Load Forecasting Using Machine Learning And Statistical Techniques

Eshaan Agarwal, *Student,* Aadish Jain, *Student,* Tanvi, *Student*, under Dr. Shobhita Meher, *Assistant Professor,*
Department of Electrical Engineering, Indian Institute of Technology (BHU), Varanasi.

***Abstract*—In the field of electricity, predicting demand is a critical component of planning for routine operations and infrastructure development. The liberalization of the energy markets has led to an increase in the complexity of demand patterns. As a result, it's challenging to find an appropriate forecasting model for a specific electricity network. While various forecasting techniques have been developed, no single method can universally apply to all demand patterns. This review paper aims to analyze the intricacies of various state-of-the-art existing solutions, assess their advantages and limitations, and identify the opportunities and challenges forecasting tools present. With the geolocalized dataset from SLDC Delhi, we aim to benchmark these solutions, particularly on the Indian metropolitan scenario and the recent load demand surges, to gain new insights into the load forecasting task.**

***Index Terms*— load forecasting, time series forecasting**

I. INTRODUCTION

Electricity is a unique product that cannot be stored and must be generated as soon as it is demanded. Commercial electric power companies aim to provide safe and stable electricity to end-users. Electric Load Forecasting is critical for planning and operating the electricity industry and power systems. Precise predictions of electricity consumption result in reduced expenses for maintenance and operations, enhanced dependability of power supply and delivery, and well-informed choices for forthcoming advancements.

Forecasts are classified based on the planning horizon duration, with short-term covering up to one day/week, medium-term covering one day/week to one year, and long-term covering more than one year. Long-term forecasts are used to plan the development of the power supply, medium-term forecasts are used to plan the acquisition of fuel, and short-term forecasts are used to schedule the production and transmission of energy.

Several factors, including time, social, economic, and environmental factors, affect the electricity demand pattern, leading to various complex variations. Social and environmental factors are significant sources of randomness (noise) in the load pattern. Due to the diversity and complexity of demand patterns, the Electric Load Forecasting methods have become increasingly complex. The literature is enriched with various Load Forecasting methods, including exponential smoothing, ARMA, BoxJenkins ARIMA, neural networks, and fuzzy logic.

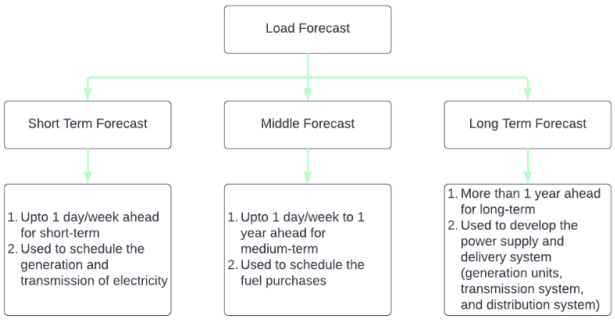


Fig.1. Flow chart of types of Load Forecasting.

Electricity is a unique product that cannot be stored and must be generated on demand. The goal of electric power providers is to supply end users with steady, safe electricity. For this reason, precise electric load forecasting is essential to the organization and management of the electrical sector. For hourly, daily, weekly, monthly, and annual intervals, load forecasting is carried out. The forecasting is classified as short-term, medium-term, and long-term, based on the duration of the planning horizon.

The complexity of demand patterns depends on various factors such as time, social, economic, and environmental factors, which lead to the development of complicated forecasting methods. ARIMA models have been successful in Electric Load Forecasting. But no single approach can be used universally to work well in every situation, particularly when a variety of elements are taken into account. Therefore, each electric power plant must follow its forecasting method, adopting general methods with modifications that suit the case. This paper presents a practical forecasting methodology that can integrate different forecasting models for short, medium, and long terms. With the realtime-data of SLDC Delhi and weather details. We aim to create a benchmarking platform for analyzing and developing algorithms in the task of load forecasting.

The dataset used in the proposed methodologies is presented in Section 2, and the methodology is described in Section 3. Section 4 applies the methodology to a typical power load pattern. The source code for this project is open source and available on GitHub.

[1]https://github.com/eshaanagarwal/Real-Time-Load-Forecasting

II. Dataset for Benchmarking

A. SLDC Dataset

This research project aims to develop a real-time prediction benchmark using the demand of the Delhi Electricity Board time series data. The dataset used for this project is the state's load values (in MW), which is collected in 5-minute time steps from October 2022 to February 2023 using a data scraping script. This provides 288 values for each day, resulting in approximately 43,200 data points. To enhance the model's accuracy, humidity and temperature data for New Delhi were also scraped from the Wunderground weather site in 5-minute intervals during the corresponding period. The features used in the model include the time of day, day of the week, temperature, and humidity at the corresponding time. The model aims to provide accurate predictions that will aid in efficiently managing electricity demand, thereby reducing costs and improving reliability.

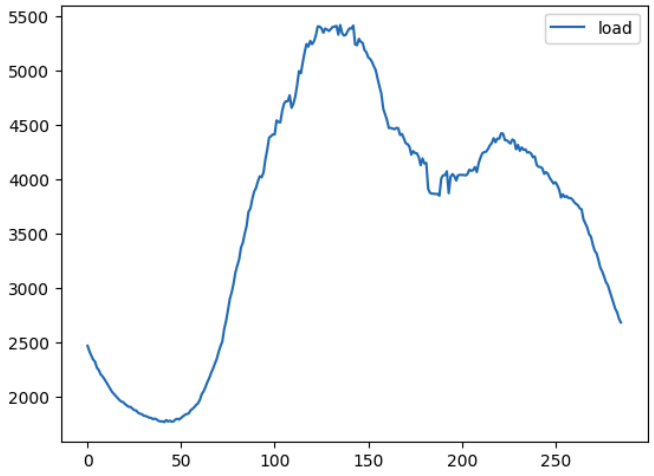


Fig.2. Load curve for Delhi from 5th January 2023.

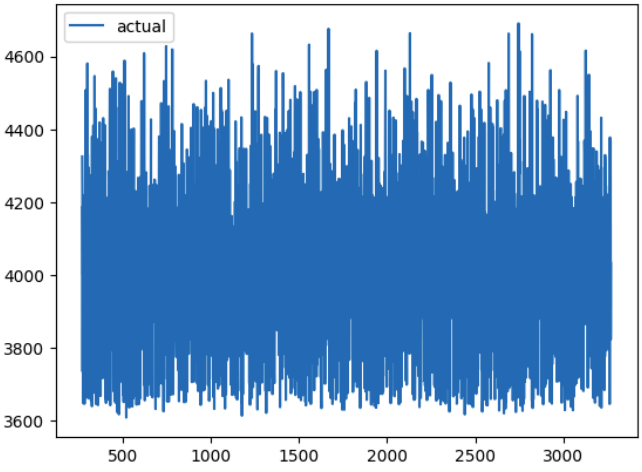


Fig.3. Weekly Load Curve for Delhi

The load data, obtained from the SLDC website of Delhi[2] using an automated script, is plotted in Fig(1) and Fig(2) to illustrate its daily and monthly trends, respectively. The data is updated on the website every 5 minutes, resulting in a corresponding time step for the extracted data. We collected data for the previous year up to the present, although some missing values may exist due to website maintenance or other factors.

B. Data Preprocessing

Ensuring the completeness and uniformity of data is essential for accurate analysis and prediction. However, data collected from online sources may have missing values due to website downtime or maintenance. In our case, the data extracted from the SLDC website of Delhi might have some missing values due to such reasons. Standard techniques like forward fill and backfill were used to impute missing values in the dataset to overcome this issue. These techniques helped make the data more uniform and complete, ensuring reliable analysis and prediction.

C. Data Analysis (Trends and Seasonality)

Time series data analysis methods such as additive and multiplicative decomposition are widely used to find trends, seasonality, and other underlying patterns. The process of additive decomposition entails breaking down the time series data into its fundamental elements, namely trend, seasonality, and residual. While the seasonality component displays the data's periodic patterns, the trend component depicts the data's overall direction across time. The residual component is the leftover data part not explained by the trend and seasonality components.

On the other hand, multiplicative decomposition involves breaking down the time series data into its components: the trend, seasonality, and residual components, but with a multiplication approach. This method is used when the magnitude of the seasonality component varies with the trend component.

These techniques have been widely applied in a variety of domains, such as finance, economics, weather forecasting, and energy demand analysis, to detect patterns and seasonality in time series data. By applying these techniques to time series data, analysts can gain insights into trends, seasonality, and other underlying patterns that can inform decision-making, such as predicting future demand or identifying potential areas for improvement.

III. Methods

In this paper, various techniques ranging from moving averages to unconventional algorithms like FFNNs have been applied to the dataset described earlier. A comprehensive discussion of the functioning of each of these methods follows below.

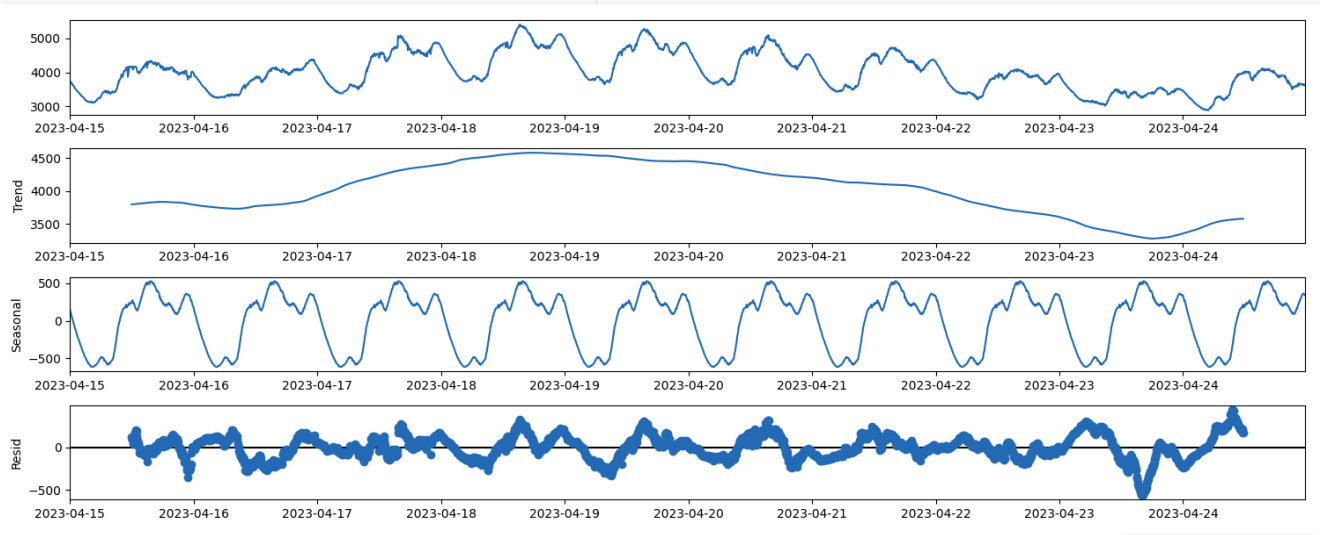


Fig.4. Trends and Seasonality from 15th - 24th April 2023

A. Simple Moving Average (SMA)

By minimizing random fluctuations, a moving average can be a useful tool in predicting when the demand for a product is steady and does not exhibit seasonal swings. Even though there are frequent disagreements on the ideal moving average length, historical data is typically utilized to forecast the future period.

Longer moving average periods smooth out random elements to a greater extent, but different period lengths may have conflicting outcomes. The more the random elements are smoothed (which may be good in many circumstances), the longer the moving average period. However, the moving average has the drawback of trailing the trend if the data show a rising or declining trend. Hence, there is a closer trend following even though a shorter time generates more oscillation. On the other hand, a longer period lags the trend but produces a smoother response. The basic moving average formula is:

$$F_t = \frac{\sum_{i=1}^{n} A_{t-i}}{n}$$

Where Ft = Forecast for the coming period, n = Forecast for the coming period, where the actual events from the previous period, two periods ago, three periods ago, and so on are represented by the numbers At−1, At−2, At−3, and so forth.

B. Weighted Moving Average (WMA)

The weighted moving average differs from the simple moving average in that it assigns specific weights to each component of the moving average database, with the condition that the sum of all weights equals 1. This allows for greater flexibility in assigning different levels of importance to different elements. The formula for the weighted moving average is determined by multiplying each data point by its corresponding weights and then summing up the results.

$$F_t = \sum_{i=1}^{n} w_i A_{t-i}$$

$$\sum_{i=1}^{n} w_i = 1$$

Where Ai is the actual occurrence for the period t-i, wi is the weight to be applied to the actual occurrence for the period t-i, and Ft is the prediction for the upcoming period, n is the total number of periods in the forecast.

As long as the total of all weights equals 1, the weighted moving average gives you the freedom to give each component of the moving average database a distinct weight. The overall sum of the weights must equal 1, even though some periods may be given zero weight and the weighting method may be in any sequence. The choice of weights can be determined through experience and trial and error. Typically, recent data points are given higher weights as they better indicate what can be expected in the future. For this reason, while calculating the moving average, the more recent past is given a higher weight and is regarded as significant indicator.

C. Simple Exponential Smoothing (SES)

One of the main disadvantages of using simple and weighted moving averages for forecasting is that they require a lot of previous data to be continuously processed. The same problem arises with regression analysis approaches as well. As new information is gathered, the oldest observation is eliminated and a fresh forecast is produced. On the other hand, current events are generally more predictive of the future than events from the distant past. If this assumption is correct, exponential smoothing could be the most practical and simple technique to apply since it takes into consideration the fact that older data becomes less significant as it gets farther away.

$$F_t = \alpha A_{t-1} + (1-\alpha) F_{t-i}$$

where Ft = exponentially smoothed forecast for period t

D. AutoRegressive Integrated Moving Average (ARIMA)

An other technique for time series forecasting is the use of ARIMA models. Their goal is to characterize the autocorrelations in the data, which is a complement to exponential smoothing models that describe the trend and seasonality in the data. Numerous disciplines, including finance, economics, and engineering, heavily rely on ARIMA.

An explanation of an ARIMA model's constituent parts is as follows:

- A model that displays a variable that is evolving and regresses on its own lagged or beginning values is known as an **autoregression (AR)** model.
- **Integrated (I)** is the difference between the data values and the values that were replaced by the prior values, or the differencing of raw observations to enable the time series to become stationary.
- When a moving average model is applied to lagged observations, the dependency between an observation and a residual error is taken into account by the **moving average (MA)**.

Every component has a standard syntax and operates as a parameter. The typical notation for ARIMA models is ARIMA with p, d, and q, where the parameters are replaced with integer values to identify the type of ARIMA model that is being utilized. One definition of the parameters is:

- **p:** the lag order, which is the number of lag observations in the model.
- **d:** the degree of differencing, or the number of times the raw observations are differed.
- **q:** the moving average window's size, sometimes referred to as the moving average's order.

We used the last month's data at a frequency of 30 minutes instead of 5 minutes due to computational overload. Best hyper-parameters were searched using the grid search method and used for model training. As the data is seasonal with seasonality and their varying nature of trends in data given the year's season, the optimized values of p, d, q were obtained accordingly.

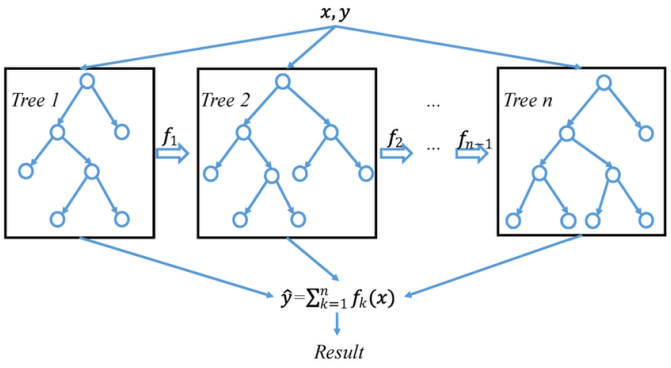


Fig.5. Flow of XGBoost Algorithm

E. XGBoost Gradient Boosting Algorithm

Gradient boosting is an ensemble learning method that combines the predictions of multiple weak learners (typically decision trees) to create a more accurate and robust model. The key idea behind gradient boosting is to sequentially fit new base learners to the residuals (or errors) of the previous learners, to reduce the residuals at each step.

One of the main features of XGBoost is its optimized algorithm for finding the best splits in the decision trees. Instead of exhaustively searching all possible splits, XGBoost uses "approximate tree learning" to efficiently find the optimal splits, greatly speeding up the training process.

Benefits of the XGBoost Algorithm are:

- **Accuracy:** Known for its improved accuracy compared to traditional methods.
- **Efficiency:** Optimized for speed and efficiency, making it well-suited for large-scale load forecasting tasks. It can handle large datasets and nonlinear relationships.
- **Flexibility:** XGBoost can handle various data types, including numerical, categorical, and ordinal data. This makes it well-suited for load forecasting tasks involving different features.
- **Regularization:** Handles missing data values effectively by assigning a default direction for them when splitting a tree node. This means the missing values are treated as either left or right children, depending on which direction gives the best split.

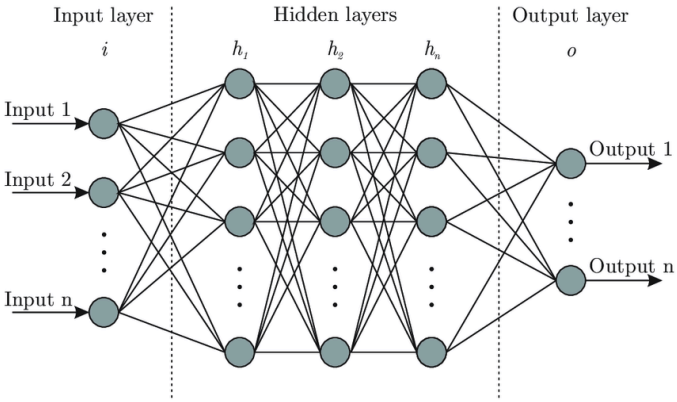


Fig.6. Flow of Artificial Feed-Forward Neural Network

E. Simple Feedforward Neural Network (FFNN)

Artificial neural networks that process data in a single direction, from input to output, without the use of feedback loops or connections, are called feedforward neural networks (FFNNs). Every layer's neurons are completely coupled to every layer's neurons. The network can recognize intricate patterns in the data by introducing non-linearity through the use of a non-linear activation function. Because they can capture intricate non-linear correlations between the load demand and several input inputs, FFNNs are especially useful in load forecasting. Depending on the complexity and amount of data given into the network, they can be used for short-, mid-, and long-term forecasting.

Data Preprocessing: For FFNN models, the training involved utilizing the most recent 120 days of data. Each training data vector encompassed the load of a particular time over the preceding 5 days, with the label representing the load on the 5th day at the corresponding time.

Model Architecture: The model comprised three layers, featuring Dense layers with ReLU activation functions, followed by a dense layer without sigmoid activation. The mean square error was employed as the loss metric, optimized using the Adam optimizer with a learning rate of 0.001. 20 epochs were used to train the model. Using the Keras library, the Python code was written.

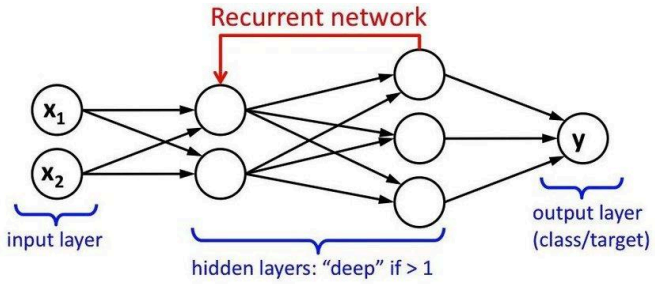


Fig.7. Flow of Recurrent Neural Network

F. Recurrent Neural Network (RNN)

RNNs are designed to work with sequential data, making them well-suited for tasks where the order of input data is crucial, such as time series forecasting. RNNs excel at capturing temporal dependencies in sequential data, allowing them to learn patterns and trends in short-term load fluctuations.

The key feature of RNNs is the presence of recurrent connections that enable the network to maintain a hidden state, preserving information from previous time steps. RNNs are effective in recognizing and predicting short-term patterns in load data. They can capture daily and weekly fluctuations, responding well to variations in load demand.

Data Preprocessing: For RNN, LSTM, GRU, and CNN models : One lag differencing was used to detrend the data, making it stationary, and the scale was changed to [-1, 1].
This helps in faster convergence and prevents features with relatively large magnitudes, like previous year load demand to carry a larger weight during training.
The last 100 days worth of data were used to train the model. Each training data vector contained the load for the previous five days at a given time, and the label represented the load for the sixth day at the same time.

Model Architecture: The model was composed of two layers: a dense layer with no activation and a single layer of simple RNN cells with an activation function of the hyperbolic tangent function (tanh). Mean square error optimized with Adam optimizer was the metric utilized for loss with a learning rate of 0.001. The model was trained for 30 epochs. The code was written in Python using Keras library

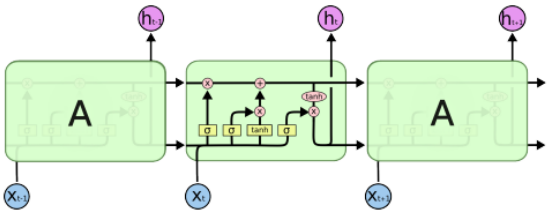


Fig.8. Flow of LSTM

G. Long Short Term Memory (LSTM)

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.Recurrent neural networks are extended by Long Short-Term Memory (LSTM) networks, which essentially increases their memory. As a result, it is ideal for learning from significant events that occur over very long lags.

RNNs can retain their inputs for a very long time thanks to LSTMs. This is due to the fact that LSTMs store their data in memory, which functions similarly to a computer's memory in that it allows for the reading, writing, and deletion of data. This memory can be thought of as a gated cell. A gated cell is one that makes decisions about what information to store or remove based on its perceived value, such as whether to open the gates or not. Weights are used to assign importance, and the algorithm also learns these. This essentially indicates that it gradually discovers what information is significant and what is not.

Three gates are present in an LSTM: input, forget, and output. These gates control whether to allow data to enter the system (input gate), discard data that isn't needed (forget gate), or allow data to affect the output at the current time step (output gate).

Data Preprocessing: The RNN model and LSTM both require a similar kind of data preparation.

Model Architecture: The model was composed of two layers: a dense layer with no activation and a layer with two LSTM cells with an activation function of hyperbolic tangent function (tanh). Mean square error optimized with Adam optimizer at 0.001 learning rate was the loss metric employed. Fifteen epochs were used to train the model. Using the Keras library, the Python code was written.

H. Gated Recurrent Unit (GRU)

The Gated Recurrent Unit, sometimes known as the GRU, is a somewhat severer variant of the LSTM. It creates a single update gate by merging the input and forget gates. In addition, it modifies a few more things and combines the concealed and cell states. The resulting model has been gaining popularity than conventional LSTM models due to its simplicity.

Data Preprocessing: The RNN model and LSTM both use a similar kind of data pre-processing.

Model Architecture: The model was composed of two layers: one GRU cell with activation function as hyperbolic tangent function (tanh) proceeded by dropout layer and three GRU cell by a dense layer without any activation. The mean square error was employed as the loss metric, optimized using the Adam optimizer with a learning rate of 0.001. 30 epochs were used to train the model. Using the Keras library, the Python code was written.

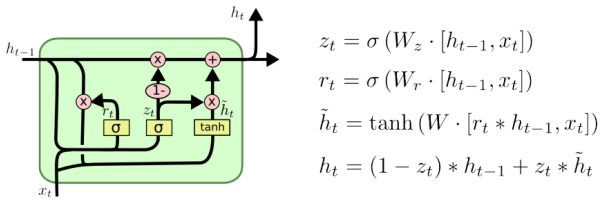


Fig.9. Flow of Gated Recurrent Unit

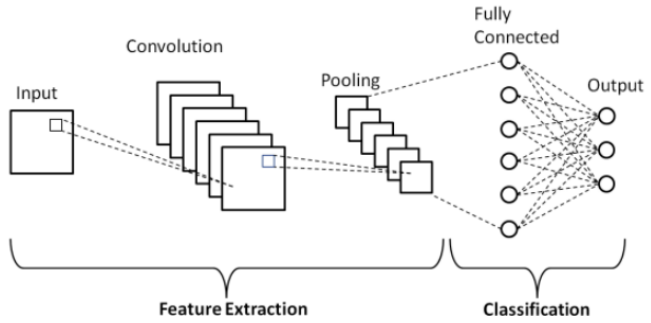


Fig.10. Flow of Convolutional Neural Network

I. Convolutional Neural Network(CNN)

CNNs adapt to temporal sequences, capturing short-term load patterns through local feature extraction. Learns hierarchical features at different scales, enabling the identification of short-term dependencies in the load data.

The convolutional layer captures local temporal patterns, while the pooling layer efficiently retains salient features, allowing a CNN to discern crucial short-term dependencies in load data. Adaptable to multivariate STLF, accommodating multiple input features for a comprehensive understanding of short-term load patterns.

Merit :

- CNNs identify and extract important local features in the data, enhancing their ability to discern critical patterns.
- The architecture of CNNs is well-suited for capturing and modeling short-term dependencies, crucial in short-term load forecasting.
- CNNs are tailored to adapt and utilize temporal convolutions, enabling them to effectively process and learn temporal patterns in sequential data.
- CNNs can seamlessly handle multivariate input data.

Demerits:

- CNNs may have constraints in modeling long-range dependencies compared to dedicated sequential models like RNNs.
- The performance of CNNs in STLF can be influenced by the sensitivity of their outcomes to the choices made in hyperparameter tuning.
- The architecture of CNNs, especially with multiple layers, can introduce computational complexity and resource-intensive training.
- CNNs typically necessitate fixed-size inputs, potentially requiring preprocessing steps for handling variable-length sequences in STLF.

Data Preprocessing: The RNN model and LSTM both use a similar kind of data pre-processing.

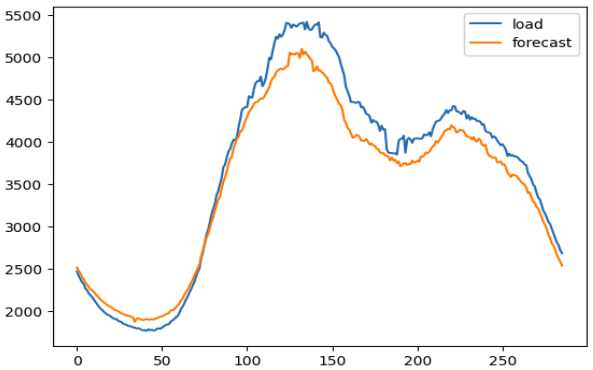
Model Architecture: There are two sections to the model:

Feature Extractor Layers: One 1-D Convolution layer with 64 filters and 2 strides followed by a max pooling layer of 2. This is used to extract the rquired feature map which os is flattened and passed to a fully connected network (FCN).

Fully Connected Network: Two dense layers with 256 flat units. Metric used for loss was mean square error optimized with Adam optimizer with a learning rate of 0.001. A 1000 epochs were used to train the model. Using the Keras library, the Python code was written.

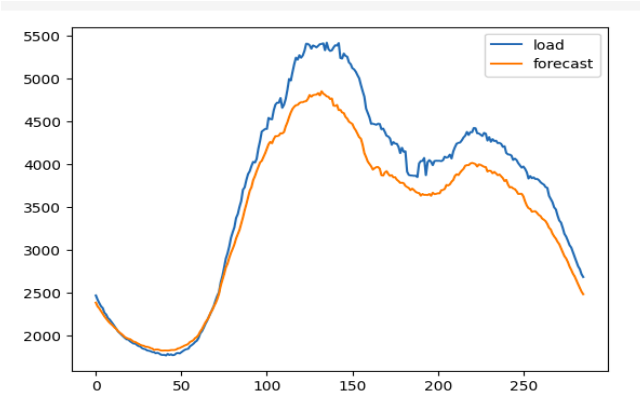
IV. Results

The outcomes of the aforementioned approaches on the load data for additional different dates are displayed in this section. The load curve predicted by the aforementioned techniques is displayed with the actual load data that was collected from the SLDC Delhi in each of the charts below. The regular day-wise changes are clearly seen in the real load curve.



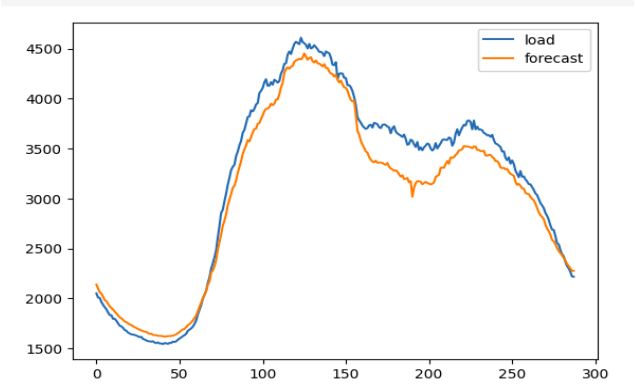
Actual Load vs forecasted load for SMA at 14-04-23

Fig.11. SMA



Actual Load vs forecasted load for WMA at 14-04-23

Fig.12. WMA



Actual Load vs forecasted load for SES at 14-04-23

Fig.13. SES

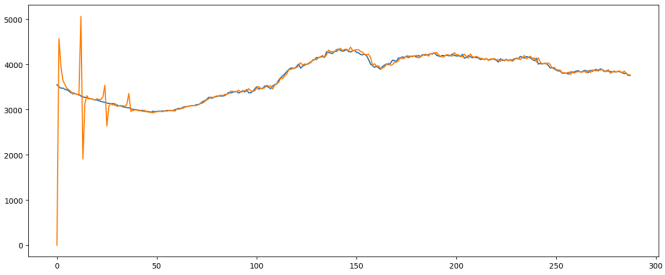
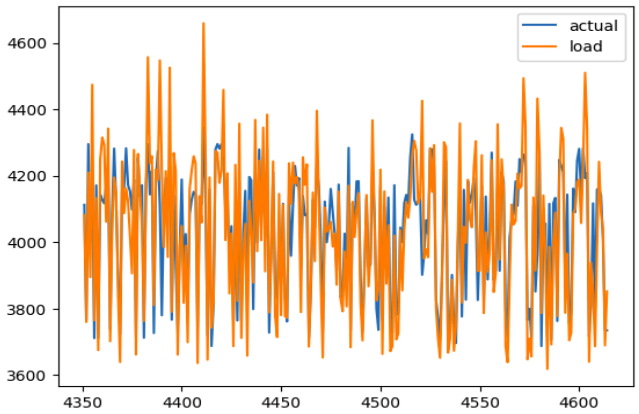
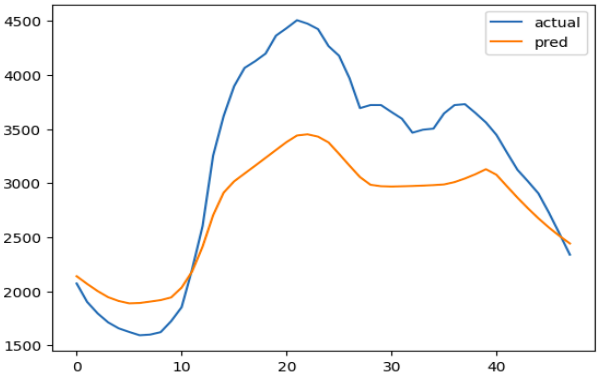


Fig.14. ARIMA

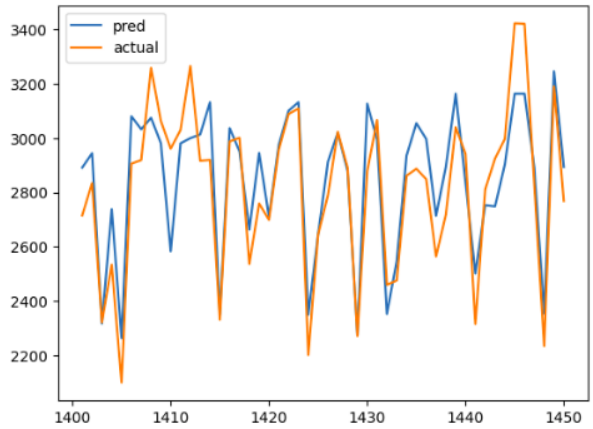


Actual Load vs forecasted load for XGBoost across a period of 50 days

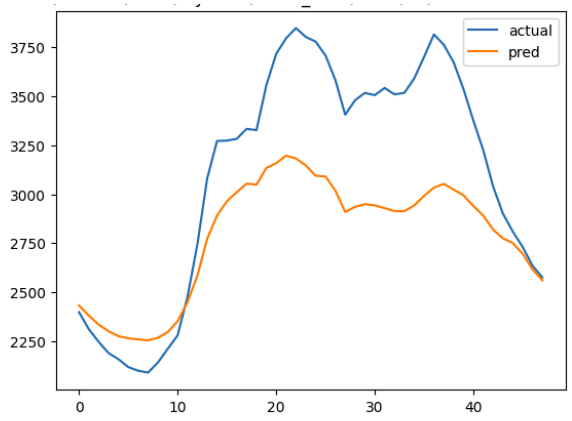


Actual Load vs forecasted load for XGBoost on 15-03-23

Fig.15. XGBoost

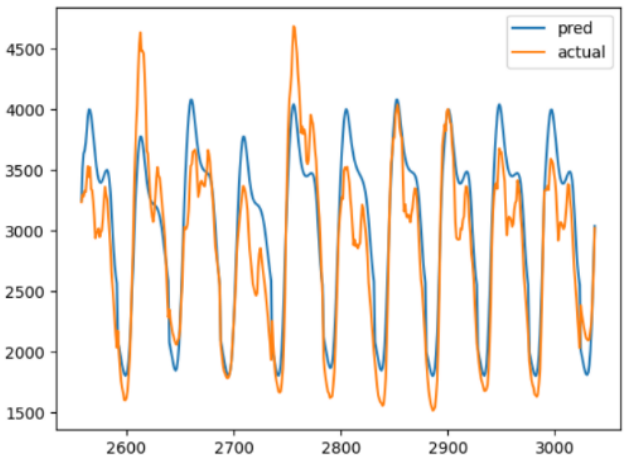


Actual Load vs forecasted load Load for FFNN across a period of 50 days

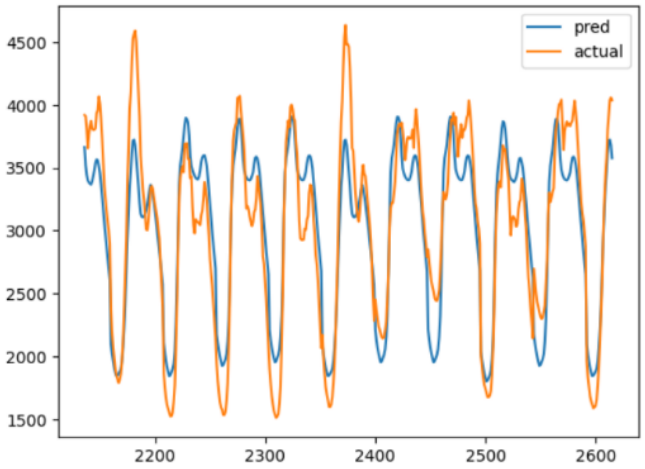


Actual Load vs forecasted load for FFNN on 13-03-23

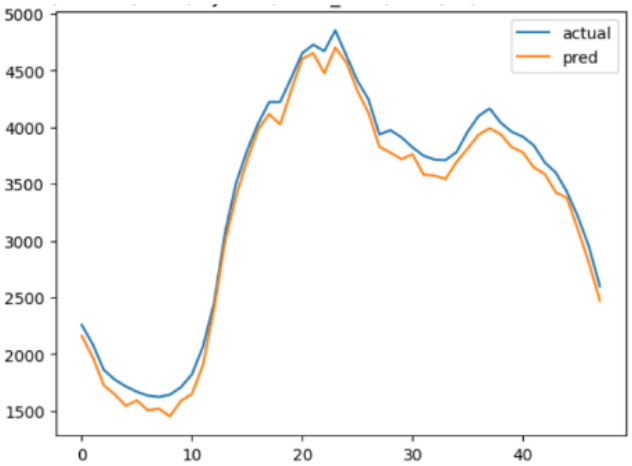
Fig.16. FFNN



Actual Load vs forecasted load for LSTM across a period of 40 days

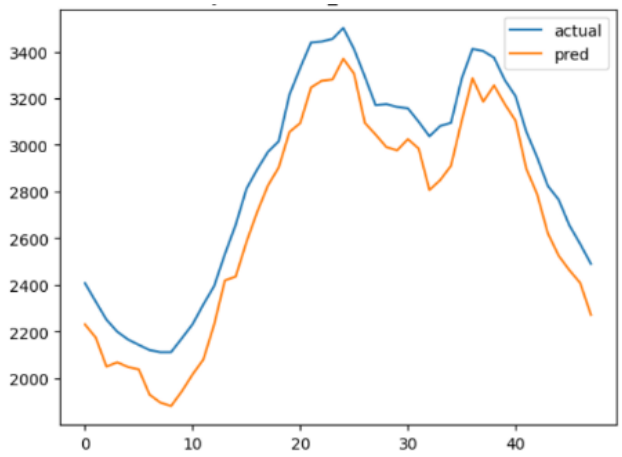


Actual Load vs forecasted load for RNN across a period of 40 days



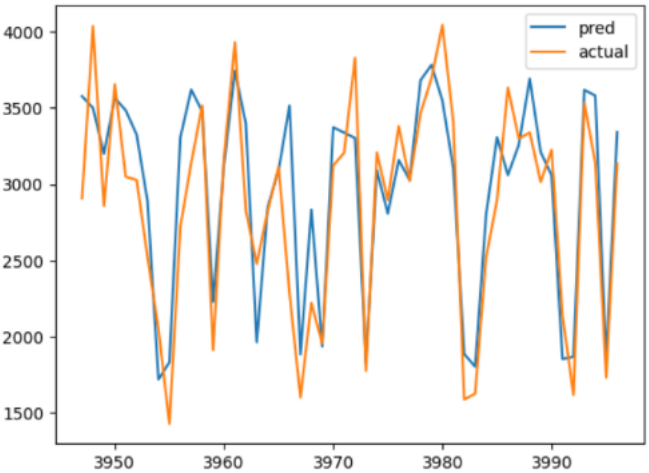
Actual Load vs forecasted load for LSTM on 27-12-22

Fig.18. LSTM

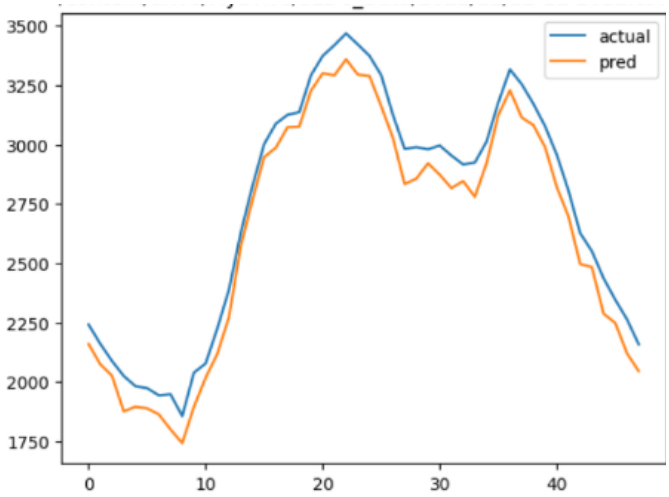


Actual Load vs forecasted load for RNN on 06-11-22

Fig.17. RNN

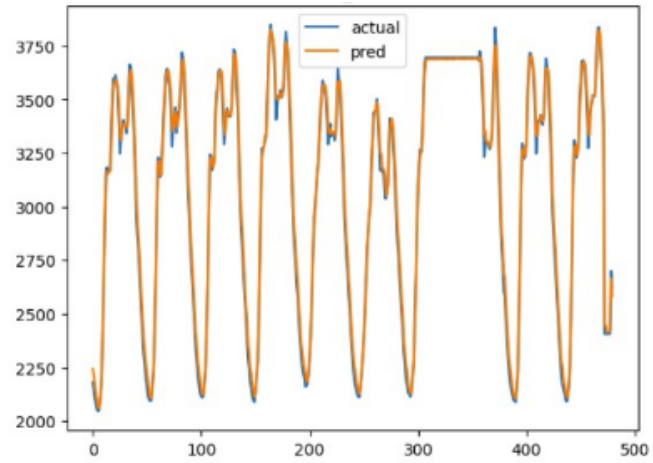


Actual Load vs forecasted load for GRU across a period of 40 days

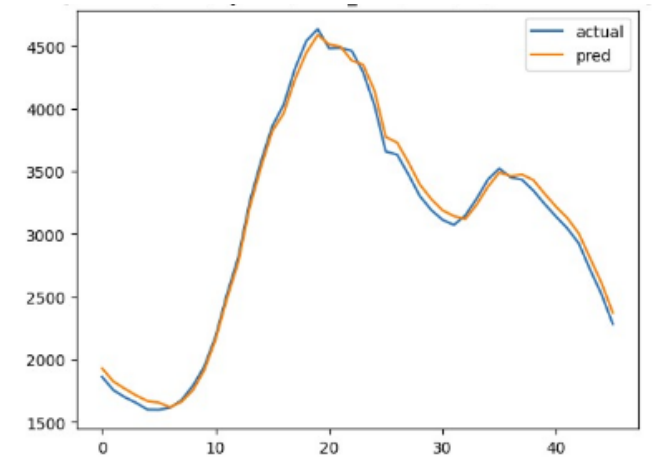


Actual Load vs forecasted load for GRU on 12-11-22

Fig.19. GRU



Actual Load vs forecasted load for CNN across a period of 50 days



Actual Load vs forecasted load for CNN on 22-01-23

Fig.20. CNN

Algorithm	MAPE
SMA	5.21
WMA	3.16
SES	4.3
SARIMA	2.71
XGboost	5.82
FFNN	4.85
RNN	3.1
GRU	2.76
LSTM	2.6
CNN	2.48

Fig.21. MAPE Table for all implemented algorithms

Algorithm	RMSE
SMA	195.66
WMA	101.61
SES	167.88
SARIMA	83.78
XGboost	206.77
FFNN	188.67
RNN	105.66
GRU	93.61
LSTM	78.88
CNN	57.78

Fig.21. RMSE Table for all implemented algorithms

IV. Conclusion and future prospects

In conclusion, Load Forecasting is a significant application of machine learning in electrical engineering, and the complexity of transmission and load networks highlights its importance. In this research, we conducted a literature survey and implemented various state-of-the-art techniques for load forecasting. Statistical techniques like SMA, WMA, SES, and ARIMA showed good performance in short-term load forecasting but failed to map the complex temporal patterns in load time series analysis.

Deep learning techniques such as FFNN and RNN provided a promising solution for understanding these complex patterns while accounting for other factors like weather and humidity in load forecasting. Advanced time series forecasting techniques like LSTM, GRU, and CNN have been implemented in load forecasting, which could provide significant results in big data volumes, designed to address the vanishing gradient problem that occurs in traditional RNNs when trying to learn long-term dependencies. All the algorithms were implemented on the real-time dataset of the Delhi Load Dispatch Centre to benchmark and understand their practical importance in load forecasting.

Additionally, more research should be conducted to explore the use of hybrid models that combine the strengths of different approaches to improve the accuracy of load forecasting. Additionally, the development of new techniques, such as deep learning and machine learning, could offer further improvements in load forecasting accuracy.

References

[1] Dataset: https://www.delhisldc.org/

[2] Arghavan Zare-Noghabi; Morteza Shabanzadeh, *Medium-Term Load Forecasting Using Support Vector Regression, Feature Selection, and Symbiotic Organism Search Optimization*, Department of Power System Operation and Planning Niroo Research Institute (NRI) Tehran, Iran

[3] Guanyu Gao, Yonggang Wen, Fellow, IEEE, Xiaohu Wu and Ran Wang, *Distributed Energy Trading and Scheduling among Microgrids via Multiagent Reinforcement Learning*

[4] Hou Shengren , Edgar Mauricio Salazar , Pedro P. Vergara1, Peter Palensky, *Performance Comparison of Deep RL Algorithms for Energy Systems Optimal Scheduling,* Delft University of Technology

[5] Slawek Smyl, Grzegorz Dudek, Pawel Pelka, *ES-dRNN: A Hybrid Exponential Smoothing and Dilated Recurrent Neural Network Model for Short-Term Load Forecasting*

[6] Yufan Zhang, Honglin Wen, Qiuwei Wu, Qian Ai, *Optimal Adaptive Prediction Intervals for Electricity Load Forecasting in Distribution Systems via Reinforcement Learning*

[7] Zachary C. Lipton, *A Critical Review of Recurrent Neural Networks for Sequence Learning,* University of California, San Diego

[8] Mohammad Safayet Hossain; Hisham Mahmood, *Short-Term Load Forecasting Using an LSTM Neural Network,* Department of Electrical and Computer Engineering, Florida Polytechnic University, Lakeland, FL, USA

[9] Gao Xiuyun; Wang Ying; Gao Yang; Sun Chengzhi; Xiang Wen; Yue Yimiao, *Short-term Load Forecasting Model of GRU Network Based on Deep Learning Framework*, Economic & Technology Research Institute, State Grid Hei longjiang Electric Power Company, Harbin, China