



**University of Piraeus**  
**School of Information and Communication Technologies**  
**Department of Digital Systems**

**Master's degree program**  
**M.Sc. "Information Systems and Services"**  
**Big Data and Analytics**

M.Sc. Thesis Title:  
**"Optimizing Short-Term Electricity Load Predictions: A Study of Feed-Forward and Recurrent Neural Network Models"**

by  
**Georgios Kaltsas**

Submitted  
in partial fulfilment of the requirements for the degree of Master of Science  
in Information Systems & Services  
in the specialization of Big Data and Analytics  
at the University of Piraeus

Thesis Supervisor: Filippakis Michael - Professor  
Member of the Examination Committee: Kyriazis Dimosthenis - Professor  
Member of the Examination Committee: Halkidi Maria - Professor

September 2024

## Table of Contents

Table of Figures .....	4
Table of Tables .....	5
Abstract.....	6
Acknowledgements.....	7
Section I : Theoretical Part .....	8
1.1 Historical Overview of Power Systems .....	8
1.2 Electric Power System (EPS).....	9
1.3 Electric Power Load Forecasting (EPLF) .....	10
1.4 Types of Load Forecasting.....	11
1.5 Key Determinants Influencing Electrical Load Dynamics.....	13
1.6 Fundamentals of Neural Networks: Understanding the Biological and Computational Paradigms .....	14
1.6.1 A Journey Through Time: The Evolution of Artificial Neural Networks .....	16
1.6.2 Diverse Architectures of Artificial Neural Networks .....	17
1.6.3 The Role of Activation Functions in Artificial Neural Networks.....	19
1.6.4 Understanding Recurrent Neural Networks .....	20
1.6.5 The Evolution and Mechanisms of Long Short-Term Memory (LSTM) Networks in Depth: A Comparative Analysis with Traditional Recurrent Neural Networks (RNNs).....	22
1.6.6 Unveiling the Gated Recurrent Unit (GRU): A Comparative Analysis with RNNs and LSTMs.....	23
1.6.7 Operation and Training of Artificial Neural Networks (ANNs).....	24
1.6.8 Supervised and Unsupervised Learning: The Two Pillars of Training Artificial Neural Networks.....	25
1.6.9 Key Error Metrics for Evaluating ANN Predictions .....	26
1.7 Time Series Analysis and Forecasting: Concepts and Techniques .....	28
1.7.1 Main Aspects of Time Series Forecasting .....	30
1.7.2 Time Series Models: AR, MA, ARMA, ARIMA.....	31
Section II : Implementation .....	33
2.1 Electric Load Data Extraction and Preparation .....	33
2.1.1 Weather Data Extraction and Preparation.....	35
2.1.2 Explanation for Merging Data and Importance of Temporal Features in Forecasting .	37

---

2.2 Exploratory data analysis (EDA) .....	38
2.2.1 Key Insights from Descriptive Statistics.....	39
2.2.2 Key Insights from Heatmap Visual.....	39
2.2.3 Key Insights from Scatter Plot.....	41
2.2.4 Key Insights from Time Series Data.....	45
2.3 Architecting Effective Time Series Models for Energy Forecasting .....	48
2.4 Feed Forward Model.....	50
2.4.1 Detailed Explanation of Feed-Forward Model Code and Implementation.....	51
2.4.2 Comparative Analysis of Feed-Forward Model Scenarios and Performance Evaluation .....	55
2.5 Recurrent Neural Network.....	59
2.5.1 Detailed Explanation of Recurrent Neural Networks Code and Implementation.....	60
2.5.2 Comparative Analysis of Recurrent Neural Models Scenarios and Performance Evaluation.....	68
Section III : Conclusion.....	81
3.1 Observations .....	81
3.2 Future Study.....	82
References.....	84

## Table of Figures

Figure 1: Edison's Jumbo dynamo. Courtesy: National Park Service, Edison National Historic Site.(Source)	8
Figure 2: Representation of the Electric Power System (EPS) (Source)	9
Figure 3: Architecture of an ideal smart grid (SG) with bi-directional information flow, including distributed energy sources.....	11
Figure 4: Types of load forecasting models according to the forecast horizon Identification of the scope and type of input parameters (Source).....	12
Figure 5: Example of the factors that can affect the LF model (Source)	14
Figure 6: Illustration of a neuron (Source)	15
Figure 7: Example of a Feedforward Neural Network (ANN) (Source).....	17
Figure 8: Example of a Recurrent NN (RNN) (Source).....	18
Figure 9: Example of Convolutional Neural Network (CNN) (Source).....	19
Figure 10: Activation Functions for ANNs (Source).....	20
Figure 11: Architecture of RNN (Source).....	21
Figure 12: Architecture of LSTM (Source)	22
Figure 13: Architectures comparison of RNN - LSTM – GRU (Source)	24
Figure 14: Supervised vs Unsupervised Learning (Source)	26
Figure 15: Example of Seasonality (Source)	29
Figure 16: Trend, Seasonality, Cyclic behavior, and Irregular fluctuations (Source)	30
Figure 17: Heatmap Visual	40
Figure 18: Scatter Plot Temperature 2m ATH (°C)	42
Figure 19: Scatter Plot Temperature 2m SKG (°C)	42
Figure 20: Scatter Plot Humidity 2m ATH (%).....	43
Figure 21: Scatter Plot Humidity 2m SKG (%).....	44
Figure 22: Original Energy Load Data Over Time	45
Figure 23: Energy Load with Savitzky-Golay and Polynomial Regression Trend Lines	46
Figure 24: Residuals of Energy Load After Trend Removal.....	47
Figure 25: Moving Window Function Example (Source).....	49
Figure 26: Training and Validation Loss for Each Scenario	57
Figure 27: Actual vs. Predicted Load for Training and Validation Data Across Scenarios	58
Figure 28: Training and Validation Loss for Scenario1 and LSTM model.....	68
Figure 29: Training and Validation Loss for Scenario1 and GRU model	69
Figure 30: Training and Validation Loss for Scenario1 and SimpleRNN model	69
Figure 31: Actual vs. Predicted Load for LSTM Scenario 1	70
Figure 32: Actual vs. Predicted Load for GRU Scenario 1.....	70
Figure 33: Actual vs. Predicted Load for SimpleRNN Scenario 1.....	71
Figure 34: Training and Validation Loss for Scenario 2 and LSTM model.....	73
Figure 35: Training and Validation Loss for Scenario 2 and GRU model	73
Figure 36: Training and Validation Loss for Scenario 2 and SimpleRNN model	74
Figure 37: Actual vs. Predicted Load for LSTM Scenario 2	74
Figure 38: Actual vs. Predicted Load for GRU Scenario 2.....	75

Figure 39: Actual vs. Predicted Load for SimpleRNN Scenario 2.....	75
Figure 40: Training and Validation Loss for Scenario 3 and LSTM model.....	77
Figure 41: Training and Validation Loss for Scenario 3 and GRU model.....	78
Figure 42: Training and Validation Loss for Scenario 3 and SimpleRNN model.....	78
Figure 43: Actual vs. Predicted Load for LSTM Scenario 3.....	79
Figure 44: Actual vs. Predicted Load for GRU Scenario 3.....	79
Figure 45: Actual vs. Predicted Load for SimpleRNN Scenario 3.....	80

## Table of Tables

Table 1: Initial table of consumption demand.....	34
Table 2: Table of consumption after transformation.....	35
Table 3: Weather Data example of Athens.....	37
Table 4: Final table Format.....	38
Table 5: Performance Metrics for Different Scenarios.....	55
Table 6: Performance Metrics for Scenario 1.....	68
Table 7: Performance Metrics for Scenario 2.....	72
Table 8: Performance Metrics for Scenario 3.....	77

# Abstract

This thesis aims to focus on the usage of recurrent neural networks (RNNs) for the exact short-term electrical load forecasting in Greece. This research is quite relevant, particularly in the context of the energy issues in Greece, which is plagued with expensive electricity. For instance, in Greece in August 2022, the prices for electricity were the highest in Europe with tariffs ranging from 436 Euros for every Megawatt hour. These high rates thus imply that there is the need to enhance the development of efficient and precise methods of electric load forecasting.

RNN, a type of ANN, is designed to handle sequential data and hence, is applied in the modeling and prediction of time series data, including electricity load profiles. To this end, purpose of this study is to examine how various structures and characteristics of these networks influence their effectiveness in event prediction. The research involved developing and implementing state of the art deep learning methods with Python. These algorithms were derived from historical electricity consumption data and statistical error analysis to evaluate the algorithms' forecast of load demand. Most of the contributions were centered on the LSTM models which can be viewed to be rather efficient at making extremely good predictions of time series data due to their intricate architecture.

Other than the LSTM models, this work also explores other types of Recurrent Neural Networks; those include Gated Recurrent Units, which are also known as GRU and the Simple Recurrent Neural Network, which is abbreviated as RNN. As a result of the fact that LSTMs have a more complex gating mechanism than GRUs, the latter are effectively less complex in terms of computation while still providing considerable results in load forecasting. Being the least complex of all the other types, Simple RNNs do assist in capturing the temporal characteristics of load data. The paper also draws comparisons between the models and provides a detailed and exhaustive approach to the hyperparameters' tuning of these models through grid search, the study also outlines the strengths and weaknesses of each model. The application of these various types of RNNs strengthens the performance of the forecasting models, allowing for the understanding of the versatility of the RNN architectures in the context of Greece's energy market.

**Key words:** electric load demand, recurrent neural networks, deep learning, time series forecasting, short-term forecast, LSTM, GRU, Simple RNN, Python, grid search optimization, hyperparameter tuning, model evaluation metrics, RMSE, MAE,  $R^2$  score, historical electricity usage data, predictive accuracy, data scaling, sliding window method, sequential data, advanced neural networks, machine learning, visualization.

# Acknowledgements

This dissertation is submitted during my Master's Degree in Information Systems & Services in the specialization of Big Data and Analytics of the University of Piraeus. Beginning in November 2023 and ending in September 2024, this work has been one of the most important parts of my education.

I am grateful to Professor Michael Filippakis for his valuable guidance and always being there to provide help during the whole process of my thesis. It helped me a lot in my study as he possesses profound knowledge and experience in the field. He not only suggested a number of source materials to me that greatly added to my list of sources but also shared critical feedback and suggestions that guided me in the propagation and focus of my research. I am especially grateful for his encouragement and guidance, without which my thesis could not have been complete.

I would also like to thank my friends for their help and motivation. They supported me in this research study by being there for me emotionally and practically during the difficult times. I am grateful that you have faith in me and for standing by me.

# Section I : Theoretical Part

## 1.1 Historical Overview of Power Systems

This paper provides chronological and geographic information about electricity starting from ancient Greece, for instance, when Thales of Miletus discovered that rubbing fur on amber would result in the amber being able to attract small objects like hair due to the electric charge that had built up on the amber. Over the course of time, the human civilization has come a long way in terms of comprehending and applying the concept of electricity. The name 'electric' was introduced by English physician William Gilbert in the early seventeenth century from the Greek word for amber 'elektron'.

The 18th and 19th centuries marked major developments in inventions that led to development of today's power systems. The first battery was invented by an Italian scientist Alessandro Volta in 1800 AD; therefore, the electric potential got its unit named Volt. The French mathematician and physicist André-Marie Ampère expounded on the relation between magnetism and electric currents, which paved the way for the electromagnetic technologies like motors, generators, and transformers.

As for the sphere of electric power generation and distribution Thomas Edison was a leading figure, and he owned 1093 patents in his practice. Edison's practical incandescent light bulb came in to the market in 1878 and by the end of the nineteenth century he had put up the Pearl Street Station, the first ever commercial electric power station that supplied DC power within a limited area. However, problems with DC power supply for long distance transmission emerged and brought the war of currents between DC supplied by Edison and AC supplied by Tesla. Tesla, an inventor with more than 800 patents, advocated for AC technology in power systems against the DC technology that was already dominant at the time but he was right as AC technology was efficient in the transmission of power over long distances.

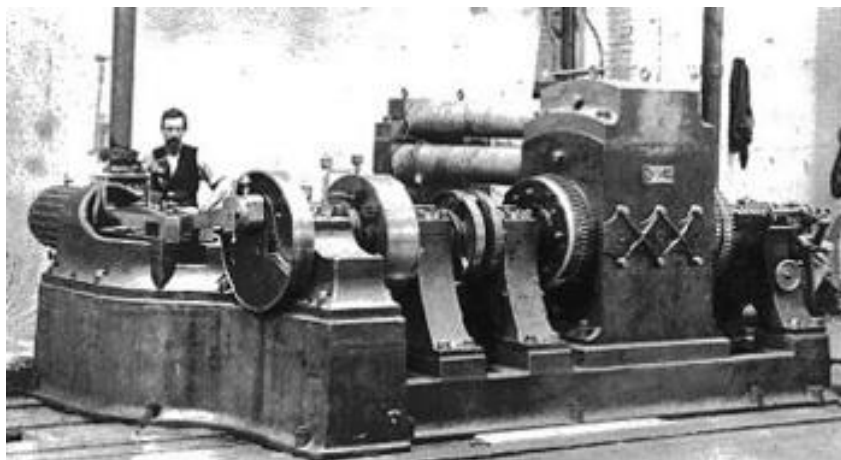


Figure 1: Edison's Jumbo dynamo. Courtesy: National Park Service, Edison National Historic Site. [\[Source\]](#)



More advanced improvements were made in the early 20th Century through the identification of field effect by Lilienfeld and the creation of the first useful bipolar transistor through the efforts of physicists from Bell Laboratories namely Walter Brattain, Bardeen and Shockley which led to the power electronics revolution.

These personalities and their works form the milestones in the development of electric power systems, which has become an essential part of the today's world. The history of electricity can be divided into several stages starting from the initial phenomenon of static electricity to the general use of AC power systems and it has been a process full of innovations, competition and achievements.

## 1.2 Electric Power System (EPS)

An EPS is a complex network that is responsible for the important function of transmitting electricity from power plants where it is produced to the consumers. At the core of this network are three primary components: generation, transmission, and distribution.

Generation is the first stage of power production. Power plants are involved in this process and they can be of different types that are powered by different energy sources, be it the conventional sources of energy such as coal, natural gas, and oil or the new age sources such as wind, solar, hydro and geothermal energy. Energy source selection is a critical factor that influences the system's environmental impact and sustainability.

Transmission is the high voltage electricity transport from the power stations to the substations which are nearer to the consumption centers. Transmission lines are the 'highways' of the EPS which are used to transport large amount of electricity from one place to another within the EPS network with minimum loss. In order to increase the capacity and reliability of these lines, concepts of material science and engineering are used.

Electricity distribution can be defined as the last stage through which electricity is supplied from the substations to the consumers. This step brings down the voltage to levels that are safe to be used in homes, companies, and other facilities. The distribution network becomes more complex due to the need to meet the needs of many consumers while still providing dependable and high-quality power.

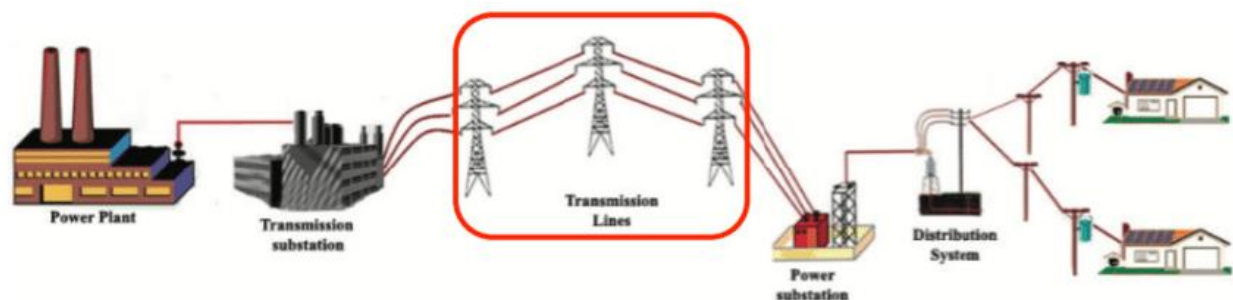


Figure 2: Representation of the Electric Power System (EPS) [\[Source\]](#)

The modern EPS also includes advanced control systems in which remote detection and control as well as automatic switching to maintain balance between the load and the supply, to provide stable network, and to address any issues that may occur. These systems also incorporate IT and big data to enhance efficiency and accommodate DER such as rooftops solar PVs and energy storage systems into the grid. Thus, EPS is developing with the utilization of IoT devices, smart meters, and real-time data communication based on the smart grid technology to improve efficiency and manage energy. Thanks to this technology, consumers are engaged in energy management, demand response, and the connection of renewable energy sources to the grid and adoption of the distributed model.

This paper aims at exploring the various aspects of modernization of Electric Power Systems in order to cope up with the ever rising energy demands of the world population and the shift towards the environmentally friendly energy resources. Technological advancement in energy storage systems like batteries and other types of energy storage systems are crucial for managing the variability of renewable energy sources and hence the energy supply.

### 1.3 Electric Power Load Forecasting (EPLF)

Electric Power Load Forecasting (EPLF) is a critical component of operational and strategic planning of modern electric power systems (EPS). It also covers all stages of the electricity provision process from generation through transmission and distribution to the final consumer, as well as both tactical and strategic levels of the power sector's evolution. In terms of functionality, EPLF has a positive impact on the effectiveness and dependability of power generation and transmission. Thus, load prediction can help power companies to manage the plant production, cut down the costs of operations and prevent unwanted pressures on the grid system. This forecasting lets the utility providers manage the generation in real-time, so there is a correct balance between the supply and demand and prevent the risk of blackouts or systems overload. Also, through forecasting of demand variations, EPLF aids in the efficient management of the grid with regard to the incorporation of intermittent renewable energy sources leading to a more stable and reliable energy supply.

Therefore, EPLF is very important in the formulation of investment on infrastructure and technology. Thus, the long term load demand forecasts are useful in decisions such as the construction of new power plants, upgrading of the transmission system and the creation of distribution networks. This foresight plays a significant role in the management of the EPS in a way that the future energy requirements and technological development are met hence fostering a healthy and sustainable energy industry. EPLF also has a great economic effect on the society. By managing the time at which fuel is bought and the time at which generation takes place, it helps in cutting down losses and thus increases the revenue of power companies. Also, it is capable of dynamic pricing schemes hence, consumers can pay less during off-peak hours, encourage the use of energy-saving products, and reduction in energy wastage. The importance of EPLF to the society cannot be over emphasized as follows. Thus, it supports the stability of the essential services and the functioning of life in general by guaranteeing the availability of electricity. This is because EPLF assist in the lowering of green house emissions through the encouragement of renewable energy sources as opposed to conventional power from fossil fuel. With the shift of energy systems internationally

towards sustainable production, EPLF helps to harmonize the energy production with the environmental objectives.

In conclusion, the EPLF is a pillar in the management and development of electric power systems as it offers a myriad of advantages that are imperative for operational efficiency, planning, financial health, and community well-being. Thus, the role of EPLF in addressing the challenges of the growing demand for electricity and the complexity of the energy sector and directing the EPS to a sustainable development becomes more important.

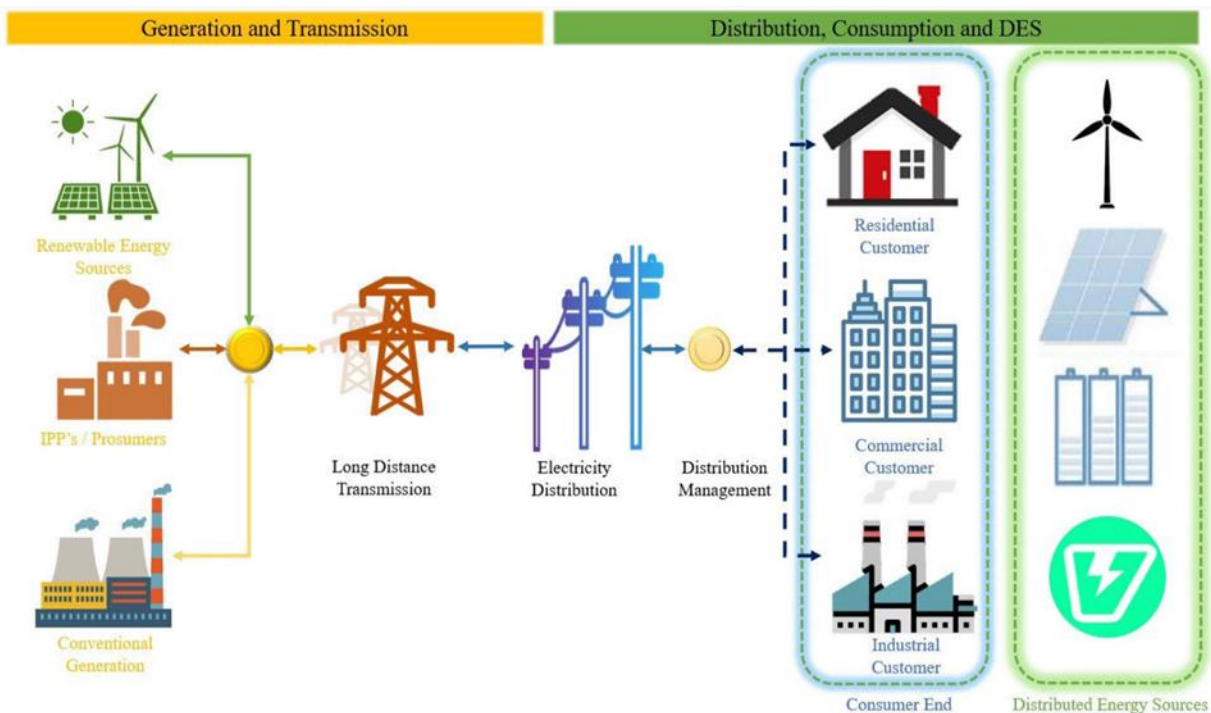


Figure 3: Architecture of an ideal smart grid (SG) with bi-directional information flow, including distributed energy sources

## 1.4 Types of Load Forecasting

EPS require electricity demand forecasting which can be categorized based on the horizon of the forecast. All the categories are useful in the EPS in terms of operation and strategy.

- **Very Short-Term Forecasting (VSTF):** VSTF is used in periods of less than one hour, thus it is vital for the real-time management of supply and demands. It needs to have fast and flexible models to address the changes that occur in the grid at any given time.
- **Short-Term Forecasting (STF):** STF is employed for daily operational planning and takes a few hours to a day, which includes generation and transmission scheduling. This is useful for load profiling which helps in load balancing and also for incorporating the daily renewable energy production.

- **Medium-Term Forecasting (MTF):** MTF is a short to medium term assistance that helps in system planning and procurement which may take a few days to several months. It is useful for maintenance planning and short term resource planning, and can include the effect of the seasonality or market conditions.
- **Long-Term Forecasting (LTF):** Having a time frame of one year or more, LTF is employed in the formulation of strategies and policies, as well as in the provision of infrastructure. It entails the assessment of the long-term dynamics of the economic, demographic and technological factors in relation to the major investments in the EPS.

Every time period has its own peculiarities and requires particular approaches to the forecast to be as precise and credible as possible. The incorporation of renewable energy sources has added more challenges to these tasks of forecasting, and therefore, has made it very crucial to employ sophisticated models to achieve better estimates.

The objective of this research paper is to identify and implement the state-of-the-art DL methods in the field of Short-term Load Forecasting (STLF). This advanced research methodology aims at improving the accuracy and efficiency of the forecasting models in the field of electric power systems that is constantly changing.

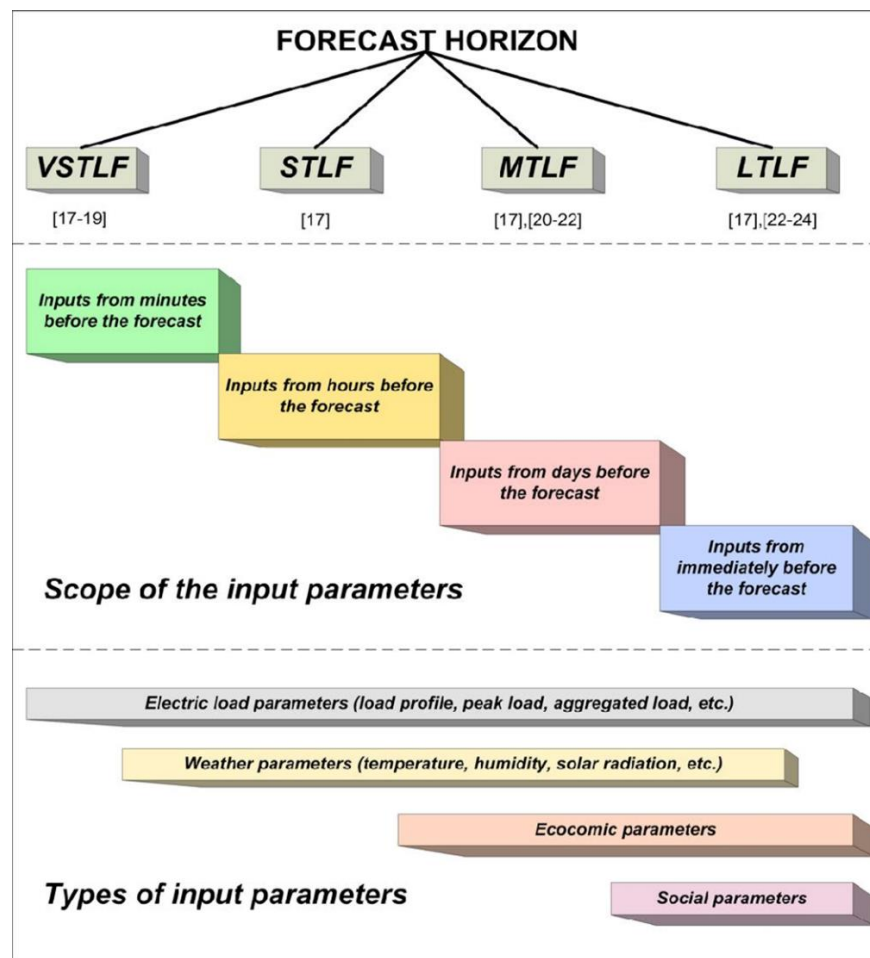


Figure 4: Types of load forecasting models according to the forecast horizon Identification of the scope and type of input parameters [\(Source\)](#)

## 1.5 Key Determinants Influencing Electrical Load Dynamics

In the field of Electric Power Load Forecasting (EPLF), there are several factors that affect the electrical load each of which makes the process of demand forecasting rather complex. Understanding all these elements is crucial in the formulation of an accurate and efficient model for forecasting.

**Temporal Dynamics:** One of the most important factors in EPLF is the temporal aspect, especially in the short-term forecasting. The electricity consumption has a clear trend with daily, weekly, and seasonal patterns. This periodicity is due to the daily, weekly, or annual patterns of consumers' behavior, when specific hours of the day, days of the week, or seasons of the year are characterized by different levels of electricity consumption. These patterns are useful for regulating the supply of electricity in relation to the consumption levels during different time intervals.

**Economic Conditions:** The economic situation of a country is a critical factor that influences the use of electricity. The rate and pattern of electricity usage are dependent on the economic growth, industrial advancement, and market trends. Although this factor is more obvious in the long-term load forecasting, it cannot be ignored in short-term changes in load. GDP growth, industrial production, and consumer expenditures are some of the factors used to forecast short term electricity demand.

**Climatic Influence:** Weather conditions are one of the most important factors that affect the load of electricity especially in the residential and agricultural sectors. Weather conditions such as temperature and humidity affect the consumption of electricity in areas of heating, cooling, and agricultural activities. Therefore, it is crucial to incorporate the weather forecasting into the EPLF models to capture the impacts of climate on the electricity demand.

**Random Variables:** In addition to the above mentioned systematic factors, stochastic factors and black swans also influence the electricity demand. Random events that cannot be anticipated, for instance, a halt in production or an event that is not planned for the public, affects the load in a manner that is hard to predict. These random factors call for some level of flexibility in the models used in making the forecast.

To conclude, the proper forecasting of electricity demand within an EPS depends on the understanding of the various factors that influence it. The dynamics of time, economic factors, climate factors, and other factors define the dynamics of electricity demand, which means that EPLF models must be highly sophisticated and flexible.

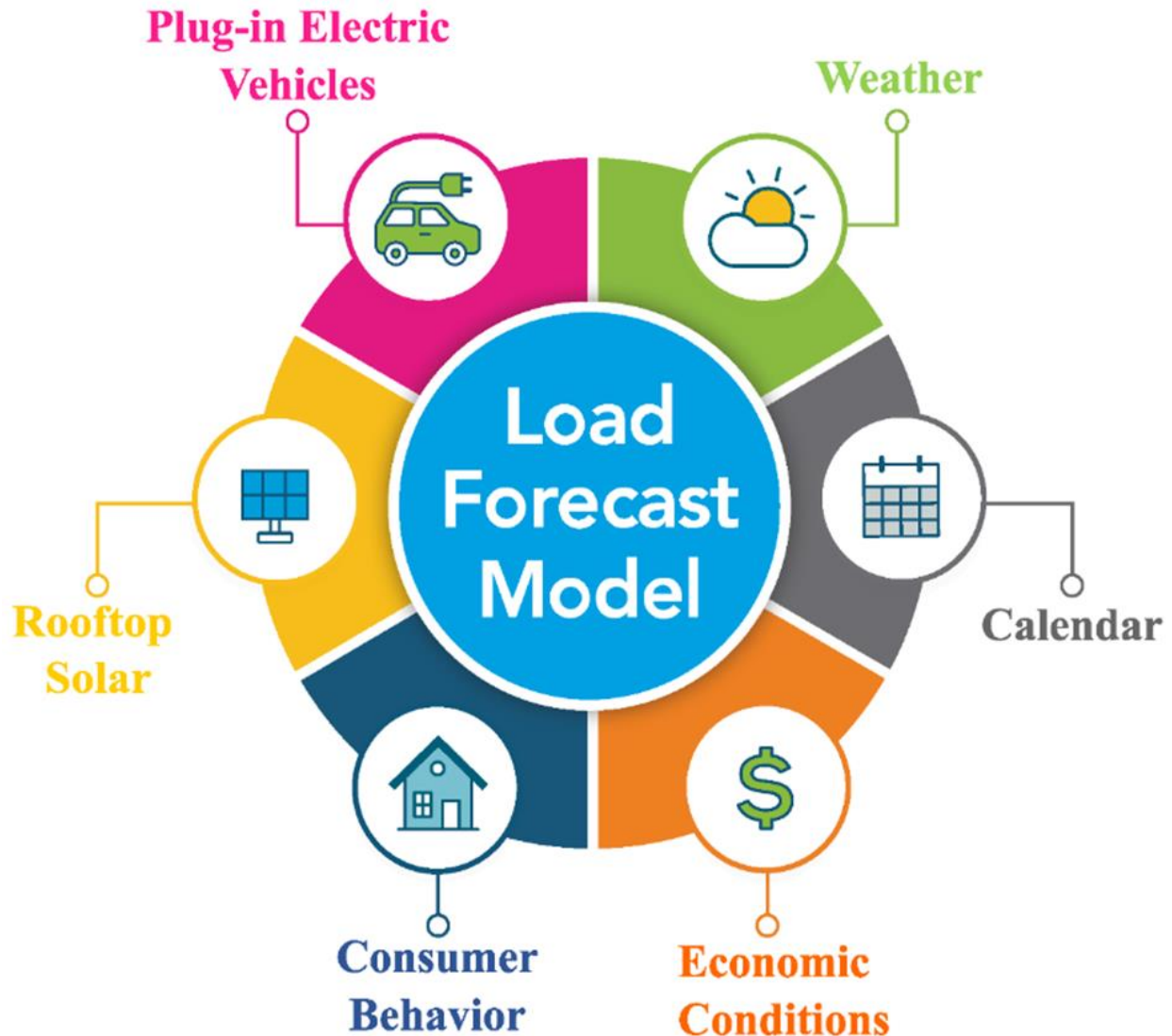


Figure 5: Example of the factors that can affect the LF model [\[Source\]](#)

## 1.6 Fundamentals of Neural Networks: Understanding the Biological and Computational Paradigms

Neural Networks, a general term for a class of models, are based on the biological systems, especially the human brain. Going back to the 19th century, scientists identified that the brain is made up of different units known as neurons that are necessary for communication. These neurons are about 10 billion and are the primary structures of the brain creating many physical connections and pathways. In these networks, every neuron is connected to the others and each neuron has 1000 to 10000 connections which allows for intricate communication in the brain.



A neuron, different in structure, is enclosed by a membrane and has a special function of conducting electrical impulses. The anatomy of a neuron is comprised of three main components: dendrites for the input, the main body of the cell, and the axon which connects with other neurons. The axon transfers the signal to the dendrites of the next neurons through synapses. Neurons receive and integrate input signals in their dendrites and when the total amount of the signals is above a certain level, the neuron produces an output signal along its axon to other neurons.

This neural communication entails several forms of signals. The neuron's potential changes in response to incoming signals, leading to two distinct states: The resting potential and energy potential can be defined as. If the cumulative potential crosses a certain level, then the neuron gets activated and sends an electrical impulse. This process is controlled by the electrical characteristics of the synapses that are the points of contact between neurons and which mediate the transmission of the signals to certain extent. The transmission of information through the brain's neural pathways is a function of energy potential irrespective of the signal type, underlining the complexity of neural communication.

## Neuron

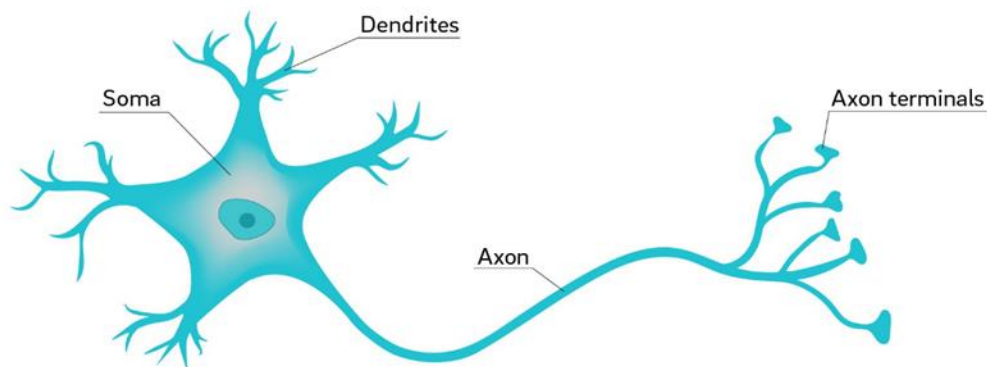


Figure 6: Illustration of a neuron [\(Source\)](#)

The Perceptron is one of the simplest ANNs and it is a model of a neuron itself. It works on input vectors to generate an output using a transition function. ANNs are divided into layers; the input layer, the hidden layer(s), and the output layer. The units in these layers are also connected to other units in the layers which makes them influence each other's activation. The inputs are introduced into the input layer and move through the hidden layers and the final response is obtained from the output layer. ANNs are characterized by several architectural aspects such as the number of hidden layers, number of nodes in each layer, the connections between the nodes, the thresholds, the transfer functions, the initial weights and the training algorithms.

The creation of such models, both natural and artificial, neural networks are shown to be intricate and versatile structures. Artificial networks try to mimic biological neurons, but are in reality, a scaled down version of the marvelous structure of the human brain. This ongoing exploration and simulation of

neural processes are still contributing to the enhancement of knowledge concerning both the biological and artificial intelligence systems, and thus creating a way to new developments in the future.

## 1.6.1 A Journey Through Time: The Evolution of Artificial Neural Networks

Artificial Neural Networks (ANNs) as a field has come a long way from the time it was born. The first appearance of a neural network was in the year 1943, when the neurophysiologist Warren McCulloch and the mathematician Walter Pitts published “A Logical Calculus of the Ideas Immanent in Nervous Activity.” This paper laid down the foundation for neural network.

Hebb introduced the concept of artificial neuron in the year 1949. This was a major advance because it offered at least some foundation upon which more advanced and practical NNs could be constructed. Following the aforementioned, in 1957, the Perceptron, the first neural network computer was developed by Frank Rosenblatt. This invention was critical, because it showed how the concepts of neural networks can be used in computation.

The backpropagation algorithm, which is considered to be the primary method of training of neural network was proposed by Paul Werbos in 1965. This method learning greatly improved the capability of neural networks. After this, in 1980, Kunihiko Fukushima introduced the first Convolutional Neural Network (CNN) which opened up a new avenue in the pattern recognition area for instance, handwritten digit recognition.

The ensuing two decades of the 1990s and the 2000s were significant with major achievements including the development of LSTM by Sepp Hochreiter and Jürgen Schmidhuber in 1998 and the deep learning model by Geoffrey Hinton and Ruslan Salakhutdinov in 2006. They were very helpful in solving problems which demand sequence data analysis as well as in the construction of advanced neural networks.

Another demonstration of the capabilities of deep learning was seen in 2012 when Geoffrey Hinton and his team came up with AlexNet, a deep convolutional neural network which significantly enhanced the performance of image recognition. The advancement of deep learning was further boosted with outstanding performances in AI; for instance, Google’s AlphaGo triumphing over world champion Lee Sedol in Go in 2015 and AlphaZero surpassing the best chess engines globally in 2017.

To the rapid advancement of ANNs in the year 2022, the creation of the ChatGPT, one of the most sophisticated language models, can also be attributed. ChatGPT has achieved excellent performance in many NLP tasks, which has proved the potential of neural networks in processing and generating human language.

Moving forward, the area of ANNs has a lot of possibilities. There could be neural networks that do not replicate human conscience but enhance it and therefore, come up with solutions that will revolutionize health, environment and many other fields. It can be expected that in the future there will be ANNs with the ability to make decisions on their own, outsmarting the current ones and stepping into the creative and innovative territory that was previously only accessible to the human mind.



Thus, the history of ANNs is not only a sequence of technical advancements, but also a history of people's inspiration and creativity. It depicts the society's efforts to demystify the human intelligence and attempt to emulate it through artificial systems and innovations that are being made one after the other.

## 1.6.2 Diverse Architectures of Artificial Neural Networks

ANNs have been developed into many structures, each of which is intended to solve certain tasks and operate with certain data. In this regard, we discuss some of the most well-known architectures including feedforward neural networks, recurrent neural networks (RNN), convolutional neural networks (CNN).

**Feedforward Neural Networks (FNNs):** Multi-layer perceptron (MLP) is the most basic structure of ANNs which is a feedforward neural network. These are networks in which data passes in one direction from the input layer to the output layer through one or more layers of hidden nodes. These hidden layers are very important for capturing the relationships in the data and to convert the input data into output. Feedforward networks have one major feature, and that is the adjustment of weights between inter-neurons through a process referred to as backpropagation that aims at improving the network's predictive capabilities using gradient descent algorithms.

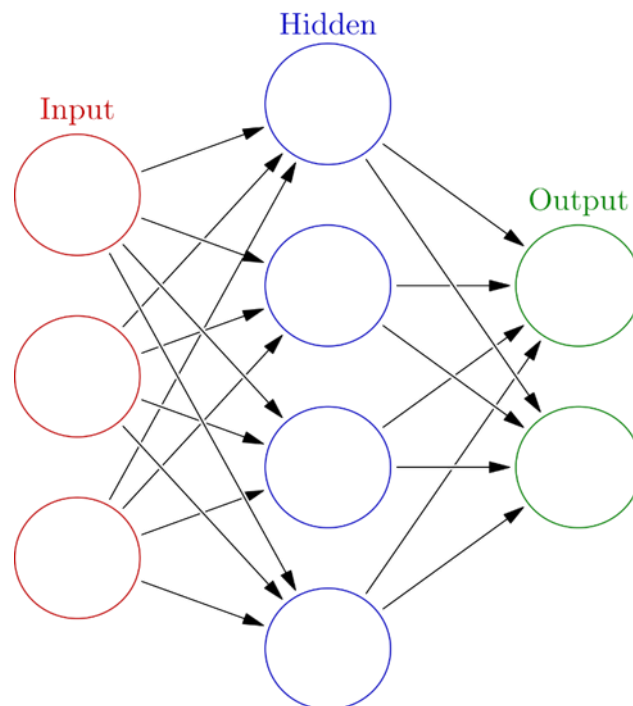


Figure 7: Example of a Feedforward Neural Network (ANN) [\[Source\]](#)

**Recurrent Neural Networks (RNNs):** RNNs are designed to work with sequential data such as textual data or data with temporal relation. Among the characteristics of RNNs, it is necessary to single

out feedback connections, which allow information to be stored and transferred across different input sequences. This architecture generally includes a hidden state that passes information between different time steps and, as a result, the model's outputs depend on the previous and current inputs. LSTM networks that fall under the category of RNNs lessen the vanishing gradient problem with the help of specific control gates. RNNs are especially good at applications where sequence of data is used, like speech recognition and language translation.

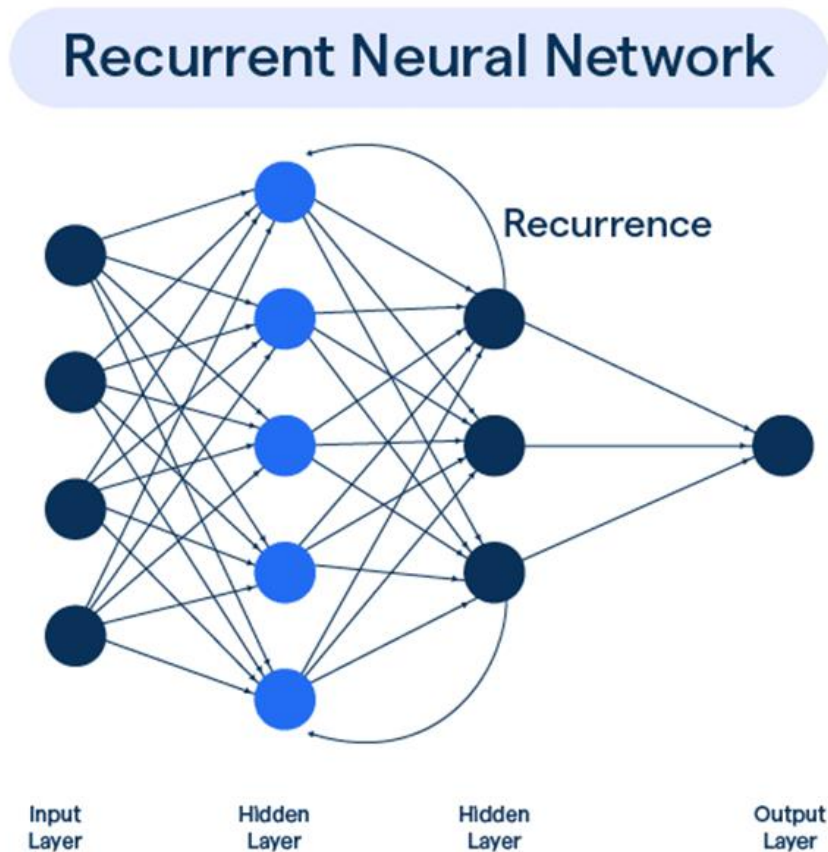


Figure 8: Example of a Recurrent NN (RNN) [\(Source\)](#)

**Convolutional Neural Networks (CNNs):** CNNs are mainly applied in image processing and they are packaged to work for data that has a grid like structure. The architecture has convolutional layers which apply number of filters to the input and extracts features from it and then there are pooling layers which basically down sample the data. This design allows CNN to self-learn and extract the spatial pyramids of the data which makes them suitable for computer vision applications.

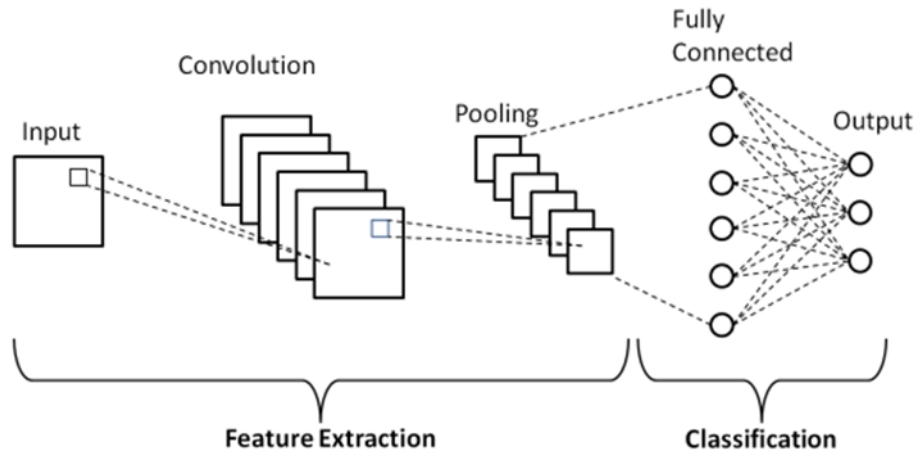


Figure 9: Example of Convolutional Neural Network (CNN) [\[Source\]](#)

**Layered Complexity in ANNs:** Most of the ANNs have at least one hidden layer containing nodes which may be fully or partially connected. Generally, the networks with one hidden layer and all-to-all interconnection are used in various fields. The placement of these layers and nodes is a basic feature of ANN's architecture and enhances its capacity to identify and details in data.

Every type of ANN from the simplest feedforward network to the more complicated RNNs and CNNs, has certain strengths. These include from the fundamental data classification in feedforward networks to the complex applications of natural language processing in RNNs and image recognition in CNNs, which confirms the effectiveness of the neural networks.

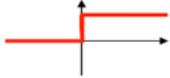
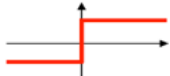
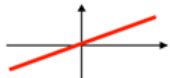
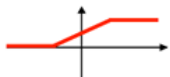
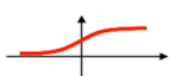


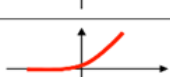
### 1.6.3 The Role of Activation Functions in Artificial Neural Networks

The functions used to introduce non-linearity in the output of neural network are called activation functions or transfer functions and are an essential part of neural networks. This non-linearity is crucial to the network's capacity to learn and imitate patterns in the data. In the realm of ANNs, activation functions are typically divided into two broad categories: Hence, there are two types, namely linear and non-linear.

Sigmoid, Tanh, and ReLU functions are some of the linear activation functions that work with direct proportionality between the input and the output to provide reliable results. The Hard Limiter Function and the Step Function that give a constant output when a certain boundary is crossed and the Signum Function that partitions the input into three regions of output are the examples of this category. Despite the fact that they are among the simplest types of models, the linear functions are not capable of analyzing multivariate data because of their linear structure.

On the other hand, non-linear activation functions are important to the enhanced features of neural networks especially in deep learning. The Sigmoid Function that transforms inputs to the output layer into a range between 0 and 1 is suitable for output layers that are supposed to give probabilities. The

Hyperbolic Tangent (Tanh) function scales the input between -1 and 1 and produces an output that is symmetric which is beneficial in some neural network architectures. Among them, there is the Rectified Linear Unit (ReLU) which simply sets negative values to zero and keeps positive values, known for speeding up the training process and adding non-linearity whereas introduced non-linearity is one of the most important factors in the learning process of the network.

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016  
(<http://sebastianraschka.com>)

Figure 10: Activation Functions for ANNs [\(Source\)](#)

It is imperative to choose a suitable activation function and the choice is determined by the structure of the network and the type of data used. Even though linear functions are useful for the simpler models, non-linear functions are essential for the networks that are working with patterns and deep learning. Each of them, with their own properties, occupies a certain level in neural networks, affecting the learning process of the networks and the efficiency in various tasks. Therefore, activation functions are important components of the ANNs as they define the learning algorithm and performance of the network in regards to classification and data modeling problems.

## 1.6.4 Understanding Recurrent Neural Networks

Earlier in this paper, Recurrent Neural Networks (RNNs) have been introduced as one of the most important subcategories of Artificial Neural Networks because of its capacity to handle sequential data.

However, the major difference between RNNs and feedforward neural networks is that RNNs contain feedback connection which enable it to keep some form of memory concerning previous inputs. This section focuses on explaining the working and the mechanism of RNNs.

RNNs are known to have loops which make it possible for the networks to store information and therefore, they have memory. This characteristic enables RNNs to continue having a chain of thoughts which is very useful when working with sequences. In other words, they do not received each input independently: they also have a memory of what has been received before, which makes them particularly suitable for sequential data. This architecture is different from the conventional neural networks that do not have this memory part and work with each input separately.

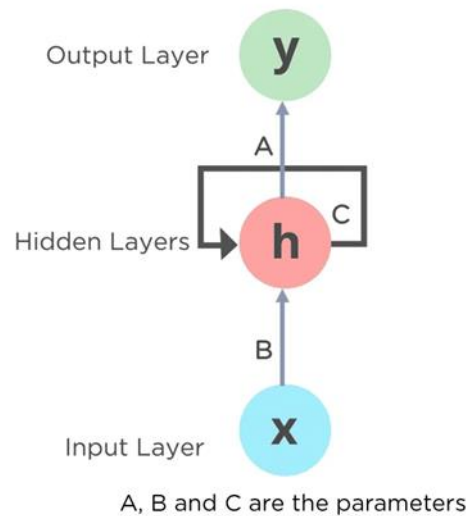


Figure 11: Architecture of RNN [\[Source\]](#)

An example of such RNN architecture is depicted in Figure 11. RNNs have a structure that is characterized by an iterative process through which information is passed from one layer to another. This can be depicted as a chain of numerous neural network units where information is transferred to the next unit. This chainlike structure indicates that the operation of RNNs is mainly related to the sequential data, for instance, lists and sequences. RNNs are well suited to numerous applications due to the fact that sequence input/output is inherent within their architecture, and include speech recognition, language modeling, translation, and image captioning.

Furthermore, RNNs are very effective in cases when there is a relationship between the past and the present information. It is worth using RNNs for implementing simple cases when the most recent data is enough to make a prediction or classification and where there is no need to store all the data. Nevertheless, there is a problem when the context goes deeper in the past, and that is when RNNs work worse. This is because of the fact that the probability of being able to learn how to associate information from one point to another reduces as the distance between the two points in question rises.

On the other hand, one of the main issues of RNNs is that they suffer from the vanishing and exploding gradient problem as a result of the depth of the network in time. This issue can complicate the training of the network and its ability to recognize long-term relations. New structures for architectures

like Long Short-Term Memory (LSTM) have been designed to solve this issue and enhance the capability to learn the dependency over long sequences.

### 1.6.5 The Evolution and Mechanisms of Long Short-Term Memory (LSTM) Networks in Depth: A Comparative Analysis with Traditional Recurrent Neural Networks (RNNs)

The LSTMs are a type of Recurrent Neural Network (RNN) which has architecture that is capable of addressing the issues with other RNNs and their limited capability to learn relationships between distant time steps. Understanding LSTM thus boils down to understanding the structure that is used in order to learn long term dependencies. The building block of an LSTM network is known as LSTM cell which incorporates the gates. These are the input gate, output gate, and forget gate which regulate the input and output of the cell and what should be retained or discarded. This property of managing memory is helpful in solving the vanishing gradient problem which is a typical challenge with standard RNNs in that the network's ability to capture information from previous time steps diminishes with increasing time steps.

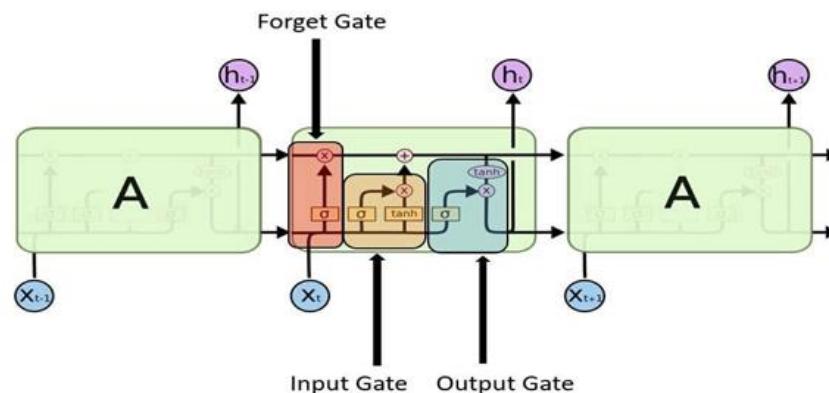


Figure 12: Architecture of LSTM [\(Source\)](#)

Hence the use of LSTM networks has been widely applied in the analysis of time series and sequential data to deliver very accurate results. For instance, they have been applied in the area of the financial markets where they have been observed to outperform other methods that do not incorporate memory. This is because the LSTMs have the ability to learn long-term dependencies in the sequential data which helps in making better out-of-sample directional movements' predictions.

Hence the use of LSTM networks has been widely applied in the analysis of time series and sequential data to deliver very accurate results. For instance, they have been applied in the area of the financial markets where they have been observed to outperform other methods that do not incorporate memory. This is because the LSTMs have the ability to learn long-term dependencies in the sequential data which helps in making better out-of-sample directional movements' predictions. [8]

In this way, the main distinction between LSTMs and RNNs is the architecture and the feature of solving the issues with long-term dependencies. However, it is known that RNNs are supposed to be able to model long dependency; in practice, they do not because of the problem of gradient explosion. This is because the proposed specialized LSTMs, which have gate mechanisms, are designed to address this issue and thus to be capable of capturing information from more time-stamps' steps. LSTMs are thus very useful for sequences that are long and situations where there is a large time difference between two useful pieces of information. Thus, the LSTMs have been designed in a manner that their performance and efficiency are improved by the use of peephole connections and other gating structures for memory control.

Consequently, LSTMs can be viewed as one of the most effective solutions for developing nn in the context of sequential data. As opposed to the normal RNNs, they have improved performance in the management of long-term dependencies; therefore they can be applied in language translation, financial analysis, among others. Shift from RNNs to LSTMs is seen as a solution to the above problems in sequence modeling and especially tasks where context in a sequence is important.

### 1.6.6 Unveiling the Gated Recurrent Unit (GRU): A Comparative Analysis with RNNs and LSTMs

The GRU networks are an enhanced version of RNNs that were proposed to help overcome the issue of vanishing gradient problem, which affects the performance of RNNs. The major novelty of GRU is the gating mechanism which aims at controlling the information flow. This mechanism is composed of two gates: The two gates which are the update gate and the reset gate are also used. The update gate is used to decide which part of the past information (from previous time steps) has to be passed on to the future while the reset gate decides which part of the previous information has to be forgotten. This structure provides the ability to GRUs to learn the dependence from the long sequences data without the drawbacks of the gradient vanishing so widespread in the traditional RNNs. Also, the results have shown that the GRUs are almost as effective as another sophisticated RNN model – the LSTM for tasks including polyphonic music modeling, speech signal modeling, and other tasks, but with fewer numbers of parameters and a simpler architecture that enhances the computational efficiency.

LSTMs are made simpler by GRUs where the input and forget gates are combined into an update gate. They also combine cell state and hidden state, which simplifies the structure and the computations without having much impact on the network performance. This makes GRUs a great candidate for those applications where model complexity and efficiency are of the same significance as prediction quality. For instance, in Sequential modeling problems such as NLP and speech recognition, GRUs have given good performance with fairly lesser time consumption than the RNNs and LSTMs.

In relation to RNNs and LSTMs, the major strength of GRUs is that they reduce the vanishing gradient problem while requiring fewer resources than LSTMs. This is why GRUs are ideal to use in devices that have a low computational power or when the processing has to be done in real time. While LSTMs are still the go to model for tasks that may require the additional layers of the LSTM due to the handling of long term dependencies, GRUs present a good balance of architecture complexity, computational



requirements and accuracy. Comparing the two, GRU and LSTM, the decision mainly depends on the particular task to be solved and the number of calculations that can be made.

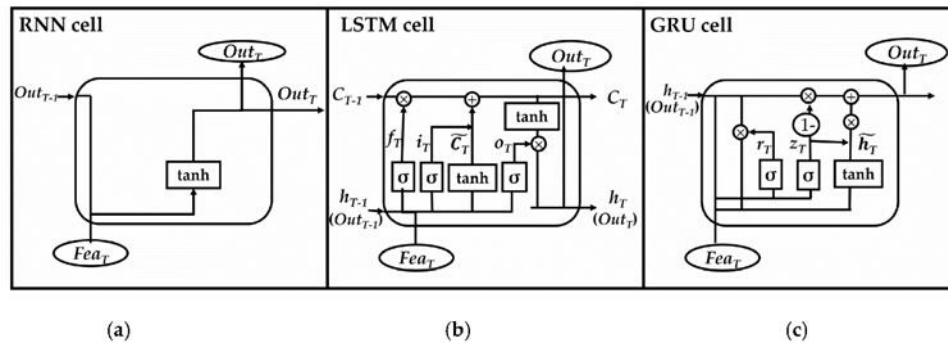


Figure 13: Architectures comparison of RNN - LSTM - GRU [\[Source\]](#)

Considering all that has been said, GRU can be considered as a major contribution to the creation of neural networks for sequential data processing. Due to the problems with traditional RNNs and as a more accessible solution than LSTMs, GRUs have extended the range of fields that can be effectively used with deep learning models with sequence and time series data.

### 1.6.7 Operation and Training of Artificial Neural Networks (ANNs)

The operation of artificial neural networks (ANNs) involves two primary stages: the training phase and the prediction phase.

In the training phase, the neural network is trained on data known as the training set, through a process of adjustments of the internal weights of the neural network. This process starts with the feed of a number of observation values into the network. The network's weights and biases are gradually adjusted in an attempt to reach the smallest possible prediction errors. The objective of this phase is to fine-tune these parameters in order to better describe the data by employing an adequate learning algorithm. In training, there are several strategies that can be used for instance gradient descent technique to fine tune the weights of the given network.

After the training stage, the neural network enters the prediction phase in which its efficiency is measured with the help of a test set. In this stage the network uses the weights and the biases which have been determined during the training process to new input data and produces output values. The above values are then compared to the actual values in the test set to compute the statistical prediction error. This error measurement is very important because it allows for the determination of the networks' accuracy and its ability to generalize.

However, there are Several aspects that have to be carefully taken into consideration to enable an efficient operation of an ANN. First of all, the architectural design of the network has to be optimal. This entails; the number of hidden layers, the number of neurons per layer, the activation functions, the training algorithm, and the number of epochs. All of these elements are very important in the network and its learning process as well as ability to work with data and deduce from it. Secondly, data partitioning



is important as follows. The data which is available must be split into two, the training set and the test set. It is a general tendency to allocate 75-80 percent of the data for training and the remaining fifteen to twenty percent for testing. This division helps the network to have enough data for learning while at the same time having enough data for performance assessment. Lastly, the neural network must have a good prediction of the training data set as well as the testing data set while maintaining low statistical prediction errors. The training process is a successive process that can be rather slow, particularly in the case of big data or a large number of neurons. Generally, the number of neurons in the hidden layer is selected based on the nature of the problem and experimentation. If the number is rather small, then the network may not be able to learn from the data at all which is called underfitting. Similarly, if the number of neurons is too high, the network is likely to over-fit and produce good results for the training data but poor results for new data. There is an important consideration on the neurons' count and it is entirely dependent on the problem at hand and the size of data. Furthermore, it is necessary to look at the sorts of activation functions used in the network, as well. The activation functions help to bring non linearity in the network and hence to learn the complex relations in the data. Other activation functions that are frequently used are sigmoid function, hyperbolic tangent function (tanh), and rectified linear unit (ReLU). The type of activation function used in the network greatly affects the training process as well as the network's ability to predict the output. Also, the choice of the training algorithm may influence the time and efficiency of the learning process. There are various algorithms like Stochastic Gradient Descent, Adam, RMSprop and each of these has its own pros and cons in terms of convergence and computation.

## 1.6.8 Supervised and Unsupervised Learning: The Two Pillars of Training Artificial Neural Networks

ANNs are the computational systems that have the capability of learning from input data and appearing the output through a number of internal mechanisms. Two major learning paradigms that are commonly used in neural networks include Supervised learning and unsupervised learning.

In supervised learning, the network is trained using the labeled data where for every input there is one correct output. This method is mainly used in activities like regression and classification to decide on the outcome based on the input data. Some of the most used algorithms of supervised learning are logistic regression, naive Bayes, support vector machines, artificial neural networks and random forests. Supervised learning is widely used in speech recognition, image classification, and predictive analytics among other applications.

On the other hand, unsupervised learning is the process of training the network with the data which is not accompanied with the labeled responses. This manner of analysis is primarily preoccupied with the goal of identifying hidden patterns within the data, which may include grouping similar data points. This technique is common in applications that include image classification for instance in photo recognition; the system sorts the images based on the similarity in the patterns without prior information on the categories. Some of the other important areas where unsupervised learning can be applied are as follows; anomaly detection, market basket analysis and dimensionality reduction.

Therefore, it is crucial to state that the two main categories of learning techniques that are used in the training of ANNs are supervised and unsupervised learning and that each of these techniques is used in a different set of applications ranging from predictive analytics to image recognition.

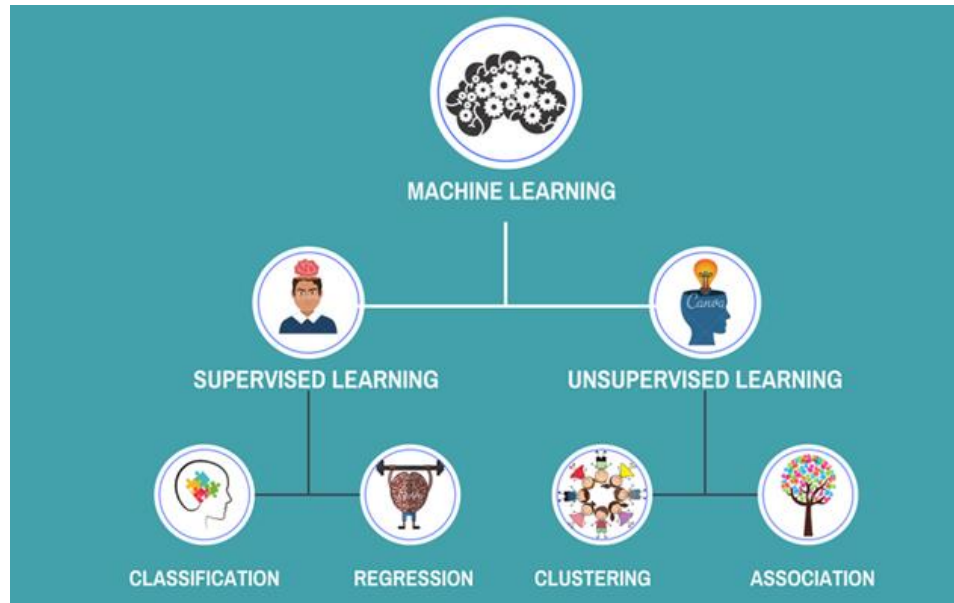


Figure 14: Supervised vs Unsupervised Learning [\[Source\]](#)

## 1.6.9 Key Error Metrics for Evaluating ANN Predictions

While comparing the results of artificial neural network (ANN) with the actual results several statistical measures for finding the error are used. These metrics assist in determining the level of risk in estimating a particular physical quantity, presented as a value with its uncertainty. Below are the key error metrics used to evaluate the precision of ANN forecasts, along with detailed descriptions of each.

Below are the key error metrics used to evaluate the precision of ANN forecasts, along with detailed descriptions of each:

- Mean Squared Error (MSE):** Mean Squared Error is a fundamental metric that represents the average of the squared differences between predicted values and actual values. This metric penalizes larger errors more severely, making it useful for assessing the overall performance of the prediction model. Mathematically, MSE is expressed as:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

where  $y_i$  are the actual values,  $\hat{y}_i$  are the predicted values, and  $n$  is the number of observations. By squaring the errors, MSE ensures that larger discrepancies between predicted and actual values have a more significant impact on the final error metric.

- **Root Mean Squared Error (RMSE):** Root Mean Squared Error is derived from MSE and provides an error metric in the same units as the original data. It is calculated as:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

RMSE is advantageous for understanding the magnitude of prediction errors, as it translates the error into the same scale as the data being analyzed. It is widely used in regression analysis to evaluate model performance and interpret the practical significance of the prediction errors.

- **Mean Absolute Error (MAE):** Mean Absolute Error measures the average absolute difference between predicted values and actual values, providing a straightforward interpretation of the average error magnitude. It is calculated as:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

MAE is less sensitive to outliers compared to MSE, making it a reliable metric for understanding the typical error magnitude in predictions without disproportionately emphasizing larger errors.

- **Mean Absolute Percentage Error (MAPE):** Mean Absolute Percentage Error expresses the prediction error as a percentage of the actual values, offering a normalized measure of accuracy that facilitates comparisons across different datasets or models. MAPE is defined as:

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100$$

This metric is particularly useful for comparing model performance in contexts where the scale of data varies significantly, as it provides a percentage-based error measurement.

- **Mean Bias Deviation (MBD):** Mean Bias Deviation measures the average bias in predictions, indicating whether the model tends to systematically overestimate or underestimate actual values. It is calculated as:

$$MBD = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)$$

MBD helps in understanding the direction and magnitude of bias in model predictions, providing insights into systematic errors that might need correction.

- **Symmetric Mean Absolute Percentage Error (sMAPE):** Symmetric Mean Absolute Percentage Error is a variation of MAPE that ensures both overestimation and underestimation are equally penalized. It is defined as:

$$sMAPE = \frac{1}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{(|y_i - \hat{y}_i|)/2} \times 100$$

- MAPE is particularly useful in ensuring symmetry in error measurement, making it a fair metric for evaluating the accuracy of forecasts where both types of errors are critical.

These statistical error metrics are essential for thoroughly evaluating the performance and reliability of ANN predictions. They allow for the refinement and optimization of models to achieve greater accuracy and robustness in various forecasting applications.

In addition to the key error metrics, the Coefficient of Determination ( $R^2$ ) and Adjusted  $R^2$  are also important for evaluating ANN predictions.  $R^2$  measures the proportion of variance in the dependent variable that is predictable from the independent variables, providing an indication of the model's goodness of fit. It is defined as  $R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$ , where  $\bar{y}$  is the mean of the actual values.

Adjusted  $R^2$  modifies the  $R^2$  value based on the number of predictors in the model, offering a more accurate measure for models with multiple predictors. These metrics, along with the previously discussed error metrics, provide a comprehensive framework for assessing the accuracy and reliability of ANN predictions, facilitating the development of more precise and effective forecasting models.

## 1.7 Time Series Analysis and Forecasting: Concepts and Techniques

A time series is a set of data which is arranged in a manner that it has a time axis with each data point being associated with a particular point in time. These are data that are usually collected at particular intervals say daily and the data could be numerical or categorical. Time series data is often in tabular form where the data is divided by time intervals and usually stored in a CSV format where one column has time stamp and the other has the data values.

Time series represent data indexed over time, with time as the independent variable and the dependent variable being factors such as:

- Stock market prices
- Company sales data
- Sensor readings from smart devices
- Electrical energy consumption measurements

To derive meaningful insights from time series data, it is crucial to decompose the series and analyze its fundamental components. These include the general direction or trend of the data, systematic changes that occur at regular intervals known as seasonality, repetitive cycles of business activity, and other variations which are irregular. For instance, when analyzing time series data of an airline, one could identify seasonality that is related to holidays, passengers' traffic, and certain cycles that are tied to the

economic cycles. Describing these components is important, as it allows one to choose suitable models for time series forecasting.

**Seasonality** means the occurrences that happen at certain intervals that could be daily, weekly, monthly, or yearly. Seasonality is an important aspect that should be considered when making forecasts on data as it reveals the cyclic trends of data. Techniques of data transformation include seasonal decomposition, differencing, and Holt-Winters method that can help to eliminate seasonality. However, not all time series exhibit seasonality since this aspect depends on the data that is under analysis and the context in which it is collected.

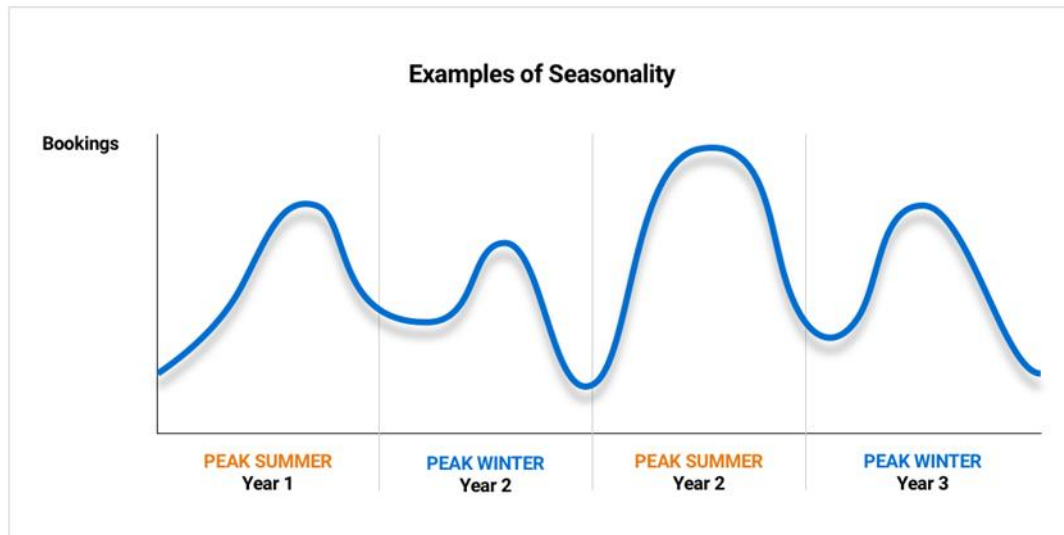


Figure 15: Example of Seasonality [\[Source\]](#)

**Trend** in a time series refers to the course of action, which depicts the direction of the data either rising, falling or stable in the given period. Trends are long lasting patterns which are shaped by such factors as economic and demography. Trends are a way of analyzing the situation and predicting future behavior of individuals or groups of people. A few of the common methods include linear regression, moving averages, exponential smoothing to name but a few. It is also necessary to identify the points of change when trends may turn.

**Cyclic behavior** is a pattern that recurs over a time span longer than a season and is often attributed to exogenous factors such as the business cycle. Therefore, cyclicity is evident in variables such as GDP, employment, and production rates. It is possible to identify cyclic behavior in order to reveal general trends and patterns and increase the level of forecast precision. The techniques that are used to identify cyclic behavior include decomposition, spectral analysis, and wavelet analysis.

**Irregular fluctuations** are variations that cannot be attributed to trend, seasonality or cyclical patterns. Such fluctuations are usually brought about by events that are beyond the control of the company or changes in the market needs. These variations are crucial to detect in order to get to know the tendencies of the data and to enhance the results of the predictive models.

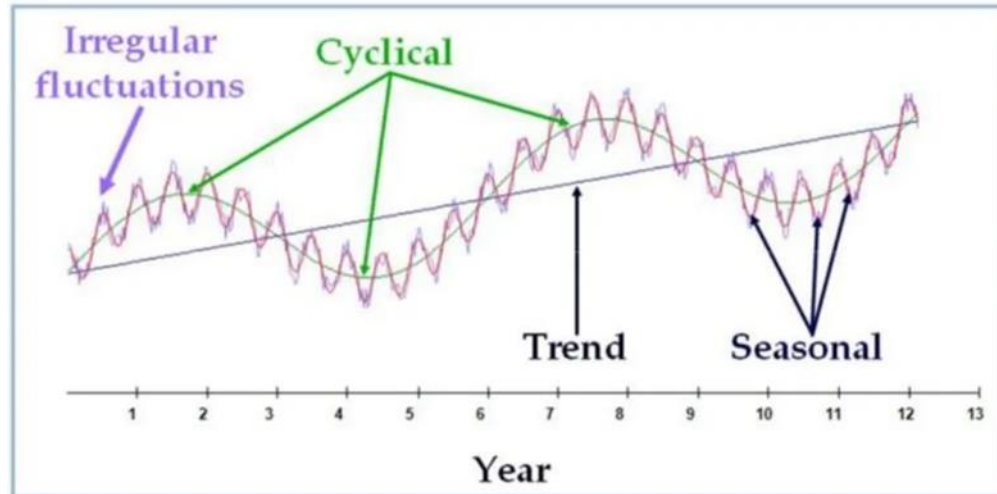


Figure 16: Trend, Seasonality, Cyclic behavior, and Irregular fluctuations ([Source](#))

### 1.7.1 Main Aspects of Time Series Forecasting

Time series analysis is a statistical technique that deals with the prediction of future values from the past record. This forecasting can be categorized into two primary aspects: the quantity of the time series to be predicted and the time span for the prediction. The two types of time series forecasting are univariate and multivariate.

**Univariate Forecasting** is a type of forecasting that deals with a single variable at a time. The main goal is to forecast the values of this one variable based on the past observations of this variable. For instance, predicting the sales of a certain product in the next three months based on the previous sales records is a univariate forecast. It is preferable because it uses less complex models, is easy to understand, and takes less time to compute. Nevertheless, it has some drawbacks, including the restriction of the information search space and the possibility of ignoring other factors that may affect the results.

On the other hand, **Multivariate Forecasting** is a type of forecasting that deals with more than one time series variable whereby the forecast of one variable is dependent on several other related variables. For example, predicting the sales of a given product based on the past sales records, weather conditions, the amount of money spent on advertisement, and the general economic situation. This approach provides a better model comprehensiveness, better accuracy, and the ability to model more intricate relationships between the variables. However, it also comes with its own set of problems such as the models becoming more complex, the computations becoming more demanding, and the results being harder to explain.

It is also possible to classify time series forecasting in terms of the time horizon for the prediction, namely, short-term and long-term.

**Short-term forecasting** is the process of estimating the values in the near future within days, weeks, or months. An example of short term forecasting is the estimation of the daily electricity demand

for the following week. This type of forecasting is mostly based on frequent data, including hourly or daily data and is used for the operational and tactical level planning, inventory management, and scheduling. Some of the most popular models used in short-term forecasting are exponential smoothing, ARIMA, and simple moving averages because they are able to respond to changes in the recent data quickly.

**Long-term Forecasting**, on the other hand, is the process of making predictions about values over a longer period of time, for instance, months, years, or even decades. For instance, predicting the population change in the next ten years is a case of long-term forecasting. This type usually deals with less frequent data, for instance, monthly, quarterly, or yearly data and is useful for strategic planning, capacity planning, and long-term investments. Long term forecasting is done using models like SARIMA (Seasonal ARIMA), Holt-Winters exponential smoothing and various machine learning models like recurrent neural networks, long short-term memory networks etc.

## 1.7.2 Time Series Models: AR, MA, ARMA, ARIMA

Time series models are critical tools for the analysis and forecasting of data points collected or recorded in sequential order over time. Among the most prominent and widely utilized models in this domain are the AutoRegressive (AR) model, the Moving Average (MA) model, the AutoRegressive Moving Average (ARMA) model, and the AutoRegressive Integrated Moving Average (ARIMA) model.

The **AutoRegressive (AR) model** utilizes past values of a variable to predict its future values, assuming that the current value is a linear combination of its previous values and a stochastic term. Formally, an AR model of order  $p$  (AR( $p$ )) is represented as:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \epsilon_t$$

where  $y_t$  is the current value,  $\phi_1, \phi_2, \dots, \phi_p$  are the parameters of the model, and  $\epsilon_t$  is the white noise error term.

The **Moving Average (MA) model** focuses on the dependency between an observation and a residual error from a moving average model applied to lagged observations. It models the current value of the series as a linear combination of past forecast errors. An MA model of order  $q$  (MA( $q$ )) is expressed as:

$$y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q}$$

where  $\epsilon_t$  is the white noise error term, and  $\theta_1, \theta_2, \dots, \theta_q$  are the parameters of the model.

The **AutoRegressive Moving Average (ARMA) model** combines both AR and MA models to describe a time series. It captures both the relationships between past values and past errors, making it suitable for stationary time series data. An ARMA model of order  $p$  and  $q$  (ARMA ( $p, q$ )) is defined as:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q}$$

where  $y_t$  is the current value,  $\phi_1, \phi_2, \dots, \phi_p$  are the autoregressive parameters,  $\theta_1, \theta_2, \dots, \theta_q$  are the moving average parameters, and  $\epsilon_t$  is the white noise error term.

The **AutoRegressive Integrated Moving Average (ARIMA) model** extends the ARMA model by including differencing to make the time series stationary. This model is particularly useful for non-stationary data. ARIMA incorporates the AR and MA components along with an integration component that represents the differencing of raw observations to make the series stationary. An ARIMA model of order  $p$ ,  $d$  and  $q$  (ARIMA( $p$ ,  $d$ ,  $q$ )) is represented as:

$$\Delta^d y_t = \phi_1 \Delta^d y_{t-1} + \phi_2 \Delta^d y_{t-2} + \dots + \phi_p \Delta^d y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q}$$

where  $\Delta^d$  denotes the differencing operator applied  $d$  times,  $\phi_1, \phi_2, \dots, \phi_p$  are the autoregressive parameters,  $\theta_1, \theta_2, \dots, \theta_q$  are the moving average parameters, and  $\epsilon_t$  is the white noise error term.

These models form the foundation of time series analysis and forecasting, providing powerful tools for understanding and predicting future values based on historical data.



# Section II : Implementation

## 2.1 Electric Load Data Extraction and Preparation

In this thesis, the daily electricity demand was obtained from the ENTSO-E website instead of directly from ADMIE as the Independent Power Transmission Operator. The following are the reasons that informed this decision.

First of all, a Selenium script was developed to download and format the data from the ADMIE website. The following script was then used to extract and clean the data so that it was in a format that could be analysed. The purpose of the script was to move through ADMIE's website, obtain daily SCADA system load files, and save them in a specific directory.

This script first defines the start and end dates for the data extraction period and forms the URLs for each of the days in between. It then proceeds to set the Chrome WebDriver to download the data files to a folder of the user's choice and it proceeds to loop through the URLs to download each file.

However, after conducting detailed checks, it was revealed that there were some discrepancies in the ADMIE data where 33 days of electricity demand records were missing. Plugging in these gaps with median values would have brought in errors and inconsistency which could have affected the credibility of the models used in forecasting.

To ensure high quality of the data, it was decided to obtain the data from ENTSO-E instead. ENTSO-E provides several advantages:

1. **Completeness of Data:** ENTSO-E provides complete and daily data sets, which are not interrupted, this is very important for the time series analysis. This helps to make the forecasting models developed to be accurate and reliable.
2. **Clarity and Transparency:** The data from ENTSO-E is easily understandable and all the information is well explained. Detailed explanations of how the data is calculated can be found on their data views [page](#).
3. **Standardization and Harmonization:** The information is collected from various transmission system operators in Europe including ADMIE for [Greece](#). This makes the analysis easy and comparable across different countries due to standardization.
4. **Timeliness and Accessibility:** ENTSO-E offers data that is constantly being updated and easily retrievable and thus useful in the current and future research and analyses.

Therefore, by using the comprehensive, well-documented, and up-to-date data from ENTSO-E, this thesis guarantees the usage of high-quality data for the development of reliable and precise electricity load forecasting models. The selection of ENTSO-E as the data source mitigates the issue of limited data from ADMIE and thus improves the credibility of the study findings. This shows that the selection of data used in this thesis was done with a lot of care to ensure that only the most accurate data was used in the analysis of the forecasting methods used in this study.

Initially, a script in Python is meant to read, aggregate, and store electricity load data from several CSV files obtained from the ENTSO-E website. This process ensures that all the data that is to be included in the analysis is integrated into a single format, which is beneficial for analysis and forecasting.

This data spans from the beginning of the year 2021 to the end of the year 2023 to give a clear picture of the trends in electricity demand during this period. All the necessary data has been included in the dataset to help in the analysis of the electricity load patterns. The format of the dataset is following:

Time (EET/EEST)	Day-ahead Total Load Forecast [MW] - BZN GR	Actual Total Load [MW] - BZN GR
01.01.2021 00:00 - 01.01.2021 01:00	4756	4671
01.01.2021 01:00 - 01.01.2021 02:00	4556	4316
01.01.2021 02:00 - 01.01.2021 03:00	4415	4136
01.01.2021 03:00 - 01.01.2021 04:00	4215	3830
01.01.2021 04:00 - 01.01.2021 05:00	4045	3652
...	...	...

*Table 1: Initial table of consumption demand*

The column **“Time (EET/EEST)”** indicates the specific date and time for each recorded in the correct time zone for Greece, while the column **“Day-ahead Total Load Forecast [MW] - BZN|GR”** represents the day-ahead forecast of the total electricity load per market time unit within the Greek bidding zone. This forecast is generated at the latest two hours before the gate closure time of the day-ahead market or by 12:00 PM local time on the day before (D-1), if no specific gate closure time applies. The forecast aims to provide an estimate of the electricity demand based on historical load profiles for similar days, incorporating variables such as weather conditions, climate, and socioeconomic factors. It is intended for informational purposes and is subject to updates in the event of significant changes, defined as alterations of at least 10% of the total load forecast for any market time unit. This forecast is primarily produced by Transmission System Operators (TSOs) and Distribution System Operators (DSOs) using advanced meteorological data and other predictive analytics to anticipate the load. The column **“Actual Total Load [MW] - BZN|GR”** provides the actual total electricity load recorded per market time unit within the Greek bidding zone. The actual load is determined by summing the power generated by plants connected to both TSO and DSO networks, and then adjusting for the balance of imports and exports on interconnections with neighboring bidding zones and the power absorbed by energy storage resources. This information is published at most one hour (H+1) after the end of each operating period, ensuring timely and accurate reporting of actual electricity consumption.

The first step of the transformation of the dataset is to remove the second part after the dash ('-') from column **“Time (EET/EEST)”**, the reason is to ensure that only the relevant date information is retained, making the date format consistent across the dataset. After that, the type will be datetime and the format YYYY-MM-DD HH:MM:SS. Then, we delete column **“Day-ahead Total Load Forecast [MW] - BZN|GR”** because we need only actual load. Lastly, in case there is no 1-hour granularity, we resample per hour and forward fill the missing hours. In that way, we have transformed the load consumption data in the following way:

Time	Load
2021-01-01 00:00:00	4671
2021-01-01 01:00:00	4316
2021-01-01 02:00:00	4136
2021-01-01 03:00:00	3830
2021-01-01 04:00:00	3652
...	...

Table 2: Table of consumption after transformation

## 2.1.1 Weather Data Extraction and Preparation

The weather data were sourced from the “Open-Meteo” API using a robust script in Python.

```
import openmeteo_requests
import requests_cache
import pandas as pd
from retry_requests import retry

cities_dict = {"ATH": {"latitude": 37.9838, "longitude": 23.7278},
              "SKG": {"latitude": 40.6436, "longitude": 22.9309}}

# Setup the Open-Meteo API client with cache and retry on error
cache_session = requests_cache.CachedSession('.cache', expire_after = -1)
retry_session = retry(cache_session, retries = 5, backoff_factor = 0.2)
openmeteo = openmeteo_requests.Client(session = retry_session)

for i in cities_dict:
    print(cities_dict[i]["latitude"])
    params = {
        "latitude": cities_dict[i]["latitude"],
        "longitude": cities_dict[i]["longitude"],
        "start_date": "2021-01-01",
        "end_date": "2023-31-12",
        "hourly": ["temperature_2m", "relative_humidity_2m", "rain", "wind_speed_10m"],
        "timezone": "Europe/Athens"
    }
    responses = openmeteo.weather_api(url, params=params)
```

```

# Process first location. Add a for-loop for multiple locations or weather models
response = responses[0]
print(f"Coordinates {response.Latitude()}°N {response.Longitude()}°E")
print(f"Elevation {response.Elevation()} m asl")
print(f"Timezone {response.Timezone()} {response.TimezoneAbbreviation()}")
print(f"Timezone difference to GMT+0 {response.UtcOffsetSeconds()} s")

# Process hourly data. The order of variables needs to be the same as requested.
hourly = response.Hourly()
hourly_temperature_2m = hourly.Variables(0).ValuesAsNumpy()
hourly_relative_humidity_2m = hourly.Variables(1).ValuesAsNumpy()
hourly_rain = hourly.Variables(2).ValuesAsNumpy()
hourly_wind_speed_10m = hourly.Variables(3).ValuesAsNumpy()
hourly_data = {"date": pd.date_range(
    start = pd.to_datetime(hourly.Time(), unit = "s", utc = False),
    end = pd.to_datetime(hourly.TimeEnd(), unit = "s", utc = False),
    freq = pd.Timedelta(seconds = hourly.Interval()),
    inclusive = "left"
)}
hourly_data["temperature_2m"] = hourly_temperature_2m
hourly_data["relative_humidity_2m"] = hourly_relative_humidity_2m
hourly_data["rain"] = hourly_rain
hourly_data["wind_speed_10m"] = hourly_wind_speed_10m
hourly_dataframe = pd.DataFrame(data = hourly_data)

print(hourly_dataframe)

hourly_dataframe.to_excel(f'weather_data_{i}.xlsx', header=True, index=False)

```

The code shows how to get and clean historical weather data for several cities in Greece, particularly Athens (ATH) and Thessaloniki (SKG) using the Open-Meteo API. First, the required libraries are imported for the API interactions such as 'openmeteo\_requests', for caching the responses 'requests\_cache', for data manipulation 'pandas', and for handling the request failures 'retry\_requests'. This configuration allows the script to be prepared for handling API calls and be able to deal with possible errors.

A cached session is created using "requests\_cache" to store API responses locally, which helps to avoid redundant requests and improves performance. This session is configured to retry failed requests up to five times with an exponential backoff, enhancing the robustness of the data retrieval process. The "openmeteo\_requests.Client" is then initialized with this session, setting up the Open-Meteo API client to manage API interactions effectively.

The API endpoint and parameters are defined next, specifying the latitude and longitude for each city (Athens and Thessaloniki), the date range for the data (from January 1, 2021, to December 31, 2023), the desired weather variables, and the time zone (Europe/Athens). The parameters ensure that

the retrieved data is relevant and specific to the locations and time period of interest. The chosen variables include temperature at 2 meters, relative humidity at 2 meters, rain, and wind speed at 10 meters. Temperature is measured in degrees Celsius (°C), relative humidity in percentage (%), rain in millimeters (mm), and wind speed in kilometers per hour (km/h).

The script then extracts the hourly weather data from the response, converting each requested variable (temperature, relative humidity, rain, and wind speed) into a NumPy array for efficient handling. A pandas DataFrame is created to organize the hourly weather data, including a date range generated based on the start and end times of the data, with a frequency corresponding to the data's interval. The extracted weather variables are then added to the DataFrame, ensuring that the data is structured and ready for analysis.

Finally, the DataFrame containing the hourly weather data for each city is exported to an Excel file, saving it to the specified directory. This step ensures that the processed data is easily accessible for further analysis and integration with other datasets, such as electricity load forecasting models. This automated process ensures efficient retrieval, processing, and storage of historical weather data, facilitating robust and accurate forecasting analysis in this thesis.

Date	temperature_2 m	relative_humidity_2 m	rain	wind_speed_10m
2021-01-01 00:00:00	9.712	79.64074	0	7.771331
2021-01-01 01:00:00	9.262	82.36791	0	6.439876
2021-01-01 02:00:00	9.262	81.80262	0	6.12
2021-01-01 03:00:00	8.712	82.86843	0	6.409617
2021-01-01 04:00:00	9.012	78.71679	0	5.804825
...	...	...	...	...

Table 3: Weather Data example of Athens

## 2.1.2 Explanation for Merging Data and Importance of Temporal Features in Forecasting

After obtaining and cleaning the electricity load and weather data, the subsequent important step was to combine these two data sets in order to enable the prediction of electricity demand. Thus, we combined the data from various sources for the same dates and obtained a single dataset that contains the actual electricity load and weather conditions for the given time intervals. This approach helps in gathering all the necessary data in one place and therefore, provides a better understanding of the factors that affect electricity consumption.

Combining weather data with electricity load data is important for the following reasons. First of all, it enables the analysis of the effects of weather variables including temperature, humidity, rainfall and wind speed on electricity consumption. This integration is important in the creation of good predictive models that can help in determining the future electricity consumption patterns given the past

consumption rates and other environmental factors. The merged dataset is useful in that it gives a more complete picture of the interaction between weather and electricity usage.

Furthermore, the integration of time variables like day of the week and month is crucial in enhancing the accuracy of the demand forecasting. It is often observed that electricity consumption is not constant throughout the week because of the differences in the industrial, commercial, and residential activities between the weekdays and weekends. For instance, electricity consumption is normally low during the weekend because many companies and industries are not in operation. Thus, by flagging the weekends in the dataset, the model can take into consideration such changes and, therefore, provide a better estimate of the daily fluctuations in demand.

Likewise, identifying the month of the year in the dataset is crucial because seasonal changes affect the electricity consumption. Variations in temperature and daylight hours in the course of the year influence the heating, cooling and lighting requirements, which in turn cause variations in electricity consumption. This is because the month is incorporated as a feature in the model and this will help the model to capture these seasonal patterns of demand in the long run. This consideration of temporal factors is important in the generation of accurate and timely electricity load forecasts which are important in energy management and planning.

Combining weather and electricity load data with time features improves the performance of the models and also reveals the factors that affect electricity consumption. This holistic approach shows that data cleaning and data integration are critical to the production of accurate and reliable energy management forecasts.

date	Load	temperature_2m_ATH	relative_humidity_2m_ATH	rain_ATH	wind_speed_10m_ATH	temperature_2m_SKG	relative_humidity_2m_SKG	rain_SKG	wind_speed_10m_SKG	Weekend	Month
2021-01-01 00:00:00	4671	9.7119999	79.640739	0	7.7713308	6.2639999	88.253304	0	8.6400003	0	1
2021-01-01 01:00:00	4316	9.2620001	82.367912	0	6.4398761	7.7639999	81.037544	0	3.5999999	0	1
2021-01-01 02:00:00	4136	9.2620001	81.80262	0	6.1199999	7.9639997	81.064873	0	3.96	0	1
2021-01-01 03:00:00	3830	8.7119999	82.868431	0	6.4096174	6.6139998	84.36219	0	6.4799995	0	1

Table 4: Final table Format

## 2.2 Exploratory data analysis (EDA)

Exploratory Data Analysis (EDA) helps in understanding the characteristics of the data and the structure of the dataset. This process is important in order to understand the key characteristics of the data including the means, spread, and shape of the data which are the basis of constructing good and reliable forecasting models.

## 2.2.1 Key Insights from Descriptive Statistics

**Central Tendency:** The mean values obtained from the dataset help in establishing the normal conditions during the time of analysis. The electricity load per year is about 5,624.03 MW, which is useful for general consumption where trends or changes may be observed. For weather conditions, the average temperature in Athens (18.33°C) and Thessaloniki (16.04°C) which are the average conditions in the course of the year and therefore, give an idea of the normal weather that may affect the demand for electricity.

**Dispersion and Variability:** The standard deviation values reveal the degree of fluctuation within the dataset. The electricity load's standard deviation is significant at 1,233.39 MW, suggesting substantial variability in consumption. This variability underscores the importance of developing forecasting models that can effectively accommodate these fluctuations. Similarly, the temperature variability in Athens (8.03°C) and Thessaloniki (8.52°C) indicates considerable changes in weather conditions, which are crucial factors in predicting energy usage.

**Range and Extremes:** Analyzing the minimum and maximum values provides insights into the dataset's range and extremes. The electricity load varies widely, from a minimum of 408 MW to a maximum of 10,967 MW, highlighting the need to manage both low and high demand periods effectively. The temperature range in Athens (-2.04°C to 43.81°C) and Thessaloniki (-5.44°C to 40.46°C) reflects the diverse climatic conditions that must be factored into the forecasting models.

**Quartiles and Distribution:** The quartiles offer a detailed view of the data distribution, helping to understand the spread and potential skewness. The lower quartile (Q1: 4,689 MW) and upper quartile (Q3: 6,345.25 MW) of electricity load reveal a substantial interquartile range, indicating a wide distribution of demand values. The median temperatures in Athens (17.66°C) and Thessaloniki (15.46°C) suggest that the central tendency is around moderate climatic conditions, which is vital for understanding seasonal variations in electricity demand.

## 2.2.2 Key Insights from Heatmap Visual

A heatmap is a graphical representation of data in the form of a matrix where the intensity of the color is used to represent the value of the data point. Every cell in the matrix is painted in a certain color depending on the value it contains, which helps to distinguish certain patterns, tendencies and interconnections between the variables. In the case of exploratory data analysis, a heatmap is especially helpful in visualizing the relationship between several variables and easily distinguish between positive and negative correlations. The values are between -1 which is a perfect negative correlation to +1 which is a perfect positive correlation.

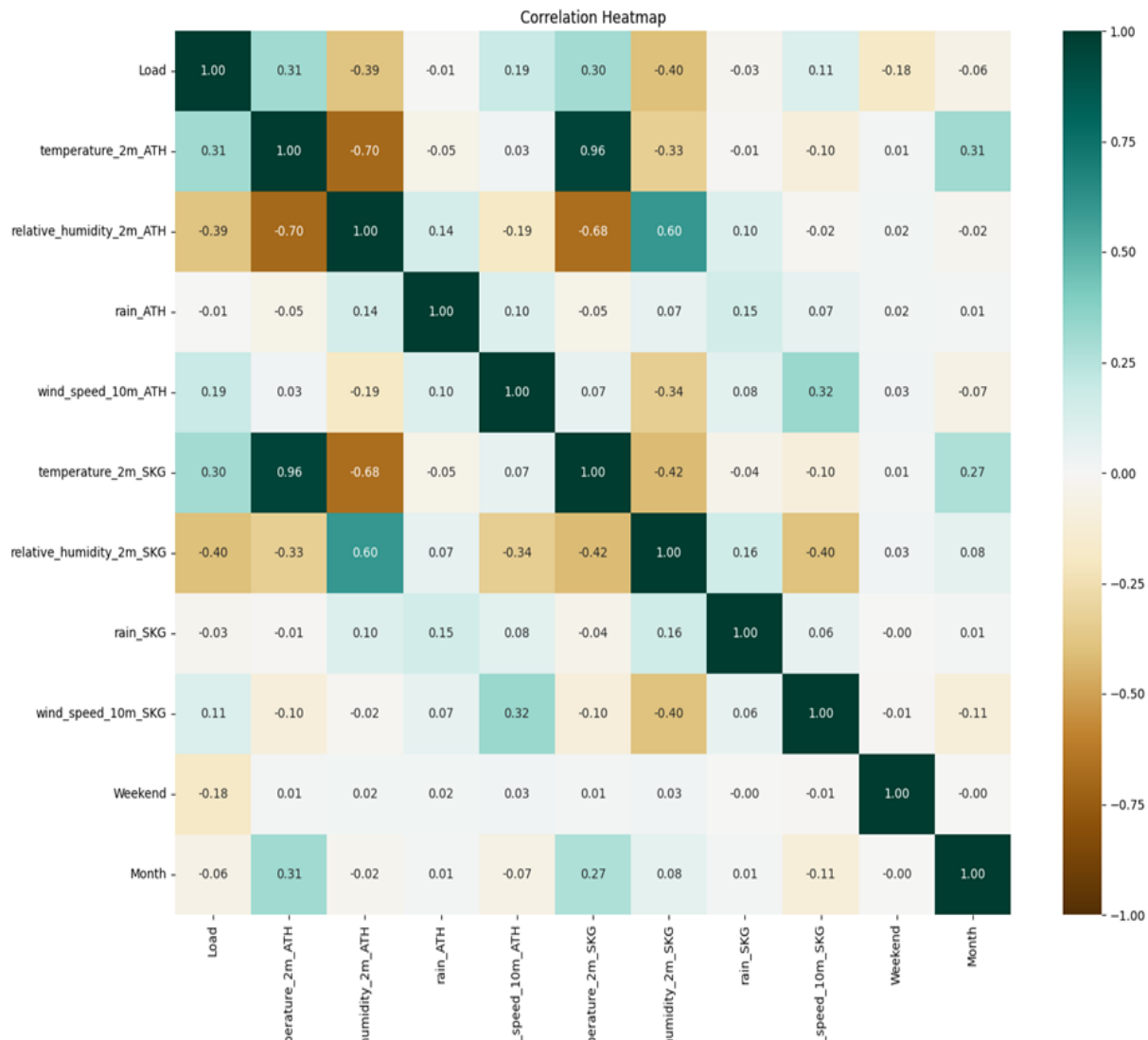


Figure 17: Heatmap Visual

Interestingly, there is a moderate positive correlation between the temperature in Athens (0.31) and Thessaloniki (0.30) with electricity load, suggesting that higher temperatures in these cities lead to an increase in electricity consumption. Similarly, wind speed in Athens shows a weak positive correlation (0.19) with electricity load. On the other hand, relative humidity in both Athens (-0.39) and Thessaloniki (-0.40) demonstrates moderate negative correlations with electricity load, indicating that higher humidity levels are associated with reduced electricity usage. Additionally, the weekend variable exhibits a weak negative correlation (-0.18), implying that electricity load tends to be lower on weekends, while the month variable shows a very weak negative correlation (-0.06), suggesting minimal seasonal variation.

The correlation heatmap also reveals significant temperature correlations between Athens and Thessaloniki, with a very strong positive correlation (0.96), indicating that temperature changes in one city



closely mirror those in the other. Furthermore, there is a moderate positive correlation (0.60) between relative humidity in Athens and Thessaloniki, and a moderate positive correlation (0.32) between wind speeds in these two cities. Rainfall in both Athens and Thessaloniki shows very weak correlations with other variables, suggesting that rain does not significantly impact electricity load or other weather conditions. Temporal factors such as weekends and months show very weak correlations with most variables, highlighting that the distinction between weekends and weekdays has a minor influence on the variables, while seasonal temperature variations are more pronounced.

In conclusion, the heatmap shows the main dependencies between the weather and the electricity load, with temperature and humidity playing the most important role. Temperature in Athens and Thessaloniki has a positive relationship with electricity load while humidity has a negative relationship with load. Weekend is another factor that slightly affects the electricity load, while the seasonal variations have a very little direct effect. Knowing these relationships can help in creating better models for the estimation of the electricity load depending on the weather and time, which will in turn enhance the management of energy and its consumption.

### 2.2.3 Key Insights from Scatter Plot

A scatter plot is a type of data visualization that displays values for typically two variables for a set of data. The data points are plotted on a two-dimensional graph, where each point represents an observation in the dataset. The position of each point on the graph is determined by the values of the two variables being compared. Scatter plots are particularly useful for identifying relationships, trends, and patterns between the variables. They can highlight correlations, clusters, and outliers, providing a visual way to understand complex data relationships.

The following scatter plots show the U-shaped relationship between temperature and energy load in both Athens and Thessaloniki.

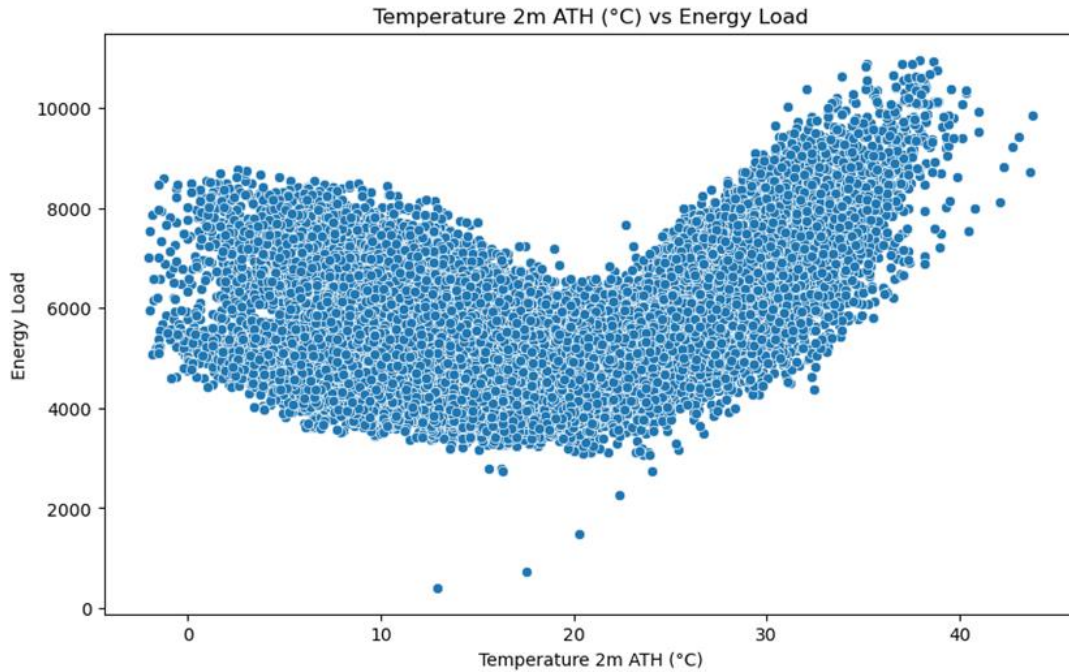


Figure 18: Scatter Plot Temperature 2m ATH (°C)

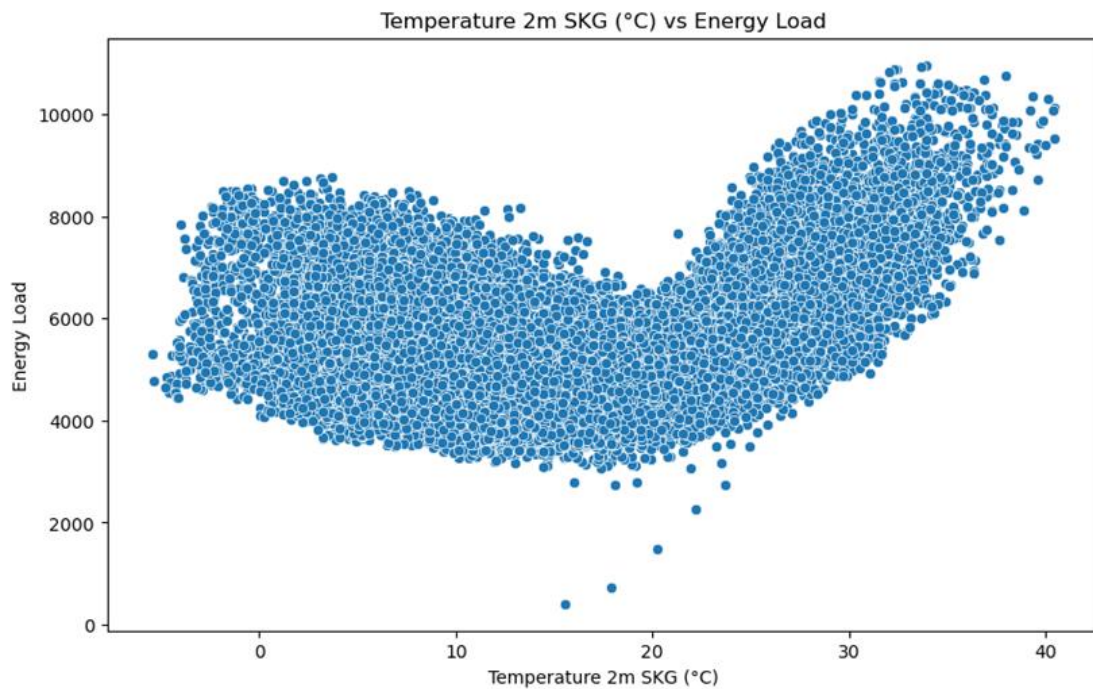


Figure 19: Scatter Plot Temperature 2m SKG (°C)

The scatter plots give a clear picture of the U-shaped relationship between temperature and energy load in both Athens and Thessaloniki. A U-shaped pattern in a scatter plot means that one variable decreases to a certain point and then increases again while the other variable also decreases to a certain point and then increases again. In the case of a U-shaped pattern, the dependent variable is usually lower

in the middle range of the independent variable and higher at both ends. This implies that the effect of the independent variable is relatively low at the middle level and high at the low and high levels of the independent variable.

In Athens, the U-shaped relationship is characterized by a sharp rise in energy consumption at both low and high temperatures. In particular, when the temperature is below 10°C, the energy load increases sharply because of the increased demand for heating. On the other hand, as temperatures go up to 30°C, energy demand rises again due to the need for cooling. This pattern shows that the energy consumption is greatly influenced by the weather conditions with the energy load rising to 10,000MW during the extreme weather conditions. The energy consumption pattern in relation to temperature shows that energy consumption is relatively low in the temperature range of 15°C to 25°C where neither heating nor cooling is required to a large extent.

Likewise, the scatter plot for Thessaloniki also depicts the U-shaped pattern similar to the one observed in Athens, which shows high energy consumption at both ends of the temperature scale. This consistency points to a regional pattern where extreme temperatures have a direct impact on energy consumption, requiring more heating or cooling. The energy load in Thessaloniki also decreases within the moderate temperature range, which also points to a period of low energy consumption. However, there is a clear trend of variability in energy load at the two ends with some of the values going up to 10,000MW which shows the impact of temperature on energy demand. The slight variations in the data spread at lower temperatures in Thessaloniki could be attributed to differences in energy consumption or temperature responsiveness across the region due to climatic or behavioral factors.

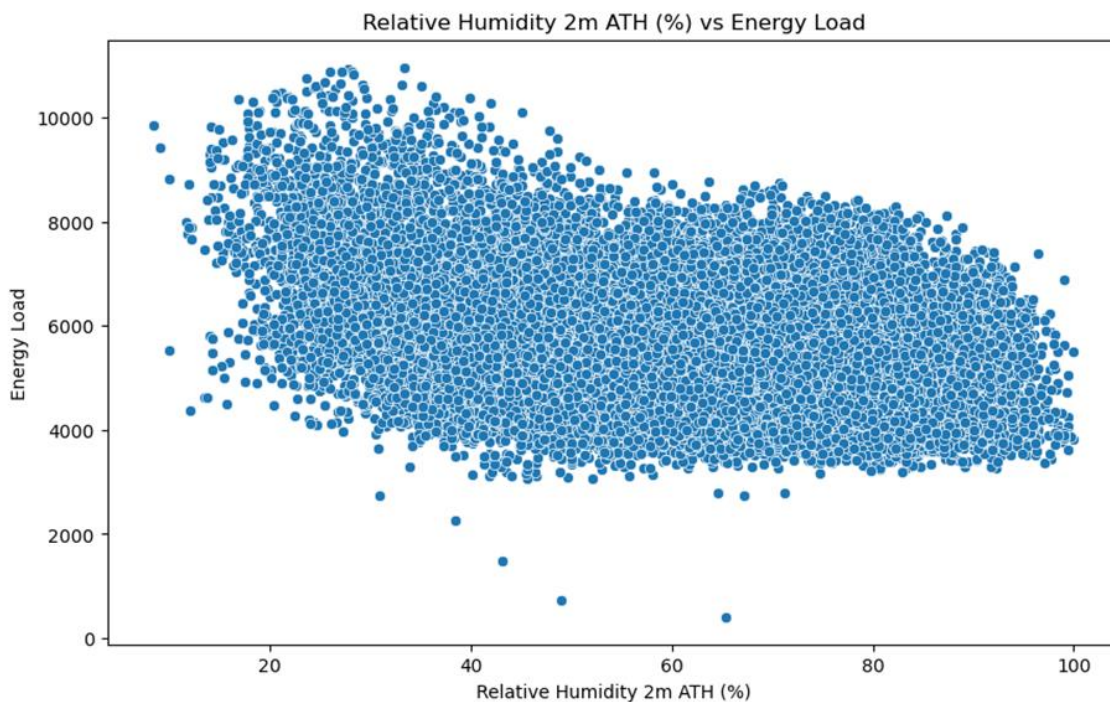


Figure 20: Scatter Plot Humidity 2m ATH (%)

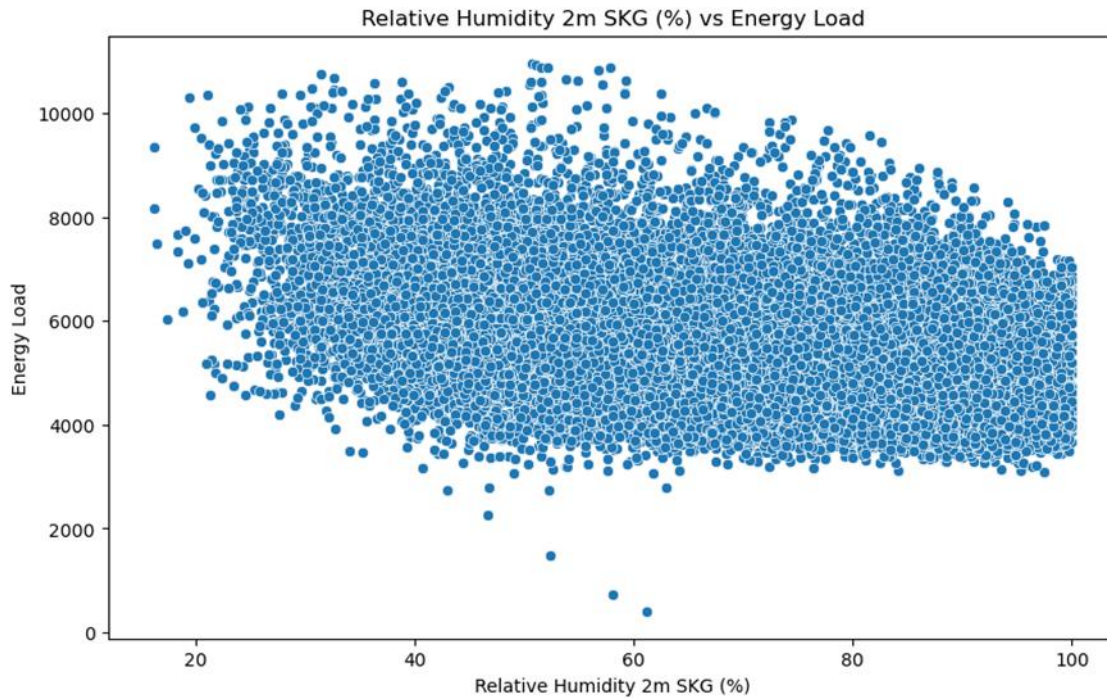


Figure 21: Scatter Plot Humidity 2m SKG (%)

While the temperature vs. energy load scatter plots depicted a U-shaped pattern, the scatter plots for relative humidity do not have a clear U-shaped pattern. Rather, it is more of a linear or slightly random pattern. The data points indicate that there is an inverse relationship between relative humidity and energy load. It is observed that as the relative humidity increases, the energy load decreases. This suggests that higher humidity levels could be linked to lower energy consumption, possibly because there is less requirement for heating and cooling in humid climates. At low humidity levels, the energy load values are quite dispersed and can be low, moderate, or high depending on the level of humidity. This suggests that energy demand is more volatile when the air is dry, which could be attributed to the fact that different temperatures may be needed to regulate the temperature indoors. When the humidity level is 60% and above, the energy load fluctuation is less and the data points are clustered closely together. This means that in the humid conditions, the energy load does not vary as much as it does in the dry conditions. The data points are clustered between 20% and 80% humidity, which means that these levels of humidity are typical for Athens. The energy load in this range is relatively stable, which indicates that the energy consumption is relatively stable under normal humidity conditions.

As in the case of Athens, Thessaloniki also shows a negative relationship between relative humidity and energy load. The energy load reduces with the increase in relative humidity, which supports the trend identified in the case of Athens. The energy load for Thessaloniki also exhibits less variability at higher humidity levels, as is the case with Athens. The data points are more clustered, which means that energy demand is more predictable in these circumstances. At low humidity levels, energy load is quite volatile, which means that people use different amounts of energy when the air is dry. This implies that other factors such as temperature may be the key determinants of energy demand in these conditions. The energy load values are quite dispersed in the range of 40-60% humidity, which means that the energy



consumption patterns are quite different in the moderate humidity conditions. This could be attributed to differences in weather or any other factor that may have been in play.

## 2.2.4 Key Insights from Time Series Data

The analysis of the electricity load data in the form of time series allows to reveal the patterns and tendencies that govern the energy consumption in the course of time. Breaking down the data into its original, trend, seasonal, and residual components, we can determine the key factors that affect the changes in electricity consumption. This detailed examination reveals the cyclical pattern of energy usage, the effects of long-term trends, and the importance of short-term deviations, thus providing a clear understanding of how all the factors interact to shape the energy load. These are important in the formulation of sound energy management and forecasting plans to help in the proper utilization of energy resources and to guarantee adequate energy supply for the year.

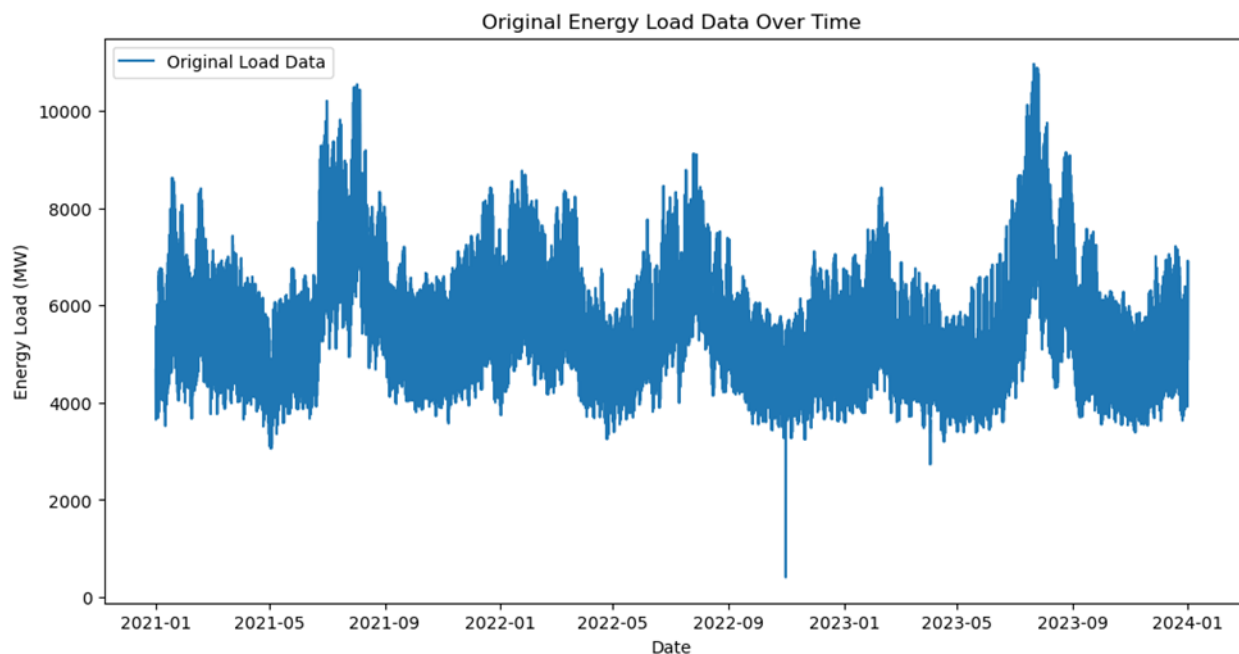


Figure 22: Original Energy Load Data Over Time

The graph provided is called “Original Energy Load Data Over Time” and covers the period from January 2021 to December 2023, which shows the changes in energy consumption during different seasons and years. The most apparent feature is the cyclical fluctuations that are observed, which are related to the fluctuations in electricity consumption due to seasonal factors. For example, there is a high energy consumption in the winter because of heating and another spike in the summer because of cooling. This seasonality shows that the electricity demand is very much dependent on temperature variations and therefore it is crucial to prepare for the high demand during the hot or cold seasons.

Also, the graph shows that energy consumption is quite volatile from one year to another. The highest peaks are seen at the middle of 2021 and at the end of 2023, which may indicate the time of the

highest energy consumption. The data also reveal a general trend of decreasing baseline energy load over the years which could be due to increased efficiency in energy use or changes in energy use patterns. This trend underlines the need to track the energy demand and adjust the management approaches to the changes in the demand, both planned and unplanned. This time series data is very important in the analysis of energy consumption and in decision making that will help in the provision of energy supply that is stable and sustainable.

To gain a more nuanced understanding of these variations in energy demand, it is essential to delve deeper into the underlying trends and patterns captured in the data. The following analysis, utilizing advanced trend line techniques such as the Savitzky-Golay filter and polynomial regression, will help elucidate these patterns further. By examining these trend lines, we can identify both short-term fluctuations and long-term trends, offering valuable insights into the factors driving electricity consumption.

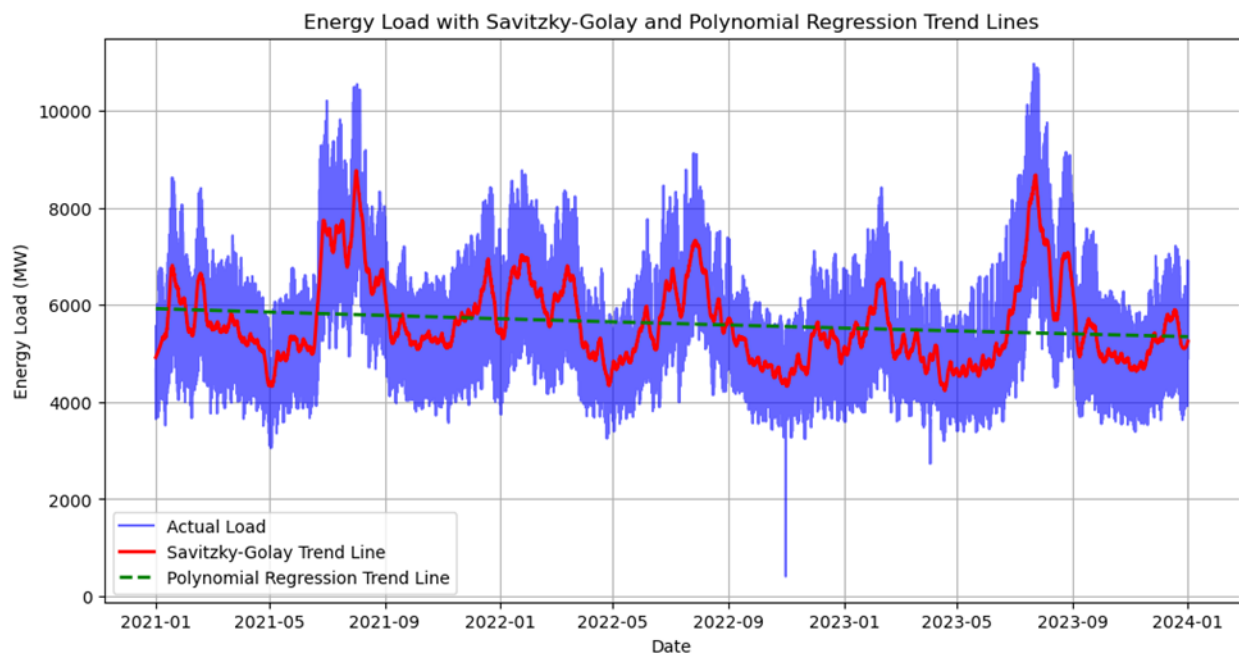


Figure 23: Energy Load with Savitzky-Golay and Polynomial Regression Trend Lines

The application of Savitzky-Golay and Polynomial Regression for the combined trend analysis of energy load data is useful in identifying the patterns and fluctuations in the data. The Savitzky-Golay filter, depicted by the red line in the graph, is the best at removing short-term oscillations while keeping the overall shape and trend of the data intact. This feature makes it a perfect tool to show the periodic rise and fall patterns that are evident in the course of a year. For example, the filter clearly shows the seasonal trends of energy consumption with the peaks in the winter and summer seasons due to the increased usage of heating and cooling systems, respectively. These patterns show that the energy consumption is influenced by the seasons and this means that the energy providers need to prepare for the increased demand that comes with the particular seasons.

The green dashed line represents the Polynomial Regression trend line which gives a wider view of the overall trend of the data. Thus, the second-degree polynomial that is fitted to the energy load data

captures this overall trend of decreasing baseline energy consumption over the period of observation. This decline may be attributed to increased efficiency in energy use, which could have been brought about by the use of energy efficient technologies and measures. The Polynomial Regression trend line helps to eliminate the short-term fluctuations and gives a better understanding of the general tendencies of energy consumption. This approach is important for strategic planning because it allows to understand the causes of the long-term trend of energy consumption decrease and, thus, to define the ways of energy resources optimization and their further effective use.

Building on this analysis, the residuals graph offers a detailed look at the deviations from the expected energy load once these long-term trends have been accounted for.

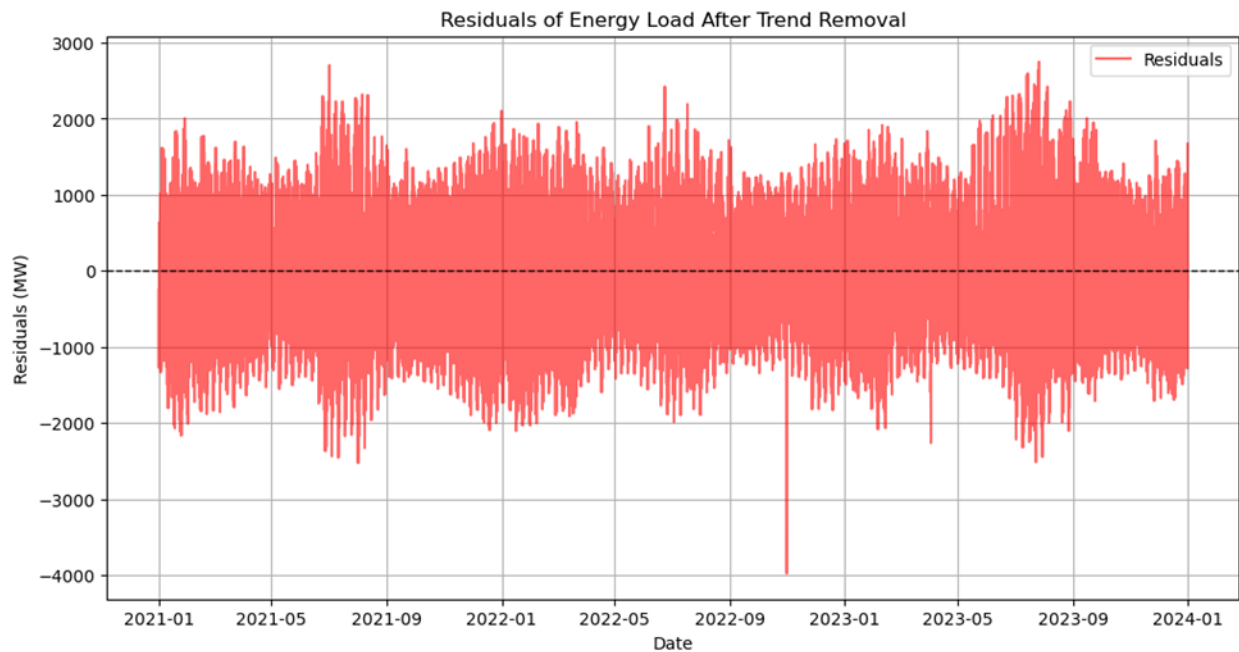


Figure 24: Residuals of Energy Load After Trend Removal

The above graph gives a more detailed view of the deviations from the expected energy load, with the overall trend line having been removed. This visual representation is concerned with the variations in energy demand that cannot be attributed to the overall increase or decrease observed in the trend analysis.

On the graph, the x-axis represents the dates from January 2021 to December 2023, while the y-axis measures the residuals in megawatts (MW). The residuals are depicted in red, with a dashed line at  $y=0$  indicating the baseline. Values above this line represent positive deviations from the trend, and values below it represent negative deviations. The consistent range of fluctuations around the zero line suggests that while there are significant deviations from the trend, these deviations are fairly regular. This consistency implies that the trend model effectively captures the cyclical variations in the data.

Noteworthy anomalies are evident, particularly in late 2021 and mid-2023, where residuals sharply increase or decrease. These spikes likely indicate extraordinary events or anomalies such as sudden changes in temperature, shifts in economic activity, or policy changes impacting electricity consumption. The residuals also exhibit a noticeable seasonal pattern, with higher variability during the summer and

winter months. This aligns with periods of extreme weather conditions when the demand for electricity for heating or cooling typically varies more widely

In the previous visual, we examined the trend lines derived from Savitzky-Golay filtering and polynomial regression, which highlighted the general movement of energy load over time and smoothed out short-term fluctuations to reveal long-term trends. This analysis provided a clear picture of the overall direction of electricity demand and identified significant peaks and troughs that correlate with seasonal changes.

The residuals graph complements this analysis by focusing on the deviations from these trends. While the trend lines depict the overarching patterns in the data, the residuals expose the finer details, such as irregular fluctuations and anomalies that are not explained by the trend alone. Together, these visuals offer a comprehensive view of the energy load dynamics: the trend lines help in understanding the broad, long-term movements, while the residuals highlight the short-term variations and unexpected changes. This combined approach enables a deeper understanding of the factors influencing energy consumption and is crucial for developing more accurate forecasting models and effective energy management strategies.

## 2.3 Architecting Effective Time Series Models for Energy Forecasting

First of all, it was important to define the forecasting task as a supervised learning task. In this context, the task is to predict a target variable  $y$  given a set of input features  $x$ , and unlike most supervised learning problems, time series forecasting requires taking into account the temporal dimension, which means that it is necessary to take into account not only the relationships between external variables but also the time series patterns and trends. This integration of the time component is important in the context of forecasting since it enables one to make precise predictions.

In our case, the target variable represents  $y$  the future values of the electrical load. The input features include not only past values of the electrical load but also relevant weather and calendar data. To effectively construct our forecasting models, three key concepts must be defined: forecasting length, past values length, and the sliding/rolling window method.

For the purpose of this study that is concerned with short-term forecasting, the forecasting length or the output length is set at 24 hours. This means that our model will be able to predict the electrical load in MW every hour for the next day in detail. This way, we are able to model daily patterns and forecast short-term fluctuations in energy consumption effectively.

Below, we will focus exclusively on the 24-hour input length. This decision is based on our findings that the 24-hour input length provides sufficient historical context for accurate predictions while also reducing computational complexity. This balance between accuracy and efficiency makes the 24-hour input length the most effective duration of past data to include, significantly enhancing the predictive capability of the model.



The sliding or rolling window method is a crucial technique for handling time series data in model training. This method involves creating a fixed-length input array of  $n$  past values that is progressively shifted forward to forecast the next set of future values. Specifically, in our implementation, the sliding window approach creates overlapping input and output pairs, where each input array contains  $n$  historical data points, and the corresponding output array holds the subsequent 24-hour forecast. This method effectively transforms the dataset into a structured format suitable for neural network algorithms, facilitating the model's ability to learn from historical patterns.

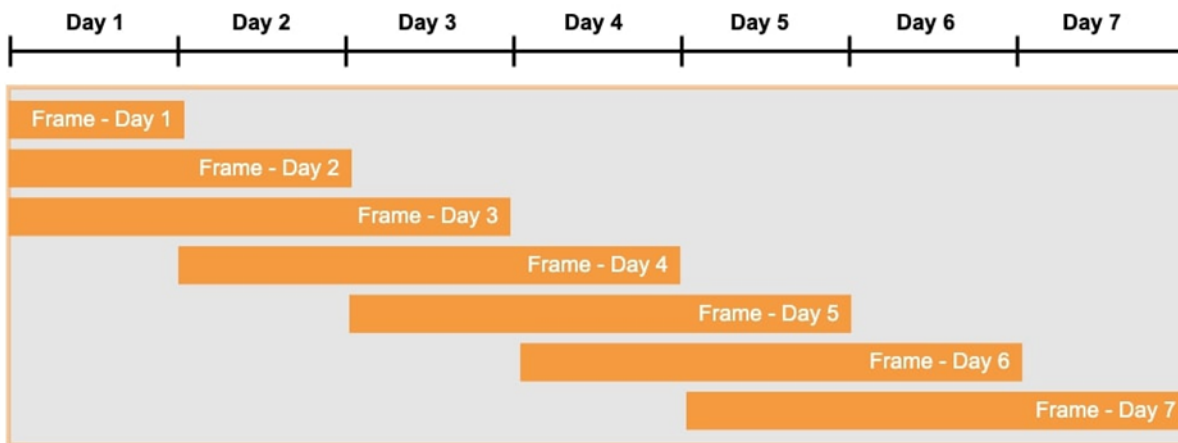


Figure 25: Moving Window Function Example [\(Source\)](#)

To prepare the data for modeling, we apply the Min-Max scaler, which normalizes the data to a specified range, typically  $[0, 1]$ . This step is crucial for several reasons. First, it ensures that all features contribute equally to the learning process, preventing the model from being dominated by variables with larger scales. In the context of electrical load forecasting, where we integrate diverse data types like weather conditions and past load values, Min-Max scaling ensures that the numerical differences do not skew the learning algorithm's focus. Below is the formula:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

The key components of this transformation include:

- $x$ : The original data point.
- $x_{min}$ : The minimum value in the dataset for the feature being scaled.
- $x_{max}$ : The maximum value in the dataset for the feature being scaled.
- $x_{scaled}$ : The normalized value after applying the Min-Max scaling.

Furthermore, applying the Min-Max scaler speeds up the learning process and leads to faster convergence. This is because the learning algorithms, especially neural networks, perform better when the input data is normalized. By bringing all features into the same scale, we reduce the complexity of the optimization problem, allowing the learning algorithm to find the optimal weights more efficiently. This is

particularly beneficial for our time series data, which has varying ranges across different variables and time steps. Faster convergence not only reduces training time but also helps in achieving better generalization, thus improving the model's predictive accuracy on unseen data.

Furthermore, applying the Min-Max scaler speeds up the learning process and leads to faster convergence. This is because the learning algorithms, especially neural networks, perform better when the input data is normalized. By bringing all features into the same scale, we reduce the complexity of the optimization problem, allowing the learning algorithm to find the optimal weights more efficiently. This is particularly beneficial for our time series data, which has varying ranges across different variables and time steps. Faster convergence not only reduces training time but also helps in achieving better generalization, thus improving the model's predictive accuracy on unseen data.

## 2.4 Feed Forward Model

In this section, we chose three specific cases for constructing and assessing feed-forward models, each of which is aimed at solving different problems of short-term electricity load forecasting. These scenarios were selected to investigate the performance of the feed-forward neural network with different input variables and prediction steps, thus offering a detailed assessment of the network's forecasting potential.

The first scenario is a straightforward yet efficient way of making the prediction where the next hour load is predicted based on the previous 24 hours load. This model uses a one hour lagged data, where the data is shifted one hour for each observation. Thus, we guarantee that the model will learn the short-term temporal dependencies in the load data. The input of the last 24 hours of data is used in the model to capture the most recent patterns and trends that are useful for predicting the next hour, thus making it suitable for very short-term forecasting.

The second scenario also uses the past 24 hours of load data but the prediction is made for the next 48 hours. In this model, the objective is to predict the load 48 hours in advance, which gives a longer view of the load forecast. This approach is especially helpful when the most recent data is not available or is not very relevant, and we have to make our predictions based on the data that is somewhat outdated. Thus, training the model with a 48-hour forecast horizon allows for the use of data that is at most two days old to predict the load for the next hour, which is useful for real-world applications of the model.

The third scenario incorporates high correlation features such as temperature and wind speed from two major cities, Athens and Thessaloniki, in addition to the load data. By including these exogenous variables, the model can capture the influence of weather conditions on electricity demand. This comprehensive approach allows the model to leverage the strong correlations between weather variables and electricity load, thereby improving its predictive accuracy. The inclusion of multiple features provides a holistic view of the factors affecting load patterns, making this model more versatile and robust in handling various external influences.

All these scenarios use feed-forward neural network with different structures, activation functions and methods of regularization. Thus, by changing these parameters and analyzing the results of the models' performance, we will try to determine the best settings for load forecasting. The outcomes of

these models will be useful in identifying the best practices for short-term electricity load prediction, which will help in the enhancement of energy management systems.

In the following section, we will discuss the code and the output for each of the cases. This analysis will also involve a thorough assessment of the model structure, training procedure, and evaluation criteria. We will also plot the training and validation loss and the actual and predicted load to determine the performance of each model. This will enable the author to identify the various merits and demerits of each of the scenarios and enable the author to make the right decision on the best approach to use in electricity load forecasting.

## 2.4.1 Detailed Explanation of Feed-Forward Model Code and Implementation

Before training the models, it is crucial to prepare and scale the data appropriately. The dataset is first loaded and preprocessed, including setting the index to the date column and dropping the original date column. Three scenarios are defined for different model configurations and purposes. For each scenario, a set of features is selected and past values are generated using a sliding window approach, which ensures that the model has a sequence of past observations as input for making predictions.

```
# Load and preprocess data

file_path = "data_with_weekend_and_month.xlsx"
data = pd.read_excel(file_path)
data.index = data['date']
data = data.drop('date', axis=1)
```

- **Scenario 1** focuses on predicting the load one hour ahead using the past 24 hours of load data.

```
features_request = {"window": [24]}
build_df = build_features(data['Load'], features_request, target_lag=1, nclude_tzero=False)
X1 = build_df.drop(columns=['Target_Tplus1']).values
y1 = build_df['Target_Tplus1'].values.reshape(-1, 1)
```

- **Scenario 2** extends this approach to a forecasting horizon of 48 hours, still using the past 24 hours of load data.

```

features_request = {"window": 24}
build_df = build_features(data["Load"], features_request, target_lag=48, nclude_tzero=False)
X1 = build_df.drop(columns=["Target_Tplus1"]).values
y1 = build_df["Target_Tplus1"].values.reshape(-1, 1)

```

- **Scenario 3** uses high-correlation features along with the load data to predict one hour ahead with a 48-hour horizon

```

high_corr_features = ['Load', 'temperature_2m_ATH', 'temperature_2m_SKG', 'wind_speed_10m_ATH',
'wind_speed_10m_SKG']
input_feats = []
for feature in high_corr_features:
    features_request = {"window": 24}
    temp_df = build_features(data[feature], features_request, target_lag=48, include_tzero=False)
    if feature == 'Load':
        target_col = temp_df["Target_Tplus48"]
        temp_df = temp_df.drop("Target_Tplus48", axis=1)
        temp_df.columns = [feature + '-' + str(i) for i in range(24, 0, -1)]
        input_feats.append(temp_df)
processed_data = pd.concat(input_feats, axis=1)
processed_data["Target_Tplus48"] = target_col
X3 = processed_data.drop(columns=["Target_Tplus48"]).values
y3 = processed_data["Target_Tplus48"].values.reshape(-1, 1)

```

The input features and target variable are then normalized to the range of [0,1] using the MinMaxScaler from the scikit-learn library to make sure that all features are equally important to the model and to help the gradient descent optimization process.

In order to determine the best model configuration, a grid search is performed over a set of hyperparameters. The grid search explores combinations of hidden layer architectures, dropout rates, learning rates, and batch sizes. The “product” function from the Itertools library is used to generate all possible combinations of these hyperparameters. For each combination, the model is trained and evaluated on the validation set, and the results are stored. The model with the lowest validation RMSE (Root Mean Squared Error) is selected as the best model. This exhaustive search helps in identifying the most effective model architecture and training parameters, ensuring that the model is well-optimized and performs robustly on unseen data.

```

def train_and_evaluate_model(X, y, scenario_name, epochs=100):
    input_scaler = MinMaxScaler()
    output_scaler = MinMaxScaler()

    X_scaled = input_scaler.fit_transform(X)
    y_scaled = output_scaler.fit_transform(y)

```

```

split_index = int(len(X_scaled) * 0.8)
X_train, X_val = X_scaled[:split_index], X_scaled[split_index:]
y_train, y_val = y_scaled[:split_index], y_scaled[split_index:]

# Hyperparameter combinations
hidden_layers_options = [[64, 32], [128, 64], [256, 128, 64]]
dropout_rate_options = [0.1, 0.01, 0.05]
learning_rate_options = [0.001, 0.005, 0.01, 0.1]
batch_size_options = [120, 240, 360]

# Grid search
results = []
for hidden_layers, dropout_rate, learning_rate, batch_size in product(hidden_layers_options,
dropout_rate_options, learning_rate_options, batch_size_options):
    model = create_model(X_train.shape[1], hidden_layers, dropout_rate, learning_rate)
    es_callback = EarlyStopping(monitor='val_loss', patience=5, verbose=0)
    history = model.fit(X_train, y_train,
                        epochs=epochs, batch_size=batch_size,
                        shuffle=True, validation_split=0.20,
                        verbose=0, callbacks=[es_callback])

    val_predictions = model.predict(X_val)
    val_predictions = output_scaler.inverse_transform(val_predictions)
    y_val_inv = output_scaler.inverse_transform(y_val)

    val_score = mean_squared_error(y_val_inv, val_predictions, squared=False) # RMSE
    results.append((hidden_layers, dropout_rate, learning_rate, batch_size, val_score))

best_params = min(results, key=lambda x: x[-1])
print(f'{scenario_name} - Best parameters: hidden_layers={best_params[0]}, dropout_rate={best_params[1]},
learning_rate={best_params[2]}, batch_size={best_params[3]}, val_score={best_params[4]:.2f}')

best_model = create_model(X_train.shape[1], best_params[0], best_params[1], best_params[2])
# Save model summary to a text file
with open(f'{scenario_name}_model_summary.txt', 'w', encoding='utf-8') as f:
    best_model.summary(print_fn=lambda x: f.write(x + '\n'))

history = best_model.fit(X_train, y_train,
                        epochs=epochs,
                        batch_size=best_params[3],
                        shuffle=True, validation_split=0.20,
                        verbose=2, callbacks=[es_callback])

predictions_train = best_model.predict(X_train)
predictions_val = best_model.predict(X_val)

predictions_train = output_scaler.inverse_transform(predictions_train)
y_train = output_scaler.inverse_transform(y_train)

```

```

predictions_val = output_scaler.inverse_transform(predictions_val)
y_val = output_scaler.inverse_transform(y_val)

train_rmse = mean_squared_error(y_train, predictions_train, squared=False)
val_rmse = mean_squared_error(y_val, predictions_val, squared=False)
train_mae = mean_absolute_error(y_train, predictions_train)
val_mae = mean_absolute_error(y_val, predictions_val)
train_r2 = r2_score(y_train, predictions_train)
val_r2 = r2_score(y_val, predictions_val)

print(f'{scenario_name} - Train RMSE: {train_rmse:.2f}')
print(f'{scenario_name} - Validation RMSE: {val_rmse:.2f}')
print(f'{scenario_name} - Train MAE: {train_mae:.2f}')
print(f'{scenario_name} - Validation MAE: {val_mae:.2f}')
print(f'{scenario_name} - Train R2: {train_r2:.2f}')
print(f'{scenario_name} - Validation R2: {val_r2:.2f}')

return history, y_train, predictions_train, y_val, predictions_val

```

The best model, as determined by the grid search, is then trained using the entire training set and validated against the validation set. The EarlyStopping callback is used to halt training early if the validation loss does not improve for a specified number of epochs, preventing overfitting and saving computational resources. During training, both training and validation loss are monitored, and the best model weights are saved for evaluation. Once training is complete, the model's predictions are transformed back to the original scale using the inverse transformation of the output scaler, and the RMSE is calculated for both training and validation sets to assess the model's performance.

The process begins by defining a function, 'create\_model', which constructs and compiles a feed-forward neural network model. This function accepts several parameters including the number of input features ('input\_dim'), the architecture of hidden layers ('hidden\_layers'), the dropout rate (dropout\_rate), and the learning rate ('learning\_rate'). The model is built using the Sequential API from TensorFlow's Keras library. Each hidden layer is followed by a dropout layer to mitigate overfitting by randomly setting a fraction of input units to zero at each update during training time, which helps prevent overfitting. The model ends with a Dense layer with a single unit to predict the target variable. The Adam optimizer is chosen for its efficiency and ability to handle sparse gradients on noisy problems, and the mean squared error (MSE) is used as the loss function for regression tasks.

```

def create_model(input_dim, hidden_layers, dropout_rate, learning_rate):
    model = Sequential()
    model.add(Dense(hidden_layers[0], input_dim=input_dim, activation='relu'))
    model.add(Dropout(dropout_rate))
    for units in hidden_layers[1:]: model.add(Dense(units,
        activation='relu')) model.add(Dropout(dropout_rate))
    model.add(Dense(1))
    optimizer = Adam(learning_rate=learning_rate) model.compile(optimizer=optimizer, loss='mse',
        metrics=['mean_squared_error'])
    return model

```

Finally, the results of the models are visualized. The training and validation loss for each scenario are plotted to show the convergence behavior and to check for overfitting or underfitting. Additionally, the actual versus predicted load values are plotted for both training and validation sets for each scenario. This visualization helps in understanding how well the model captures the patterns in the data and where it might be making errors. By comparing the performance across different scenarios, insights can be drawn about the effectiveness of different feature sets and model configurations in forecasting the electricity load.

```
plt.figure(figsize=(14, 8))
plt.subplot(3, 1, 1)

pd.DataFrame(history1.history)[['loss', 'val_loss']].plot(ax=plt.gca(), title='Scenario 1: Training and Validation Loss') plt.subplot(3, 1, 2)

pd.DataFrame(history2.history)[['loss', 'val_loss']].plot(ax=plt.gca(), title='Scenario 2: Training and Validation Loss') plt.subplot(3, 1, 3)

pd.DataFrame(history3.history)[['loss', 'val_loss']].plot(ax=plt.gca(),title='Scenario 3: Training and Validation Loss') plt.tight_layout()
```

## 2.4.2 Comparative Analysis of Feed-Forward Model Scenarios and Performance Evaluation

The above three scenarios analysis of the feed-forward neural network models in electricity load forecasting provide useful information on the models' behavior and characteristics. The scenarios are based on different considerations for implementation and the impact that each has on the forecast quality and characteristics of the model's ability to generalize.

Performance Metric	Scenario 1	Scenario 2	Scenario 3
Train RMSE	187.24	618.14	559.11
Validation RMSE	200.65	649.02	594.71
Train MAE	140.95	472.04	424.83
Validation MAE	151.09	472.19	462.88
Train $R^2$	0.98	0.73	0.78
Validation $R^2$	0.98	0.78	0.81

Table 5: Performance Metrics for Different Scenarios

It is observed in Table 3 that Scenario 1 performed the best, with the lowest train and validation RMSE being 187.24 and 200.65, respectively. The given scenario employed a 24-hour sliding window for predicting the load of the next hour, which appears to be an optimal strategy for short-term forecasting. The best parameters for this setting were hidden layer sizes: [128, 64], dropout rate: 0.01, learning rate: 0.001, and batch size: 360. The low dropout rate would have meant very little regularization was necessary, and this may insinuate that the model might be able to properly learn from data without overfitting. This is supported by high  $R^2$  values (0.98 for both training and validation) and thus a good fit for the model to the data.

Scenario 2 was found to be forecast with high RMSE values, where train and validation scores were 618.14 and 649.02, respectively. The parameters for this scenario mainly involved more extensive hidden layers of [256, 128, 64], a slightly higher dropout rate of 0.05, and a learning rate of 0.001 with a batch size of 240. Since the forecasting horizon here is much larger, the model is made even more complex and, therefore, requires a high dropout rate. However, it is still found that the performance lags Scenario 1, attributed to the increased level of uncertainty and variability in forecasting much further ahead, which is inherently challenging.

Scenario 3 incorporated additional high-correlation features such as temperature and wind speed from Athens and Thessaloniki, along with the load data. Despite the inclusion of these potentially informative features, the model's performance did not improve significantly compared to Scenario 2. The best parameters for Scenario 3 were similar to those of Scenario 1 but included a learning rate of 0.005 and a batch size of 240. The train and validation RMSE scores were 559.11 and 594.71, respectively, with  $R^2$  values of 0.78 and 0.81. The results suggest that adding more features did not necessarily enhance the model's accuracy and may have introduced noise, complicating the learning process without providing substantial additional predictive power.

Dense layers are designed to capture non-linear relationships between input features and output predictions, but they might not be as effective in capturing temporal dependencies and correlations in time-series data compared to sequence-based models like GRU, RNN, and LSTM. These models are specifically designed to handle sequential data and can retain information over time, making them more suitable for tasks involving time-dependent features and long-term dependencies.



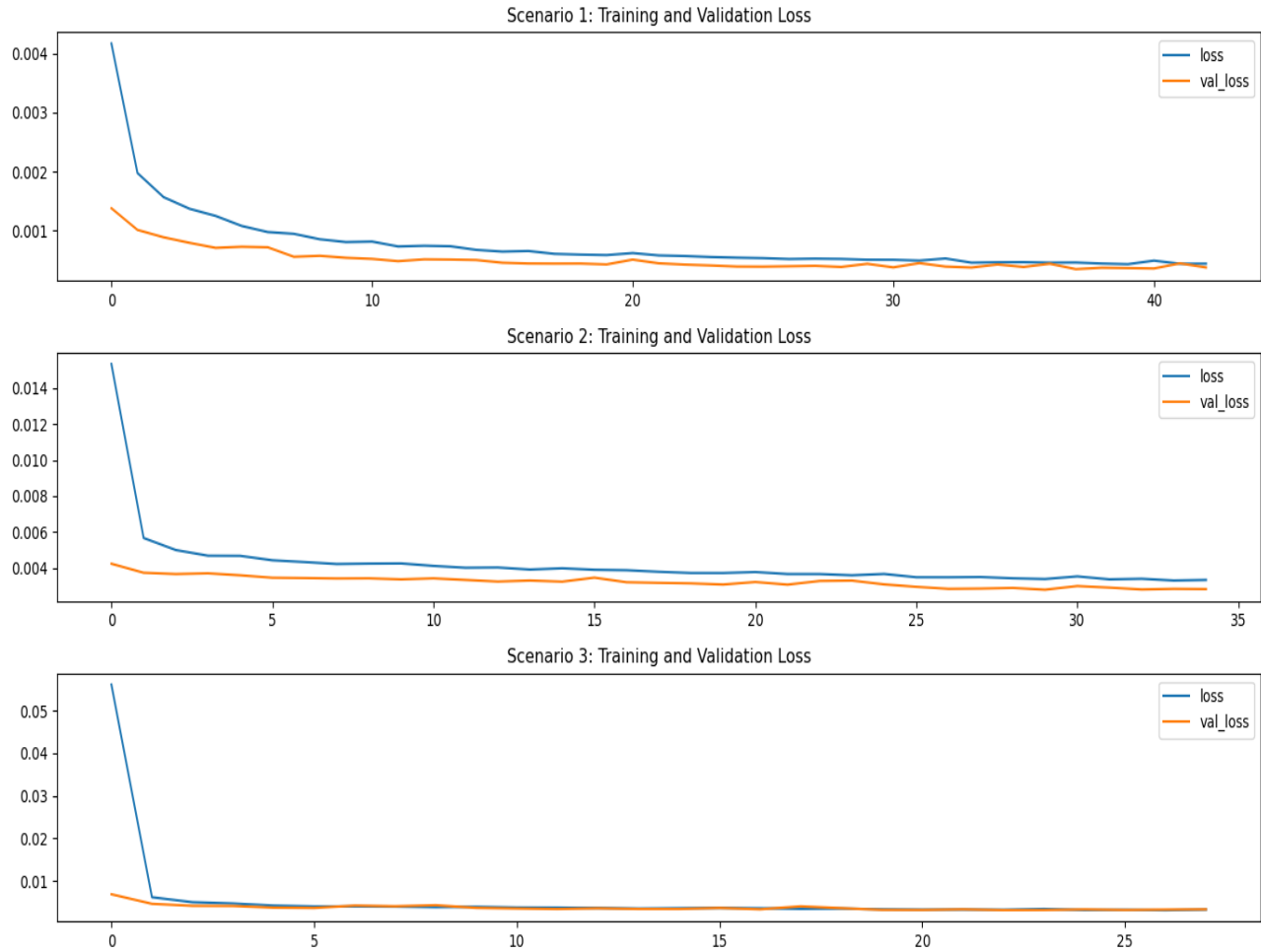


Figure 26: Training and Validation Loss for Each Scenario

The loss graphs for all scenarios present this typical training behavior: rapid initial decline for training and validation loss and flattening at the convergence of models. The first scenario shows a drastic decrease in loss with a small number of undulations that appear very smooth, suggestive of stable training and good generalization. Scenarios 2 and 3 show more fluctuations in validation loss, again indicating more significant challenges in prediction at longer horizons, with more features involved. In turn, this analysis suggests potential overfitting, especially in the third scenario, where validation loss diverges from the training loss. This can be a result of overfitting of the model and the addition of further features that didn't represent proportional predictive value. On the other hand, the loss curves for Scenario 1 denote well-regularized models with good performance on both training and validation sets.

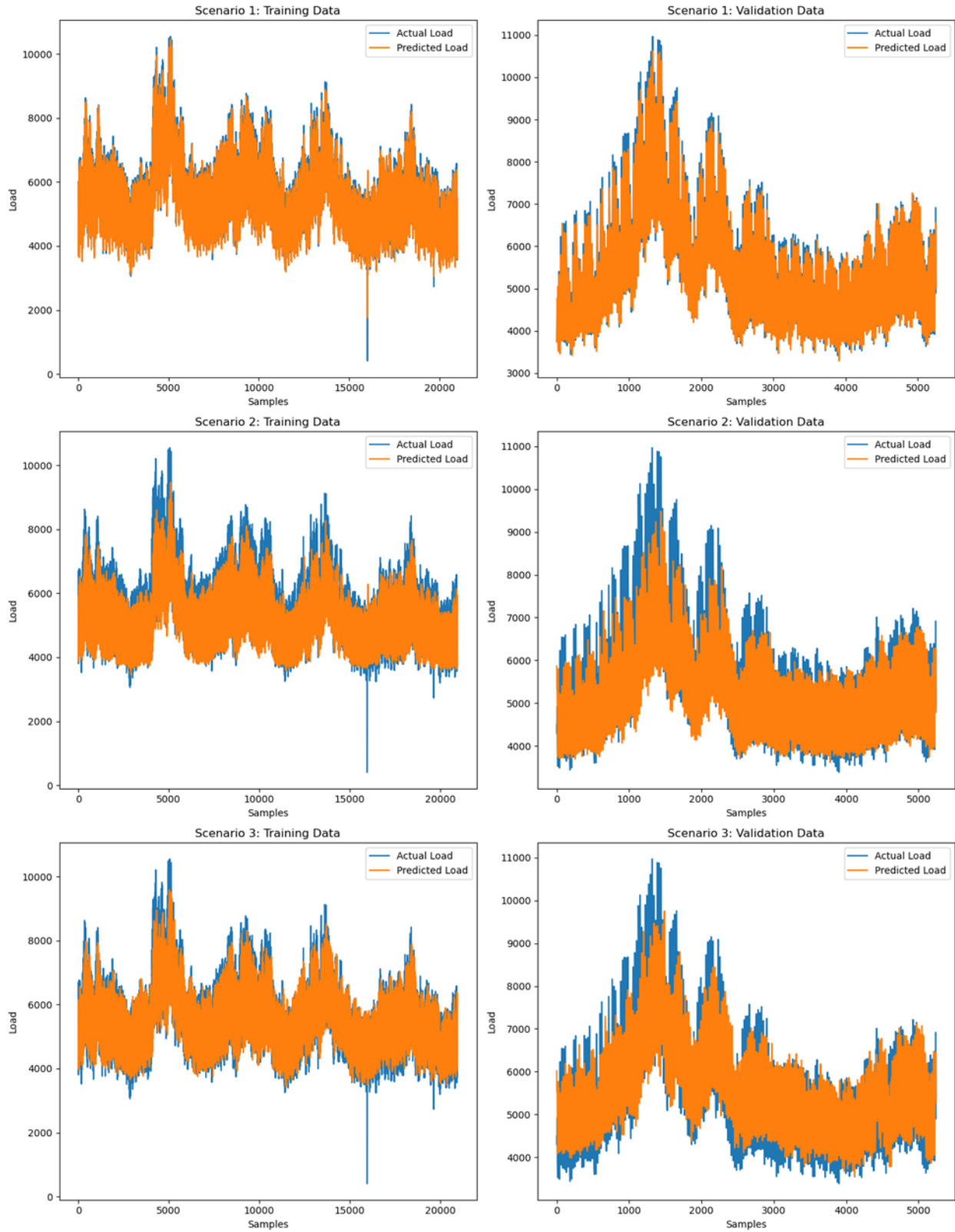


Figure 27: Actual vs. Predicted Load for Training and Validation Data Across Scenarios

For the first scenario, the model is able to predict the load data trend and seasonality quite well. The predicted values are very close to the actual load values, which means that the model has a good fit. The slight difference between the training and validation performance metrics shows that the model does not overfit the data, and performs well on new data. The model's performance in predicting one hour ahead using the previous 24 hours of data is seen to be accurate from the high  $R^2$  values and low RMSE and MAE scores.

In Scenario 2, although the trend is well captured, the model has higher errors compared to Scenario 1. The predictions are less accurate as seen by the larger dispersion around the actual values especially when the forecast is made for 48 hours in advance.

The last scenario, which includes high-correlation features, does not enhance the performance compared to the second one. The additional features may bring in noise which may make the learning process difficult without adding much value to the prediction.

In general, there is no severe overfitting across the scenarios; however, the Dense layers might not be exploiting the increased complexity of the models, which indicates that more complex models tailored for sequential data are required.

## 2.5 Recurrent Neural Network

In this section, we will discuss how the RNN models such as LSTM, GRU, and Simple RNN can be applied in electricity load forecasting. The RNN models are especially effective for time series prediction tasks because they are capable of modeling the temporal relationships and maintain the information over time. These models are intended for processing the sequential data and can adequately describe the dependencies between the past and future observations.

Due to the computational intensity and the requirement of a large amount of computational power, we have decided to employ Google Colab's virtual machine (VM) environment with GPU. The Colab VM offers access to GPUs which are very useful in the training of deep learning models. This makes it easier to search through the different model architectures and hyperparameters to use.

For the recurrent neural network models, we will examine three particular cases, each of which will be aimed at capturing different facets of electricity load forecasting with the help of different input-output window sizes and feature vectors.

For the first scenario, the input window size is set to 24 hours and the output window size is set to 48 hours. The factors that have been incorporated in this scenario are the load, temperature and wind speed. The main goal is to make a forecast of the load for the next 48 hours. The goal of this scenario is to identify the temporal patterns in the last 24 hours of data to make precise predictions for the next 2 days. This configuration is designed to use the most recent data to identify the current tendencies and short-term behavior of the load. Notably, the results of Scenario 1 will be compared with the results of Scenario 3 from the feed-forward model, which has the same feature set and the same target for the prediction,

which will allow assessing the efficiency of recurrent neural networks in comparison with feed-forward ones.

The second scenario uses an input window size of 48 hours and an output window size of 1 hour. This scenario is based on the same set of features as in the first scenario; the load, temperature and wind speed. The focus here is to assess the effect of having a longer history of data on the short-term prediction accuracy. This scenario aims at finding out whether including a longer history of input data can enhance the model's capability of forecasting the next few minutes of load demand.

In the third case, the input window size is set to 96 hours, and the output window size remains 48 hours. This scenario includes high-correlation variables like temperature and wind speed from two cities, Athens and Thessaloniki along with the load data. The goal is still to predict the load for the next 48 hours. The goal of this scenario is to compare the performance of the model when using a longer history of data and other variables that are highly related to the load. Thus, this scenario is intended to determine whether the enhanced complexity and the longer input time contribute to the significant improvement of the prediction accuracy in comparison with the shorter and less complex configurations.

These scenarios are designed to provide a comprehensive analysis of the performance of RNN models in short-term electricity load forecasting. By comparing the results across different configurations, we aim to identify the most effective model architecture and feature set for accurate load prediction. Comparing Scenario 1 of the RNN model with Scenario 3 of the feed-forward model will also help in understanding the strengths and weaknesses of recurrent versus feed-forward approaches.

## 2.5.1 Detailed Explanation of Recurrent Neural Networks Code and Implementation

The first step of the data preparation is to normalize the input and output data. Initially, the columns to be scaled are defined, including load and various weather-related features. The input data is then extracted and transformed as needed. The `MinMaxScaler` is used to normalize the input data in the range of `[0,1]` so that no feature is dominating the training of the model. Also, the 'Month' and 'Weekend' columns are added to the scaled input data without scaling as they are categorical variables. The output data (load) is also normalized. This step is important in standardizing the data to a range that is more suitable for the neural network training and also to enhance the speed and stability of the models' convergence.

```
columns_to_scale = ['Load', 'temperature_2m_ATH',  
'temperature_2m_SKG', 'wind_speed_10m_ATH', 'wind_speed_10m_SKG'] # Columns to scale  
  
input_data = data[columns_to_scale].values  
output_data = data['Load'].values.reshape(-1, 1) # Define input and output data  
input_scaler = MinMaxScaler()  
output_scaler = MinMaxScaler() # Initialize the scalers
```

```

Scale the input data
input_scaled = input_scaler.fit_transform(input_data)
# Concatenate 'Month' and 'Weekend' columns without scaling
input_scaled = np.concatenate([input_scaled, data[['Month', 'Weekend']].values], axis=1)
# Scale the output data
output_scaled = output_scaler.fit_transform(output_data)
# Verify the scaling
print("Input Scaled Shape:", input_scaled.shape)
print("Output Scaled Shape:", output_scaled.shape)

```

The script then moves to the creation of sliding window data which is crucial for time series forecasting since the data has been prepared. A function is created to produce input-output pairs by using a sliding window technique where “input\_window\_size” is the number of past observations to be used in making predictions and “output\_window\_size” is the number of steps ahead to be predicted. This function takes the dataset and for each input sequence, it appends the input sequence and its output value and then converts the lists to “numpy” arrays. This method enables the models to learn temporal dependencies in the data which is useful in time series analysis for prediction.

```

def create_sliding_window_data(x, y, input_window_size, output_window_size):
    input_x, output_y = [], []
    for i in range(len(x) - input_window_size - output_window_size + 1):
        input_x.append(x[i:i + input_window_size])
        output_y.append(y[i + input_window_size + output_window_size - 1])

    return np.array(input_x), np.array(output_y)

```

On the basis of the sliding window data, the script then defines a function to create different types of RNN models such as LSTM, GRU, and SimpleRNN based on the given parameters. This function takes the model type, input shape, hidden layers, dropout rate and learning rate as the parameters. It builds a sequential model and includes the RNN layers of the type LSTM, GRU, or SimpleRNN with the desired number of units and dropout layers to avoid overfitting. The last layer is a Dense layer with a single neuron for regression problems. The model is trained with the Adam optimizer and the mean squared error (MSE) loss function as this is a regression model. This makes it easy to test various RNN architectures since the code is broken down into modules.

```

# Define the RNN model creation function
def create_rnn_model(model_type, input_shape, hidden_layers, dropout_rate, learning_rate):
    model = Sequential()
    if model_type == 'LSTM':
        model.add(LSTM(hidden_layers[0], input_shape=input_shape,
            return_sequences=True))
    elif model_type == 'GRU':
        model.add(GRU(hidden_layers[0], input_shape=input_shape,
            return_sequences=True))
    elif model_type == 'SimpleRNN':
        model.add(SimpleRNN(hidden_layers[0], input_shape=input_shape,
            return_sequences=True))
    model.add(Dropout(dropout_rate))

    for units in hidden_layers[1:]:
        if model_type == 'LSTM':
            model.add(LSTM(units, return_sequences=(units != hidden_layers[-1])))
        elif model_type == 'GRU':
            model.add(GRU(units, return_sequences=(units != hidden_layers[-1])))
        elif model_type == 'SimpleRNN':
            model.add(SimpleRNN(units, return_sequences=(units != hidden_layers[-1])))

    model.add(Dropout(dropout_rate))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse', metrics=['mean_squared_error'])
    return model

```

The training and the evaluation of the models are done by the “train\_and\_evaluate\_model” function. This function does a grid search over the number of hidden layers, dropout rate, learning rate, and batch size to find the best model configuration. For each set of hyperparameters, the model is trained on the training data set with the help of early stopping and model checkpoint to avoid overfitting and to store the best model. Once the model is trained, the predicted values on the validation set are compared with the actual values and the performance metrics such as RMSE, MAE, and  $R^2$  are calculated. The best model configuration is selected using the validation RMSE and the model along with the training history is returned.

```
def train_and_evaluate_model(model_fn, model_name, X_train, y_train, X_val, y_val, input_shape,
hidden_layers_options, dropout_rate_options, learning_rate_options, batch_size_options,
epochs=100):
```

```
    best_params = None
    best_val_score = float('inf')
    best_model = None
    best_history = None
    results = []
```

```
    for hidden_layers, dropout_rate, learning_rate, batch_size in
product(hidden_layers_options, dropout_rate_options, learning_rate_options,
batch_size_options):
```

```
        print(f'Testing {model_name} with hidden_layers={hidden_layers},
dropout_rate={dropout_rate}, learning_rate={learning_rate},
batch_size={batch_size}')

```

```
        model = model_fn(input_shape, hidden_layers, dropout_rate, learning_rate)
        es_callback = EarlyStopping(monitor='val_loss', patience=5, verbose=0)
```

```
        checkpointer = ModelCheckpoint(filepath=f"best_{model_name}_model.h5",
verbose=0, save_best_only=True)
```

```
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
validation_split=0.2,
verbose=0,
callbacks=[es_callback, checkpointer])
```

```
        val_predictions = model.predict(X_val)
        val_predictions = output_scaler.inverse_transform(val_predictions)
        y_val_inv = output_scaler.inverse_transform(y_val)
```

```
        train_predictions = model.predict(X_train)
        train_predictions = output_scaler.inverse_transform(train_predictions)
        y_train_inv = output_scaler.inverse_transform(y_train)
```

```
        val_rmse = mean_squared_error(y_val_inv, val_predictions, squared=False)
        train_rmse = mean_squared_error(y_train_inv, train_predictions, squared=False)
        val_mae = mean_absolute_error(y_val_inv, val_predictions)
        train_mae = mean_absolute_error(y_train_inv, train_predictions)
        val_r2 = r2_score(y_val_inv, val_predictions)
        train_r2 = r2_score(y_train_inv, train_predictions)
```

```
        results.append((hidden_layers, dropout_rate, learning_rate, batch_size, val_rmse,
train_rmse, val_mae, train_mae, val_r2, train_r2))
```

```

        if val_rmse < best_val_score:
            best_val_score = val_rmse
            best_params = (hidden_layers, dropout_rate, learning_rate, batch_size)
            best_model = model
            best_history = history

hidden_layers, dropout_rate, learning_rate, batch_size = best_params

print(f'Best parameters for {model_name}: hidden_layers={hidden_layers},
dropout_rate={dropout_rate}, learning_rate={learning_rate}, batch_size={batch_size},
val_score={best_val_score:.2f}')

print(f'Best {model_name} - Train RMSE: {train_rmse:.2f}')
print(f'Best {model_name} - Validation RMSE: {val_rmse:.2f}')
print(f'Best {model_name} - Train MAE: {train_mae:.2f}')
print(f'Best {model_name} - Validation MAE: {val_mae:.2f}')
print(f'Best {model_name} - Train R2: {train_r2:.2f}')
print(f'Best {model_name} - Validation R2: {val_r2:.2f}')

# Plot training and validation loss
plt.figure(figsize=(14, 8))
plt.plot(best_history.history['loss'], label='Train Loss')
plt.plot(best_history.history['val_loss'], label='Validation Loss')
plt.title(f'{model_name}: Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

return best_model, best_history, y_train_inv, train_predictions, y_val_inv, val_predictions

```

To help in the visualization of the model's performance, a function is created that plots the actual versus predicted load for both the training and validation data sets. This function creates two subplots: There is one for the training set and the other one is for the validation set. It shows the actual and predicted load values of the first 100 samples to assess the model's effectiveness visually. This visualization allows for determining the degree to which the model fits the patterns and trends of the data and identifies the differences between the actual and predicted values. It is a very important step in the assessment of the model for electricity load forecasting.



```

# Plotting function
def plot_predictions(y_true_train, y_pred_train, y_true_val, y_pred_val, title):

    plt.figure(figsize=(14, 8))
    plt.subplot(2, 1, 1)
    plt.plot(y_true_train[:100], label='Actual Load')
    plt.plot(y_pred_train[:100], label='Predicted Load')
    plt.title(f'{title}: Training Data')
    plt.xlabel('Samples')
    plt.ylabel('Load')
    plt.legend()
    plt.subplot(2, 1, 2)
    plt.plot(y_true_val[:100], label='Actual Load')
    plt.plot(y_pred_val[:100], label='Predicted Load')
    plt.title(f'{title}: Validation Data')
    plt.xlabel('Samples')
    plt.ylabel('Load')
    plt.legend()
    plt.tight_layout()
    plt.show()

```

Finally, the script runs the defined scenarios and compares the performance of the LSTM, GRU, and SimpleRNN models. For each scenario, sliding window data is created, and the data is split into training and validation sets. The “train\_and\_evaluate\_model” function is called for each model type, and the results are stored. After training all models, the combined results are printed, and the actual versus predicted load plots are generated for each model and scenario. This comprehensive evaluation and comparison of different RNN models and scenarios provide valuable insights into the models' performance and suitability for short-term electricity load forecasting tasks.

```

# Collect and print results
lstm_results = []
gru_results = []
rnn_results = []

# Scenarios
scenarios = [{"input_window_size": 24, "output_window_size": 48, "name": "Scenario 1"},
             {"input_window_size": 48, "output_window_size": 1, "name": "Scenario 2"},
             {"input_window_size": 96, "output_window_size": 48, "name": "Scenario 3"}]

# Model configurations
hidden_layers_options = [[64, 32], [128, 64], [256, 128, 64]]
dropout_rate_options = [0.1, 0.01, 0.05]
learning_rate_options = [0.001, 0.005, 0.01, 0.1]
batch_size_options = [120, 240, 360]

```

```

for scenario in scenarios:
    # Create sliding window data
    X, y = create_sliding_window_data(input_scaled, output_scaled,
        input_window_size=scenario["input_window_size"],
        output_window_size=scenario["output_window_size"])

    split_index = int(len(X) * 0.8)
    X_train, X_val = X[:split_index], X[split_index:]
    y_train, y_val = y[:split_index], y[split_index:]

    # Train and evaluate LSTM model
    best_lstm_model, lstm_history, lstm_y_train_inv, lstm_train_predictions, lstm_y_val_inv,
    lstm_val_predictions = train_and_evaluate_model(lambda input_shape, hidden_layers,
        dropout_rate, learning_rate: create_rnn_model('LSTM', input_shape, hidden_layers,
        dropout_rate, learning_rate), 'LSTM', X_train, y_train, X_val, y_val,
        (X_train.shape[1], X_train.shape[2]), hidden_layers_options, dropout_rate_options,
        learning_rate_options, batch_size_options)

    # Store LSTM results
    lstm_results.append({
        "scenario_name": scenario["name"],
        "train_rmse": mean_squared_error(lstm_y_train_inv, lstm_train_predictions, squared=False),
        "val_rmse": mean_squared_error(lstm_y_val_inv, lstm_val_predictions, squared=False),
        "train_mae": mean_absolute_error(lstm_y_train_inv, lstm_train_predictions),
        "val_mae": mean_absolute_error(lstm_y_val_inv, lstm_val_predictions),
        "train_r2": r2_score(lstm_y_train_inv, lstm_train_predictions),
        "val_r2": r2_score(lstm_y_val_inv, lstm_val_predictions)})

    # Train and evaluate GRU model
    best_gru_model, gru_history, gru_y_train_inv, gru_train_predictions, gru_y_val_inv,
    gru_val_predictions = train_and_evaluate_model(lambda input_shape, hidden_layers,
        dropout_rate, learning_rate: create_rnn_model('GRU', input_shape, hidden_layers,
        dropout_rate, learning_rate), 'GRU', X_train, y_train, X_val, y_val, (X_train.shape[1],
        X_train.shape[2]), hidden_layers_options, dropout_rate_options,
        learning_rate_options, batch_size_options)

    # Store GRU results
    gru_results.append({
        "scenario_name": scenario["name"],
        "train_rmse": mean_squared_error(gru_y_train_inv, gru_train_predictions, squared=False),
        "val_rmse": mean_squared_error(gru_y_val_inv, gru_val_predictions, squared=False),
        "train_mae": mean_absolute_error(gru_y_train_inv, gru_train_predictions),
        "val_mae": mean_absolute_error(gru_y_val_inv, gru_val_predictions),
        "train_r2": r2_score(gru_y_train_inv, gru_train_predictions),
        "val_r2": r2_score(gru_y_val_inv, gru_val_predictions)})

```

```

# Train and evaluate SimpleRNN model
best_rnn_model, rnn_history, rnn_y_train_inv, rnn_train_predictions, \
↳ rnn_y_val_inv, rnn_val_predictions = train_and_evaluate_model(
lambda input_shape, hidden_layers, dropout_rate, learning_rate: \
↳ create_rnn_model('SimpleRNN', input_shape, hidden_layers, dropout_rate, \
↳ learning_rate),
'SimpleRNN',
X_train, y_train, X_val, y_val,
(X_train.shape[1], X_train.shape[2]),
hidden_layers_options, dropout_rate_options, learning_rate_options, \
↳ batch_size_options
)
# Store SimpleRNN results
rnn_results.append({
"scenario_name": scenario["name"],
"train_rmse": mean_squared_error(rnn_y_train_inv, \
↳ rnn_train_predictions, squared=False),
"val_rmse": mean_squared_error(rnn_y_val_inv, rnn_val_predictions, \
↳ squared=False),
"train_mae": mean_absolute_error(rnn_y_train_inv, \
↳ rnn_train_predictions),
"val_mae": mean_absolute_error(rnn_y_val_inv, rnn_val_predictions),
"train_r2": r2_score(rnn_y_train_inv, rnn_train_predictions),
"val_r2": r2_score(rnn_y_val_inv, rnn_val_predictions)
})

# Plot predictions for LSTM
plot_predictions(lstm_y_train_inv, lstm_train_predictions, lstm_y_val_inv,
lstm_val_predictions, f'LSTM {scenario["name"]}')
# Plot predictions for GRU
plot_predictions(gru_y_train_inv, gru_train_predictions, gru_y_val_inv, gru_val_predictions,
f'GRU {scenario["name"]}')
# Plot predictions for SimpleRNN
plot_predictions(rnn_y_train_inv, rnn_train_predictions, rnn_y_val_inv, rnn_val_predictions,
f'SimpleRNN {scenario["name"]}')
# Print combined results
print_combined_results(lstm_results, "LSTM")
print_combined_results(gru_results, "GRU")
print_combined_results(rnn_results, "SimpleRNN")

```

## 2.5.2 Comparative Analysis of Recurrent Neural Models Scenarios and Performance Evaluation

In this section, we present a comprehensive analysis of the results obtained from the various recurrent neural network (RNN) models: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and SimpleRNN. We compare their performance on Scenario 1, which involves predicting electricity load using high-correlation features such as temperature and wind speed from Athens and Thessaloniki, with a 24-hour input window size and a 48-hour prediction horizon. The performance metrics for each model include RMSE, MAE, and  $R^2$  scores, which provide insights into the accuracy and reliability of the models.

Performance Metric	LTSM	GRU	SimpleRNN
Train RMSE	537.75	456.10	605.65
Validation RMSE	573.17	485.32	601.06
Train MAE	407.91	339.34	474.03
Validation MAE	430.66	359.03	483.40
Train $R^2$	0.80	0.85	0.74
Validation $R^2$	0.83	0.88	0.81

Table 6: Performance Metrics for Scenario 1

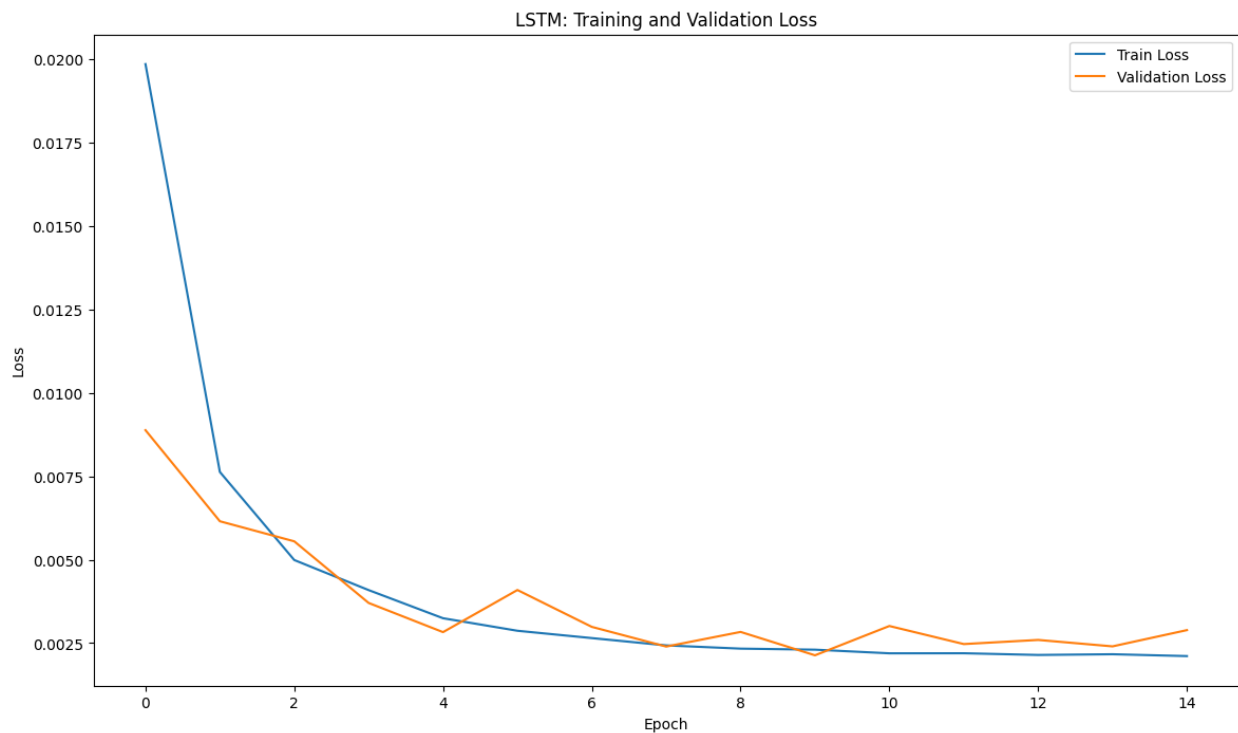


Figure 28: Training and Validation Loss for Scenario1 and LSTM model

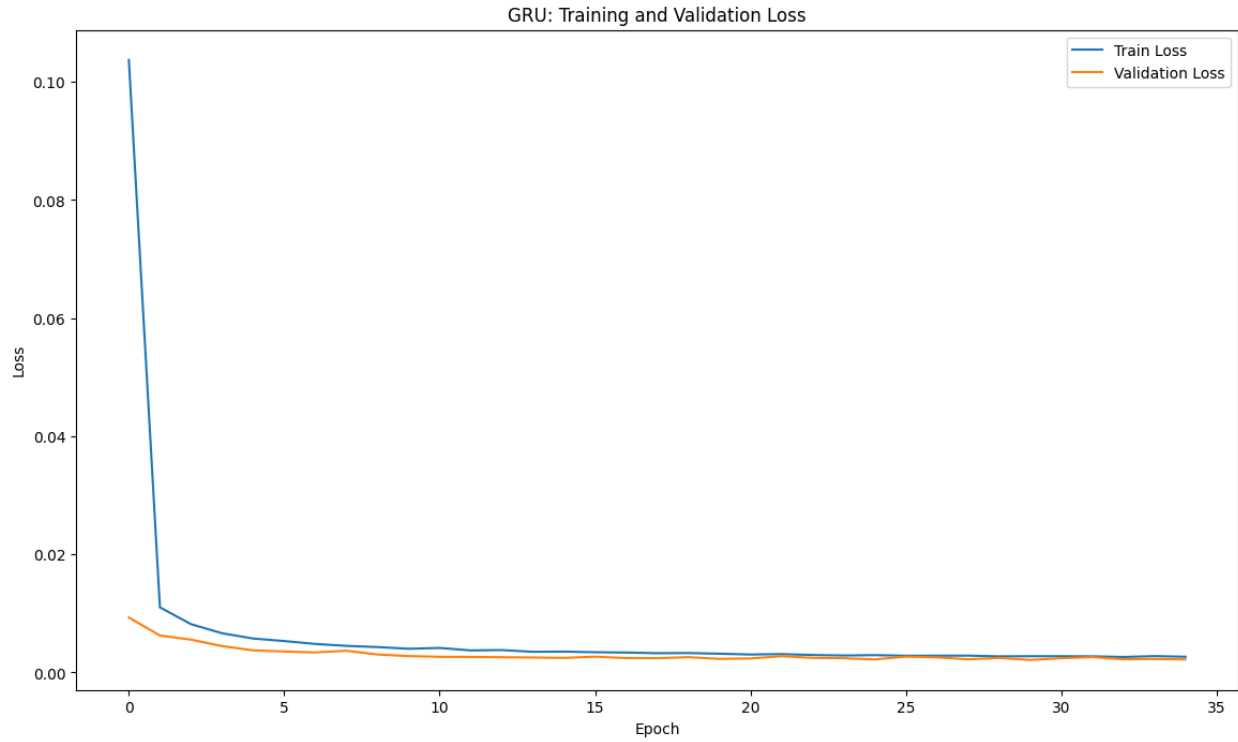


Figure 29: Training and Validation Loss for Scenario1 and GRU model

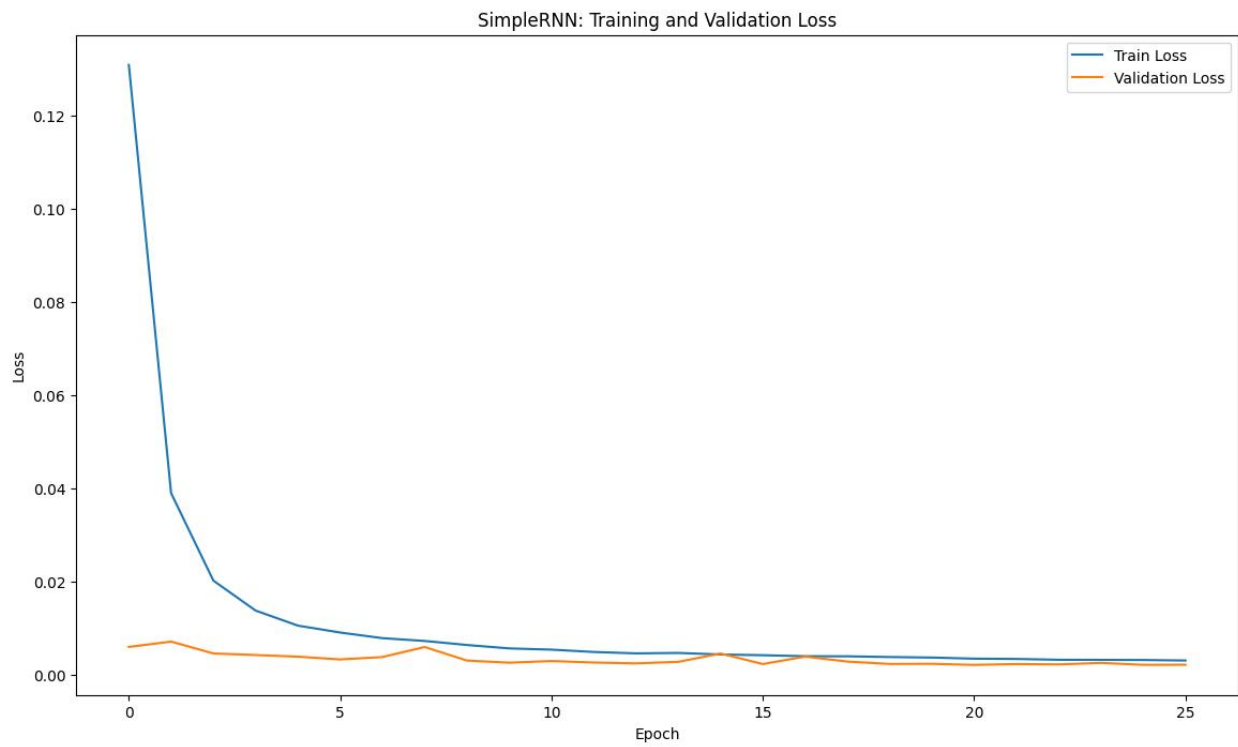


Figure 30: Training and Validation Loss for Scenario1 and SimpleRNN model

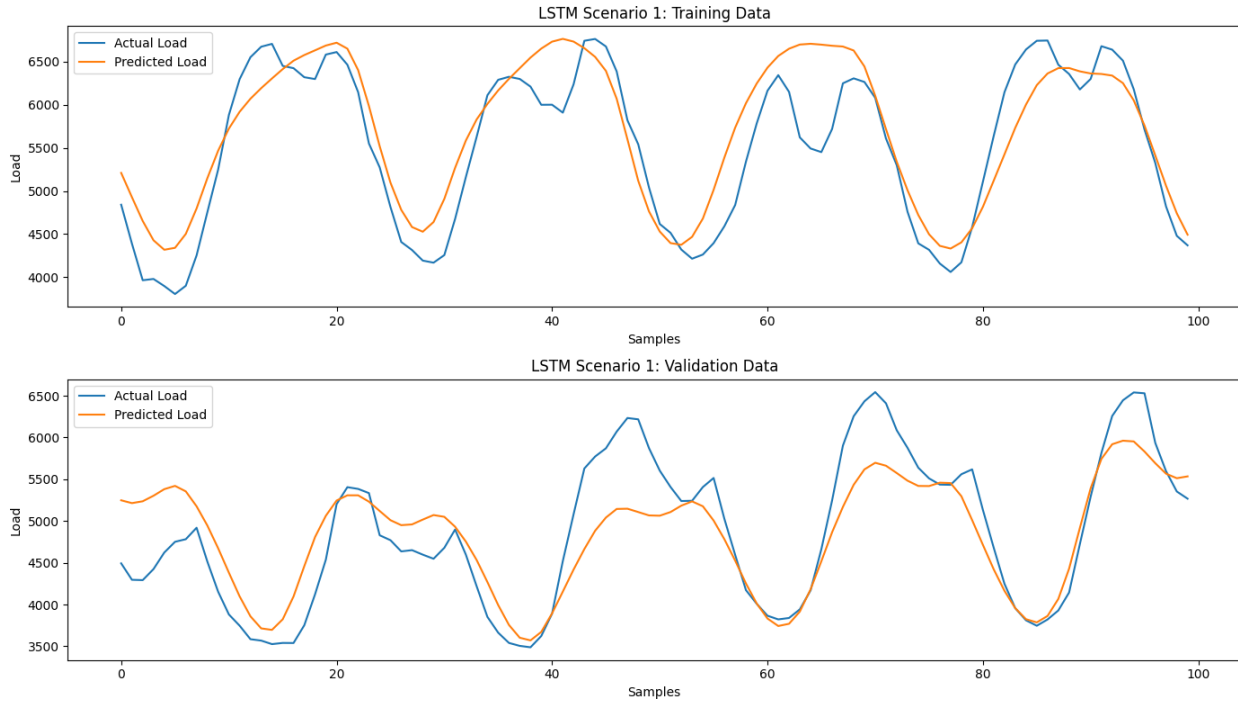


Figure 31: Actual vs. Predicted Load for LSTM Scenario 1

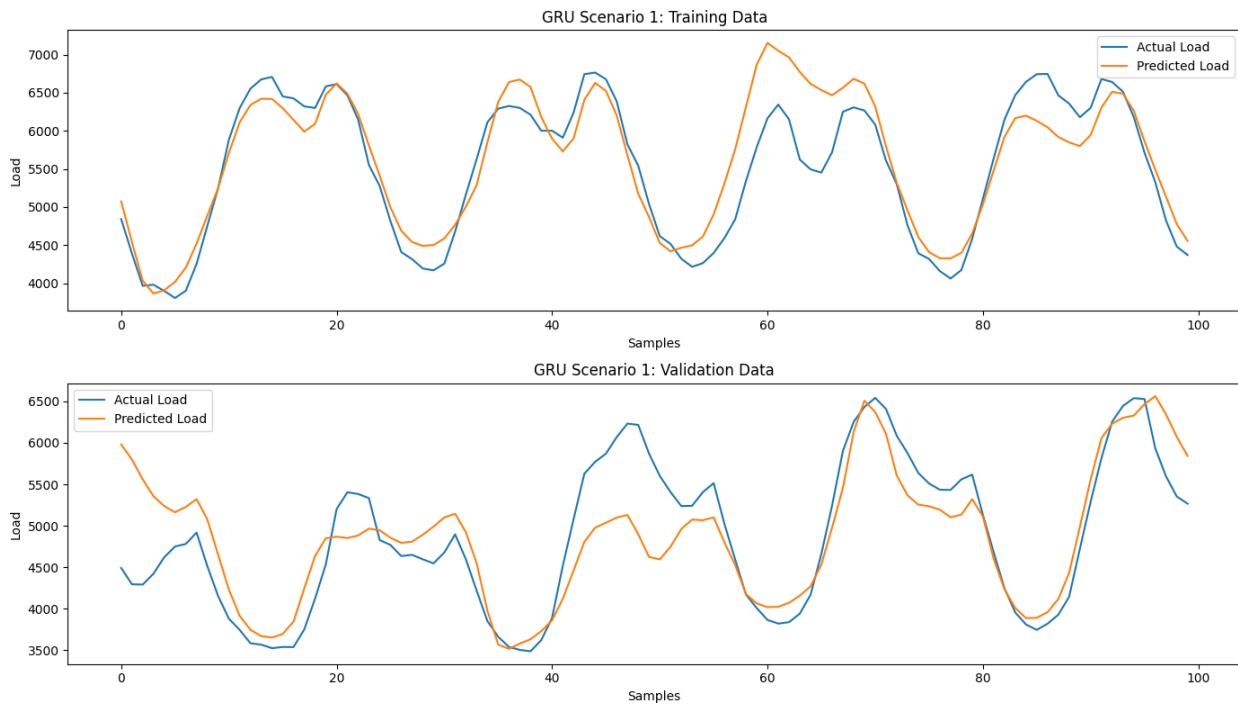


Figure 32: Actual vs. Predicted Load for GRU Scenario 1

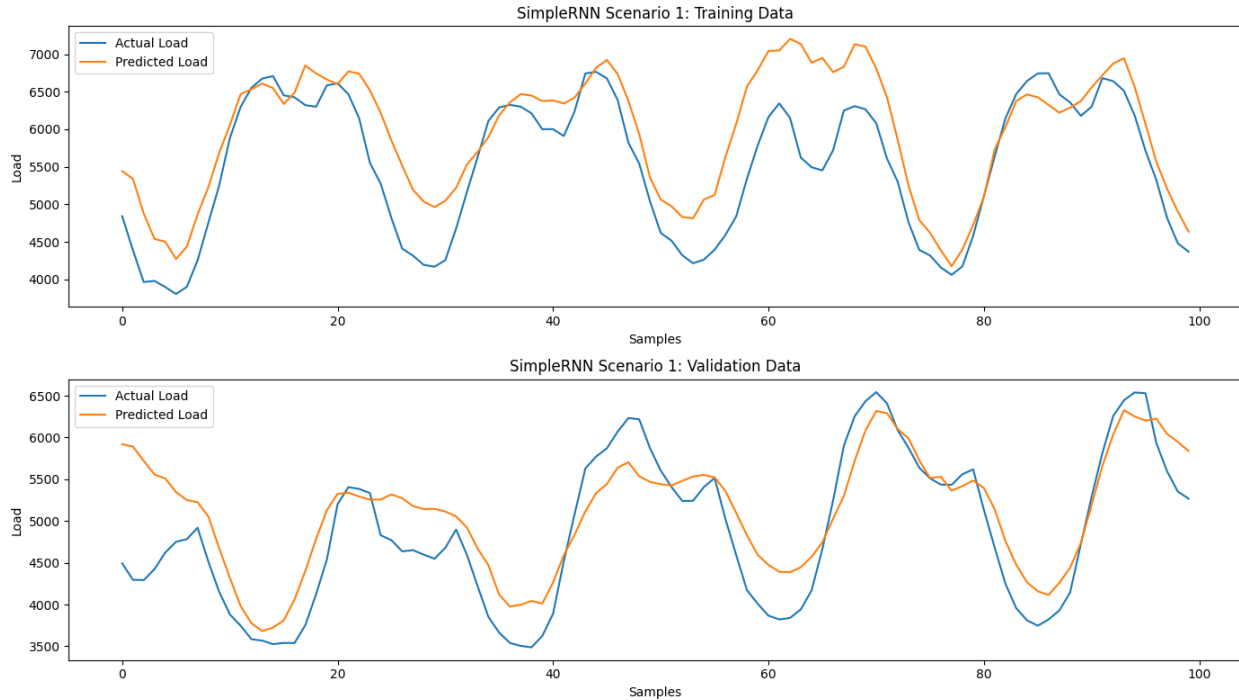


Figure 33: Actual vs. Predicted Load for SimpleRNN Scenario 1

For the LSTM model, the best parameters identified were hidden layers [128,64], a dropout rate of 0.01, a learning rate of 0.001, and a batch size of 120, achieved a validation RMSE of 573.17. The LSTM model showed a strong fit with an  $R^2$  score of 0.83 for validation data, indicating it captured the patterns in the load data effectively. The training and validation loss curves for the LSTM model showed a steady decrease, suggesting a well-generalized model with minimal overfitting.

In this case, the GRU model was found to be better than the LSTM model with a validation RMSE of 485.32 with hidden layers [256,128,64], dropout rate of 0.1, a learning rate of 0.01, and a batch size of 240. The validation  $R^2$  score was 0.88, indicating that the GRU model was particularly effective in learning the temporal dependencies in the data. The training and validation loss curves demonstrated stable training, with the GRU model effectively balancing the trade-off between bias and variance.

The SimpleRNN model, with best parameters of hidden layers [128,64], a dropout rate of 0.01, a learning rate of 0.01, and a batch size of 120, had the highest validation RMSE of 601.06 among the three RNN models. Although the validation  $R^2$  score was still reasonably high at 0.81, indicating that SimpleRNN was able to capture significant patterns in the data, it was less effective compared to LSTM and GRU. The training and validation loss curves showed more fluctuations, suggesting that the SimpleRNN model struggled more with generalization compared to LSTM and GRU.

Analyzing the plots of actual vs. predicted load values for each model further reveals the effectiveness of these models in capturing the load trends and seasonality. The GRU model's predictions were closely aligned with the actual load values, showing smooth transitions and fewer deviations, reflecting its superior performance metrics. The LSTM model also demonstrated strong predictive capability, with the predicted values closely following the actual load patterns. On the other hand, the

SimpleRNN model exhibited larger deviations and inconsistencies, indicating its relative weakness in capturing the complex temporal dependencies.

Comparing the best RNN model, which is the GRU, with the Feed-Forward model in Scenario 3, we observe significant improvements. The Feed-Forward model achieved a validation RMSE of 594.71 and a validation  $R^2$  of 0.81. In contrast, the GRU model's validation RMSE of 485.32 and validation  $R^2$  of 0.88 indicate a more accurate and reliable performance. This improvement can be attributed to the GRU's ability to capture temporal dependencies and interactions between multiple correlated features more effectively than the feed-forward architecture. The comparison highlights the superior capability of recurrent models like GRU in handling time-series data, especially when long-term dependencies and sequential patterns are crucial for accurate predictions.

This comprehensive comparison demonstrates the effectiveness of the GRU model in outperforming the feed-forward model for electricity load forecasting. The GRU model's ability to capture the sequential dependencies in the data leads to more accurate predictions and better generalization on the validation set. The inclusion of high-correlation features in the feed-forward model did not significantly enhance its performance, likely due to the model's inability to effectively leverage the temporal relationships between the features. In contrast, the GRU model, designed to handle sequential data, successfully utilized these features to improve forecasting accuracy.

In Scenario 2, the input window size is increased to 48 hours while the output window size is reduced to 1 hour. This scenario uses the same features as the first scenario—load, temperature, and wind speed—to evaluate whether including a longer history of input data can enhance the model's capability of forecasting the next hour of load demand. Additionally, the sliding window approach is adjusted to focus on capturing short-term changes in load by predicting one hour ahead. The focus here is to determine if a longer history provides a significant advantage in capturing the temporal patterns necessary for accurate short-term predictions.

Performance Metric	LSTM	GRU	SimpleRNN
Train RMSE	248.56	178.75	456.99
Validation RMSE	312.35	161.64	523.77
Train MAE	181.63	134.5	369.36
Validation MAE	231.15	123.84	437.8
Train $R^2$	0.96	0.98	0.85
Validation $R^2$	0.95	0.99	0.86

Table 7: Performance Metrics for Scenario 2



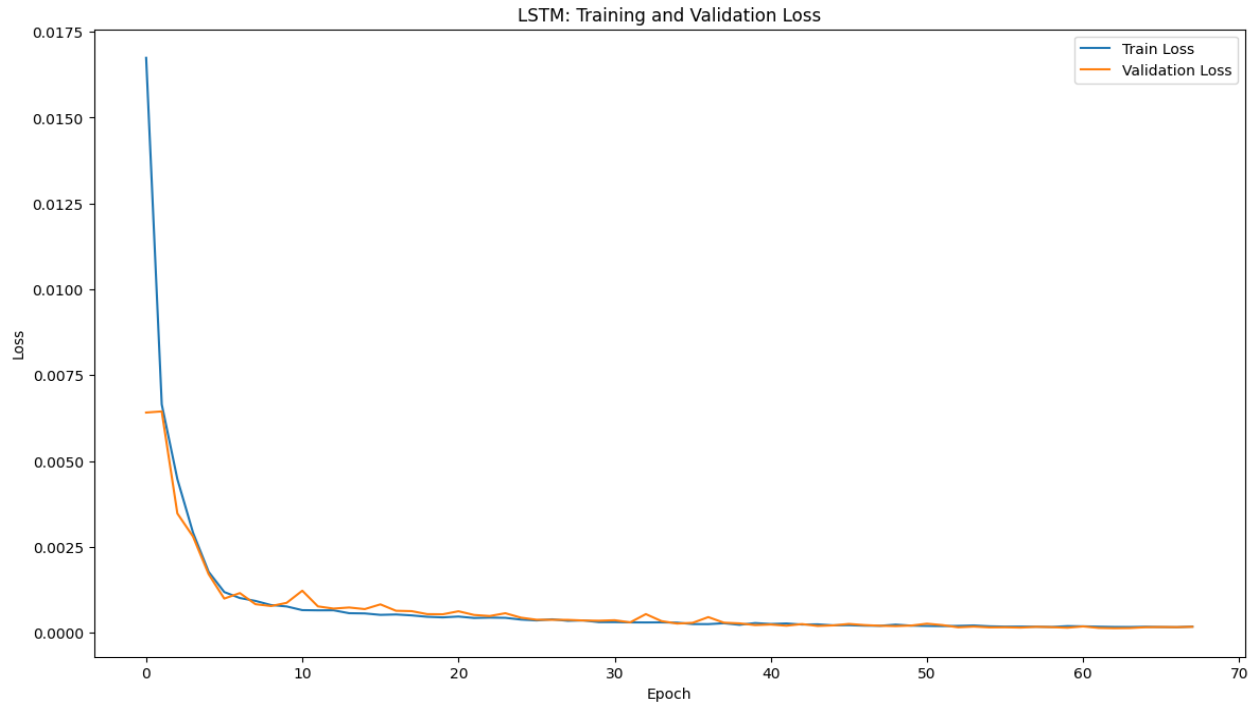


Figure 34: Training and Validation Loss for Scenario 2 and LSTM model

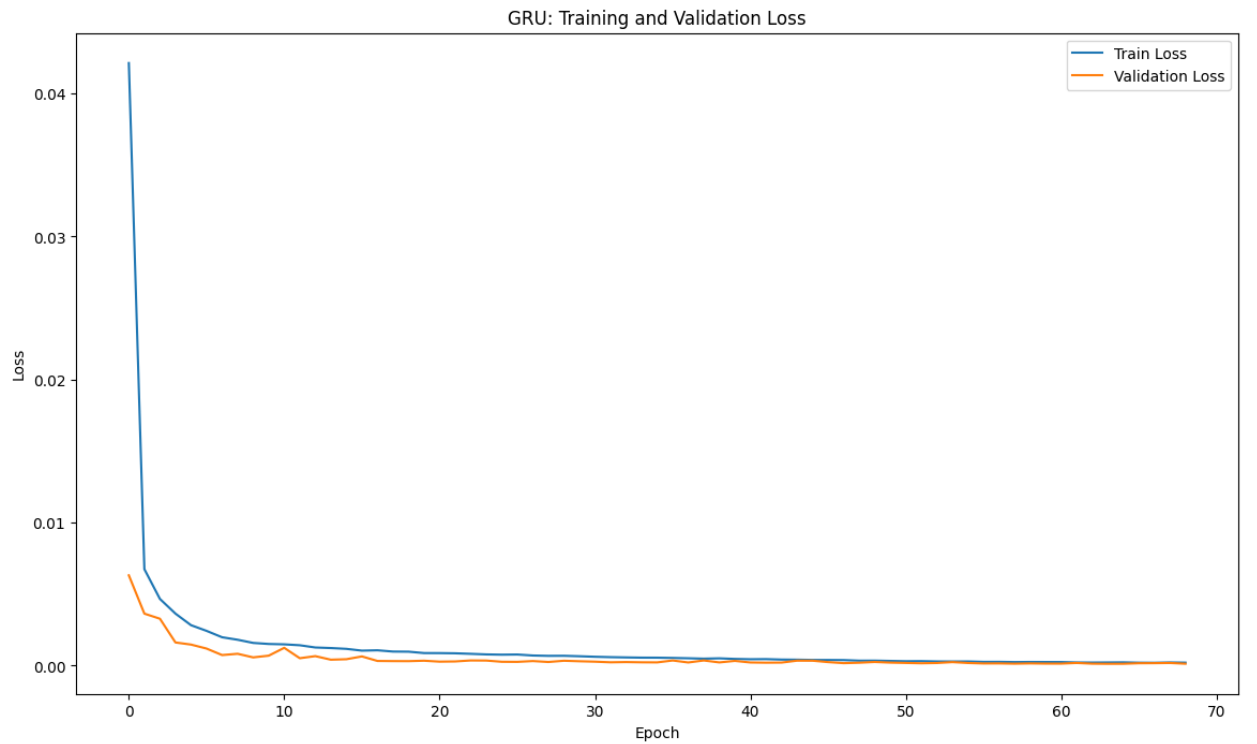


Figure 35: Training and Validation Loss for Scenario 2 and GRU model

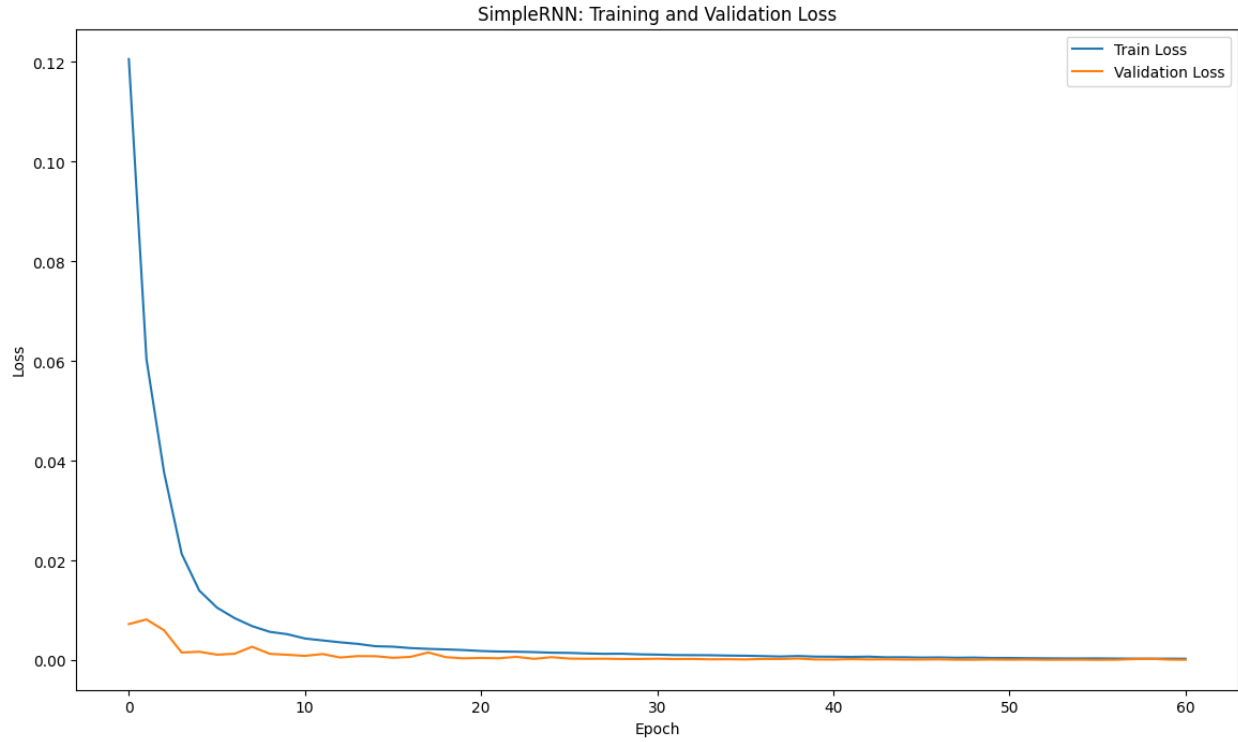


Figure 36: Training and Validation Loss for Scenario 2 and SimpleRNN model

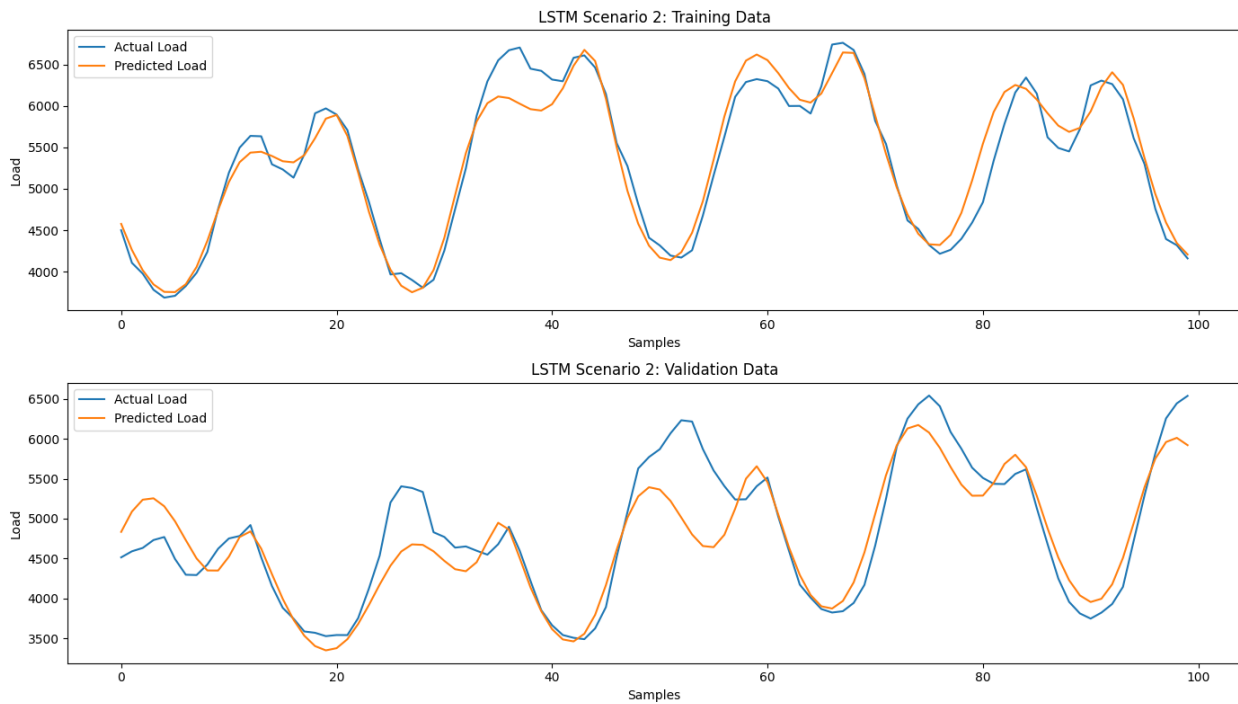


Figure 37: Actual vs. Predicted Load for LSTM Scenario 2

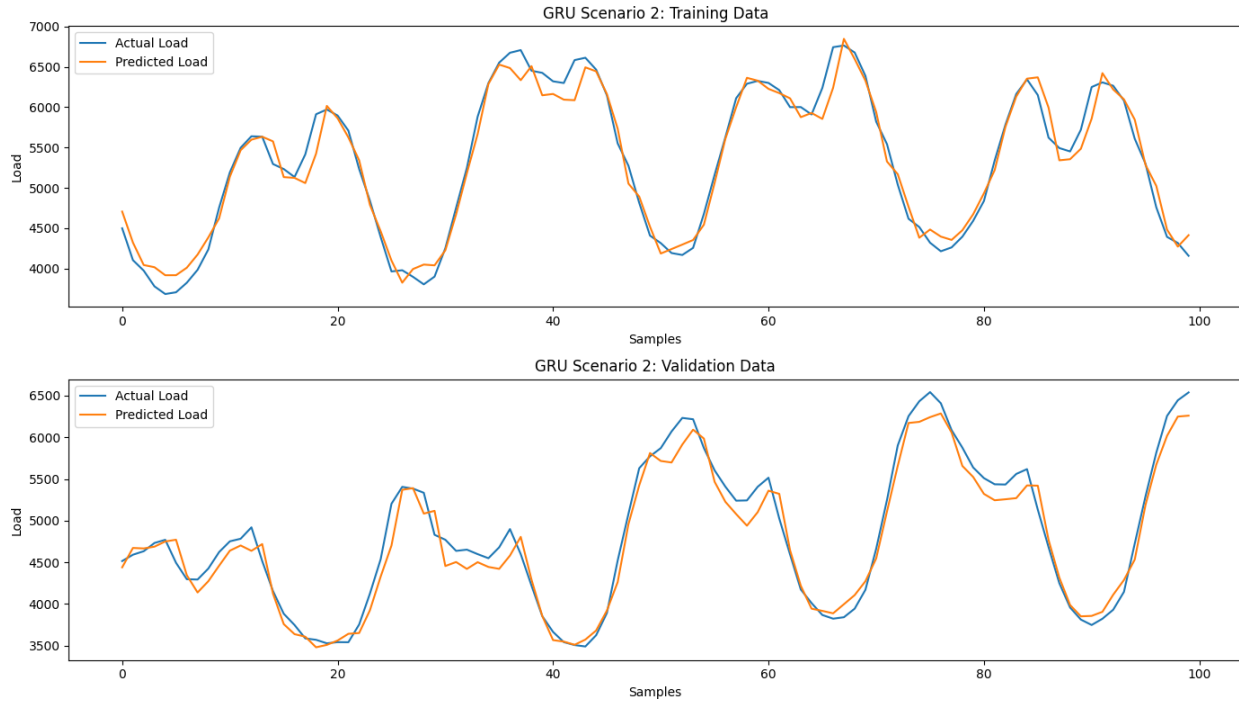


Figure 38: Actual vs. Predicted Load for GRU Scenario 2

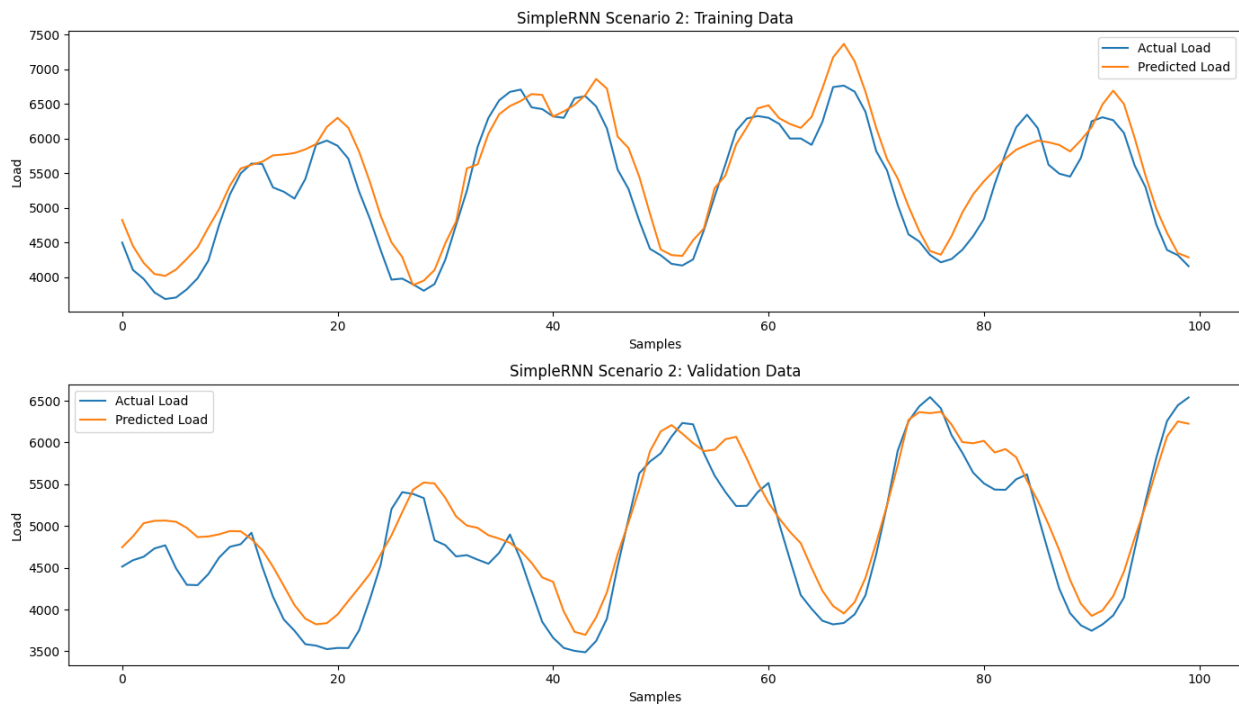


Figure 39: Actual vs. Predicted Load for SimpleRNN Scenario 2

For the LSTM model in Scenario 2, the best parameters identified were hidden layers of [128, 64], a dropout rate of 0.01, a learning rate of 0.001, and a batch size of 120, achieving a validation RMSE of 312.35. The LSTM model demonstrated substantial improvement in performance compared to Scenario

1, with an  $R^2$  score of 0.95 for the validation data, indicating it captured the load data patterns effectively. The training and validation loss curves (Figure 34) showed a consistent decrease, suggesting a well-generalized model with minimal overfitting. The actual vs. predicted load graphs (Figure 37) reveal that the LSTM model closely followed the actual load trends with minimal deviations, showing strong predictive capability.

For this case, the GRU model outperformed the LSTM model with a validation RMSE of 161.64, using hidden layers of [256, 128, 64], a dropout rate of 0.1, a learning rate of 0.001, and a batch size of 120. The validation  $R^2$  score was 0.99, demonstrating the GRU model's effectiveness in learning the temporal dependencies in the data. The training and validation loss curves (Figure 35) exhibited stable training, with the GRU model maintaining a balance between bias and variance. The actual vs. predicted load graphs (Figure 38) for the GRU model showed that the predictions were closely aligned with the actual load values, indicating smooth transitions and fewer deviations, reflecting its superior performance metrics.

The SimpleRNN model, with best parameters of hidden layers of [64, 32], a dropout rate of 0.05, a learning rate of 0.1, and a batch size of 120, achieved a validation RMSE of 523.77. Although the validation  $R^2$  score was reasonably high at 0.86, indicating that SimpleRNN could capture significant patterns in the data, it was less effective compared to LSTM and GRU. The training and validation loss curves (Figure 36) showed more fluctuations, suggesting that the SimpleRNN model struggled more with generalization. The actual vs. predicted load graphs (Figure 39) for SimpleRNN exhibited larger deviations and inconsistencies, indicating its relative weakness in capturing complex temporal dependencies.

As a result of comparing the results of the two scenarios, it is evident that using a larger number of input data history and a different sliding window approach enhanced the predictive capability of both the LSTM and GRU models. In particular, the GRU model provided a significant improvement with a validation RMSE of 161.64 in Scenario 2 compared to 485.32 in Scenario 1. This enhancement can be ascribed to the GRU's capacity to model temporal relationships with a larger input window and its appropriateness for short-term load forecasting. The comparison shows that the GRU model is more efficient in working with time-series data particularly when long term dependencies and short term predictions are relevant for the accurate prediction. The decrease in the RMSE and the increase in the  $R^2$  in the Scenario 2 show that the usage of a longer time period and a sliding window with a specific focus helps to increase the accuracy of the forecast. This comparison shows that the GRU model is better than LSTM and SimpleRNN in Scenario 2, and to achieve better results, it is necessary to consider the longer history of data and the right sliding window settings.

In the third scenario, the input window size is 96 hours and the output window size is still 48 hours. This configuration, which includes the high-correlation variables of temperature and wind speed from Athens and Thessaloniki along with the load data, is used to predict the load for the next 48 hours. The goal is to determine if there is a notable gain in the prediction performance when using a larger input history size than the smaller ones.

Performance Metric	LSTM	GRU	SimpleRNN
Train RMSE	423.5	456.52	769.31
Validation RMSE	508.09	466.72	845.89
Train MAE	321.79	349.71	640.34
Validation MAE	378.62	361.41	726.55
Train $R^2$	0.87	0.85	0.59
Validation $R^2$	0.86	0.89	0.62

Table 8: Performance Metrics for Scenario 3

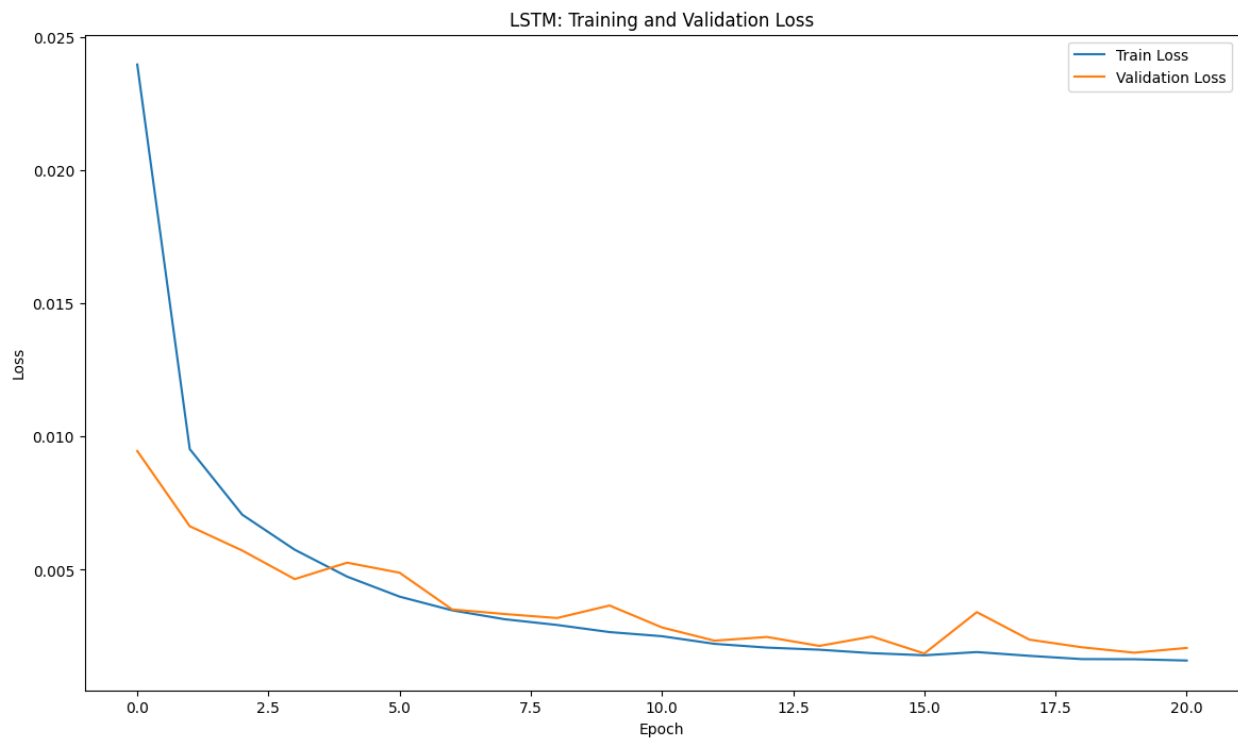


Figure 40: Training and Validation Loss for Scenario 3 and LSTM model

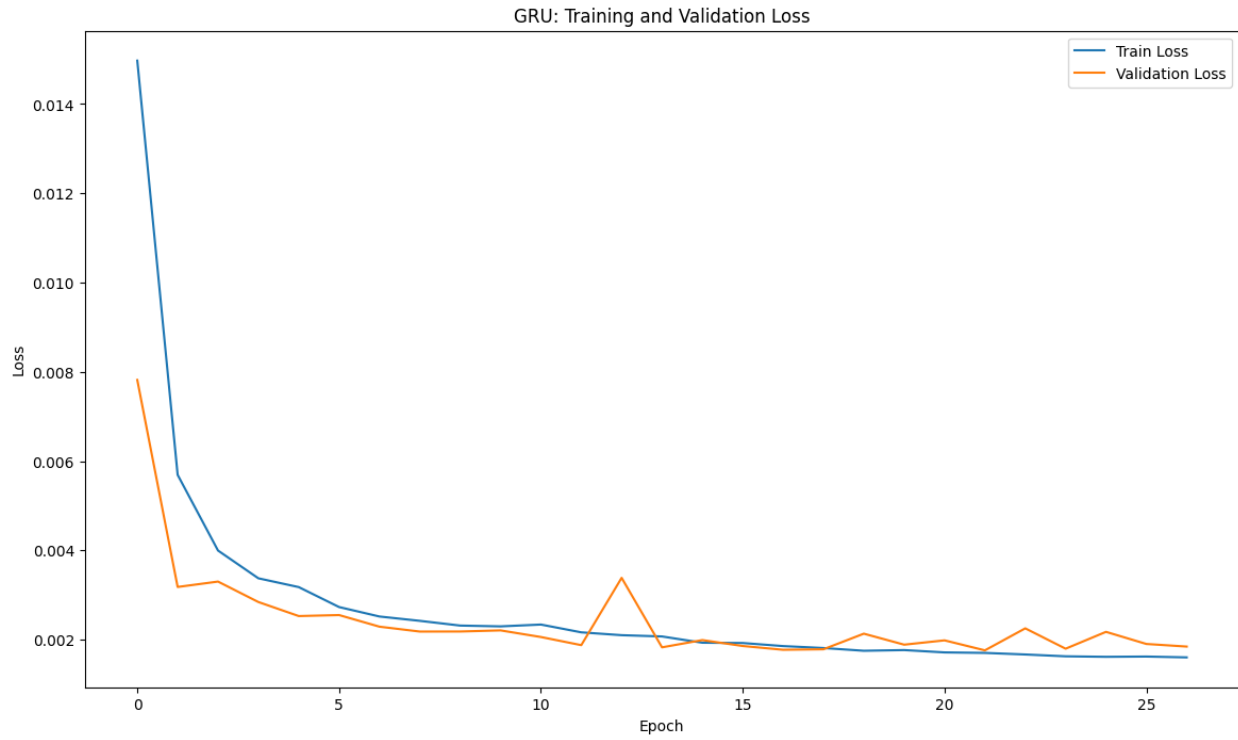


Figure 41: Training and Validation Loss for Scenario 3 and GRU model

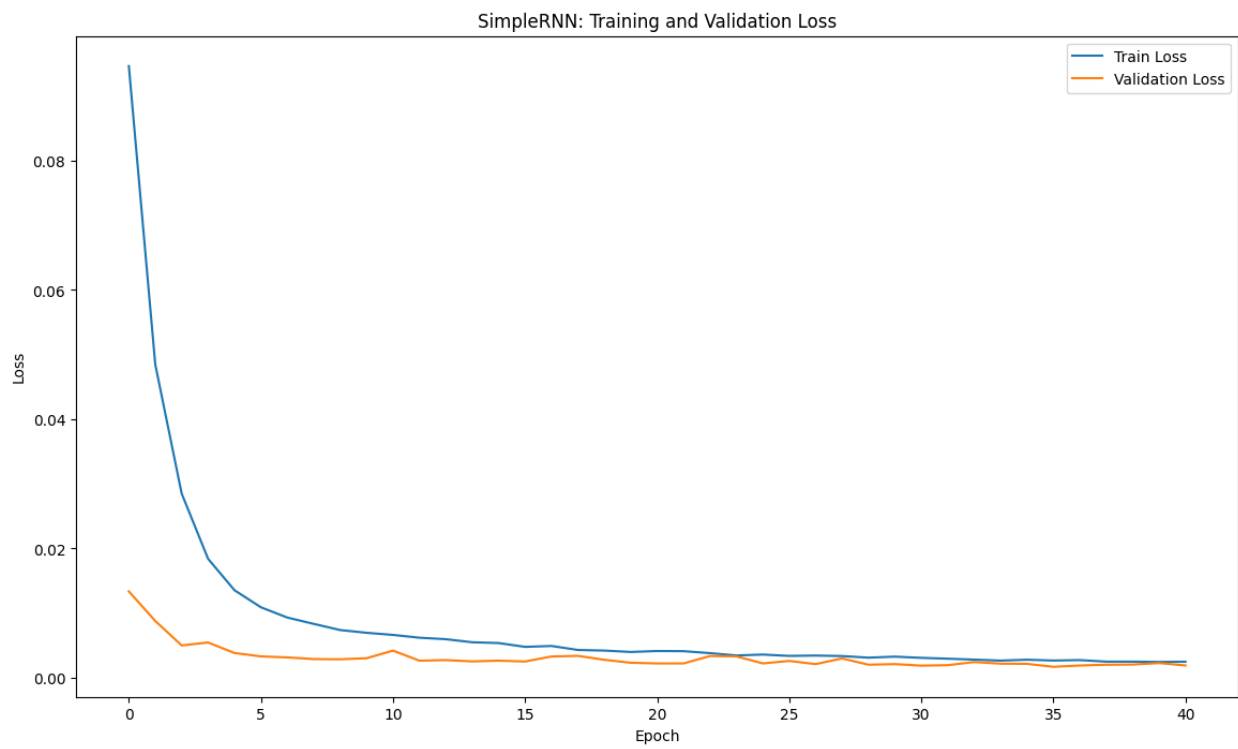


Figure 42: Training and Validation Loss for Scenario 3 and SimpleRNN model

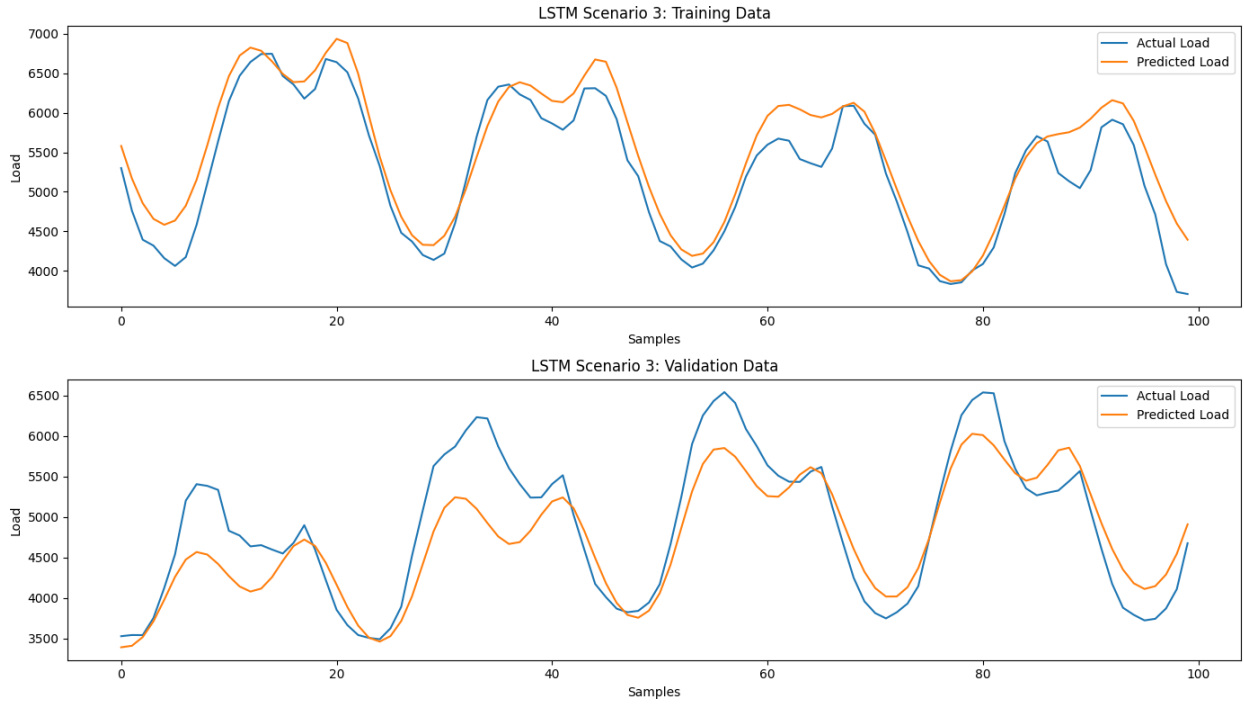


Figure 43: Actual vs. Predicted Load for LSTM Scenario 3

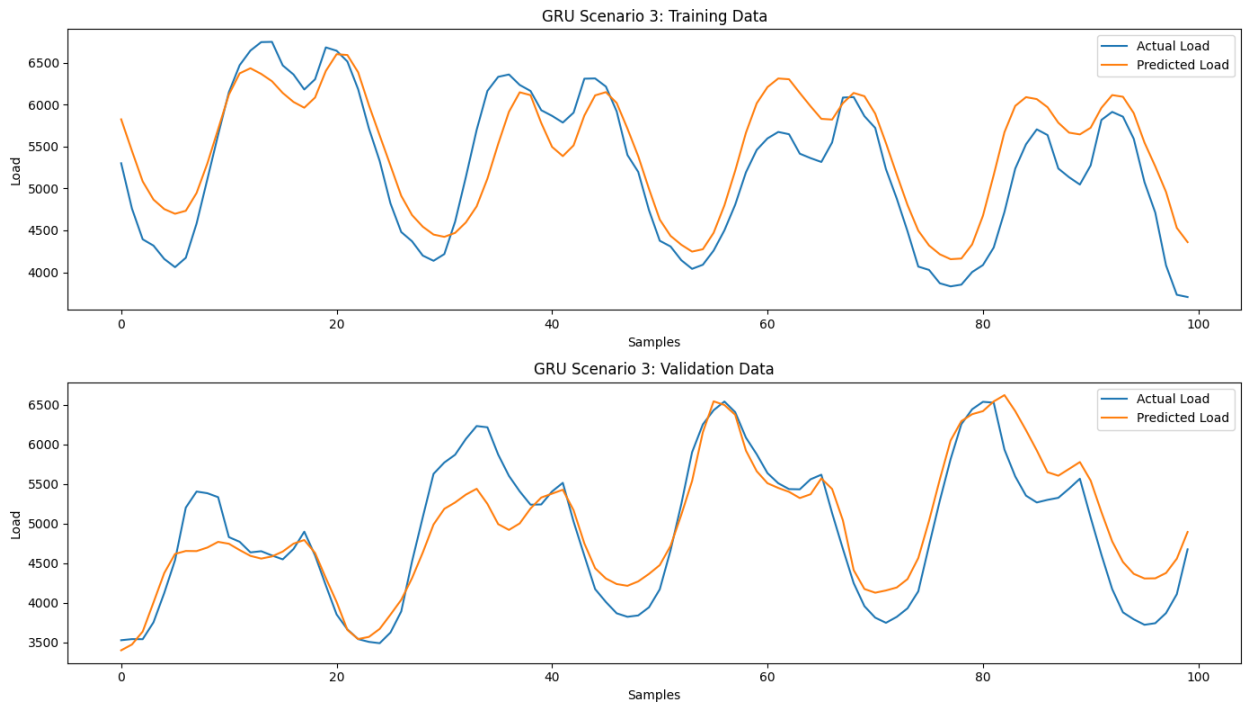


Figure 44: Actual vs. Predicted Load for GRU Scenario 3

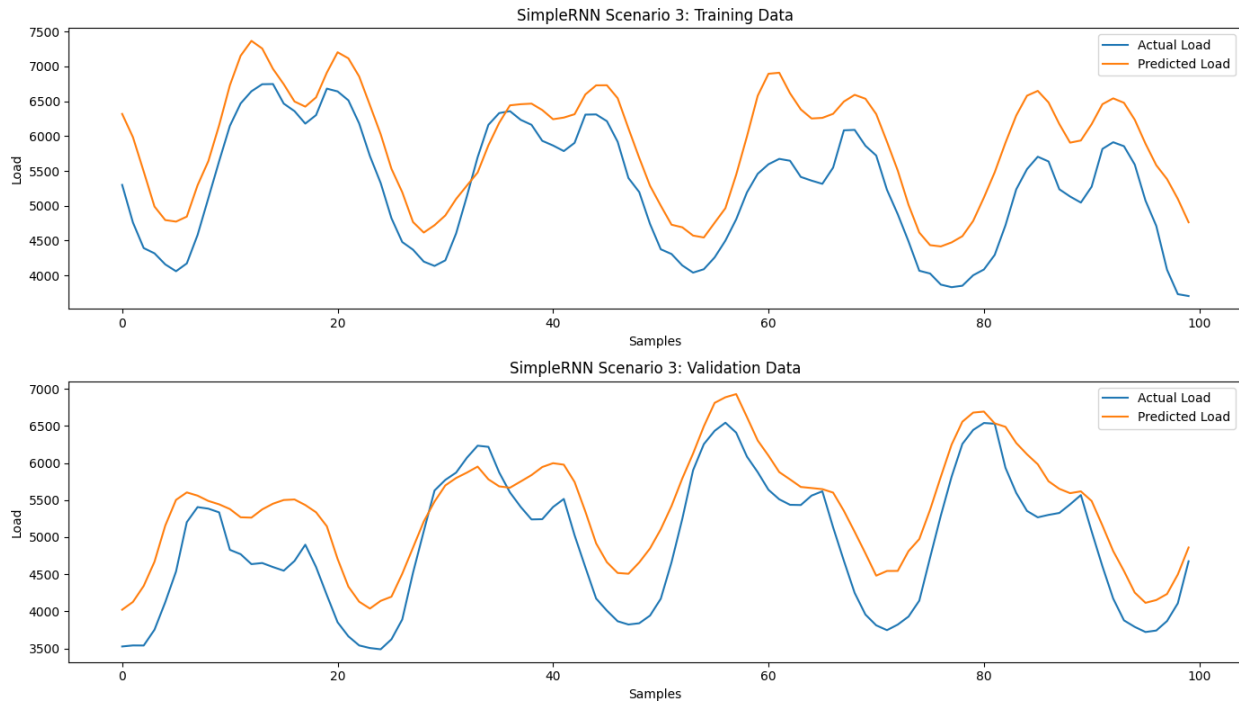


Figure 45: Actual vs. Predicted Load for SimpleRNN Scenario 3

For the LSTM model in Scenario 3, the best parameters that were found were hidden layers [128, 64], and dropout rate of 0.01, a learning rate of 0.001 and a batch size of 240, which gave it a validation RMSE of 508.09. The LSTM model showed good results with the validation  $R^2$  score of 0.86, which shows that it is efficient in the identification of load data patterns. The training and validation loss curves depicted a smooth declining pattern, which is a good sign of a well-learned model with little chances of overfitting. Comparing the actual and predicted load values, the LSTM model closely mimicked the actual load trends, thus proving the model's efficiency in load demand forecasting.

In this case, the GRU model was better than the LSTM model with the validation RMSE of 466.72 with hidden layers of 64 and 32 neurons, the dropout rate of 0.01, a learning rate of 0.005, and a batch size of 120. The validation  $R^2$  score was 0.89, which is an indication of the GRU model's ability to learn and capture temporal dependencies in the data better than the other models. The training and validation loss curves of the GRU model showed that the model was well trained and the model did not overfit or underfit the data. The actual vs. predicted load graphs for the GRU model were in good agreement with the actual load values, which confirmed the model's high accuracy.

The SimpleRNN model, with best parameters of hidden layers [128, 64], a dropout rate of 0.05, a learning rate of 0.1, and a batch size of 240, had the highest validation RMSE of 845.89 among the three RNN models. Although it achieved a validation  $R^2$  score of 0.62, indicating it captured significant patterns in the data, it was less effective compared to LSTM and GRU. The training and validation loss curves exhibited more fluctuations, suggesting that the SimpleRNN model struggled more with generalization compared to LSTM and GRU. The actual vs. predicted load plots for SimpleRNN displayed larger deviations and inconsistencies, highlighting its relative weakness in capturing complex temporal dependencies.



Comparing Scenario 3 with Scenario 1 reveals significant insights into the impact of longer input windows on model performance across different RNN architectures. For the LSTM model, Scenario 3, which uses a 96-hour input window, significantly improved performance over Scenario 1 with a 24-hour input window. The validation RMSE in Scenario 3 decreased to 508.09 from 573.17 in Scenario 1, and the validation R2 slightly improved to 0.86 from 0.83. These improvements indicate that the LSTM model benefits from the additional temporal data, allowing it to capture more intricate patterns and dependencies. The training and validation loss curves in Scenario 3 showed a steady decline, further suggesting a well-generalized model with minimal overfitting. The actual vs. predicted load graphs also confirmed the LSTM model's enhanced predictive capability with the extended input window, closely aligning with actual load values.

The GRU model demonstrated its superior adaptability to longer input windows, as evidenced by its performance in Scenario 3. The GRU model's validation RMSE in Scenario 3 was 466.72, an improvement from 485.32 in Scenario 1, and the validation R2 increased to 0.89 from 0.88. These metrics highlight the GRU model's exceptional ability to learn from longer historical data and capture temporal dependencies effectively. The training and validation loss curves for the GRU model exhibited stable training, indicating no signs of overfitting or underfitting. The actual vs. predicted load graphs showed that the GRU model's predictions were in close agreement with the actual load values, reflecting its high accuracy and robustness.

Conversely, the SimpleRNN model, despite having a longer input window in Scenario 3, struggled with performance. The validation RMSE increased to 845.89 from 601.06 in Scenario 1, and the validation R2 dropped to 0.62 from 0.81, indicating its limitations in handling extended input sequences. The training and validation loss curves showed fluctuations, and the actual vs. predicted load plots displayed larger deviations, underscoring the SimpleRNN's difficulty in generalizing with increased data complexity. These findings emphasize the necessity of advanced architectures like LSTM and GRU for leveraging longer input windows effectively in time-series forecasting.

## Section III : Conclusion

### 3.1 Observations

This thesis has aimed at comparing the performance of various neural network architectures for short-term electricity load forecasting with an emphasis on feed-forward neural networks and RNNs including LSTM and GRU models. The analysis was carried out across three distinct scenarios to evaluate the performance and robustness of each model configuration.

The feed-forward neural network model was compared in three different situations with different inputs and output forecasts. The first scenario which used a 24 hour sliding window to predict the load for the next hour was the best among the feed-forward models. This configuration was able to capture short-term temporal dependencies well, as evidenced by the low RMSE and high R<sup>2</sup> values, meaning that the model fit the data well without being overly complex. However, when the prediction horizon was increased

to 48 hours in the second scenario, the model's accuracy was not as good. The higher level of complexity and the longer forecast horizon made the model more unpredictable and thus produced higher RMSE and lower  $R^2$ . In Scenario 3, the addition of features that are highly correlated with the target variable like temperature and wind speed did not enhance the feed-forward model's performance. This implies that feed-forward networks may not be able to exploit more features and may even be hampered by them when the features introduce noise.

In contrast, recurrent neural networks demonstrated superior performance across different scenarios, especially in handling sequential data. The LSTM and GRU models excelled in capturing temporal dependencies and long-term patterns in the data. In Scenario 1, the GRU model outperformed both LSTM and SimpleRNN, showcasing its ability to handle short-term predictions effectively with lower RMSE and higher  $R^2$  scores. The GRU's simpler architecture compared to LSTM contributed to its robustness and efficiency in learning temporal patterns.

When the input window was extended to 48 hours and the prediction horizon was reduced to 1 hour in Scenario 2, the GRU model again exhibited the best performance. This scenario highlighted the GRU's capacity to model long-term dependencies and its adaptability to varying input lengths. The improved performance metrics underscored the importance of utilizing a longer input history for more accurate short-term predictions. On the other hand, the SimpleRNN model struggled with the increased input window size, leading to higher RMSE and lower  $R^2$ , indicating its limitations in handling complex temporal data compared to the GRU and LSTM models.

In Scenario 3, which involved a 96-hour input window and a 48-hour prediction horizon, along with the inclusion of high-correlation features, the RNN models, particularly GRU, continued to show significant improvements over shorter input windows. The GRU model achieved the best performance metrics, further validating its effectiveness in capturing intricate temporal dependencies. The LSTM model also benefitted from the extended input window, showing better performance than in shorter scenarios. However, the SimpleRNN model's performance declined, highlighting its inadequacy for complex and longer sequential data in comparison to the GRU and LSTM models.

## 3.2 Future Study

Future research should focus on several areas to enhance the predictive accuracy and robustness of these models. Firstly, further fine-tuning of model parameters using advanced optimization techniques such as Bayesian optimization or genetic algorithms could yield better results. Secondly, exploring ensemble methods that combine multiple models, such as GRU and LSTM, might leverage the strengths of different architectures and improve overall performance. Thirdly, incorporating additional influential factors like economic indicators, social activities, and more granular weather data could provide more comprehensive insights and improve model accuracy.

Furthermore, the models that can be used with real-time data and give the load forecast in real-time could be useful for operational energy management. Expanding the prediction horizon from the short-term to the weekly or monthly level would be useful for energy planning. Using these models in

other areas or countries with different climatic and social-economic conditions may assess the transferability of the models.

In conclusion, this thesis demonstrates the superior performance of GRU models in short-term electricity load forecasting, particularly when leveraging longer input histories and capturing temporal dependencies. The insights gained from this research provide a solid foundation for further advancements in the field of energy demand forecasting, ultimately contributing to more efficient and reliable energy management systems. The comparison of feed-forward and recurrent models highlights the critical role of temporal modeling in improving forecasting accuracy and underscores the need for advanced architectures in handling sequential data.

## References

1. Assis, A. K. T. (2010). The experimental and historical foundations of electricity, Apeiron Montreal.
2. Forrester, R. (2016). "History of electricity." Available at SSRN 2876929.
3. Neidhofer, G. (2007). "Early three-phase power [History]." IEEE Power and Energy Magazine 5(5): 88-100.
4. Rizwan, M., et al. (2023). Evaluation and Characterization of Power Generation Trainer EM-PRT-EG3 $\phi$  for Laboratory Education. 2023 International Conference on Energy, Power, Environment, Control, and Computing (ICEPECC), IEEE.
5. Kumar, S., et al. (2020). "Reliability enhancement of electrical power system including impacts of renewable energy sources: a comprehensive review." IET Generation, Transmission & Distribution 14(10): 1799-1815.
6. Emmanuel, M. and R. Rayudu (2017). "Evolution of dispatchable photovoltaic system integration with the electric power network for smart grid applications: A review." Renewable and Sustainable energy reviews 67: 207-224.
7. Setlak, L. and R. Kowalik (2017). Modern technological solutions in generation, transmission and distribution of electricity in "conventional" vs. "More Electric" Aircrafts. 2017 Progress in Applied Electrical Engineering (PAEE), IEEE.
8. Fan, S. and R. J. Hyndman (2012). Forecasting electricity demand in Australian national electricity market. 2012 IEEE Power and Energy Society General Meeting, IEEE.
9. Hsu, Y.-Y. and C.-C. Yang (1995). "Electrical load forecasting." Applications of Neural Networks: 157-189.
10. Khan, I. (2021). "Household factors and electrical peak demand: a review for further assessment." Advances in Building Energy Research 15(4): 409-441.
11. Mohan, N., et al. (2018). "A data-driven strategy for short-term electric load forecasting using dynamic mode decomposition model." Applied energy 232: 229-244.
12. Han, L., et al. (2018). "Enhanced deep networks for short-term and medium-term load forecasting." IEEE Access 7: 4045-4055.
13. Haq, M. R. and Z. Ni (2019). "A new hybrid model for short-term electricity load forecasting." IEEE Access 7: 125413-125423.
14. Bendaoud, N. M. M. and N. Farah (2020). "Using deep learning for short-term load forecasting." Neural computing and applications 32(18): 15029-15041.
15. Zhang, X.-S. and X.-S. Zhang (2000). "Introduction to artificial neural network." Neural Networks in Optimization: 83-93.
16. Rosa, J. L. G. (2013). Biologically plausible artificial neural networks. Artificial Neural Networks- Architectures and Applications, IntechOpen.
17. Pircher, T., et al. (2021). "The structure dilemma in biological and artificial neural networks." Scientific Reports 11(1): 5621.
18. Dayhoff, J. E. and J. M. DeLeo (2001). "Artificial neural networks: opening the black box." Cancer: Interdisciplinary International Journal of the American Cancer Society 91(S8): 1615-1635.
19. Park, W. J. and J.-B. Park (2018). "History and application of artificial neural networks in dentistry." European journal of dentistry 12(04): 594-601.

20. Yadav, N., et al. (2015). "History of neural networks." An introduction to neural network methods for differential equations: 13-15.
21. Wu, Y.-c. and J.-w. Feng (2018). "Development and application of artificial neural network." *Wireless Personal Communications* 102: 1645-1656.
22. Bailer-Jones, C. A., et al. (2001). "An introduction to artificial neural networks." arXiv preprint astro-ph/0102224.
23. Khan, A., et al. (2020). "A survey of the recent architectures of deep convolutional neural networks." *Artificial intelligence review* 53: 5455-5516.
24. Sharkawy, A.-N. (2020). "Principle of neural network and its main types." *Journal of Advances in Applied & Computational Mathematics* 7: 8-19.
25. Pedomonti, D. (2018). "Comparison of non-linear activation functions for deep neural networks on MNIST classification task." arXiv preprint arXiv:1804.02763.
26. Misra, D. (2019). "Mish: A self regularized non-monotonic activation function." arXiv preprint arXiv:1908.08681.
27. Ma, Y., et al. (2018). Random projection recurrent neural networks for time series classification. 2018 8th International Conference on Electronics Information and Emergency Communication (ICEIEC), IEEE.
28. Orhan, A. E. and X. Pitkow (2019). "Improved memory in recurrent neural networks with sequential non-normal dynamics." arXiv preprint arXiv:1905.13715.
29. Nugaliyadde, A., et al. (2019). Language modeling through Long-Term memory network. 2019 international joint conference on neural networks (IJCNN), IEEE.
30. Staudemeyer, R. C. and E. R. Morris (2019). "Understanding LSTM--a tutorial into long short-term memory recurrent neural networks." arXiv preprint arXiv:1909.09586
31. Fischer, T. and C. Krauss (2018). "Deep learning with long short-term memory networks for financial market predictions." *European journal of operational research* 270(2): 654-669.
32. Gers, F. A., et al. (2002). "Learning precise timing with LSTM recurrent networks." *Journal of machine learning research* 3(Aug): 115-143.
33. Yang, S., et al. (2020). Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example. 2020 International workshop on electronic communication and artificial intelligence (IWECAI), IEEE.
34. Shewalkar, A., et al. (2019). "Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU." *Journal of Artificial Intelligence and Soft Computing Research* 9(4): 235-245.
35. Mienye, I. D., et al. (2020). "Improved sparse autoencoder based artificial neural network approach for prediction of heart disease." *Informatics in Medicine Unlocked* 18: 100307.
36. Yaghini, M., et al. (2013). "A hybrid algorithm for artificial neural network training." *Engineering Applications of Artificial Intelligence* 26(1): 293-301.
37. Zhang, Y., et al. (2016). Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. International conference on machine learning, PMLR.
38. Sridhar, S. and M. Ponnaivaikko (2011). "Improved adaptive learning algorithm for constructive neural networks." *International Journal of Computer and Electrical Engineering* 3(1): 30-36.
39. Adnan, R. M., et al. (2017). "Streamflow forecasting using artificial neural network and support vector machine models." *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)* 29(1): 286-294.

40. Bee Dagum, E., et al. (2016). "Time series components." *Seasonal Adjustment Methods and Real Time Trend-Cycle Estimation*: 29-57.
41. Heij, C., et al. (2021). "Cycles and Trends." *Introduction to Mathematical Systems Theory: Discrete Time Linear Systems, Control and Identification*: 157-176.
42. Dudek, G. (2023). "STD: a seasonal-trend-dispersion decomposition of time series." *IEEE Transactions on Knowledge and Data Engineering* 35(10): 10339-10350.
43. Sulandari, W. and H. Utami (2017). Forecasting time series with trend and seasonal patterns based on SSA. 2017 3rd International Conference on Science in Information Technology (ICSITech), IEEE.
44. Zhang, Z. and J. C. Moore (2015). "Chapter 8-autoregressive moving average models." *Mathematical and Physical Fundamentals of Climate Change* (239-290). Boston: Elsevier.
45. Momin, B. and G. Chavan (2018). "Univariate time series models for forecasting stationary and non-stationary data: A brief review." *Information and Communication Technology for Intelligent Systems (ICTIS 2017)-Volume 2 2*: 219-226.
46. Scripts used for analysis and modeling in this thesis. Available at: [URL](#)