

## Overview

Neural networks are excellent classification tools, and a typical back propagation neural network is known to be a good function approximator. As such, neural nets have been shown to provide as useful tools in predicting trends in financial markets. While financial markets are as a whole unpredictable, driven by many dynamic inputs and fluctuations, neural networks are often able to recognize patterns in the history of financial data for a single security. The assumption is, then, that if general patterns are present in historical financial data, and a system can approximate this pattern as a function, then it may be possible to predict future trends by applying this learned function. In research, many such models claim accuracy in the 90% range, however, these results are difficult to reproduce in practice, with a 60% prediction accuracy to be considered very good [1.]

Neural networks belong to a general class of algorithms called Supervised Learning algorithms because they are corrected (or supervised) against errors in the classification. Another supervised learning algorithm is a Decision Tree Learner. This work will exercise a simple back propagation neural network and a decision tree (with both GINI and GAIN attribute splitting measures.) Each algorithm will be tested against two classification problems.

We shall see that Decision trees perform well at classification problems that have discrete valued inputs. Neural networks, on the other hand, can take continuous values as input. Both are “eager learning” algorithms in that they perform their search through the hypothesis space up front, and once trained, can provide answers to a query very quickly.

## Classification Problem #1 – Stock Prediction

This problem consists of a set of historical price data for a single stock, Time Warner Inc (TWX). While this is not a novel machine learning problem, however; I find it very interesting that an algorithm might find patterns in price data, especially as these algorithms tend to not perform well in practice [1]. In addition I plan to pursue related work for the semester group project.

Twelve years of raw historical data were pulled from the Yahoo Finance website [7] for this classification problem. The goal of this classification is to take as input the closing prices of a stock and classify the stock price as increasing or decreasing in two days time.

### Data Format

The original data was transformed to pull time series information into each row of data. In the transformed data set, each row consisted of input values as presented in Tables 1 and 2. Two output values, increase and decrease, were used. These were derived from the change in price at day t+2 (two days later.)

<i>Attribute</i>	<i>Description</i>	<i>Range</i>
t-20	Percent change in price from 20 days prior	-1.0 to 1.0
t-10	Percent change in price from 10 days prior	-1.0 to 1.0
t-2	Percent change in price from 2 days prior	-1.0 to 1.0
t	Percent change in price from day before	-1.0 to 1.0
High low swing	Normalized difference between high and low price that day	0 to 1.0
volume	Normalized trading volume that day	0 to 1.0

**Table 1: Stock Data Input**

PI(t-20)	PI(t-10)	PI(t-2)	PI(t)	PI(t+2)	HLSwing	Volume	increase	decrease
0.046875	0.032471	0.020675	-0.014706	0.016524	0.033000	0.172282	0.016524	0.000000
0.051934	0.047881	0.024207	0.035909	-0.003151	0.049000	0.311859	0.000000	0.003151

**Table2: Example Input**

### *Data Set Size*

The twelve year data set was split into separate training and testing data sets with a 2/3, 1/3 distribution. The first 1989 rows of data consist of the training set (prices from 03/92 – 02/00). The last 970 rows of data consist of the test set (02/00 – 12/03.)

## **Classification Problem #2 – Handwritten Digit Recognition**

This problem attempts to classify a condensed image of a handwritten digit. This is of interest as a classical machine learning application. The digits range from 0 to 9. The data set was obtained from Alpaydin, et al [6]. To create this data set, normalized 32 x 32 bitmaps of handwritten digits (gathered from 43 people) were transformed into 8x8 input matrices.

### *Data Format*

The data format consists of 64 input attributes, representing the pixels of the image. Each attribute ranges from 0 to 16. The output is in the range 0 to 9, and represents the image as drawn in the bitmap.

### *Data Set Size*

The training data set consists of 3823 images and the testing data set consists of 1797 images.

## **Neural Network Implementation**

I implemented the neural network for this project, using C++. It is a simple back propagation network with adjustable learning rates and momentum. The topology is a single input layer, a configurable number of hidden nodes and a single output layer. The hidden nodes are fully connected to the input and output layers. The algorithm used is from the Mitchell text [3]. All nodes and weight values are randomly initialized. The logic for back propagation was based on the code accompanying the text [4].

## **Decision Tree Implementation**

The decision tree implements the ID3 algorithm from Russell, et. Al [5]. In addition to the information gain splitting criteria, the gini index is also implemented.

The decision tree code was implemented in Mat lab and the implementation was borrowed from colleagues[8]. Alden implemented the decision tree learner in Mat lab, and Lamstein implemented the framework for loading and analyzing the data files. All analysis work, manipulation and further configurations are mine own.

My initial approach to the decision tree was to modify an existing implementation in LISP, from a previous project in order to support multiple attribute values (rather than just binary trees) as well as the gini attribute splitting rule. However I ran into difficulties with multiple attribute values in the data sets and made a decision to forego this approach in the interest of time. This code is, nevertheless included with the project code as reference, and a snippet relating to gini attribute splitting is included below.

```
(defun gini-gtg593n ( classVals classes attrValue attribPos examples
  subsetSize )
  (setf sum 0)
  (setf Pj 0)

  (loop for cVal in classVals do
    (setf Pj (classFreq-gtg593n attrValue cVal attribPos examples
      classes subsetSize))
    (setf sum (+ sum (* Pj Pj))))
  (- 1 sum)
  )
```

## **Results and Analysis**

### **Neural Networks**

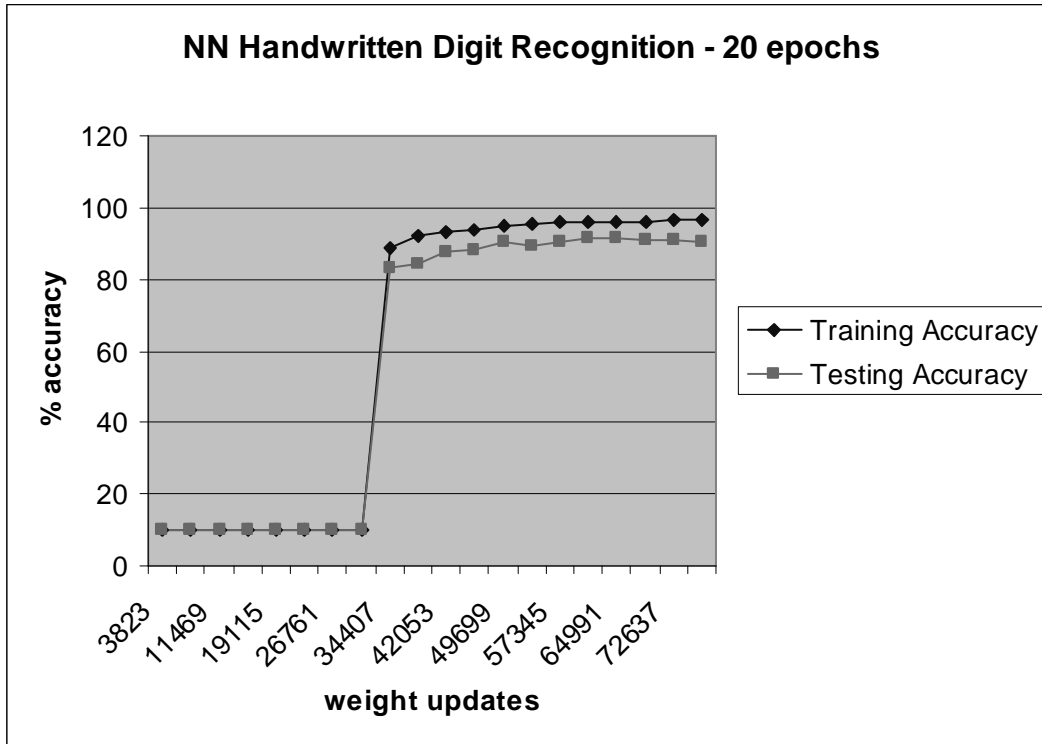
The implementation of the neural network ran the same training data multiple times, to allow the weights more updates for better convergence. Each run of the training data is referred to an epoch. By varying the number of epochs, we can easily vary the training data size (effectively running the training data multiple times.) Also, as a rough cross-validation mechanism, at the end of each training run, the end of each epoch, the network is tested, *but not updated* against the test set. The back propagation of weight values only occurs as a result of training error, not testing error. As such, we can see plot the training data accuracy against the test data accuracy for the number of weight updates.

### **Neural Network – Handwritten Digit Recognition**

Various combinations of learning rate, momentum, number of epochs and number of hidden nodes were tried. Some of these examples are listed below. However, in the best case, the neural network converged to approximately 93% accuracy against the test data set, refer to Run 2.

Early on in the training, at about 34,000 weight updates, the network quickly converges to an 80% accuracy level and slowly increases from there. It is of interest that the jump in accuracy occurred so steeply, and suggests that the network might have been caught in a local minima.

**Run 1**

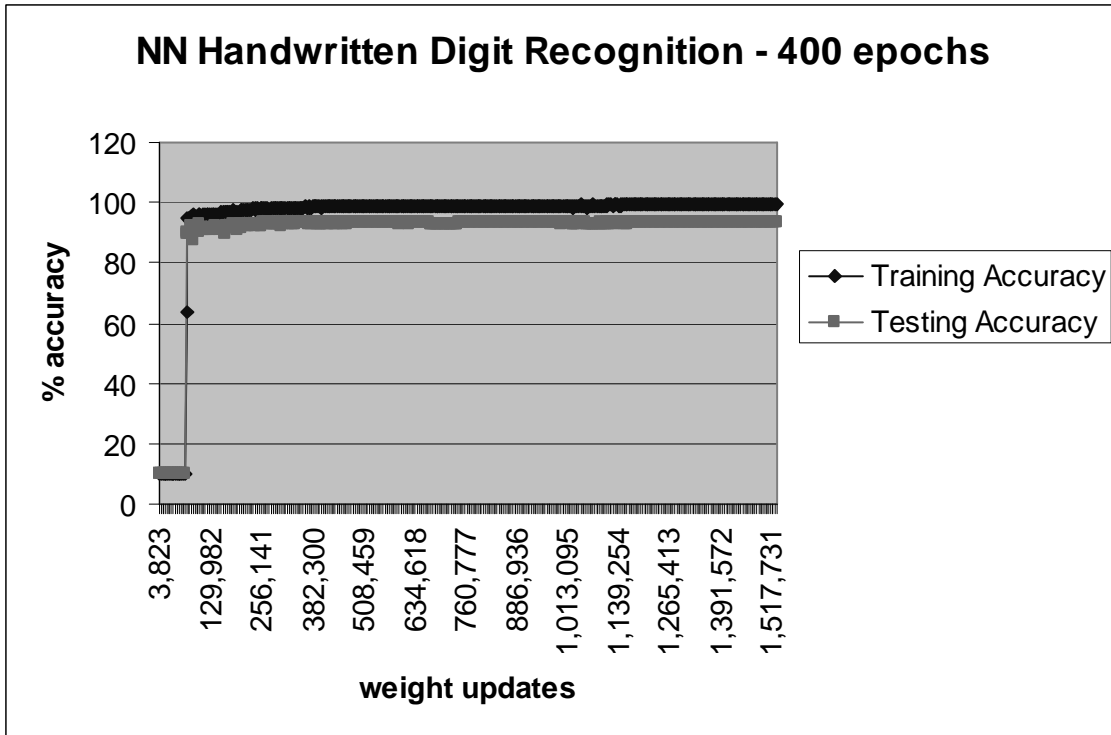


<b>Accuracy</b>	<b>90.5</b>
<b># Epochs</b>	20
<b>Learning rate</b>	0.5
<b>Momentum</b>	0.3
<b>Hidden units</b>	30

The output value of the network is defined by the 10 output values (representing the digits [0..9]). The maximum of these values is defined as the output. In the training data, these are discrete outputs with a 1 representing the output for that digit and a 0 representing all others. As the network converges, these values grow closer to the trained discrete limits of 1 and 0.

When the network was run for a large number of weight updates, after the initial convergence, it only slowly increased until it reached a maximum performance of 93.4%, refer to Run 2. This test is of interest to determine if after a large number of runs, the performance would improve further. However, the performance remained constant and could have been stopped sooner to prevent over fitting.

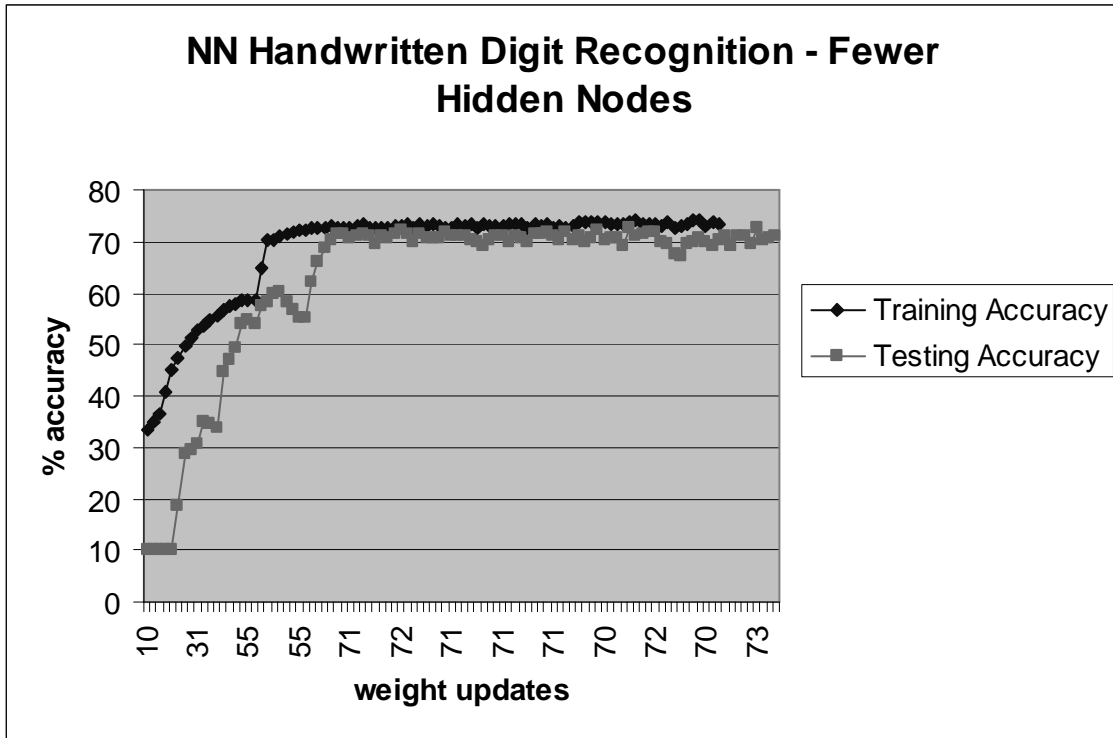
Run 2



<b>Accuracy</b>	<b>93.4</b>
<b># Epochs</b>	400
<b>Learning rate</b>	0.5
<b>Momentum</b>	0.3
<b>Hidden units</b>	30

I found that the network performs well for both models with a weight value of 0.5, and with a number of hidden nodes close to 30. With much more than 30 hidden nodes, the network fails to converge at all. In Run 3 below, when we lower the number of hidden nodes to 3, the network takes longer to learn and has a more gradual learning curve. Unfortunately, the network does not converge to as high of an accuracy.

Run 3



Accuracy	72.7
# Epochs	100
Learning rate	0.5
Momentum	0.3
Hidden units	3

### Neural Network – Stock Prediction

Various combinations of learning rate, momentum, # of epochs and number of hidden nodes were tried. Some of these examples are listed below. However, in the best case, the neural network converged to approximately 97% accuracy against the test data set. As was the case with both classification problems, if the number of hidden nodes exceeded 30, the model failed to learn at all, as all output values immediately converged to one and stayed there. Additionally, if the learning rate was too low, below 0.3, the model did not converge effectively. Further, for the stock prediction model, the number of epochs needed to be very high, in the best case 300, to give the model time to converge.

The network outputs a single positive value for the classification that the stock will increase in two days time, and a single positive value for the classification that it will decrease. The larger of the two values is the classification. For instance, if the increase output value is larger than the decrease output value, the network classifies the data as being in the *increase* classification. Accuracy in the network is defined as the percentage of times that the network classifies an increase (or decrease) against the actual result of an increase (or decrease.) Magnitude or confidence of classification is not considered. An example of the raw output data is displayed below in Table 3, with the network output values, the correct answer, and whether the network classified correctly.

## CS7641 Machine Learning Assignment #1 – Supervised Learning

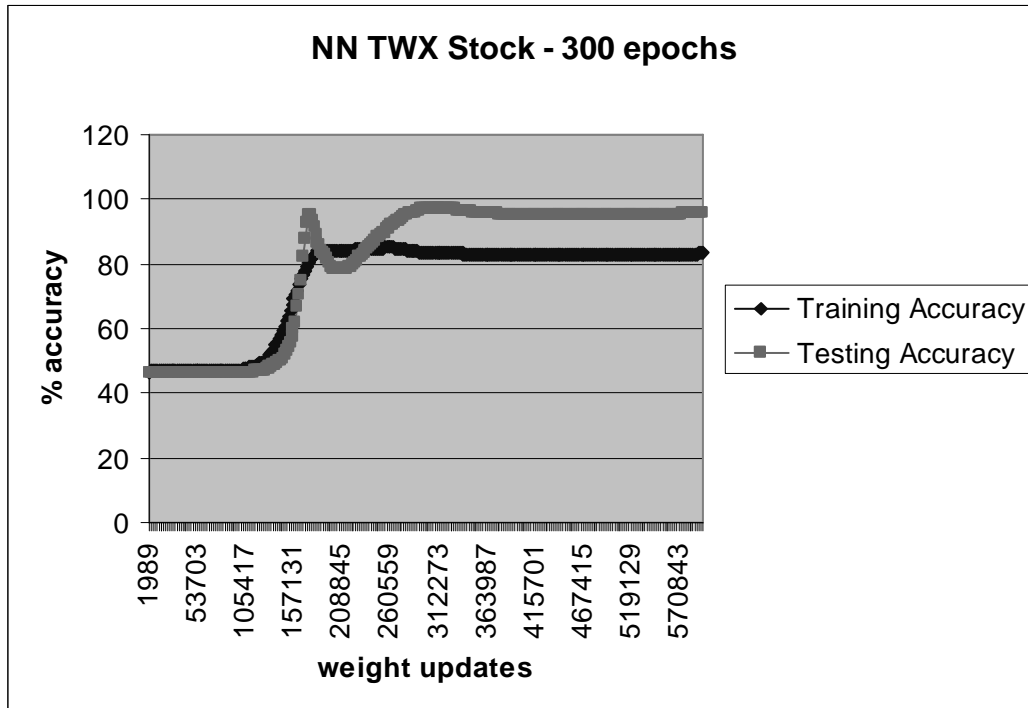
Charles Pippin, cepippin@cc.gatech.edu

Out: 0.034781 0.019219      Act: 0.016524 0.000000, c: yes  
Out: 0.033140 0.017760      Act: 0.000000 0.003151, c: no  
Out: 0.035958 0.018771      Act: 0.020675 0.000000, c: yes

**Table3: Example Output**

Given more time, it would be interesting to run the network in a real time simulation with an initial pool of virtual funds to gauge its performance in practice.

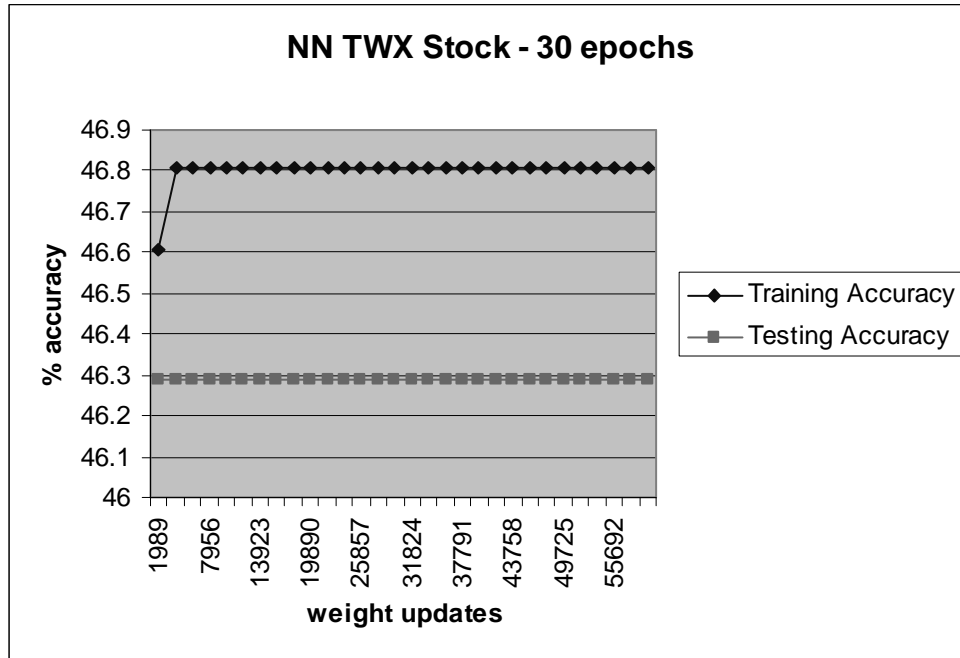
### Run 1: Best



<b>Accuracy</b>	<b>97.3</b>
<b># Epochs</b>	<b>300</b>
<b>Learning rate</b>	<b>0.5</b>
<b>Momentum</b>	<b>0.3</b>
<b>Hidden units</b>	<b>30</b>

The dip in the above plot of the accuracy against the test set is of interest. If training had stopped at 95% accuracy, at the sight of the first dip, the better accuracy of 97% would not have been seen. This is why it is important to test with different values for the number of epochs, allowing for more weight updates. The network also converged best with a learning rate at 0.5 and momentum at 0.3

**Run 2: Non-Convergence: Too Few Updates**



<b>Accuracy</b>	<b>97.3</b>
<b># Epochs</b>	30
<b>Learning rate</b>	0.5
<b>Momentum</b>	0.3
<b>Hidden units</b>	30

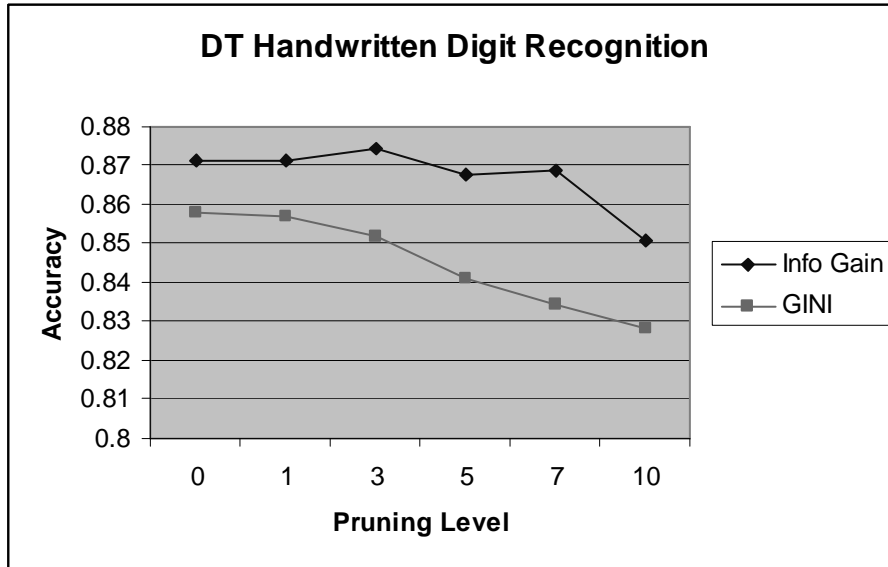
As is alluded to above, the number of weight updates (via epochs) is important. In this next run, the network never converged; it never “learned” because it was not given enough examples during training. Another interesting dimension is in varying the number of hidden units. Past about 30 hidden units, the network fails to converge at all for either classification problem.

**Decision Trees**

**Decision Tree – Handwritten Digit Recognition**

It is interesting to note that the decision tree, with information gain splitting, performed close to the level of the neural network, with an 87.4% accuracy compared to an 93.4% accuracy in the network. More pruning of the trees in general did not improve the results, except for a slight improvement at pruning level 3 for information gain splitting. The information gain split outperformed the GINI split in all instances.

**Attribute Splitting Rule Performance**



**Decision Tree – Stock Prediction**

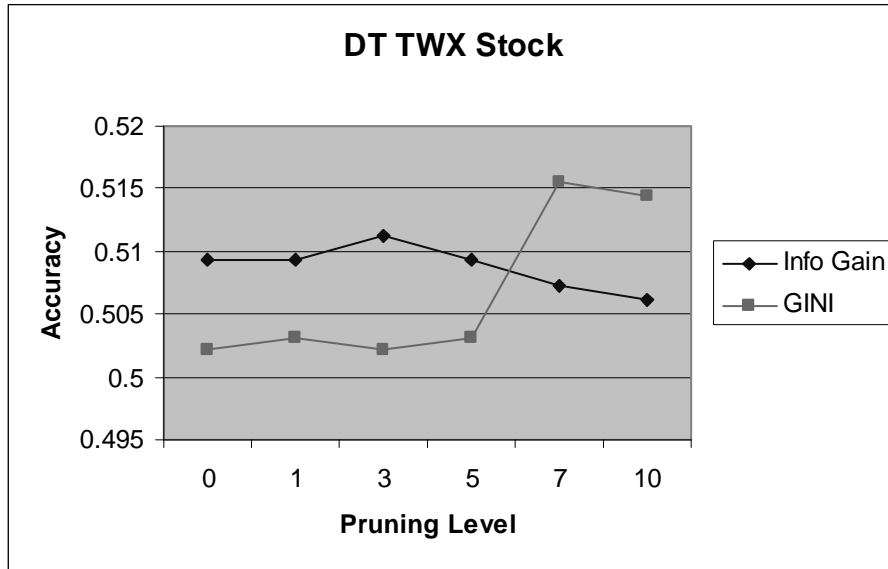
Compared to the neural network, the decision tree for stock prediction performed very poorly. In fact, its performance hovered slightly above 50% for both splitting functions. Notably, as more pruning of the tree was performed, the gini index method outperformed the information gain method.

As the decision tree algorithm requires discrete valued inputs, it was necessary to transform the real valued financial data into discrete values for input into the decision tree. As part of this process, I analyzed the input data to the neural network to determine logical intervals for each attribute. I then transformed the data into these intervals. Table 4 below lists examples of the data before and after conversion. This suggests that the real to discrete value interval selection, as well as the granularity of the division is key to the performance of the decision tree learner.

PI(t-20)	PI(t-10)	PI(t-2)	PI(t)	PI(t+2)	HLSwing	Volume	increase	decrease	
0.046875	0.032471	0.020675	-0.014706	0.016524	0.033000	0.172282	0.016524	0.000000	8,8,7,4,2,8,1
0.051934	0.047881	0.024207	0.035909	-0.003151	0.049000	0.311859	0.000000	0.003151	8,8,7,8,2,9,-1
0.028539	0.011557	-0.013419	-0.011296	0.020675	0.058000	0.233181	0.020675	0.000000	7,7,4,4,2,8,1

**Table4: Transformed Stock Input**

**Attribute Splitting Rule Performance**



**Conclusions**

The neural network outperformed the decision tree implementation in both problems, especially the stock classification problem. However, the neural network did take longer to train than the decision tree. The decision tree likely performed close to the level of the neural network for the handwritten digit recognition problem because the input and output values are all discrete, and therefore they did not have to be approximated by transforming them before loading them into the decision tree.

Similarly, the decision tree was not able to perform well for the financial dataset. It is my conclusion that this is because the financial data is continuous in nature and even when converting it to discrete values, much of the information in the attributes is lost, and the learner is unable to generalize. Surprisingly, the neural network performed with high accuracy in the financial classification problem. This is an area that has been identified for future work in the semester project.

## References

- [1] JingTAO YAO and Chew Lim TAN. “Guidelines for Financial Forecasting with Neural Networks”
- [2] Steven Walczak. “An Empirical Analysis of Data Requirements for Financial Forecasting with Neural Networks”, in *Journal of Management Information Systems*, Vol. 17, No. 4. Spring 2001.
- [3] Mitchell, Tom M. Machine Learning. McGraw-Hill. 1997.
- [4] Mitchell, Tom M. Software and Data from Machine Learning.  
<http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/ml-examples.html>
- [5] Russell, Stewart and Norvig, Peter. Artificial Intelligence A Modern Approach. Prentice Hall. 2003.
- [6] Data Set: Alpadin, E. and Kaynak, C. *Optical Recognition of Handwritten Digits*:  
<http://www.ics.uci.edu/~mlearn/MLSummary.html>
- [7] Data Set: *Yahoo Finance, TWX*:  
<http://finance.yahoo.com/q?s=twx>
- [8] Decision Tree Matlab code assets. Lamstein, Ari and Alden, Patrick. Georgia Tech College of Computing.