Day Ahead Load Forecast in ISO New England Market using Artificial Neural Network

6th Semester Minor Project

Final Report

Mentor: Dr. M. M. Tripathi

Ushmita Seth 2K14/EL/100

Shruti Sinha 2K14/EL/092

Vartika Chaudhary 2K14/EL/104

Acknowledgement

We would like to thank Dr. M. M. Tripathi [Department of Electrical Engineering, Delhi Technological University] for taking out the time to guide us through this project. We would also like to thank the authors of various forums on basics of Python and creators of the website Datacamp for painstakingly creating the content we actively referred to.

Dr. M. M. Tripathi

Introduction

European transparency regulations require transmission system operators to publish data about the electricity market to foster an even playing field between all market makers. The available data allows for the maturation of the electricity market and encourages analysis of data to improve the generation, usage and management of electrical power.

Our project is based upon the New England ISO market 2012- 2014 yearly data which gives the total load on the New England electric grid measured in hourly intervals for the past several years. The project uses a feed forward neural network to predict hourly load based on hour of the day, dew point temperature, dry bulb temperature and lagged load. The results obtained will be practically beneficial as utilities can use the predicted values to generate an adequate amount of energy to avoid grid outages and electrical losses as well as construct dynamic pricing schemes based upon future load.

Load Forecasting

The management of power systems is a complex task for transmission operators and is heavily reliant on knowledge of future energy demand. A model that can accurately forecast load is essential in energy generation as the predicted load can determine which devices should be operated in order to meet demand. Failure to generate an adequate amount of energy can lead to grid failures and conversely oversupply can lead to a waste of energy and resources. With the advent of decentralized electricity markets it is essential to develop accurate pricing protocols based on current demand for which an understanding of future demand is critical. We aim to use artificial neural networks in an attempt to outperform the orthodox approaches.

Load forecasting is a critically important task for utility companies since it is crucial in determining the amount of electricity that the company should provide. Failure to meet the demand can result in the overloading of network components and widespread grid outages, both of which are financially expensive and damaging to a utility company’s reputation. Estimating future network flow accurately can allow providers to plan electricity generation, develop grid infrastructure and respond to varying frequencies so as to avoid these outcomes and improve overall network reliability. An economic benefit of accurate load forecasts is the ability to price electricity in response to the aggregate demand. Prices fluctuate in response to the demand for electricity, with peak periods being characterized by high electricity prices and likewise off peak periods with lower prices. We aim to apply machine learning algorithms to the load forecasting problem and determine whether they can outperform these traditional techniques. Moreover we will use well known time series analysis methods for scaling and de-trending the dataset to prepare it for the learning algorithms. This could potentially lead to more accurate load predictions and prove valuable to utilities for the aforementioned reasons.

Neural Networks and ANN

A neural network is designed to model the way in which the brain performs a particular task.

It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.

2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

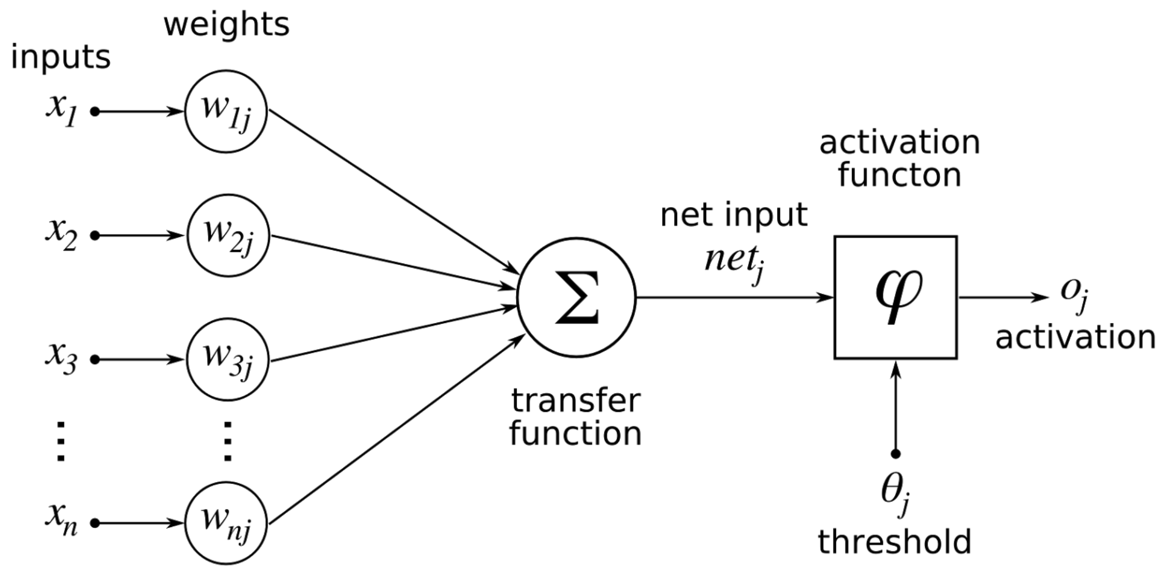
The procedure used to perform the learning process is called a learning algorithm, the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective.

A neuron is an information processing unit that is fundamental to the operation of a neural network. The three basic elements of the neuron model are:

1. A set of weights, each of which is characterized by a strength of its own. A signal connected to neuron is multiplied by the weight. The weight of an artificial neuron may lie in a range that includes negative as well as positive values.

2. An adder for summing the input signals, weighted by the respective weights of the neuron.

3. An activation function for limiting the amplitude of the output of a neuron. It is also referred to as squashing function which squashes the amplitude range of the output signal to some finite value.



Learning Processes

1. Supervised learning

The learning rule is provided with a set of training data of proper network behavior. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets.

2. Unsupervised learning

The weights and biases are modified in response to network inputs only. There are no target outputs available. Most of these algorithms perform clustering to categorize the input patterns into a finite number of classes.

3. Reinforcement learning

It is similar to supervised learning, except that, instead of being provided with the correct output for each network input, the algorithm is only given a grade. The grade is a measure of the network performance over some sequence of inputs.

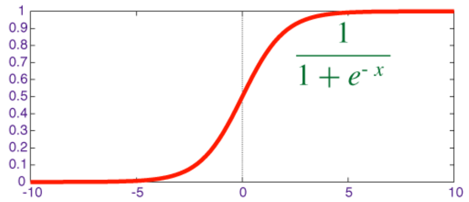
Backpropagation Algorithm

MLPs are generally trained in a supervised manner with the error back-propagation algorithm. This algorithm is based on the error-correction learning rule.

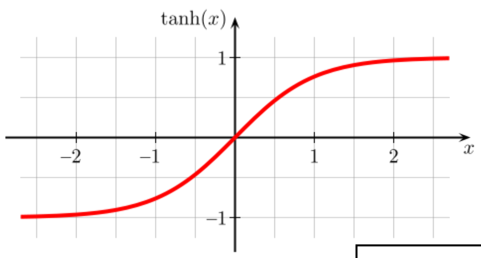
This consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, the weights are all fixed and an input vector is applied to the nodes of the network, and its effect propagates through the network layer by layer. A set of outputs is produced as the actual response of the network. During the backward pass, the weights are all adjusted in accordance with an error correction rule. The actual response of the network is subtracted from a desired response to produce an error signal. This error signal is then propagated backward through the network. The weights are adjusted to make the actual response of the network move closer to the desired response.

A multilayer perceptron has three distinctive characteristics:

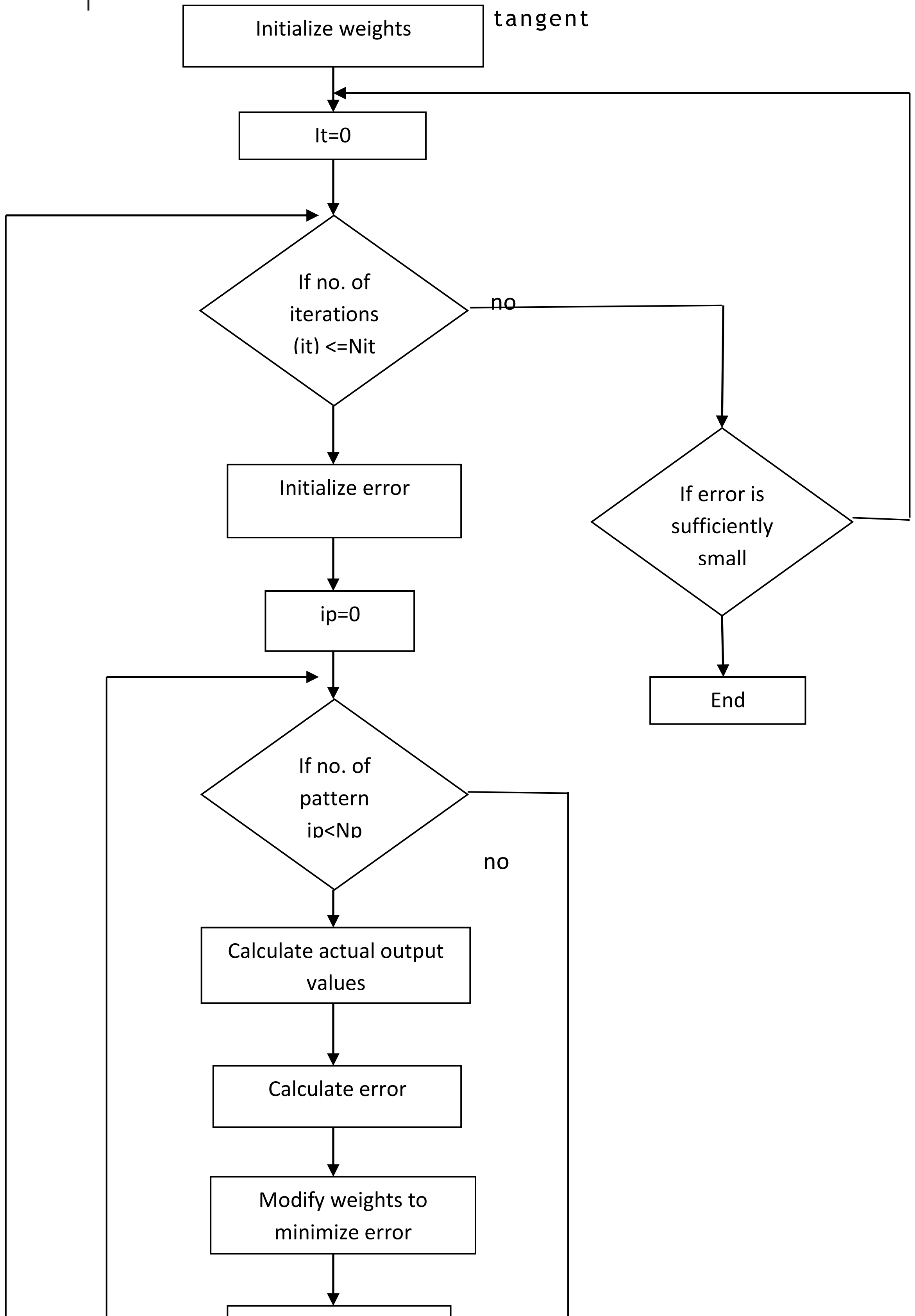
The model of each neuron in the network includes a nonlinear activation function. The sigmoid and hyperbolic tangent functions are commonly used.



sigmoid


tanh(x)
1
−1
−2
−1
1
2
x

tangent


Initialize weights
It=0
If no. of iterations (it) <=Nit
no
If error is sufficiently small
End
Initialize error
ip=0
If no. of pattern ip<Np
no
Calculate actual output values
Calculate error
Modify weights to minimize error

Language and frameworks

Python

Python is a high-level, general purpose, interpreted, dynamic programming language which focuses primarily on increased readability of code. Its syntax is closer to speaking english and allows programmers to use comparatively fewer lines of code, as opposed to the more popular languages like C++ or Java. Python is highly platform independent and interpreters are available for all popular operating systems.

All the code for modelling, training, predicting and analysis has been written in Python 3.

Numpy and Matplotlib

NumPy is an extension to the Python programming language, adding support for large, multidimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like wxPython, Qt, or GTK+. All the graphs for analysis have been plotted using matplotlib.

Scikit-learn

It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The neural network used for forecasting has been constructed using this package.

Forecasting

Data:

Hourly data for 2012-2014 of ISO New England Market

Original Features in Dataset:

Feature	Description
Date	Date(MM/DD/YYYY)
Hour	Hour of the day (24 Load values in a day)
DryBulb	Dry bulb temperature (Fahrenheit)
DewBulb	Dew bulb temperature (Fahrenheit)
SYSLoad	NEPOOL system load = [generation – pumping load + net interchange] as determined by metering (MWh)

Inputs:

Parameters:

- Hour of the day
- Dew Point temperature
- Dry bulb temperature
- Previous Week lagged SYSload
- Previous day lagged SYSload
- Previous hour lagged SYSload

Output:

SYSLoad at specified hour of the day

Training data: 2012-2013 data; 17208 observations

Test data: 2014 data; 8592 observations

Code

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


inputdata = pd.read_csv('input2012and13.csv', header = 0)

inputdata2 = inputdata.iloc[0:17208,1:7] # column 0 was date column

print(inputdata2.head())

inputdata2 = inputdata2.values

print(type(inputdata2))


targetdata = inputdata.iloc[0:17208,7]

targetdata = targetdata.values


X_test1 = pd.read_csv('input2014.csv', header = 0)

X_test = X_test1.iloc[0:8592, 1:7]

X_test = X_test.values

y_test = X_test1.iloc[0:8592, 7]
y_test = y_test.values


X_train = inputdata2

y_train = targetdata


from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(inputdata2, targetdata)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)


from sklearn.neural_network import MLPRegressor

mlp  =   MLPRegressor(hidden_layer_sizes=(50),  activation  =  'logistic',  solver  =
'lbfgs', max_iter = 10000)

mlp.fit(X_train, y_train)


# x = day of the year

x = 61

start = 24 * x

end = 24 * (x + 1)

predictionsday = mlp.predict(X_test[start:end])
```

```
print(predictionsday)


dateselect = X_test1.iloc[start:end,0]

dateselect = dateselect.values

plt.plot(predictionsday, color = 'blue')

plt.plot(y_test[(24*x):end], color = 'red')

plt.ylabel('test load 2014 for day')

plt.show()

print('MAPE')

mape1 = np.mean(np.absolute((y_test[start : end]- predictionsday) / y_test[start :
end])) * 100

print (mape1)

x0 = np.abs((y_test[start : end] - predictionsday) / y_test[start : end])*100

plt.plot(x0)

plt.ylabel('MAPE error')

plt.show()


# weekly----- y = week of the year

y = 10

start = (y*24*7)+1

end = (y+1)*24*7


predictionsweek = mlp.predict(X_test[start : end])

print(predictionsweek)
```

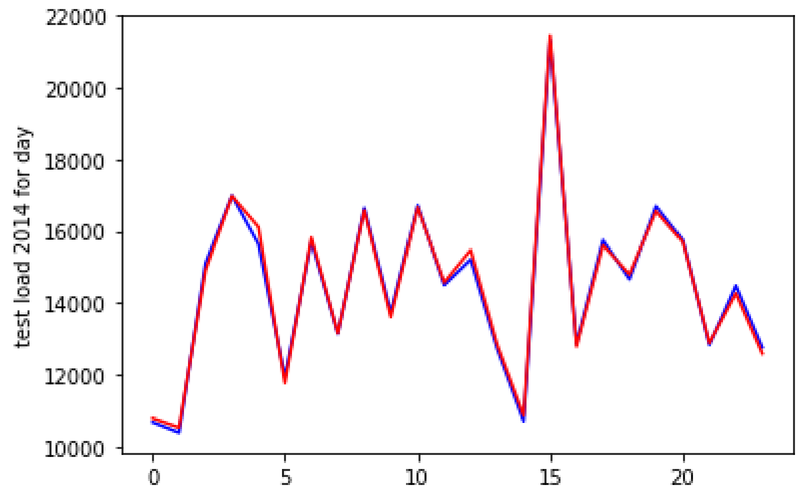
```
plt.plot(predictionsweek, color = 'blue')

plt.plot(y_test[start : end], color = 'red')

plt.ylabel('test load 2014 for week')

plt.show()

print('MAPE')

mape1 = np.mean(np.absolute((y_test[start : end]- predictionsweek) / y_test[start :
end])) * 100

print (mape1)

x0 = np.abs((y_test[start : end] - predictionsweek) / y_test[start : end])*100

plt.plot(x0)

plt.ylabel('MAPE error')

plt.show()


predictions = mlp.predict(X_test)

predictions2 = mlp.predict(X_train)

#mlp.score(predictions, y_test)

plt.plot(y_test, color = 'red')

plt.plot(predictions, color = 'blue')

plt.ylabel('Test Load 2014')

plt.show()

plt.plot(y_train, color='red')

plt.plot(predictions2, color='green')

plt.ylabel('Training Load 2012 and 2013')
plt.show()

#metrics to evaluate performance

print('MAPE')

mape1 = np.mean(np.absolute((y_test - predictions) / y_test)) * 100

print (mape1)


print ('RMSE of testing data')

#error1 = ((predictions-y_test)**2)

#print(error1.sum()/len(y_test))

from sklearn.metrics import mean_squared_error

from math import sqrt


print(sqrt(mean_squared_error(y_test, predictions)))


x0 = np.abs((y_test - predictions) / y_test)*100

plt.plot(x0)

plt.ylabel('MAPE error')

plt.show()

data_to_plot = [x0]

import matplotlib.pyplot as plt1


fig = plt1.figure(1, figsize=(9, 6))

ax = fig.add_subplot(111)

bp = ax.boxplot(data_to_plot)
```

```
fig.savefig('fig1.png', bbox_inches='tight')
```

Outputs

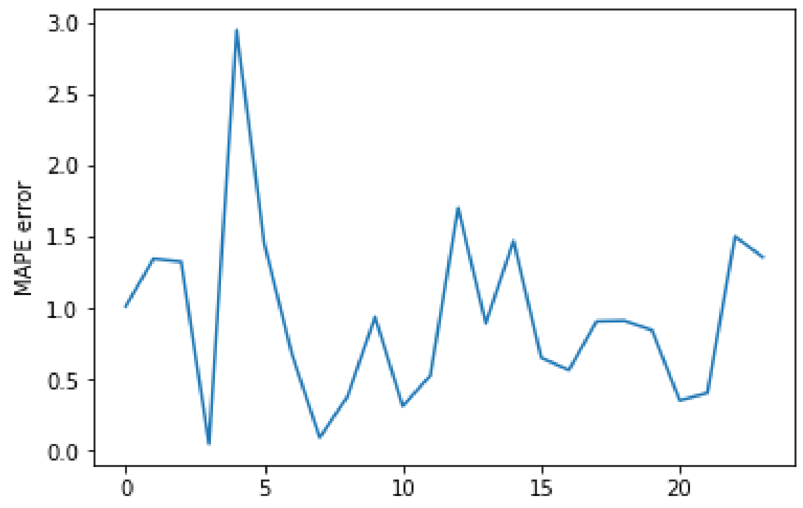
DAY FORECAST (for day 61 of the year)

```
[ 10670.70925091  10380.39444479  15099.39702195 ...,  12826.61001821
  14479.20967336  12751.77055187]
```



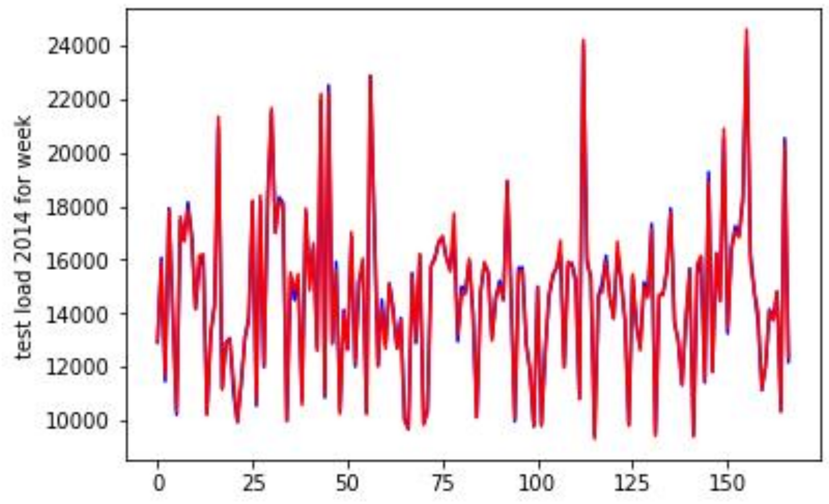
MAPE

0.943484459352



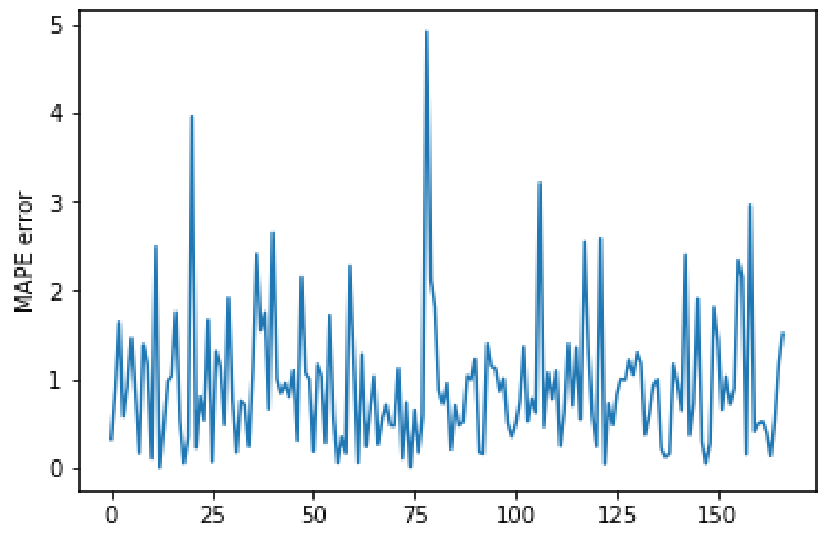
<u>WEEK FORECAST (for week 10 of the yaar)</u>

```
[ 12899.94438161  16072.76480462  11455.5985477  ...,  10373.83264236
  20533.20158452  12180.96716532]
```



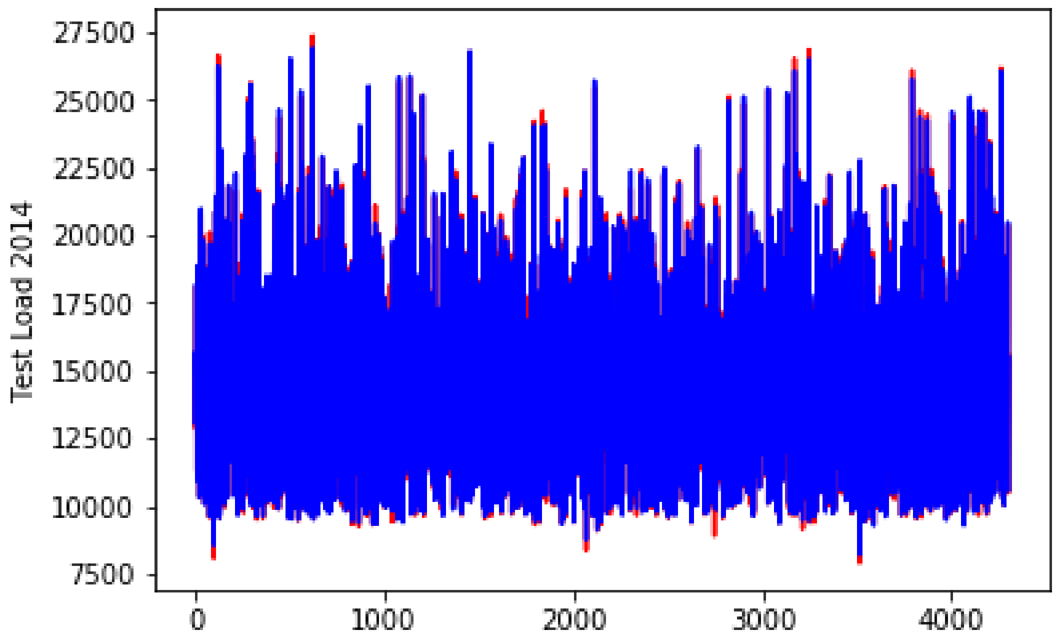
MAPE

0.940255071756



YEAR FORECAST (for 2014)

```
[15622 12853 14426 ..., 20364 10472 15560]

(4302,)

[ 15642.28287381  13014.54063632  14624.65964734 ...,  20484.78168611

  10545.28755608  15485.91941825]
```

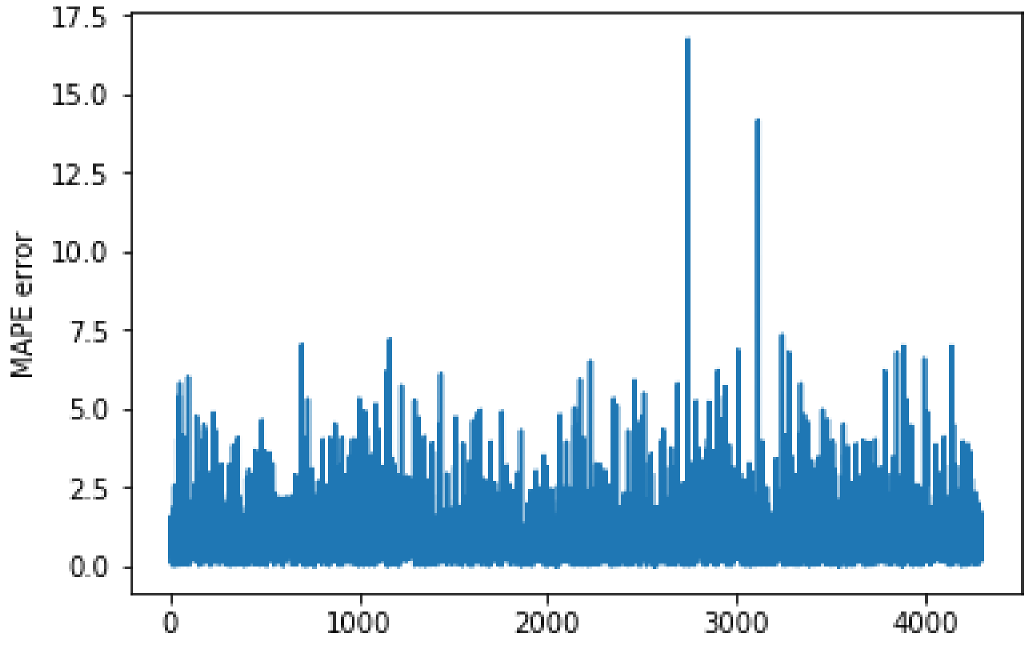


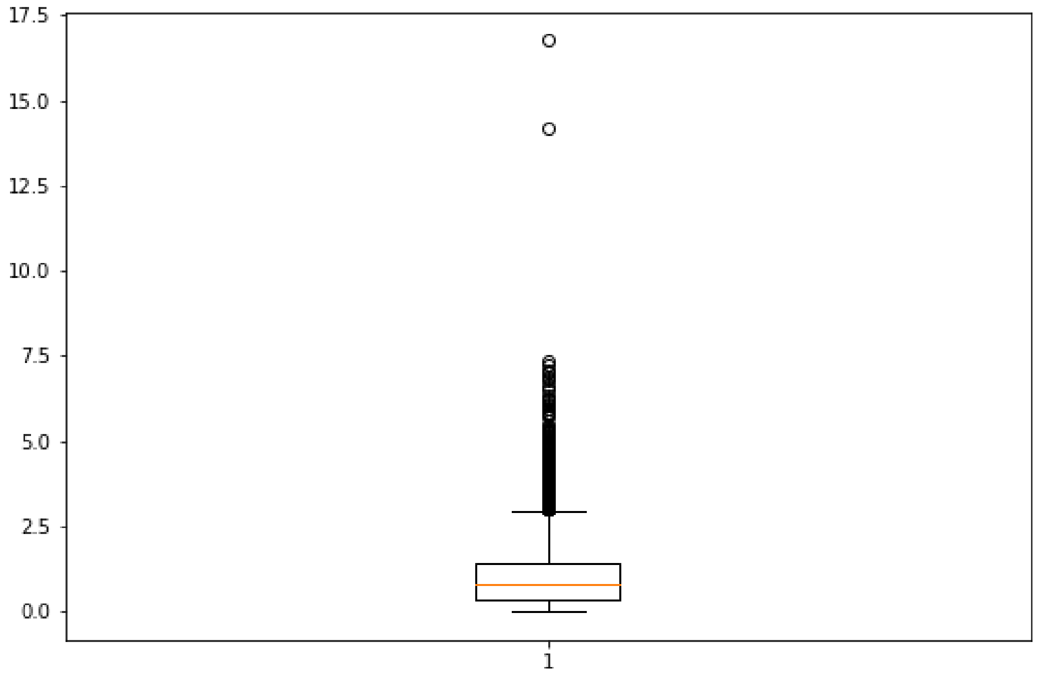
MAPE

1.05226744313

RMSE of testing data

214.86809740317275





Error box plot

References

Short Term Power Load Forecasting using Deep Neural Network – Mohi Ud Din, Marnerides

Short term Load Forecasting using Artificial Neural Network – Lee, Park

Day Ahead Load Forecast in ISO New England and Ontario Market using a Novel ANN – Pandey, Sahay, Tripathi

https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d

https://github.com/lbenning/Load-Forecasting/blob/master/writeup/writeup.pdf

Datacamp.com

Sklearn.com

Stackoverflow.com