# Time Series Forecasting Using Neural Networks

# Introduction:

A time series [3] is a sequence of [data points](http://en.wikipedia.org/wiki/Data_point), typically consisting of successive measurements made over a time interval. Examples of time series are ocean [tides](http://en.wikipedia.org/wiki/Tides), counts of [sunspots](http://en.wikipedia.org/wiki/Sunspots), and the daily closing value of the [Dow Jones Industrial Average](http://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average). Time series are very frequently plotted via [line charts](http://en.wikipedia.org/wiki/Line_chart). Time series are used in [statistics](http://en.wikipedia.org/wiki/Statistics), [signal processing](http://en.wikipedia.org/wiki/Signal_processing), [pattern recognition](http://en.wikipedia.org/wiki/Pattern_recognition), [econometrics](http://en.wikipedia.org/wiki/Econometrics), [mathematical finance](http://en.wikipedia.org/wiki/Mathematical_finance), [weather forecasting](http://en.wikipedia.org/wiki/Weather_forecasting), [earthquake prediction](http://en.wikipedia.org/wiki/Earthquake_prediction), [electroencephalography](http://en.wikipedia.org/wiki/Electroencephalography), [control engineering](http://en.wikipedia.org/wiki/Control_engineering), [astronomy](http://en.wikipedia.org/wiki/Astronomy), [communications engineering](http://en.wikipedia.org/wiki/Communications_engineering), and largely in any domain of applied [science](http://en.wikipedia.org/wiki/Applied_science) and [engineering](http://en.wikipedia.org/wiki/Engineering) which involves [temporal](http://en.wikipedia.org/wiki/Time) measurements.

Time series analysis comprises methods for analysing time series data in order to extract meaningful statistics and other characteristics of the data. Time series forecasting is the use of a [model](http://en.wikipedia.org/wiki/Model_%28abstract%29) to predict future values based on previously observed values. While [regression analysis](http://en.wikipedia.org/wiki/Regression_analysis) is often employed in such a way as to test theories that the current values of one or more independent time series affect the current value of another time series, this type of analysis of time series is not called "time series analysis", which focuses on comparing values of a single time series or multiple dependent time series at different points in time.

I will be concentrating here on the Neural Networks for the time series forecasting. I will use 3 different methods.

1. First Order Neural Networks with no regularization.
2. First Order Neural Networks with regularization.
3. Second Order Neural Networks using LM.

# Process:

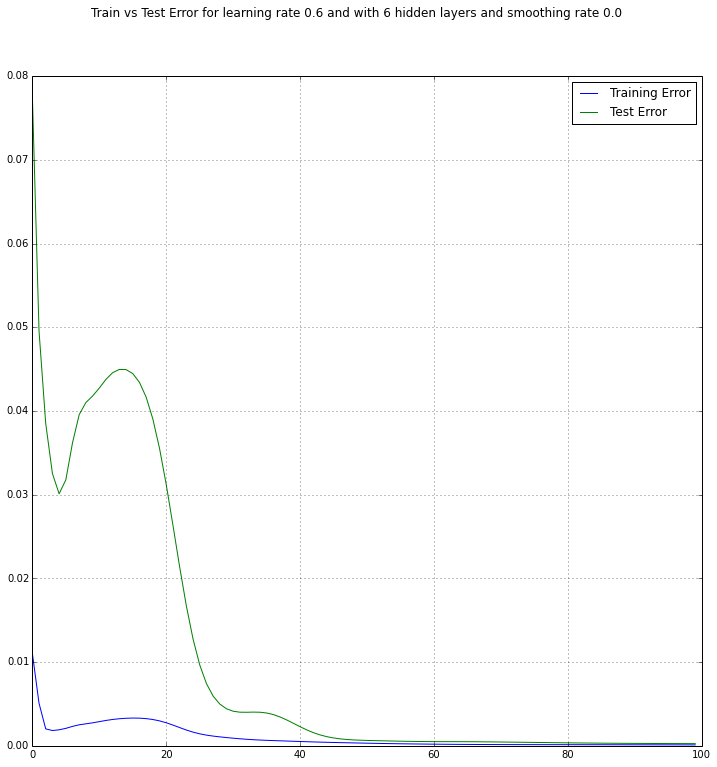
As mentioned above I have used 3 different neural network algorithms to train the time series. I have not used any early stopping as I was looking to see how it would behave. Also, I played with number of hidden layers, learning rate and the weight decay parameters (for regularization). This is a single hidden layer (with multiple neurons) fully connected network. I have used python as the programming language. Also, while predicting future value for time series I have used actual values till that day and then predicted on next day. I could have done it other way as well e.g. I could have predicted say for Jan 1 2015 based on data till Dec 31 2014 and then for predicting Jan 2 2015, I could have been used the predicted value of Jan 1 2015 rather than actual value of the Jan 1 2015. It also looks like that there is no noise in the data and that is why test and train data seems to fit very well.

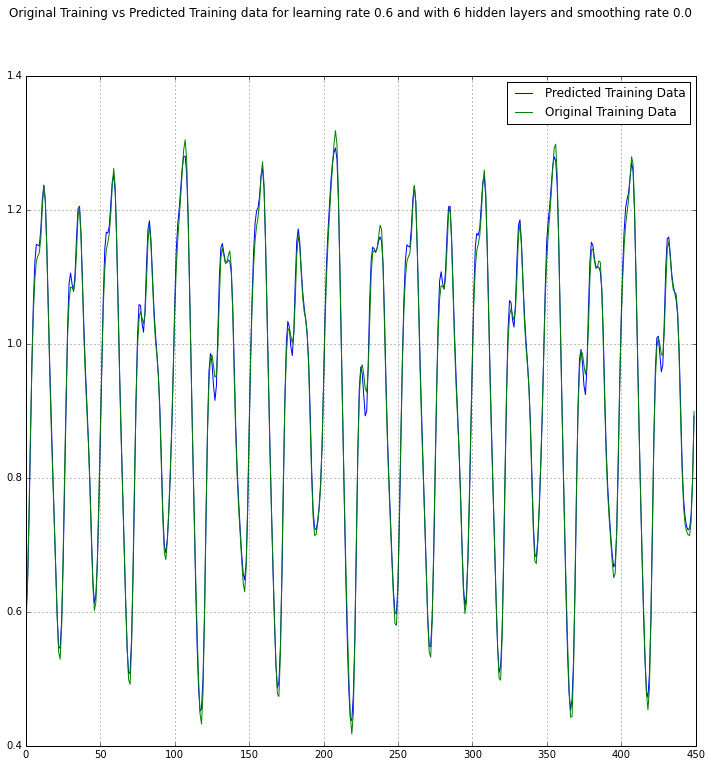
I have not used any biases either for the hidden layer or for output layers. I will be presenting only selected graphs here but you can look all the graphs in github or in attached directory. These documents are. “Output\_figures.pdf” This has all the first order graphs with and without regularization. “Output\_figures\_second\_order.pdf” This has all the second order graphs. The code is quite flexible and you can change the different parameters like number of hidden layers, training examples, number of previous values to determine future value of time series, learning eta and smoothing parameters.

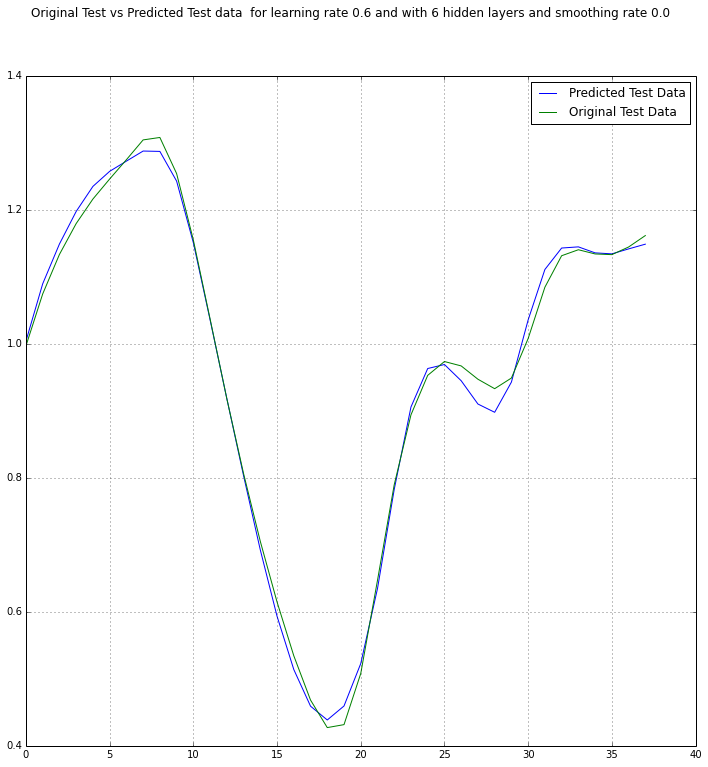
## First Order Neural Networks with no regularization:

The code for this is in the file NN.py and the TS\_first\_order.py. This one is generic. Passing wgt\_decay as 0.0 will make sure that no regularisation is being used. I have run it for different number of hidden layers and different values of learning eta. Below shown are some graphs.

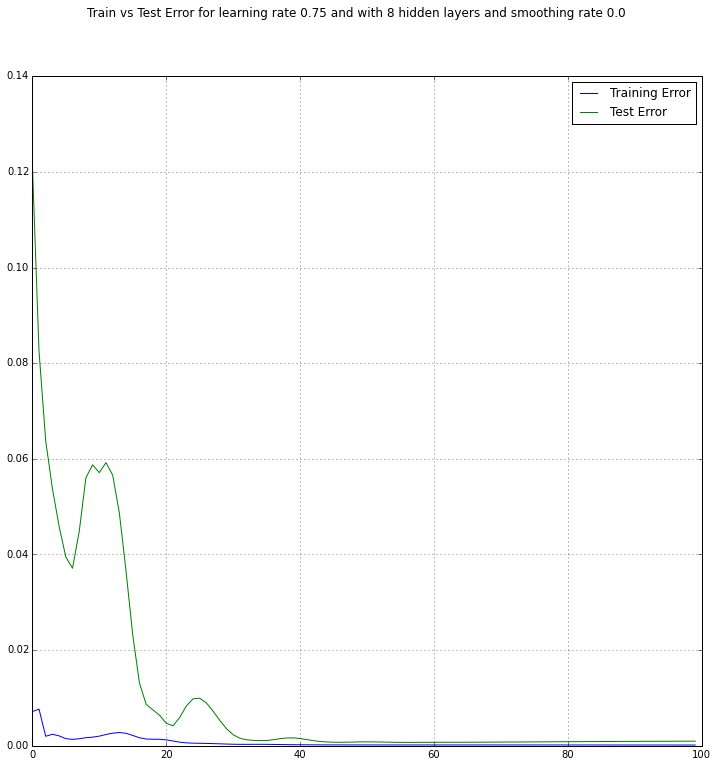
Below is a graph showing 3 hidden neurons and smoothing rate of 0.0 (no regularization) and a learning rate of 0.6. Around epoch 50 the error came to its minimal and the fit is not so great but it is quite good fit even on test data.

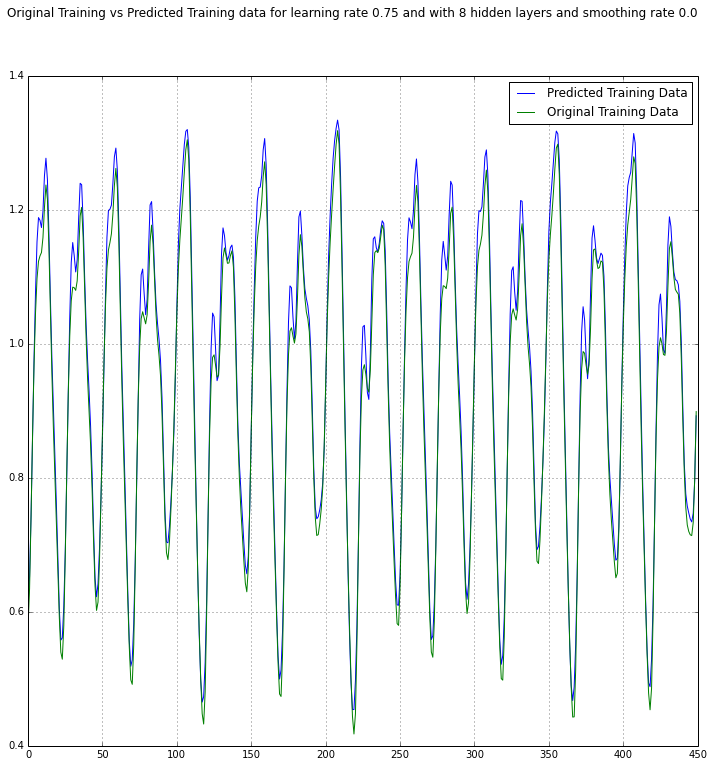


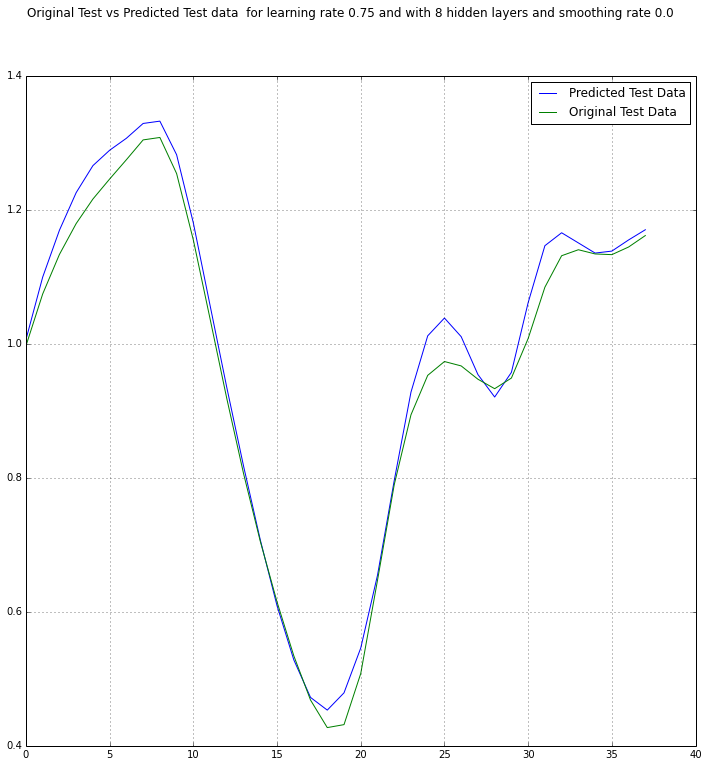




Now I will show with 8 hidden layers and learning rate of 0.75. Here again the fit is quite good and training data seems to over fit but the test data fit is quite good as well. This might be because there is no noise in the data. The error was at its minimal around 50th epoch.



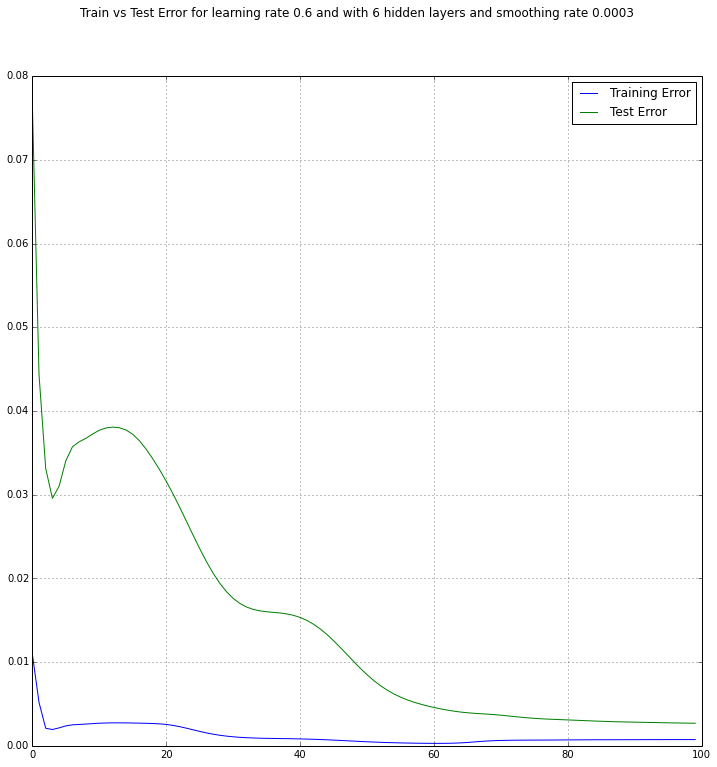


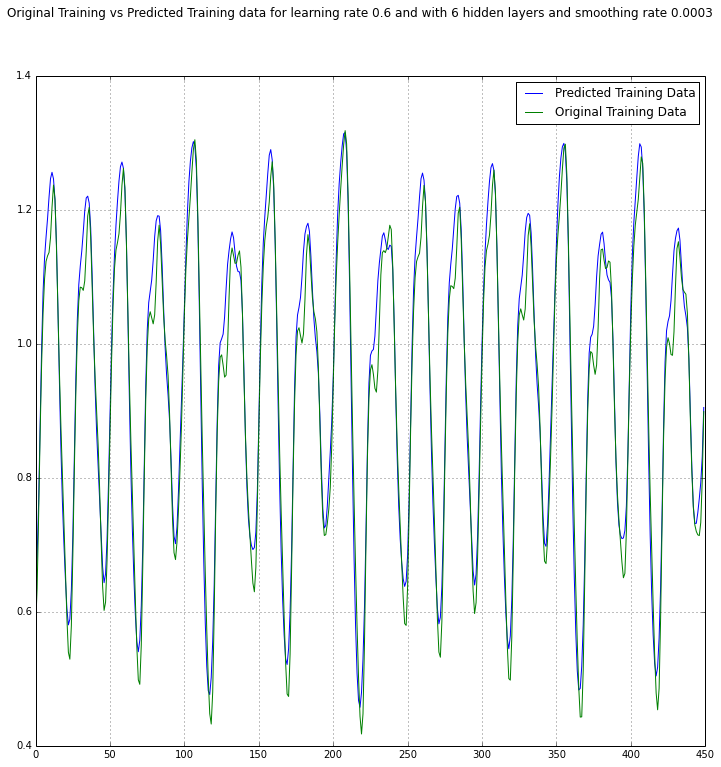


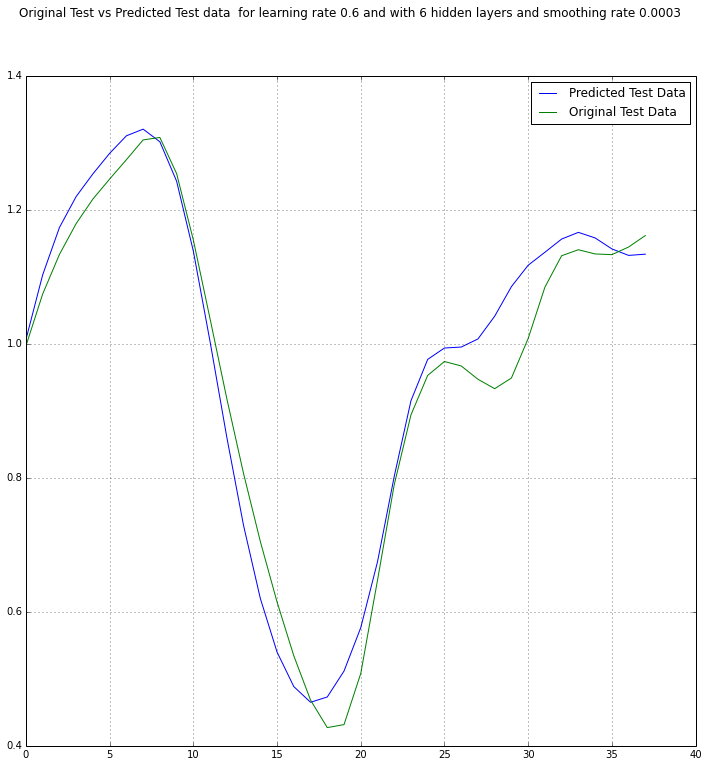
## First Order Neural Networks with regularization:

The code for this is in the file **NN.py** and the **TS\_first\_order.py**. This one is generic. Passing wgt\_decay greater than 0.0 will make sure that regularisation is being used. I have run it for different number of hidden layers and different values of learning eta an different values of weight decays.. Below shown are some graphs. Usually regularization is quite good when there is some noise in the data. The data is quite fine and there doesn’t seems to be any noise and thus any regularization will make curves more smoother and thus will make the data fit less well as compared to unregularized in this data set.

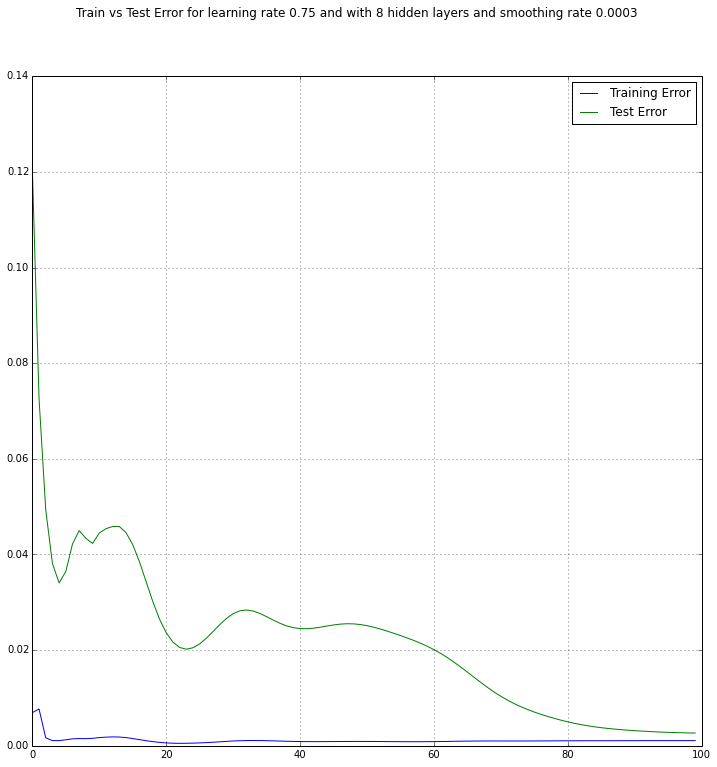
Below is a graph showing 3 hidden layers and smoothing rate of 0.0003 and a learning rate of 0.6.Error was minimal around 50th Epoch and then it increased and finally went down again at epoch 80. The fit is good but it is smoother and thus it is less fit than without regularization.

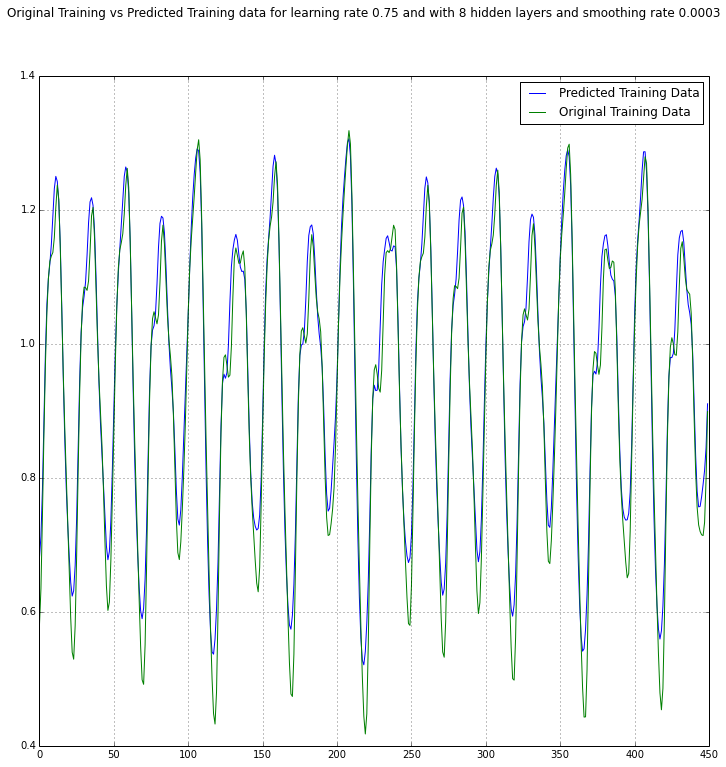


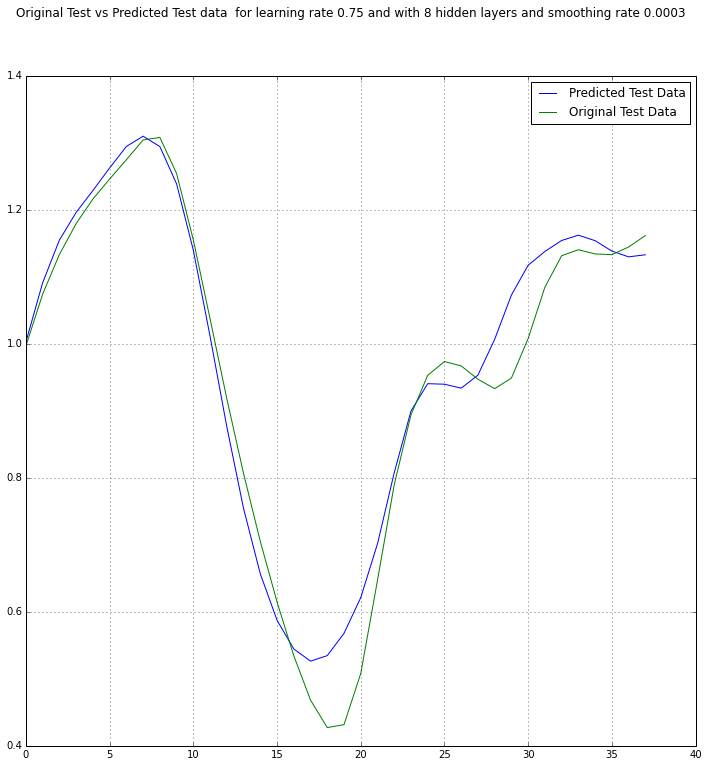




Now I will show with 8 hidden layers and learning rate of 0.75. and smoothing rate of 0.0003 The error was minimal at epoch 20 and then it increased and finally went stabilized after epoch 40. The fit is quite good but as there is smoothing in the curve this fit is not better than the unregularized one.



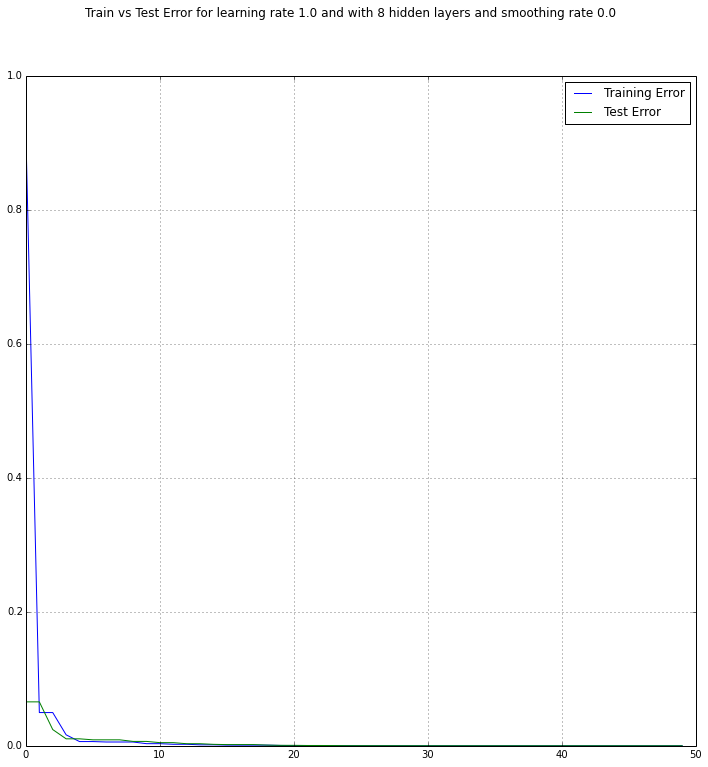


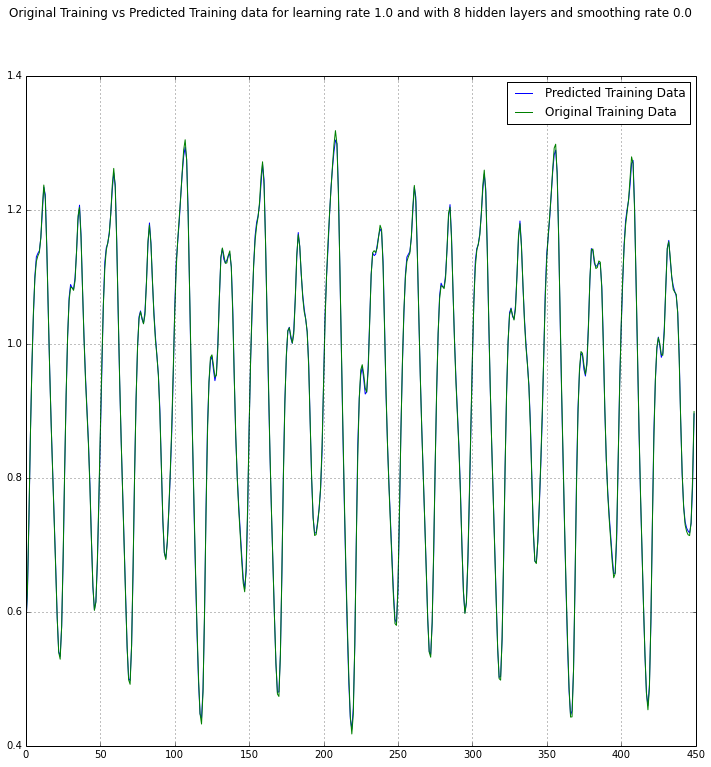


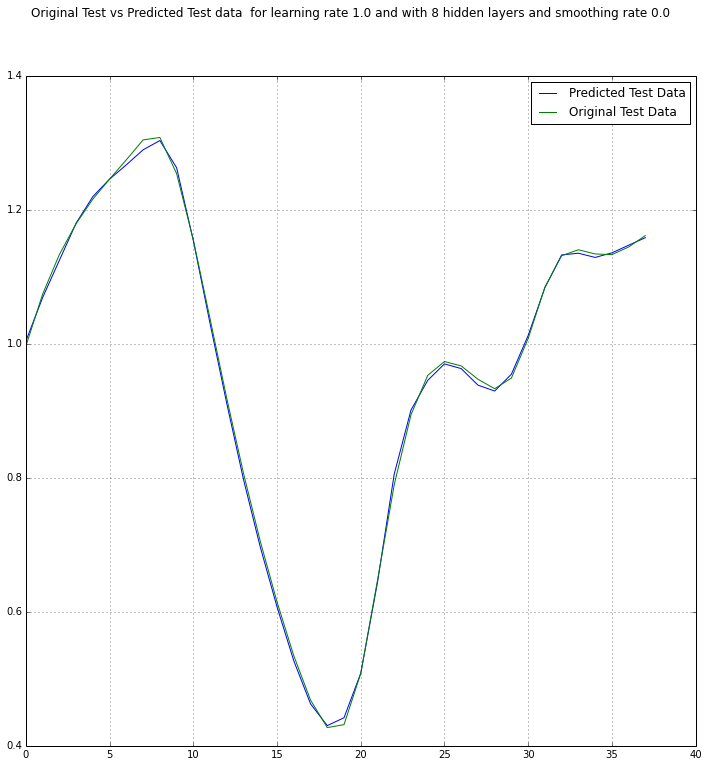
## Second Order Neural Networks using LM:

The code for this is in the file **NN\_2\_1.py** and **TS\_second\_order.py**. Here I have used a single value of the learning\_eta(This is mu and this value will be multiplied with identity matrix and will be added to Hessian and then it will be reversed. Here I experimented with different values of the hidden layers. Graphs are shown below. I have not used any early stopping as I was looking to see how it would behave. I have followed my approach as well as approaches discussed in [1] and [2].

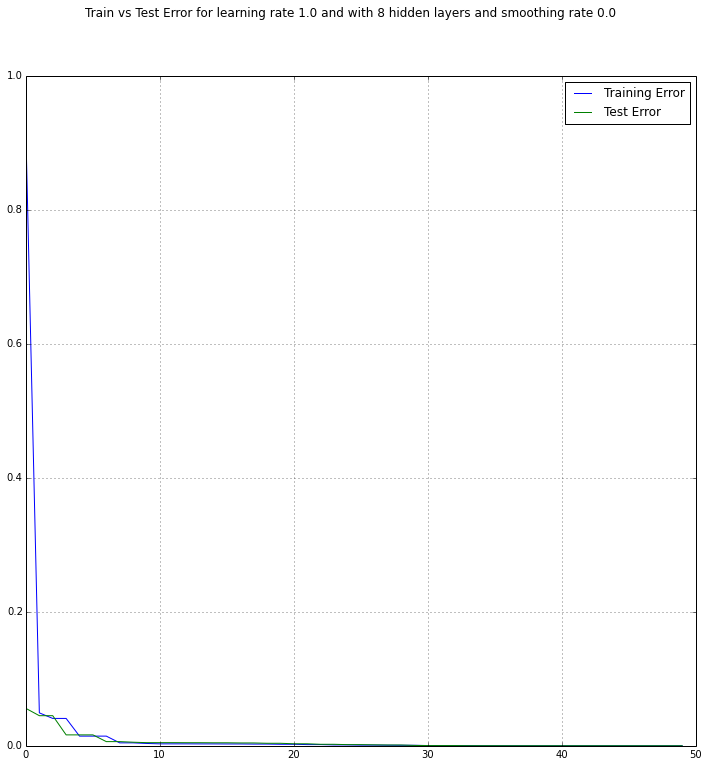
I have chosen the number of hidden layers as 8 to show the graphs. As you could see, from the below graph that the error dropped to very minimal levels in 3rd-4th Epoch. After 9th Epoch it reached its minimal.

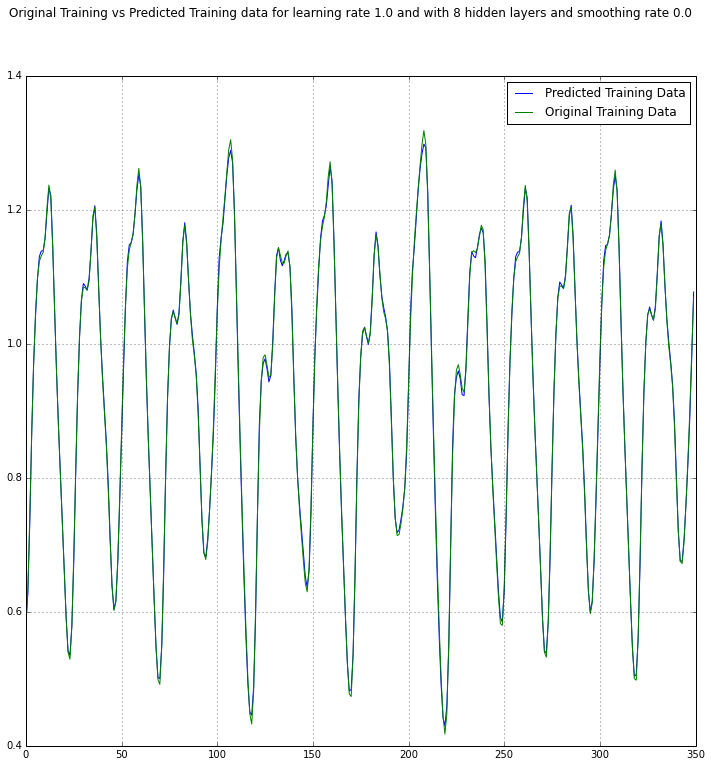


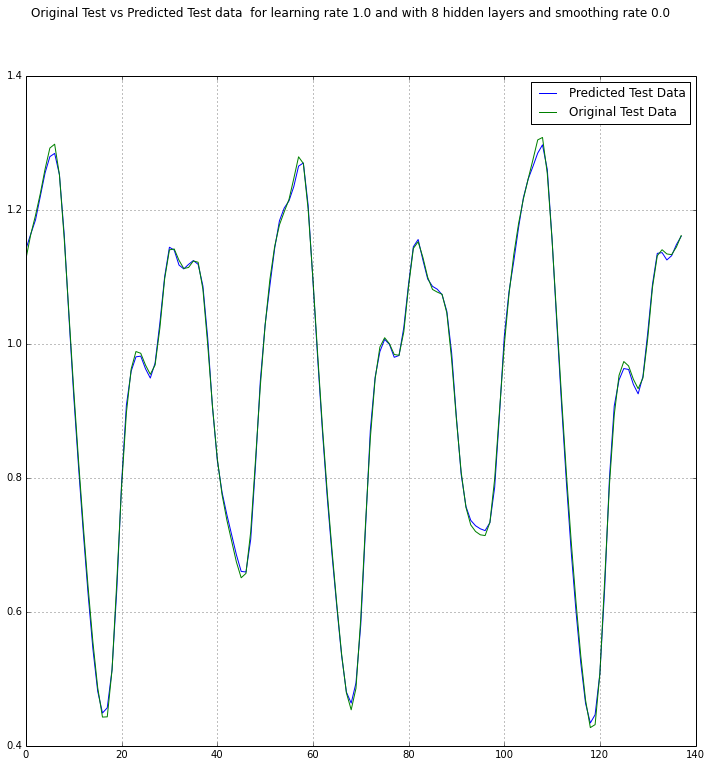




Below are graphs when I used 350 examples for training and rest for the test. I wanted to see how would it look like. The fit is similar when I used 450 examples. Thus, the neural network is working fine.







# Conclusion:

I have tried 3 different methods for time series forecasting. The Second Order LM training gave really good fit and error was minimized between epochs 3 and 8, which is quite fast as compared to other 2 methods where it was 40 or more epochs. Also, regularization provided less fitted data and smoother curve and was not that beneficial in this case as the data did not have any noise. But, using regularization is much better way to handle the data which over fits. I will use some of this code especially order 1 with regularization in my final project when I will predicate the price of CPI or some other items which are being used to find CPI.

Full code is in github [4].

# References:

1. <http://www.eng.auburn.edu/~wilambm/pap/2011/Neural%20Network%20Training%20with%20Second%20Order%20Algorithms.PDF>
2. <http://www.eng.auburn.edu/~wilambm/pap/2011/K10149_C012.pdf>
3. <http://en.wikipedia.org/wiki/Time_series>
4. <https://github.com/darshanmeel/NN2>
5. <http://research.microsoft.com/en-us/um/people/cmbishop/prml/>
6. <http://homepages.gold.ac.uk/nikolaev/cis311.htm>