Vulnerability Assessment of Machine Learning Based Short-Term Residential Load Forecast against Cyber Attacks on Smart Meters

by

Alanoud Alrasheedi

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Sustainable Energy

Carleton University
Ottawa, Ontario

Abstract

Short-term load forecast at residential house level plays a critical role in home energy management system. While a variety of machine learning based load forecasting methods have been proposed, their prediction performance have not been assessed against cyber threats on smart meters which have been increasingly reported. This thesis investigates the vulnerability of four extensively used machine learning algorithms for residential short-term load forecast against cyberattacks, including nonlinear auto regression with external input (NARX) neural network, support vector machine (SVM), decision tree (DT), and long-short-term memory (LSTM) deep learning. We use two datasets with different data sampling rates, including the REFIT dataset which collected whole-house aggregated loads at 8-second intervals continuously from 20 houses over a two-year period in the U.K, and the AMPds dataset which collected hourly load of one residential house in Canada for one year. The predication performances on these two datasets are compared by using NARX, SVM, DT, and LSTM. Four cyberattack models are investigated, including pulse attack, scale attack, ramp attack, and random attack. The vulnerability assessment results indicate the LSTM provides the most accurate prediction without cyberattacks. However, the prediction accuracy of the LSTM fluctuates when there are cyberattacks. Among the four cyberattacks, the random attack triggered the largest variations on the predication results.

Keywords— Decision tree, Short-term Load prediction, Support vector machine, Time series forecasting, Nonlinear Auto Regression with external input (NARX) neural network.

Acknowledgments

I would like to express my deepest appreciation to my supervisor, Dr Shichao Liu, for his precious patience and support. Dr Shichao's valuable beyond estimation encouragement and continuous feedback provided me with tremendous guidance throughout my master's journey. Throughout the harder times in the start of my journey during the outbreak of the Covid-19 crisis Dr. Shichao Liu was prominently present. He made it his priority to continuously check up on me and provided me with extensive personal and professional guidance. His critique, assistance and dedicated involvement throughout this process made this paper possible.

I am grateful to all of those with whom I have had the pleasure to work with and learn from during this study.

Most importantly, I could not have undertaken this journey without my parents and siblings. Their belief in me has maintained my high motivation and spirit.

Lastly, I would like to extend my sincere thanks to Carleton University for their space, inclusivity, and resources.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

This chapter introduces the motivation for this thesis work, gaps of related existing work, thesis objectives and the organization of this paper. The motivation of this study is discussed in Section 1.1. The gaps of existing work in this field are introduced in Section 1.2. The thesis objectives are listed in Section 1.3. The thesis organization in Section 1.4 lists how each chapter is divided.

1.1 Motivation

Energy management is an essential part of energy usage in all industries that utilize energy for power production. Home energy management systems (HEMS) exist because energy consumption within houses differ from that of a bigger commercial or industrial building [1]. Home energy management systems are an ever-growing technology that is centered around smart grids used within homes [2]. These systems consist of load management to adjust energy demands, this adds value to the homeowner and the utility as it will save money and energy. The system can communicate with the monitoring and control devices found within the home.

Demand response (DR) programs are used by utilities to send signals to the HEMS to create schedules based on the signals, the system goals, and the homeowner's priorities. Machine learning algorithms need to be used to develop an understanding of the pattern of energy consumption of the homeowners, later adjustments can be made to save energy and reduce peak demands.

Demand side management (DSM) plays a critical role in active electrical distribution system planning and operation, such as ensuring low-cost power supply, postponing new generation constructions, and satisfying clients' load demand through electricity conservation and

change of power consumption pattens [3]. Accuracy and correctness of short-term load forecast on residential household level is one of the key requirements for a high-performance DSM system.

This thesis targets on machine learning -based load predictions. Having the knowledge of how much energy is used and predicting energy future demand can help energy producers pinpoint exactly where their energy problems are so they can achieve their predicted load target. Predicting the accurate amount of generation can minimize over generation and under generation [4]. Knowing when and how much electricity is needed is important to infrastructure, network, and availability. Load prediction is vital for electricity production, as electricity demand is linearly rising day by day. Electrical power consumption is ever changing due to the various effects of environment and human behavior [5]. This change is random depending on social, environmental, time, and economical changes. This makes it very complex for energy producers to predict and analyze future energy demand. This project is focused on practicing load prediction through various methods of machine learning algorithms. Since electricity cannot be stored, machine learning algorithms are explored to predict load demand and how much electricity is needed for the days to come. The forecasting allows for demand knowledge and precaution to be set by utility companies in their operation and management of electricity power supply [4]. The prediction algorithms can be used for data that can range from long, medium- and short-term forecasting [5]. There are two approaches for the demand forecasting models, such as the horizon approach and the purpose of the forecasting approach.

The horizon approach is usually divided into three forms:

- Long term predictions range from 20 years to 1-year, medium term predictions range from 1 week to 1 year and short-term predictions range from 1 hour to 1 week [6]. Long

term predictions are used to analyze economic effects and planning of new future generation and its transmission [7].

- Medium term predictions are used to schedule refueling, maintenance planning, financial planning, and formulation of tariff implied by the government [6].
- Short term predictions are used to analyze when to start and end the generation of electricity, spinning reserve planning, transmission constraints and system security assessment [6].

The purpose of the forecasting approach is divided into two forms:

- Single Output Forecast is used to output one possible value which predicts either next hour's load, next day's peak load etc.
- Multiple Output Forecast is used to output multiple values for different objectives such as one output for next hour's load with another output for another.

When accurate forecasting is produced, this allows for major economical savings in operating and maintenance costs, increased security in the supply of power and delivery systems [5]. Consumers are also affected by load predictions. Inaccurate load prediction may add fluctuation in the electricity costs, and this will negatively impact consumers [4]. In power plants, predicting the accurate amount of generation can minimize over generation and under generation [4]. The location of the power plant can also be determined using prediction forecasting. The utilities can choose the location and size of the power plant depending on the load prediction; for example, if demand is high in a region the power plant can be located closer to the load [4]. This minimizes the transmission and distribution infrastructure and any of their energy losses [4]. The applications that use load forecasting are capacity planning, network planning, generation and

transmission capital investment planning, financial forecast, efficient power procurement, selling excess power, renewable planning, and fuel mix selection planning [6].

1.2 Limitations

Electricity distribution is done through advanced metering infrastructure [8]. A system that incorporates smart meters, communications network and data management is called an advanced metering infrastructure [8]. This system offers energy consumers advanced functionality while being safe, reliable, and efficient [8].

The evolution of load forecasting comes with its own cyber security issues. Advanced metering infrastructure faces many challenges that rise from unknown cyber-attacks. The likely impacts of cyber-attacks are theft of data, theft of power, localized denial of power, widespread denial of power, and disruption of grid [8].

Theft of data attacks can be done through attacking smart meters by unauthorized monitoring of data in transit or at rest in a meter or data collector [8]. Theft of data is usually used to determine patterns in residences to rule out whether the residence is occupied or not [8]. Reverse engineering attacks may take place when data is stolen through the capture of infrastructure configuration information [8].

Theft of power attacks can be used by customers and producers [8]. Theft of power is commonly used by customers who fail to pay the electricity charges and try to obtain electricity free of charge [8]. Reconnection of power by unpaying customers is another form of power theft [8]. Inflated power bills causing customers to pay above their actual usage is a power theft from the producer side [8].

Denial of power attacks are commonly known as localized and widespread denial of power. Localized denial of power occurs when power is turned off for a customer or a group of customers by either permanently disabling or disconnecting their smart meter [8]. Moreover, Widespread denial of power is concentrated on thousands or millions of smart meter disconnections through a single attack [8]. This is problematic as it will cause major power insecurity to large areas and fixing the issue may take months [8].

Disruption of grid attacks can happen through the disconnection and reconnection of smart meters that causes rapid instability in the power grid [8]. This mischievous control of enormous number of smart meters has widespread power impacts as it affects the bulk power grid [8].

These issues are resolved by first identifying the behaviour of load forecasts tampered with by cyber adversaries. Identifying the issue will minimize the challenge of facing cyberattacks later after the mitigation of the data is done [9]. This paper investigates the vulnerability of four extensively used machine learning algorithms for residential short-term load forecast against cyberattacks, including Nonlinear Auto Regression with external input (NARX) neural network, support vector machine (SVM), decision tree (DT), and long-short-term memory (LSTM) deep learning. Two cases of cyberattack models were determined. The first case is by using each attack model separately to the original data. The second case is by using all the four attack models together to the original data. Two data sets were used in this study the REFIT and the Canada datasets. The REFIT dataset [10] is used for the analysis and this dataset collected whole-house aggregated loads at 8-second intervals continuously from 20 houses over a two-year period in the U.K. The Canadian data set AMPds [11] were sampled in hours from 6th of March 2016 to 7th of May 2016.

1.3 Thesis Objectives

In this work, we assessed four representative machine learning algorithms extensively used in residential short-term load forecasting. The four machine learning methods investigated in this work include Nonlinear autoregression with external input (NARX), SVM, Decision Tree, and LSTM. The REFIT dataset [10] is used for the analysis and this dataset collected whole-house aggregated loads at 8-second intervals continuously from 20 houses over a two-year period in the U.K.

To further test the capability of each method, their error propagation under different cyberattacks were also tested. Two cases of cyberattack models were determined. The first case is by using each attack model separately to the original data. The second case is by using all the four attack models together to the original data.

The entire training process is repeated for a different dataset which is the AMPds dataset [11]. The second dataset is based on Canadian values specifically in Burnaby, British Colombia, Canada [11]. The house ID used from the set of data was House ID: 1, this house is a residential single detached house with rental suite. The data chosen to train were from 6th of March 2016 to 7th of May 2016. The AMPds data were sampled in hours making it a lot smaller in size compared to the REFIT dataset. This dataset was chosen as another case study to compare and help analyze the data to justify the differences between UK and Canada. Impact of time on the update frequency is explored between the REFIT and AMPds datasets.

The following peer-reviewed conference paper is recently accepted.

Alanoud S O F A SH Alrasheedi, Shichao Liu, Osarodion Edgomwan, and Nowayer Alrashidi, Vulnerability Assessment of Machine Learning Based Short-Term Residential Load Forecast against Cyber Attacks on Smart Meters, 2022 IEEE International Conference on

Communications, Control, and Computing Technologies for Smart Grids (IEEE SmartGridComm 2022), in-person and virtual, Singapore, October 25-28, 2022

1.4 Thesis Organization

This thesis is organized into five different chapters.

Chapter 1 is focused on introducing the objectives, knowledge gaps, and motivation behind this paper. The motivation of this study is an introduction to define the purpose of this study. The limitations of existing literature are discussed to provide an overview of problems that are related to this study and the solutions that have been drawn out because of this paper. The thesis objectives summarize the steps taken to obtain the results and conclusion of this study. The thesis organization lists how each chapter is divided.

Chapter 2 consists of a literature review of machine learning algorithms, deep learning, and the way the data is preprocessed. This chapter is important to introduce machine learning algorithms and their different types to lay a foundational understanding of the base technology used in this study. Deep learning is explained as it is an important subset of machine learning. Data preprocessing is an essential step to discuss in this paper, as it is used to clean the data before training them.

Chapter 3 dives deep into the different types of machine learning algorithms and explains how their architecture works to produce desired results. Single and multilayer feed-forward neural network, feedback, recurrent, cascaded, and artificial neural networks architectures are reviewed. The methods NARX, LSTM, SVM and decision tree are also explained in chapter 3 as they are the methods being used in the training of the data used in this paper.

Chapter 4 is the data implementation and testing, all the results are analyzed and explained in this chapter. The two datasets used in this paper are explained. The parameters that were used for evaluating and quantifying the performance measure of the forecasting machine learning algorithm are listed.

Chapter 5 is vulnerability assessment of load predictions under different cyberattacks. In this chapter, the results are analyzed and explained. There are two cases of cyberattacks used to analyze the data; the first case is to use different cyberattacks alone and the second case is to use the cyberattacks all together at the same time.

In the conclusions, the results are concluded, and future work is discussed.

Chapter 2: Literature Review

This chapter briefly reviews machine learning based load prediction in the existing literature. Deep learning is explained, and data preprocessing is explored to understand how the data was filtered before the training process. Machine learning algorithms for load prediction is discussed in Section 2.1, Deep learning is introduced in Section 2.2 and Data preprocessing is explored in Section 2.3.

2.1 Machine Learning Algorithms for Load Prediction

Accuracy and correctness of short-term load forecast on residential household level is one of the key requirements for a high-performance DSM system.

Three main group of methods have been introduced for load forecasting, namely time series models, artificial intelligence models and hybrid models. Linear regression [12], seasonal autoregressive [13], threshold autoregressive [14], Kalman filtering [15], autoregressive integrate moving average (ARIMA) [16] and seasonal autoregressive integrate moving average (SARIMA) are the most common time series models that are utilized for electricity load forecasting purpose in different articles. However, the mentioned methods are not performed well in the presence of uncertainties. As a result, artificial intelligence base methods are proposed to forecast the future load of power systems in the presence of uncertainties. Support vector machines (SVM) [17], support vector regression (SVR) [18], fuzzy logic [19], artificial neural networks [20] are the example of AI based methods. Recently, hybrid models are developed for improving the forecasting accuracy. These models have two main categories. For the first category model, electricity load was predicted individually by different models such as back propagation neural network (BPNN) [21], genetic algorithm back propagation neural network (GABPNN) [22],

ARIMA and support vector machine (SVM). Then, the weight of each model was calculated by a suitable method. The final forecasting value was the sum of each model's forecasting value multiplied by its weight. In paper [23], the authors developed a load prediction model combined with regression trees, neural networks, and support vector regression. Adaptive neuro-fuzzy inference system (ANFIS) and SARIMA for load forecasting is proposed in [24]. Reference [25] suggested a deep learning-based model combined with support vector regression method to forecast electricity load. Recently, load prediction using Long Short-Term Memory (LSTM) neural network was conducted in [26].

While these load prediction methods are based on data collected by smart meters, advanced metering infrastructures (AMIs) are found vulnerable to cyberattacks. Based on the U.S. Federal Bureau of Investigation (FBI), a well-coordinated energy theft attempts against AMI which could cost a utility company up to $400 M annually in 2009 [27]. Several extensively used load forecasting models have been failed to offer accurate predictions under cyberattacks [28]. Though there have been increasing efforts to secure load forecasting, such as anomaly detection [29], [30], real-time data injection detection [31], vulnerability analysis [32], they are on grid-side operation and ignore the cyber threats on load forecasts at residential house level.

Machine learning is a type of artificial intelligence that is centered around predictive analysis and modeling [33]. Machine learning algorithms branches into four different types, supervised, semi-supervised, unsupervised and reinforcement learning [33], [34].

- Supervised learning algorithm works through the act of teaching [33], [35]. This occurs when the user provides the computer with an accurate known input and desired output [33], [35], [36]. The computer then studies the relationship and pattern between the given input and the target prediction output [33], [35], [36]. When the algorithm

interprets the prediction, the operator then corrects it until the algorithm accuracy is at a high level of accuracy and performance [33], [35]. There are three different tasks that work within the supervised learning algorithm called classification, regression, and forecasting [33]. Classification tasks are when the computer program is expected to conclude which category the resulting output belongs to. The program observes present inputs to process which labels should be used for the resulting outputs [33], [36]. The regression task is applicable in cases where the relationship between dependent and independent variables is unclear [33], [36]. The estimation between variables is proven to be useful in cases of prediction and forecasting [33], [36]. Forecasting is proportional to future predictions resulting from past and present data [33]. There are many challenges that come with this algorithm, such as training can be time consuming [36]. Human error is more likely to occur in datasets using this algorithm as it requires experienced operators [36]. Compared to the unsupervised algorithm, this algorithm cannot classify given data into their defined categories [36].

- Semi-supervised learning algorithm is almost identical to supervised learning, however, this algorithm uses both unlabeled and labeled data [33]. This algorithm uses labeled data to label unlabeled data [33].
- Unsupervised learning algorithm, the program is trained with unlabeled data to detect unknown patterns [33]. This algorithm is useful to the user by providing comprehensive and descriptive outputs to the unlabeled data [35]. There are two different tasks that work within the unsupervised learning algorithm called clustering, and dimension reduction [33]. Clustering is the act of grouping similar data within their defined

categories to later analyze them [33]. Dimension reduction is the act of reducing variables to refine the information and get the exact information needed [33].

- Reinforcement learning algorithm works like a game, where there are rewards and penalties for positive and negative learning outcomes [37], [38]. The rules are given by the user and the program is expected to perform the tasks by undergoing random trials to come up with the maximum reward outcome [37], [38]. Those trials are part of the computer's exploration of possibilities, monitoring and evaluating results [33]. Then through the penalty and reward evaluation the maximum reward possibility is chosen [33], [37], [38]. Thus, the machine develops trial and error understanding to further apply it later [33], [37], [38].

2.2 Deep Learning

Deep learning is a subset of machine learning which is a subset of artificial intelligence [41]. Artificial intelligence imitates the intelligence of humans or living entities [41]. While machine learning is a technique used by computers to learn from given data by training using different algorithms [41]. Deep learning provides the computational ability for models with multiple processing layers to learn representations of data with multiple levels of abstraction [39]. The word deep is used in deep learning because of the use of multiple layers in the network [39]. When compared to other technologies deep learning is exemplary at discovering intricate structures in high-dimensional data [39]. Through the use of various hidden layers, heterogeneous layers are acceptable, as well as an unbounded number of layers within the bounded size [41]. This technology uses backpropagation algorithm to analyze large data sets [39]. Deep learning in neural networks can be used in a supervised and unsupervised manner as discussed in section 2.1. To

develop a deep neural network model for predictions supervised learning is used and to develop a deep auto-encoder model for feature extraction unsupervised learning is used [40]. In this paper, deep learning is analyzed using real world data through various training and under various conditions. Deep learning technologies are vulnerable to deception and cyber-attacks, identifying patterns will help eliminate unwanted attacks.

2.3 Data Preprocessing

The process of load prediction requires careful data preprocessing to transform data into clear usable inputs [42]. This data optimizing step branches into multiple sessions of data preprocessing which are removing missing values, removing outliers, data normalization and data division.

Removing missing values methods help improve the accuracy of the forecast [42]. Removing a row of data where values may be missing within this row is one form of removing missing values [42]. Also, compiling the mean of all values and then filling in the missing value using the mean [42]. Depending on the data used, forecasting algorithms may also be employed to predict the value of the missing value based on the current data [42].

Outliers are data inputs that vary significantly when compared to the other inputs within the dataset [43]. These kinds of inputs are often a result of public holidays where load differentiates greatly compared to normal day to day uses [42], [43]. The removal of such inputs is important to ensure the concluded forecast is reliable for normal days predictions [42], [43]. Hence, the Z-score technique can be used to remove outliers, this can be seen in equation 1 [42].

$$Z = (x - \mu) / (\sigma) \text{ [42]} \quad (1)$$

Here, Z is Z-score, μ is the average, σ is standard deviation from the mean and x is the sample in the data. Equation 1 is used on every sample in the data, then a threshold is set to remove outliers [42]. This technique is used when the data does not contain extreme outliers [44].

To further improve the accuracy of the data, two normalization methods are introduced for preferred outputs. The extremum method or linear scaling and the standard deviation method.

- Extremum method or linear scaling consists of normalizing the dataset by scaling to a range using the following equation,

$$x' = (x - xmin) / (xmax - xmin) \text{ [44]} \quad (2)$$

Here, x' is the normalized data, x is the actual data point, x_{max} is the data point at the maximum in the time series and x_{min} is the data point at the minimum. Data normalization ensures stability of weight and hyperparameters convergence. This is important as unnormalized data can lead to an impaired network [42]. Equation 2 can only be used if the approximate upper and lower bounds are known with few or no outliers [44]. In addition, it can only be used if the data is approximately in uniform distribution across the range [44].

- Standard deviation method is used when the range of the dataset exceeds 0 and 1 [45]. There are three equations associated with this method which are,

$$Normalized\ (e_i) = e_i / std(E) \text{ [45]} \quad (3)$$

$$std\ (E) = \sqrt{\frac{1}{(n-1)}} \sum_{i=1}^{n} (e_i - \bar{E})^2 \text{ [45]} \quad (4)$$

$$\bar{E} = \frac{1}{n} \sum_{i=1}^{n} e_i \text{ [45]} \quad (5)$$

Equation 3 is the normalizing equation where std is calculated in equation 4 and E' is calculated in equation 5. E is the row, i is the column and if row E values were identical then std(E) is zero, hence all values for row E will also be zero [45].

The REFIT dataset is the main dataset which collected whole-house aggregated loads at 8-second intervals continuously from 20 houses over a two-year period in the U.K. The main dataset used within the different forecasting methods tested in this paper has been divided accordingly for simpler and efficient usage. The inputs used to predict September 23, 24, 25 of 2013 were from September 20, 21, 22 of 2013. These inputs were used to predict the next day's prediction and then compare it to the actual data of September 23, 24, 25 which are the next days.

The second dataset is based on Canadian values specifically in Burnaby, British Colombia, Canada [11]. The house ID used from the set of data was House ID: 1, this house is a residential single detached house with rental suite. The data chosen to train were from 6th of March 2016 to 7th of May 2016. The AMPds data were sampled in hours, meaning there is an average of all the measurements within that hour considering the load is constant. This makes the data low resolution compared to the REFIT UK dataset.

Chapter 3: Machine Learning Algorithms for Residential Load Forecast

This chapter introduces the four extensively used machine learning algorithms, including Nonlinear Auto Regression with external input (NARX) neural network, support vector machine (SVM), decision tree (DT), and long-short-term memory (LSTM) deep learning.

3.1 Artificial Neural Network

Artificial neural networks or ANN span from the biological makeup of the human body. This is closely correlated with the human brain as its functionality resemblant to that of an artificial neural network. The imitation of the human brain is common within the electric and software sector to solve computational problems. ANN is formulated from a group of interconnected neurons working together to solve a problem through learning by example [46]. Usually, ANN cannot be programmed to solve in a particular manner, this is what makes ANN special when compared to conventional computer algorithms [46]. Although artificial neural network methodology is inspired by the human brain, ultimately, this technology is greatly more advanced. Artificial neural networks can now derive complex patterns that humans and other computational methods can never achieve [46]. Artificial neural networks have the ability to learn how to perform a task based on the data imputed within the training session [46]. Classifying the information received in different categories is another advantage of ANN [46]. Operations within this form of technology are often in real time and carried out in parallel [46].

3.1.1 Architecture of Biological Neurons

To understand the similarity between artificial neural networks and the human brain, it is important to understand how biological neurons work.

Neurons as seen on Figure 3.1.1 are cells that make up part of the nervous system in the human body [47]. Their main function is to transmit information between the brain and the nervous system.

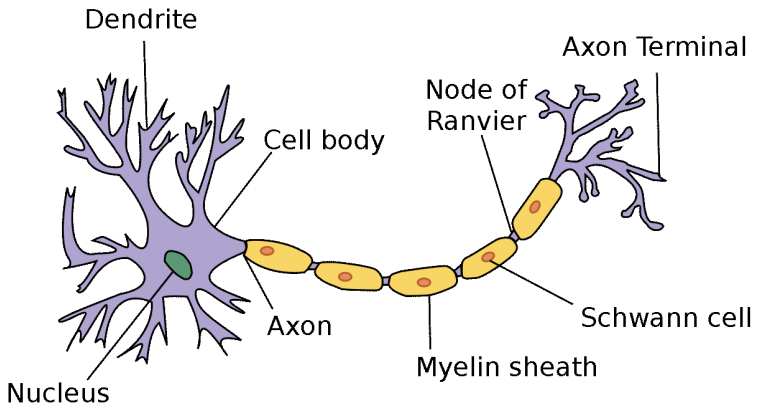


Figure 3.1.3 Human neuron cell architecture [48]

As seen on Figure 3.1.1 the dendrites receive messages for the neuron cell and the axon transmits messages from the cell [47]. The nucleus shown in Figure 3.1.1 controls the neuron's activities [47]. To communicate with other neurons chemicals are sent through a small space called a synapse from the axon of a current neuron to the dendrite of the adjacent neuron [47]. These chemicals are spikes of electrical activities that are then converted into electrical effects within the receiving neuron [48]. In the human body change of neuron activity occurs when the effectiveness of the synapses is changed which eventually changes the effect of one neuron on another [48]. This change is often associated with learning [48].

3.1.2 Architecture of Artificial Neurons

The traditional unit or element of computation in ANN is the neuron [49]. A simple artificial neuron is a device with inputs and outputs [46]. Usually, these neurons have training mode and using mode [46]. The neuron is trained to fire or not fire in the training mode depending on the pattern detected from the inputs [46]. After the input pattern is taught and detected, the output desired becomes the present output, this happens in the using mode [46]. This is how a simple artificial neuron works.

ANN must follow firing rules to maintain their popular high flexibility [46]. These rules consist of how to calculate whether a neuron should fire or not for any given input patterns [46]. There are many firing rules being used in ANN such as the Hamming distance technique [46]. This technique works by using a set of taught patterns annotated as '1' and a set of not taught patterns annotated as '0'. In this case the taught patterns '1' will cause the node to fire, while the not taught patterns will not [46]. In addition, the pattern not in the collection will compare itself to the taught and not taught patterns [46]. As a result of the comparison if the patterns not in the collection have more input elements in common with the taught patterns than the not taught patterns this results in triggering the node to fire [46]. Finally, if there is a tie when the comparison is done then the pattern not in the collection remains undefined [46].

To produce advanced results more complex neurons through various models such McCulloch and Pitts model or MCP, the Delta rule and the back error propagation [46]. In the MCP model the inputs are weighted, this provides the sensitivity of each neuron weight change [46]. The neuron weight is directly related to the decision-making process [46]. A threshold value is set to determine when to fire the neuron [46]. The neuron only fires if the weighted inputs exceed the threshold value [46]. Figure 3.1.2.1 shows the simplified structure of an MCP neuron.

As shown in Figure 3.1.2.1, the inputs and their weight are multiplied to calculate total weightage and then they are compared with the pre-set threshold. Equation 1 below determine when the neuron is fired or not fired.

$$X1W1 + X2W2 + X3W3 + \ldots > T \text{ [46]} \quad (6)$$

Equation 6 shows the terms set to determine when the neuron is fired or not fired. This complex neuron model is important as it allows for changing weights and thresholds to satisfy desired outcomes and work with different situations [46].

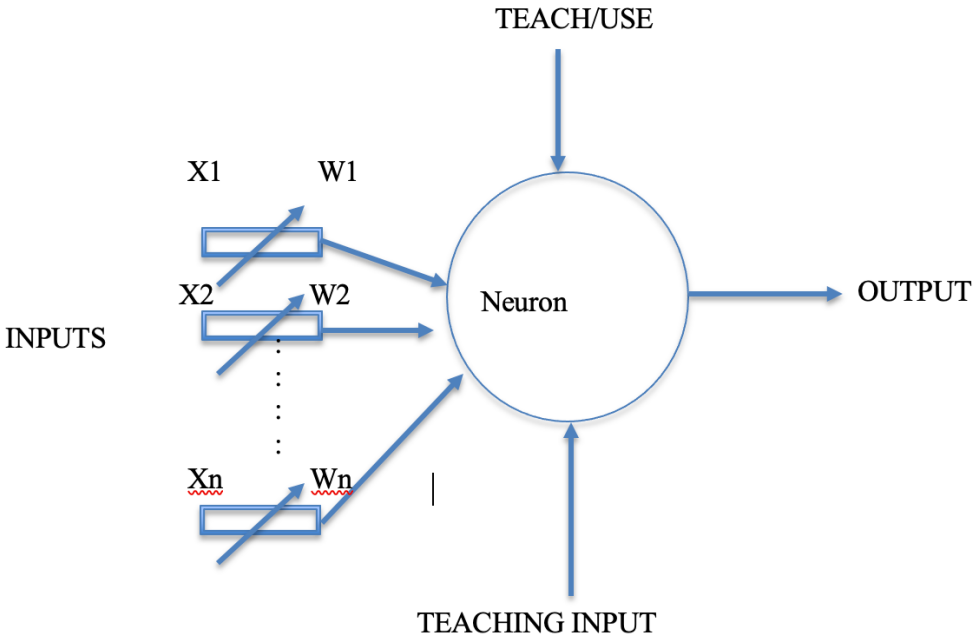


Figure 3.1.4.1 modified diagram of MCP Neuron Structure [46]

The way ANN processing elements are connected is called their interconnection [49]. The interconnection arrangement of ANN elements commonly has three layers which are the input layer, the output layer, and the hidden layer [50]. The network interconnections transfer the output

of each neuron to the next neuron and through this process the connections have a weight as mentioned earlier [50], [46]. This weight is basically the signals or information from one layer to another [50]. The system is dependent on the weights to adjust the network repeatedly to get the desired outputs [34]. ANN train the system to find a set of weights that perfectly interprets all the sets of given inputs [34]. The input layer is the layer where external data is imported into the machine to perform the prediction [50]. The output layer is where the problem solution is outputted to the user who is interacting with the machine [50]. The solution is outputted because of the activation function found in the output node [49]. The hidden layer is an intermediate layer which separates the input and output layer [50]. Mainly computation is done in the hidden layer then the weight is transferred between layers [50]. The data imputed into the input layer is run through a training algorithm to get the output solution that is depending on the error rate between the target and actual output [50]. Finding the right weights that would deliver the desired output is only possible if the number of neurons, the layers of the network and the corresponding number of weights form an adequate complex system [34]. These parameters are important to allow the system to fine tune itself until the desired output is produced [34].

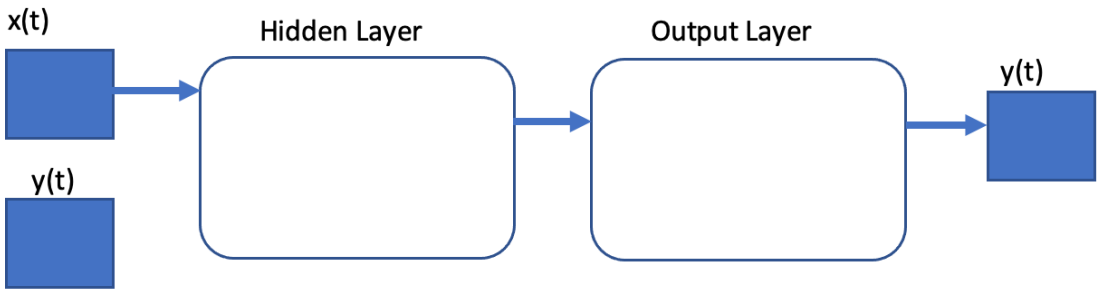


Figure 3.1.2.2 NARX Interconnections between layers

As shown in Figure 3.1.2.2 the hidden layer utilizes a sigmoid transfer function and the output layer a linear transfer function [50]. The interconnections between the layers are shown on Figure 3.1.2.3 below:

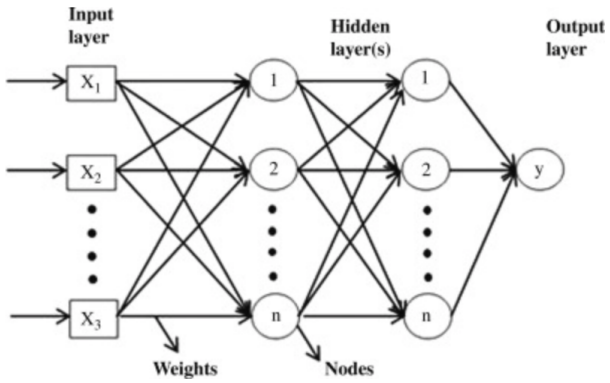


Figure 3.1.2.3 Architecture of an ANN machine [50]

3.1.3 Single Layer and Multilayer Feed-Forward Neural Network

Feed-forward neural networks have specific architecture where the neurons are arranged in a network of layers [51]. These networks of neurons allow signals to travel from input to output only, hence, its name is feed-forward for forward signals only [46]. There are two types of feed-forward neural network which are the single layer and the multilayer feed-forward neural network.

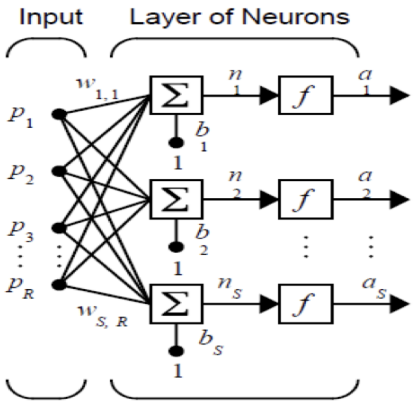


Figure 3.1.3.1 Single Layer Feed-Forward Network [52]

A single layer feed-forward network has a single input layer that is the source nodes feeding an output layer of receiving neurons [51]. As shown in Figure 3.1.3.1, single layer feed-forward networks represent the relationship between the input p and output a that is based on the adjustment of weights w [52]. To determine the output vector an expression is given, this expression is later changed into a vector form using the equations shown below,

$$n_s = \sum_{j=1}^{R} w_{si}\, p_j \; + b_s \text{ [52]} \tag{7}$$

$$a_s = f(n_s) = f(\sum_{i=1}^{S} \sum_{j=1}^{R} w_{ij}\, p_j \; + b_s) \text{ [52]} \tag{8}$$

In equations 7 and 8, R is the number of inputs and S is the number of neurons [52]

$$a = W_p \; + b \text{ [52]} \tag{9}$$

The matrices of the input, bias, weight, and output are given below following equation 9,

$$p = [p_1 \cdots p_R]$$

$$W = \begin{bmatrix} w_{1,1} & \cdots & w_{1,R} \\ \vdots & \ddots & \vdots \\ w_{s,1} & \cdots & w_{s,R} \end{bmatrix}$$

$$b = [b_1 \cdots b_s]$$

$$a = [a_1 \cdots a_s] \text{ [52]}$$

The multilayer feed-forward network works the same as the single layer network but with one or more hidden layers. Hidden layers were discussed earlier in section 3.1.2 and can be seen

in Figure 3.1.3.1 and 3.1.3.2. Multilayer feed-forward networks can learn linear and non-linear function, whereas single layer feed-forward networks can only learn linear functions [53].

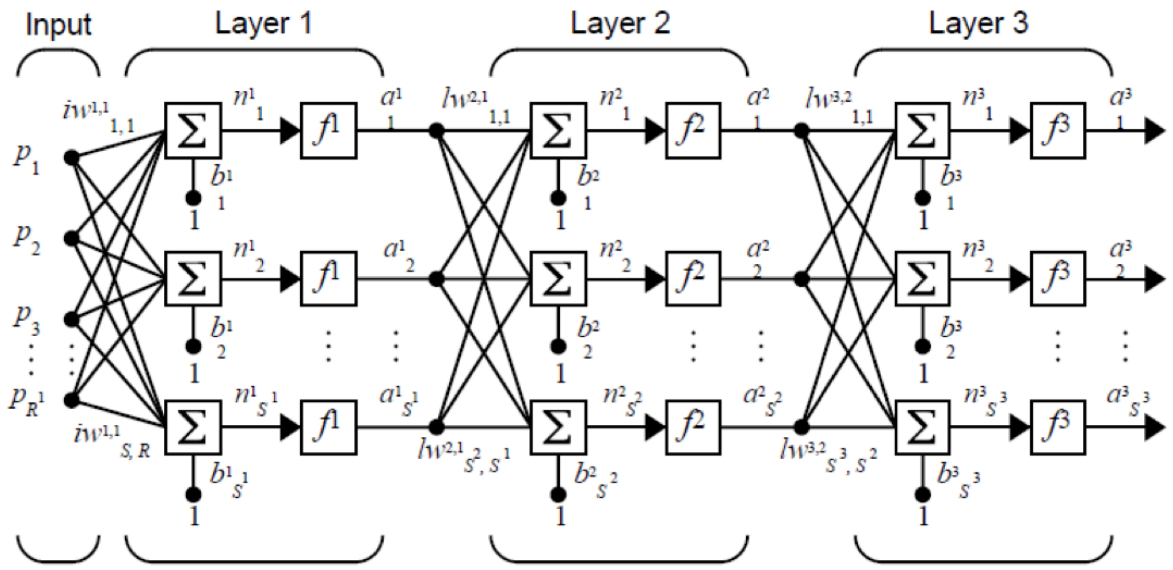


Figure 3.1.3.2 Multilayer Perceptron Network [52]

Figure 3.1.3.2 depicts an example of a multilayer feed-forward neural network. The number of artificial neural networks layers are defined as perceptrons [54]. In the case of multilayer feed-forward network it is a multilayer perceptron or MLP as it has one or more hidden layers [53]. MLP consist of sensory units, which are the input layer, one or more hidden layers of computation neurons, and an output layer of computation nodes [51]. Each layer within the network has its own number of neurons whether it be the same number or different number of neurons [52]. Hidden layers can be single or multiple, and each hidden layer depend on the layer before it [54]. The dependence on the previous layer is present because the input of the next hidden layer is the output of the previous one [52]. This is a recurring thing until the output layer is reached [52]. This is important because each layer or perceptron have the same or different weight vectors and activation functions [52]. For MLP to learn the process is called adapting to synaptic weights [51]. This learning process uses the error back-propagation rule, this has a forward and backward

pass [51]. The forward pass is when the input is propagated through the network layer by layer [51]. This process leads to the synaptic weights being fixed, then the output is the actual response of the network found at the output layer [51]. The backward pass is when the synaptic weights are adapted based on the error-correction rule, which is the gradient descent in weight space [51]. The error-correction rule is done by subtracting the actual response from the target value to produce an error signal that will be propagated backward through the network [51]. This is done to get the desired response out of the actual response [51]. Increasing the amount of perceptron in the network increases the weights, hence, the time it takes to execute the function also increases [54].

To form equations for every hidden layer and the final output based on Figure 3.1.3.2, let l be the number of hidden layers and for the output layer l = L,

Hidden layer 1 (l = 1):

$$a_s = f_1(n_{1s^1}) = f_1(\sum_{i=1}^{s^1} \sum_{j=1}^{R} w_{ij}\, p_j + b_{1s^1}) \text{ [52]} \tag{10}$$

Here S^l = number of neurons in layer l

Hidden Layer 2 to the output layer (l = 2 … L):

$$a_{ls^1} = f_1(n_{1s^1}) = f_1(\sum_{i=1}^{s^l} \sum_{j=1}^{s^{l-1}} w_{ij}\, a_{(l-1)j} + b_{1s^l}) \text{ [52]} \tag{11}$$

Then, L = 3:

$$a_{lS^3} = f_3(n_{3s^3}) = f_3(\sum_{i=1}^{s^3} \sum_{j=1}^{s^2} w_{ij}\, a_{2j} + b_{3s^3}) \text{ [52]} \tag{12}$$

$$a_{3S^3} = f_3\big(\sum_{i=1}^{s^3} \sum_{j=1}^{s^2} w_{ij}\, f_2\,(\sum_{i=1}^{s^2} \sum_{j=1}^{s^1} w_{ij}\, f_1\,(\sum_{i=1}^{s^1} \sum_{j=1}^{R} w_{ij}\, p_j + b_{1s^1}) + b_{2s^2}) + b_{3s^3}\big)\text{[52]} \tag{13}$$

3.1.4 Feedback Neural Network

Unlike feed-forward networks there are feedback networks that have signals traveling in both forward and backward directions [46]. This is done through loops, which often is looked at as a complicated network. Figure 3.1.4 illustrates what a complicated feedback network looks like.

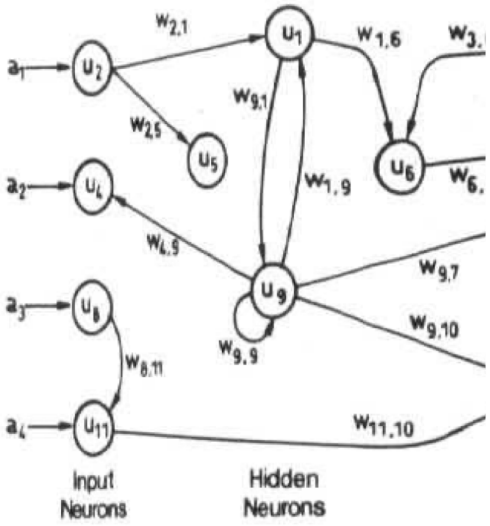


Figure 3.1.4 A Complicated Feedback Network with Loops [46]

Feedback networks allow signals to travel back and forward which makes this type of neural network dynamic [46]. These networks are centered around finding equilibrium points. The signal state is continuously changing until the equilibrium point is reached [46]. Equilibrium points need to be found every time an input changes [46]. The temporal accumulation effect is shown in feedback networks because the current output depends on the current input as well as the accumulation of the previous inputs [55]. To further exploit these networks, control items are introduced, and rules are set to achieve optimal results [55]. There are two types of feedback paths, the local feedback and global feedback paths [52].

This form of artificial neural network has been applied to optimal computation, signal processing, convex nonlinear programming, seismic data filtering [55].

3.1.5 Recurrent Neural Network

Recurrent neural network or RNN is a form of feedback network. As discussed in section 3.1.4, local feedback and global feedback paths are types of feedback paths that can be found in RNN. Figure 3.1.5.1 shows the local feedback path of RNN, and Figure 3.1.5.2 shows the global feedback path of RNN [52].

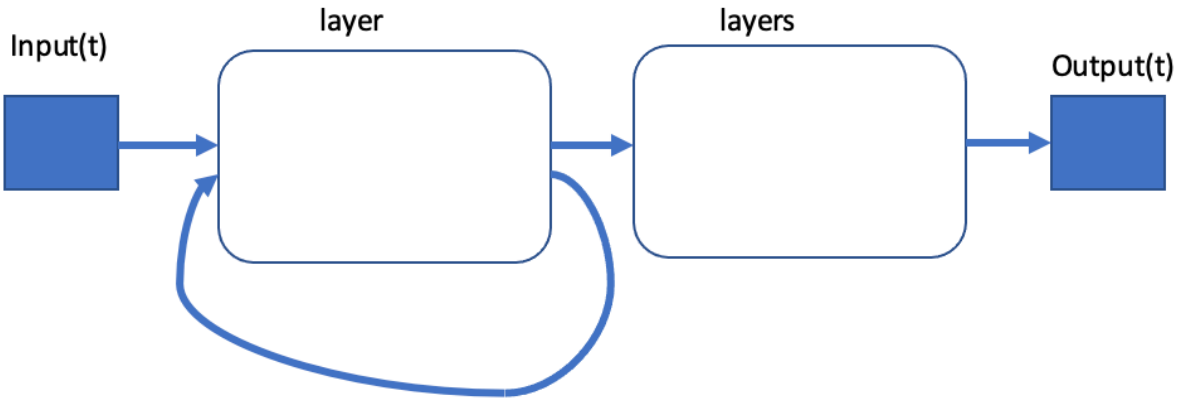


Figure 3.1.5.1 Local Feedback Path of RNN

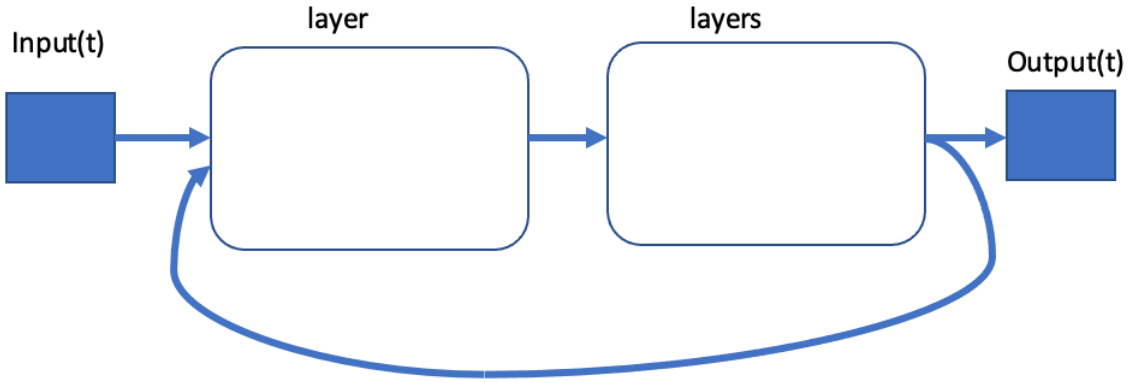


Figure 3.1.5.2 Global Feedback Path of RNN

In Figure 3.1.5.1, the recurrent path found in the hidden layer maintains the feedback functionality [52]. Whereas, in Figure 3.1.5.2 the recurrent path is found as a connecting path made

between the output layer and the input layer [52]. To further interpert the outputs of the local and global recurrent paths some equations are introduced below.

Equation 14, represent the output of a recurrent neural network with a local feedback path or RNNL where t is the current time.

$$a_{1S^1}(t) = f_1(n_{1S^1}) = f_1\left(\sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, a_{1j}\,(t-1) + b_{1s^1}\right)[52] \tag{14}$$

Equation 15, represent the output of a recurrent neural network with a local feedback path or RNNL where t is the current time.

$$a_{2S^2}(t) = f_2(n_{2S^2}) = f_2\left(\sum_{i=1}^{s^2}\sum_{j=1}^{s^1} w_{ij}\, a_{1j}\,(t) + b_{2s^2}\right)[52] \tag{15}$$

Equation 16, represent the output of a recurrent neural network with a local feedback path or RNNL where t is the current time

$$a_{2S^3}(t) = f_2\left(\sum_{i=1}^{s^2}\sum_{j=1}^{s^1} w_{ij}\, f_1\,(\sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j\,(\sum_{i=1}^{s^1}\sum_{j=1}^{s^2} w_{ij}\, a_{1j}\,(t-1) + b_{1s^1}) + b_{2s^2}\right)[52] \tag{16}$$

Equations 14, 15 and 16 represent the output of a recurrent neural network with a local feedback path or RNNL where t is the current time [52].

Equation 17, represent the output of a recurrent neural network with a global feedback path or RNNG where t is the current time.

$$a_{1S^1}(t) = f_1(n_{1S^1}) = f_1\left(\sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j \; + \sum_{i=1}^{s^1}\sum_{j=1}^{S^3} w_{ij}\, a_{2\,j}\,(t-1) + b_{1s^1}\right)[52] \tag{17}$$

Equation 2 represent the output of a recurrent neural network with a global feedback path or RNNG where t is the current time.

$$a_{2S^2}(t) = f_2(n_{2S^2}) = f_2\left(\sum_{i=1}^{s^2}\sum_{j=1}^{s^1} w_{ij}\, a_{1j}\,(t) + b_{2s^2}\right)[52] \quad (18)$$

Equation 19, represent the output of a recurrent neural network with a global feedback path or RNNG where t is the current time

$$a_{2S^2}(t) = f_2\left(\sum_{i=1}^{s^2}\sum_{j=1}^{s^1} w_{ij}\, f_1\,(\sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j\,(\sum_{i=1}^{s^1}\sum_{j=1}^{s^3} w_{ij}\, a_{2j}\,(t-1) + b_{1s^1}) + b_{2s^2}\right)[52] \quad (19)$$

Equations 17, 18 and 19 represent the output of a recurrent neural network with a global feedback path or RNNG where t is the current time [52].

There are many types of RNN architectures such as the bidirectional recurrent neural networks, long short-term memory, and gated recurrent units [56]. Long short-term memory or LSTM is further discussed in section 3.5 and then data is trained using this form of neural network in section 3.6.

To further understand recurrent neural networks, a sequence x = ($x^{(1)}$, $x^{(2)}$, …. $x^{(l)}$,) is considered, here each element $x^{(i)} \in R^d$ is a vector of dimension d [57]. Figure 3.1.5.3 and Figure 3.1.5.4 are compared to differentiate between non-recurrent and recurrent neural networks. In non-recurrent neural networks, information is fed in about the sequence in one go [57]. This methodology disregards the temporary dependencies found in sequence x [57]. This allows the number of weights in the network to increase linearly with length *l* [57]. Figure 3.1.5.3 illustrates a graphical design of a single non-recurrent neuron with *l* inputs and dimension d [57]. Here weights are w1, w2,….., w*l* $\in R^d$, b $\in$ R are bias and σ is an activation function [57].

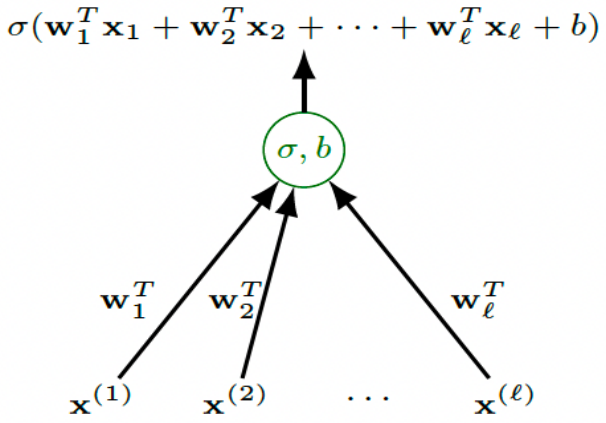


Figure 3.1.5.3 A single non-recurrent neuron [57]

In recurrent neural networks, the sequential quality of the data is taken into account [57]. The elements within the sequence are processed orderly one after another [57]. The unit takes two points which are an element $x_i \in R^d$ and the output of the same unit at the previous time point, $h_{i-1} \in R$ [57]. This approach allows the whole sequence to be conducted with a fixed number of weights and be independent from the sequence length [57]. Figure 3.1.5.4 illustrates a graphical design of a single recurrent neuron with *l* inputs and dimension d [57]. Here weights are $w \in R^d$, $w_h \in R$, and $b \in R$ are bias [57].

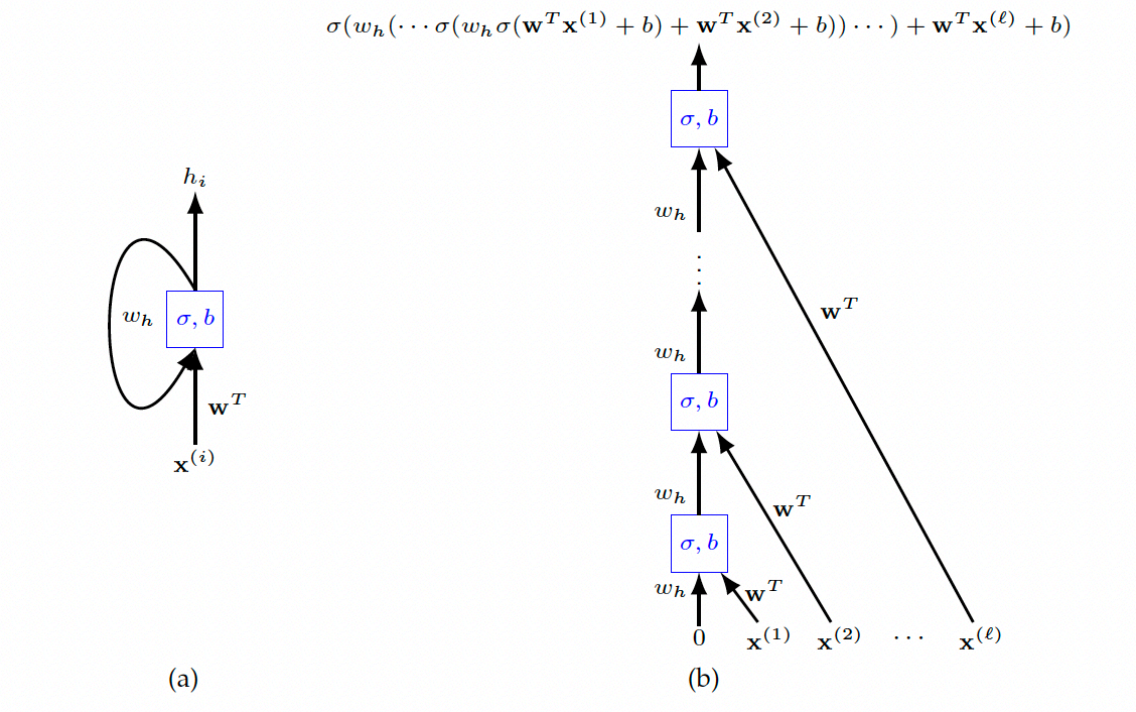


Figure 3.1.5.4 A single recurrent unit [57]

Figure 3.1.5.4 has part a which is the graphical representation of a recurrent neural unit and part b which is what happens when one unravels the time direction of the unit [57]. The blue squares found in Figure 3.1.5.4 represent the same unit [57].

3.1.6 Cascaded Neural Network

Cascaded neural networks are similar to feed-forward networks in the sense that they are also a MLP network [52], [58]. The difference is that cascaded neural networks has a connection from the input and every previous layer to the following layers, while feed-forward networks does not [58]. This means the input and output layers are directly connected [58]. The word cascaded alludes that each input and output of every hidden layer is cascaded to the next layer [52]. This is important as it always for the learning of any finite input-output relationship if enough hidden neurons are given [58]. In addition, this connection allows for integrating the execution of a linear

relationship alongside a non-linear relationship [58]. Figure 3.1.6.1 shows the structure of a cascaded neural network.

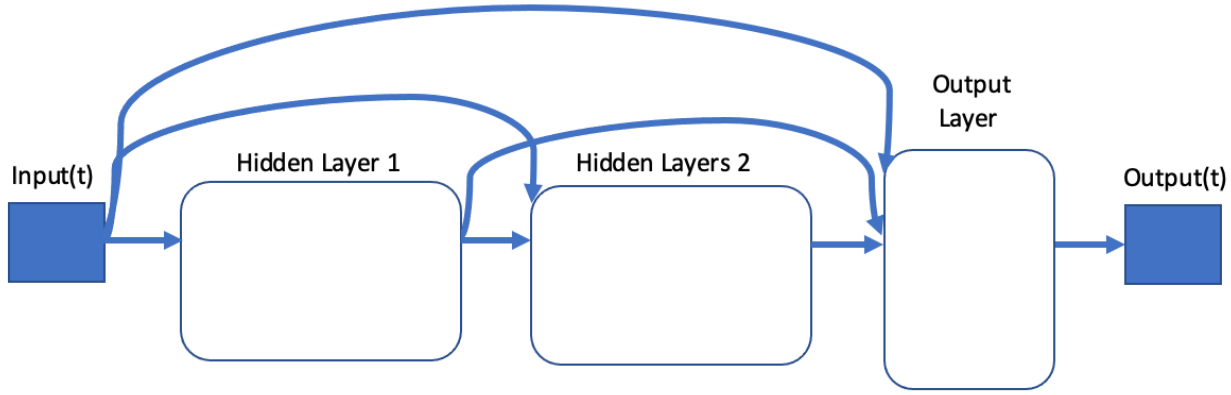


Figure 3.1.6.1 Cascaded Neural Network Layers

To form equations for every hidden layer and the final output based on Figure 3.1.6.1, let l be the number of hidden layers and for the output layer l = L = 3,

Hidden layer 1 (l = 1):

$$a_{1S^1}(t) = f_1(n_{1S^1}) = f_1\left(\sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j + b_{1s^1}\right) \text{[52]} \tag{20}$$

Hidden Layer 2 to the last hidden layer (l = 2):

$$a_{2S^2}(t) = f_2(n_{2S^2}) = f_2\left(\sum_{i=1}^{s^2}\sum_{j=1}^{s^1} w_{ij}\, a_{1j} + \sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j + b_{2s^2}\right)\text{[52]} \tag{21}$$

$a_{2s}{}^2$ can be viewed as the output of a cascaded neural network with one hidden layer.

Output Layer (l = L = 3):

$$a_{3S^3}(t) = f_3(n_{3S^3}) = f_3\left(\sum_{i=1}^{s^3}\sum_{j=1}^{s^2} w_{ij}\, a_{2j} + \sum_{i=1}^{s^2}\sum_{j=1}^{s^1} w_{ij}\, a_{1j} + \left(\sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij} p_j + b_{3s^3}\right)\right) \text{[52]} \tag{22}$$

$$a_{3S^3}(t) = f_3\left(\sum_{i=1}^{s^3}\sum_{j=1}^{s^2} w_{ij}\, f_2\left(\sum_{i=1}^{s^2}\sum_{j=1}^{s^1} w_{ij}\, f_1\left(\sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j\ + b_{1s^1}\right) + \sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j\ + b_{2s^2}\right) + \sum_{i=1}^{s^2}\sum_{j=1}^{s^1} w_{ij}\, f_1\left(\sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j\ + b_{1s^1}\right) + \sum_{i=1}^{s^1}\sum_{j=1}^{R} w_{ij}\, p_j\ + b_{3s^3}\right)[52] \quad (23)$$

3.2 Neural Network Model (NARX)

Nonlinear autoregression with external input (NARX) is a dynamic feedback (recurrent) neural network with a delayed output value that is feedback as an endogenous input and another time series exogenous input. Therefore, the next output is regressed on the previous output values and the previous values of independent variables of an input signal. NARX neural network is based on an autoregression model and is suitable for modeling a nonlinear dynamic system used in time series modeling because of its learning ability [29], [59]. It is a nonlinear system in discrete time, and it is defined mathematically as:

$$\boldsymbol{y(t+1) = f\,[y(t), y(t-1), .., y(t-dy+1), .., x(t), x(t-1), ..., x(t-d+1)],} \quad (24)$$

y(t) and x(t) are the outputs and inputs respectively and are the corresponding output and input delays. The output in the next time step, y(t + 1), is a function of y(t) and x(t). Mathematically, y(t + 1) = f [y(t), x(t)]. Consequently, NARX output is considered as output of a linear dynamic system which it seeks to model.

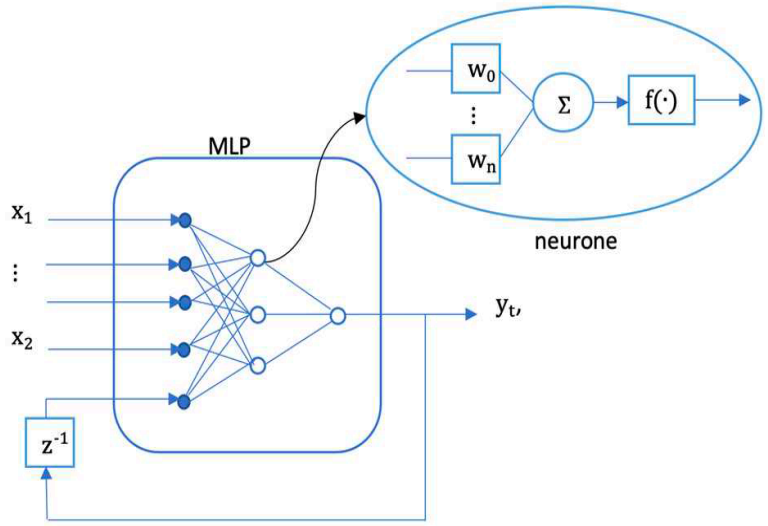


Figure 3.2.1 modified diagram Recurrent Neural Network [59]

3.3 Support Vector Machine Method (SVM)

Support vector machine (SVM) have been applied in classification problems to separate data into classes or categories. But it can also be used for finding regression lines from a set of non-linear data.

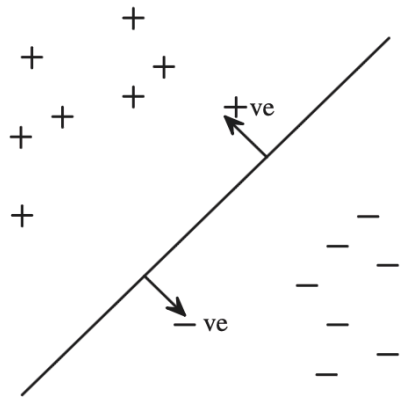


Fig 3.3.1 SVM For Classification [60]

Figure 3.3.1 shows SVM for classification for two classes of separated data in a 2-D representation [60]. The straight line in Figure 3.3.1 depicts an oriented hyperplane that

corresponds to w . x + b = 0 [60]. This hyperplane has different labels of each side in this case yi = +1 and yi = -1 [60]. The decision function of a classifier used here is w . x + b, here b is the offset, . is a scalar product, x are points located within the hyperplane and the normal to the hyperplane and w being the weights which determines the hyperplane orientation [60].

In this paper, our interest is to use the SVM to solve linear regression problems. The SVM works by transforming the data to a new space. For example, data can be transformed from 2-dimensional space to 3-dimensional space. As such, non-linear data in a 2-dimensional space may all lie in a single plane in a 3- dimensional space. A linear regression line in the new space can represent a line in the original space. Figure 3.3.1 illustrates readily separable clusters; this is hypothetical as usually the two clusters could be extremely intermeshed with overlapping datapoints; therefore, kernels are introduced to separate data [60]. Kernel functions such as the linear, gaussian and cubic are usually used in transforming the original dataset to a new space to find a linear regression line. The aim of the SVM is to minimize the distance between the regression line and the data points. To find the optimal line, a tube, which allows some margin of error, around the optimum regression curve is used. Points on and outside the tube boundary are the support vectors and are used for computing the total error while points inside the tube boundary have no errors and are not taken into consideration in computation. Furthermore, slack variables act as connecting weights between the data points and the regression line. Due to the strict allowable total error constraint, data points that are too far away from the tube boundary are penalized. The further away the data point is, the more the point is penalized and the larger the slack variable. The task of the SVM is to minimize the distance between the data points subject to the slack variables (weights) and the total error constraint of the tube [17]. Therefore, the goal of the SVM is an optimization problem whose optimal solution is the regression coefficient. The

major difference between the SVM, NARX and MLR is that for NARX and MLR, all data points are used in the computation while in the case of SVM, only the support vectors are used in the computation.

3.4 Decision tree (DT)

Decision trees are often used in data mining such as word recognition, medical problem diagnosis etc. It uses supervised learning to learn the relationship between input and output dataset. It is basically a flow chart from top to bottom. It consists of internal nodes. The fundamental node is called the root node and the end of the tree is called the leaves node. The fundamental nodes represent sets of tests performed on an attribute while sets of branches represent the outcome. The leaves node represents the decisions that are performed after the computation on the attributes [61]. A model of a decision tree is shown in figure 3.4 where x is the root node, a and b are the internal nodes. The following steps are followed in constructing a decision tree.

1) **Step 1: Given a tuple training set T for a class tag and assuming n output test, and freq(Ci,T) representing the number of cases in T belonging to the class , then the training set is partition into subset such that the entropy of the set T (in bits) is computed as**

$$Info(T) = -\sum_{i=1}^{k} \frac{freq(C_i, T)}{|T|} \times log_2(\frac{freq(C_i, T)}{|T|}) \quad (25)$$

2) **Step 2: After T has been partitioned in accordance with the n outcome of a test X, the information entropy of the property T is found as the weighted sum of over the subsets as**

$$Info_x(T) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times log_2(T_i) \tag{26}$$

3) **Step 3: The information gain represents the difference between the initial information and the new one. Equations (25) and (26) can be combined as**

$$Gain(x) = Info(T) - Info_x(T) \tag{27}$$

The gain measures the information that is derived by partitioning T in accordance with the test x. The aim is to select a test that maximizes the information gained.

4) **Step 4: The gain is normalized as follows**

$$\left.\begin{aligned} Split\_info(T) &= -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times log_2\left(\frac{|T_i|}{|T|}\right) \\ Gain\_ratio(x) &= \frac{Gain(x)}{Split\_info(x)} \end{aligned}\right\} \tag{28}$$

Equation (28) expresses the proportion of information generated by the split for classification to produce a final decision tree [62-63].

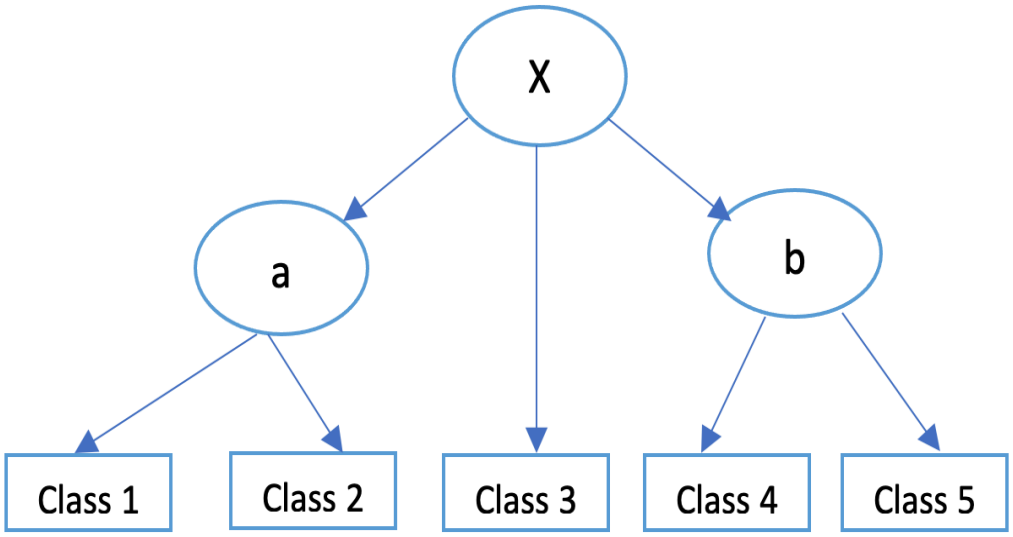


Figure 3.4 modified diagram Model of a Decision Tree [62-63]

3.5 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) algorithm is an artificial Recurrent Neural Network (RNN) architecture that is capable of learning order dependence in the field of deep learning. Unlike typical feedforward neural networks, LSTM introduces feedback neural connections, which can be used to effectively minimize or eliminate the raining unstable issues caused by RNN due to called vanishing/exploding gradient problems. An LSTM is presumed as a good fit for classifying sequence and time-series data when the prediction or output of the network must be based on a remembered sequence of data points.

$$f_t = \sigma(U^f h_{t-1} + W^f x_t) \quad (29)$$

$$i_t = \sigma(U^i h_{t-1} + W^i x_t) \quad (30)$$

$$o_t = \sigma(U^o h_{t-1} + W^o x_t) \quad (31)$$

$$\tilde{c}_t = \sigma(U^c h_{t-1} + W^c x_t) \quad (32)$$

$$c_t = f_t \odot \tilde{c}_{t-1} + i_t \odot \tilde{c}_{t-1} \quad (33)$$

$$h_t = o_t \odot \tanh(c_t) \quad (34)$$

Equation 29 is the forget gate equation where it determines how important is the past state and delete the information from the cell state that is no longer needed [64]. The input gate is Equation 30, this gate determines how important the input is and select the information to add to the new cell state from the input [64]. Equation 31 is the output gate which is the gate that determines how important the new state and whether it is useful for the output [64]. Equation 32 is the new memory cell which extracts information from the previous hidden cell and input to create candidate memory [64]. The final memory cell is the cell that computes new cell state which can be seen in Equation 33 [64]. The final hidden state is in Equation 34, this updates the hidden state [64].

To put it in a nutshell, a general LSTM unit contains a cell, an input gate, a forget gate, and an output gate. The cell unit remembers values at random time points, and the three gates control the direction of the information pathway into and out of the cell. The simple flow chart of the LSTM is illustrated in Fig. 3.5.

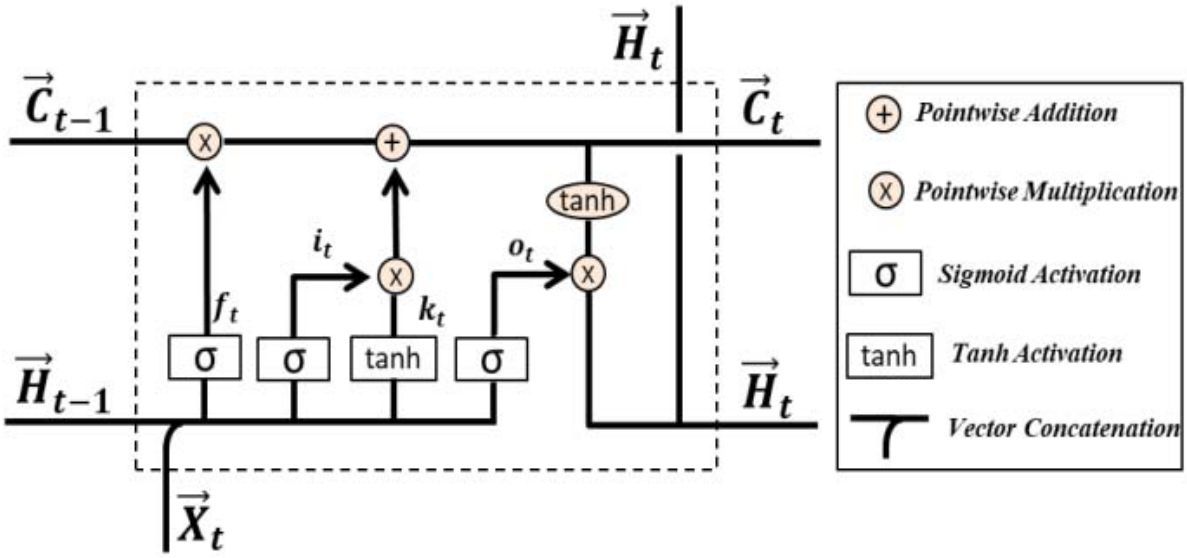


Figure 3.5 An LSTM unit Diagram [64]

Chapter 4: Machine Learning Based Load Forecast Results

Neural network training consists of carefully redesigning the architecture of the network so it can follow a particular behaviour manipulated in a desired way. In this chapter, predictions using different neural network algorithms are discussed and compared. Two datasets are used for further comparison. The main dataset is the REFIT dataset which collected whole-house aggregated loads at 8-second intervals continuously from 20 houses over a two-year period in the U.K [10]. The second dataset is based on Canadian values specifically in Burnaby, British Colombia, Canada [11]. The house ID used from the set of data was House ID: 1, this house is a residential single detached house with rental suite. The data chosen to train were from 6th of March 2016 to 7th of May 2016. The AMPds data were sampled in hours, this explains why the data is smaller in size compared to the REFIT dataset which is 8-second intervals [11], [10]. Also, this means that there is an average of all the measurements within that hour considering the load is constant. This makes the data low resolution compared to the REFIT UK dataset.

Commonly used parameters of evaluating and quantifying the performance measure and, or accuracy of any forecasting machine learning algorithms include the mean absolute error (MAE), mean absolute percentage error (MAPE), root mean square error (RMSE), and standard deviation (SD). The mathematical descriptions of the performance metrics as using the apps in MATLAB- regression linear were used to find neural network predictions (the minimum and accurate trees and were determined and chosen to plot against actual and NARX and LSTM):

Equation 35 is the mean absolute error (MAE), Equation 3 is the mean absolute percentage error (MAPE), Equation 4 is the root mean square error (RMSE) and Equation 5 is the standard deviation (SD).

$$MAE = \frac{1}{N}\sum_{t=0}^{N}\left|Y_{real} - Y_{forecast}\right| \tag{35}$$

$$MAPE = \frac{1}{N}\sum_{t=0}^{N}\frac{\left|Y_{real} - Y_{forecast}\right|}{Y_{real}} \times 100 \tag{36}$$

$$RMSE = \frac{1}{N}\sqrt{\sum_{t=0}^{N}\left(Y_{real} - Y_{forecast}\right)^2} \tag{37}$$

$$SD = \sqrt{\frac{1}{N}\sum_{t=0}^{N}\left(Y_{forecast} - Y_{mean}\right)^2} \tag{38}$$

Y_{real} , $Y_{forecast}$, Y_{mean}, N and t are the actual value, the predicted value, the mean of the actual values, the testing dataset and the test instant index respectively [65]. The performance metrics measure the difference between the real value and the predicted value. The smaller the metrics quantity, the better the prediction accuracy [66].

Getting the reasonable forecasting of the decision tree and SVM by choosing the smallest RMSE

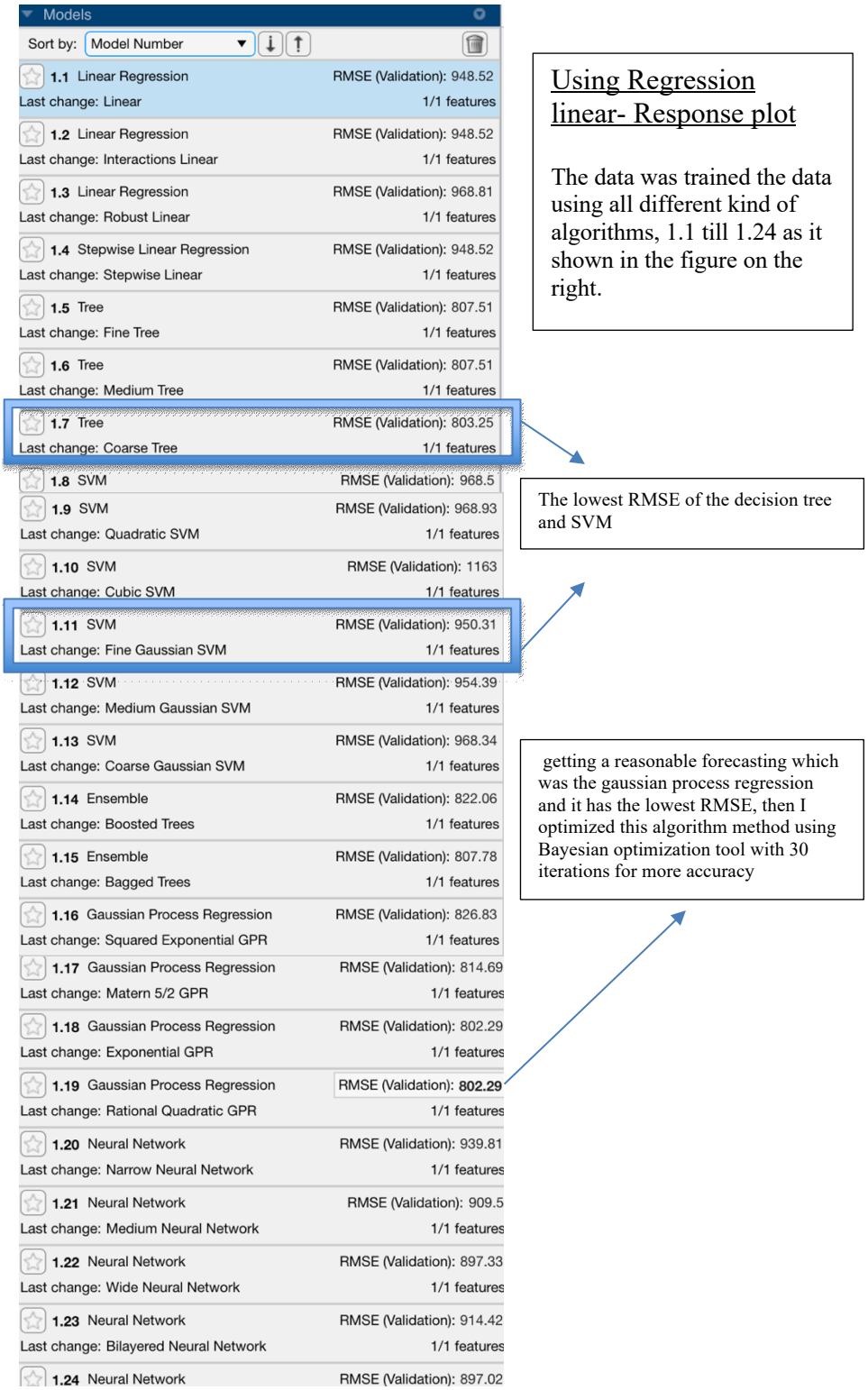


Figure 4.0 Regression linear to TRAIN SVM AND DT

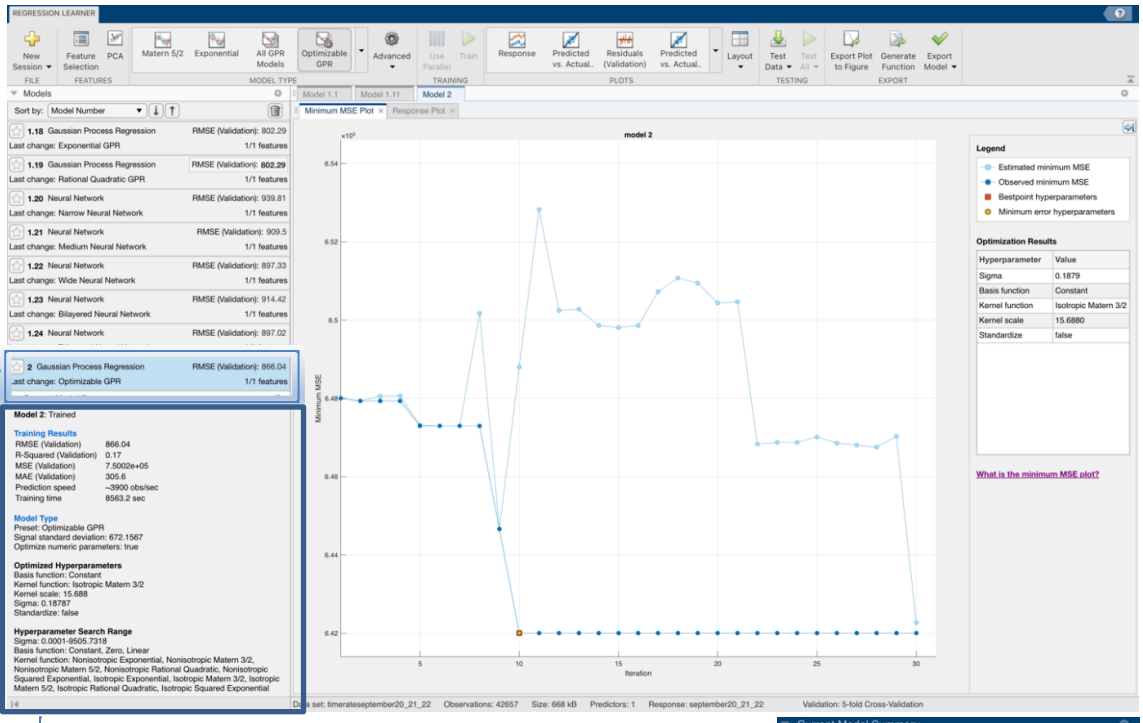


Figure 4.1 Minimum MSE Plot

This is the optimization RMSE value which is 866.04

This box shows the model optimization summary

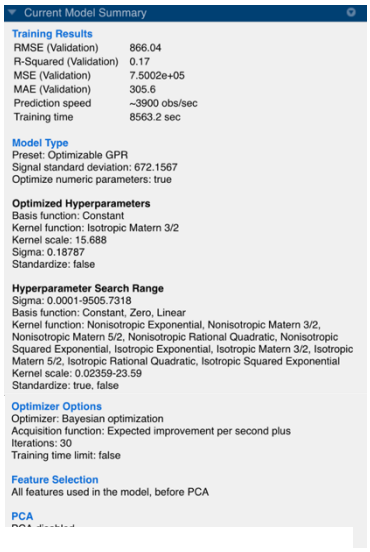


Figure 4.2 current Model Summary

The **Current Model Summary** contains:

- Training Results – **Shows the performance of the optimizable model.**
- Model Type – **Displays the type of optimizable model and lists any fixed hyperparameter values**
- Optimized Hyperparameters – **Lists the values of the optimized hyperparameters**
- Hyperparameter Search Range – **Displays the search ranges for the optimized hyperparameters**
- Optimizer Options – **Shows the selected optimizer options**

4.1 NARX Prediction Results

Error results for NARX in REFIT dataset:

Table 1 Prediction errors for NARX in UK REFIT dataset:

	MAPE %	MAE	RMSE	SD
NARX	1.5519	420.2535	6.547	925.8733

Error results for NARX in AMPds dataset:

Table 2 Prediction errors for NARX in Canada AMPds dataset:

	MAPE %	MAE	RMSE
NARX	0.24517	133.097	159.260

Using the REFIT dataset, NARX produced a lower RMSE value, but a higher MAPE and MAE values compared to the AMPds dataset. This is because NARX works better with fluctuating datasets which in this case it was the AMPds dataset.

4.2 LSTM Prediction Results

Error results for LSTM in REFIT dataset:

Table 3 Prediction errors for LSTM in UK REFIT dataset:

	MAPE %	MAE	RMSE	SD
LSTM	0.2820	52.3055	0.8261	940.3203

Error results for LSTM in AMPds dataset:

Table 4 Prediction errors for LSTM in Canada AMPDs dataset:

	MAPE %	MAE	RMSE
LSTM	1.474	11.402	59.477

Using the REFIT dataset, LSTM produced a lower RMSE and MAPE values, but a higher MAE value compared to the AMPds dataset. This is because LSTM works better with steady and bigger size datasets which in this case it was the REFIT dataset.

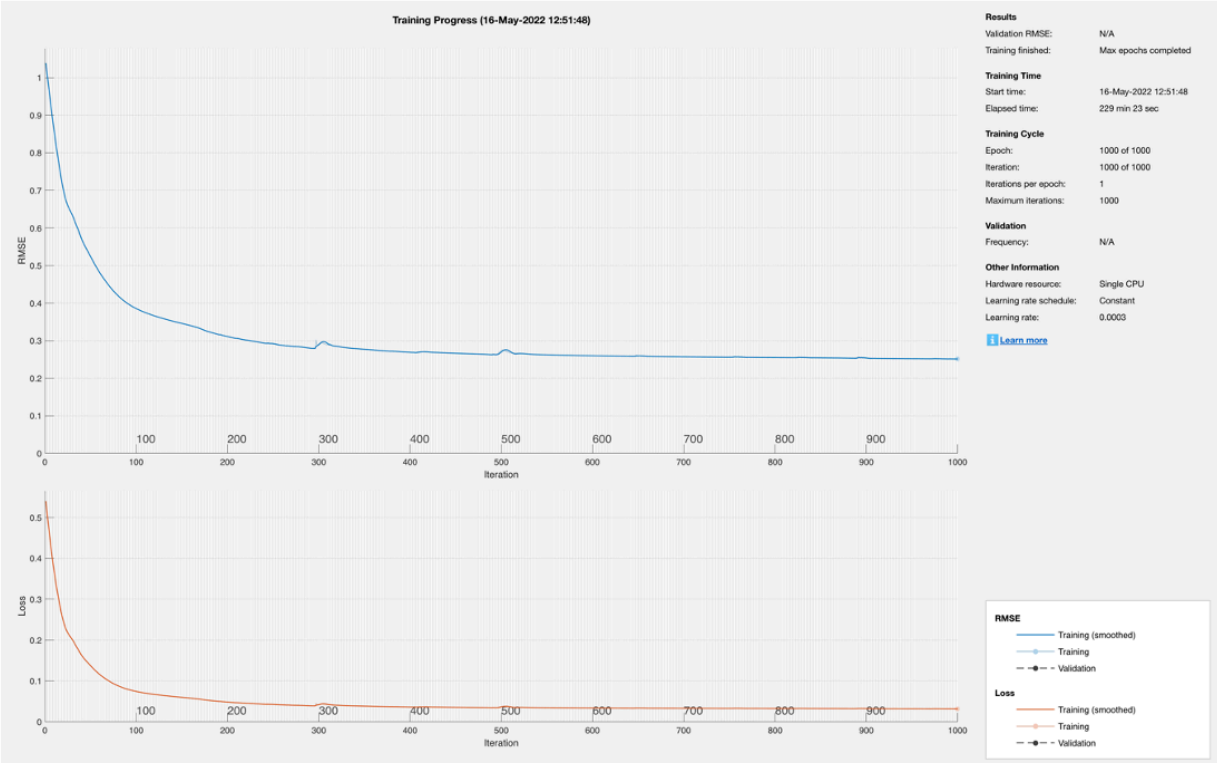


Figure 4.2.1 LSTM REFIT Dataset Training Progress of the Actual dataset in UK

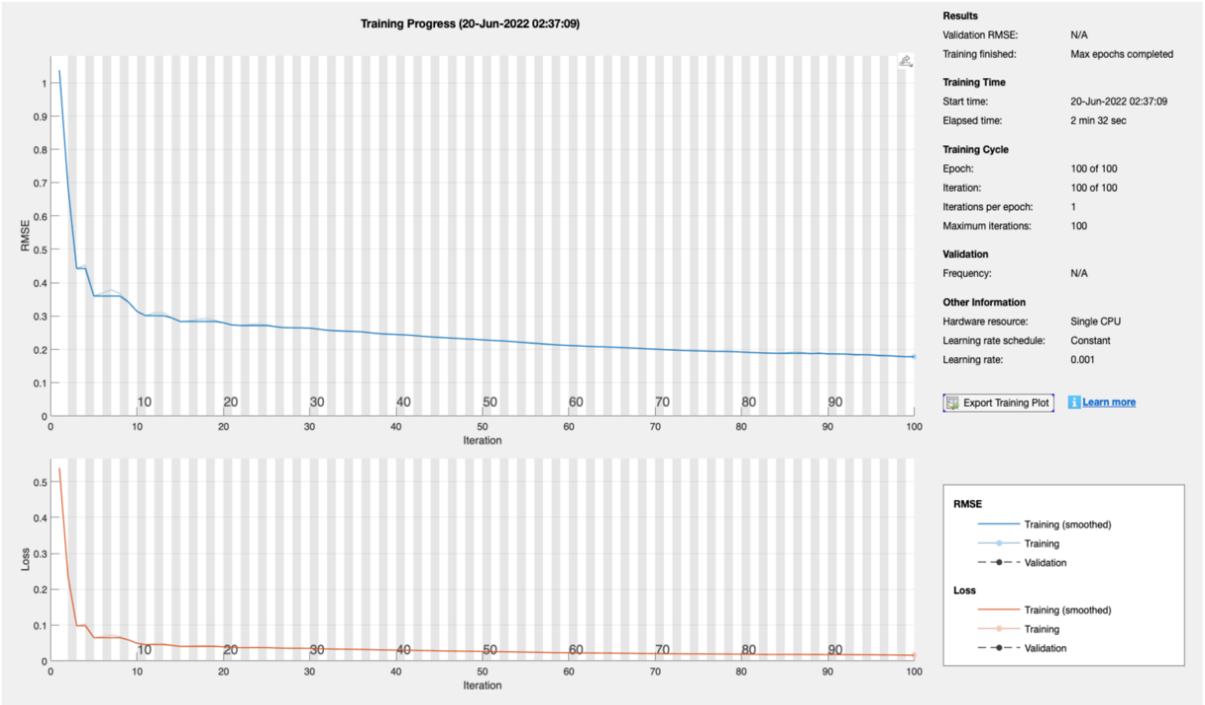


Figure 4.2.2 LSTM AMPds Dataset Training progress of the Actual dataset in Canada

4.3 SVM Prediction Results

Error results for SVM in REFIT dataset:

Table 5 Prediction errors for SVM in UK REFIT dataset:

	MAPE %	MAE	RMSE	SD
SVM	0.3971	259.8025	5.1538	236.4332

Error results for SVM in AMPds dataset:

Table 6 Prediction errors for SVM in Canada AMPds dataset:

	MAPE %	MAE	RMSE
SVM	18.879	99.653	124.335

Using the REFIT dataset, SVM produced a lower RMSE and MAPE values, but a higher MAE value compared to the AMPds dataset. This is because SVM works better with steady and bigger size datasets which in this case it was the REFIT dataset.

4.4 Decision Tree Prediction Results

Error results for DT in REFIT dataset:

Table 7 Prediction errors for DT in UK REFIT dataset:

	MAPE %	MAE	RMSE	SD
DT	1.4312	388.3178	5.4412	294.8280

Error results for DT in AMPds dataset:

Table 8 Prediction errors for DT in Canada AMPds dataset:

	MAPE %	MAE	RMSE
DT	18.724	99.017	123.883

Using the REFIT dataset, DT produced a lower RMSE and MAPE values, but a higher MAE value compared to the AMPds dataset. This is because DT works better with steady and bigger size datasets which in this case it was the REFIT dataset.

4.5 Actual vs Forecasted Results

The actual vs forecasted predictions for the UK (REFIT) dataset using NARX, LSTM, SVM and DT is shown in fig. 4.3.1 below.

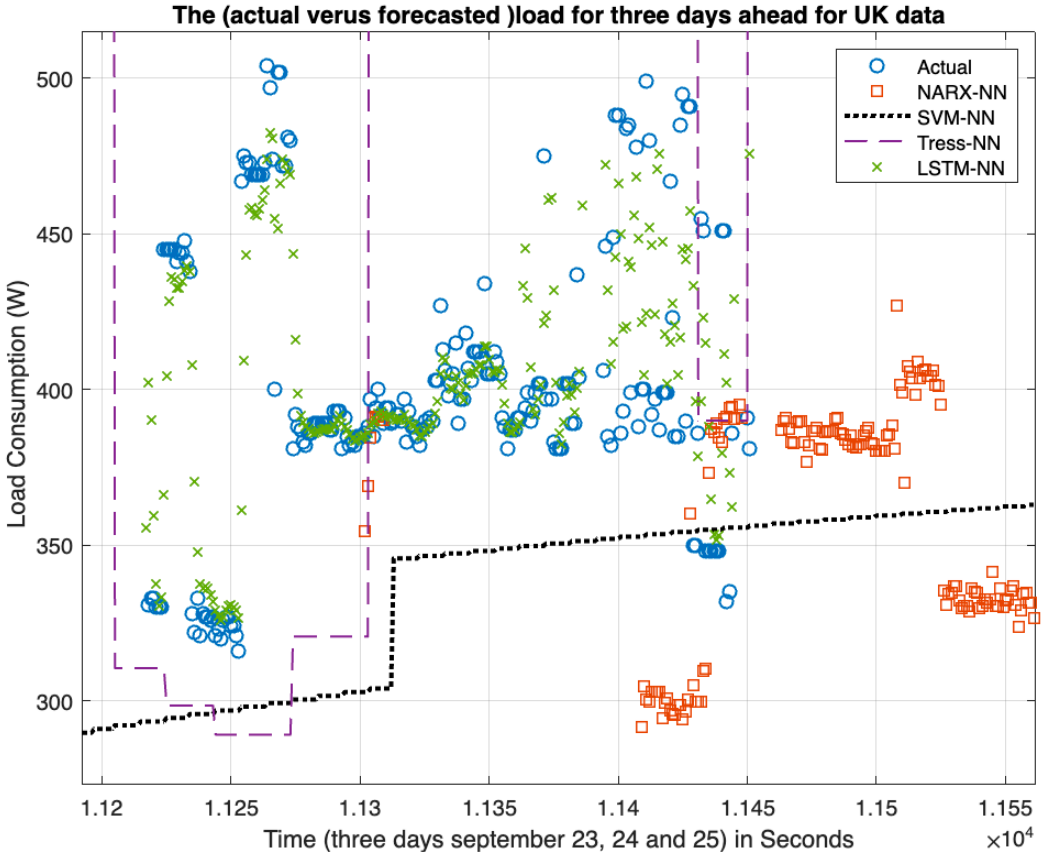


Figure 4.3.1 The REFIT actual vs forecasted load forecasting (NARX, LSTM, SVM and DT)

Figure 4.3.1 shows a chart that gives figures for actual load versus forecasted load of four different prediction methods which are NARX, LSTM, SVM and DT with a period from 1.12x10^4 to 1.155x10^4 seconds. The blue circle represents the actual data for the REFIT dataset. The red square, dotted black line, dash purple line, and green x are representing NARX, SVM, DT and LSTM, respectively used on the REFIT dataset. To choose the best method it is important to observe which shaped in the chart follows the blue circle shape most accurately as it is the actual load data. In this case the green x which represents LSTM follows the blue circle most accurately.

In this prediction project the long short-term memory method always has the best results, mainly because the data size is large and, we are doing near real time prediction. Also, the REFIT dataset has steady data which does not significantly fluctuate.

Table 9 Errors summaries of the REFIT original load forecasting

	MAPE	**MAE**	**RMSE**	**SD**
NARX	1.5519	420.2535	6.5947	925.8733
LSTM	**0.2820**	**52.3055**	**0.8261**	**940.3203**
SVM	0.3971	259.8025	5.1538	236.4332
DT	1.4312	388.3178	5.4412	294.8280

In the actual vs forecasted load forecasting as shown from Table 9 LSTM has the best results. The LSTM method has a mean absolute percentage error (MAPE) of 0.282, mean absolute error (MAE) of 52.3055, root mean square error (RMSE) of 0.8261, and standard deviation (SD) of 940.3203. Although its standard deviation value was greater than all the other methods. The SVM method has the lowest SD value which is 236.4332. Table 9 agrees with the results of the chart shown in Fig 4.3.1.

The actual vs forecasted load predictions for the Canada dataset (AMPds) using NARX, LSTM, SVM and DT is shown in fig.4.3.2 below.

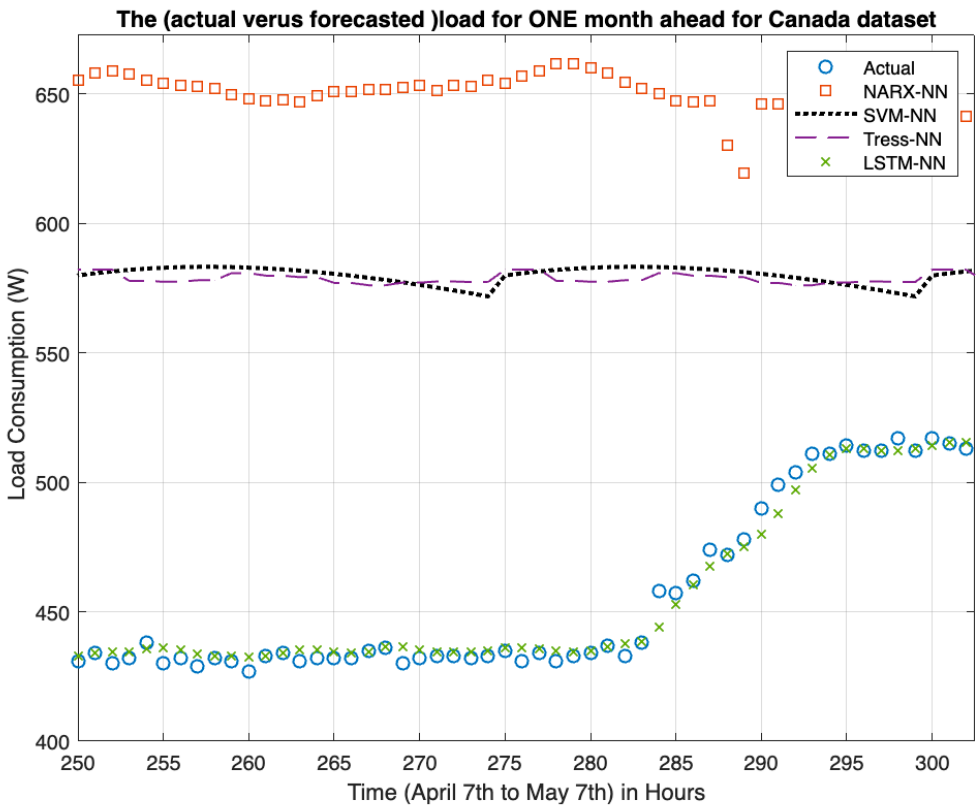


Figure 4.3.2 The Canada AMPds actual vs forecasted load forecasting (NARX, LSTM, SVM and DT)

In Figure 4.3.2, the figures were provided for the AMPds dataset of the actual load versus forecasted load of four different predictions methods which are NARX, LSTM, SVM and DT with a zoomed period in hours. The blue circle represents the actual data for the AMPds dataset, whereas the red square, dotted black line, dash purple line, and green x are representing NARX, SVM, DT and LSTM, respectively used on the AMPds dataset. The shape that perfectly trails the blue circle is the best and the most precise. In this case the green x which represents LSTM follows the blue circle most accurately. LSTM's green x-shaped is the only shape that can be seen fluctuating up with the actual data's blue circle seen at 285 till 295 hours.

Table 10 Errors summaries of the Canada AMPds data set actual versus forecasted load forecasting

	MAPE	MAE	RMSE
NARX	0.24517	133.097	159.260
LSTM	1.474	11.402	59.477
SVM	18.724	99.017	123.883
DT	18.879	99.653	124.335

In the actual versus forecasted load forecasting for the Canada AMPds dataset as shown from Table 10 although LSTM's MAPE value was greater than the NARX method, ultimately, LSTM had the best overall results. The LSTM method has a mean absolute percentage error (MAPE) of 1.474, mean absolute error (MAE) of 11.402, root mean square error (RMSE) of 59.477. The methods SVM and DT have almost exactly the same error values. In addition, both SVM and DT performed better than NARX based on their MAE and RMSE values. NARX had the lowest value for the MAPE error which is 0.24517.

When comparing the REFIT and AMPds datasets, it is observed that their size differentiates greatly. This is because the REFIT dataset is far bigger in size than the AMPds dataset. In addition, LSTM works the best with both datasets. SVM and DT both worked better with the steady dataset, which is the REFIT dataset, although, their MAE values were better in the AMPds dataset.

Chapter 5: Vulnerability Assessment of Machine Learning based Short-Term Load Forecast Algorithms under Smart Meter Cyberattacks

To assess the impact of cyberattacks on the four residential load forecasting machine learning algorithms, the UK dataset for 20 households [23] was employed. This is one of the first publicly available datasets collected primarily to promote ED research. The REFIT electrical load measurements dataset includes whole-house aggregate loads and nine individual appliance measurements at 8-second intervals per house, the data collected continuously from 20 houses over a two-year period. This will also include the other case study in the AMPds dataset, the data collect in house 1 hourly based [11]. This will show the differentiations between both countries and the effect of the time intervals.

5.1 Modeling of Cyberattacks on Smart Meters

The following four cyber-attacks on smart meters of residential houses are considered in this work. These are previously made cyber-attack templates; they have been used in cyber security literature of other smart grid applications [67].

Table 11 Math model used in the evaluation of cyber-attack models. [67]

Measures	Equations	Descriptions
pulse	$P_t = (1+ \lambda_P)$ X $load_t$ For $t = t_p$	Pulse attacks at a certain location, that changed load to higher/lower levels at a given time.
scaling	$S_t = (1+ \lambda_S)$ X $load_t$ For $t_S < t < t_e$	Scale attacks over a predetermined duration.
ramping	*Type 1* $R_t = \lambda_R$ X(t-ts) X $load_t$ For $t_S < t < t_e$ *Type 2*	Ramping attacks in two different types: Type1: attacks up-ramping anomaly. Type 2: attacks both up- and down – ramping anomalies.

	$R_t = [1 + \lambda_R X (t - t_s)] X load_t$ For $t_S < t < [\frac{ts+te}{2}]$ $R_t = [1 + \lambda_R X (t_e - t)] X load_t$ For $[\frac{ts+te}{2}] < t < t_e$	
random	$Ran_t = load_t + \lambda_{RAN} X rand(t)$ For $ts < t < te$	Random attacks randomly using uniform random function. Where rand is the random generator that is available in MATLAB.

In Table 1, P_t is the pulse attack, S_t is the Scale attack, R_t is the ramp attack and Rant is the random attack [24]. They were observed at time t, t_p (is the amount time of one pulse attack), t_s (the start of one cyber-attack), t_e (the end-time of one cyber-attack), and λ_P, λ_S , λ_R , λ_{RAN} are the attack parameters under pulse, Scale, Ramping and random, respectively. The original load is $load_t$ [67].

In this project two cases of cyber-attack models were determined. The first case is by using each attack model separately to the original data. The second case is by using all the four attack models together to the original data in both UK and Canada datasets.

5.2 The load prediction Under Pulse cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

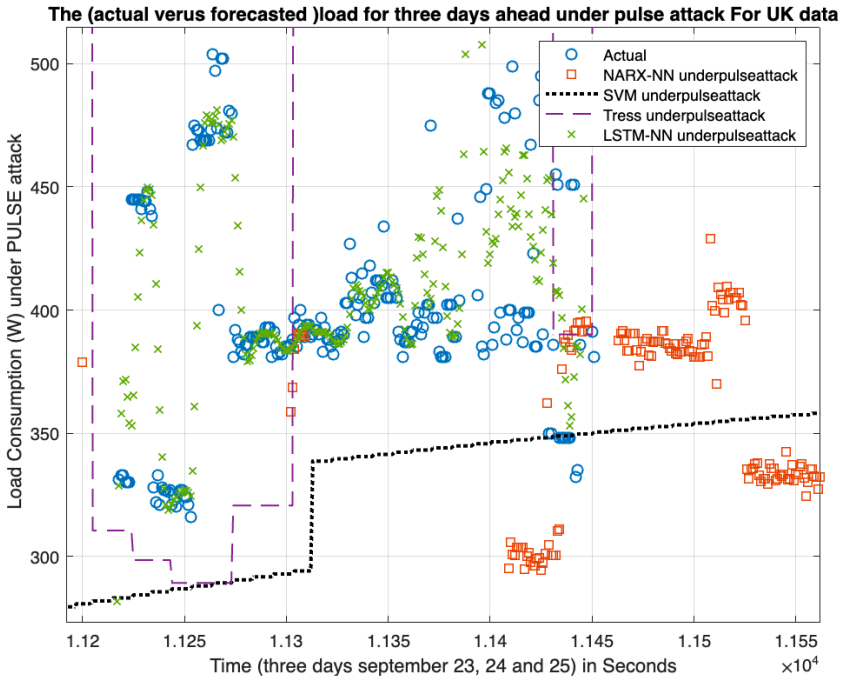


Figure 5.1.1 The load prediction Under Pulse cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

Figure 5.1.1 above present the modified load under pulse attack in the UK dataset. To select the best method, it is important to observe which shape follows the blue circle most accurately. In this case the green x-shape which represents LSTM follows the blue circle most accurately.

Table 12 Errors summaries for the REFIT dataset of the load prediction Under Pulse cyber-attack forecasting (NARX, LSTM, SVM and DT) vs Actual

	MAPE	MAE	RMSE	SD
NARX	1.5268	418.8209	6.5978	926.5126
LSTM	**0.1061**	**58.2567**	**1.7030**	**843.2734**
SVM	0.4000	259.8690	5.1557	236.0923
DT	1.4319	388.3964	5.4412	290.2558

In the load prediction Under Pulse cyber-attack forecasting (NARX, LSTM, SVM and DT) versus actual as shown on Table 12 LSTM has the best results. The LSTM method has a mean absolute percentage error (MAPE) of 0.1061, mean absolute error (MAE) of 58.2567, root mean square error (RMSE) of 1.703, and standard deviation (SD) of 1029. However, its standard deviation value was greater than all the other methods. The SVM method has the lowest SD value which is 236.0923. Table 12 agrees with Figure 5.1.1 as they both showed that LSTM works best for the REFIT dataset under the pulse cyber-attack.

The load predictions under pulse cyber-attack forecasting for AMPds dataset using NARX, LSTM, SVM and DT is shown in fig 5.1.2 below.

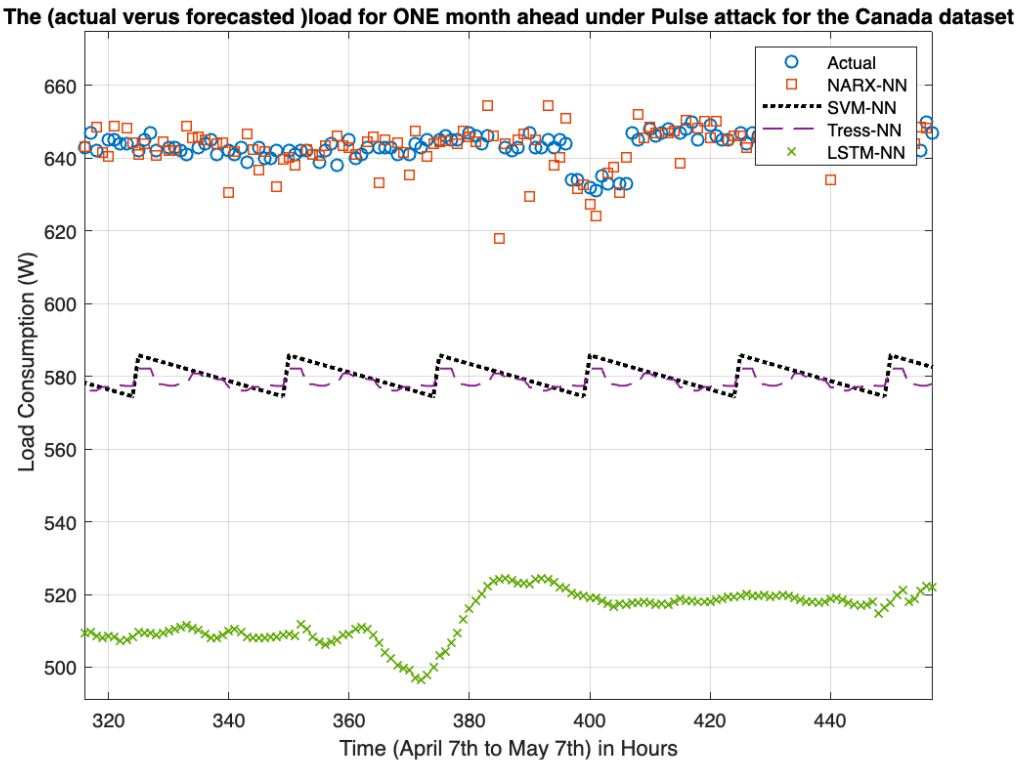


Figure 5.1.2 The AMPds load prediction under pulse cyber-attack forecasting (NARX, LSTM, SVM and DT) vs Actual

In Figure 5.1.2 the blue circle represents the actual data for the AMPds dataset. To choose the best method it is important to observe which shape follows the blue circle most accurately. In this case the red square which represents NARX follows the blue circle most accurately.

Table 13 Errors summaries for the AMPds dataset of the load prediction Under Pulse cyber-attack vs actual

	MAPE	MAE	RMSE
NARX	**16.446**	**113.2301**	**178.3073**
LSTM	22.823	159.0882	207.5454
SVM	13.164	79.9960	102.924

DT	12.233	74.1633	88.0982

In the load prediction under Pulse cyber-attack forecasting as shown on Table 13 DT has the best results. Although, DT has the lowest errors and best error results when compared to NARX, LSTM and SVM, ultimately, the NARX method that is shown in figure 5.1.2 has the best fit amongst the other methods. Thus, NARX is the closest and more accurate as it follows the trail of the original data under pulse attack. NARX has a mean absolute percentage error (MAPE) of 16.446, the mean absolute error (MAE) of 113.2301, and the root mean square error (RMSE) of 178.3073. Although its standard deviation value was greater than all the other methods. The values in Table 13 agrees with the results of the chart shown in Figure 5.1.2.

The differences between the REFIT and AMPds datasets under pulse attack, it is detected that there is a huge distinguishes between the two datasets. This is because the data of the REFIT dataset size which is almost five times the size of the AMPds dataset. In addition, with the AMPds dataset, NARX works well compared to LSTM. This is because it is a fluctuating dataset, while LSTM works better with steady data like the REFIT dataset. Moreover, LSTM is better with a bigger dataset this explains why it showed better results when training the REFIT dataset. SVM and DT both worked better with the steady dataset, which is REFIT, although, their MAE values were better in the AMPds dataset.

5.3 The load prediction Under Scale cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

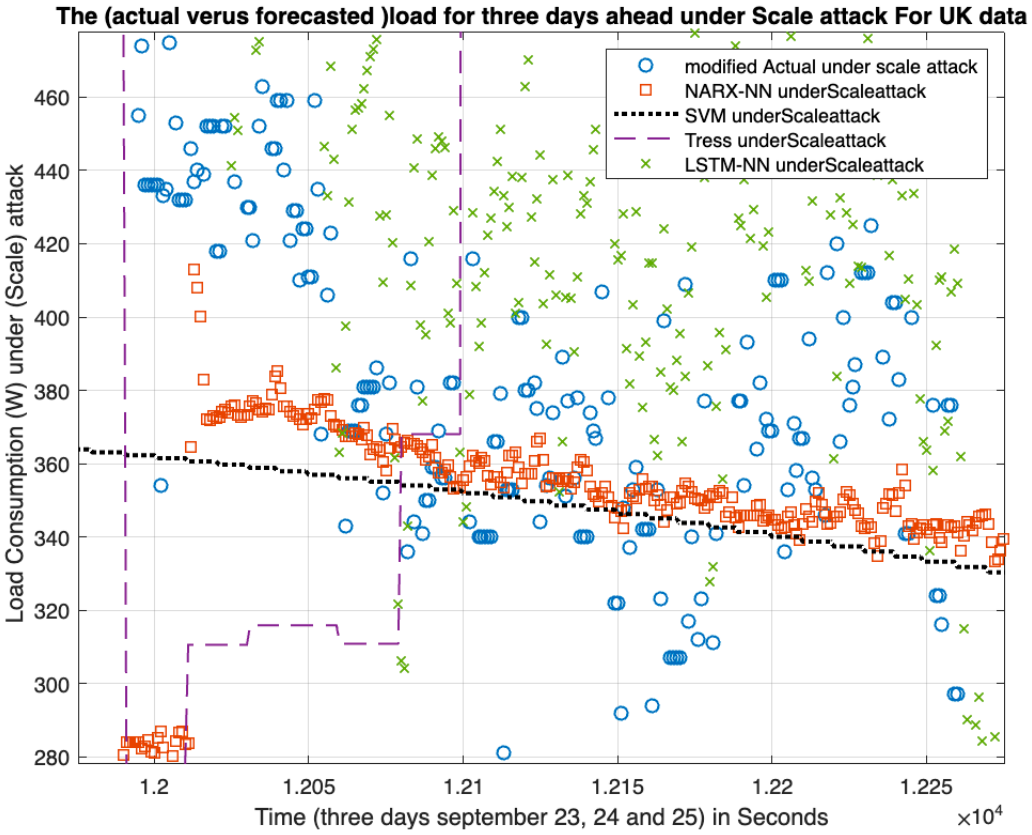


Figure 5.2.1 The REFIT load prediction under Scaling cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

Table 14 Errors summaries for the REFIT dataset of the load prediction under Scaling cyber-attack forecasting vs Actual

	MAPE	MAE	RMSE	SD
NARX	1.5369	419.2005	6.5999	926.3797
LSTM	**0.3**	**56.8**	**0.7**	**1029**
SVM	0.4069	260.0093	5.1527	234.4309
DT	1.4374	388.3488	5.4412	294.8280

In the load prediction under Scaling cyber-attack forecasting vs actual as shown on Table 14 LSTM has the best results. The LSTM method has a mean absolute percentage error (MAPE) of 0.3, mean absolute error (MAE) of 56.8, root mean square error (RMSE) of 0.7, and standard deviation (SD) of 1029. Although its standard deviation value was greater than all the other methods. The SVM method has the lowest SD value which is 234.4309.

In Figure 5.2.1, LSTM represents the green x-shape in the chart which follows the actual data's blue circle most accurately compared to the other methods. Hence, Table 14 agrees with Figure 5.2.1, as they both depict LSTM as being the best method under the scaling cyber-attack.

The load predictions under scale cyber-attack for the AMPds dataset using NARX, LSTM, SVM and DT vs actual shown in fig.5.2.2. below.

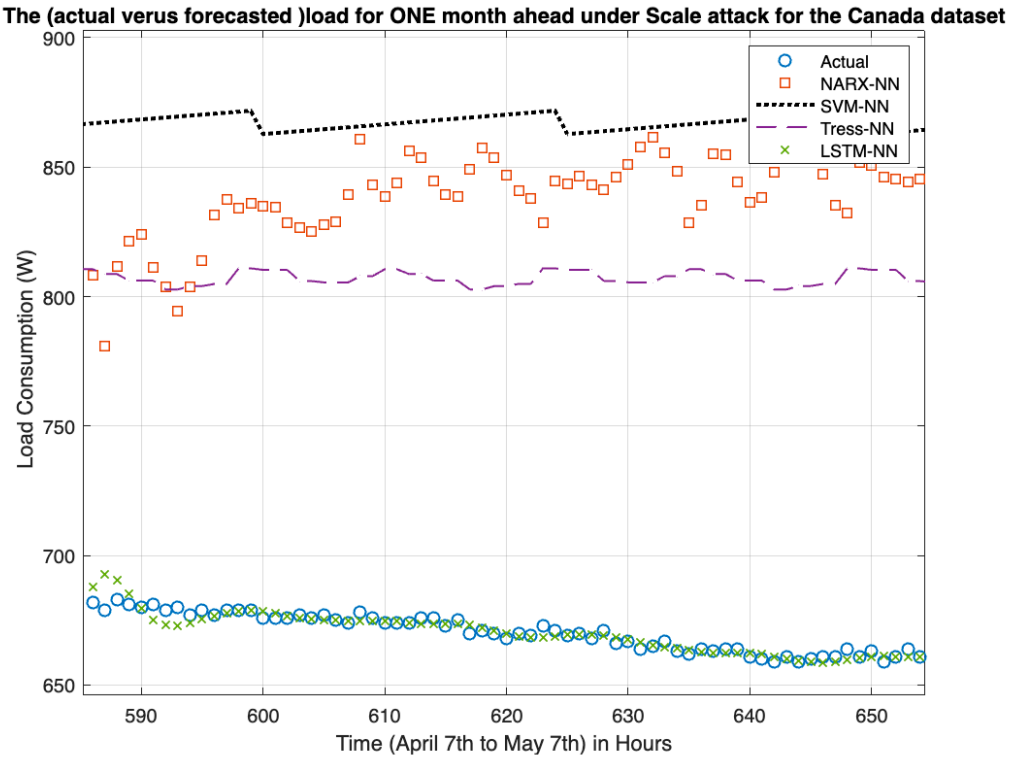


Figure 5.2.2 The Canada AMPds load prediction under scale cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

In Figure 5.2.2, LSTM represents the green x-shaped in the chart which follows the actual data's blue circle most accurately compared to the other methods. This means that in the case where the AMPds data is under the scaling cyber-attack, the method LSTM works best.

Table 15 Errors summaries for the AMPds dataset of the load prediction forecasting under Scaling attack vs actual

	MAPE	**MAE**	**RMSE**
NARX	**1.104**	**8.22**	**31.284**
LSTM	36.313	321.766	378.325
SVM	32.962	204.141	239.539
DT	31.195	208.320	231.817

In the load prediction Under Scale cyber-attack forecasting of the AMPds dataset as shown on Table 15, NARX has the best results. The NARX method has a mean absolute percentage error (MAPE) of 1.104, mean absolute error (MAE) of 8.22, and the root mean square error (RMSE) of 31.284.

For the AMPds data, in scaling, there is no evident change in the connections of the data. There is no fluctuation because the characteristics of the data are unchanged. Under scale LSTM works the best based on figure 5.2.2 because there is no fluctuation and data are not affected as much as they are in other cyberattacks. This observation is justifiable based on the equations that were used in Table 11. However, Table 15 does not agree with the chart on Figure 5.2.2.

When comparing the REFIT and AMPds datasets under scale attack, it is detected that there are some differences between the two datasets results. The REFIT dataset results that are

shown in Table 14 and Figure 5.2.1 both showed the same results and agreed that LSTM has the best performance. However, in the AMPds dataset the results shown in Table 15 and Figure 5.2.2 differed greatly. This is because NARX performed the best in terms of values in Table 15, while LSTM showed the best accuracy in following the actual data's prediction line making it the best method in Figure 5.2.2.

NARX works well compared to LSTM values in the AMPds dataset only in the error table. LSTM is better with a bigger dataset this explains why it showed better results when training the REFIT dataset.

5.4 The load prediction Under Ramping cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

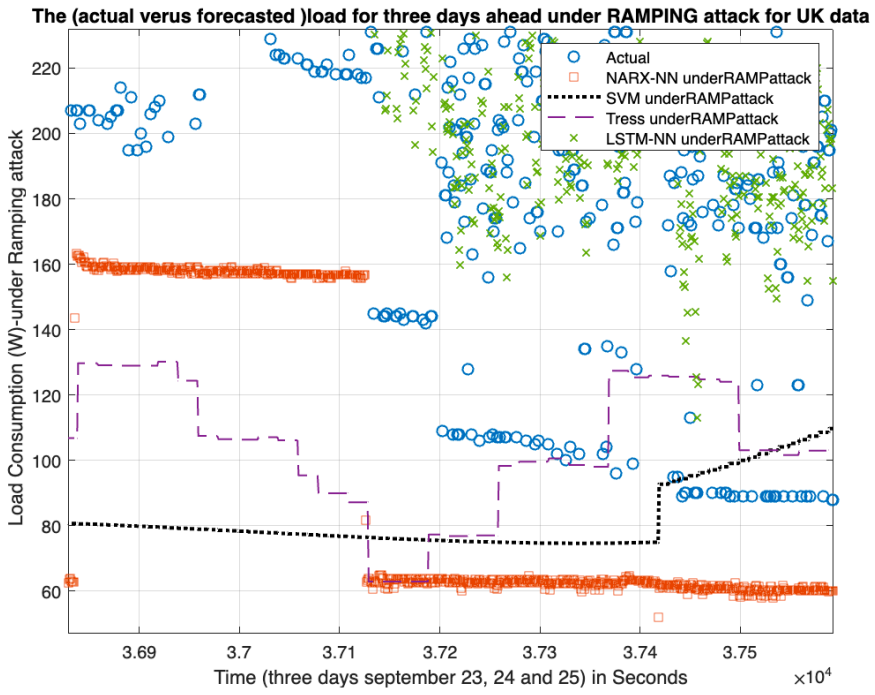


Figure 5.3.1 The REFIT load prediction Under ramping cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

In Figure 5.3.1, LSTM represents the green x-shape in the chart which follows the actual data's blue circle most accurately compared to the other methods. This means that in the case where the REFIT data is under the ramping cyber-attack, the method LSTM works best.

Table 16 Errors summaries for the REFIT dataset of the load prediction under Ramping cyber-attack forecasting vs actual

	MAPE	MAE	RMSE	SD
NARX	1.7043	433.8518	6.6881	950.8665
LSTM	**0.1**	**33.3**	**0.6**	**1005.4**
SVM	0.4040	259.9630	5.1534	235.2796
DT	1.6230	401.8941	5.4864	294.8280

In the load prediction under Ramping cyber-attack forecasting as shown on Table 16 LSTM has the best results. The LSTM method has a mean absolute percentage error (MAPE) of 0.1, mean absolute error (MAE) of 33.3, root mean square error (RMSE) of 0.6, and standard deviation (SD) of 1005.4. Although its standard deviation value was greater than all the other methods. The SVM method has the lowest SD value which is 235.2796.

The load predictions under ramping cyber-attack for the AMPds dataset using NARX, LSTM, SVM and DT vs actual shown in fig 5.3.2. below.

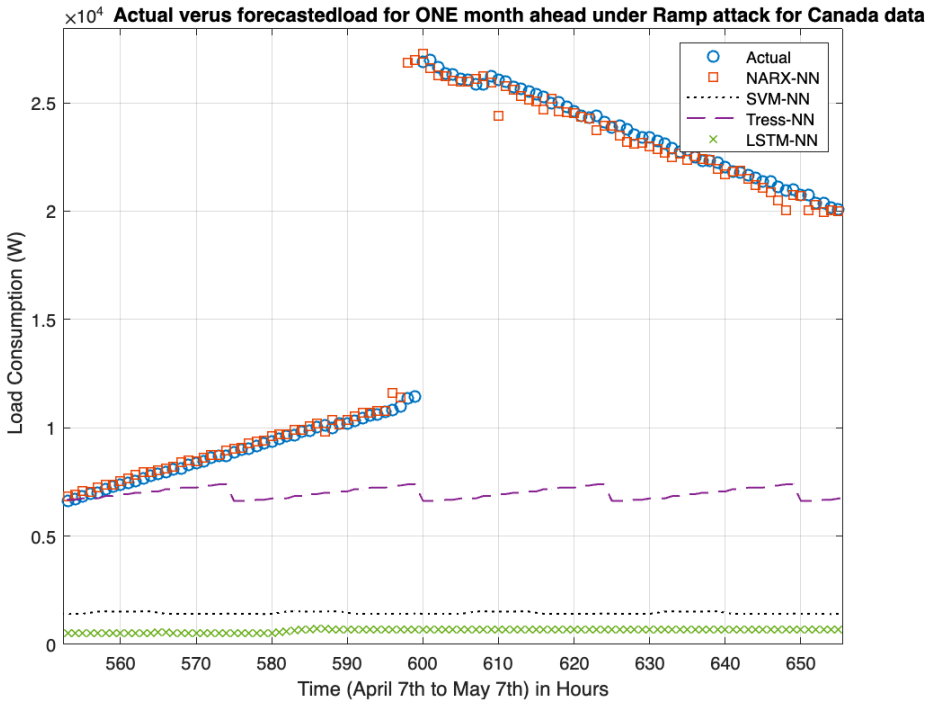


Figure 5.3.2 The Canada AMPds load prediction under ramping cyber attack forecasting (NARX, LSTM, SVM and DT) vs Actual

In Figure 5.3.2, NARX represents the red square in the chart which follows the actual data's blue circle most accurately compared to the other methods. This means that in the case where the AMPds data is under the ramping cyber-attack, the method NARX works best.

Table 17 Errors summaries for the AMPds dataset of the modified original load forecasting under Ramping cyber-attack vs actual

	MAPE	MAE	RMSE
NARX	**4.684**	**445.6099**	**3105.121**
LSTM	55.897	6.4456e+3	11000.544
SVM	112.102	6379.063	10517.745
DT	573.532	7221.648	8940.199

In the load prediction under ramping cyber-attack forecasting of the AMPds dataset as shown on Table 17, NARX has the best results. The NARX method has a mean absolute percentage error (MAPE) of 4.684, mean absolute error (MAE) of 445.8099, and the root mean square error (RMSE) of 3105.121.

As seen on Figure 5.3.2, the ramping attacks type 2 is observed. The type 2 of ramping attack happens at the 500th hour mark and at the 600th hour mark. NARXs works best here because the data is fluctuating.

Under the ramping cyber-attack, the two datasets showed different results as usual. The REFIT dataset had LSTM as the best method and the AMPds dataset has NARX as the best method. This is because AMPds is a fluctuating dataset, while LSTM works better with steady data like the REFIT dataset. Also, LSTM is better with a bigger dataset this explains why it showed better results when training the REFIT dataset.

5.5 The load prediction Under Random cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

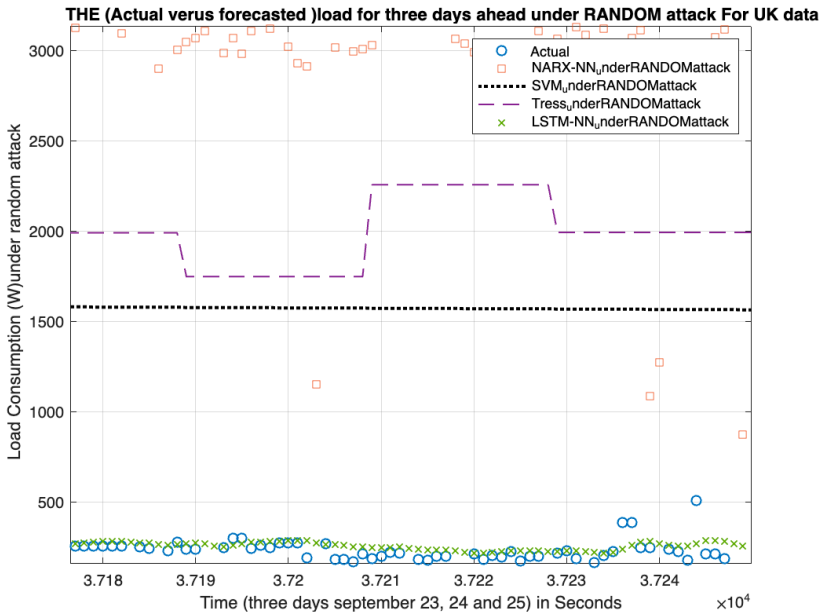


Figure 5.4.1 The REFIT load prediction under Random cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

In Figure 5.4.1, LSTM represents the green x-shape in the chart which follows the actual data's blue circle most accurately compared to the other methods. This means that in the case where the REFIT data is under the random cyber-attack, the method LSTM works best.

Table 18 Errors summaries for the REFIT dataset of the load prediction under Random cyber-attack forecasting

	MAPE	MAE	RMSE	SD
NARX	19.2	2291.1	13.5	2634.1
LSTM	**0.4855**	**108.3814**	**1.6823**	**853.3391**
SVM	66	6196.8	42.2	8574.5
DT	19.3	2244.4	11.7	1645.5

In the load prediction under Random cyber-attack as shown on Table 18 LSTM has the best results. The LSTM method has a mean absolute percentage error (MAPE) of 0.4855, mean absolute error (MAE) of 108.3814, root mean square error (RMSE) of 1.6823, and standard deviation (SD) of 853.3391. Table 18 agrees with the chart shown in Figure 5.4.1.

The load predictions under random cyber-attack forecasting for the AMPds dataset using NARX, LSTM, SVM and DT vs actual is shown in fig 5.4.2. below.

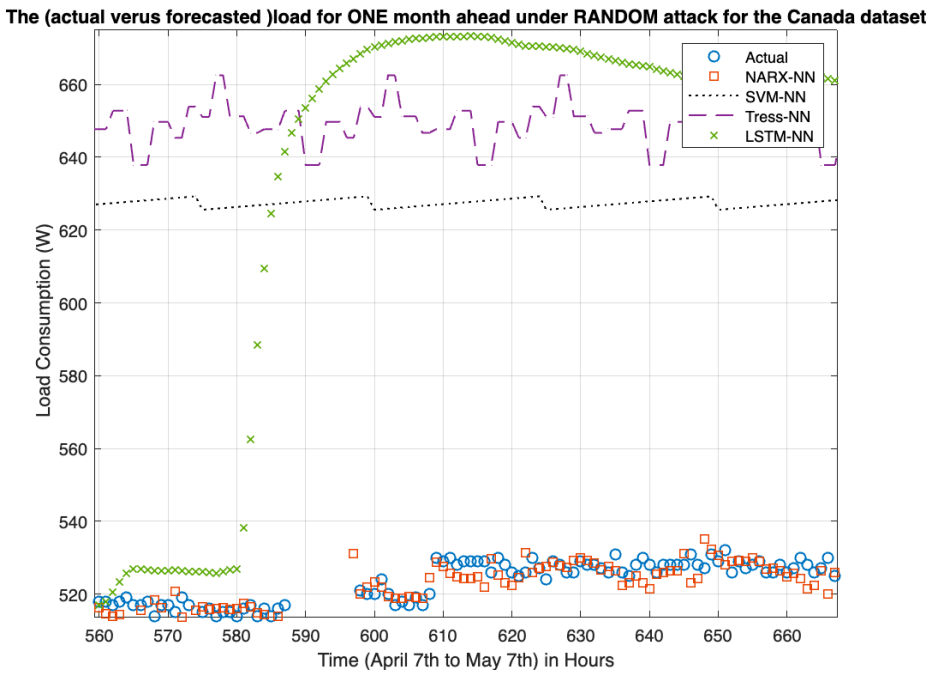


Figure 5.4.2 The Canada AMPds load prediction under random cyber-attack forecasting (NARX, LSTM, SVM and DT) versus Actual

In Figure 5.4.2, NARX represents the red square shape in the chart which follows the actual data's blue circle most accurately compared to the other methods. This means that in the case where the AMPds data is under the random cyber-attack, the method NARX works best.

Table 19 Errors summaries of the load prediction under Ramping cyber-attack forecasting of the AMPds dataset vs actual

	MAPE	MAE	RMSE
NARX	**5.162**	**39.5246**	**68.361**
LSTM	28.287	200.502	245.638
SVM	18.987	124.689	157.441
DT	19.645	124.668	155.755

In the load prediction under random cyber-attack forecasting of the AMPds dataset as shown on Table 16, NARX has the best results. The NARX method has a mean absolute percentage error (MAPE) of 5.162, mean absolute error (MAE) of 39.5246, and the root mean square error (RMSE) of 68.361. Table 16 agrees with the chart shown in Figure 5.4.2.

The REFIT dataset under ramping attack compared to the AMPds dataset as stated above in the sections discussing the other cyber-attack results the datasets results differ when it comes to the best method. In the case of a random cyber-attack the best result for the REFIT dataset was LSTM, while it was NARX for the AMPds dataset. To restate the reason, the AMPds dataset is a fluctuating dataset, while LSTM works better with steady data like the REFIT dataset. Also, LSTM is better with a bigger dataset this explains why it showed better results when training the REFIT dataset.

Although, Tables 13, 14, 15, 16 and 17 indicate that LSTM outperformed all the other methods simply based on having the lowest values for the error performances. Ultimately, LSTM is the method that varied the most in error performance. The LSTM error values changed the most between each cyber-attack which implied that it is more vulnerable to attacks.

When LSTM is compared to NARX it is important to understand that they both have different ANN methodology. LSTM are recurrent neural networks, while NARX are autoregressive models [68]. In the results shown earlier, NARX has a higher forecasting error than LSTM due to the predicted output being fed back to the feedback network instead of the actual output [68]. This carries on the error further into the predictions which increases the errors [68]. On the other hand, LSTM has cell memory that can handle long-term dependencies, because

information can remain in the memory for many steps [68]. LSTM can remember or forget the data efficiently, whereas SVM cannot [69]. Also, the gating system of LSTM adds to its efficiency in predicting. This is because LSTM gating system filters out irrelevant input information, thus, predicting in a higher accuracy rate [70].

5.6 Load Predictions for Data Under All Cyberattacks types (pulse, scale, ramping and random)

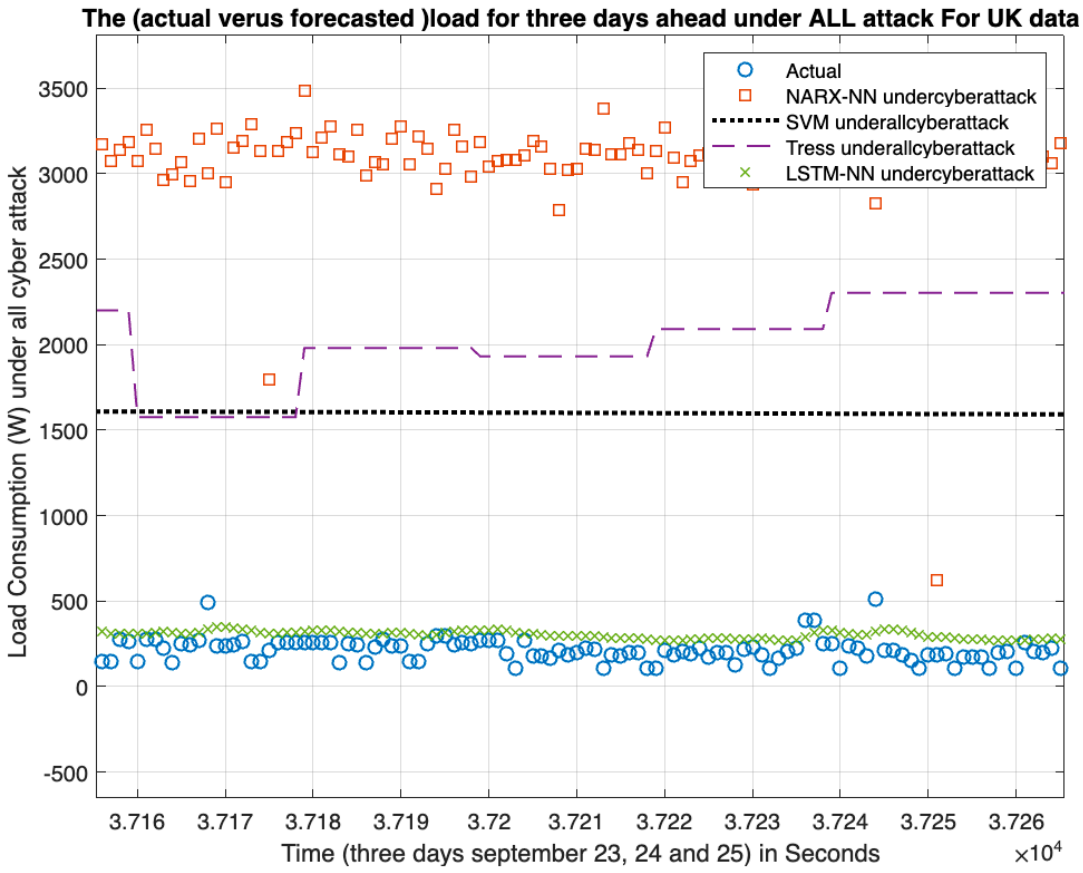


Figure 5.5.1 The load prediction under all cyber-attack forecasting (NARX, LSTM, SVM and DT) vs Actual

In Figure 5.5.1, LSTM represents the green x-shape in the chart which follows the actual data's blue circle most accurately compared to the other methods. This means that in the case where the REFIT data is under the combined all four cyber-attacks, the method LSTM works best.

Table 20 Errors summaries for the REFIT dataset of the load prediction under All four cyber-attack models forecasting

	MAPE	MAE	RMSE	SD
NARX	19.4	2295.7	13.6	2636.9
LSTM	**0.9692**	**161.7596**	**1.7672**	**808.2965**
SVM	15.4	1852.8	9.8	1795.5
DT	19.5	2258.8	11.8	2380.1

In the load prediction under all cyber-attack models forecasting as shown in Table 20 LSTM has the best results. The LSTM method has a mean absolute percentage error (MAPE) of 0.9692, mean absolute error (MAE) of 161.7596, root mean square error (RMSE) of 1.7672, and standard deviation (SD) of 808.2965. Table 20 agrees with the chart on Figure 5.5.1.

The load predictions under the combined all four cyber-attacks for the AMPds dataset using NARX, LSTM, SVM and DT is shown in fig 5.5.2. below.

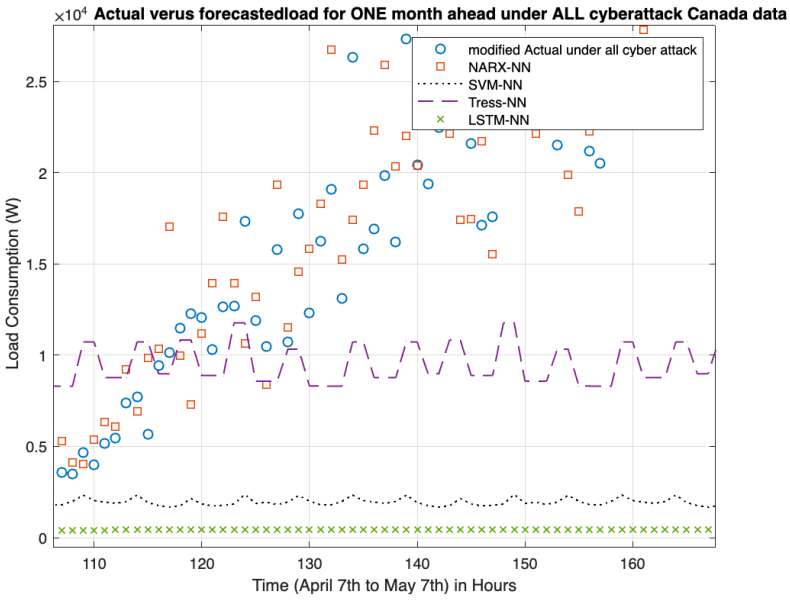


Figure 5.5.2 The Canada AMPds load prediction under all cyber-attack forecasting (NARX, LSTM, SVM and DT) vs Actual

In Figure 5.5.2, NARX represents the red square shape in the chart which follows the actual data's blue circle most accurately compared to the other methods. This means that in the case where the AMPds data is under the combined all four cyber-attacks, the method NARX works best.

Table 21 Errors summaries for the AMPds dataset of the load prediction under all cyber-attack forecasting

	MAPE	MAE	RMSE
NARX	**56.436**	**2.7693e+03**	**11335.318**
LSTM	64.511	9.0060e+03	17406.836
SVM	99.492	8.7124e+03	16725.339
DT	613.153	11457.692	14877.364

In the load prediction under the combined all four cyber-attacks of the AMPds dataset forecasting as shown on Table 21, NARX has the best results. The NARX method has a mean absolute percentage error (MAPE) of 56.436, mean absolute error (MAE) of 2.7693e+03, and the root mean square error (RMSE) of 11335.318. Table 21 agrees with the chart on Figure 5.5.2.

The REFIT dataset under the combined all four cyber-attacks compared to the AMPds dataset results differ when it comes to the best method. In the case of combined all four cyber-attacks the best result for the REFIT dataset was LSTM, while it was NARX for the AMPds dataset. To restate the reason, the AMPds dataset is a fluctuating dataset, while LSTM works better with steady data like the REFIT dataset. Also, LSTM is better with a bigger dataset this explains why it showed better results when training the REFIT dataset.

Although, the best methods for the two datasets were determined and the best method result was like the other tests done in Sections 5.2, 5.3, 5.4 and 5.5, ultimately, in this section the cyber-attack is the worst case since all the cyber-attacks are combined to modify the original data. The magnitude went over the reasonable region making the results unreliable.

Conclusion:

This work analyzed the vulnerability of four representative machine learning algorithms for residential short-term load forecast against cyberattacks, including Nonlinear Auto Regression with external input (NARX) neural network, support vector machine (SVM), decision tree (DT), and long-short-term memory (LSTM) deep learning. Four cyberattack models are used, namely, pulse, scale, ramp, and random attacks. The REFIT open dataset is used for the assessment as well as the AMPds dataset. The REFIT dataset collected whole-house aggregated loads at 8-second intervals continuously from 20 houses over a two-year period in the U.K. The AMPds data set were sampled in hours from 6th of March 2016 to 7th of May 2016. The vulnerability assessment results indicate the LSTM provides the most accurate prediction without cyberattacks. However, the prediction accuracy of the LSTM fluctuates most, compared to other machine learning methods, when there are cyberattacks. The AMPds dataset graphs presented sharper results compared to the REFIT dataset. Under scale attack, LSTM works the best because there is no fluctuation and data are not affected as much as they are in other cyberattacks. Among the four cyberattacks, the random attack triggered the largest variations on the predication results. The test where all four of the cyber-attacks are used to modify the original data, the magnitude went over the reasonable region. This resulted with the worst results for all the four neural network methods. Ultimately, the two datasets differed greatly when it comes to their size, results, and fluctuation in values. The common pattern observed when studying the results for the two datasets is that NARX works better with the AMPds set because it is a fluctuating dataset, while LSTM works better with steady data like the REFIT dataset. In addition, LSTM is better with a bigger dataset this explains why it showed better results when training the REFIT dataset.

For future work it is critically viable to test the two datasets under different machine learning methods to further study their patterns.

References

[1] ortega19, “A guide to home energy management systems (HEMS),” *KNX*, 04-Nov-2021.

[2] H. Zandi, T. Kuruganti, E. A. Vineyard, and D. Fugate, “Home Energy Management Systems: An overview,” *Home Energy Management Systems: An Overview (Conference) | OSTI.GOV*, 01-Jan-2018.

[3] P. Palensky and D. Dietrich, "Demand Side Management: Demand Response, Intelligent Energy Systems, and Smart Loads," *in IEEE Transactions on Industrial Informatics*, vol. 7, no. 3, pp. 381-388, Aug.2011

[4] S. Mill, “Electric Load Forecasting: Advantages and Challenges,” Electrical Equipment, 20-Oct-2016.

[5] E. Almeshaiei and H. Soltan, “A methodology for Electric Power Load Forecasting,” *Alexandria Engineering Journal*, vol. 50, no. 2, pp. 137–144, 2011.

[6] K. Metaxiotis, A. Kagiannas, D. Askounis, and J. Psarras, “Artificial Intelligence in short term electric load forecasting: A state-of-the-art survey for the researcher,” *Energy Conversion and Management,* vol. 44, no. 9, pp. 1525–1534, 2003.

[7] S. R. Khuntia, J. L. Rueda and M. A. M. M. van der Meijden, "Forecasting the load of electrical power systems in mid- and long-term horizons: a review," *in IET Generation, Transmission & Distribution,* vol. 10, no. 16, pp. 3971-3977, 8 12 2016.

[8] A. Hansen, J. Staggs, and S. Shenoi, “Security analysis of an advanced metering infrastructure,” *International Journal of Critical Infrastructure Protection*, vol. 18, pp. 3–19, 2017.

[9] M. Cui, J. Wang, and M. Yue, “Machine learning-based anomaly detection for load forecasting under cyberattacks,” *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5724–5734, 2019.

[10] Firth, Steven; Kane, Tom; Dimitriou, Vanda; Hassan, Tarek; Fouchal, Farid; Coleman, Michael; et al. (2017): REFIT Smart Home dataset. figshare. Dataset. https://doi.org/10.17028/rd.lboro.2070091.v1

[11] Makonin, S., Ellert, B., Bajić, I. et al. “Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014.” *Sci Data3*, 160037 (2016).

[12] Papalexopoulos, Alex D., and Timothy C. Hesterberg. "A regressionbased approach to short-term system load forecasting." *IEEE Transactions on Power Systems,* 5.4 (1990): 1535-1547.

[13] Mbamalu, G. A. N., and M. E. El-Hawary. "Load forecasting via suboptimal seasonal autoregressive models and iteratively reweighted least squares estimation." *IEEE Transactions on Power Systems*, 8.1 (1993): 343-348.

[14] Huang, S. R. "Short-term load forecasting using threshold autoregressive models." *IEE Proceedings-Generation, Transmission and Distribution*, 144.5 (1997): 477-481.

[15] Zhang, Min, et al. "Research on processing of short-term historical data of daily load based on Kalman filter." *Power System Technology,* 10.009 (2003): 200.

[16] Chen, Jiann-Fuh, Wei-Ming Wang, and Chao-Ming Huang. "Analysis of an adaptive time-series autoregressive moving-average (ARMA) model for short-term load forecasting." *Electric Power Systems Research,* 34.3 (1995): 187-196.

[17] Pai, Ping-Feng, and Wei-Chiang Hong. "Forecasting regional electricity load based on recurrent support vector machines with genetic algorithms." *Electric Power Systems Research*, 74.3 (2005): 417-425.

[18] Yang, YouLong, et al. "An incremental electric load forecasting model based on support vector regression." *Energy* 113 (2016): 796-808.

[19] Pandian, S. Chenthur, et al. "Fuzzy approach for short term load forecasting." *Electric power systems research* 76.6-7 (2006): 541-548.

[20] Kouhi, Sajjad, and Farshid Keynia. "A new cascade NN based method to short-term load forecast in deregulated electricity market." *Energy Conversion and Management* 71 (2013): 76-83.

[21] Xiao, Liye, et al. "Research and application of a combined model based on multi-objective optimization for electrical load forecasting." *Energy* 119 (2017): 1057-1074.

[22] Xiao, Liye, et al. "Research and application of a hybrid model based on multi-objective optimization for electrical load forecasting." *Applied energy* 180 (2016): 213-233.

[23] Lusis, Peter, et al. "Short-term residential load forecasting: Impact of calendar effects and forecast granularity." *Applied Energy* 205 (2017): 654-669.

[24] Chahkoutahi, Fatemeh, and Mehdi Khashei. "A seasonal direct optimal hybrid model of computational intelligence and soft computing techniques for electricity load forecasting." Energy 140 (2017): 988- 1004.

[25] Tong, Chao, et al. "An efficient deep model for day-ahead electricity load forecasting with stacked denoising auto-encoders." *Journal of parallel and distributed computing* 117 (2018): 267-273.

[26] Li, Song, Lalit Goel, and Peng Wang. "An ensemble approach for shortterm load forecasting by extreme learning machine." *Applied Energy* 170 (2016): 22-29.

[27] FEDERAL BUREAU OF INVESTIGATION. (2010) Cyber intelligence section: Smart grid electric meters altered to steal electricity.

[28] J. Luo, T. Hong, and S.-C. Fang, “Benchmarking robustness of load forecasting models under data integrity attacks,” *Int. J. Forecasting*, vol. 34, no. 1, pp. 89–104, 2018.

[29] M. Yue, T. Hong and J. Wang, "Descriptive Analytics-Based Anomaly Detection for Cybersecure Load Forecasting," *in IEEE Transactions on Smart Grid*, vol. 10, no. 6, pp. 5964-5974, Nov. 2019

[30] M. Cui, J. Wang and M. Yue, "Machine Learning-Based Anomaly Detection for Load Forecasting Under Cyberattacks," *in IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5724-5734, Sept. 2019

[31] Y. Deng, K. Zhu, R. Wang and Y. Wan, "Real-time Detection of False Data Injection Attacks Based on Load Forecasting in Smart Grid," *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2019, pp. 1-6

[32] Y. Chen, Y. Tan, L. Zhang and B. Zhang, "Vulnerabilities of Power System Operations to Load Forecasting Data Injection Attacks," *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2021, pp. 398-404

[33] K. Wakefield, “A guide to the types of machine learning algorithms,” A guide to the types of machine learning algorithms | SAS UK.

[34] L. R. Medsker, Hybrid Neural Network and expert systems. Boston: Kluwer Academic, 1994.

[35] D. Fumo, “Types of machine learning algorithms you should know,” Medium, 17-Aug-2017.

[36] By: IBM Cloud Education, “What is supervised learning?,” IBM.

[37] S. L. AI, “Reinforcement learning algorithms - an intuitive overview,” Medium, 18-Feb-2019.

[38] B. Osiński, “What is reinforcement learning? The Complete Guide,” deepsense.ai, 15-Feb-2022.

[39] Lecun, Yann, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning.” *Nature (London)* 521.7553 (2015): 436–444. Web.

[40] C. Fan, F. Xiao, and Y. Zhao, “A short-term building cooling load prediction method using deep learning algorithms,” *Applied Energy*, vol. 195, pp. 222–233, 2017.

[41] Mueller, John Paul., and Luca Massaron. Deep Learning. 1st edition. Hoboken, New Jersey: For Dummies, 2019. Print.

[42] R. Khalid, N. Javaid, F. A. Al-Zahrani, K. Aurangzeb, E.-U.-H. Qazi, and T. Ashfaq, "Electricity load and price forecasting using Jaya-Long Short Term Memory (JLSTM) in smart grids," Entropy (Basel, Switzerland), 19-Dec-2019.

[43] G. F. Martin Nascimento, F. Wurtz, P. Kuo-Peng, B. Delinchant, and N. Jhoe Batistela, "Outlier detection in buildings' power consumption data using forecast error," MDPI, 10-Dec-2021.

[44] "Normalization | machine learning | google developers," Google.

[45] "Normalization by Standard Deviation," Normalization by standard deviation.

[46] C. Stergiou and D. Signs, "Neural networks."

[47] "Brain basics: The life and death of a neuron," National Institute of Neurological Disorders and Stroke.

[48] US National Cancer Institute, "Anatomy & Physiology," Anatomy & Physiology | SEER Training.

[49] A. Rajput, "Introduction to ann: Set 4 (Network Architectures)," GeeksforGeeks, 09-Nov-2021.

[50] N. J. Sairamya, L. Susmitha, S. Thomas George, and M. S. P. Subathra, "Hybrid approach for classification of electroencephalographic signals using time–frequency images with wavelets and texture features," *Intelligent Data Analysis for Biomedical Applications*, pp. 253–273, 2019.

[51] M. Trebar, "Use of Matlab Neural Networks Toolbox in a character recognition problem," *Computer Applications in Engineering Education*, vol. 13, no. 1, pp. 66–71, 2005.

[52] S. T. Mehmood and M. El-Hawary, "Performance evaluation of new and Advanced Neural Networks for short term load forecasting," 2014 *IEEE Electrical Power and Energy Conference*, 2014.

[53]. U. Karn, "A quick introduction to Neural Networks," KDnuggets.

[54] Visual Numerics Inc, "Multilayer Feedforward Neural Networks," CNLSTAT.

[55] "Feedback Process Neural Networks," SpringerLink, 01-Jan-1970.

[56] By: IBM Cloud Education, "What are recurrent neural networks?," IBM.

[57] S. Elsworth and S. Guttel, "Time Series Forecasting Using LSTM Networks: A Symbolic Approach," Mar. 2020.

[58] B. Warsito, R. Santoso, Suparti, and H. Yasin, “Cascade Forward Neural Network for time series prediction,” *Journal of Physics: Conference Series*, vol. 1025, p. 012097, 2018.

[59] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[60] Campbell, Colin., and Yiming. Ying. Learning with Support Vector Machines. San Rafael, Calif. (1537 Fourth Street, San Rafael, CA 94901 USA): Morgan & Claypool, 2011. Web.

[61] P. Prashar and T. Choudhury, "Suicide Forecast System Over Linear Regression, Decision Tree, Naïve Bayesian Networks and Precision Recall," *2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence),* Noida, 2018, pp. 310-313

[62] Q. Ding, "Long-Term Load Forecast Using Decision Tree Method," *2006 IEEE PES Power Systems Conference and Exposition*, *Atlanta, GA,* 2006, pp. 1541-1543.

[63] L. Dongming, L. Yan, Y. Chao, L. Chaoran, L. Huan and Z. Lijuan, "The application of decision tree C4.5 algorithm to soil quality grade forecasting model," *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI), Wuhan,* 2016, pp. 552-555.

[64] LSTM - notes on ai

[65] E. E. Elattar, J. Goulermas and Q. H. Wu, "Electric Load Forecasting Based on Locally Weighted Support Vector Regression," *in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 40, no. 4, pp. 438-447, July 2010.

[66] L. Fan, J. Li and X. Zhang, "Load prediction methods using machine learning for home energy management systems based on human behavior patterns recognition," *in CSEE Journal of Power and Energy Systems,* vol. 6, no. 3, pp. 563-571, Sept. 2020.

[67] M. Cui, J. Wang, and M. Yue, “Machine learning-based anomaly detection for load forecasting under cyberattacks,” *IEEE Transactions on Smart Grid,* vol. 10, no. 5, pp. 5724–5734, 2019.

[68] M. M. I. C. L. S. M. T. S. S. R. F. S. O. 2016, A Novel Approach Based Deep RNN Using Hybrid NARX-LSTM Model For Solar Power Forecasting.

[69] S. Paul, V. Kumar, and P. Jha, Artificial neural network and its applications: Unraveling the efficiency for hydrogen production. Punjab, india : elsevier, 2021.

[70] L. Sai Krishna, and M. John, A Comparative Study of SVM and LSTM Deep Learning Algorithms for Stock Market Prediction.

Appendix

NARX- NEURAL NETWORK PREDICTIONS:

USING THE ORIGINAL DATA SEPTEMBER20_21_22 TO TRAIN THE DATA AND PREDICTS K2 AND COMPARE IT WITH SEPTEMBER23_24_25

DATA USED FOR NARX PREDICTIONS:

I used September 20,21,22, 2013 data to train and test in order to predict September 23,24,25, 2013 data which is the next days predications and compare the actual data i.e. September 23,24,25, 2013 with K2 data (the predicted data).

% prediction code and finding K2 the predicted data in order to compare it with the actual data

```
Timeinput = timerateseptember20_21_22;
Targetload = september20_21_22;
M = max(Timeinput);
N = min(Timeinput);
J = max(Targetload);
L = min(Targetload);
A = (Timeinput - N)/(M - N);  %Normalization
B = (Targetload - L)/(J - L); %Normalization
inputSeries = (A.')
targetSeries = (B.')
X = tonndata(inputSeries,true,false);
T = tonndata(targetSeries,true,false);
trainFcn = 'trainbr';
inputDelays = 1:2;
feedbackDelays = 1:2;
hiddenLayerSize = (200);
net =
narxnet(inputDelays,feedbackDelays,hiddenLayerSize,'open',trainFcn);
% Prepare the Data for Training and Simulation
[x,xi,ai,t] = preparets(net,X,{},T);
% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Train the Network
[net,tr] = train(net,x,t,xi,ai);
% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)
% View the Network
view(net)
% Closed Loop Network
netc = closeloop(net);
netc.name = [net.name ' - Closed Loop'];
view(netc)
```

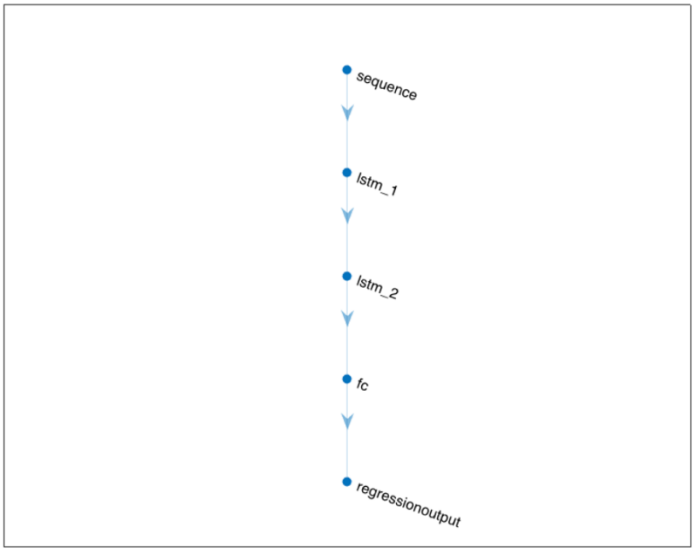
```
[xc,xic,aic,tc] = preparets(netc,X,{},T);
yc = netc(xc,xic,aic);
closedLoopPerformance = perform(net,tc,yc)
nets = removedelay(net);
nets.name = [net.name ' - Predict One Step Ahead'];
view(nets)
[xs,xis,ais,ts] = preparets(nets,X,{},T);
ys = nets(xs,xis,ais)
stepAheadPerformance = perform(nets,ts,ys)
Z = [ys{ : }]
K = (Z.')
K2 = K*(J - L) + L %denormalization
```

PREDICTIONS USING NEURAL NETWORK LSTM

FORECAST TIME SERIES DATA USING A LONG SHORT-TERM MEMORY (LSTM) NETWORK OF HOUSE2

LSTM STEPS:

- Load data
    - Data used in the forecasting were
    - september20_21_22
    - september23_24_25
- Define LSTM Network Architecture using APPs > Deep network designer
    - Use a sequence input layer with an input size that matches the number of channels of the input data, ours was 1 since we have one input.
    - Use an LSTM layer with 128 hidden units. The number of hidden units determines how much information is learned by the layer. Using more hidden units can yield more accurate results but can be more likely to lead to overfitting to the training data.
    - To output sequences with the same number of channels as the input data, include a fully connected layer with an output size that matches the number of channels of the input data.
    - Finally, include a regression layer.



- Specify Training Options
    - Train using Adam optimization.
    - Train for different numbers of epochs until a good fit was taken
        - as our data is a larger data sets and our epochs was 1000
    - In each mini-batch, left-pad the sequences so they have the same length. Left-padding prevents the network from predicting padding values at the ends of sequences.
    - Shuffle the data every epoch.
    - Display the training progress in a plot.
    - Disable the verbose output.
- Train Neural Network

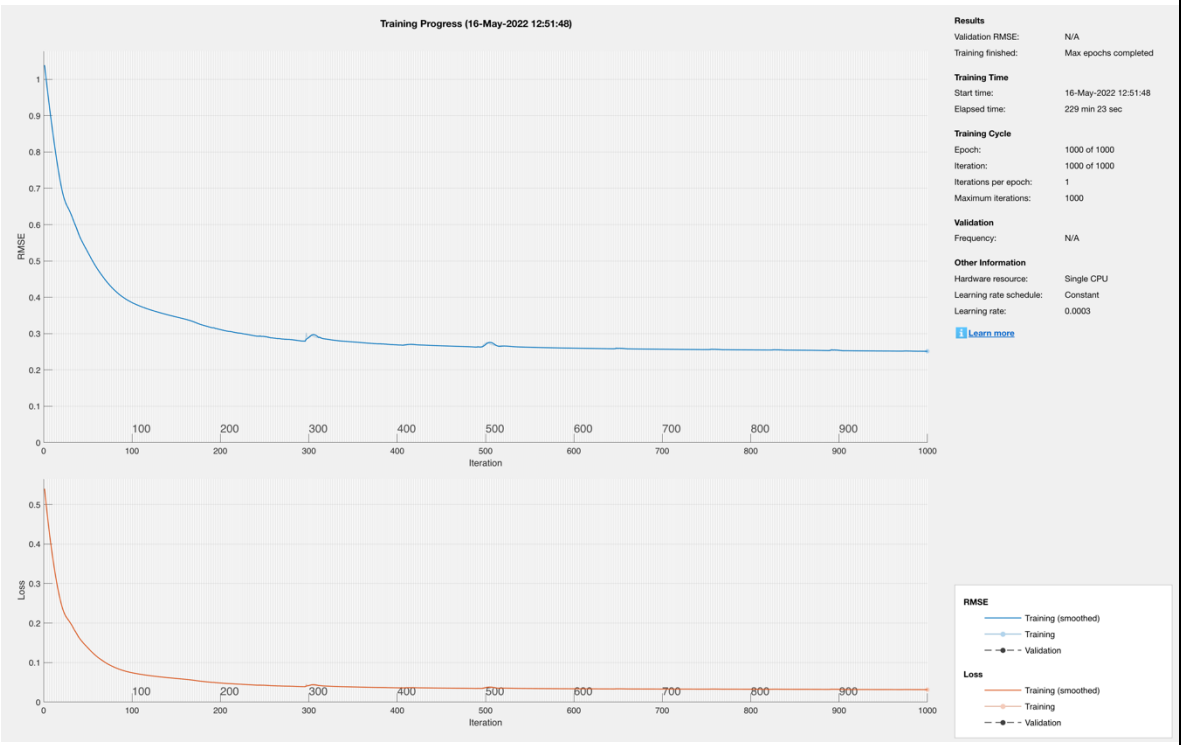
% prediction code and finding YTest_final the predicted data in order to compare it with the actual data

```
layers = [
    sequenceInputLayer(1)
    lstmLayer(512)
    fullyConnectedLayer(1)
    regressionLayer];


options = trainingOptions("adam", ...
```

```
    MaxEpochs=100, ...
    SequencePaddingDirection="left", ...
    Shuffle="every-epoch", ...
    Plots="training-progress", ...
    Verbose=0);


%divide the data
% N=floor(0.9*numel(september20_21_22));
dataTrain = september20_21_22;
%dataTest = september20_21_22(N+1:end);


%standardize data


muX = mean(dataTrain);
sigmaX = std(dataTrain);


dataTrainStandardized = (dataTrain - muX)/sigmaX;



    XTrain = dataTrainStandardized(1:end-1);

    TTrain = dataTrainStandardized(2:end);


net= trainNetwork(XTrain.',TTrain.',layers,options)


plot(layerGraph(layers));


%testing the data
%standardize data
muX=mean(september23_24_25);
sigmaX= std(september23_24_25);


dataTestStandardized = (september23_24_25- muX)/sigmaX;


XTest= dataTestStandardized(1:end-1);


TTest= dataTestStandardized(2:end);
```

```
YTest = predict(net,XTest.',SequencePaddingDirection="left");
net.Layers


      err = TTest - YTest;
%     errpct = abs((err)./TTest)*100; %Absolute percentage error
%     MAPE = mean(errpct(~isinf(errpct)));
%     fprintf('\nMean Absolute Percent Error (MAPE) for the forecasted
load of day ahead: %0.3f%%\n',MAPE);



YTest_Tempor=(YTest * sigmaX)+muX;
YTest_Final=YTest_Tempor.';
```

- Graph results of the Training:



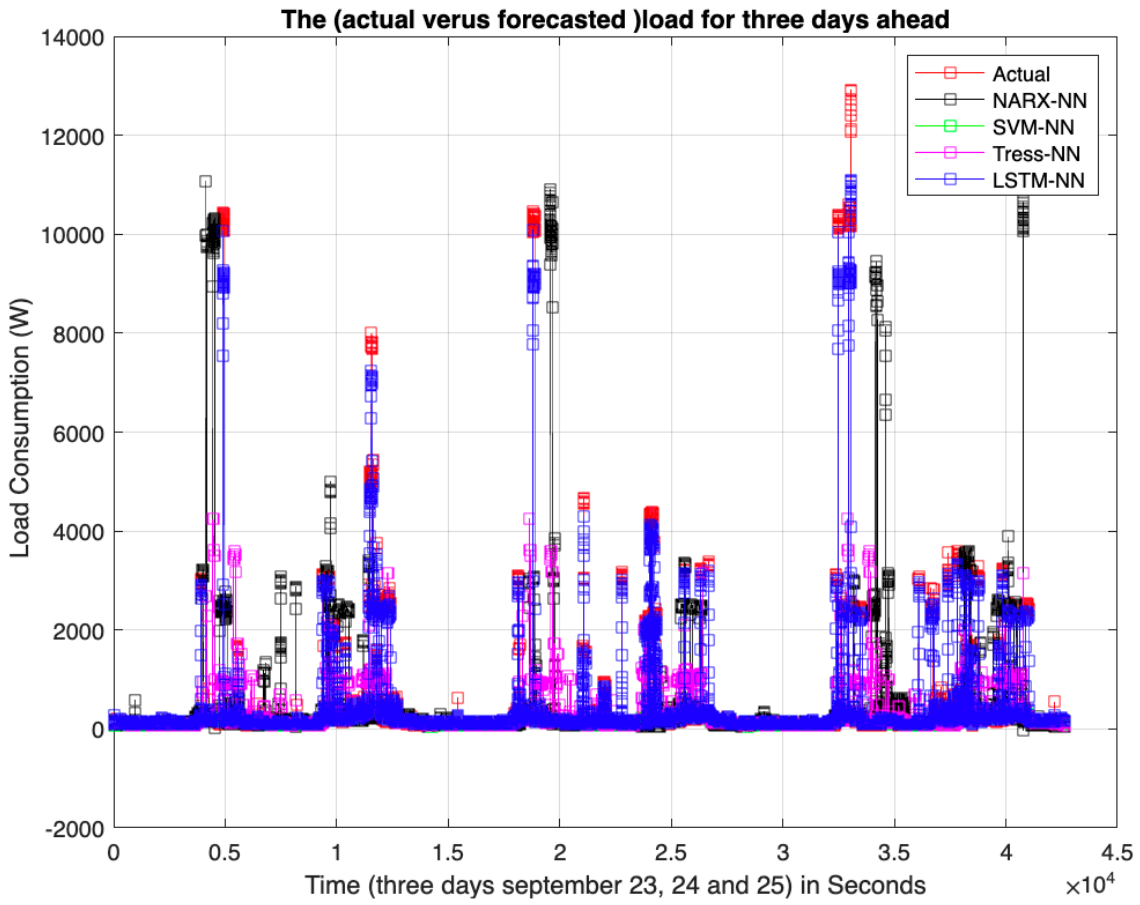
PREDICTIONS USING NEURAL NETWORK SVM AND TREE

USING THE APPS IN MATLAB- REGRESSION LINEAR WERE USED TO FIND NEURAL NETWORK PREDICTIONS (THE MINIMUM AND ACCURATE TREES AND WERE DETERMINED AND CHOOSEN TO PLOT AGAINST ACTUAL AND NARX AND LSTM):

NARX,LSTM,SVM AND TREE NEURAL NETWORK PREDICTIONS (GRAPH)-CODE VERSUS ACTUAL DATA, SVN, TREE AND GAUSSIAN METHODS:

```
q = (yfit_tree1.'); % coarse tree training data
Z = (yfit_SVM1.'); %SVM using fine gaussian training data
g = (YTest_Final.'); % LSTM
x = (september23_24_25.'); %actual data
y = (K2.'); % NEURAL NETWORK PREDITED DATA
%Z = (data.');
%q = (data.');
ts1 = timeseries(x,1:42657); %No. of data
ts1.Name = ' Load Consumption (W)';
ts1.TimeInfo.Units = 'second';
% Set start date.
%ts1.TimeInfo.StartDate = '0:00';
% Set format for display on x-axis.
ts1.TimeInfo.Format = 'SS';
ts1.Time = ts1.Time - ts1.Time(1);
%ts1 = setuniformtime(ts1,'Interval',20)
ts2 = timeseries(y,1:42657);
ts2.Name = ' Load Consumption (W)';
ts2.TimeInfo.Units = 'second';
%$ts2.TimeInfo.StartDate = '23-09-2013 00:00:00';  %starts from 12am
0:00
ts2.TimeInfo.Format = 'SS';  %goes with seconds, minutes, hours
ts2.Time = ts2.Time - ts2.Time(1);


ts3 = timeseries(Z,1:42657);
ts3.Name = ' Load Consumption (W)';
ts3.TimeInfo.Units = 'second';
%ts3.TimeInfo.StartDate = '0:00';
ts3.TimeInfo.Format = 'SS';
```

```
ts3.Time = ts3.Time - ts3.Time(1);


ts4 = timeseries(q,1:42657);
ts4.Name = ' Load Consumption (W)';
ts4.TimeInfo.Units = 'second';
%ts4.TimeInfo.StartDate = '0:00';
ts4.TimeInfo.Format = 'SS';
ts4.Time = ts4.Time - ts4.Time(1);


ts5 = timeseries(g,1:42657);
ts5.Name = ' Load Consumption (W)';
ts5.TimeInfo.Units = 'second';
%ts5.TimeInfo.StartDate = '0:00';
ts5.TimeInfo.Format = 'SS';
ts5.Time = ts5.Time - ts5.Time(1);


%ploting all methods together
%ts5.TimeInfo.Format = 'ss,mm,hh';
plot(ts1,'s-r','MarkerSize',6),
hold on
plot(ts2,'s-k','MarkerSize',6),
hold on
plot(ts3,'s-g','MarkerSize',6),
hold on
plot(ts4,'s-m','MarkerSize',6),
hold on
plot(ts5,'s-b','MarkerSize',6),
grid on


legend('show')
xlim('auto')
ylim('auto')
title(['The (actual verus forecasted )load for three days ahead'])
legend('Actual','NARX-NN','SVM-NN', 'Tress-NN', 'LSTM-NN' )
xlabel('Time (three days september 23, 24 and 25) in Seconds')


zoom on


xlim([0 15000])
ylim([-1440 11977])
```



THE ABOVE GRAPH REPRESENTED THE DIFFERENCES OF COLOURED LINES.

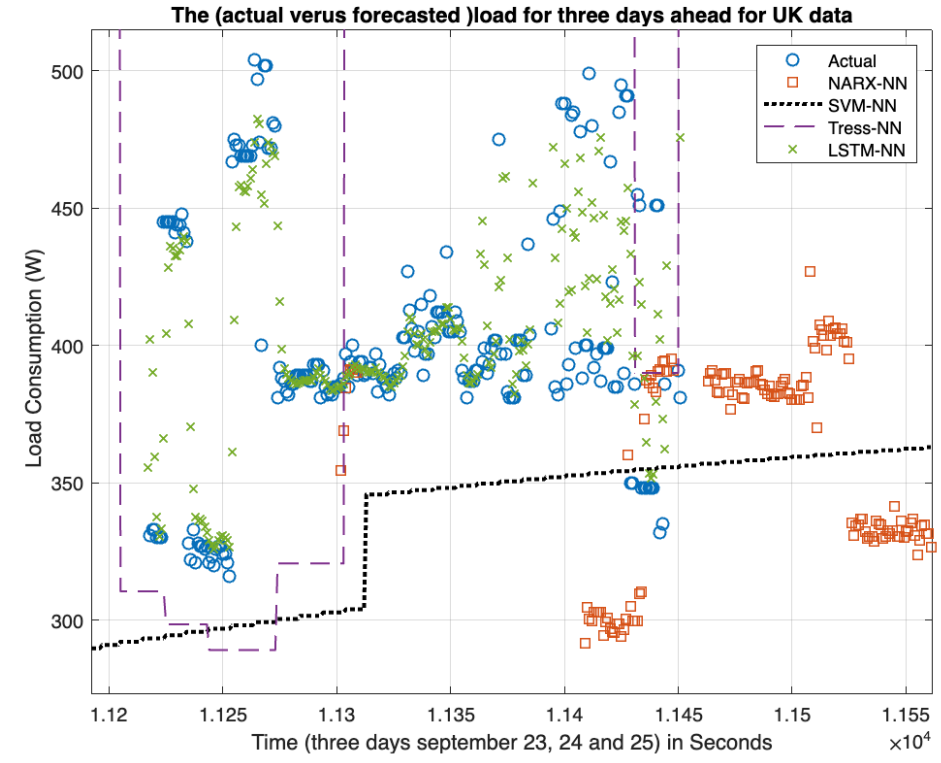
THE SAME RESULT GRAPHS BUT SHAPED INSTEAD OF COLOURED WERE WRITTEN IN THE CODE BELOW:

NARX,LSTM,SVM AND TREE NEURAL NETWORK PREDICTIONS (GRAPH)-CODE VERSUS ACTUAL DATA, SVN, TREE AND GAUSSIAN METHODS:

```
q = (yfit_tree1.'); % coarse tree training data
Z = (yfit_SVM1.'); %SVM using fine gaussian training data
g = (YTest_Final.'); % LSTM
x = (september23_24_25.'); %actual data
y = (K2.'); % NEURAL NETWORK PREDITED DATA
%Z = (data.');
%q = (data.');
ts1 = timeseries(x,1:42657); %No. of data
ts1.Name = ' Load Consumption (W)';
ts1.TimeInfo.Units = 'second';
% Set start date.
%ts1.TimeInfo.StartDate = '0:00';
```

```
% Set format for display on x-axis.
ts1.TimeInfo.Format = 'SS';
ts1.Time = ts1.Time - ts1.Time(1);
%ts1 = setuniformtime(ts1,'Interval',20)
ts2 = timeseries(y,1:42657);
ts2.Name = ' Load Consumption (W)';
ts2.TimeInfo.Units = 'second';
%$ts2.TimeInfo.StartDate = '23-09-2013 00:00:00';  %starts from 12am
0:00
ts2.TimeInfo.Format = 'SS';  %goes with seconds, minutes, hours
ts2.Time = ts2.Time - ts2.Time(1);


ts3 = timeseries(Z,1:42657);
ts3.Name = ' Load Consumption (W)';
ts3.TimeInfo.Units = 'second';
%ts3.TimeInfo.StartDate = '0:00';
ts3.TimeInfo.Format = 'SS';
ts3.Time = ts3.Time - ts3.Time(1);


ts4 = timeseries(q,1:42657);
ts4.Name = ' Load Consumption (W)';
ts4.TimeInfo.Units = 'second';
%ts4.TimeInfo.StartDate = '0:00';
ts4.TimeInfo.Format = 'SS';
ts4.Time = ts4.Time - ts4.Time(1);


ts5 = timeseries(g,1:42657);
ts5.Name = ' Load Consumption (W)';
ts5.TimeInfo.Units = 'second';
%ts5.TimeInfo.StartDate = '0:00';
ts5.TimeInfo.Format = 'SS';
ts5.Time = ts5.Time - ts5.Time(1);


%ploting all methods together
%ts5.TimeInfo.Format = 'ss,mm,hh';
plot(ts1,'o','LineWidth',1),
hold on
plot(ts2,'s','LineWidth',1),
hold on
plot(ts3,':k','LineWidth',2),
hold on
plot(ts4,'--','LineWidth',1),
hold on
plot(ts5,'x','LineWidth',1),
```

```
grid on


legend('show')
xlim('auto')
ylim('auto')
title(['The (actual verus forecasted )load for three days ahead for UK
data'])
legend('Actual','NARX-NN','SVM-NN', 'Tress-NN', 'LSTM-NN' )
xlabel('Time (three days september 23, 24 and 25) in Seconds')


zoom on


xlim([11192 11562])
ylim([273 515])
```



NARX NEURAL NETWORK PREDICTIONS ERRORS CODE AND RESULTS:

```
% Compare NARX predictions of the original load
Load_Data_NARX= september23_24_25;
PData_NARX = K2;
CompareData_NARX= [Load_Data_NARX,PData_NARX];
xlswrite('Compare_NARX.xlsx',CompareData_NARX);


Miu_NARX= mean(september23_24_25);
numPred_NARX = numel(september23_24_25);
MAPE1_NARX= 0.0;
MAE1_NARX = 0.0;
RMSE1_NARX = 0.0;
SD1_NARX= 0.0;
for k = 1:numPred_NARX
    PE(k) = abs(september23_24_25(k)-
PData_NARX(k))/september23_24_25(k);
    ME(k) = abs(september23_24_25(k)-PData_NARX(k));
    ZE(k) = (september23_24_25(k)-PData_NARX(k)).^2;
    SE(k) = (PData_NARX(k)-Miu_NARX).^2;
    MAPE1_NARX = PE(k)+MAPE1_NARX;
    MAE1_NARX = ME(k)+MAE1_NARX;
    RMSE1_NARX = ZE(k)+RMSE1_NARX;
    SD1_NARX = SE(k)+SD1_NARX;
end
MAPE_NARX = MAPE1_NARX/numPred_NARX;
MAE_NARX = MAE1_NARX/numPred_NARX;
RMSET_NARX= sqrt(RMSE1_NARX)/numPred_NARX;
SD_NARX = sqrt(SD1_NARX/numPred_NARX);
PEI_NARX = [MAPE_NARX ,MAE_NARX ,RMSET_NARX ,SD_NARX];
xlswrite('Prediction_Error_Index_NARX.xlsx',PEI_NARX);
fprintf('The errors in NARX of the original load');
disp(['NARX:    MAPE     MAE     RMSE     SD']);
disp(PEI_NARX);


% % Compare LSTM predictions of the original load
Load_Data_LSTM = september23_24_25;
PData_LSTM = YTest_Final;
CompareData_LSTM_under_pulseattack = [Load_Data_LSTM,PData_LSTM];
xlswrite('Compare_LSTM.xlsx',CompareData_LSTM_under_pulseattack);


Miu_LSTM = mean(september23_24_25);
numPred_LSTM = numel(september23_24_25);
MAPE1_LSTM= 0.0;
MAE1_LSTM= 0.0;
```

```
RMSE1_LSTM = 0.0;
SD1_LSTM = 0.0;
for k_LSTM = 1:numPred_LSTM
    PE_LSTM(k_LSTM) = abs(september23_24_25(k_LSTM)-
PData_LSTM(k_LSTM))/september23_24_25(k_LSTM);
    ME_LSTM(k_LSTM) = abs(september23_24_25(k_LSTM)-
PData_LSTM(k_LSTM));
    ZE_LSTM(k_LSTM) = (september23_24_25(k_LSTM)-
PData_LSTM(k_LSTM)).^2;
    SE_LSTM(k_LSTM) = (PData_LSTM(k_LSTM)-Miu_LSTM).^2;
    MAPE1_LSTM = PE_LSTM(k_LSTM)+MAPE1_LSTM;
    MAE1_LSTM = ME_LSTM(k_LSTM)+MAE1_LSTM;
    RMSE1_LSTM = ZE_LSTM(k_LSTM)+RMSE1_LSTM;
    SD1_LSTM = SE_LSTM(k_LSTM)+SD1_LSTM;
end
MAPE_LSTM = MAPE1_LSTM/numPred_LSTM;
MAE_LSTM = MAE1_LSTM/numPred_LSTM;
RMSET_LSTM = sqrt(RMSE1_LSTM)/numPred_LSTM;
SD_LSTM = sqrt(SD1_LSTM/numPred_LSTM);
PEI_LSTM = [MAPE_LSTM ,MAE_LSTM ,RMSET_LSTM ,SD_LSTM];
xlswrite('Prediction_Error_Index_LSTM.xlsx',PEI_LSTM);
fprintf('The errors in LSTM of the original load');
disp(['LSTM:    MAPE     MAE     RMSE     SD']);
disp(PEI_LSTM);



% % Compare SVM predictions of the original load
Load_Data_SVM = september23_24_25;
PData_SVM = yfit_SVM1;
CompareData_SVM = [Load_Data_SVM,PData_SVM];
xlswrite('Compare_SVM.xlsx',CompareData_SVM);


Miu_SVM= mean(september23_24_25);
numPred_SVM = numel(september23_24_25);
MAPE1_SVM= 0.0;
MAE1_SVM= 0.0;
RMSE1_SVM= 0.0;
SD1_SVM= 0.0;
for k_SVM = 1:numPred_SVM
    PE_SVM(k_SVM) = abs(september23_24_25(k_SVM)-
PData_SVM(k_SVM))/september23_24_25(k_SVM);
    ME_SVM(k_SVM) = abs(september23_24_25(k_SVM)-PData_SVM(k_SVM));
    ZE_SVM(k_SVM) = (september23_24_25(k_SVM)-PData_SVM(k_SVM)).^2;
    SE_SVM(k_SVM) = (PData_SVM(k_SVM)-Miu_SVM).^2;
    MAPE1_SVM = PE_SVM(k_SVM)+MAPE1_SVM;
```

```
    MAE1_SVM = ME_SVM(k_SVM)+MAE1_SVM;
    RMSE1_SVM = ZE_SVM(k_SVM)+RMSE1_SVM;
    SD1_SVM = SE_SVM(k_SVM)+SD1_SVM;
end
MAPE_SVM = MAPE1_SVM/numPred_SVM;
MAE_SVM = MAE1_SVM/numPred_SVM;
RMSET_SVM = sqrt(RMSE1_SVM)/numPred_SVM;
SD_SVM = sqrt(SD1_SVM/numPred_SVM);
PEI_SVM = [MAPE_SVM ,MAE_SVM ,RMSET_SVM ,SD_SVM];
xlswrite('Prediction_Error_Index_SVM.xlsx',PEI_SVM);
fprintf('The errors in SVM of the original load');
disp(['SVM:    MAPE     MAE     RMSE     SD']);
 disp(PEI_SVM);



% % Compare TREE predictions of the original load
Load_Data_TREE = september23_24_25;
PData_TREE= yfit_tree1;
CompareData_TREE = [Load_Data_TREE,PData_TREE];
xlswrite('Compare_TREE.xlsx',CompareData_TREE);


Miu_TREE= mean(september23_24_25);
numPred_TREE= numel(september23_24_25);
MAPE1_TREE= 0.0;
MAE1_TREE= 0.0;
RMSE1_TREE = 0.0;
SD1_TREE = 0.0;
for k_TREE= 1:numPred_TREE
    PE_TREE(k_TREE) = abs(september23_24_25(k_TREE)-
PData_TREE(k_TREE))/september23_24_25(k_TREE);
    ME_TREE(k_TREE) = abs(september23_24_25(k_TREE)-
PData_TREE(k_TREE));
    ZE_TREE(k_TREE) = (september23_24_25(k_TREE)-
PData_TREE(k_TREE)).^2;
    SE_TREE(k_TREE) = (PData_TREE(k_LSTM)-Miu_TREE).^2;
    MAPE1_TREE = PE_TREE(k_TREE)+MAPE1_TREE;
    MAE1_TREE = ME_TREE(k_TREE)+MAE1_TREE;
    RMSE1_TREE = ZE_TREE(k_TREE)+RMSE1_TREE;
    SD1_TREE = SE_TREE(k_TREE)+SD1_TREE;
end
MAPE_TREE = MAPE1_TREE/numPred_TREE;
MAE_TREE = MAE1_TREE/numPred_TREE;
RMSET_TREE = sqrt(RMSE1_TREE)/numPred_TREE;
SD_TREE = sqrt(SD1_TREE/numPred_TREE);
PEI_TREE = [MAPE_TREE ,MAE_TREE ,RMSET_TREE ,SD_TREE];
```

```
xlswrite('Prediction_Error_Index_TREE.xlsx',PEI_TREE);
fprintf('The errors in TREE of the original load');
disp(['TREE:    MAPE     MAE     RMSE     SD']);
disp(PEI_TREE);
```

MAE, RMSE, SD AND MAPE ERRORS IN NARX-NEURAL NETWORK RESULTS:

```
The errors in NARX of the original load

NARX:    MAPE     MAE     RMSE     SD

  Columns 1 through 2

    1.5519  420.2535

  Columns 3 through 4

    6.5947  925.8733
```

MAE, RMSE, SD AND MAPE ERRORS IN LSTM-NEURAL NETWORK RESULTS:

```
The errors in LSTM of the original load

LSTM:    MAPE     MAE     RMSE     SD

  Columns 1 through 2

    0.2820   52.3055

  Columns 3 through 4

    0.8261  940.3203
```

MAE, RMSE, SD AND MAPE ERRORS IN SVM-NEURAL NETWORK RESULTS:

```
The errors in SVM of the original load

SVM:    MAPE     MAE     RMSE     SD

  Columns 1 through 2

    0.3971  259.8025

  Columns 3 through 4

    5.1538  236.4332
```

MAE, RMSE, SD AND MAPE ERRORS IN TREE-NEURAL NETWORK RESULTS:

```
The errors in TREE of the original load

TREE:    MAPE     MAE     RMSE     SD

  Columns 1 through 2

    1.4312  388.3178

  Columns 3 through 4

    5.4412  294.8280
```

CYBER ATTACK TEMPLATES:

USING THE ORIGINAL DATA SEPTEMBER20_21_22 AND MODIFIED THEM USING CYBER TEMPLATES TWO CASES: CASE 1- USE EACH ATTACK ALONE, CASE2- USE ALL ATTACKS TOGETHER

1) PULSE ATTACK:

Load forecasts are modified to higher/lower values at a specific point during the entire duration of an attack. The attack parameter is set as λP.

$$\dot{p}_t^{\mathrm{F}} = (1 + \lambda_{\mathrm{P}}) \times p_t^{\mathrm{F}}, \quad \text{for} \quad t = t_{\mathrm{P}} \qquad (1)$$

where tP is the occurrence time of one pulse attack. pFt is the original load forecast that is not tampered with any cyber attack .p.Ft is the load forecast tampered with cyberattacks.

PULSE CODE:

```
%p ˙PFt=(1+λP)×load, fort=tP
lamda=0.5;
original_load=september20_21_22;
tt=[10,35,100,135,200,235,300,335,400,435,500,535,600,635,700,735,800,835,900,935,1000,1350,
2000,2350,3000,3350,4000,4350,5000,5350,6000,6350,7000,7350,8000,8350,9000,9350,10000,13500,
20000,23500,30000,33500,40000,42657];
Pulse=original_load;
Pulse(tt,1)=(1+lamda)∗original_load(tt,1);
modified_load_Pulse=Pulse;
```

Note: Pulse attack happen every at tt, data at 10 seconds and 35 seconds and so on.

2) SCALING ATTACK:

Scaling attacks involve modifying the values in a specified duration multiplied by a scaling attack parameter λS.

$$\dot{p}_t^{\mathrm{F}} = (1 + \lambda_{\mathrm{S}}) \times p_t^{\mathrm{F}}, \quad \text{for } t_s < t < t_e \qquad (2)$$

where ts and te represent the start- and end-time of one cyber attack, respectively.

SCALE CODE:

```
original_load=september20_21_22;
%scaling code
%p ˙Ft=(1+λS)×pFt, for ts<t<te
lamda_scale= 0.6;
t_scale= 2000:3000;
Scale=original_load;
Scale(t_scale,1)=(1+lamda_scale)*original_load(t_scale,1);
```

Note: Scale attack happen every at t_scale, data at 2000 seconds till 3000 seconds and so on.

3) RAMPING ATTACK:

There are two types of ramping attacks.

Type I ramping attack only considers up-ramping anomaly. The values in the specified range are multiplied by a ramping function λRt.

$$\dot{p}_t^{\mathrm{F}} = \lambda_{\mathrm{R}} \times (t - t_s) \times p_t^{\mathrm{F}}, \quad \text{for } t_s < t < t_e \qquad (3)$$

Type II ramping attack considers both up- and down- ramping anomalies. This attack is more challenging to detect for operators.

$$\dot{p}_t^{\mathrm{F}} = [1 + \lambda_{\mathrm{R}} \times (t - t_s)] \times p_t^{\mathrm{F}}, \quad \text{for } t_s < t < \lfloor \frac{t_s + t_e}{2} \rfloor \quad (4)$$

$$\dot{p}_t^{\mathrm{F}} = [1 + \lambda_{\mathrm{R}} \times (t_e - t)] \times p_t^{\mathrm{F}}, \quad \text{for } \lfloor \frac{t_s + t_e}{2} \rfloor < t < t_e \quad (5)$$

where $\lfloor \cdot \rfloor$ indicates the floored value which is used to present the approximate intermediate point between ts an te.

RAMPING CODE:

```
original_load=september20_21_22;
%Raming attack
%p ˙Ft =λR×(t-ts)×pFt, forts<t<te (3)
```

```
%type 1 up ramping
ramp1 = original_load;
%% Ramp Attack : Type-1
lamda_r = 0.6; % lambda_r for equation 3
t_ramp1 = 100:200;  % Time window for ramp attack
for t=t_ramp1
    temp(t,1) = lamda_r*(t-t_ramp1(1))*original_load(t,1);
end
ramp1(t_ramp1,1) = ramp1(t_ramp1,1) + temp(t_ramp1,1);
clear temp;
%% Ramp Attack: Type-2
lamda_r21 = 0.2; % For equation 4
lamda_r22 = 0.25; % For equation 5
t_ramp21 = 700:800;
t_ramp22 = 801:900;


ramp2 = ramp1;
for t = t_ramp21    % Adding up ramp anomalities in previous forecasts
    temp(t,1) = (1+lamda_r21*(t-t_ramp21(1)))*ramp1(t,1);
end
ramp2(t_ramp21,1) = temp(t_ramp21,1)+ramp1(t_ramp21,1);
for t = t_ramp22    % Adding down ramp anomalities in previous forecasts
    temp(t,1) = (1+lamda_r22*(t_ramp22(length(t_ramp22))-t))*ramp1(t,1);
end
ramp2(t_ramp22,1) = temp(t_ramp22,1)+ramp1(t_ramp22,1);
ramping=ramp2;
```

Note: Ramp attack happen every t_ramp, data at 100 till 200 seconds and till 700 till 800 seconds and so on.

4) RANDOM ATTACK:

This attack involves the addition of positive values returned by a uniform random function to load forecasts.

$$\dot{p}_t^{\mathrm{F}} = p_t^{\mathrm{F}} + \lambda_{\mathrm{RA}} \times \mathbf{rand}\,(t)\,, \quad \text{for } t_s < t < t_e \qquad (6)$$

where rand is a uniformly distributed random number gener- ator that can be achieved by a built-in function in MATLAB. λRA is a scale factor and defined as half of the maximum of load forecast value, i.e., λRA = max □pFt □ /2. The start- and end-time of one random attack is assumed to be randomly set by attackers.

RANDOM CODE:

```
original_load=september20_21_22;
```

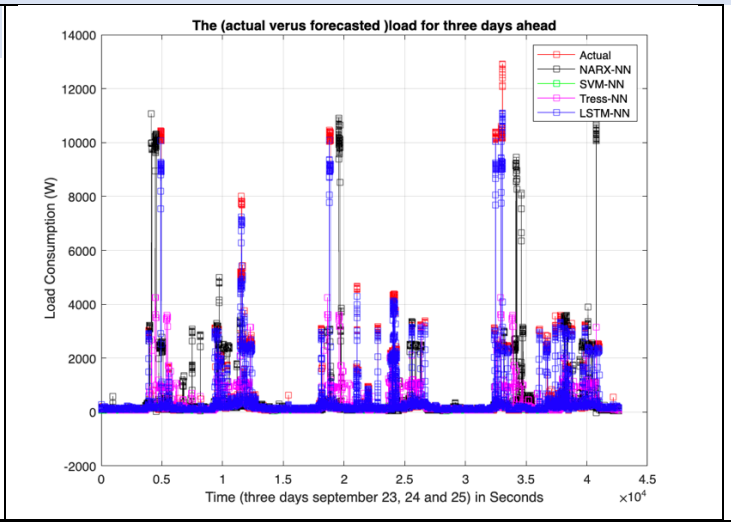
```
%random code
%p ˙Ft=pFt+λRA×rand(t), forts<t<te
t_randam= 10000:40000;
randam_scale= rand(length(t_randam),1);
lamda_randam=max(original_load)/2;
randam=original_load; %replicated the data
randam(t_randam,1)= randam(t_randam)+lamda_randam*randam_scale;
```

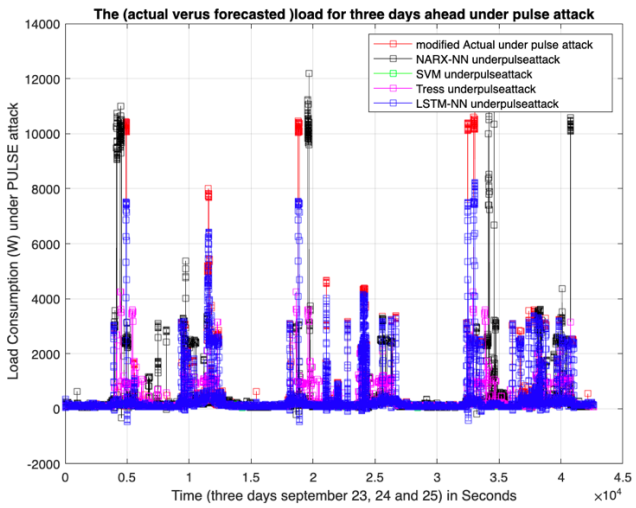
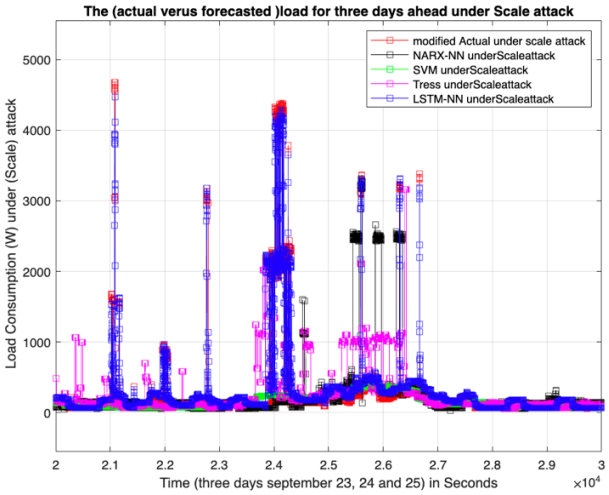
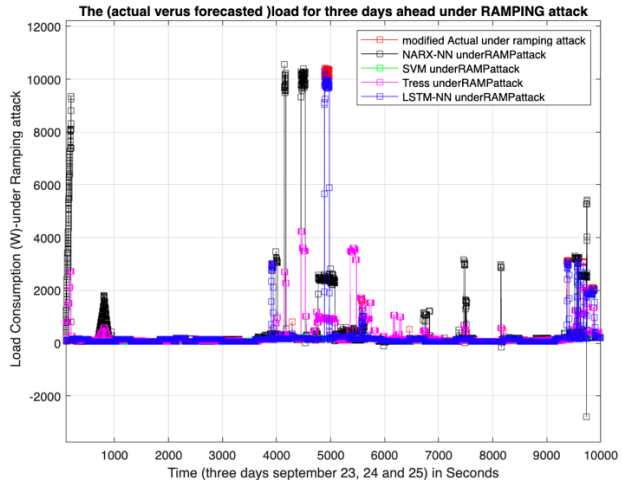
Note: Random attack happen every t_random, data at 10000 till 40000 seconds.

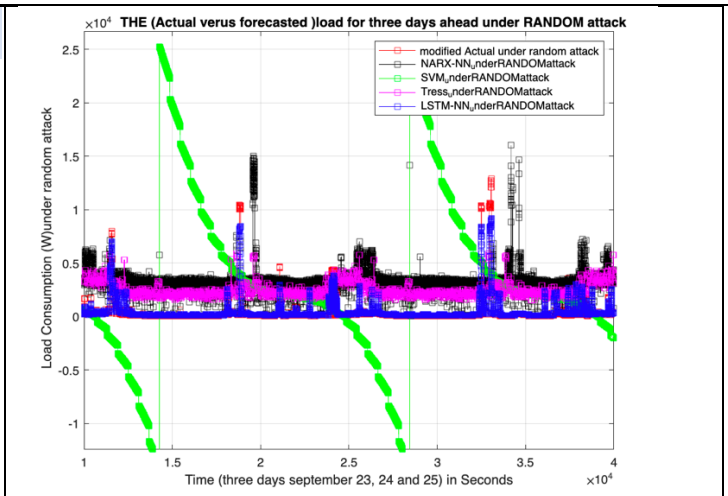
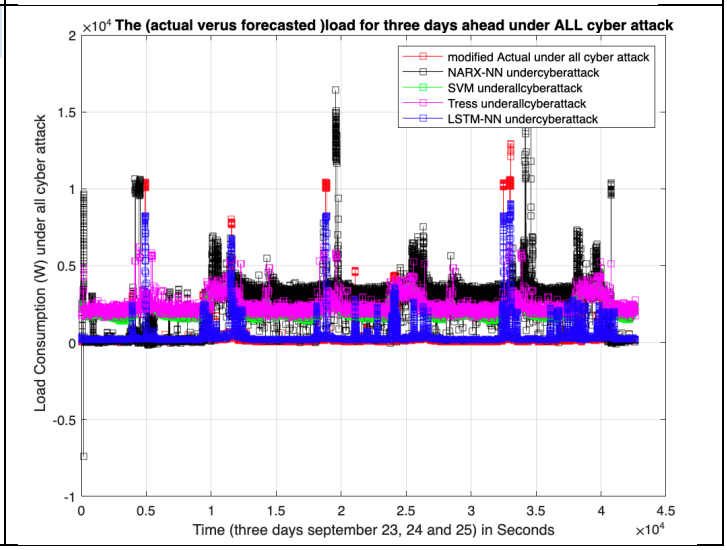
THE ORIGINAL DATA SEPTEMBER20_21_22 HAVE BEEN MODIFIED UNDER THESE ABOVE CYBER ATTACKS AND HAVE BEEN PREDICTED USING NARX-NN, LSTM-NN, SVM-NN AND TREE-NN USING THE SAME CODE ABOVE BUT WITH THE MODIFIED DATA.

(GRAPHS) WERE THE ATTACK IS RESULTS:

ACTUAL LOAD



PULSE ATTACK		
SCALE ATTACK		
RAMP ATTACK		

RANDOM ATTACK	
ALL CYBER ATTACK	

CASE2 :

DATA WERE MODIFIED USING ALL FOUR TYPES OF CYBER ATTACK TEMPLETES TOGETHER (PULSE,SCALE,RAMPING AND RANDOM)

CODE USED TO MODIFIED THE DATA UNDER ALL CYBER ATTACK TEMPLATES :

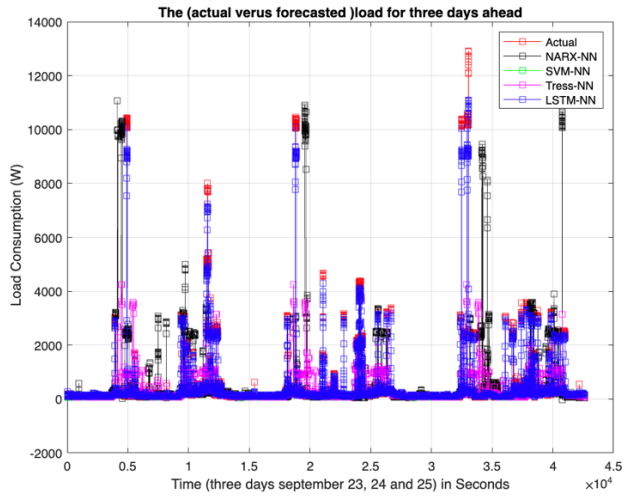
```
%p ˙PFt=(1+λP)×load, fort=tP
lamda=0.5;
original_load=september20_21_22;
tt=[10,35,100,135,200,235,300,335,400,435,500,535,600,635,700,735,800,835,900,935,1000,1350,
2000,2350,3000,3350,4000,4350,5000,5350,6000,6350,7000,7350,8000,8350,9000,9350,10000,13500,
20000,23500,30000,33500,40000,42657];
Pulse=original_load;
Pulse(tt,1)=(1+lamda)*original_load(tt,1);
modified_load=Pulse;
```

```
%OR OTHER CODE FOR PULSE:
% scale_pulse = rand(1000,1); % Your load forecast which having pulse and scaling attack
%
% % Pulse Attack
% lamda_pulse = 0.65;
% t = randi([100,300],10,1);     % Time Instances, when pulse attack will be created
% % In above line-5
% %   [100,300] - Shows the time range, within which pulse attack need to be
% %   created.
% %   10 - Number to times pulse attack need to be created within the
% %   [100,300] time range.
% %   1 - this is to create a single coloumn.
%
% scale_pulse(t) = (1+lamda_pulse)*scale_pulse(t);

%scaling attack code
%p ˙Ft=(1+λS)×pFt, for ts<t<te
lamda_scale= 0.6;
t_scale= 2000:3000;
Scale=modified_load;
Scale(t_scale,1)=(1+lamda_scale)*modified_load(t_scale,1);


%random code
%p ˙Ft=pFt+λRA×rand(t), forts<t<te
t_randam= 10000:40000;
randam_scale= rand(length(t_randam),1);
lamda_randam=max(original_load)/2;
randam=Scale; %replicated the data
randam(t_randam,1)= randam(t_randam)+lamda_randam*randam_scale;


%cyber_attack=randam;


%Raming attack
%p ˙Ft =λR×(t-ts)×pFt, forts<t<te (3)
%type 1 up ramping
t_ramp1= 100:200;
lamda_ramp= 0.6;
ramp1=randam;
for t= t_ramp1
    ramp_temp(t,1)= lamda_ramp*(t-t_ramp1(1))*ramp1(t,1);
end

ramp1(t_ramp1,1)=ramp1(t_ramp1,1)+ramp_temp(t_ramp1,1);
clear ramp_temp;
% %% Ramp Attack: Type-2
%type 2 down ramping
lamda_r21 = 0.2; % For equation 4
```

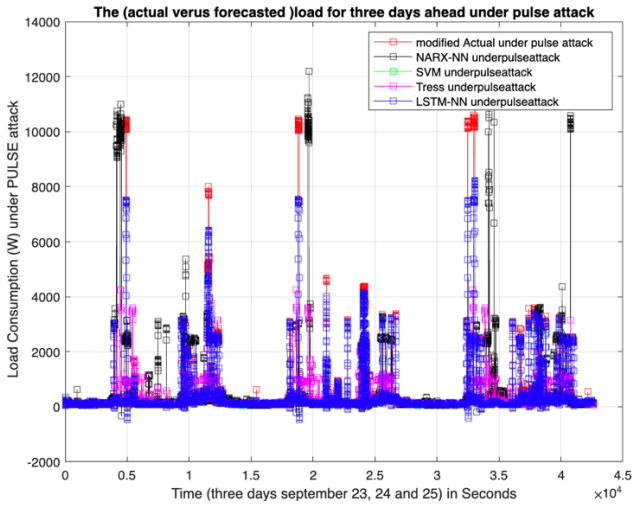
```
lamda_r22 = 0.25; % For equation 5
t_ramp21 = 700:800;
t_ramp22 = 801:900;


ramp2=ramp1;
for t= t_ramp2  % Adding up ramp anomalities in previous forecasts
    ramp_temp2(t,1)= (1+lamda_r21*(t-t_ramp21(1)))*ramp1(t);
end
for t = t_ramp22    % Adding down ramp anomalities in previous forecasts
   ramp_temp(t,1) = (1+lamda_r22*(t_ramp22(length(t_ramp22))-t))*ramp1(t,1);
end
ramp2(t_ramp22,1) = ramp_temp(t_ramp22,1)+ramp1(t_ramp22,1);
cyber_attack=ramp2;
```

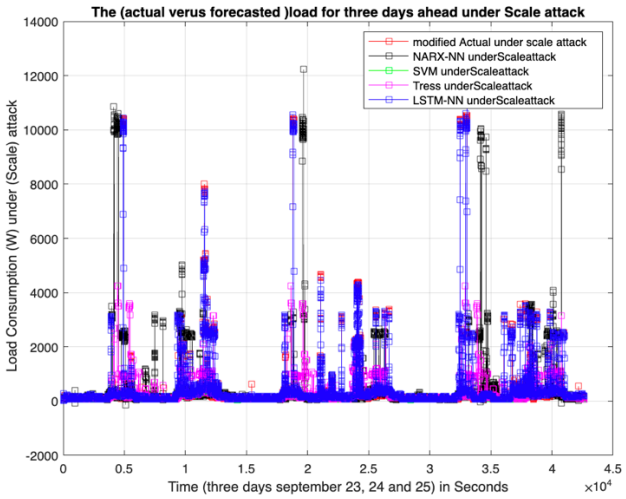
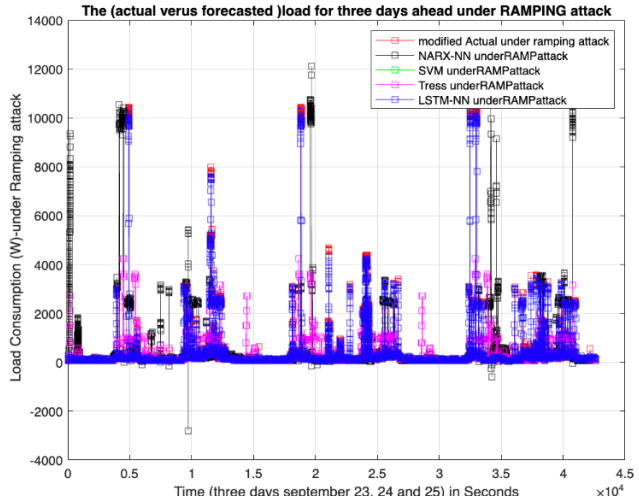
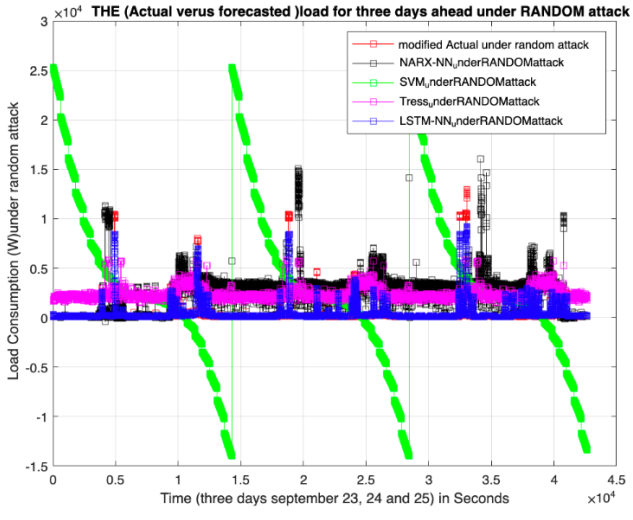
(GRAPH) RESULTS:

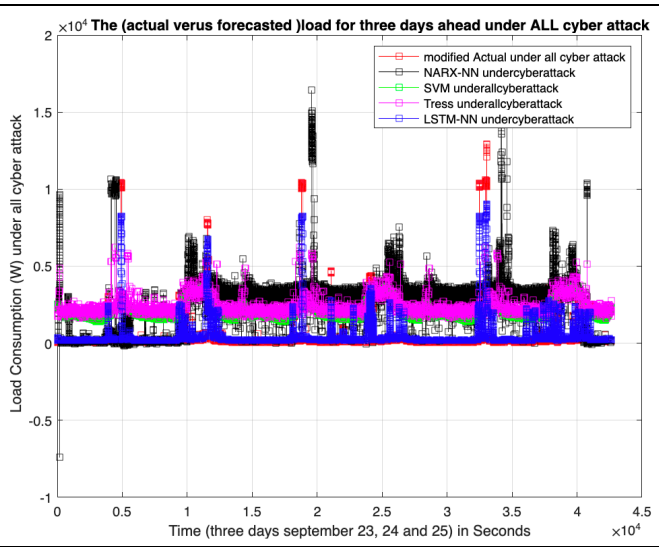
ACTUAL LOAD



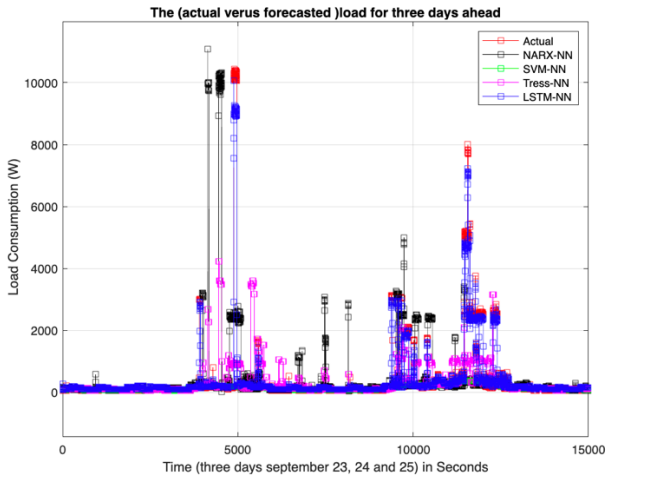
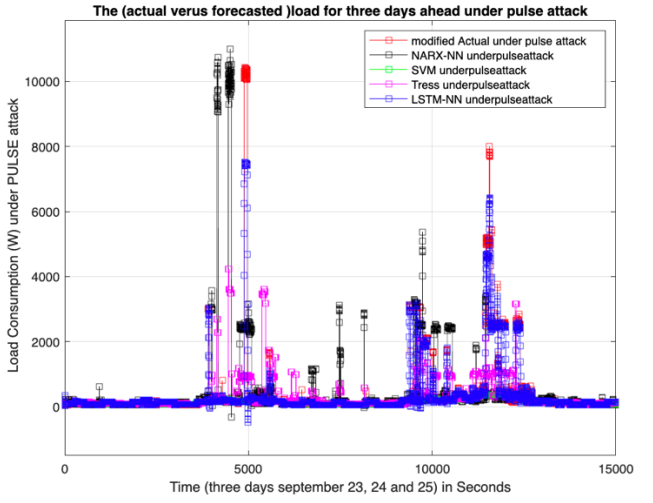
PULSE ATTACK

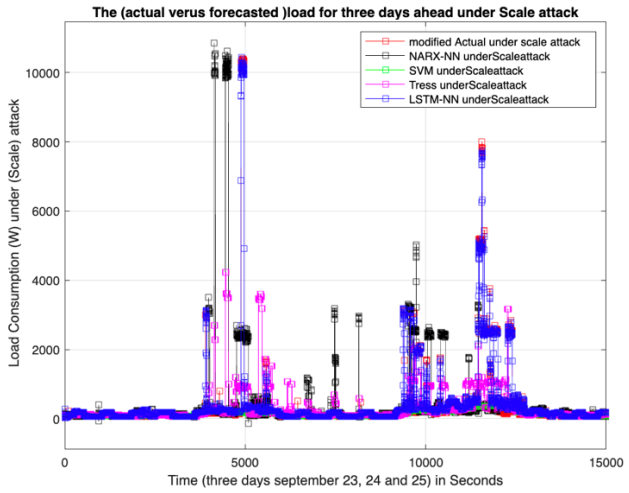
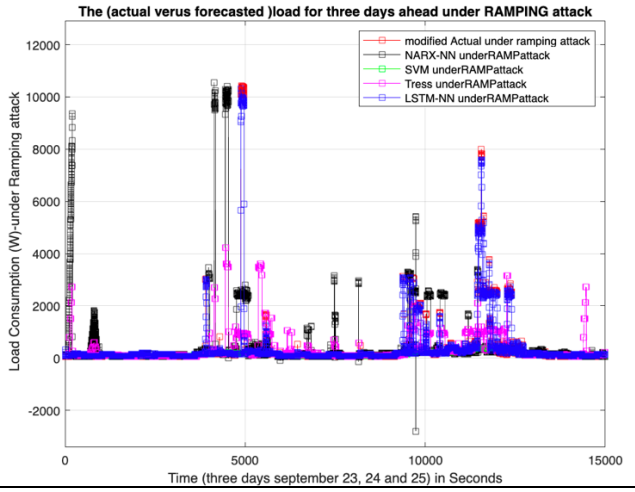
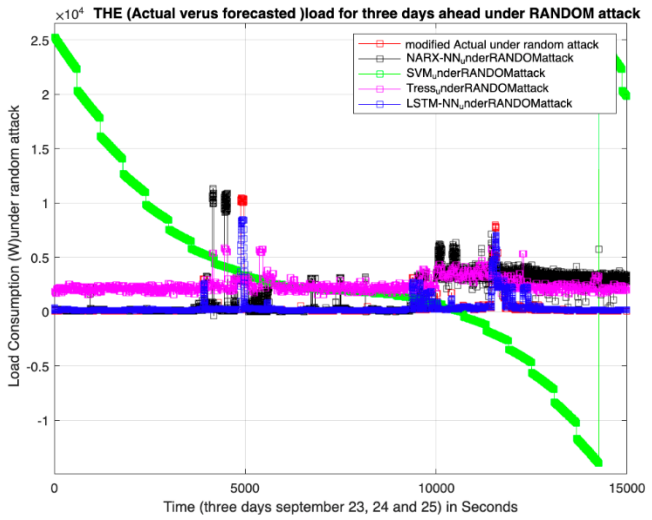


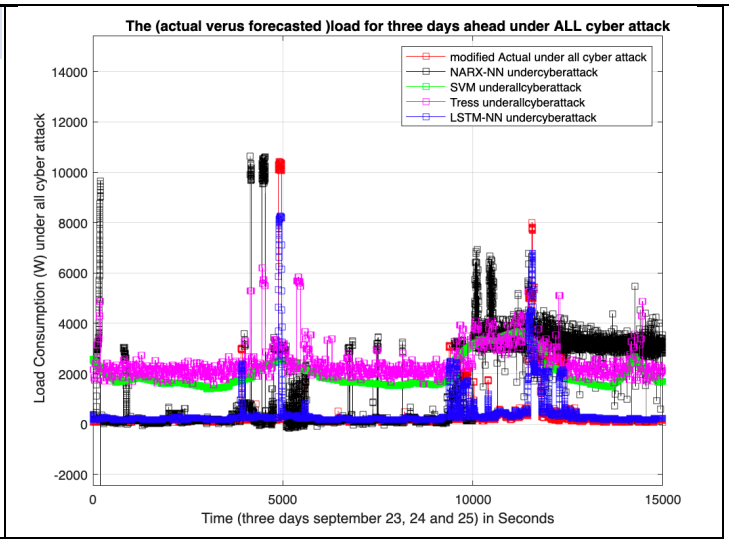
SCALE ATTACK		
RAMP ATTACK		
RANDOM ATTACK		

ALL CYBER ATTACK	

(GRAPH) RESULTS ZOOMED IN FROM 0 TO 15000 SEC:

ACTUAL LOAD	
PULSE ATTACK	

SCALE ATTACK		
RAMP ATTACK		
RANDOM ATTACK		

ALL CYBER ATTACK	

MAE, RMSE, SD AND MAPE ERRORS UNDER ALL CYBER ATTACK RESULTS:

- UNDER PULSE ATTACK ERRORS

The errors in NARX under pulse attack

NARX: MAPE MAE RMSE SD

Column 1

1.5268

Column 2

418.8209

Column 3

6.5978

Column 4

926.5126

The errors in LSTM under pulse

LSTM: MAPE MAE RMSE SD

Column 1

0.1061

Column 2

58.2567

Column 3

1.7030

Column 4

843.2734

The errors in SVM under pulse attack

SVM: MAPE MAE RMSE SD

Column 1

0.4000

Column 2

259.8690

Column 3

5.1557

Column 4

236.0923

The errors in TREE under pulse attack

TREE: MAPE MAE RMSE SD

Column 1

1.4319

Column 2

388.3964

Column 3

5.4412

Column 4

290.2558

- UNDER SCALE ATTACK ERRORS

The errors in NARX under Scale attack

NARX:	MAPE	MAE	RMSE	SD
	1.5369	419.2005	6.5999	926.3797

The errors in LSTM under Scale attack

LSTM:	MAPE	MAE	RMSE	SD
	1.0e+03 *			
	0.0003	0.0568	0.0007	1.0290

The errors in SVM under scale attack

SVM:	MAPE	MAE	RMSE	SD
	0.4069	260.0093	5.1527	234.4309

The errors in TREE under Scale attack

TREE:	MAPE	MAE	RMSE	SD
	1.4374	388.3488	5.4412	294.8280

- UNDER RAMP ATTACK ERRORS

The errors in NARX under Ramp attack

NARX:	MAPE	MAE	RMSE	SD
Column 1				
1.7043				
Column 2				
433.8518				
Column 3				
6.6881				
Column 4				

The errors in LSTM under ramp attack

LSTM:	MAPE	MAE	RMSE	SD
1.0e+03 *				
Column 1				
0.0001				
Column 2				
0.0333				
Column 3				
0.0008				
Column 4				

The errors in SVM under ramp attack

SVM:	MAPE	MAE	RMSE	SD
Column 1				
0.4040				
Column 2				
259.9630				
Column 3				
5.1534				
Column 4				
235.2796				

The errors in TREE under ramp attack

TREE:	MAPE	MAE	RMSE	SD
Column 1				
1.6230				
Column 2				
401.8941				
Column 3				
5.4864				
Column 4				
294.8280				

- UNDER RANDOM ATTACK ERRORS

The errors in NARX under Random attack

```
NARX:    MAPE      MAE      RMSE      SD

   1.0e+03 *

  Column 1

    0.0192

  Column 2

    2.2911

  Column 3

    0.0135

  Column 4

    2.6341
```

The errors in LSTM under random attack

```
LSTM:    MAPE      MAE      RMSE      SD

  Column 1

    0.4855

  Column 2

  108.3814

  Column 3

    1.6823

  Column 4

  853.3391
```

The errors in SVM under random attack

```
SVM:    MAPE      MAE      RMSE      SD

   1.0e+03 *

  Column 1

    0.0660

  Column 2

    6.1968

  Column 3

    0.0422

  Column 4

    8.5745
```

The errors in TREE under random attack

```
TREE:    MAPE      MAE      RMSE      SD

   1.0e+03 *

  Column 1

    0.0193

  Column 2

    2.2444

  Column 3

    0.0117

  Column 4

    1.6455
```

- UNDER ALL CYBER ATTACKS ERRORS

```
The errors in NARX under ALL attack

NARX:    MAPE     MAE     RMSE     SD

   1.0e+03 *

    0.0194    2.2957    0.0136    2.6369

The errors in LSTM under all attack

LSTM:    MAPE     MAE     RMSE     SD

    0.9692  161.7596    1.7672  808.2965

The errors in SVM under all attack

SVM:    MAPE     MAE     RMSE     SD

   1.0e+03 *

    0.0154    1.8528    0.0098    1.7955


The errors in TREE under all attack

TREE:    MAPE     MAE     RMSE     SD

   1.0e+03 *

    0.0195    2.2588    0.0118    2.3801
```

The above code was done using MATLAB, to replicate the original data and modify it to new data under all cyber-attack types.