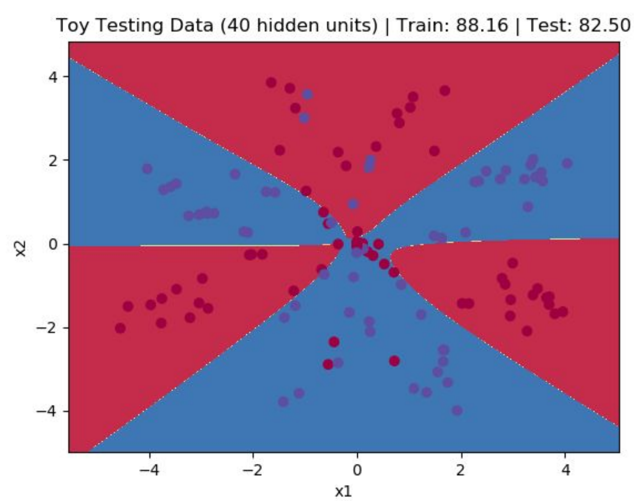

Toy Testing Data (1 hidden units) | Train: 57.37 | Test: 61.67
x1
x2


Toy Testing Data (2 hidden units) | Train: 57.11 | Test: 54.17
x1
x2


Toy Testing Data (3 hidden units) | Train: 87.63 | Test: 81.67
x1
x2


Toy Testing Data (10 hidden units) | Train: 88.42 | Test: 82.50
x1
x2

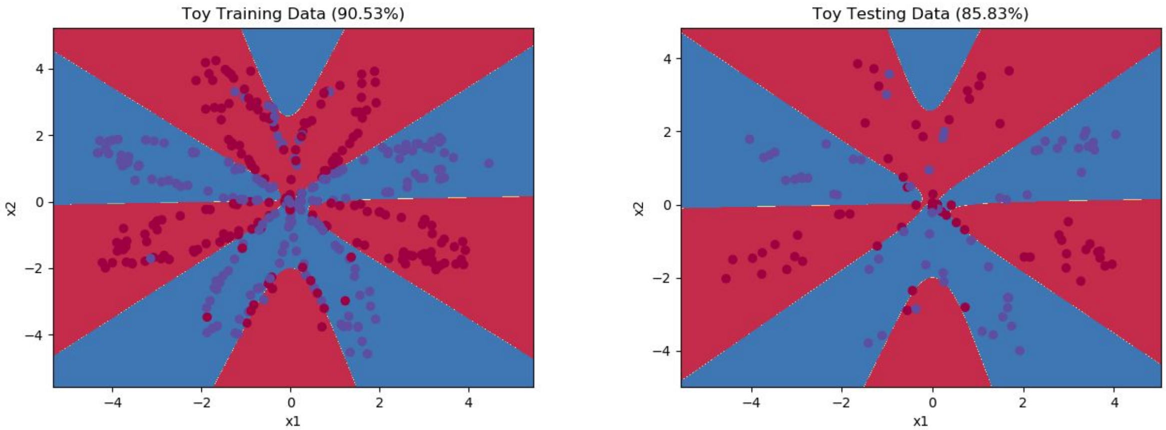

Toy Testing Data (20 hidden units) | Train: 87.89 | Test: 82.50
x1
x2


Toy Testing Data (40 hidden units) | Train: 88.16 | Test: 82.50
x1
x2

**v.)** Network with Two hidden layers of 5 - 5 neurons each.

After training for 5485 epochs, the accuracy is:

Training Accuracy 90.53% and Testing Accuracy 85.83%




We observe that this is certainly better than the previous cases, where there wasn’t a lot of change in the accuracies.

## c.) MNIST Handwritten Digits recognition - Only 6 & 8

### i.) libsvm & single perceptron

**SVM with linear kernel** and C = 1 results in:

Training Accuracy = 99.87% (9987/10000)
Testing Accuracy = 98.8333% (3558/3600)

Since linear SVM is able to do such a good job at classification, it shows that the data is inherently linear and we expect a single perceptron to be good enough too.

**Single Perceptron** results in:

Training Accuracy = 99.08%
Testing Accuracy = 99.0%

This takes 15 epochs to converge to a final average error of 0.00377 (with threshold $10^{-5}$.)

### ii.) A network with 100 neurons in hidden layer

The stopping criteria used was the change in average error <= $10^{-5}$ (it was not converging on $10^{-6}$ even after 100 epochs.)

Training Accuracy 99.27%
Testing Accuracy 99.10%

This runs for 31 epochs, resulting in a final average error of 0.00326.

### **iii.)** Effect of ReLU activation unit in hidden layers

The accuracies obtained when we use ReLU are:

Training Accuracy 99.49
Testing Accuracy 99.13

After only 12 epochs, the network converges with a threshold of $10^{-5}$, the total average error is 0.00238.

Even though there is not a marked improvement in accuracy (because we're already at ~99%), we observe that the network converges faster than it did before, this is perhaps because of the fact that it is simpler (faster) to compute and its gradient does not saturate (always stays 1 for positive input.)