NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EEPE19 - ARTIFICIAL NEURAL NETWORKS

PROJECT REPORT

LOAD FORECASTING USING ANN

AYUSH KUMAR SINGH
107118022

JAGDISH KURDIYA
107118038

MANDAR BURANDE
107118056

PATHIKRIT SAHA
107118071

TARUN JANGIR
107118104

I. PROBLEM STATEMENT

Load forecasting of an electrical transmission system using an Artificial Neural Network.

II. DESCRIPTION OF PROBLEM

Introduction

For optimal power operation, electrical generation must follow electrical load demand. Load varies frequently on a daily basis. If the generation is not able to cope up with the demand, there is a change in frequency. If the frequency exceeds the limits, there is a loss of synchronism which affects the power system on a great scale. In order to cope up with the demand, generation needs to predict the values of the load, and this requires the need for load forecasting. The generation, transmission and distribution utilities require some means to forecast the electrical load so they can utilize their electrical infrastructure efficiently, securely and economically. Generation utilities use electrical load forecasting techniques to optimize the power flow on the transmission network to reduce congestion and overloads. Distribution utilities would not have much interest in short term electric load forecasts as their distribution systems are predominantly radial with predictable maximum load demands.

Load forecasting is a technique by which the future load demand can be predicted based on physical factors like temperature, pressure, loading on lines, losses, weather conditions, etc. With load forecasting, it is desirable to meet the load demands in the future. Electrical failures like blackout, faults due to unnecessary loading can be avoided if proper care can be taken on the load demands.

Moreover, load demands are increasing day by day. It varies a lot on an hourly basis. The load forecasting techniques are categorised into three groups:

1. *Short-term Load forecasting(SLTF)* - The short term load forecasting represents the electric load forecast for a time interval for a few days.
2. *Medium-term Load Forecasting(MLTF)* - The medium-term load forecast represents the electric load forecast for a time interval of a few months.
3. *Long-term Load Forecasting(LTLF)* - The long term load forecast represents the electric load forecast over a time interval of years.

We are focussing on the Short term Load forecasting (SLTF) in this project. There are several techniques used and implemented to create SLTF like the linear regression technique, stochastic time series technique, General exponential smoothing technique, State-space method, Knowledge-based expert approach and the Artificial Neural Network model.

ANN techniques are highly precise and extremely useful in solving the load forecasting problem. It uses an algorithm that combines previous system load data as well as weather data and based on these data, it predicts the future load pattern.

SLTF has been extensively researched and modelled at the generation and transmission level. But the problem arises at the distribution level perhaps because the distribution utilities are concerned with generation scheduling and system peaking and these utilities are not concerned with an individual customer’s short term load demand.

Factors Affecting Electrical Load

Typically, the usage of a single electrical device on a large power system is random and usage patterns of other devices may differ from each other. There is often a large diversity in individual loads, yet when these loads are summed into larger facility loads, patterns emerge which can be statistically predicted. There are four main factors that affect electrical load:

1. *Economic Factors* - These consist of investment in the facility's infrastructure through the construction of new buildings, labs, and experiments that add load to the electric system. Funding profiles for the site dictate how and when equipment, processes and experiments can be operated.

2. *Time factors* - The three-time factors that have the most influence on electric loads are - seasonal effects, weekly- daily cycle and holidays. Seasonal effects account for long term changes in weather patterns. Weekly, daily cycles are electric load patterns that are periodic over the course of the week and each day. The daily electric profile for a holiday is similar to that of a weekend profile. The magnitudes of the electric loads are lower.

3. *Weather* - Weather factors have a significant effect on the short term electric load profile of a power system. Weather factors like humidity, solar irradiance, wind speed, barometric pressure and precipitation can affect the electric hourly load profile.

4. *Random effects* - These random disturbances can consist of significant loads that do not have a set operating schedule which makes prediction difficult. Other disturbances such as widespread employee absences(due to sickness, inclement weather, etc.) and planned or unplanned utility system outages can have significant effects on the facility's load profile.

So while constructing an Artificial neural network model, we have to keep in mind these factors that influence the electrical load.

III. SUITABILITY OF THE RESPECTIVE NEURAL NETWORK MODEL

We have chosen a Long Short Term Memory (LSTM) model to implement the load forecasting solution. A usual RNN has a short term memory. There are two major obstacles RNNs have or had to deal with, exploding gradients and vanishing gradients. LSTM networks are an extension of recurrent neural networks, which basically extends their memory. Therefore, it is well suited to learn from important experiences that have very long time lags in between.

LSTMs enable RNNS to remember their inputs over a long long period of time. This is because LSTMs contain their information in a memory, which is much like the memory of a computer because the LSTM can read, write and delete information from its memory.

This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information (for example, if it opens the gates or not), based on the importance it assigns to the information. The assignment of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which information is not.

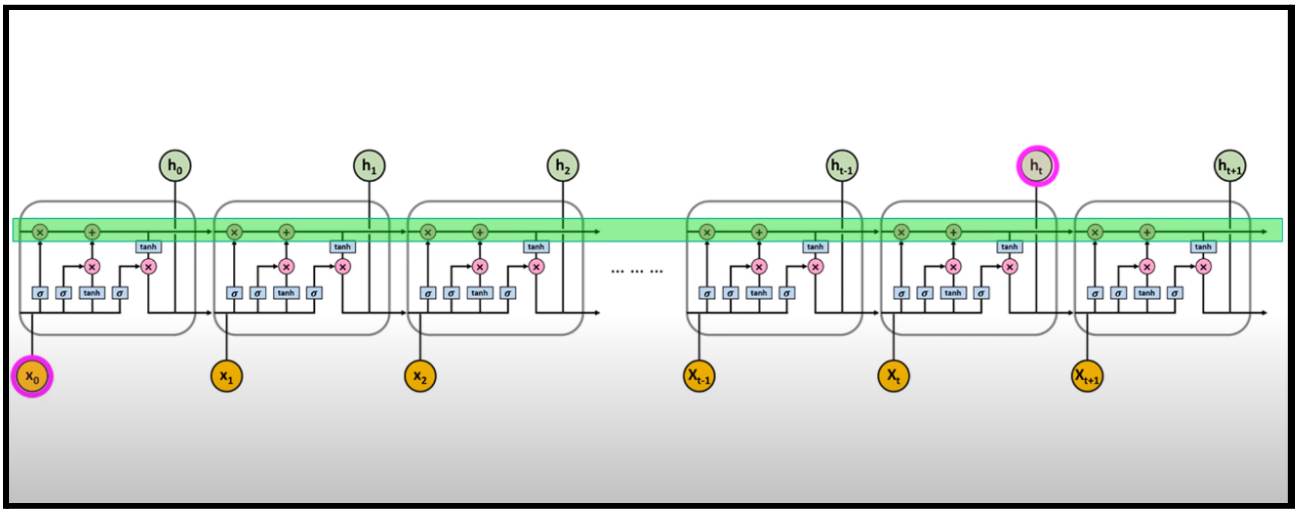


Fig.1 Core idea behind LSTM

In an LSTM, we have three gates - input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it is not important (forget gate), or let it impact the output at the output step (output gate).
The data was made stationary by detrending using one lag differentiating and was rescaled to [-1.1] scale.

This helps in faster convergence and prevents features with relatively large magnitudes like the previous year load demand to carry a larger weight during training.

The model consisted of two layers with one LSTM cell with hyperbolic tangent function as activation function followed by a dense layer without any activation.

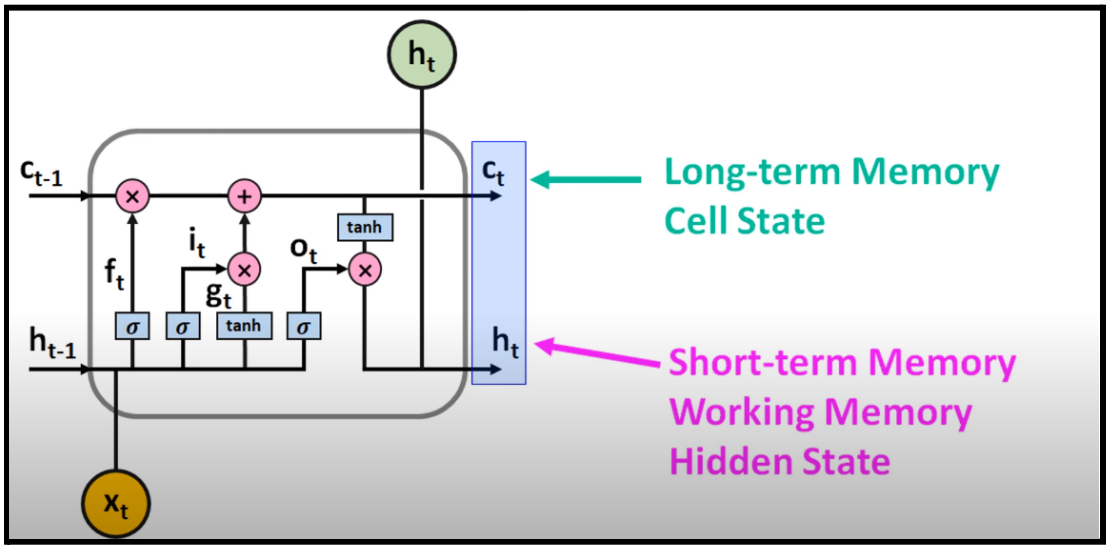


Fig.2 LSTM - C_t and h_t

IV. CODE

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


import requests

import csv

from bs4 import BeautifulSoup

from datetime import datetime, timedelta

import statsmodels.api as sm

from statsmodels.tsa import stattools

from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

from statsmodels.tsa.stattools import acf, pacf

import os

from pandas import Series

from pandas import concat

from pandas import read_csv

from sklearn.metrics import mean_squared_error

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from math import sqrt

from matplotlib import pyplot

from numpy import array

from sklearn.preprocessing import MinMaxScaler
```

```
os.environ['CUDA_VISIBLE_DEVICES']=''

%matplotlib inline

plt.rcParams['figure.figsize'] = (15, 6)


def get_load_data(date):

    url = 'http://www.delhisldc.org/Loaddata.aspx?mode='

    print('Scraping ' + date, end=' ')

    resp = requests.get(url + date) # send a get request to the url, get response

    soup = BeautifulSoup(resp.text, 'lxml') # Yummy HTML soup

    table = soup.find('table', {'id':'ContentPlaceHolder3_DGGridAv'}) # get the table
from html

    trs = table.findAll('tr') # extract all rows of the table


    if len(trs[1:])==288: # no need to create csv file, if there's no data

        with open('monthdata.csv', 'a') as f:  #'a' makes sure the values are appended
at the end of the already existing file

            writer = csv.writer(f)

            for tr in trs[1:]:

                time, delhi = tr.findChildren('font')[:2]

                writer.writerow([date + ' ' + time.text, delhi.text])

    if len(trs[1:]) != 288:

        print('Some of the load values are missing..')

    else:

        print('Done')


for i in range(31, 0, -1):

    yesterday = datetime.today() - timedelta(i)
```

```
    yesterday = yesterday.strftime("%d/%m/%Y")
    get_load_data(yesterday)
data = pd.read_csv('monthdata.csv', header=None, names=['datetime', 'load'],
index_col=[0], parse_dates=[0], infer_datetime_format=True)
data.info()
for i in range(31, 0, -1):
    yesterday = datetime.today() - timedelta(i)
    yesterday = yesterday.strftime('%Y-%m-%d')
    try:
        print(data[yesterday].shape[0])
    except:
        print(yesterday, 'not found')
data.shape
data.shape[0] / 288
data.head()
data.head(1)
data.plot()
plt.rcParams['figure.figsize'] = (20, 6)
plt.show()
set(data.index.date)
data[:'2021-05-20'].plot()
plt.rcParams['figure.figsize'] = (15, 6)
plt.show()
data.shape
# number of unique dates in the data
len(set(data.index.date))
data.tail()
```

```
plt.rcParams['figure.figsize'] = (15, 6)

decompfreq = 288 #daily freq

from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(data, period=decompfreq, model='aditive')


result.plot()

plt.show()
```

#Making the data stationary

#Detrending

```
dt_data = data.diff(1).dropna()  # detrended data

plt.rcParams['figure.figsize'] = (15, 6)
decompfreq = 288 #daily freq
result = seasonal_decompose(dt_data, period=decompfreq, model='aditive')
result.plot()
plt.show()
```

#Removing seasonality

```
dt_data.plot()

lag_pacf = pacf(dt_data['load'].values, nlags = 2000)

lag_acf = acf(dt_data['load'].values, nlags = 2000)

np.argsort(lag_pacf)

np.argsort(lag_acf)

plt.subplot(121)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--')
plt.axhline(y=-1.96/np.sqrt(len(dt_data['load'])), linestyle='--')
plt.axhline(y=1.96/np.sqrt(len(dt_data['load'])), linestyle='--')
```

```
# critical value determination: https://stats.stackexchange.com/a/185553/181916

plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--')
plt.axhline(y=-1.96/np.sqrt(len(dt_data['load'])), linestyle='--')
plt.axhline(y=1.96/np.sqrt(len(dt_data['load'])), linestyle='--')

# critical value determination: https://stats.stackexchange.com/a/185553/181916

ds_dt_data = dt_data.diff(288).dropna()  # deseasonalized + detrended data

decompfreq = 288 #daily freq
result = seasonal_decompose(ds_dt_data, period=decompfreq, model='aditive')
result.plot()
plt.show()

ds_dt_data.plot()
```

#Rescale the data

```
ds_dt_data.shape

scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(ds_dt_data['load'].values.reshape(-1, 1))

ds_dt_data['load1'] = ds_dt_data.shift(-1)['load']
ds_dt_data['load2'] = ds_dt_data.shift(-2)['load']
ds_dt_data['load3'] = ds_dt_data.shift(-3)['load']
ds_dt_data = ds_dt_data.dropna()

ds_dt_data = scaler.transform(ds_dt_data)  # don't use .values

split_idx = int(0.9 * len(ds_dt_data))
train, val = ds_dt_data[:split_idx], ds_dt_data[split_idx:]

train.shape, val.shape

X, y = train[:, 0], train[:, 1:]
X_val, y_val = val[:, 0], val[:, 1:]

y_val
```

```
plt.plot(range(len(X)), X)
plt.plot(range(len(X), len(X_val) + len(X)), X_val);
```

#LSTM model development

#The LSTM layer expects input to be in a matrix with the dimensions: [samples, time steps, features].

```
X.shape, y.shape

X = X.reshape(X.shape[0], 1, 1)

model = Sequential()
model.add(LSTM(1, batch_input_shape=(1, X.shape[1], X.shape[2]), stateful=True))
model.add(Dense(y.shape[1]))
model.compile(loss='mean_squared_error', optimizer='adam')

# fit network
for i in range(10):
    model.fit(X, y, epochs=1, batch_size=1, verbose=0, shuffle=False)
    model.reset_states()

# make one forecast with an LSTM,
def forecast_lstm(model, X, n_batch):
    # reshape input pattern to [samples, timesteps, features]
    X = X.reshape(1, 1, len(X))
    # make forecast
    forecast = model.predict(X, batch_size=n_batch)
    # convert to array
    return [x for x in forecast[0, :]]

def make_forecasts(model, n_batch, train, test, n_lag, n_seq):
    forecasts = list()
    for i in range(len(test)):
        X, y = test[i, 0:n_lag], test[i, n_lag:]
        # make forecast
        forecast = forecast_lstm(model, X, n_batch)
        # store the forecast
        forecasts.append(forecast)
    return forecasts
forecasts = make_forecasts(model, 1, train, val, 1, 3)
```

```
forecasts

y_val_hat = forecasts

y_val_hat.__len__(), y_val.shape

plt.plot(y_val)
plt.plot(y_val_hat)

y_val_hat = scaler.inverse_transform(np.asarray(y_val_hat).reshape(-1, 1))
y_val = scaler.inverse_transform(np.asarray(y_val[:, 0]).reshape(-1, 1))

plt.plot(y_val)
plt.plot(y_val_hat)

val_starting_idx = 1+288+2+split_idx # 1 = detrend, 288 = deseasonalize, 2=input seq
created nans
for i in range(len(y_val_hat)):
    extra = dt_data['load'].values[val_starting_idx + i - 288] +
data['load'].values[val_starting_idx + i - 1]
#     y_val_hat[i] += extra
#     y_val[i] += extra
    print(y_val_hat[i], y_val[i], extra)
```

#so, the difference values are so large in comparison to the #predictions that it makes minuscule change in the y_val, and #y_val_hat plots

```
plt.plot(data['load'].values[1+288+2+split_idx:])
plt.plot(y_val_hat)

np.array([1]).shape
```

#Forecasting with the predicted values instead of val set values:

```
def make_forecasts(model, n_batch, train, test, n_lag, n_seq):
    forecasts = list()
    X = test[0, 0:n_lag]
    for i in range(len(test)):
        X = X.reshape(1, 1, len(X))
```

```
        # make forecast
        forecast = model.predict(X, batch_size=n_batch)[0, 0]
        X = np.array([forecast])
        # store the forecast
        forecasts.append(forecast)
    return forecasts
```

#Version 2

#Use last nlags days values at time t to predict the load of today's time t load

```
data.head(3)

data.at_time('00:05:00').plot()

df = pd.DataFrame(columns=['time'] + list(map(str, range(30))))

for idx, time in enumerate(sorted(set(data.index.time))):
    df.loc[idx] = [time.strftime(format='%H:%M:%S')] +
list(data.at_time(time)['load'].values)
#     data.at_time(time).plot()
#     if idx>10: break

df.head()

df.index = df['time']
df = df.drop('time', 1)

df.head()

df.loc['00:00:00']

plt.rcParams['figure.figsize'] = (15, 6)
decompfreq = 1 #daily freq

result = seasonal_decompose(df.loc['00:00:00'], period=decompfreq, model='aditive')

result.plot()
plt.show()
```

#Only trend is present not seasonality

```
dt_df = df.diff(1, axis=1)

dt_df.head()

plt.rcParams['figure.figsize'] = (15, 6)
decompfreq = 1 #daily freq

result = seasonal_decompose(dt_df.loc['00:00:00'].dropna(), period=decompfreq,
model='aditive')

result.plot()
plt.show()

dt_df = dt_df.dropna(axis=1)

scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(dt_df.values.reshape(-1, 1))

dt_df = scaler.transform(dt_df)  # df is now a numpy array

split_idx = int(len(dt_df) * 0.8)
train, val = dt_df[:split_idx, :], dt_df[split_idx:, :]

train.shape, val.shape

def prepare_data(data, nlags):
    '''prepares data for LSTM model, x=last nlags values, y=(nlags+1)'th value'''
    data_x, data_y = [], []
    for i in range(data.shape[0]):
        for j in range(0, data.shape[1]-nlags):
            data_x.append(data[i, j:j+nlags])
            data_y.append(data[i, j+nlags])
    data_x = np.array(data_x)
    data_y = np.array(data_y).reshape(-1, 1)
    return data_x, data_y

nlags = 20
train_x, train_y = prepare_data(train, nlags)
val_x, val_y = prepare_data(val, nlags)

train_x.shape, train_y.shape, val_x.shape, val_y.shape
```

#Model training

```
train_x = train_x.reshape(train_x.shape[0], 1, nlags)
val_x = val_x.reshape(val_x.shape[0], 1, nlags)

model = Sequential()
model.add(LSTM(1, batch_input_shape=(1, train_x.shape[1], train_x.shape[2]),
stateful=True))
model.add(Dense(train_y.shape[1]))
model.compile(loss='mean_squared_error', optimizer='adam')

# fit network
for i in range(10):
    model.fit(train_x, train_y, epochs=1, batch_size=1, verbose=1,
validation_data=(val_x, val_y), shuffle=False)
    model.reset_states()

val_y_pred = model.predict(val_x, batch_size=1)

plt.plot(val_y[:100])
plt.plot(val_y_pred[:100])

inverted_val_y = scaler.inverse_transform(val_y)
inverted_val_y_pred = scaler.inverse_transform(val_y_pred)

plt.plot(inverted_val_y[:100])
plt.plot(inverted_val_y_pred[:100])

data.tail()
```

#let's predict yesterday's load which is not present in monthdata.csv file as of now

#we need 288, 1, 20 matrix with last 20 days load for each 288 time stamp

```
df.head()

df_last_21 = df.loc[:, '9':]

df_last_21.head()
```

```
dt_df_last_20 = df_last_21.diff(1, axis=1).dropna(axis=1)  #taking last 21 days,
differencing and dropping the nan value

dt_df_last_20 = scaler.transform(dt_df_last_20)  # df is now a numpy array

X = dt_df_last_20.reshape(dt_df_last_20.shape[0], 1, nlags)  # nlags=20

Y = model.predict(X, batch_size=1)  # predict for today's values

inv_Y = scaler.inverse_transform(Y)  # invert to detrended values' scale

inv_Y.shape

rescaled_Y = [x+y for x, y in zip(inv_Y[:, 0], df.iloc[:, -1])]  # last day's values
added to inv_Y to get it to original scale

def get_load_data(date):
    load=[]
    url = 'http://www.delhisldc.org/Loaddata.aspx?mode='
    print('Scraping ' + date, end=' ')
    resp = requests.get(url + date) # send a get request to the url, get response
    soup = BeautifulSoup(resp.text, 'lxml') # Yummy HTML soup
    table = soup.find('table', {'id':'ContentPlaceHolder3_DGGridAv'}) # get the table
from html
    trs = table.findAll('tr') # extract all rows of the table
    if len(trs[1:])==288: # no need to create csv file, if there's no data
        with open('monthdata.csv', 'a') as f:  #'a' makes sure the values are appended
at the end of the already existing file

            for tr in trs[1:]:
                time, delhi = tr.findChildren('font')[:2]
                load.append(delhi.text)
    if len(trs[1:]) != 288:
        print('Some of the load values are missing..')
    else:
        print('Done')
    return load

yesterday = datetime.today() - timedelta(1)
yesterday = yesterday.strftime('%d/%m/%Y')
load = get_load_data(yesterday)

load = [float(x) for x in load]
```

```
val_RMSE = np.sqrt(np.sum(np.square(np.array(inverted_val_y_pred) -
np.array(inverted_val_y))) / len(inverted_val_y))
RMSE = np.sqrt(np.sum(np.square(np.array(rescaled_Y) - load)) / len(load))
print(val_RMSE, RMSE)

plt.plot(load, label= "actual load", color="red")
plt.plot(rescaled_Y, label= "predicted load", color="green")
```

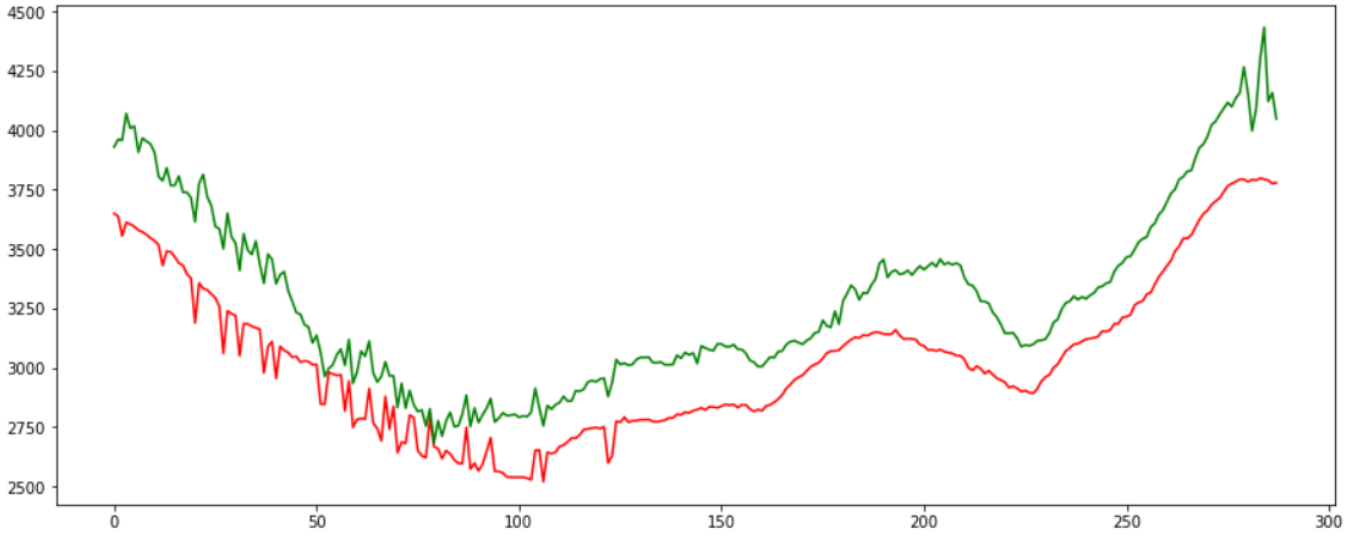
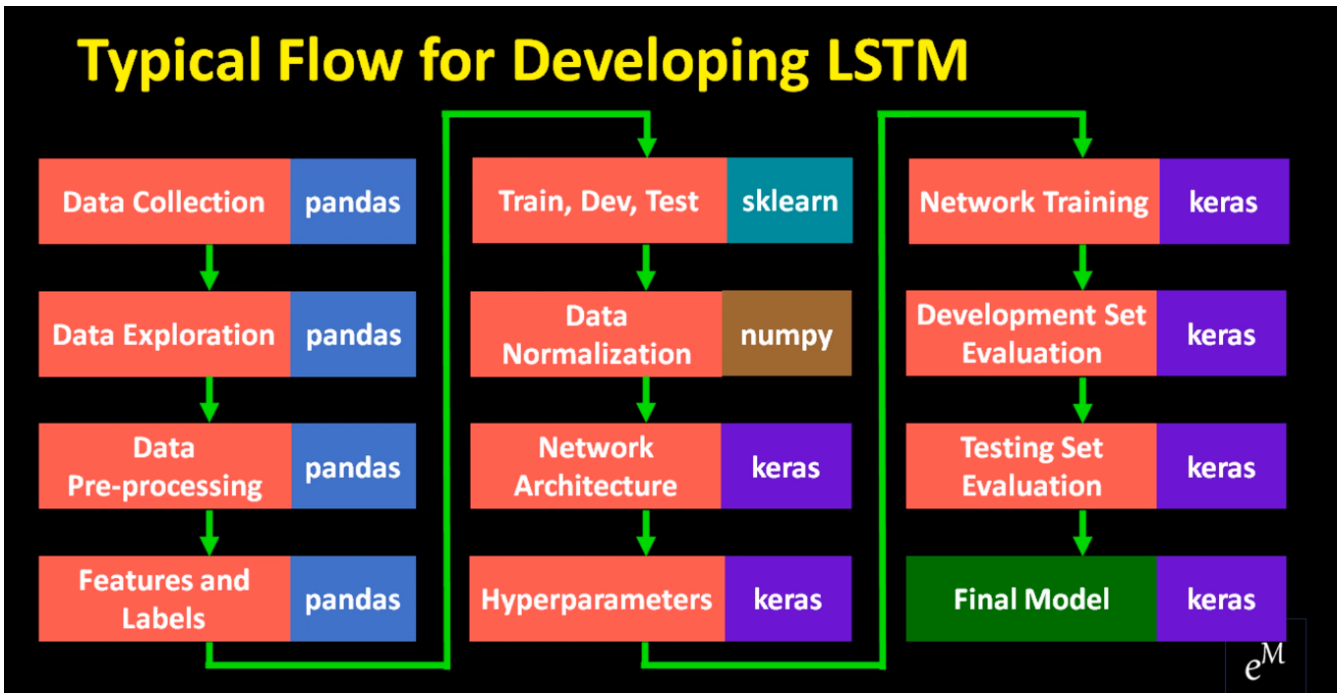


Fig.3 Output Graph

V. METHODOLOGY

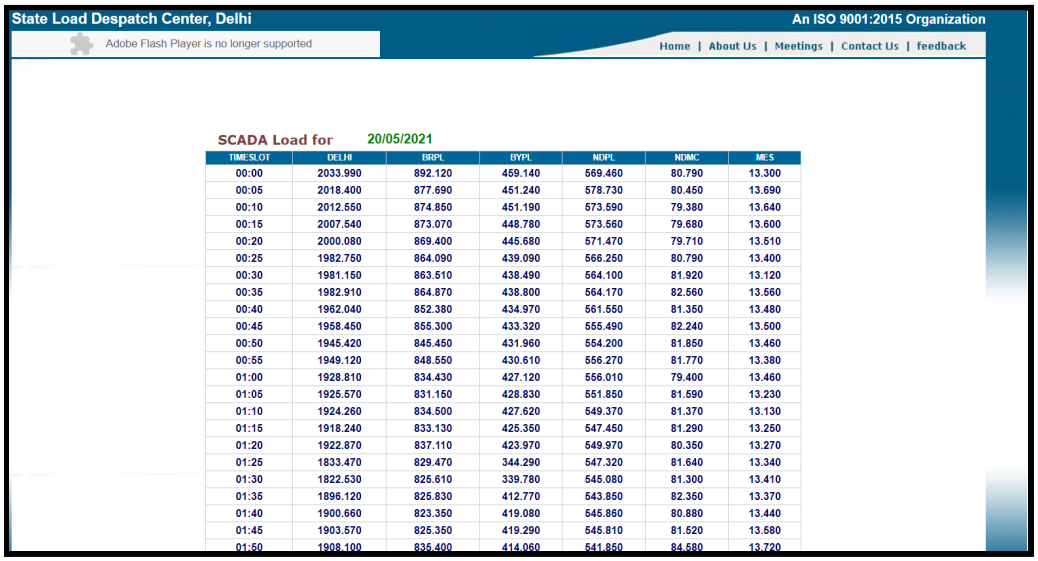


Data Collection:

SCADA Load data is collected from Delhi State Load Dispatch Centre (SLDC) site:
For example: data for 20th May, 2021 can be collected using the link:
http://www.delhisldc.org/Loaddata.aspx?mode=20/05/2021

We are using load data from State Load Dispatch Center (SLDC), Delhi. Load data is available at a time step of 5 minutes, a totally of 288 values for each day. A general trend of the load data can be seen in the day-wise plot and month-wise plot in Fig(1) and Fig(2) respectively. This data is being extracted from the SLDC website of Delhi by an automated script. The mentioned website updates the data every 5 minutes.

Thus we have data in the time steps of 5 minutes. We acquired the data of the past year till date, from the SLDC Delhi website which made the information public. However, there are certain days when the SLDC website was not functional due to maintenance or any other factors, which implies the data collected might have some missing values.


State Load Despatch Center, Delhi

An ISO 9001:2015 Organization

Adobe Flash Player is no longer supported

Home | About Us | Meetings | Contact Us | feedback

SCADA Load for 20/05/2021

TIMESLOT	DELHI	BRPL	BYPL	NDPL	NDMC	MES
00:00	2033.990	892.120	459.140	569.460	80.790	13.300
00:05	2018.400	877.690	451.240	578.730	80.450	13.690
00:10	2012.550	874.850	451.190	573.590	79.380	13.640
00:15	2007.540	873.070	448.780	573.560	79.680	13.600
00:20	2000.080	869.400	445.680	571.470	79.710	13.510
00:25	1982.750	864.090	439.090	566.250	80.790	13.400
00:30	1981.150	863.510	438.490	564.100	81.920	13.120
00:35	1982.910	864.870	438.800	564.170	82.560	13.560
00:40	1962.040	852.380	434.970	561.550	81.350	13.480
00:45	1958.450	855.300	433.320	555.490	82.240	13.500
00:50	1945.420	845.450	431.960	554.200	81.850	13.460
00:55	1949.120	848.550	430.610	556.270	81.770	13.380
01:00	1928.810	834.430	427.120	556.010	79.400	13.460
01:05	1925.570	831.150	428.830	551.850	81.590	13.230
01:10	1924.260	834.500	427.620	549.370	81.370	13.130
01:15	1918.240	833.130	425.350	547.450	81.290	13.250
01:20	1922.870	837.110	423.970	549.970	80.350	13.270
01:25	1833.470	829.470	344.290	547.320	81.640	13.340
01:30	1822.530	825.610	339.780	545.080	81.300	13.410
01:35	1896.120	825.830	412.770	543.850	82.350	13.370
01:40	1900.660	823.350	419.080	545.860	80.880	13.440
01:45	1903.670	825.350	419.290	545.810	81.520	13.580
01:50	1908.100	835.400	414.060	541.850	84.580	13.720

Fig.4 SCADA Load data for Delhi (20/05/2021)

Data Exploration:

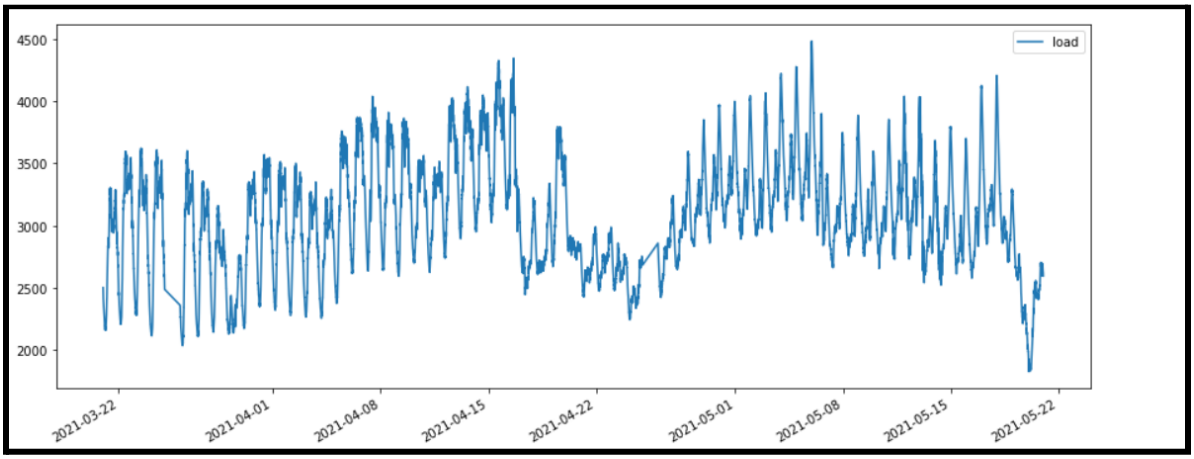


Fig.5 Load data of past two months

Data Preprocessing

For RNN, LSTM and GRU models Data was made stationary by detrending using one lag differencing and was re-scaled to [-1, 1] scale. This helps in faster convergence and prevents features with relatively large magnitudes like the previous year load demand to carry a larger weight during training.

The model was trained on the last 60 days of data, with each training data vector containing the load of a specific time for the last 10 days and the label was the load of the 11th day at the same corresponding time.

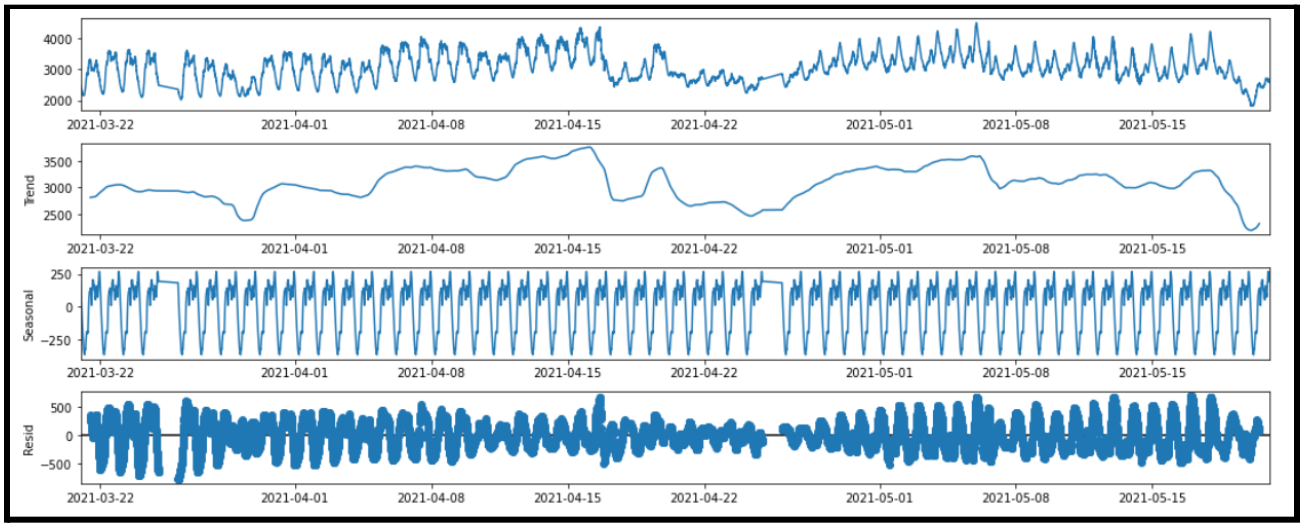


Fig.6 Seasonally decomposed data

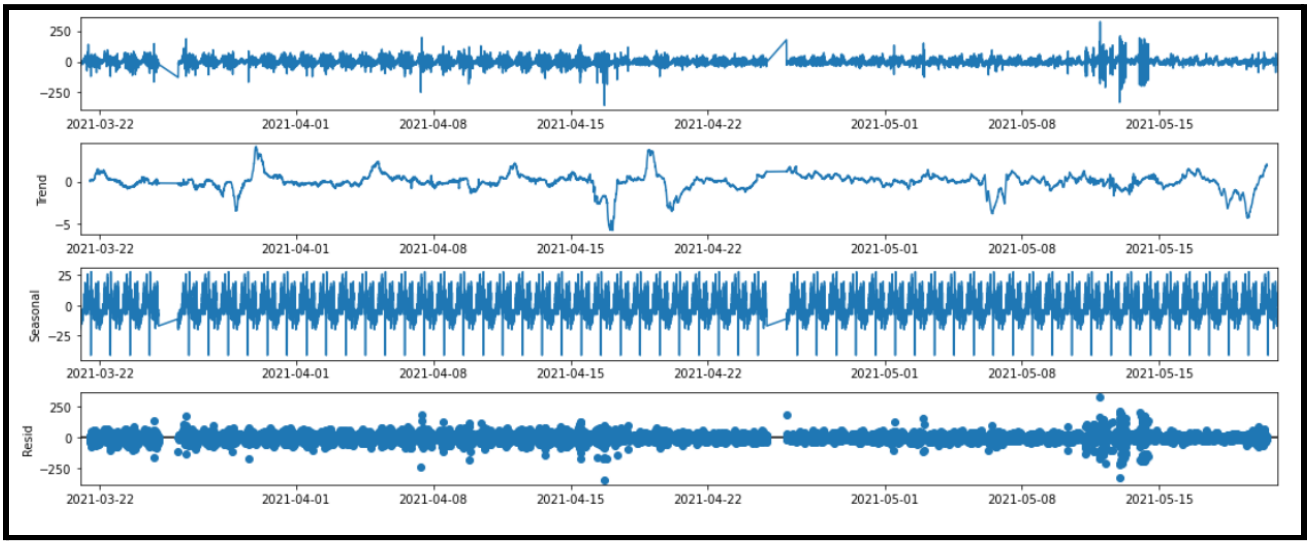
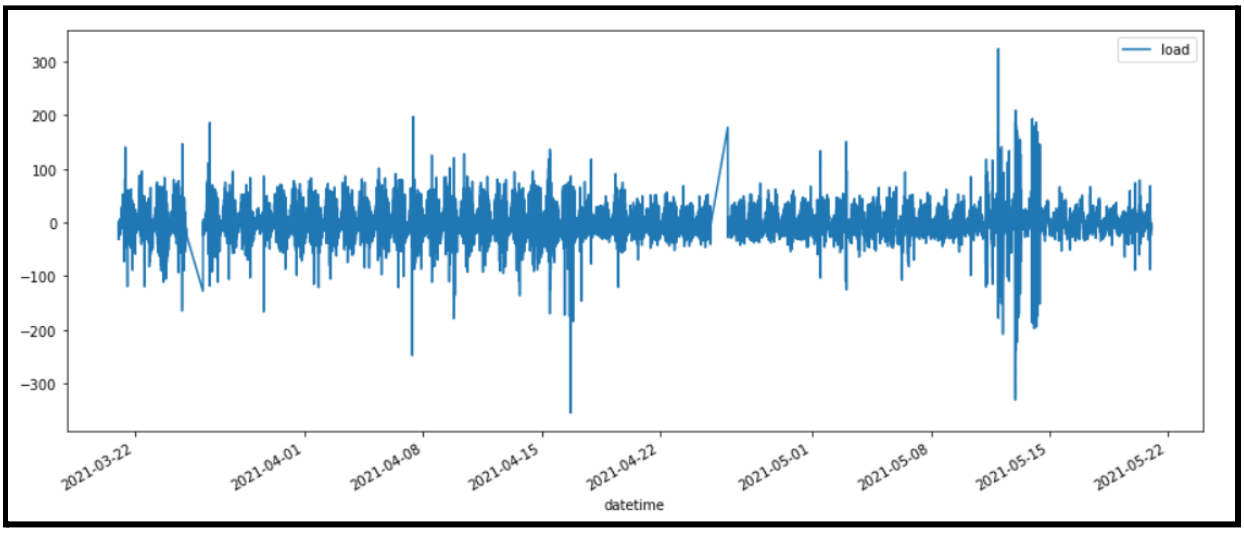
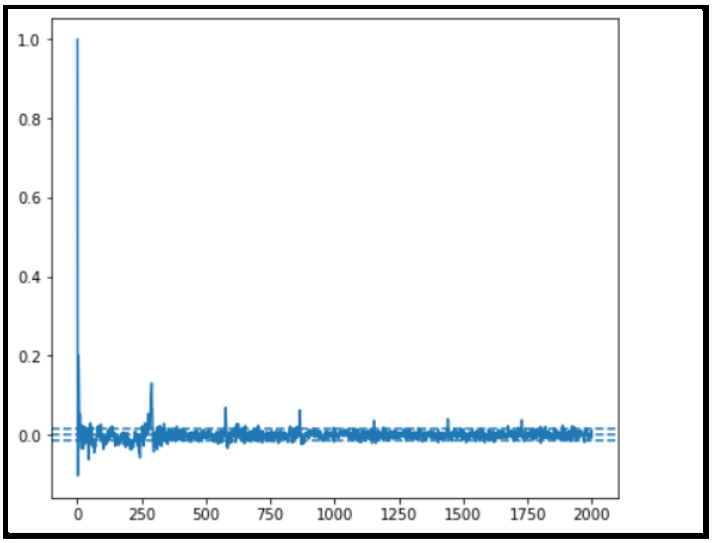
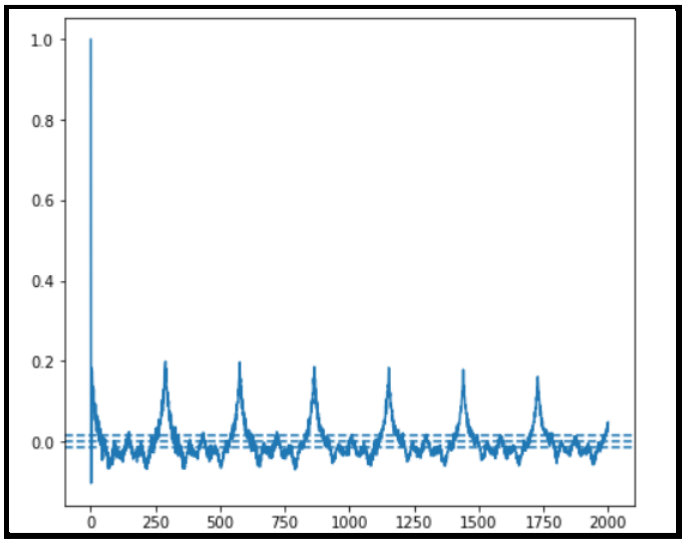


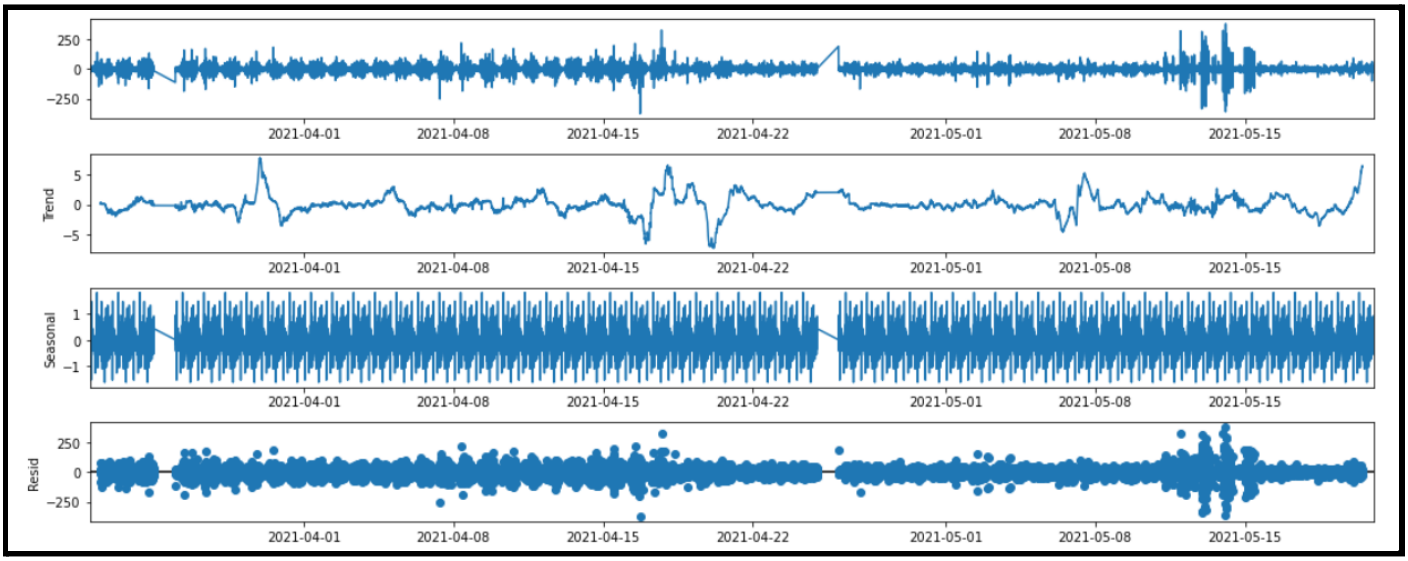
Fig.7 Detrended data

Removing Seasonality









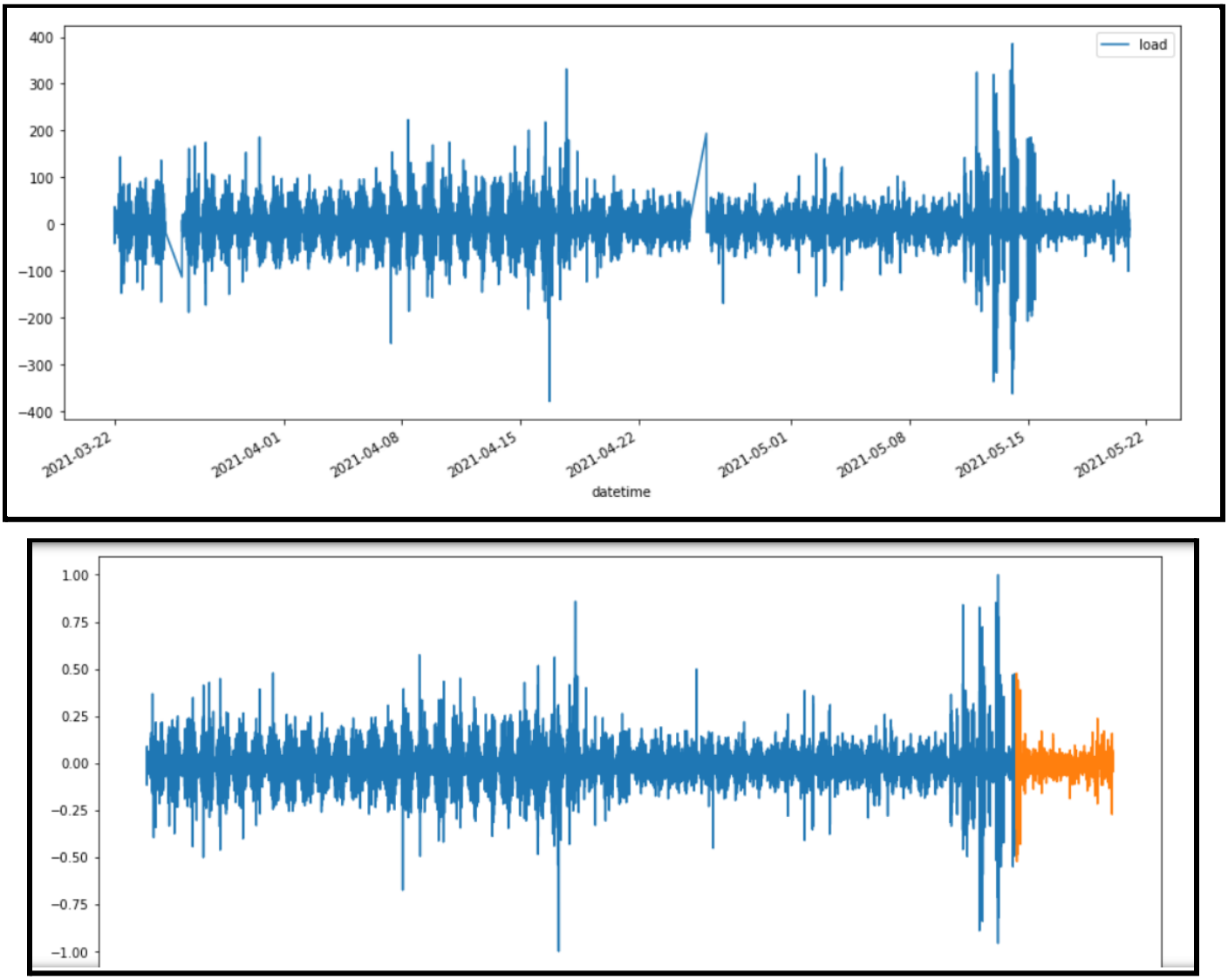


Fig.8 Normalising the data

Model Architecture: The model consisted of two layers one with one LSTM cell with hyperbolic tangent function (tanh) as activation function followed by a dense layer without any activation. Metric used for loss was mean square error optimized with Adam optimizer with a learning rate of 0.001. The model was trained for 15 epochs. The code was written in Python using Keras library.

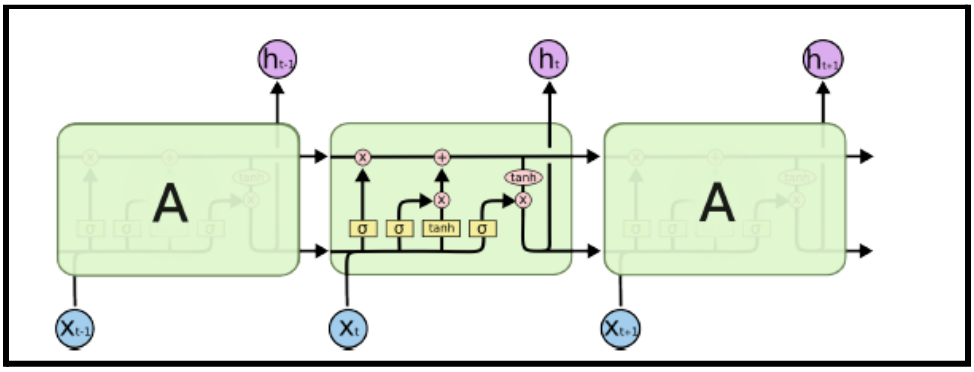
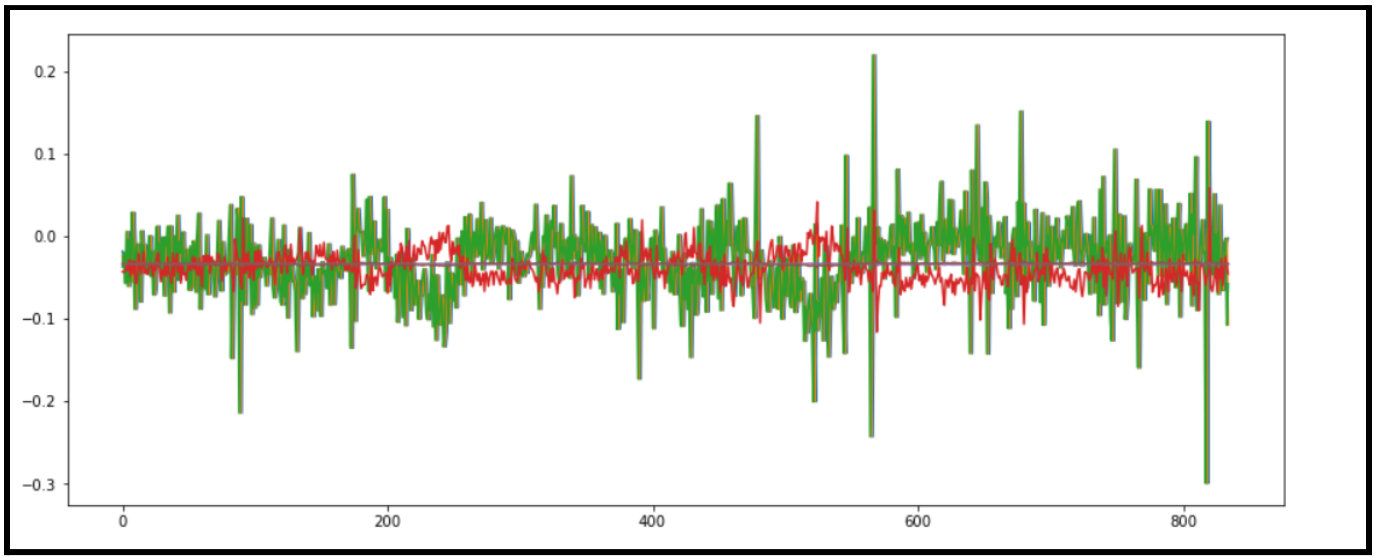
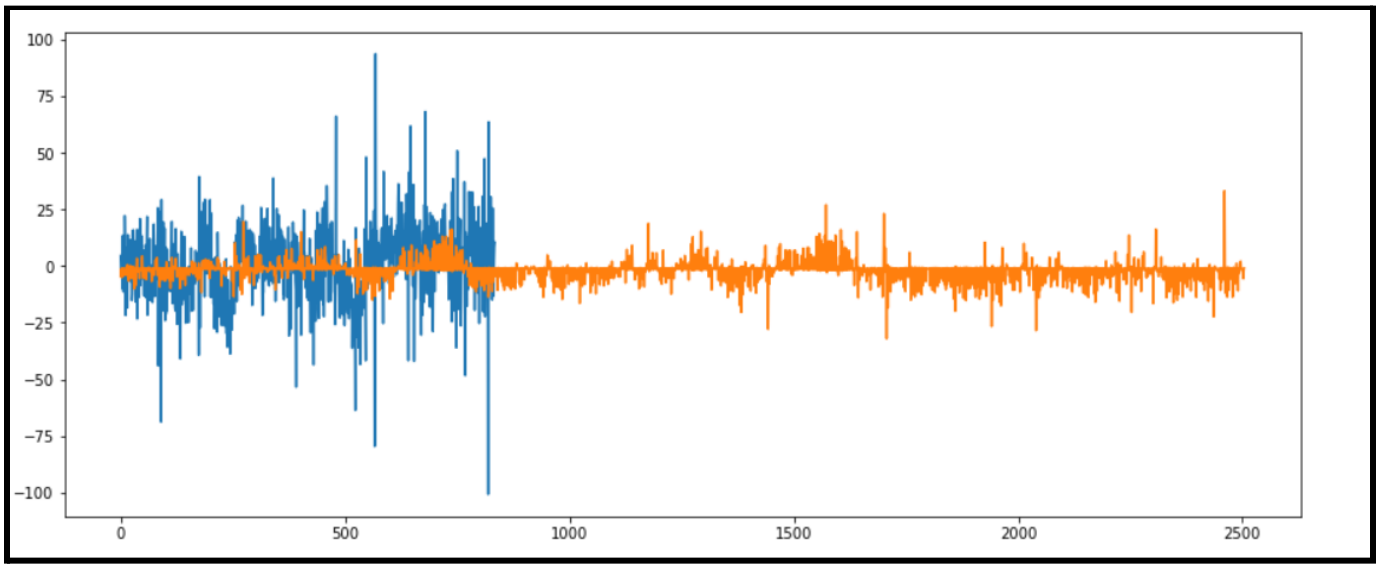


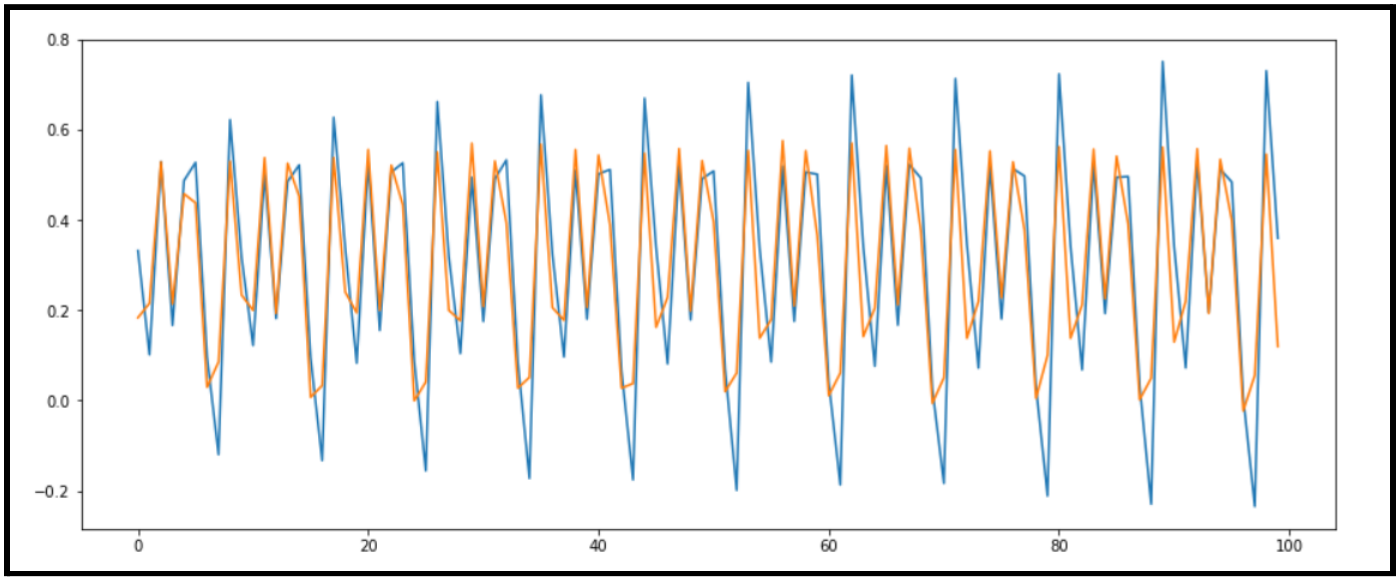
Fig.9 LSTM CELL

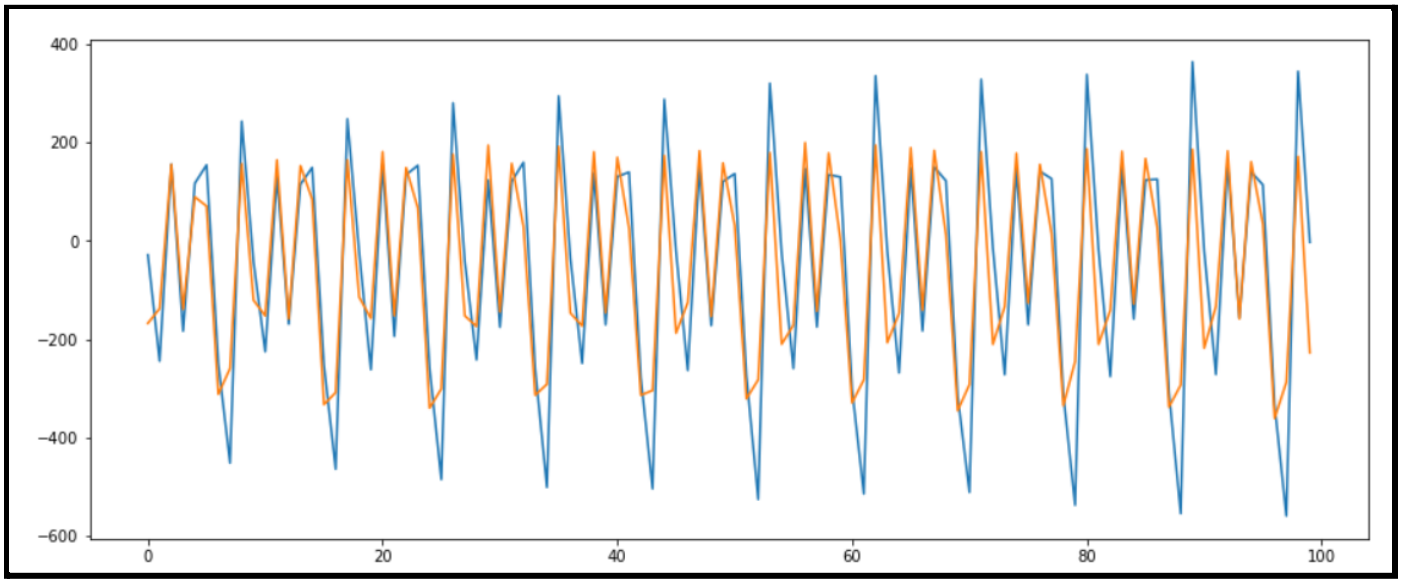
Developing the model: These are the graph of predicted values after training(y_val and y_hat)





Model Training: comparing predicted values and actual values in normalized form





VI. COMPARING WITH OTHER MODELS

For this problem, we also used the Recurrent Neural Network (RNN) and Auto-Regressive Integrated Moving Average (ARIMA) models apart from Long short-term memory (LSTM). Adding the code and results for both of them will make the size of this report unwieldy. Hence we decided against it. However, the code for the other models can be found at the following GitHub link.

https://github.com/aks05/Load_forecasting-using-Ann-Models

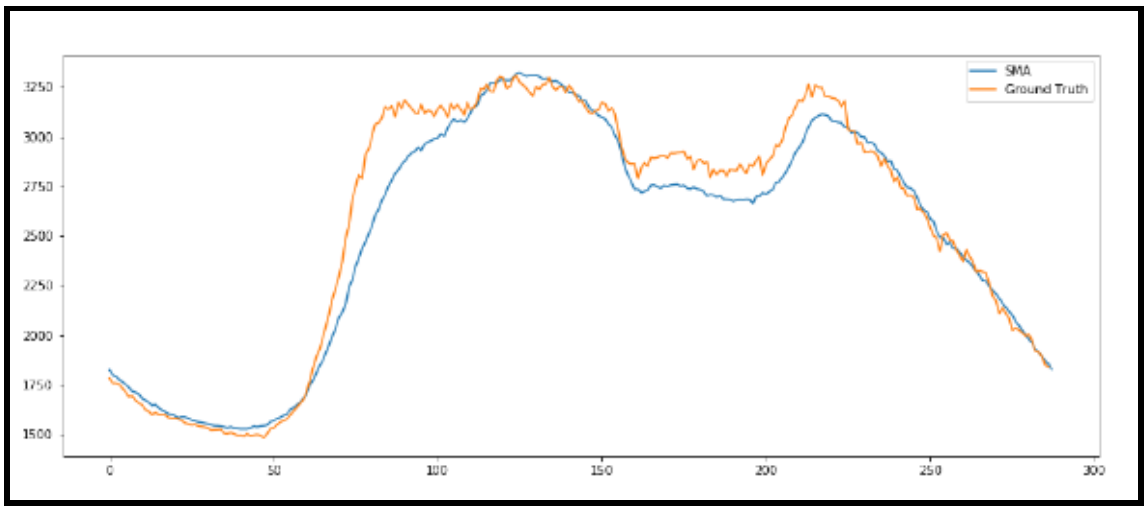


Fig.10 SMA (Simple Moving Average)

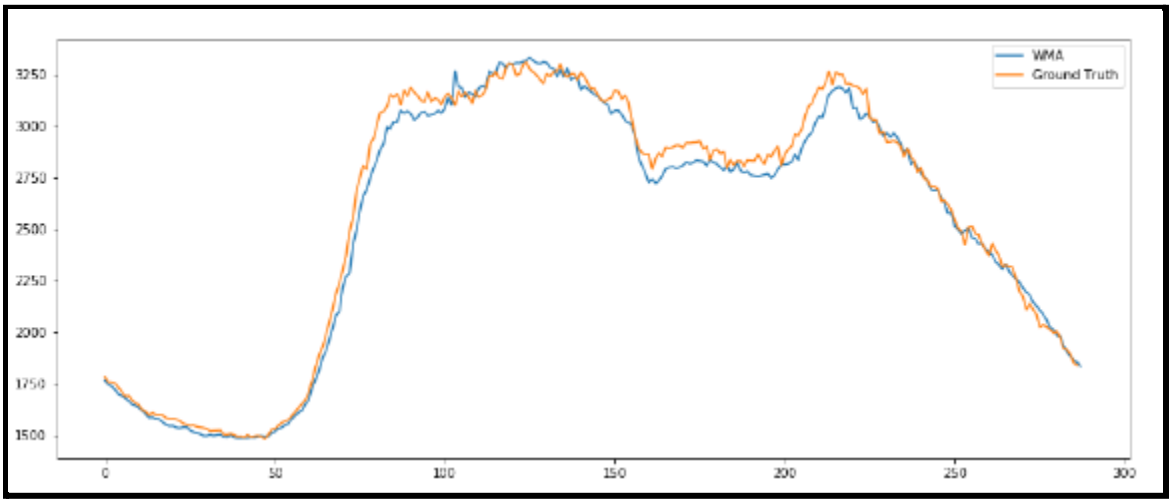


Fig.11 WMA (Weighted Moving Average)

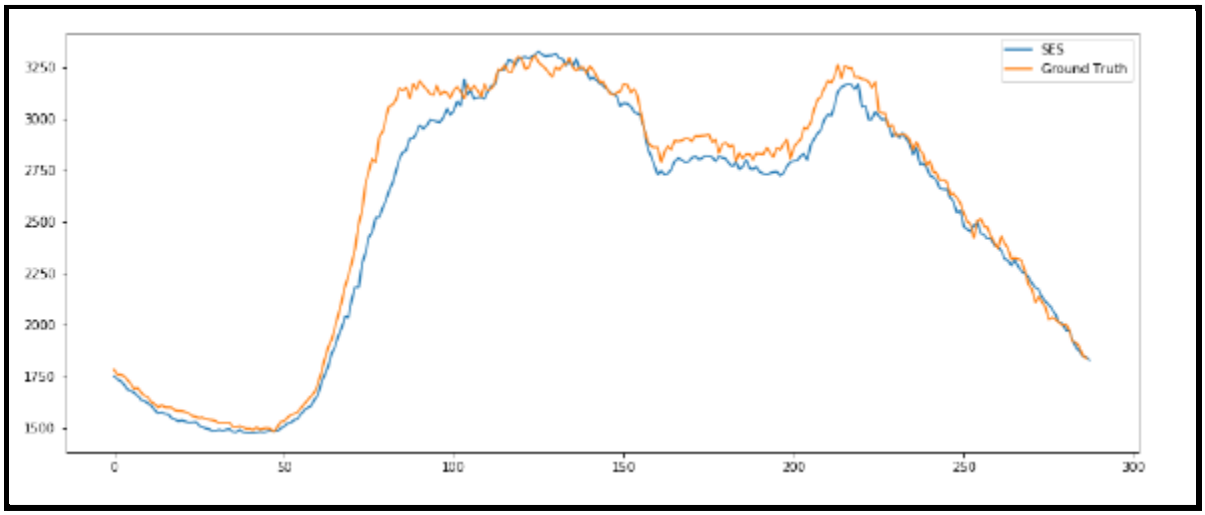


Fig.12 SES (Simple exponential smoothing)

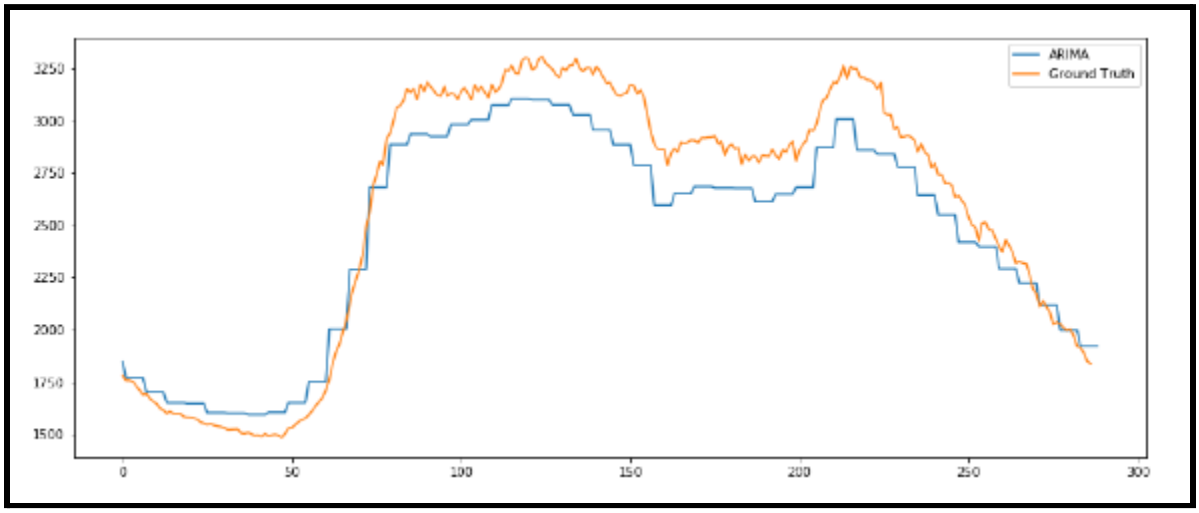


Fig.13 ARIMA (Autoregressive Integrated Moving Average)

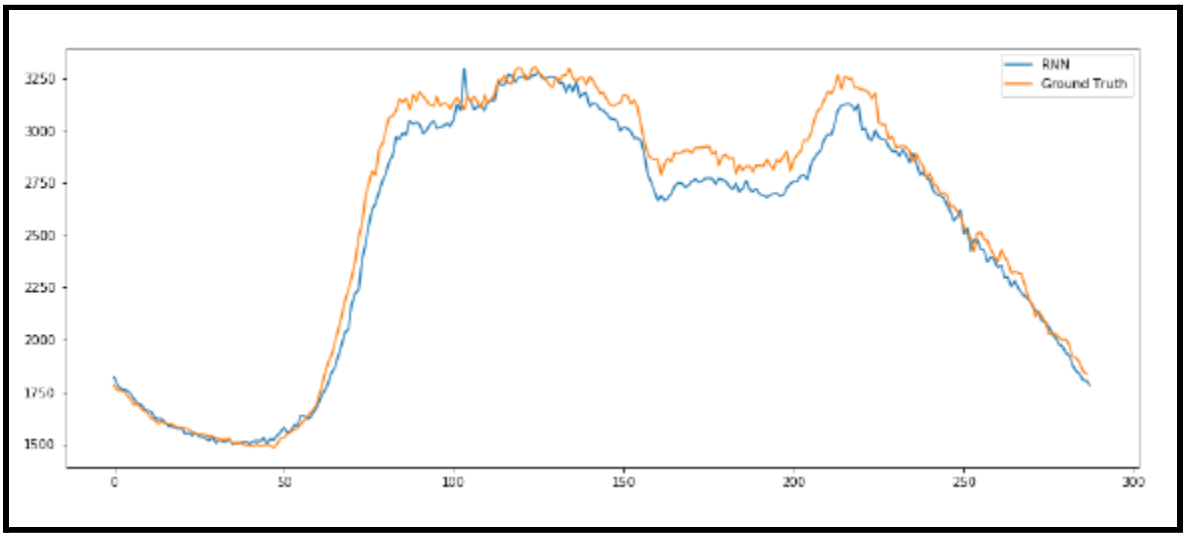


Fig.14 RNN (Recurrent Neural Network)

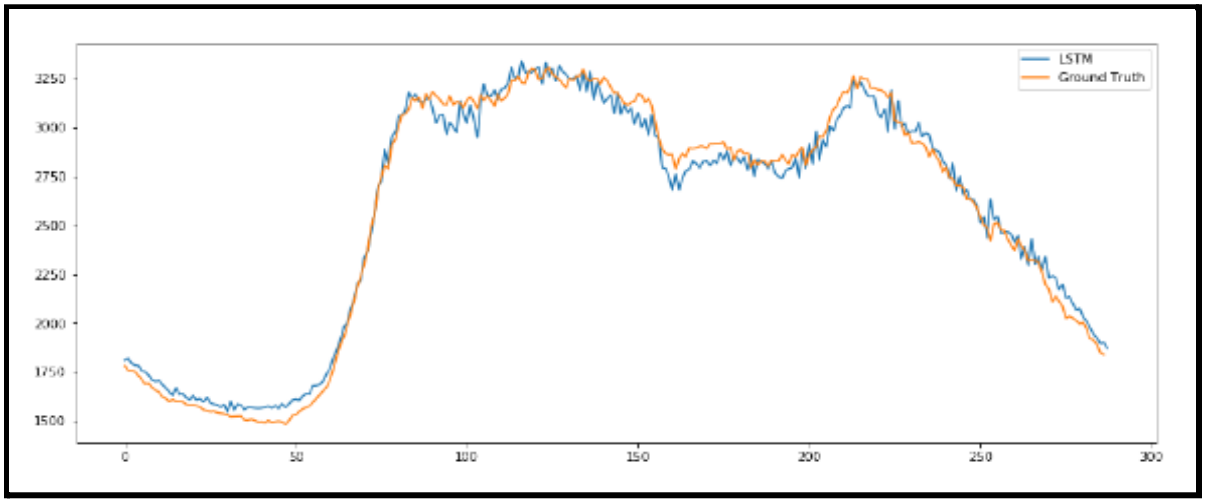


Fig.15 LSTM (Long-Short term Memory

Table.1 Comparison of Outputs using different models

S. No.	Method	RMSE	MAPE
1	SMA	126.17	3.16
2	WMA	62.86	1.85
3	SES	100.01	2.73
4	ARIMA	178.66	5.82
5	RNN	96.86	2.8
6	LSTM	73.56	2.52
7	GRU	88.35	2.48

VII. INFERENCE

This particular problem relating to Load Forecasting comes under the category of "Time Series Forecasting" problems, and LSTM and Arima models are more suitable for it. So we implemented them on a month-long data from the Delhi SLDC website and compared their accuracy using metrics such as mean square error and mean absolute percentage error.

Another point to be noted is that the LSTM model consisted of 10 neurons. In reality, such forecasting models will need upwards of 100 neurons to improve the accuracy enough to be used in field conditions. Increased accuracy is a desirable trait but comes with the catch of an increase in training time.

For our prototype-like model of 10 neurons, we trained it for 15-20 mins. Whereas, to achieve an accuracy that conforms to the safety standards of the Energy Department, we will have to use a model with more than 100 neurons trained for over 2 hours.

VIII. RESULT

A graph of the forecasted load was plotted against the time (in hours), and comparison was made against the Actual Load (test data load). A part of this graph is shown in Fig 8 (shown below).

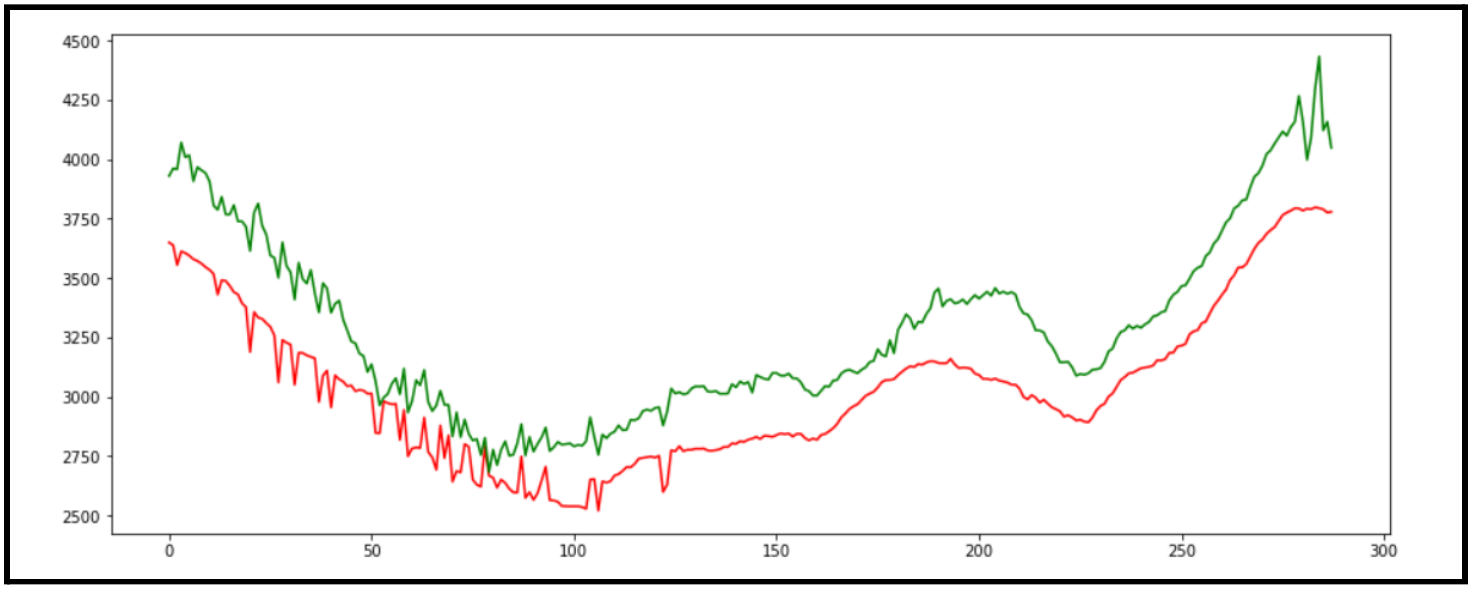


Fig.16 Output Plots with Forecasted and Actual Load Curve

In the above-resulting graph obtained using an LSTM model, the '**green**' plot represents the forecasted load curve, and the '**red**' one is the actual load curve for May 15 2021.

The graph shows a little deviation of the forecasted plot from the Test data load. The MAPE (mean absolute percentage error) came out to be 2.52 % which is bearable and RMSE (root mean square error) of 73.56.

The error values represent days where the actual load profile experienced planned or unplanned outages or other sudden load change. This is a drawback to using an ANN to forecast the load.

The ANN cannot predict these events, so a one or two hours' outage on a major distribution feeder will have a significant impact on the resulting error. If these events did not occur, then the error would have been much less. However, the high error values occurred during low load levels. This illustrates the chaotic load profiles experienced during the winter season. The winter load profiles are not as smooth as the profiles during the other seasons, so the ANN has a harder time forecasting the changes in the shape. The number of hidden layer neurons in each ANN was set at 30. This value was selected by trial and error to determine the minimum number of hidden-layer neurons that would produce the lowest forecast error.

The values selected for the minimum inputs and the number of hidden-layer neurons directly affected the python program's ANN training runtime. This added complexity would increase the program runtime and would also present the ANN with so much input data that the resulting forecast error would increase. Increasing the training weeks also required the ANN to process a larger amount of data, which increased the program runtime. If the input data set was too large, the ANN could not generalize its output, and the resulting forecast error would increase. Increasing the number of hidden-layer neurons increased the program runtime because the weights and biases of each neuron have to be calculated and optimized during network training. An overly complex network could not generalize on the out-of-sample data set and would "overfit" the in-sample data.

IX. TECH STACK USED:

Programming Language:
Python

Tools:
Jupyter IDE, Visual Studio Code, pip, anaconda

Libraries:
Matplotlib, TensorFlow, sklearn, pandas, keras, statsmodel

X. CONCLUSION

It is a goal for every power system manager to have their power system operate efficiently, securely, and economically. To meet this goal, the behaviour of their power system must be understood. Analysis of the system's normal operating bounds, response to customer demands, and reaction to weather events will provide insight on system loading. Short-term electric load forecasting can provide that insight for the following day to assist in making power system operational decisions. The demand charges and associated system loading can be reduced by energy demand management techniques such as conservation, on-site generation, or implementing demand-response agreements with the serving utility. Energy demand management is a process where the energy use for a facility is planned and coordinated with the system's various load centres.

An artificial neural network (ANN) is a mathematical model that mimics the decision-making processes of the human brain. The fundamental component of the ANN is the neuron. Neurons are programmed to behave similarly to the neurons in the brain by receiving inputs, processing those inputs, and producing an output. The neurons are connected together to form a network that can be used to solve nonlinear problems. It should not be

assumed that an ANN trained with inputs and targets for one power system will produce low error forecasts when used on a different system. The structure and size of the ANN should only be as complex as required to produce an acceptable forecast on out-of-sample data. Overly complex ANNs can overfit their training data producing a very low error on in-sample input data, but a high error on out-of-sample input data. Overfitting reduces the ANN's ability to generalize on data it has not been trained on. In addition, a complex ANN will have a high training runtime as there are more weights and biases to be estimated during the learning process.

One of the difficulties in applying ANNs to load forecasting occurs when there is an outage on the system or a significant event that causes the load to change such as the start-up or shutdown of a high demand load. The ANN cannot be trained to unplanned outages because there is no way to predict when these will occur. There is also difficulty in training the ANN to planned outages because these does not happen often. Therefore the ANN will not be exposed to many of them during its training phase.

Unplanned outages will still result in higher error forecasts, but planned outages could be used as input to the ANN. If an outage is planned, the estimated affected load could be supplied as a dedicated input to the ANN. It would be better for the outage value to be supplied to the ANN as an input than just subtracting the affected outage load from the ANN forecast. When a significant load change event occurs on a power system, this has the effect of changing the load pattern for the rest of the day. This effect is called load inertia. For example, if a large load is switched into a power system, the load profile will have a step change at the switching time. Throughout the rest of the day, the load profile will have higher values for longer time spans. In addition to the new large load, the system will experience additional losses, causing equipment to run hotter, and require additional cooling. The inertia of this load change will diminish as the system load decreases for the day. However, if the load change event was large enough, the load inertia might carry over to the following day which would affect that day's load forecast.

The same would be true if a large load was switched off of the power system. The remaining load profile for the day would be less than normal, but the difference in load levels would not be the exact value of the load that was removed. If a power system experiences these kinds of large load change events on a regular period.

XI. ACKNOWLEDGEMENTS

Working on this project of "LOAD FORECASTING USING ANN" was a source of immense knowledge to us. We are grateful because we managed to complete our ANN assignment within the time given by our Professor, Dr S. Kayalvizhi.

This project could not have been completed without the effort and co-operation from our group members, Ayush, Jagdish, Mandar, Pathikrit and Tarun. We also sincerely thank our professor of ANN Kayalvizhi ma'am for the guidance and encouragement in finishing this project and teaching us in this course.

XII. REFERENCES

1. Y. Al-Rashid and L.D. Paarmann, "Short-term electric load forecasting using neural network models," Circuits and Syst., Ames, IA, 1996
2. Moghram and S. Rahman, "Analysis and evaluation of five short-term load forecasting techniques," IEEE Trans. Power Syst., vol. 4
3. G. Gross and F.D. Galiana, "Short-Term Load Forecasting," Proc. IEEE, vol. 75
4. T. Hong, M. Gui, M. Baran, and H.L. Willis, "Modeling and forecasting hourly electric load by multiple linear regression with interactions," Power and Energy Soc. General Meeting, Minneapolis, MN, 2010
5. P.J. Santos, A.G. Martins, and A.J. Pires, "Short-term load forecasting based on ANN applied to electrical distribution substations," Universities Power Engineering Conf., Bristol, UK, 2004, vol. 1
6. https://www.delhisldc.org/
7. https://en.wikipedia.org/wiki/Long_short-term_memory
8. https://machinelearningmastery.com/
9. Hagan, Demuth, Beale, 'Neural Network Design', PWS Publishing Company, 1st Edition, 2002.
10. Freeman, J.A and Skapura, D.M., 'Neural Networks - Algorithms, Applications and ProgrammingTechniques', Addison Wesley Publications, Digitized Reprint (2007), 1991.
11. Andrew Glassner, "Deep Learning: From Basics to Practice" Vol-2, The Imaginary Institute, Seattle, WA, February 20, 2018
12. Satish Kumar, 'Neural Networks–A Classroom Approach', Tata McGraw-Hill Publishing Company Limited, 2013.
13. N.P. Padhy, S.P. Simon, 'Soft Computing with MATLAB Programming', Oxford University Press, 2015.
14. Simon Haykins, 'Neural Networks: A Comprehensive Foundation', Prentice-Hall Inc., 3rd Edition, 2008.
15. Andrew Glassner, "Deep Learning: From Basics to Practice" Vol-1, The Imaginary Institute, Seattle, WA, February 20, 2018