Load Forecasting using Deep Learning

Minor Project Report

submitted by

Ayush Raj (2102011)

Aditya Manna (2102024)

Raatan Deep (2102046)

Kumar Vaibhav (2102060)

under the supervision of

Dr. Rajib Kumar Mandal

Associate Professor & HoD

to the

Department of Electrical Engineering

National Institute of Technology, Patna

NATIONAL INSTITUTE OF TECHNOLOGY, PATNA

DEPARTMENT OF ELECTRICAL ENGINEERING

DECLARATION

We hereby declare that the work reported in the project titled **"Load Forecasting using Deep Learning"** submitted for the fulfillment of Minor Project Credits at the Department of Electrical Engineering, National Institute of Technology Patna, is record of our work carried out under the supervision of ***Dr. Rajib Kr Mandal.***

Ayush Raj(2102011)

Aditya Manna(2102024)

Ratan Deep(2102046)

Kumar Vaibhav(2102060)

Department of Electrical Engineering

National Institute of Technolgy, Patna

March 2024

ABSTRACT

Load Forecasting is an important Research Direction, which has always been the concern of academia and industry. Accurate prediction results can provide effective decisions for resource allocation of the system. However, the change of application load is very complex. How to accurately predict the change trend of load is a challenging task. Traditional prediction algorithms such as ARIMA algorithm based on statistical theory and neural network algorithm predict the target load only through the historical sequence of a single load index, ignoring the interaction between different load indexes.

Therefore, in this Project we also use the LSTM Model to predict the load, and the data at different times are given different degrees of importance. The LSTM model used in this project achieves better performance than existing prediction algorithms(ARIMA) on real load data sets.

Ayush Raj (2102011)

Aditya Manna (2102024)

Ratan Deep (2102046)

Kumar Vaibhav (2102060)

Contents

Chapter 1

Introduction

We all know with the development of technology, automatic scaling has gradually become a key technology. Automatic scaling mechanism is an ability to automatically adjust the allocation of computing resources according to the changes of application load. As the business becomes more complex, the traditional threshold based automatic scaling method is increasingly difficult to deal with the challenges brought by the dynamic changes of load. Under the above background, accurately predicting the change of load is of great significance to ensure the quality of application service and reduce resource costs.

Early load forecasting mostly used models based on statistical theory, such as autoregressive moving average model(ARMA). However, because it can only fit the linear relationship in the data series better, the fitting effect of complex time series data is not ideal. In recent years, Deep Learning algorithm has been widely used in load forecasting. We used LSTM model to predict the resources required by applications in the future.

Chapter 2

Time Series Analysis

Time series data is a sequence of observations collected or recorded over time, where each observation is associated with a specific timestamp or time interval. Examples of time series data include stock prices, weather measurements, economic indicators, and sensor readings. Time series analysis involves studying the patterns, trends, and behaviors present in the data and making predictions or forecasts about future values.

2.1 Characteristics of Time Series Data:

[2]

- **Temporal Dependency:** Observations in a time series are dependent on previous observations. The value at any given time depends on the values at earlier time points.
- **Trends:** Time series data often exhibit long-term trends, which can be increasing, decreasing, or stationary.

- **Seasonality:** Some time series display periodic patterns or seasonality, where certain patterns repeat at regular intervals (e.g., daily, weekly, or yearly).
- **Irregular Fluctuations:** Time series data may also contain random or irregular fluctuations, often referred to as noise or residuals.

2.2 Components of Time Series:

- **Level:** It refers to the underlying baseline or average value around which the data fluctuates. It represents the long-term behavior or trend in the data series,abstracting from shorter-term fluctuations such as seasonality or random noise.
- **Trend:** The long-term movement or directionality in the data, indicating whether values are generally increasing, decreasing, or stable over time.
- **Seasonality:** Repeating patterns or cycles occurring at fixed intervals within the data.
- **Cyclical Components:** Patterns that occur at irregular intervals, often associated with economic or business cycles.
- **Residuals:** Random fluctuations or noise in the data that cannot be attributed to trend, seasonality, or cyclical patterns.

2.3 LSTM Models for Time Series Analysis:

LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) designed to handle sequential data with long-term depen-

dencies. Traditional neural networks struggle with capturing temporal dependencies in time series data due to vanishing or exploding gradient problems, which LSTM addresses through its architecture. LSTM cells contain memory cells and various gates (input, output, and forget) that regulate the flow of information through the network, allowing it to selectively remember or forget past observations.

- **Memory Cells (States):** LSTMs maintain an internal state (memory cell) that can store information over long sequences.This memory cell allows them to capture long-term dependencies.'
- **Forget Gate:**Determines which information to retain or discard from the memory cell.Helps prevent vanishing gradients during training.
- **Input Gate:**Modifies the memory cell based on the current input and the previous state.Updates the memory cell with relevant information.
- **Output Gate:**Computes the output based on the modified memory cell.Produces the prediction for the next time step.

2.4 Building an LSTM Model for Time Series Prediction

- **Data Preparation:**Split the time series data into training and validation sets.Normalize the data to a common scale (e.g., [0, 1]). Create input sequences (e.g., using sliding windows).

- **Model Architecture:**Define an LSTM model using Keras or PyTorch.Specify the number of LSTM layers, hidden units, and activation functions.Add dropout layers to prevent overfitting.
- **Training:**Train the LSTM model using the training data.Optimize using backpropagation through time (BPTT).Monitor loss (e.g., Mean Squared Error) during training.
- **Prediction:** Use the trained LSTM model to predict future values.Evaluate the modelâs performance on the validation set.

Chapter 3

Stationary Time Series

3.1 Introduction to Stationary Time Series:

In time series analysis, a stationary time series is one whose statistical properties remain constant over time.
Stationarity is a fundamental concept as it simplifies the modeling process and allows for reliable predictions.

3.2 Characteristics of Stationary Time Series:

Constant Mean: The average value of the time series remains the same over time.

Constant Variance: The variability or spread of the data points around the mean remains constant over time.

Constant Autocovariance/Autocorrelation: The relationship between observations at different time points or the covariance between observations at different lags remains constant over time.

3.3 Types of Stationarity:

- **Strict Stationarity:** A time series is strictly stationary if the joint probability distribution of any set of its points does not change over time. This is a highly restrictive assumption and rarely occurs in practice.

- **Weak Stationarity (or Covariance Stationarity):** A weaker form of stationarity, where the mean, variance, and autocovariance of the time series are constant over time. This is the most common form of stationarity encountered in practice.

3.4 Tests for Stationarity:

- **Visual Inspection:** Plotting the time series data and visually examining it for trends, seasonality, or other patterns that may indicate non-stationarity.

- **Statistical Tests:** Formal statistical tests such as the Augmented Dickey-Fuller (ADF) test, Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test, and Phillips-Perron (PP) test are used to test for stationarity. These tests assess whether the time series exhibits unit root behavior (ADF test) or constant variance (KPSS test).

3.5 Importance of Stationarity:

Simplifies Modeling: Stationary time series are easier to model as their statistical properties remain constant over time, allowing for simpler and more interpretable models.

Reliable Predictions: Models built on stationary time series tend to yield more reliable predictions as they capture stable patterns in the data without being influenced by non-stationary trends or seasonality.

Assumption in Models: Many time series models, such as autoregressive (AR), moving average (MA), and autoregressive integrated moving average (ARIMA) models, assume stationarity in the data.

3.6 Techniques for Achieving Stationarity:

- **Differencing:**Taking the first difference (or higher-order differences) of the time series data can remove trends and make the series stationary. This involves subtracting each observation from its previous observation.
- **Detrending:** Removing a deterministic trend from the data to make it stationary. This can be achieved through methods such as linear regression or polynomial fitting.
- **Seasonal Adjustment:** Removing seasonal components from the data to eliminate periodic patterns and make the series stationary. Seasonal decomposition techniques such as seasonal decomposition of time series (STL) or seasonal-trend decomposition using LOESS

(STL) can be used for this purpose.

- **Transformation:**Applying mathematical transformations such as logarithmic or power transformations to stabilize the variance of the data. This can help make the series more stationary if the variance changes over time.

Chapter 4

ARIMA Model: Understanding and Application

4.1 Introduction to ARIMA Model:

ARIMA (AutoRegressive Integrated Moving Average) is a popular time series forecasting model widely used for analyzing and forecasting univariate time series data. It combines autoregressive (AR), differencing (I), and moving average (MA) components to capture different aspects of time series data, including trend, seasonality, and noise.

4.2 Components of ARIMA Model:

- **AutoRegressive (AR) Component:**Represents the relationship between an observation and a certain number of lagged observations (autoregressive terms).AR(p) model expresses the current observation as a linear combination of its p previous values.
- **Integrated (I) Component:**Refers to differencing the time series

data to make it stationary.The order of differencing (d) denotes the number of times differencing is applied to make the series stationary.

- **Moving Average (MA) Component:**Represents the relationship between an observation and a residual error from a moving average model applied to lagged observations.MA(q) model expresses the current observation as a linear combination of the residual errors from q previous observations.

4.3 ARIMA Model Notation:

ARIMA model is denoted as ARIMA(p, d, q).

- "p" denotes the order of the autoregressive component.
- "d" denotes the order of differencing.
- "q" denotes the order of the moving average component.

4.4 Model Identification:

Identification of appropriate values for p, d, and q is crucial for building an effective ARIMA model.

Methods such as autocorrelation function (ACF) and partial autocorrelation function (PACF) plots, and Akaike Information Criterion (AIC) are commonly used for model identification.

ACF and PACF plots help identify the orders of the AR and MA components, while AIC helps compare different models based on their goodness of fit and complexity.

4.5 Model Estimation:

Once the parameters (p, d, q) are identified, the ARIMA model is estimated using the least squares method or maximum likelihood estimation. Estimation involves fitting the AR, I, and MA components to the time series data to minimize the residual errors.

4.6 Model Diagnostic Checking:

After estimation, it's essential to perform diagnostic checks to ensure that the model adequately captures the underlying patterns in the data. Diagnostic checks involve examining the residuals for randomness, stationarity, and independence using techniques like residual plots, Ljung-Box test, and Q-Q plots.

4.7 Forecasting with ARIMA Model:

Once the model is validated, it can be used to generate forecasts for future time points. Forecasting involves recursively applying the ARIMA model to predict future values based on the observed historical data.[1]

4.8 Advantages of ARIMA Model:

Ability to capture both linear and non-linear patterns in time series data. Flexibility in modeling different types of time series data with varying levels of trend, seasonality, and noise. Well-established methodology with clear guidelines for model identification, estimation, and validation.

4.9 Limitations of ARIMA Model:

Assumes linear relationships between variables, which may not always hold in real-world data. Requires stationary data, and transforming non-stationary data can sometimes introduce complexity. May not perform well with irregularly spaced or missing data points.

Chapter 5

Long Short-Term Memory (LSTM) Model

5.1 Introduction to LSTM:

LSTM is a type of recurrent neural network (RNN) architecture specifically designed to address the vanishing gradient problem, which is common in traditional RNNs.
Introduced by Hochreiter and Schmidhuber in 1997, LSTM has become a cornerstone in sequence modeling and time series analysis due to its ability to capture long-term dependencies.

5.2 Architecture of LSTM:

LSTM units contain memory cells and several gates:
Forget Gate: Decides which information to discard from the cell state.
Input Gate: Determines which new information to update into the cell state.
Output Gate: Controls which parts of the cell state are output to the

next layer.

This architecture allows LSTMs to remember or forget information over long sequences, making them suitable for tasks with extended dependencies.

5.3 Key Features of LSTM:

- **Long-Term Dependency Handling:** LSTMs are capable of retaining information over many time steps, enabling them to capture long-range dependencies in sequential data.
- **Gradient Flow Preservation:**The design of LSTM cells mitigates the vanishing gradient problem, ensuring more stable and effective training.
- **Parallelization:**LSTM units can be parallelized across time steps, leading to efficient computation during both training and inference phases.

5.4 Training and Optimization of LSTM:

Data Preparation: Sequential data is preprocessed into fixed-length sequences and possibly normalized for better convergence.

Model Training: LSTMs are trained using backpropagation through time (BPTT), where gradients are computed over multiple time steps and optimized using algorithms like stochastic gradient descent (SGD), Adam, or RMSprop.

Hyperparameter Tuning: Parameters such as the number of LSTM

layers, hidden units, learning rate, and dropout rate are tuned to optimize model performance and prevent overfitting.

Regularization: Techniques like dropout and recurrent dropout are employed to prevent overfitting and improve generalization.

5.5 Evaluation and Validation of LSTM:

Evaluation Metrics: Common evaluation metrics for LSTM models include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and coefficient of determination (R-squared).

Validation Strategies: Time series data is typically split into training, validation, and test sets, with the validation set used for hyperparameter tuning and model selection, and the test set used for final evaluation.

5.6 Challenges and Considerations:

Data Quality and Preprocessing: Noisy or incomplete data can adversely affect LSTM model performance, necessitating careful preprocessing steps such as data cleaning and feature engineering.

Model Complexity: LSTMs can be computationally intensive and require large amounts of data for training, which can pose challenges in terms of computational resources and training time.

Interpretability: Despite their effectiveness, LSTM models are often considered black-box models, making it challenging to interpret their

decisions and understand the underlying mechanisms driving predictions.

[3]

Chapter 6

Gated Recurrent Unit (GRU)

6.1 Introduction

Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that was introduced by Cho et al. in 2014 as a simpler alternative to Long Short-Term Memory (LSTM) networks. Like LSTM, GRU can process sequential data such as text, speech, and time-series data.

The basic idea behind GRU is to use gating mechanisms to selectively update the hidden state of the network at each time step. The gating mechanisms are used to control the flow of information in and out of the network. The GRU has two gating mechanisms, called the reset gate and the update gate.

The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new input should be used to update the hidden state. The output of the GRU is calculated based on the updated hidden state.

The equations used to calculate the reset gate, update gate, and hidden state of a GRU are as follows:

$$Resetgate : r_t = sigmoid(W_r * [h_{t-1}, x_t]) \tag{6.1}$$

$$Updategate : z_t = sigmoid(W_z * [h_{t-1}, x_t]) \tag{6.2}$$

$$Candidatehiddenstate : h_t = tanh(W_h * [r_t * h_{t-1}, x_t]) \tag{6.3}$$

$$Hiddenstate : h_t = (1 - z_t) * h_{t-1} + z_t * h_t \tag{6.4}$$

where $W_r, W_z, andW_h$ are learnable weight matrices

x_t is the input at time step t,

h_{t-1} is the previous hidden state, and h_t is the current hidden state.

In summary, GRU networks are a type of RNN that use gating mechanisms to selectively update the hidden state at each time step, allowing them to effectively model sequential data.

6.2 Working of a Gated Recurrent Unit:

- Take input the current input and the previous hidden state as vectors.
- Calculate the values of the three different gates by following the steps given below:-

1. For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (Hadamard Product) between the concerned vector and the respective weights for each gate.
2. Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate

- The process of calculating the Current Memory Gate is a little different. First, the Hadamard product of the Reset Gate and the previously hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.

  $\overline{h}_t = tanh(W \odot x_t + W \odot (r_t \odot h_{t-1}))$

- To calculate the current hidden state, first, a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First, calculate the Hadamard Product of the update gate and the previously hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the Hadamard Product of the newly generated vector with the current memory gate. Finally, add the two vectors to get the currently hidden state vector.

  $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t$

The above-stated working is stated as below:-

Note that the blue circles denote element-wise multiplication. The positive sign in the circle denotes vector addition while the negative sign denotes vector subtraction(vector addition with negative value). The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

Just like Recurrent Neural Networks, a GRU network also generates an output at each time step and this output is used to train the network using gradient descent.

Note that just like the workflow, the training process for a GRU network is also diagrammatically similar to that of a basic Recurrent Neural Network and differs only in the internal working of each recurrent unit.

The Back-Propagation Through Time Algorithm for a Gated Recurrent Unit Network is similar to that of a Long Short Term Memory Network and differs only in the differential chain formation.

Let $\overline{y}_t$ be the predicted output at each time step and y_t be the actual output at each time step. Then the error at each time step is given by:-

$E_t = -y_t log(\overline{y}_t)$

The total error is thus given by the summation of errors at all time steps.

$E = \sum_t E_t \Rightarrow E = \sum_t -y_t log(\overline{y}_t)$

Similarly, the value $\frac{\partial E}{\partial W}$ can be calculated as the summation of the gradients at each time step.

$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$

Using the chain rule and using the fact that $\overline{y}_t$ is a function of $h_t$and

which indeed is a function of $\overline{h}_t$, the following expression arises:-

$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \overline{y}_t} \frac{\partial \overline{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots\cdots \frac{\partial h_0}{\partial W}$

Thus the total error gradient is given by the following:-

$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial \overline{y}_t} \frac{\partial \overline{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots\cdots \frac{\partial h_0}{\partial W}$

Note that the gradient equation involves a chain of ∂h_t which looks similar to that of a basic Recurrent Neural Network but this equation works differently because of the internal workings of the derivatives of h_t .

6.3 Comparison between LSTM and GRU

GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) are both types of recurrent neural networks (RNNs) designed to address the vanishing gradient problem and improve the modeling of long-range dependencies in sequential data. Here's a comparison between GRU and LSTM:

- **Structure:**

  LSTM: LSTM networks have a more complex structure compared to GRU. They have three gates (input gate, forget gate, output gate) and a memory cell that allows them to selectively remember or forget information.

  GRU: GRU networks are simpler than LSTMs, with only two gates (reset gate, update gate) and no separate memory cell.

- **Gating Mechanisms:**

  LSTM: The three gates in LSTM control the flow of information

through the cell, allowing it to retain information over long sequences and prevent vanishing gradients.

GRU: GRU uses a reset gate to decide how much past information to forget and an update gate to determine how much new information to let in.

- **Performance:**

  LSTM: Historically, LSTMs have been the preferred choice for tasks involving long sequences or when precise control over memory is required. They tend to perform well on tasks like language modeling, machine translation, and speech recognition.

  GRU: GRUs are computationally less expensive than LSTMs due to their simpler structure. They have shown comparable performance to LSTMs on various tasks and are sometimes preferred when faster training or inference times are crucial.

- **Training Speed:**

  LSTM: LSTMs can be slower to train compared to GRUs because of their additional complexity and the need to learn more parameters.

  GRU: GRUs typically train faster than LSTMs due to their simpler architecture and fewer parameters.

- **Memory Usage:**

  LSTM: LSTMs tend to use more memory because of their additional memory cell and gating mechanisms.

  GRU: GRUs are more memory-efficient compared to LSTMs.

- **Usage:**

  LSTM: LSTMs are often used in applications where precise control over long-term dependencies is critical, such as in generating text or processing sequential data with long-range dependencies.

  GRU: GRUs are used in scenarios where a balance between model complexity and performance is desired, such as in real-time applications or when training resources are limited

In summary, LSTMs and GRUs each have their strengths and are suited to different tasks and constraints. LSTMs excel in modeling long-range dependencies but are more complex and resource-intensive, while GRUs offer a simpler alternative with faster training times and lower memory usage at the cost of some expressiveness in long-term memory retention.

Chapter 7

Conclusion

Electricity load forecasting plays a pivotal role in the efficient operation and planning of power systems. In this study, we explored the performance of two widely used time series forecasting models, ARIMA (AutoRegressive Integrated Moving Average) and LSTM (Long Short-Term Memory), for load forecasting. The analysis was conducted on historical load data to predict future electricity demand.

ARIMA Model Performance:

The ARIMA model, a traditional yet powerful method for time series forecasting, demonstrated commendable performance in predicting electricity load. Through the analysis of the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots, we selected the optimal parameters (p,d and q) for the ARIMA Model. The model was trained on historical load data, capturing the underlying trends and seasonal

patterns. The ARIMA Model provided accurate short-term load forecasts with a ***Mean Absolute Error (MAE)*** of **6.8685**, ***Mean Absolute Percentage Error*** of **12.72%** and a ***Root Mean Squared Error*** (RMSE) of **8.2004**.

LSTM Model Performance:

On the other hand, the LSTM model, a deep learning-based approach known for its ability to capture complex temporal dependencies, was also employed for load forecasting. The LSTM model was trained on sequences of historical load data, allowing it to learn intricate patterns and nonlinear relationships in the time series. This resulted in impressive forecasting accuracy for both short-term and long-term load predictions.

Comparative Analysis:

Comparing the performance of the ARIMA and LSTM models, we observed that while the ARIMA model provided reliable forecasts, the LSTM model excelled in capturing the finer nuances of the electricity load data. The LSTM model's ability to learn from past sequences and adapt to changing load patterns resulted in superior forecasting accuracy, especially in scenarios

with complex load variations and irregularities.

Recommendations and Future Directions:

Based on our findings, we recommend the use of LSTM models for electricity load forecasting, particularly in scenarios where the load patterns exhibit nonlinearities and complexities. However, the choice of model should also consider the computational resources available and the trade-off between accuracy and simplicity.

Future research directions may include:

- Exploring hybrid models that combine the strengths of ARIMA and LSTM for enhanced forecasting accuracy.
- Investigating the impact of external factors such as weather conditions, holidays, and economic indicators on load forecasting.
- Implementing real-time forecasting systems using LSTM models for dynamic load management and grid optimization.

In conclusion, the combination of traditional ARIMA models and advanced LSTM models offers valuable insights and accurate forecasts for electricity load forecasting. The findings of this study contribute to the ongoing efforts to optimize energy consumption, improve grid stability, and pave the way for a sustainable energy future.

Bibliography

[1] Meftah Elsaraiti, Gama Ali, Hmeda Musbah, Adel Merabet, and Timothy Little. Time series analysis of electricity consumption forecasting using arima model. In *2021 IEEE Green Technologies Conference (GreenTech)*, pages 259–262, 2021.

[2] Mirjana IvanoviÄ and Vladimir Kurbalija. Time series analysis and possible applications. In *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 473–479, 2016.

[3] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.