UNIVERSITY OF MASSACHUSETTS DARTMOUTH

DEPARTMENT OF COMPUTER & INFORMATION SCIENCE

Prediction of Energy Usage for Smart Cities

CIS 600: MASTER'S PROJECT Final Project Report

Project adviser: -

Prof. Haiping Xu

Author: -

Hardik Sankhla

Student id: - 01617980

Table of contents

Abstract: -

Smart grids are one of the most crucial parts of a smart city, which is also the center of focus for industries around the world. The smart grids consist of various sources of power such as coal plant, hydro-electric plant, solar plant and oil plant. Currently coal, natural gas and oil are still the major sources of energy; however, the green and sustainable energy sources such as wind, water and sunlight, have become more and more important. One major function of a smart city is to deliver power in an environment-friendly way, while reducing costs and increasing its reliability and transparency. There has been a great interest in developing accurate prediction models for energy demand in recent years. However, it has been a challenging task because there are many different sources of energy, and energy usage also depends on many different factors such as month, day of the week, hour, temperature, weather, and many others. In this project, we attempt to improve the current energy demand predicting techniques using deep neural-network models. We develop our deep neural network models in Python supported by the TensorFlow deep learning library. The performance of our models is evaluated against the industry standard used by ISO New England (ISO NE) for prediction of the energy demand in the Boston Area. The experimental results show that our deep neural-network approach performs better than the industry standard with lower error rates. To further support visualization of energy usage for smart cities, we display the computation graph, namely the data flow graph, which can be derived from the deep neural network models. Note that in a computation graph, nodes represent unit operations, while the edges represent the data that flow through the graph and get modified as they pass thought each node.

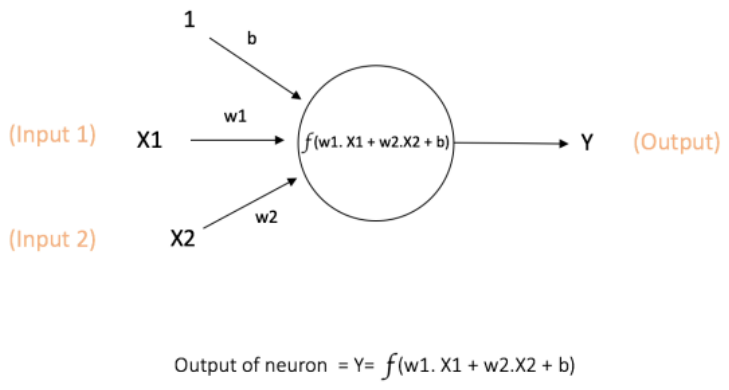
1. Introduction

1.1. Introduction to smart grid

Smart grid is a one of the imperative components of smart city. It is considered as one of the essential technologies which make power grids more efficient, more reliable and more environment-friendly. The smart grid is an electricity supply network that uses digital communications technology to detect and react to local changes in usage., to ensure energy is distributed in the most efficient way and allows control of appliances in the consumers' houses and of machines in factories to save energy, while reducing costs and increasing reliability and transparency. A smart grid includes an intelligent monitoring system that keeps track of all the electricity that flows in the system. A key component of an efficient Smart Grid operation will be the accurate prediction of future supply and demand trends.

1.2. Introduction to neural network

A Neural network is a computational model inspired by the actual biological neural network. A typical neural network consists of bunch of artificial neurons arranged in layers. Each layer is connected to the adjacent layer via edges and each edge have some weights to them. A single neuron is the basic unit of computation in neural network. It can take various inputs, then applies the weighted sum of the inputs. After the weighted sum of input, it applies a activation function that help us introduce non-linearity to the neural network output.



Here f(w1.x1+w2.x2+b) is the activation function applied on the weighted sum of inputs. There are various activation function but the most common ones are sigmoid, tanh, ReLU.

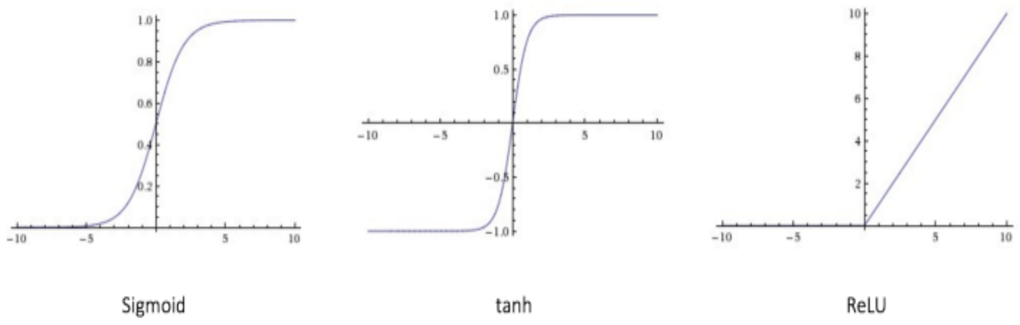
- Sigmoid: takes a real-valued input and squashes it to range between 0 and 1

  $\sigma(x) = 1 / (1 + \exp(-x))$

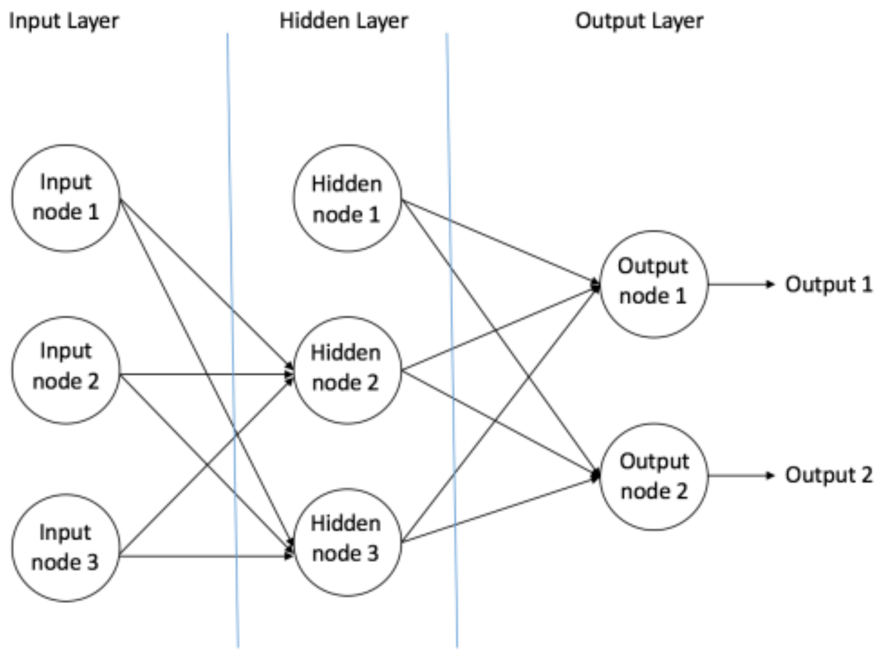
- tanh: takes a real-valued input and squashes it to the range [-1, 1]

  $\tanh(x) = 2\sigma(2x) - 1$

- ReLU: ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero) f(x) = max(0, x)



A typical neural network would have 3 major section which is classified based on the position in the neural network.



1. **Input Nodes –**

   The input nodes as the name suggest provides information from the dataset to the neural network. There is no computation performed in the input node they just pass the information onto the hidden nodes. The layer that consists of input nodes is called input layer. In a typical feed-forward neural network there is only one input layer.

2. **Hidden Nodes –**

   The hidden nodes is called hidden as they are hidden from the outside world, meaning that they have no direct connection to the outside of the neural network. They basically perform the computation and forward the information from input layer to the output layer. The layer that consists of hidden nodes is called hidden layer. In a typical feed-forward network there can be zero or multiple hidden layers

3. **Output Nodes –**

   The output nodes as the name suggests are the nodes that gives us the output of our neural network. The output nodes are the only nodes through which the information is send outside from the neural network. The layer that consists of output nodes is called output layer. In a typical feed-forward neural network there is only one output layer.

In a neural network all the weights and biases are assigned randomly and then we calculate the supposed output. After that we compare it to the actual value and then based upon the difference between the supposed output and the actual output we train the neural network by changing the values of weights and biases. We use back propagation of error to train the neural network. The error is propagated to the previous layers, it notes that how the change in values of weights and biases affects the error. To do this we use a cost function which gives us the magnitude of difference between actual and predicted output. There are various cost function that are used such as: -

1.Quadratic cost

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

2.Cross-entropy cost

$$H(p,q) = -\sum_{x} p(x)\,\log q(x).$$

3. Exponential cost

$$C_{EXP}(W,B,S^r,E^r) = \tau\ \exp(\frac{1}{\tau}\sum_j (a_j^L - E_j^r)^2)$$

Then by doing the partial differential of the cost function with respect to the individual weights gives us how changing each weight can either increase or decrees the cost function and our ultimate goal is to minimize the cost function so that our predicted values are as close as possible to the actual value. After we have calculated the differential/gradient we adjust all the weights and biases by alpha*gradient amount where alpha is the learning rate, learning rate tells the program that how rapidly the values of weights and biases should be changed. We repeat this process with all the training set, after that we repeat this training process unit we get our desired result.

1.3. Introduction to TensorFlow

Tensor flow is a recently open sourced deep learning library by Google for computation using data flow graphs. TensorFlow provides base for defining functions on tensors and automatically computing their derivatives. While it contains a wide range of functionality, TensorFlow is mainly designed for deep neural network models. Nodes in the data flow graph (computation graph) represents matamatical operations, while the graph edges represent the tensor that flow through the graph and get modified as it passes thought each node.

Formally, tensors are multilinear maps from vector spaces to the real numbers. A tensor is denoted as an organized multi-dimensional array of numerical value. The order (also degree or rank) of a tensor is basically the dimensionality of the array, or in layman terms it is the no of indices used to label a value in the tensor. For example a matrix (a 2-dimentional array) is a 2nd order tensor, a vector (a 1-dimentional array) is a 1st order tensor and a scalar number (a 0-dimentional array) is a 0th order tensor.

2. Setup

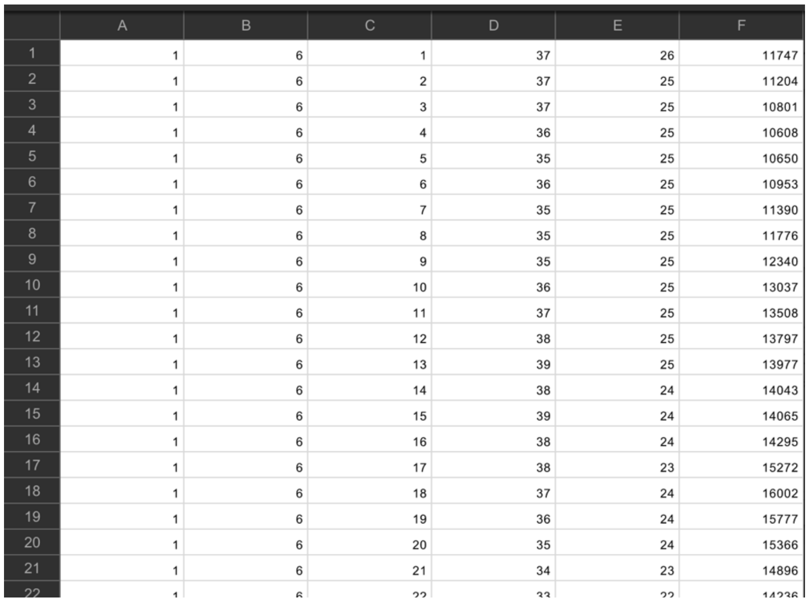
2.1. Data preprocessing

We downloaded the data in excel file from https://www.iso-ne.com/isoexpress/web/reports/load-and-demand/-/tree/zone-info. It has various sheets in which we are concerned with only ISO NE, in that sheet it has various features(Date, Hour, DA_DEMD, DEMAND, DA_LMP, DA_EC, DA_CC, DA_MLC, RT_LMP, RT_EC, RT_CC, RT_MLC, DryBulb, DewPnt, SYSLoad, RegCP), but the ones we are intrested in are i.e.Date, Hour, DryBulb, DewPnt, SYSLoad. We remove all the other features that we don't need and then we can start converting the remaining data into more relevant data for our prediction model. Firstly, we would convert the "Date" feature into "Day of week" and "Month". Then we would convert DryBulb and DewPnt values from Fahrenheit to kelvin. Now we would add more 2 more features (i.e. DryBulb^2 & DewPnt^2). "SYSLoad" is our output against which we will verify our results. After all this we will convert the data into more programing-friendly format i.e. csv file format.

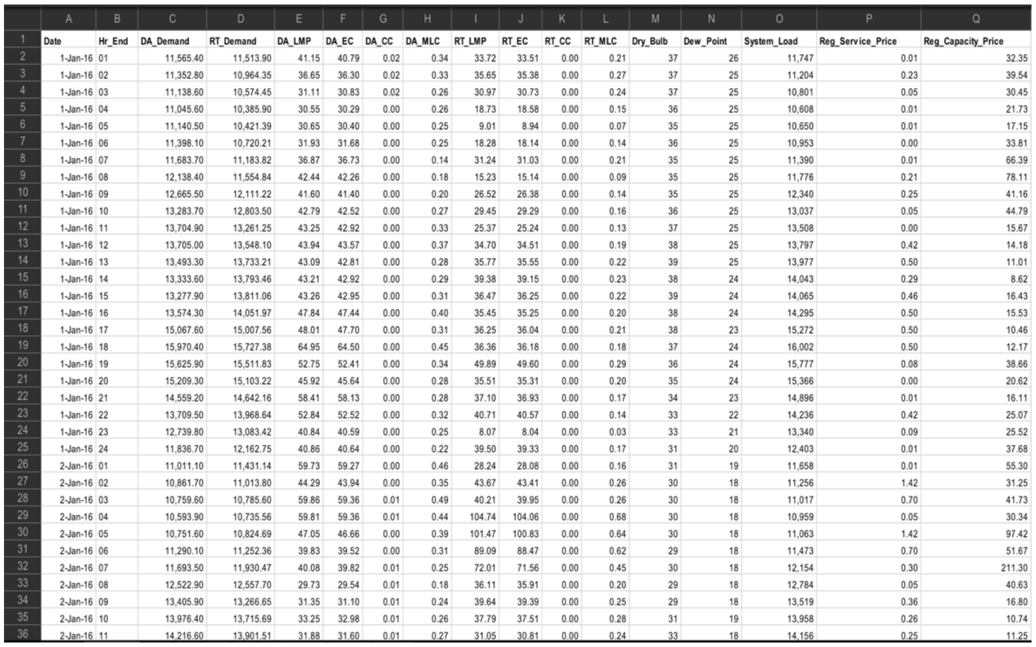
Table of features: -

No.	**Features/column**	**Describtion**
1	Month	Represents which month of the observation it is (Jan, Feb,….,Nov and Dec). Value ranging from 1-12
2	Day of week	Represents which day of the week of the observation it is (i.e. Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday). Value ranging from 1-7
3	Hour	The hour of the observation, in hour ending and 24-hour convention.Value ranging from 1-24
4	DryBulb	The dry-bulb temperature in °F for the weather station corresponding to the load zone or Trading Hub. The dry-bulb temperature (DBT) is the temperature of air measured by a thermometer freely exposed to the air but shielded from radiation and moisture. DBT is the temperature that is usually thought of as air temperature, and it is the true thermodynamic temperature.
5	DewPnt	The dewpoint temperature in °F for the weather station corresponding to the load zone or Trading Hub. Dew point is the atmospheric temperature (varying according to pressure and humidity) below which water droplets begin to condense and dew can form.
6	DryBulb^2	Square of DryBulb.
7	DewPnt^2	Square of DewPnt.
8	System_Load	It is the actual New England system load in MW as determined by metering, and is used for planning and reporting purposes, not for settlement; System Load is the sum of metered

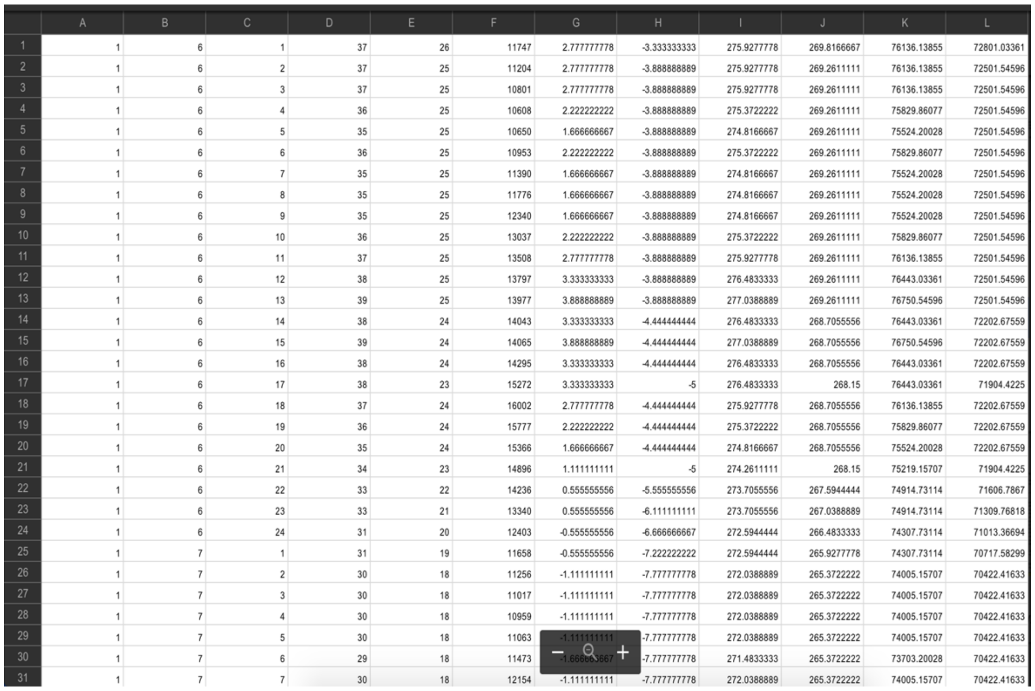
		generation and metered net interchange plus demand from pumped storage units



	A	B	C	D	E	F
1	1	6	1	37	26	11747
2	1	6	2	37	25	11204
3	1	6	3	37	25	10801
4	1	6	4	36	25	10608
5	1	6	5	35	25	10650
6	1	6	6	36	25	10953
7	1	6	7	35	25	11390
8	1	6	8	35	25	11776
9	1	6	9	35	25	12340
10	1	6	10	36	25	13037
11	1	6	11	37	25	13508
12	1	6	12	38	25	13797
13	1	6	13	39	25	13977
14	1	6	14	38	24	14043
15	1	6	15	39	24	14065
16	1	6	16	38	24	14295
17	1	6	17	38	23	15272
18	1	6	18	37	24	16002
19	1	6	19	36	24	15777
20	1	6	20	35	24	15366
21	1	6	21	34	23	14896
22	1	6	22	33	22	14236



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Date	Hr_End	DA_Demand	RT_Demand	DA_LMP	DA_EC	DA_CC	DA_MLC	RT_LMP	RT_EC	RT_CC	RT_MLC	Dry_Bulb	Dew_Point	System_Load	Reg_Service_Price	Reg_Capacity_Price
2	1-Jan-16	01	11,565.40	11,513.90	41.15	40.79	0.02	0.34	33.72	33.51	0.00	0.21	37	26	11,747	0.01	32.35
3	1-Jan-16	02	11,352.80	10,964.35	36.65	36.30	0.02	0.33	35.65	35.38	0.00	0.27	37	25	11,204	0.23	39.54
4	1-Jan-16	03	11,138.60	10,574.45	31.11	30.83	0.02	0.26	30.97	30.73	0.00	0.24	37	25	10,801	0.05	30.45
5	1-Jan-16	04	11,045.60	10,385.90	30.55	30.29	0.00	0.26	18.73	18.58	0.00	0.15	36	25	10,608	0.01	21.73
6	1-Jan-16	05	11,140.50	10,421.39	30.65	30.40	0.00	0.25	9.01	8.94	0.00	0.07	35	25	10,650	0.01	17.15
7	1-Jan-16	06	11,398.10	10,720.21	31.93	31.68	0.00	0.25	18.28	18.14	0.00	0.14	36	25	10,953	0.00	33.81
8	1-Jan-16	07	11,683.70	11,183.82	36.87	36.73	0.00	0.14	31.24	31.03	0.00	0.21	35	25	11,390	0.01	66.39
9	1-Jan-16	08	12,138.40	11,554.84	42.44	42.26	0.00	0.18	15.23	15.14	0.00	0.09	35	25	11,776	0.21	78.11
10	1-Jan-16	09	12,665.50	12,111.22	41.60	41.40	0.00	0.20	26.52	26.38	0.00	0.14	35	25	12,340	0.25	41.16
11	1-Jan-16	10	13,283.70	12,803.50	42.79	42.52	0.00	0.27	29.45	29.29	0.00	0.16	36	25	13,037	0.05	44.79
12	1-Jan-16	11	13,704.90	13,261.25	43.25	42.92	0.00	0.33	25.37	25.24	0.00	0.13	37	25	13,508	0.00	15.67
13	1-Jan-16	12	13,705.00	13,548.10	43.94	43.57	0.00	0.37	34.70	34.51	0.00	0.19	38	25	13,797	0.42	14.18
14	1-Jan-16	13	13,493.30	13,733.21	43.09	42.81	0.00	0.28	35.77	35.55	0.00	0.22	39	25	13,977	0.50	11.01
15	1-Jan-16	14	13,333.60	13,793.46	43.21	42.92	0.00	0.29	39.38	39.15	0.00	0.23	38	24	14,043	0.29	8.62
16	1-Jan-16	15	13,277.90	13,811.06	43.26	42.95	0.00	0.31	36.47	36.25	0.00	0.22	39	24	14,065	0.46	16.43
17	1-Jan-16	16	13,574.30	14,051.97	47.84	47.44	0.00	0.40	35.45	35.25	0.00	0.20	38	24	14,295	0.50	15.53
18	1-Jan-16	17	15,067.60	15,007.56	48.01	47.70	0.00	0.31	36.25	36.04	0.00	0.21	38	23	15,272	0.50	10.46
19	1-Jan-16	18	15,970.40	15,727.38	64.95	64.50	0.00	0.45	36.36	36.18	0.00	0.18	37	24	16,002	0.50	12.17
20	1-Jan-16	19	15,625.90	15,511.83	52.75	52.41	0.00	0.34	49.89	49.60	0.00	0.29	36	24	15,777	0.08	38.66
21	1-Jan-16	20	15,209.30	15,103.22	45.92	45.64	0.00	0.28	35.51	35.31	0.00	0.20	35	24	15,366	0.00	20.62
22	1-Jan-16	21	14,559.20	14,642.16	58.41	58.13	0.00	0.28	37.10	36.93	0.00	0.17	34	23	14,896	0.01	16.11
23	1-Jan-16	22	13,709.50	13,968.64	52.84	52.52	0.00	0.32	40.71	40.57	0.00	0.14	33	22	14,236	0.42	25.07
24	1-Jan-16	23	12,739.80	13,083.42	40.84	40.59	0.00	0.25	8.07	8.04	0.00	0.03	33	21	13,340	0.09	25.52
25	1-Jan-16	24	11,836.70	12,162.75	40.86	40.64	0.00	0.22	39.50	39.33	0.00	0.17	31	20	12,403	0.01	37.68
26	2-Jan-16	01	11,011.10	11,431.14	59.73	59.27	0.00	0.46	28.24	28.08	0.00	0.16	31	19	11,658	0.01	55.30
27	2-Jan-16	02	10,861.70	11,013.80	44.29	43.94	0.00	0.35	43.67	43.41	0.00	0.26	30	18	11,256	1.42	31.25
28	2-Jan-16	03	10,759.60	10,785.60	59.86	59.36	0.01	0.49	40.21	39.95	0.00	0.26	30	18	11,017	0.70	41.73
29	2-Jan-16	04	10,593.90	10,735.56	59.81	59.36	0.01	0.44	104.74	104.06	0.00	0.68	30	18	10,959	0.05	30.34
30	2-Jan-16	05	10,751.60	10,824.69	47.05	46.66	0.00	0.39	101.47	100.83	0.00	0.64	30	18	11,063	1.42	97.42
31	2-Jan-16	06	11,290.10	11,252.36	39.83	39.52	0.00	0.31	89.09	88.47	0.00	0.62	29	18	11,473	0.70	51.67
32	2-Jan-16	07	11,693.50	11,930.47	40.08	39.82	0.01	0.25	72.01	71.56	0.00	0.45	30	18	12,154	0.30	211.30
33	2-Jan-16	08	12,522.90	12,557.70	29.73	29.54	0.01	0.18	36.11	35.91	0.00	0.20	29	18	12,784	0.05	40.63
34	2-Jan-16	09	13,405.90	13,266.65	31.35	31.10	0.01	0.24	39.64	39.39	0.00	0.25	29	18	13,519	0.36	16.80
35	2-Jan-16	10	13,976.40	13,715.69	33.25	32.98	0.01	0.26	37.79	37.51	0.00	0.28	31	19	13,958	0.26	10.74
36	2-Jan-16	11	14,216.60	13,901.51	31.88	31.60	0.01	0.27	31.05	30.81	0.00	0.24	33	18	14,156	0.25	11.25



	A	B	C	D	E	F	G	H	I	J	K	L
1	1	6	1	37	26	11747	2.777777778	-3.333333333	275.9277778	269.8166667	76136.13855	72801.03361
2	1	6	2	37	25	11204	2.777777778	-3.888888889	275.9277778	269.2611111	76136.13855	72501.54596
3	1	6	3	37	25	10801	2.777777778	-3.888888889	275.9277778	269.2611111	76136.13855	72501.54596
4	1	6	4	36	25	10608	2.222222222	-3.888888889	275.3722222	269.2611111	75829.86077	72501.54596
5	1	6	5	35	25	10650	1.666666667	-3.888888889	274.8166667	269.2611111	75524.20028	72501.54596
6	1	6	6	36	25	10953	2.222222222	-3.888888889	275.3722222	269.2611111	75829.86077	72501.54596
7	1	6	7	35	25	11390	1.666666667	-3.888888889	274.8166667	269.2611111	75524.20028	72501.54596
8	1	6	8	35	25	11776	1.666666667	-3.888888889	274.8166667	269.2611111	75524.20028	72501.54596
9	1	6	9	35	25	12340	1.666666667	-3.888888889	274.8166667	269.2611111	75524.20028	72501.54596
10	1	6	10	36	25	13037	2.222222222	-3.888888889	275.3722222	269.2611111	75829.86077	72501.54596
11	1	6	11	37	25	13508	2.777777778	-3.888888889	275.9277778	269.2611111	76136.13855	72501.54596
12	1	6	12	38	25	13797	3.333333333	-3.888888889	276.4833333	269.2611111	76443.03361	72501.54596
13	1	6	13	39	25	13977	3.888888889	-3.888888889	277.0388889	269.2611111	76750.54596	72501.54596
14	1	6	14	38	24	14043	3.333333333	-4.444444444	276.4833333	268.7055556	76443.03361	72202.67559
15	1	6	15	39	24	14065	3.888888889	-4.444444444	277.0388889	268.7055556	76750.54596	72202.67559
16	1	6	16	38	24	14295	3.333333333	-4.444444444	276.4833333	268.7055556	76443.03361	72202.67559
17	1	6	17	38	23	15272	3.333333333	-5	276.4833333	268.15	76443.03361	71904.4225
18	1	6	18	37	24	16002	2.777777778	-4.444444444	275.9277778	268.7055556	76136.13855	72202.67559
19	1	6	19	36	24	15777	2.222222222	-4.444444444	275.3722222	268.7055556	75829.86077	72202.67559
20	1	6	20	35	24	15366	1.666666667	-4.444444444	274.8166667	268.7055556	75524.20028	72202.67559
21	1	6	21	34	23	14896	1.111111111	-5	274.2611111	268.15	75219.15707	71904.4225
22	1	6	22	33	22	14236	0.555555556	-5.555555556	273.7055556	267.5944444	74914.73114	71606.7867
23	1	6	23	33	21	13340	0.555555556	-6.111111111	273.7055556	267.0388889	74914.73114	71309.76818
24	1	6	24	31	20	12403	-0.555555556	-6.666666667	272.5944444	266.4833333	74307.73114	71013.36694
25	1	7	1	31	19	11658	-0.555555556	-7.222222222	272.5944444	265.9277778	74307.73114	70717.58299
26	1	7	2	30	18	11256	-1.111111111	-7.777777778	272.0388889	265.3722222	74005.15707	70422.41633
27	1	7	3	30	18	11017	-1.111111111	-7.777777778	272.0388889	265.3722222	74005.15707	70422.41633
28	1	7	4	30	18	10959	-1.111111111	-7.777777778	272.0388889	265.3722222	74005.15707	70422.41633
29	1	7	5	30	18	11063	-1.111111111	-7.777777778	272.0388889	265.3722222	74005.15707	70422.41633
30	1	7	6	29	18	11473	-1.666666667	-7.777777778	271.4833333	265.3722222	73703.20028	70422.41633
31	1	7	7	30	18	12154	-1.111111111	-7.777777778	272.0388889	265.3722222	74005.15707	70422.41633

2.2. libraries and software installation

First, we install latest version of python on a fresh machine. If you don't have fresh machine on hand then you can make a fresh installation of which ever OS of your choice on a virtual machine. After the installation of python, make sure that the PATH system variable is set for python. After we are done with all of this then we will install the libraries that we will use in our project.

list of libraries that we used: -

1.) TensorFlow(creating computation graph of neural network and training it)
2.) pandas (to read the data set)
3.) NumPy(support for large multi-dimensional array and matrix)
4.) matplotlib (to plot graphs)
5.) sklearn(Simple and efficient tools for data mining and data analysis)

We open cmd as admin and type in cmd the following: -

1.) pip3 install –upgrade tensorflow
2.) pip3 install matplotlib
3.) pip3 install scipy
4.) pip3 install numpy
5.) pip3 install sklearn
6.) pip3 install pandas
7.) pip3 install TensorBoard

Note: these installation commands are subject to change as the python version changes.

3. Method

3.1. Programing

We import all the libraries that we are going to use.

```
import tensorflow as tf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.utils import shuffle

from sklearn.model_selection import train_test_split
```

First, we loaded the data with the help of pandas library,

```
def read_dataset():

    df = 
pd.read_csv("C:\\Users\\Hardik\\Desktop\\Masters_project\\te
nsorflow\\smd_hourly_2016 copy1.csv")

    print(len(df.columns))

    x1 = df[df.columns[0:1]].values

    x2 = df[df.columns[1:2]].values
    x3 = df[df.columns[2:3]].values

    x4 = df[df.columns[3:4]].values

    x5 = df[df.columns[4:5]].values

    x6 = df[df.columns[10:11]].values

    x7 = df[df.columns[11:12]].values

    y  = df[df.columns[5:6]].values

    print(x1.shape)

    print(y.shape)

    return(x1,x2,x3,x4,x5,x6,x7,y)
```

then we shuffle it with the help of sklearn, now again we use sklearn to split the data into 2 groups i.e. test and training datasets.

```
#Shuffle the dataset

x1,x2,x3,x4,x5,x6,x7,y = 
shuffle(x1,x2,x3,x4,x5,x6,x7,y,random_state=1)

#split the dataset into test and train parts

train_x1, test_x1,train_x2, test_x2,train_x3, 
test_x3,train_x4, test_x4,train_x5, test_x5,train_x6, 
test_x6,train_x7, test_x7, train_y, test_y = 
train_test_split(x1,x2,x3,x4,x5,x6,x7,y, test_size=0.2, 
random_state=415  )
```

We define the no of epochs and learning rate,

```
#defining parameters

learning_rate =0.0013

training_epochs = 50000
```

then we start building the computation graph in TensorFlow, through which we will pass the data. During the training we will first use the training data set and after that when we are done with the training we use the test dataset to make sure it can work on other data sets on which it has not been trained. First, we define the input end of the computation graph, it takes seven features for seven input nodes.

```
X1 = tf.placeholder(tf.float32,[None,n_dim])

X2 = tf.placeholder(tf.float32,[None,n_dim])

X3 = tf.placeholder(tf.float32,[None,n_dim])

X4 = tf.placeholder(tf.float32,[None,n_dim])

X5 = tf.placeholder(tf.float32,[None,n_dim])

X6 = tf.placeholder(tf.float32,[None,n_dim])

X7 = tf.placeholder(tf.float32,[None,n_dim])

Y_ = tf.placeholder(tf.float32,[None, 1])

W = tf.Variable(tf.zeros([n_dim,1]))

b = tf.Variable(tf.zeros([1]))
```

then we define the no of nodes in the hidden layer

```
#define no of hidden layers ans no of neurons in each layers

n_hidden_11=12

n_hidden_12=4

n_hidden_13=12

n_hidden_14=25

n_hidden_15=25

n_hidden_16=50

n_hidden_17=50

n_hidden_21=16

n_hidden_22=25

n_hidden_23=50

n_hidden_3=13
```

```
n_hidden_4=10

n_hidden_5=5
```

then we define the weights and biases in the hidden layer

```
#defining the weights and biases of each layer

weights = {

    'h11': 
tf.Variable(tf.truncated_normal([n_dim,n_hidden_11])),

    'h12': 
tf.Variable(tf.truncated_normal([n_dim,n_hidden_12])),

    'h13': 
tf.Variable(tf.truncated_normal([n_dim,n_hidden_13])),

    'h14': 
tf.Variable(tf.truncated_normal([n_dim,n_hidden_14])),

    'h15': 
tf.Variable(tf.truncated_normal([n_dim,n_hidden_15])),

    'h16': 
tf.Variable(tf.truncated_normal([n_dim,n_hidden_16])),

    'h17': 
tf.Variable(tf.truncated_normal([n_dim,n_hidden_17])),

    'h2111': 
tf.Variable(tf.truncated_normal([n_hidden_11,n_hidden_21])),

    'h2112': 
tf.Variable(tf.truncated_normal([n_hidden_12,n_hidden_21])),

    'h2113': 
tf.Variable(tf.truncated_normal([n_hidden_13,n_hidden_21])),

    'h2214': 
tf.Variable(tf.truncated_normal([n_hidden_14,n_hidden_22])),

    'h2215': 
tf.Variable(tf.truncated_normal([n_hidden_15,n_hidden_22])),
    'h2316':
tf.Variable(tf.truncated_normal([n_hidden_16,n_hidden_23])),

    'h2317':
tf.Variable(tf.truncated_normal([n_hidden_17,n_hidden_23])),

    'h322':
tf.Variable(tf.truncated_normal([n_hidden_22,n_hidden_3])),

    'h323':
tf.Variable(tf.truncated_normal([n_hidden_23,n_hidden_3])),

    'h421':
tf.Variable(tf.truncated_normal([n_hidden_21,n_hidden_4])),

    'h43':
tf.Variable(tf.truncated_normal([n_hidden_3,n_hidden_4])),

    'h5':
tf.Variable(tf.truncated_normal([n_hidden_4,n_hidden_5])),

    'out': tf.Variable(tf.truncated_normal([n_hidden_5,1]))

}

biases = {

    'b11': tf.Variable(tf.truncated_normal([n_hidden_11])),

    'b12': tf.Variable(tf.truncated_normal([n_hidden_12])),

    'b13': tf.Variable(tf.truncated_normal([n_hidden_13])),

    'b14': tf.Variable(tf.truncated_normal([n_hidden_14])),

    'b15': tf.Variable(tf.truncated_normal([n_hidden_15])),

    'b16': tf.Variable(tf.truncated_normal([n_hidden_16])),

    'b17': tf.Variable(tf.truncated_normal([n_hidden_17])),

    'b2111':
tf.Variable(tf.truncated_normal([n_hidden_21])),

    'b2112':
tf.Variable(tf.truncated_normal([n_hidden_21])),
    'b2113':
tf.Variable(tf.truncated_normal([n_hidden_21])),

    'b2214':
tf.Variable(tf.truncated_normal([n_hidden_22])),

    'b2215':
tf.Variable(tf.truncated_normal([n_hidden_22])),

    'b2316':
tf.Variable(tf.truncated_normal([n_hidden_23])),

    'b2317':
tf.Variable(tf.truncated_normal([n_hidden_23])),

    'b322': tf.Variable(tf.truncated_normal([n_hidden_3])),

    'b323': tf.Variable(tf.truncated_normal([n_hidden_3])),

    'b43': tf.Variable(tf.truncated_normal([n_hidden_4])),

    'b421': tf.Variable(tf.truncated_normal([n_hidden_4])),

    'b5': tf.Variable(tf.truncated_normal([n_hidden_5])),

    'out': tf.Variable(tf.truncated_normal([1])),

}
```

and then we define how the neural network model is structured (how each nodes are connected).

```
#define the model

def multilayer_nn(X1,X2,X3,X4,X5,X6,X7, weights, biases):

    layer_11 = tf.add(tf.matmul(X1, weights['h11']),
biases['b11'])

    layer_11 = tf.nn.tanh(layer_11)

    layer_12 = tf.add(tf.matmul(X2, weights['h12']),
biases['b12'])

    layer_12 = tf.nn.tanh(layer_12)
    layer_13 = tf.add(tf.matmul(X3, weights['h13']),
biases['b13'])

    layer_13 = tf.nn.tanh(layer_13)

    layer_14 = tf.add(tf.matmul(X4, weights['h14']),
biases['b14'])

    layer_14 = tf.nn.tanh(layer_14)

    layer_15 = tf.add(tf.matmul(X5, weights['h15']),
biases['b15'])

    layer_15 = tf.nn.tanh(layer_15)

    layer_16 = tf.add(tf.matmul(X6, weights['h16']),
biases['b16'])

    layer_16 = tf.nn.tanh(layer_16)

    layer_17 = tf.add(tf.matmul(X7, weights['h17']),
biases['b17'])

    layer_17 = tf.nn.tanh(layer_17)

    layer_21 = tf.add( tf.add(tf.matmul(layer_12,
weights['h2112']), biases['b2112']),
tf.add(tf.matmul(layer_13, weights['h2113']),
biases['b2113']))

    layer_21 = tf.nn.sigmoid(layer_21)

    layer_22 = tf.add( tf.add(tf.matmul(layer_14,
weights['h2214']),biases['b2214']),tf.add(tf.matmul(layer_15
, weights['h2215']), biases['b2215']))

    layer_22= tf.nn.sigmoid(layer_22)

    layer_23 = tf.add( tf.add(tf.matmul(layer_16,
weights['h2316']),
biases['b2316']),tf.add(tf.matmul(layer_17,
weights['h2317']), biases['b2317']))

    layer_23= tf.nn.sigmoid(layer_23)
```

```
     layer_3 = tf.add( tf.add(tf.matmul(layer_22,
weights['h322']), biases['b322']),tf.add(tf.matmul(layer_23,
weights['h323']), biases['b323']) )

    layer_3 = tf.nn.tanh(layer_3)

    layer_4 = tf.add( tf.add(tf.matmul(layer_21,
weights['h421']), biases['b421']),tf.add(tf.matmul(layer_3,
weights['h43']), biases['b43']) )

    layer_4 = tf.nn.relu(layer_4)

    layer_5 = tf.add(tf.matmul(layer_4, weights['h5']),
biases['b5'])

    layer_5 = tf.nn.relu(layer_5)

    out_layer = tf.add(tf.matmul(layer_5, weights['out']),
biases['out'])

    print('outputlayer shape')

    print(out_layer.shape)

    return out_layer
```

Then we take the output (the predicted value) and compare it to the actual value and calculate the cost/loss of our model so that we can train it to minimize the cost/loss. To calculate the loss, we used mean squared error (measures the average of the squares of the errors or deviations—that is, the difference between the estimator and what is estimated).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_{di}\right)^2$$

```
#Call the model

Y = multilayer_nn(X1,X2,X3,X4,X5,X6,X7, weights, biases)

print(Y.shape)

print(Y_.shape)

#Loss
```

```
squared_delta = tf.square(Y-Y_)
print(squared_delta.shape)
loss = tf.reduce_sum(squared_delta)
loss = loss/train_x1.shape[0]
#Optimize
optimizer = tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(loss)
```

We calculate accuracy by taking the absolute value of error (diff between actual and predicted value) and then dividing it with the actual value, and then take the average of all values for each epoch.

$$\text{Forecast Error} = \left(\frac{|Actual - Forecast|}{Actual}\right) * 100\%$$

$$\text{Forecast Accuracy} = \text{Maximum } (0,\ 100\% - \text{Forecast Error})$$

```
#Accuracy by segmentaion
accuracy1 = tf.equal(tf.round(Y/2000), tf.round(Y_/2000))
accuracy = tf.reduce_sum(tf.cast(accuracy1, tf.float32 ))
accuracy = accuracy/train_x1.shape[0]
#Accuracy by percentage
acc_by_per = tf.abs((Y-Y_)/Y)
accuracybyper = tf.reduce_sum(acc_by_per)
accuracybyper = (accuracybyper/train_x1.shape[0])*100
```

We save the loss and accuracy of each epoch/iteration by appending it into an array with the help from numpy. then we plot the appended value against the no of iteration to see how our model improves in relation to no of epoch and learning rate.

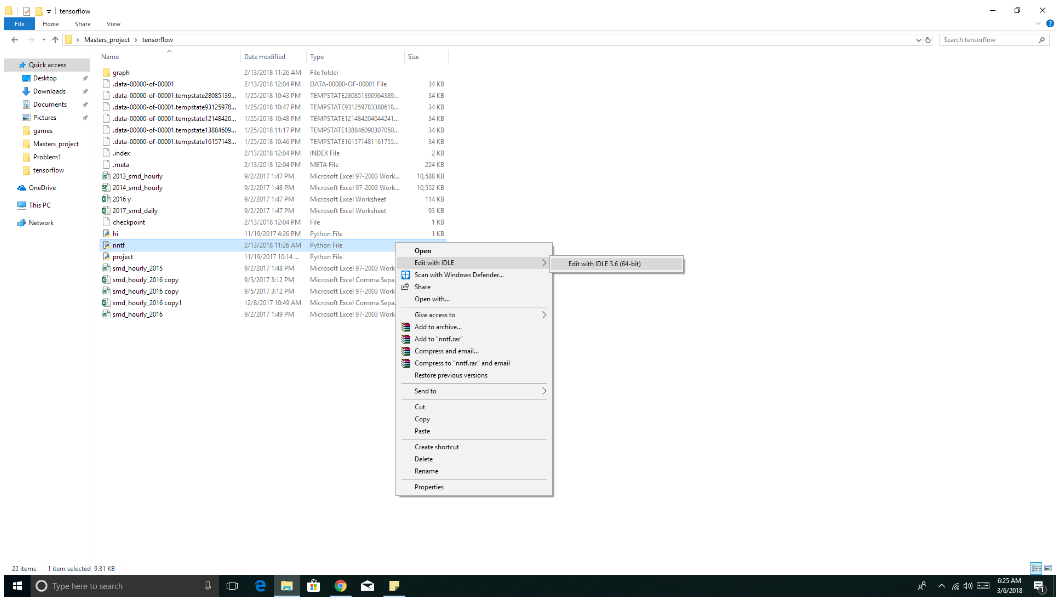
```
#session run

sess = tf.Session()

#File_Writer =
tf.summary.FileWriter('C:\\Users\\Hardik\\Desktop\\Masters_p
roject\\tensorflow\\graph', sess.graph)

sess.run(init)

File_Writer =
tf.summary.FileWriter('C:\\Users\\Hardik\\Desktop\\Masters_p
roject\\tensorflow\\graph', sess.graph)

loss_history =[]

final_accuracy =[]

final_acc =[]

final_a =[]

for i in range (training_epochs):

    sess.run(
train,{X1:train_x1,X2:train_x2,X3:train_x3,X4:train_x4,X5:tr
ain_x5,X6:train_x6,X7:train_x7,Y_:train_y} )

    losses =
sess.run(loss,{X1:train_x1,X2:train_x2,X3:train_x3,X4:train_
x4,X5:train_x5,X6:train_x6,X7:train_x7,Y_:train_y})

    accu =
sess.run(accuracy,{X1:train_x1,X2:train_x2,X3:train_x3,X4:tr
ain_x4,X5:train_x5,X6:train_x6,X7:train_x7,Y_:train_y})

    acc =
sess.run(accuracybyper,{X1:train_x1,X2:train_x2,X3:train_x3,
X4:train_x4,X5:train_x5,X6:train_x6,X7:train_x7,Y_:train_y})

    print(losses)

    print (accu)

    print (acc)
    loss_history = np.append(loss_history,losses)
    final_accuracy = np.append(final_accuracy,accu)
    final_acc = np.append(final_acc,acc)
    final_a = np.append(final_a,max(0,100-acc))
        #print(sess.run([w,b]))
save_path = saver.save(sess, model_path)
plt.plot(loss_history)
plt.show()
plt.plot(final_accuracy)
plt.show()
plt.plot(final_acc)
plt.show()
plt.plot(final_a)
plt.show()
```

After we are done with training the neural network we run it using the test data set to see if we are overfitting or not.

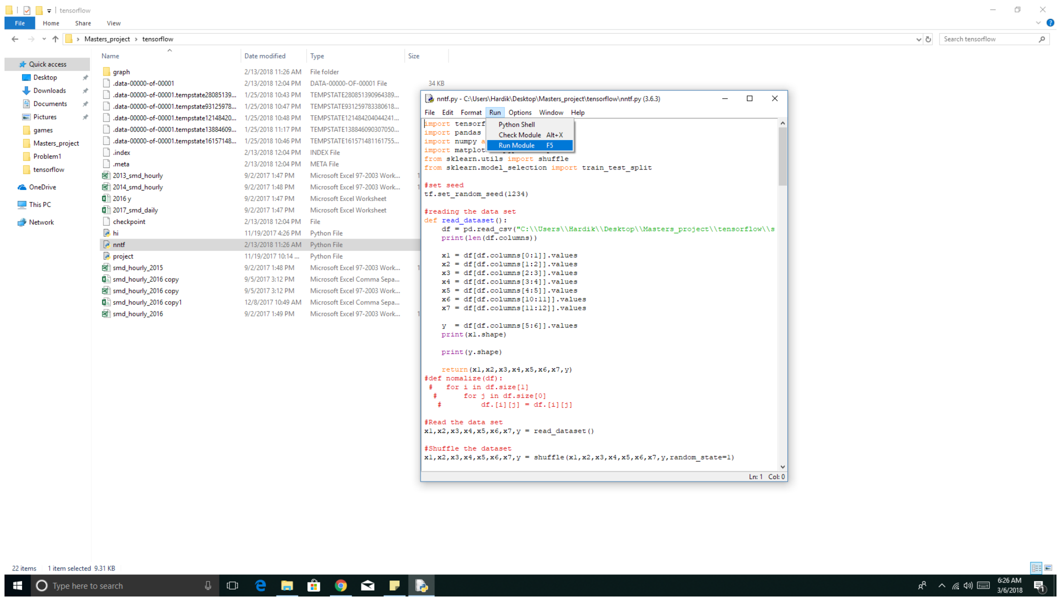
```
print("final")
print(sess.run(loss,{X1:test_x1,X2:test_x2,X3:test_x3,X4:tes
t_x4,X5:test_x5,X6:test_x6,X7:test_x7,Y_:test_y}))
```

3.2. Running the program

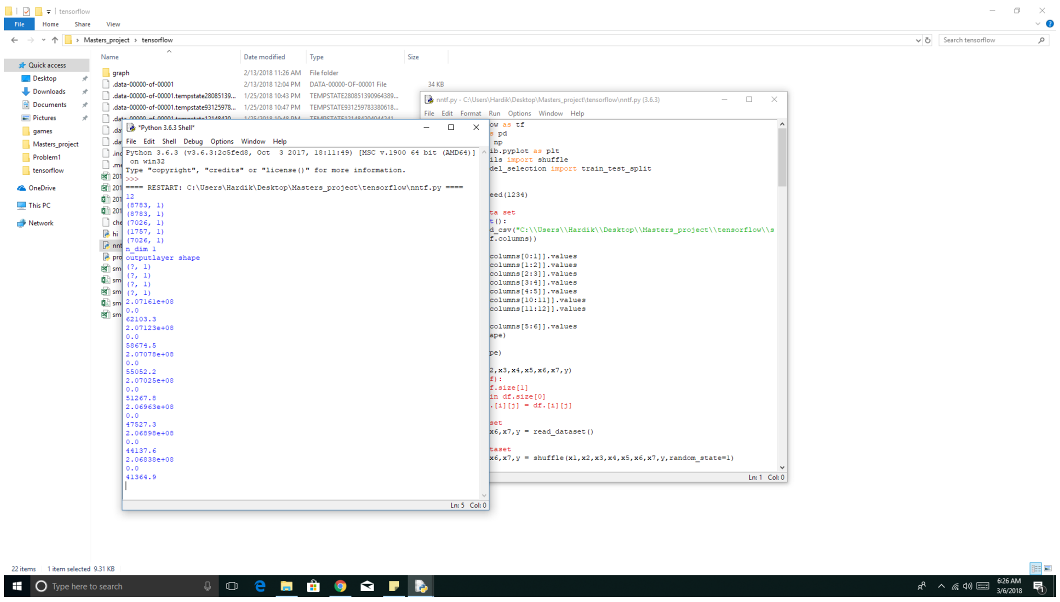
To run the program, we go to the tensorflow folder, open the nntf.py file with IDLE and press f5 to run the module.



Name	Date modified	Type	Size
graph	2/13/2018 11:26 AM	File folder	
.data-00000-of-00001	2/13/2018 12:04 PM	DATA-00000-OF-00001 File	34 KB
.data-00000-of-00001.tempstate28085139...	1/25/2018 10:43 PM	TEMPSTATE280851390964389...	34 KB
.data-00000-of-00001.tempstate93125978...	1/25/2018 10:47 PM	TEMPSTATE931259783380618...	34 KB
.data-00000-of-00001.tempstate12148420...	1/25/2018 10:48 PM	TEMPSTATE121484204044241...	34 KB
.data-00000-of-00001.tempstate13884609...	1/25/2018 11:17 PM	TEMPSTATE138846090307050...	34 KB
.data-00000-of-00001.tempstate16157148...	1/25/2018 10:46 PM	TEMPSTATE161571481161755...	34 KB
.index	2/13/2018 12:04 PM	INDEX File	2 KB
.meta	2/13/2018 12:04 PM	META File	224 KB
2013_smd_hourly	9/2/2017 1:47 PM	Microsoft Excel 97-2003 Work...	10,588 KB
2014_smd_hourly	9/2/2017 1:48 PM	Microsoft Excel 97-2003 Work...	10,552 KB
2016 y	9/2/2017 1:47 PM	Microsoft Excel Worksheet	114 KB
2017_smd_daily	9/2/2017 1:47 PM	Microsoft Excel Worksheet	93 KB
checkpoint	2/13/2018 12:04 PM	File	1 KB
hi	11/19/2017 4:26 PM	Python File	1 KB
nntf	2/13/2018 11:26 AM	Python File	
project	11/19/2017 10:14 ...	Python File	
smd_hourly_2015	9/2/2017 1:48 PM	Microsoft Excel 97-2003 Work	
smd_hourly_2016 copy	9/5/2017 3:12 PM	Microsoft Excel Comma Sepa	
smd_hourly_2016 copy	9/5/2017 3:12 PM	Microsoft Excel 97-2003 Work	
smd_hourly_2016 copy1	12/8/2017 10:49 AM	Microsoft Excel Comma Sepa	
smd_hourly_2016	9/2/2017 1:49 PM	Microsoft Excel 97-2003 Work	



```
import tensorf
import pandas
import numpy a
import matplot
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

#set seed
tf.set_random_seed(1234)

#reading the data set
def read_dataset():
    df = pd.read_csv("C:\\Users\\Hardik\\Desktop\\Masters_project\\tensorflow\\s
    print(len(df.columns))

    x1 = df[df.columns[0:1]].values
    x2 = df[df.columns[1:2]].values
    x3 = df[df.columns[2:3]].values
    x4 = df[df.columns[3:4]].values
    x5 = df[df.columns[4:5]].values
    x6 = df[df.columns[10:11]].values
    x7 = df[df.columns[11:12]].values

    y  = df[df.columns[5:6]].values
    print(x1.shape)

    print(y.shape)

    return(x1,x2,x3,x4,x5,x6,x7,y)
#def nomalize(df):
 #   for i in df.size[1]
  #      for j in df.size[0]
   #         df.[i][j] = df.[i][j]

#Read the data set
x1,x2,x3,x4,x5,x6,x7,y = read_dataset()

#Shuffle the dataset
x1,x2,x3,x4,x5,x6,x7,y = shuffle(x1,x2,x3,x4,x5,x6,x7,y,random_state=1)
```

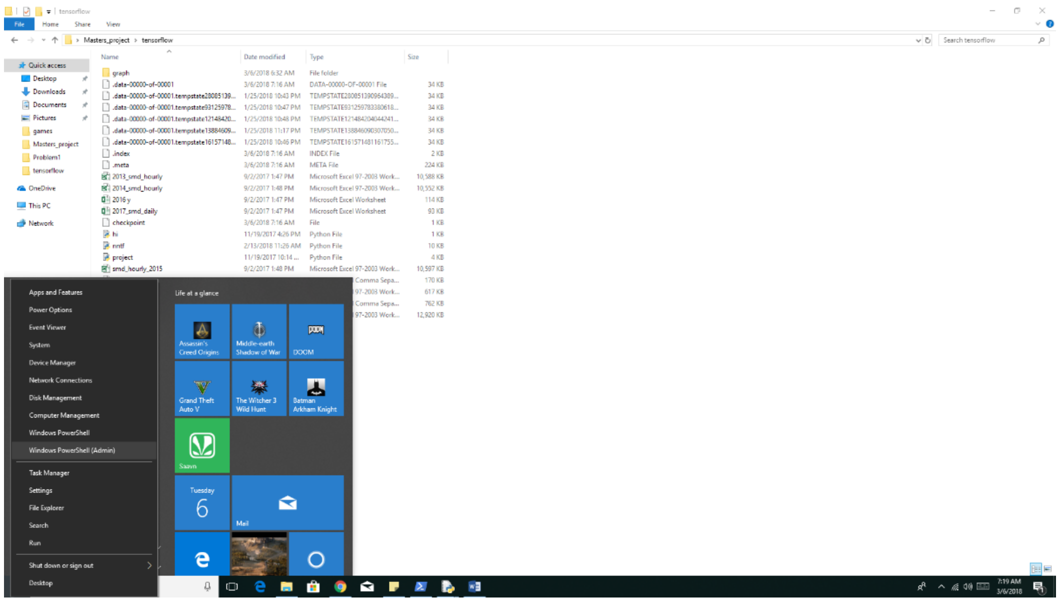


```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:\Users\Hardik\Desktop\Masters_project\tensorflow\nntf.py ====
12
(8783, 1)
(8783, 1)
(7026, 1)
(1757, 1)
(7026, 1)
n_dim 1
outputlayer shape
(?, 1)
(?, 1)
(?, 1)
(?, 1)
2.07161e+08
0.0
62103.3
2.07123e+08
0.0
58674.5
2.07078e+08
0.0
55052.2
2.07025e+08
0.0
51267.8
2.06963e+08
0.0
47527.3
2.06898e+08
0.0
44137.6
2.06838e+08
0.0
41364.9
```

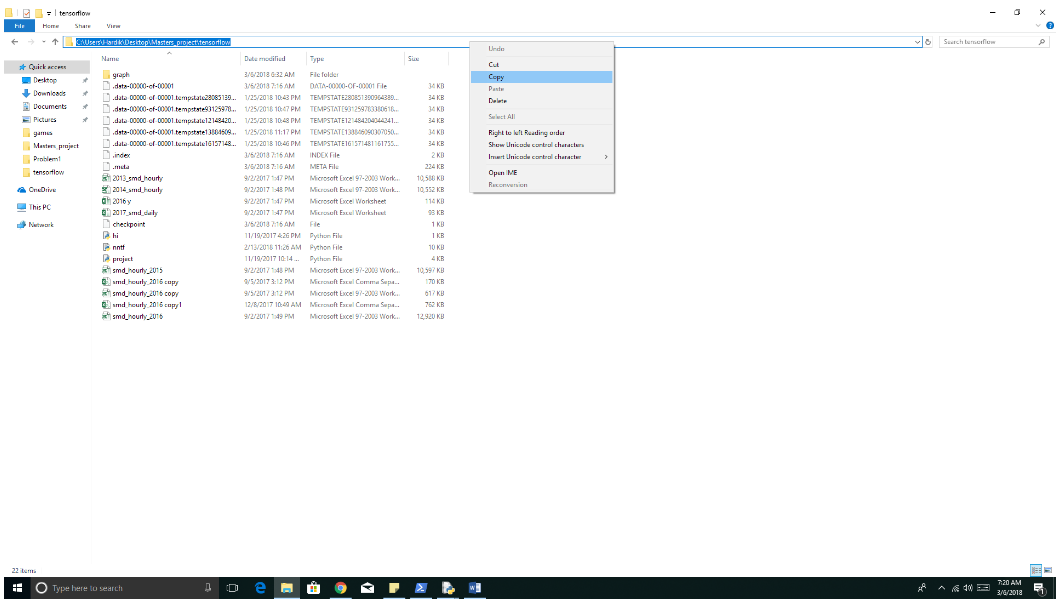
```
ow as tf
s pd
 np
ib.pyplot as plt
ils import shuffle
del_selection import train_test_split


eed(1234)

ta set
t():
d_csv("C:\\Users\\Hardik\\Desktop\\Masters_project\\tensorflow\\s
f.columns))

columns[0:1]].values
columns[1:2]].values
columns[2:3]].values
columns[3:4]].values
columns[4:5]].values
columns[10:11]].values
columns[11:12]].values

columns[5:6]].values
ape)

pe)

2,x3,x4,x5,x6,x7,y)
f):
f.size[1]
in df.size[0]
.[i][j] = df.[i][j]

set
x6,x7,y = read_dataset()

taset
x6,x7,y = shuffle(x1,x2,x3,x4,x5,x6,x7,y,random_state=1)
```

After the program has finished running, which may take from ½ hour to 10 hrs depending upon your system, we can see the computation graph (data flow graph) of out neural network.

To view the computation graph, go to tensorflow directory present in master's project folder on cmd.



Name	Date modified	Type	Size
graph	3/6/2018 6:32 AM	File folder	
.data-00000-of-00001	3/6/2018 7:16 AM	DATA-00000-OF-00001 File	34 KB
.data-00000-of-00001.tempstate28085139...	1/25/2018 10:43 PM	TEMPSTATE280851390964389...	34 KB
.data-00000-of-00001.tempstate93125978...	1/25/2018 10:47 PM	TEMPSTATE931259783380618...	34 KB
.data-00000-of-00001.tempstate12148420...	1/25/2018 10:48 PM	TEMPSTATE121484204044241...	34 KB
.data-00000-of-00001.tempstate13884609...	1/25/2018 11:17 PM	TEMPSTATE138846090307050...	34 KB
.data-00000-of-00001.tempstate16157148...	1/25/2018 10:46 PM	TEMPSTATE161571481161755...	34 KB
.index	3/6/2018 7:16 AM	INDEX File	2 KB
.meta	3/6/2018 7:16 AM	META File	224 KB
2013_smd_hourly	9/2/2017 1:47 PM	Microsoft Excel 97-2003 Work...	10,588 KB
2014_smd_hourly	9/2/2017 1:48 PM	Microsoft Excel 97-2003 Work...	10,552 KB
2016 y	9/2/2017 1:47 PM	Microsoft Excel Worksheet	114 KB
2017_smd_daily	9/2/2017 1:47 PM	Microsoft Excel Worksheet	93 KB
checkpoint	3/6/2018 7:16 AM	File	1 KB
hi	11/19/2017 4:26 PM	Python File	1 KB
nntf	2/13/2018 11:26 AM	Python File	10 KB
project	11/19/2017 10:14 ...	Python File	4 KB
smd_hourly_2015	9/2/2017 1:48 PM	Microsoft Excel 97-2003 Work...	10,597 KB
		Comma Sepa...	170 KB
		97-2003 Work...	617 KB
		Comma Sepa...	762 KB
		97-2003 Work...	12,920 KB

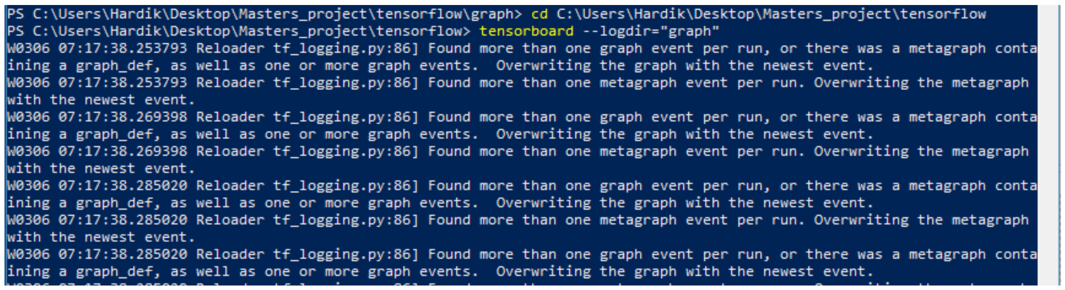


C:\Users\Hardik\Desktop\Masters_project\tensorflow

Name	Date modified	Type	Size
graph	3/6/2018 6:32 AM	File folder	
.data-00000-of-00001	3/6/2018 7:16 AM	DATA-00000-OF-00001 File	34 KB
.data-00000-of-00001.tempstate28085139...	1/25/2018 10:43 PM	TEMPSTATE280851390964389...	34 KB
.data-00000-of-00001.tempstate93125978...	1/25/2018 10:47 PM	TEMPSTATE931259783380618...	34 KB
.data-00000-of-00001.tempstate12148420...	1/25/2018 10:48 PM	TEMPSTATE121484204044241...	34 KB
.data-00000-of-00001.tempstate13884609...	1/25/2018 11:17 PM	TEMPSTATE138846090307050...	34 KB
.data-00000-of-00001.tempstate16157148...	1/25/2018 10:46 PM	TEMPSTATE161571481161755...	34 KB
.index	3/6/2018 7:16 AM	INDEX File	2 KB
.meta	3/6/2018 7:16 AM	META File	224 KB
2013_smd_hourly	9/2/2017 1:47 PM	Microsoft Excel 97-2003 Work...	10,588 KB
2014_smd_hourly	9/2/2017 1:48 PM	Microsoft Excel 97-2003 Work...	10,552 KB
2016 y	9/2/2017 1:47 PM	Microsoft Excel Worksheet	114 KB
2017_smd_daily	9/2/2017 1:47 PM	Microsoft Excel Worksheet	93 KB
checkpoint	3/6/2018 7:16 AM	File	1 KB
hi	11/19/2017 4:26 PM	Python File	1 KB
nntf	2/13/2018 11:26 AM	Python File	10 KB
project	11/19/2017 10:14 ...	Python File	4 KB
smd_hourly_2015	9/2/2017 1:48 PM	Microsoft Excel 97-2003 Work...	10,597 KB
smd_hourly_2016 copy	9/5/2017 3:12 PM	Microsoft Excel Comma Sepa...	170 KB
smd_hourly_2016 copy	9/5/2017 3:12 PM	Microsoft Excel 97-2003 Work...	617 KB
smd_hourly_2016 copy1	12/8/2017 10:49 AM	Microsoft Excel Comma Sepa...	762 KB
smd_hourly_2016	9/2/2017 1:49 PM	Microsoft Excel 97-2003 Work...	12,920 KB

type in cmd the following

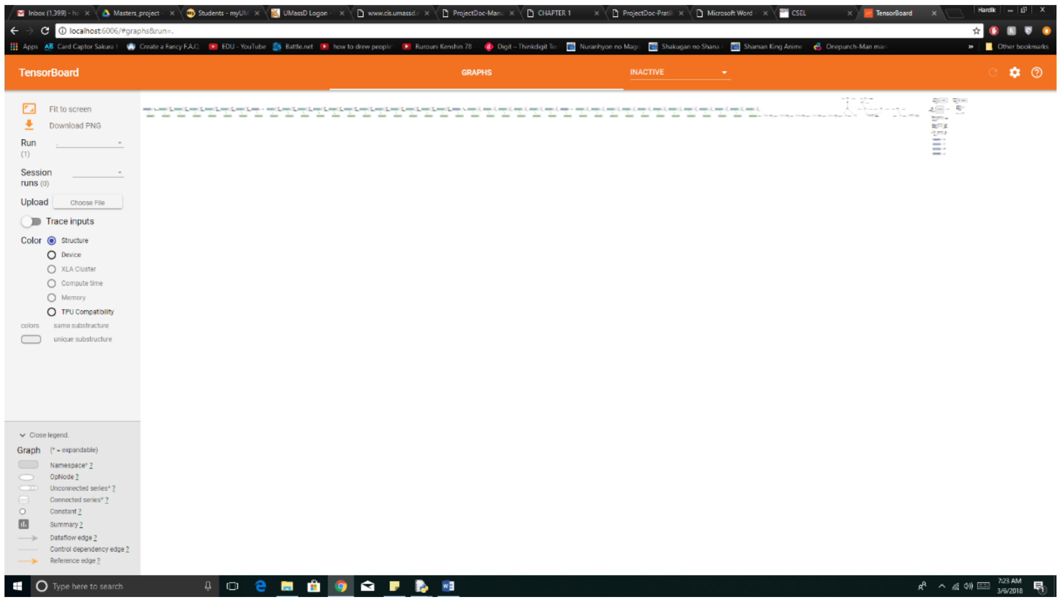
1.) tensorboard --logdir="graph"

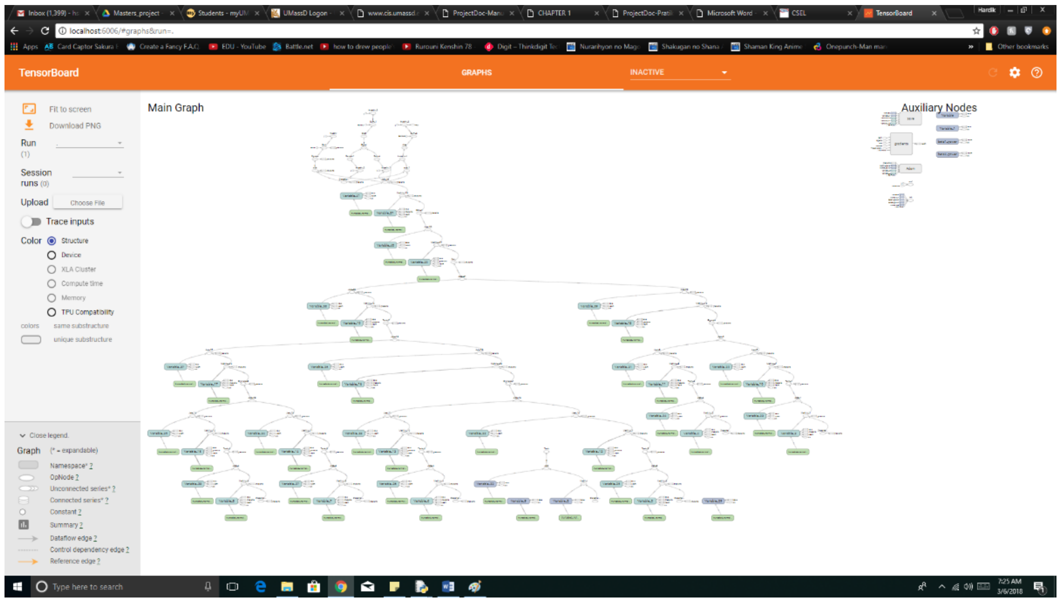


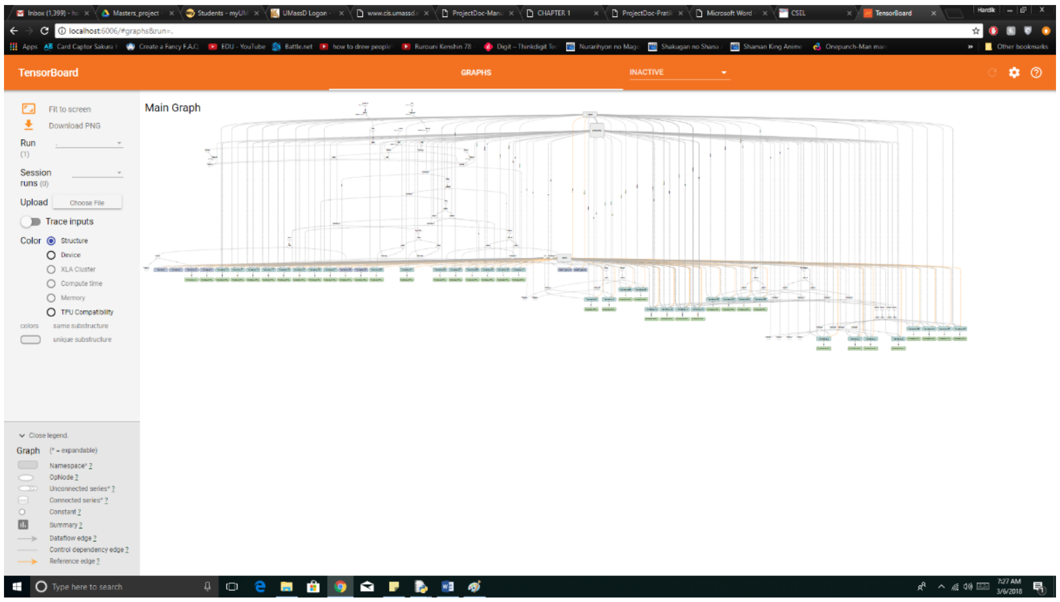
```
PS C:\Users\Hardik\Desktop\Masters_project\tensorflow\graph> cd C:\Users\Hardik\Desktop\Masters_project\tensorflow
PS C:\Users\Hardik\Desktop\Masters_project\tensorflow> tensorboard --logdir="graph"
W0306 07:17:38.253793 Reloader tf_logging.py:86] Found more than one graph event per run, or there was a metagraph conta
ining a graph_def, as well as one or more graph events.  Overwriting the graph with the newest event.
W0306 07:17:38.253793 Reloader tf_logging.py:86] Found more than one metagraph event per run. Overwriting the metagraph
with the newest event.
W0306 07:17:38.269398 Reloader tf_logging.py:86] Found more than one graph event per run, or there was a metagraph conta
ining a graph_def, as well as one or more graph events.  Overwriting the graph with the newest event.
W0306 07:17:38.269398 Reloader tf_logging.py:86] Found more than one metagraph event per run. Overwriting the metagraph
with the newest event.
W0306 07:17:38.285020 Reloader tf_logging.py:86] Found more than one graph event per run, or there was a metagraph conta
ining a graph_def, as well as one or more graph events.  Overwriting the graph with the newest event.
W0306 07:17:38.285020 Reloader tf_logging.py:86] Found more than one metagraph event per run. Overwriting the metagraph
with the newest event.
W0306 07:17:38.285020 Reloader tf_logging.py:86] Found more than one graph event per run, or there was a metagraph conta
ining a graph_def, as well as one or more graph events.  Overwriting the graph with the newest event.
```

Open web browser (to see computation graph) and go to:-

http://localhost:6006/

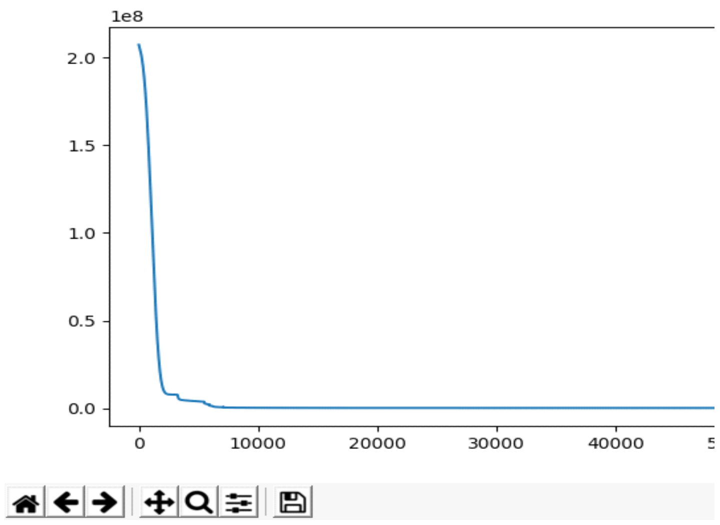

TensorBoard
GRAPHS
INACTIVE
Fit to screen
Download PNG
Run
Session runs
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
TPU Compatibility
colors
same substructure
unique substructure
Close legend.
Graph
(* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?
Type here to search


TensorBoard
GRAPHS
INACTIVE
Main Graph
Auxiliary Nodes
Fit to screen
Download PNG
Run
Session runs
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
TPU Compatibility
colors
same substructure
unique substructure
Close legend.
Graph
(* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?
Type here to search

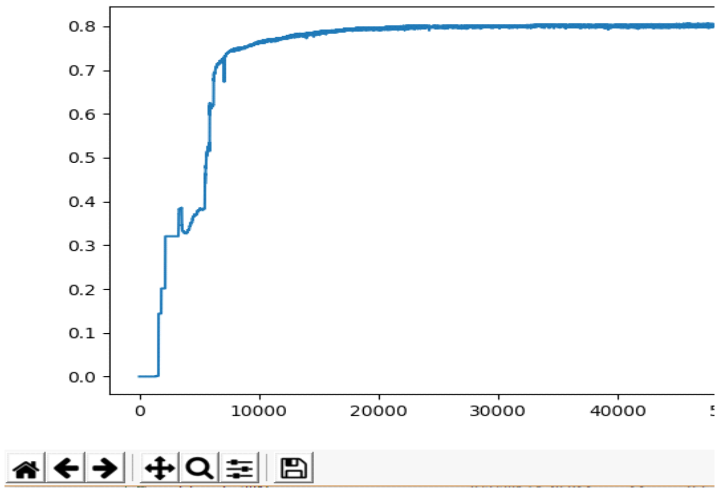


4. Result

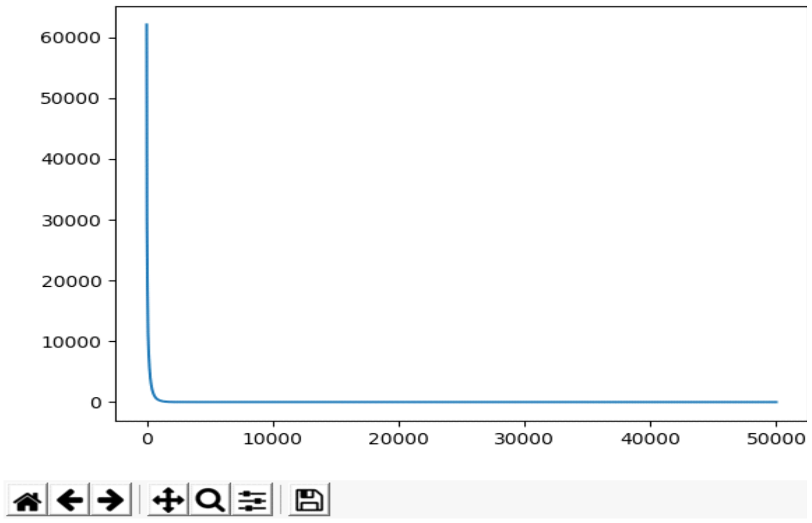
At the end of the program the final error is 2.838%, loss is ~300,000 and accuracy of +-1000-unit range of actual value is 80.3017% as shown in the figures. This neural network with more data features has some simple but effective hidden layer structure that enables us to use as few hidden layers as possible(i.e. <10) compared to 100, while predicting load values with error lower than the industry standard used by ISO New England (ISO NE) (i.e. 2.92% best case scenario).



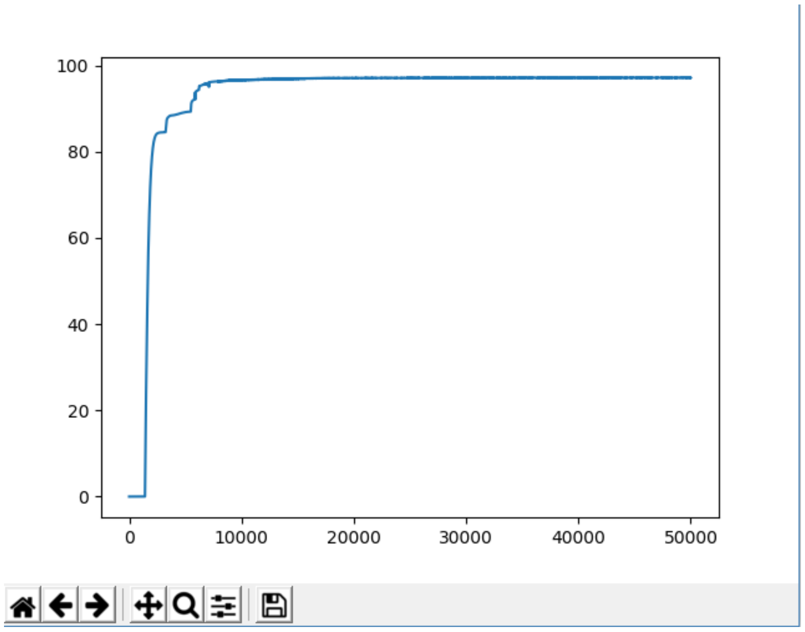
Fig(a) loss vs no of epochs



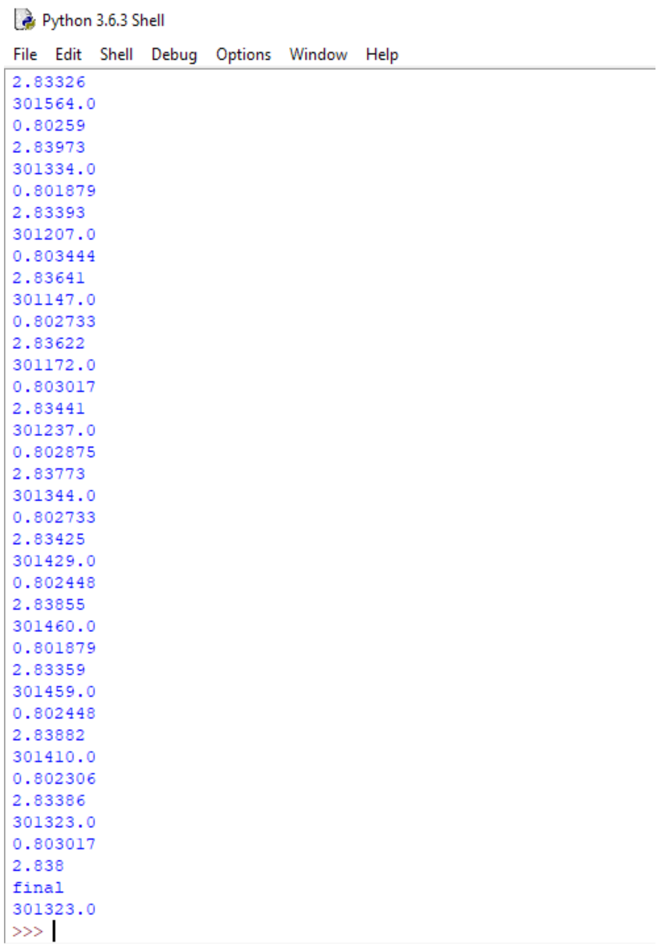
Fig(b) accuracy +-1000 units of actual value vs no of epochs



Fig(c) error percent vs no of epochs



Fig(d) accuracy



```
2.83326
301564.0
0.80259
2.83973
301334.0
0.801879
2.83393
301207.0
0.803444
2.83641
301147.0
0.802733
2.83622
301172.0
0.803017
2.83441
301237.0
0.802875
2.83773
301344.0
0.802733
2.83425
301429.0
0.802448
2.83855
301460.0
0.801879
2.83359
301459.0
0.802448
2.83882
301410.0
0.802306
2.83386
301323.0
0.803017
2.838
final
301323.0
>>> |
```

Fig(e)output at the end of the program

5. Conclusion

This project shows that with more data features directly or indirectly related to energy and some effective hidden layer structure can help us achieve better accuracy with less computation.

6. Future work

For further work we can add more features such as flag for major events (i.e. NFL, Super Bowl, etc), festivals (i.e. Christmas), holidays (i.e. New-Year), natural disaster (hurricane, snow storm, etc), which changes the lifestyle of almost everyone in New England.

7. Reference

[1] X. Fang, S. Misra, G. Xue, and D. Yang, “Smart grid—the new and improved power grid: A survey,” IEEE Commun. Surveys Tuts., vol. 14, no. 4, pp. 944–980, Dec. 2011.

[2] H. K. Alfares and M. Nazeeruddin, “Electric load forecasting: Literature survey and classification of methods,” International Journal of Systems Science, vol. 33, no. 1, 2002.

[3] F. Martinez-Alvarez, A. Troncoso, J. Riquelme, and J. A. Ruiz, “Energy time series forecasting based on pattern sequence similarity,” IEEE Transactions on Knowledge and Data Engineering, vol. 23, no. 8, 2011. - Summary of models

[4] https://www.nrdc.org/stories/how-energy-grid-works

[5] http://www.iso-ne.com/

[6] https://www.tensorflow.org/

[7] https://stackoverflow.com/

[8] https://www.youtube.com/user/sentdex/playlists

[9]http://cs229.stanford.edu/proj2014/Iliana%20Voynichka,%20Machine%20Learning%20for%20the%20Smart%20Grid.pdf

[10] https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

[11] https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications

[12] https://en.wikipedia.org/wiki/Tensor