Dynamic Load Modeling from PSSE-Simulated Disturbance Data using Machine Learning

Sanij Gyawali

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Virgilio A. Centeno, Chair
Jaime De La Reelopez
Vassilis Kekatos

September 24, 2020
Blacksburg, Virginia

Keywords: Dynamic Load Modeling, Neural Network, Long-Short Term Memory, Support Vector Regression, Phasor Measurement Units

Dynamic Load Modeling from PSSE-Simulated Disturbance Data using Machine Learning

Sanij Gyawali

ABSTRACT

There have been numerous cases recorded where a power system didn't react to disturbances as expected. This can lead to system instability and power outages with a huge loss of revenue as a consequence. To prevent this, power system operators and planners should certify the load models used during planning are precise enough to replicate the behaviour during system disturbances. Load models have evolved from simple ZIP model to composite model that incorporates the transient dynamics of motor loads. This research is to utilize the latest trend on Machine Learning and build reliable and accurate composite load model. A composite load model is a combination of static (ZIP) model paralleled with a dynamic model. A dynamic model, recommended by Western Electricity Coordinating Council (WECC), can be an induction motor representation. This research uses a dual cage induction motor with 20 parameters pertaining to its dynamic behaviour, starting behaviour, and per unit calculations. When it comes to learning with algorithms, a huge amount of data is expected. Gathering thousands of real-life disturbance data recorded by Phasor Measurement Units(PMU) at a particular bus is not feasible. The common use of PMUs hasn't been that long and getting access to the real data is a long process involving much scrutiny because of cybersecurity issues. The next best way was to use a simulating environment like PSSE. IEEE 118 bus system was taken as a test setup in PSSE and dynamic simulations were run to generate thousands of data samples. Each of the samples contained data on Bus Voltage, Bus Current, and Bus Frequency with corresponding induction motor parameters as target variables. It is found that Artificial Neural Network(ANN) with multivariate input to single parameter output approach worked best. Recurrent Neural Network(RNN) is also experimented side by side to see if an additional set of information of timestamps would help the model prediction. Moreover, a different definition of a dynamic model with transfer function based load is also studied. Here, the dynamic model is defined as a mathematical representation of the relation between bus voltage, bus frequency, and active/reactive power flowing in the bus. With this form of load representation, Long-Short Term Memory(LSTM), a variation of RNN, performed better than the concurrent algorithms like Support Vector Regression(SVR). The result of this study is a load model (basically parameters defining the load) at load bus whose predictions are compared against simulated parameters to examine the validity. After validation, they can be used in contingency analysis.

Dynamic Load Modeling from PSSE-Simulated Disturbance Data using Machine Learning

Sanij Gyawali

GENERAL AUDIENCE ABSTRACT

Independent system Operators(ISO) and Distribution system operators(DSO) have a responsibility to provide uninterrupted power supply to consumers. Keeping that in mind along with the longing to keep operating cost minimum, engineers and planners study the system beforehand and try to find the optimum strength for each of the power system elements like generators, transformers, transmission lines, etc. Then they test the overall system using power system models to verify the stability and strength of the system. However, the verification is only as good as the system models that are used. As most of the power systems components are controlled by the operators themselves, it's easy to develop a model from their perspective. The load is the only component subjected to consumers. Hence, the necessity of better load models. Several studies have been made on static load modelling and the performance is on par with real behaviour. But dynamic loading, which is a load behaviour dependent on time, is rather difficult to model. Some attempts on dynamic load modelling can be found already. Physical component-based and mathematical transfer function based dynamic models are quite widely used for the study. These forms are largely accepted as a good representation of dynamic behaviours. Then comes the task of estimating the parameters of these mathematical models. In this research, we tested out some new machine learning methods to accurately estimate the parameters. Thousands of simulated data is used to train machine learning models. After training, we validated again on some other unseen data. This study goes on to recommend better methods to load modelling.

Dedication

To my mom and dad.

Acknowledgments

I would like to express my sincere gratitude to my academic advisor and committee chair Dr Virgilio A. Centeno, for his motivation and support during my work under his guidance. He always encouraged me to believe in the path I choose and work to understand it better. I would also like to thank Dr Jaime De La Reelopez for his encouragement and belief in me. Dr Kekatos has been a constant source of encouragement and support for the past two years. He always reminded me to keep trying my best no matter what the situation. I admire them both for creating a nurturing and welcoming atmosphere at the Power Systems Laboratory.

Apart from my committee members, I want to acknowledge my interim advisor Dr Chen-Ching Liu whose guidance helped to set my graduate study path and Dr Bert Huang whose course on Advanced Machine Learning helped me with my research direction. Dr Bijaya Adhikari, assistant professor at the Computer Science department of the University of Iowa, who has been my mentor throughout this work, deserves special thanks for putting up with my endless questions. I want to express my appreciation towards all my fellow graduate students in the Power and Energy Center, whom I have had the opportunity to know in the past two years - Sangeetha, Vivek, Rounak, Manish, Tapas, Carlos, Rosie, Sherin, Nitasha, Yousef, Jenny, Genesis, Alok, Mana and Sina. I will cherish the memories we have shared.

A special mention to my friends from back home who were also here at Virginia Tech, Pratigya, Sagar, Durga, and every single one from Nepal House, you guys always made me feel at home. No amount of gratitude will ever be enough to thank my parents who have made many sacrifices while keeping my education above everything else. Also, I would like to thank everyone in my family for always believing in me and supporting me in everything I do. At last, I want to thank all friends and family here in Blacksburg who made these two years an enjoyable time of my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Today's power system is an intricate network of generations, transmission/distribution lines, and consumer loads. With a complex system, the possibility of system instability (failure) is inevitable. Power system stability is the capacity of machines to come back to the original stable state or move to a new stable state of the operating point preserving synchronism [1]. It may happen due to over the limit burden in the system or sometimes due to natural calamities. For this reason, system reinforcement is taken seriously and is done mainly by preparing against contingencies [2]. N-1 is a popular concept of preparing the system for the loss of any one of the crucial component. This is all done first with a mathematical model of all components of the power system in a simulation environment, which gives rise to the necessity of accuracy and updated component models. Among the three crude divisions of power system mentioned earlier, only the consumer loads components are the ones not in the hand of planners. The general idea is to use a predictive system modelling the consumer load [3][4][5]. This gives rise to the necessity of updated and accurate load models.

Recent failure to address this issue is the major blackout of September 8, 2011, San Diego [6]. During the afternoon of September 8^{th} of 2011, a total of eleven minutes of power outage occurred in Pacific Southwest, causing cascading outages and a total blackout to more than two and a half million customers. State of Arizona, southern California, and part of Mexico suffered a power outage that day. Only in San Diego, about one and half million residents were without power for almost twelve hours. After a study of the events, it was concluded that a single loss of 500 kV transmission line was the starting event which further cascaded into the failure of multiple components [7]. The shocking part of the whole event was that the system was designed to withstand the loss of one component; even if the component was as big as 500 kV line. Figure 1.1 shows the transmission network in the California-Arizona region. Location of first loss on the 500 kV line is highlighted by 'X'. This is a lesson on how crucial it is to have an updated and accurate load model.

The consumer load changes with the time of the day which is prominent in lower voltage levels but in transmission voltage levels, load dependency can be seen with the time of the year(seasons), and locations. This gives rise to the need for dynamic and fast updating load models. With the rise of synchrophasors, measurement-based data-driven load modelling is up for grab which amazingly enough fulfils the need for fast updating and adaptable system

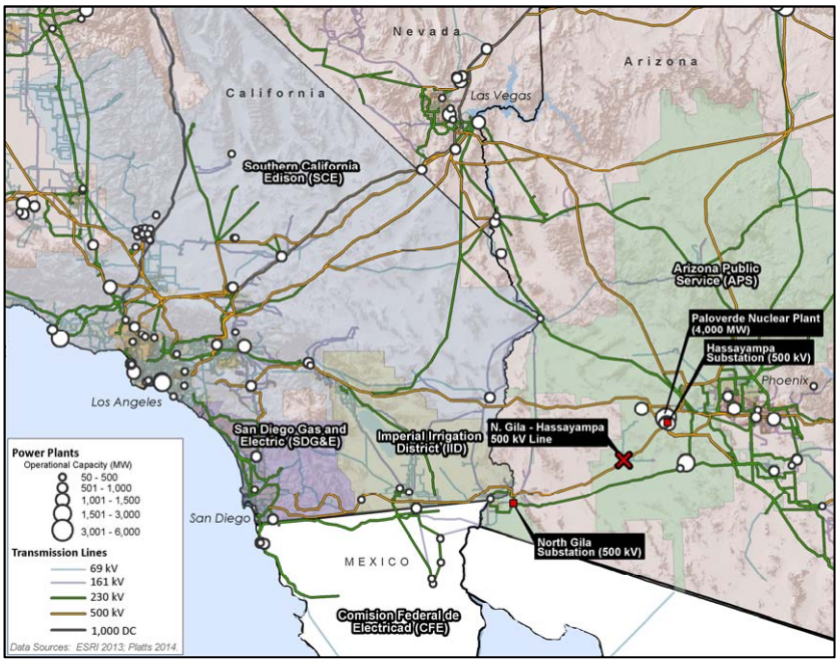


Figure 1.1: Transmission network in Southern California and Arizona

component models. In this thesis, various machine learning methods have experimented and the better ones are suggested.

1.2 Kinds of Loads

Electrical loads are a combination of several components such as lighting, thermal load, air conditioning, motor loads, etc. In the study of any electrical system, these loads are to be aggregated and represented by highly simplified models like in [8]. Mainly electrical load components are divided into two kinds: static and dynamic. Static loads are the functions of voltage and frequency at any given instant, whereas the dynamic loads are time-dependent loads with transient characteristics. There are two main types of static load models, the ZIP load and the frequency load. However, for dynamic loads, there are many models with different types and compositions [9]. A few of these will be discussed in next chapter.

1.2.1 Static Loads

In a power system, static loads are continuous loads which do not change notably over a brief period of time intervals. Lighting loads like street lights which run overnight draws essentially the same power all the time. Hence, static loads do not need differential equations

to represent them.

1.2.2 Dynamic Loads

Dynamic loads are the ones that can alter rapidly, like induction motors, which can draw very large power during starting or during a fault in the system. Power systems need to be sized to provide for dynamic loads as some of the generation capacity will have to expend on reactive power and thus reducing the total active power for static loads. A differential-based equation is required for representing dynamic loads.

1.3 Power systems dynamic Simulation

The dynamic simulation of power system network details the electro-mechanical response of a system subjected to disturbances like electrical faults, loss of system components (generation, lines, or loads). Simulation studies include a study of the transients that are responsible for system stability [1]. Predicting the response of the system during a disturbance caused internally or externally affecting the electro-mechanical balance of the system is the mission point of dynamic simulation. Planning and operations studies depend heavily on the dynamic simulation, both off-line and real-time, for the better preparation of contingencies [10] and also for system reinforcement and better control design. Offline simulations are software-based which solve the non-linear function describing the power system altogether. Whereas real-time are run in synchronism with the real system as in real-time data from measuring instruments are utilized in making decisions.

The power system network model and dynamic information are integrated into the simulation software such as PSSE from Siemens [11]. The network model includes information on buses (total numbers and types), line and shunt impedances, the power generated and load demands. While the dynamic information can be about transient characteristics of generators, exciters, governors and loads. Simple power flow on the network can give us the system state at an equilibrium state. This is a static simulation which doesn't predict the behaviour between equilibriums. To capture the transients and sub transient in between, dynamic simulation is preferred.

In an ideal world, the predicted dynamic response would exactly match the real system response under some change. But, this is usually not the case. The increasing complexity of today's power network introduces new difficulties with analyzing the system as a whole. In addition, dynamic models are usually the simplified, or approximate form of the true self. Thus, results from the simulation, more often than not, cannot exactly trace the actual behaviour. This gives rise to the necessity of updated methodologies and fast methods.

1.4 Load modeling approaches

The two methods of formulating a load model are the component-based approach and the measurement-based approach [12][13]. In recent years, there has been an increasing amount of studies based on the measurement-based approach. One reason for this change is the implementation of the PMU, which has brought in measurements at high data rates [14]. Another reason is the substantial improvements in mathematically based methodologies, such as optimization and data mining.

1.4.1 Component-based approach

The component-based method has been more popular due to its simplicity and reliability in the past. It is usually called as a bottom-up approach, involving aggregating qualitative data and combining it with statistical approaches to extrapolate the base data into determining the type of load component and property for a bus level. Figure 1.2 shows this approach.

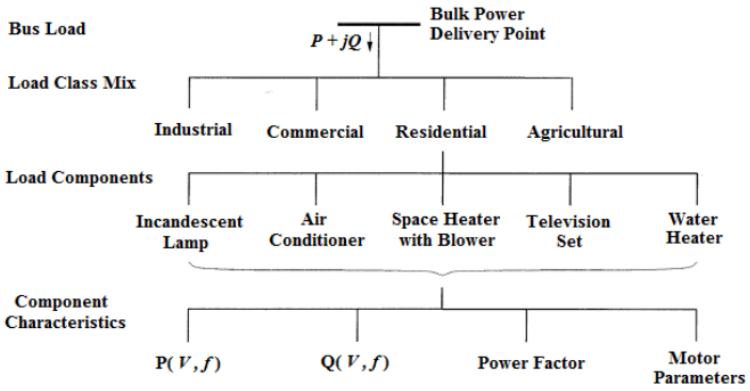


Figure 1.2: Component-based approach

1.4.2 Measurement-Based Approach

The measurement-based method or otherwise known as the top-down approach uses measurements collected over a period of time to determine the types of load components based on their attributed dynamic properties [15]. The measurements are used to study the steady-state load voltage characteristics, the steady-state load frequency characteristics and the dynamic load voltage characteristics, separately, depending on the type of load being modelled. The measured data can be used to fit an assumed expression designed for every load. Figure 1.3 shows this approach.

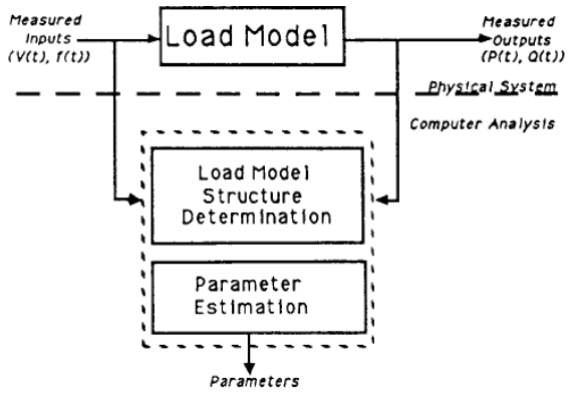


Figure 1.3: Measurement-based approach

1.5 Summary of Contributions

This thesis presents:

- Two methodologies for dynamic load modeling based on measurement data:
  1. Physical Induction motor-based model
     - Parameter Identification using Machine Learning (ML) algorithms: Artificial Neural Network(ANN) along with its variation and Long-Short Term Memory(LSTM)
  2. Differential equation-based model
     - Equation fitting using ML algorithms: Support Vector Machine Regression(SVR) and LSTM
- Both methodologies are validated using disturbance data from simulation

1.6 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 presents a detailed literature review on static and dynamic load models, contemporary parameter estimation methods and their advancements. Chapter 3 presents the induction motor based dynamic load representation. Various Neural network-based models are explained including ANN and LSTM. A scenario of the system with faulty parameters is presented and parameters are updated using a learned model which goes on to validate the methodology. Chapter 4 goes on to details on differential equation-based dynamic load. Here, SVM and LSTM are implemented and compared. Chapter 5 provides a conclusion on this thesis and presents new directions for future research.

Chapter 2

Review of Literature

This chapter presents an outline of existing research into load modelling. An overview of the literature on dynamic load representation and methods used for parameter estimation is also presented. The more traditional way of load representation is the simple static ZIP load. The present context where 50-70% of the load is motor load, this way of representing is not suitable. Figure 2.1 shows the systems with different static to dynamic load composition reacting to disturbance. First is the system without dynamic characteristics, second is the 50-50 system and third is the all dynamic load system. It is visibly clear that the traditional way of load representation is obsolete.

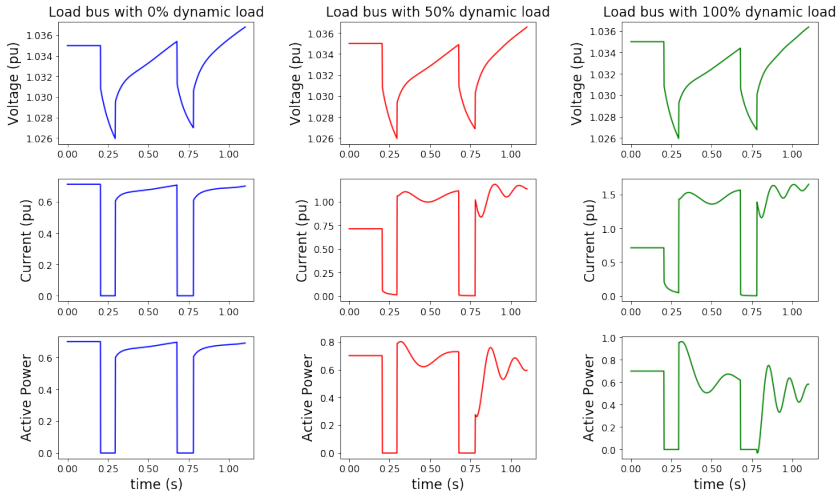


Figure 2.1: System reaction to 3 phase fault at nearby lines

2.1 Static Load Model

As defined in Chapter 1, the static load model should be representing a load component with a fixed demand. Similarly, during the steady-state study, which is study at equilibrium, all the system load can be represented as a static load. Figure 2.2 represents the basic static load model called ZIP model [16] where Z (constant impedance), I (constant current), and P (constant power) are connected in parallel. So, their composition must equal to 100 %. There

is also another way of representing static load with the addition of a frequency component to the ZIP model. In this work, our main focus is dynamic load modelling and so we will be working with just constant power part of the ZIP model.

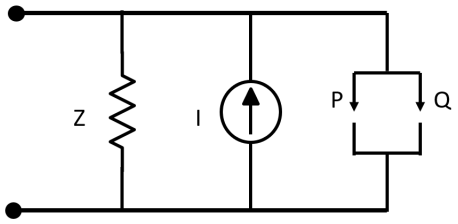


Figure 2.2: ZIP load model

Equations 2.1- 2.4 represents voltage dependent polynomial model for the static ZIP load. P_0 and Q_0 are the active and reactive power at initial point and |V| is the voltage magnitude at the load bus. Small p_xs and q_xs are the compositions respectively.

$$P = P_0[p_1|V|^2 + p_2|V| + p_3] \tag{2.1}$$
$$Q = Q_0[q_1|V|^2 + q_2|V| + q3] \tag{2.2}$$
$$p_1 + p_2 + p_3 = 1 \tag{2.3}$$
$$q_1 + q_2 + q_3 = 1 \tag{2.4}$$

2.2 Dynamic Load Model

Unlike static load modelling method, there are many types of dynamic loading representation. While motors represent 50-70% of the system load, one obvious method of dynamic load representation is using a physical motor based model. Additionally, discharge lighting, thermostatically-controlled loads, power electronic devices, speed drives, transformer tap changers, and capacitive reactive power banks, are all non-motor dynamic loads. Another method of dynamic load modelling is using a set of differential equations with respect to time. This method will be best described under 3.6 in this chapter. Until then, we will focus our study on the progression of the dynamic load representation as physical-based devices.

The Western Electricity Coordinating Council (WECC) had put forward an interim complex load model, CLODBL [9] as in Figure 2.3 which can still be found used sparsely today. Besides having only a fixed 20% induction motor load throughout the entire system, the load was also directly connected to a high voltage bus which is not recommended. Then comes the Southern California Edison (SCE) proposed model Figure 2.4. SCE model and interim models were very handy in developing the next composite model, CMLDBL [17] as in Figure 2.5. This model had special sub-models for residential air-conditioners.

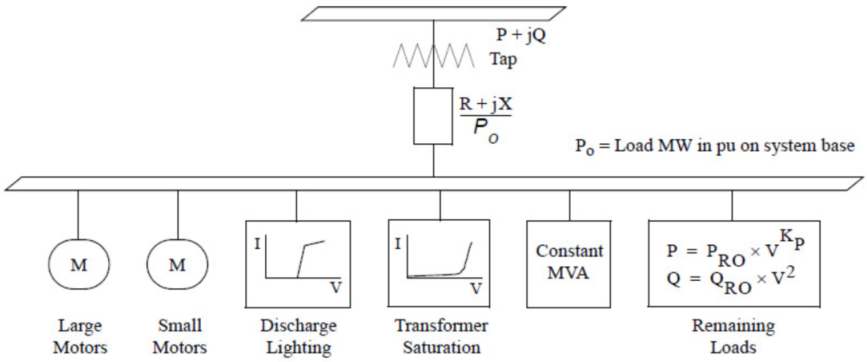


Figure 2.3: Interim complex load model from WECC

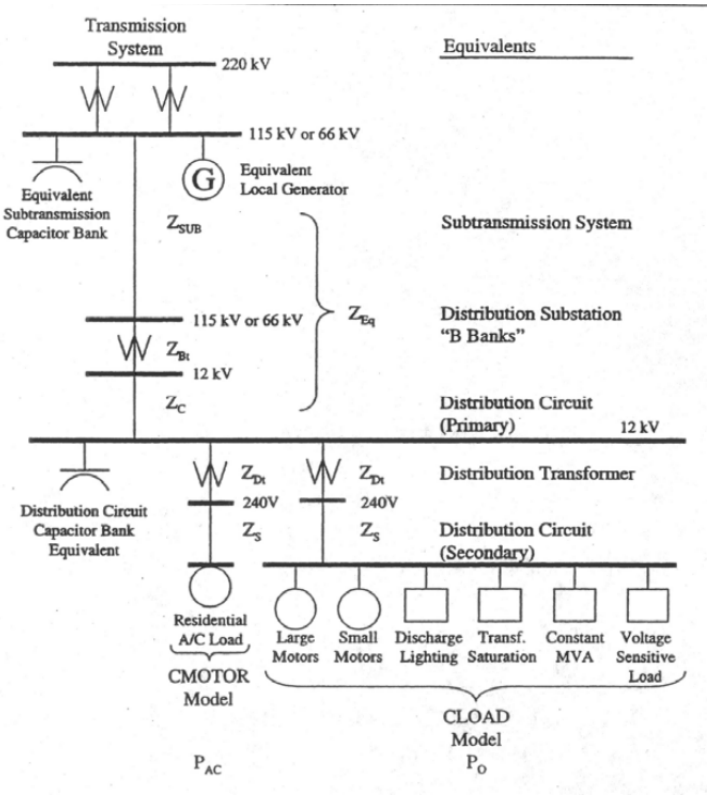


Figure 2.4: SCE Load Model structure

Composite model is the one with the combination of many components. A composite model inspired by the CMLDBL and the one used during this study is presented in Figure 2.6. Equations 2.5 and 2.6 represent equation for percentage composition of each component of model. So, one of the parameters is the percentage composition itself and rest are the ones carrying dynamic properties.

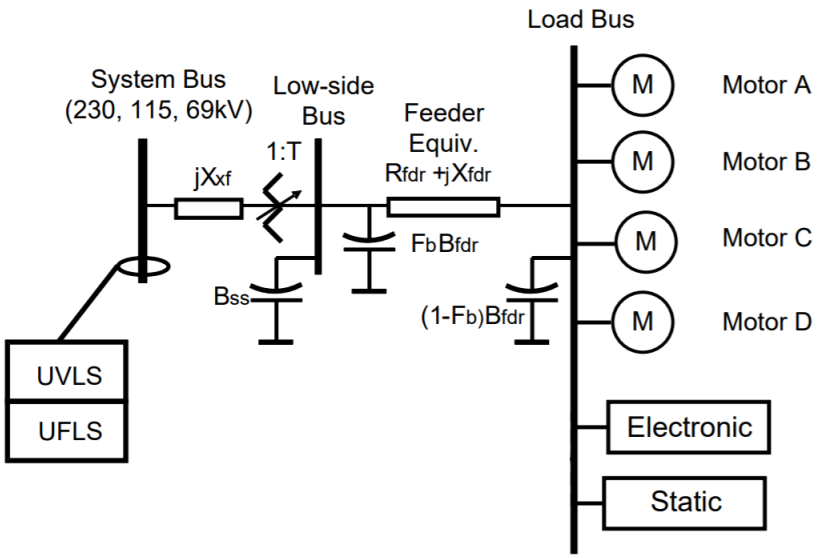
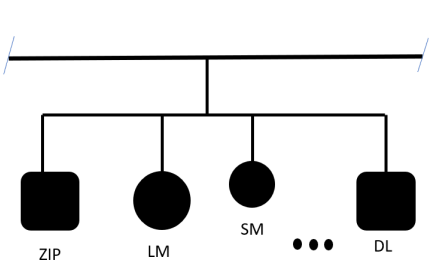
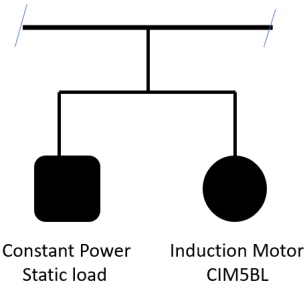


Figure 2.5: Complex load model CMLDBL



(a) Inspired from CMLDBL



(b) Composite Model used in this study

Figure 2.6: Composite model

$$p_{ZIP} + p_{LM} + p_{SM} + ... + p_{DL} = 1 \tag{2.5}$$

$$q_{ZIP} + q_{LM} + q_{SM} + ... + q_{DL} = 1 \tag{2.6}$$

2.3 Induction Motor Model

Induction motor model is the most often used dynamic model. The model parameters will vary depending on the type of model used. In this study, CIM5BL is used which has 20 parameters and out of which few are presented in the circuit diagram in Figure 2.7, this is the dual cage induction motor referred to the stator side. Appendix A.1 has all 20 parameters and some information on them as found on the manual of PSSE API [11].

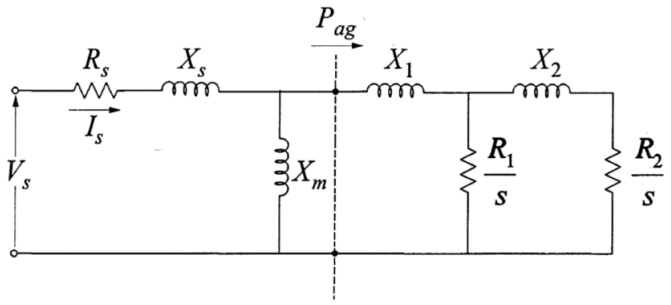


Figure 2.7: Induction motor model circuit

2.4 Parameter Estimation of Composite model

Western Electricity Coordinating Council (WECC) based composite load is the one that sufficiently represents the present electric load scenario. Figure 2.6 shows a ZIP connected to dynamic load. Hence, the job for parameter estimation is to figure out the percentage composition of both of the sub-models and estimate the dynamic behaviour parameters too. With the increase in the number of parameters to estimate, the parameter estimation process gets challenging.

Most of the prior works on dynamic load modelling with the composite model approach are based on optimization methods and genetic algorithms [18][19][20]. The drawback of optimization method and genetic algorithm is that they are hard to apply in real-time. Also, with the optimization approach, a defined function is necessary to solve the problem. Whereas Machine Learning methods just need a separation between inputs and outputs in measurement data and it learns the non-linear function itself. As mentioned before, we are also looking for real-time applications which is possible with ML models as a learned ML model does not take much time for prediction or updating of weights and biases. [8] has applied this approach with the use of neural networks. In this study, a new form of neural network called Recurrent neural network is applied to see the result again the feed-forward neural network. Also, during feature extraction, it is essential to utilize just the portion separated for training and not the whole data. This is corrected from [8] approach along with the implementation of a unique variation in ANN called multivariate input and single output or many to one.

2.5 Transfer function/Differential equation based dynamic load

Power electronics devices are increasingly used in renewable energy resources, power grids, and various types of industrial facilities. The modelling needs for such devices become more

important as their penetration level in power systems increases over time.[21] proposes a linearization approach is for a generic dynamic modelling method for power electronics devices in power systems. [22] utilizes this approach with Support vector machines and a single collection of field data. In this study, a novel approach of Long-Short Term Memory(LSTM) is tested along with the concurrent method of support vector regression (SVR). These approaches are trained with multiple disturbance data rather than just one to see the benefit in generalization.

Chapter 3

Physical based Induction motor dynamic load

With the advent of new technology and consumer needs arise. These new types of loads mainly air-conditioning load and electronic loads complicate the power network. As we saw in the previous section, power systems behave completely different from disturbances now that load composition has changed. That is why it is vital to incorporate them into the contingency study. As discussed under Literature Review, these types of time-dependent loads can be quite sufficiently represented by the addition of an induction motor model during analysis. Motors constitute between 50- 70% of electrical load in United States [23] [24]. The physical-based induction motor will be used as a dynamic load for study in this thesis. Work beyond this point will be to come up with proper methods to estimate parameters associated with induction motor load.

The criteria here is to figure out a non-linear regression practice that can be primed to estimate a set of floating value parameters in almost real-time. Table 3.1 has listed the only parameters set affecting dynamic behavior while Appendix A.1 has all 20 parameters and their briefs. Machine learning fulfils our necessity as ML is used to map multivariate inputs to output variables using a differentiable function.

3.1 Composite load model as dynamic load representation

In bulk power system studies, a load can be represented using a composite model as in Figure 3.1. This is the combination of static and induction motor load. Collecting all the power demand at a substation and treating it as a single power-consuming device is the established way of representing loads in higher voltage. As mentioned under literature review, static loads are quite easy to model and this work of ours will more closely focus on the dynamic modelling, we decided to fix out static load as a constant power load, the percentage composition of which is to estimate. Whereas, the dynamic load is represented by parameters listed on the Table 3.1 . These are selected parameters of induction motor representation, CIM5BL, which have an impact on how the load reacts during disturbances. Our job here is to come up with a strategic method to estimate them. This methodology

can be carried on to accommodate more complex dynamic load representation.

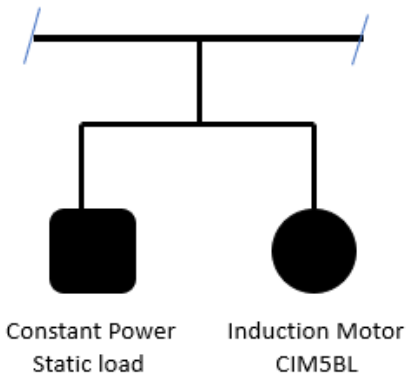


Figure 3.1: Composite load representation

Parameter	Description
LM	% Induction motor Composition
SL	% Static Load composition (100-LM)
R_s	Stator Resistance
X_s	Stator Reactance
R_r	Rotor Resistance
X_r	Rotor Reactance
X_m	Magnetizing Reactance
R_2	Rotor Resistance 2 (Dual Cage)
X_2	Rotor Reactance 2 (Dual Cage)
H	Inertia

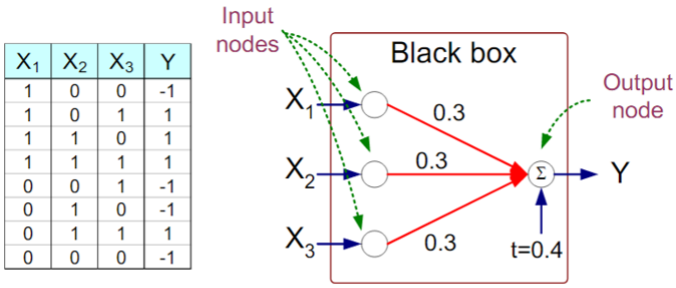
Table 3.1: Composite model parameters

There are 10 parameters listed, out of which SL can be determined if LM is known. So, all our efforts will work on to estimate the rest of the 9 parameters. As concluded in Literature Review, our problem is better solved with Machine Learning algorithms.

3.2 Machine Learning

The idea of getting machines to program themselves is very compelling [25]. That is what Artificial Intelligence(AI) is for most of us. Machine Learning(ML) is one aspect of AI, whereby a computer is programmed with the ability to learn by itself and improve its performance over time. If programming is automation, then machine learning becomes automation of the process of automation. After its initial boom in applications in the field of Computer Science, we can now see ML being utilized widely over all the fields. The only requirement is access to sizeable data. What ML does is that it parses through large data sets and learns relevant relationships between inputs and outputs to construct a trend model. The most basic ML model is a Perceptron [ANN1].

Figure 3.2 shows an example of Perceptron. Left table shows sample data set with three input dimensional in X_1, X_2,and X_3 and one categorical output Y. At one epoch, every single data sample will be passed into perceptron and the weights, fraction values above red connecting lines, are updated. This process can go up to multiple epoch depending on the size of the data set. After learning, the result of a perceptron is the updated weight values. These values can be applied to a new data set with the output Y unknown to estimated $\hat{Y}$. This phase is called the prediction.



$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Figure 3.2: A simple Perceptron structure

3.2.1 Supervised and Unsupervised learning

There are two main types of learning from data: Supervised and Unsupervised. Supervised learning is learning with knowledge of output. Here, data is labelled with a class or value. The ultimate goal is to predict the class or value label. Some of the popular supervised learning algorithms are Neural Network, Support Vector Machines, Recurrent Neural Network, and, Bayesian Classifiers.

Unsupervised learning is learning without the knowledge of output class or value. Here, data is unlabelled or value is unknown. The ultimate goal is to determine data patterns or groupings. This type of algorithms can come up with different outputs on different runs because they are self-guided and their learning function is not fixed. Some of the popular unsupervised learning algorithms are K-means, genetic algorithms, and, clustering approaches. Figure 3.3 illustrates both of these concepts graphically.

3.2.2 Classification and Regression

This work primarily focuses on supervised learning. Further supervised learning can be subdivided into classification and regression. Classification algorithms group the outputs into respective classes. It approximates a mapping function from input variables to discrete or categorical output variables. There cannot be any new output variables beyond what was used during learning. A classification problem with two classes is called binary, more than two classes is called a multi-class classification. Classifying an email as spam or non-spam is an example of the classification problem.

Regression is the task of predicting a continuous quantity. It approximates a mapping function from input variables to a continuous output variable. The predicted output can be a new continuous value without bound. A regression problem with multiple input variables

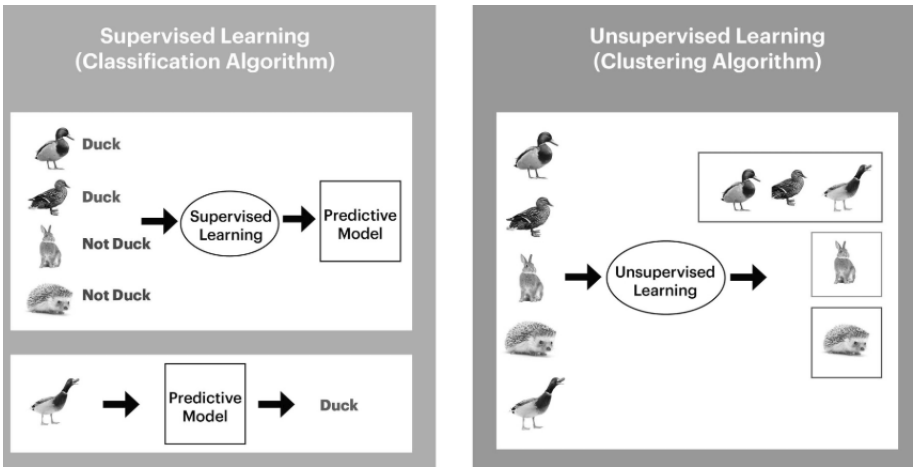


Figure 3.3: Supervised vs Unsupervised learning

is called a multivariate regression problem. Prediction the price of a stock over a period of time is a regression problem. Figure 3.4 illustrates the basic result of both types of learning.

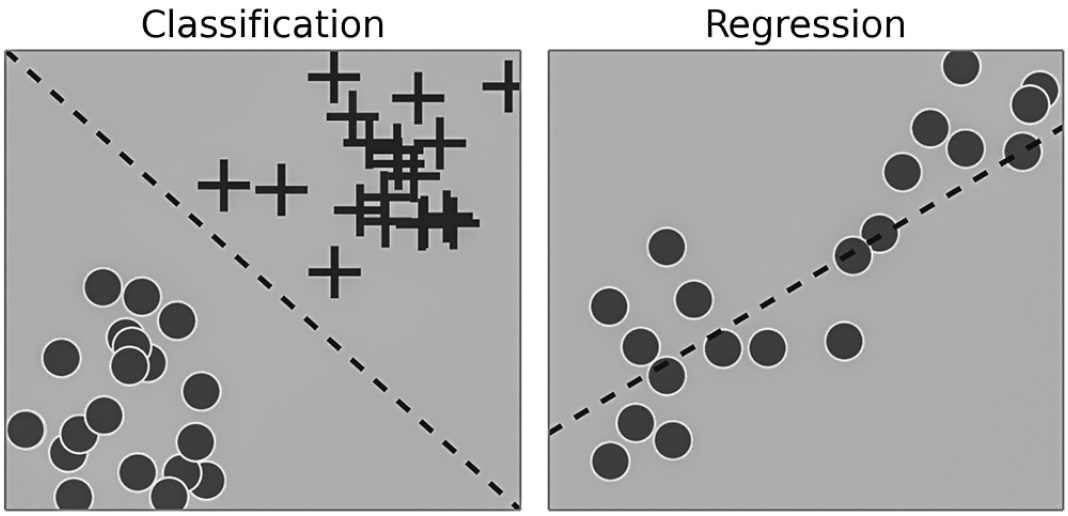


Figure 3.4: Classification vs Regression learning

3.3 Artificial Neural Network

Artificial neural network(ANN) is one of the few machine learning algorithms experimented in this work. ANNs are the feed-forward types of neural networks. The information travels only from the input end to output; however, weights of each neural link are updated during

backward travel of algorithm called propagation [26]. The simple type of neural network in perceptron which is without hidden layers. Adding hidden layers increases the ability of NN to learn more intricate behaviour but comes with the burden of increased complexity. ANN has very good scalability as any additional attributes to input is accommodated by adding neurons to the input layer. Not to mention, ANN learns by itself. The only necessary thing for the user to do it to differentiate the inputs and outputs in the data. ANN for the task of load modelling is acting as a differentiable function mapping input measurements to output parameters. In our setup, measurements are taken at a bus of bulk load, which is the collection of small individual loads with their separate parameters of representation. If we were to try to find a mapping function for such a complex system without the aid of machine learning, it would be very tedious.

3.3.1 ANN Structure

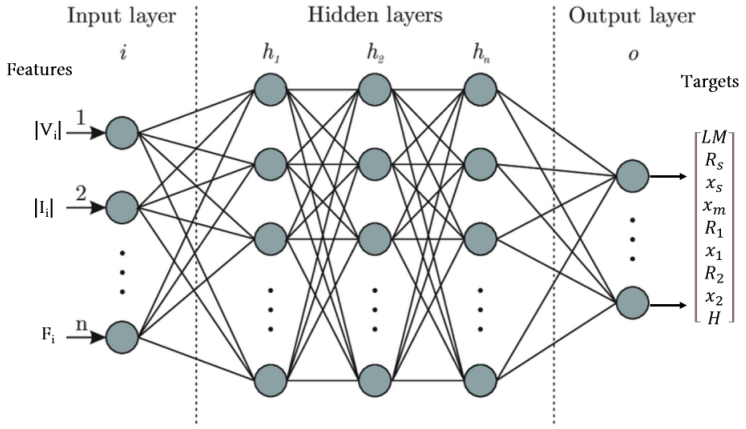


Figure 3.5: Structure of ANN

1. **Input neurons:** This is the total number of attributes that ANN used to learn. For input data, this must be the number of relevant features only. Rest of the feature which is co-related or maybe very niche to a single data-set must be removed. This helps in the generalization of the model.

2. **Output neurons:** This is several predictions that ANN is built to make. For our purpose, we will be doing regression. So, the output neurons will be giving out continuous values and the total number of output neurons will be 9- one for each parameter.

3. **Hidden Layers and Neurons:** Unlike both input and output layers and neurons, the number of hidden layers and neurons are very much dependent on the problem and chosen architecture of ANN. The aim, however, is to find the optimum number- not too big (increases complexity), not too small(compromises accuracy) [27]. The general

trend is to use the same number of neurons for all the hidden layers but gradually reducing the number of neurons as the layers increase will sometimes stop overfitting.*

4. **Activation function:** An activation function is the mathematical equations used to add non-linearity into the output of a neuron. It basically determines which neuron to activate and which to deactivate based on the relevancy of neuron's input for model's prediction. Rectified Linear Unit (ReLU) is the most common activation function. Equation 3.1 illustrates that it basically returns the positive part of the input fed. Fig 3.6 is the graphical representation.

$$f(z) = z^{+} \qquad = max(0, z) \tag{3.1}$$

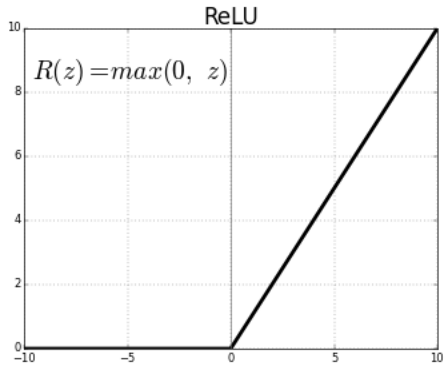


Figure 3.6: Activation function: ReLU

5. **Loss function:** The loss function is a method of evaluating how well the algorithm is modelling the dataset. The objective of the algorithm is to minimize the loss function. A general definition of the loss function can be found in Equation 3.2 where C of x is the regularization term used to soften the hard boundary and incentivize generalization. Mean squared error is the most common loss function to optimize.

$$Loss(x) = \frac{1}{N}\sum_{i=1}^{N} error(y_i - \hat{y}_i) - C(x) \tag{3.2}$$

6. **Batch size:** This is the term used for the number of training examples utilized in one run. The large batch size can be great only when the processing unit is powerful(like GPU). This helps to process more training instances per time. However, with CPU, relatively small Batch size is recommended to lessen the processing burden.

7. **Number of epochs:** This is a measure of the number of times all the training examples are used once to update the weights in ANN. With smaller batch size, multiple runs are needed to complete a single epoch. With multiple epochs, we have access to previously learnt weights rather than cold weights. Usually, starting with high epochs and stopping when training accuracy halts is recommended.

8. **Optimization and Backpropagation:** Optimizers are algorithms used to update weights of ANN with and aim to reduce the losses. Stochastic Gradient Descent is a variation of more common Gradient Descent that updates model weights after computation of loss on each training example. An extension to this is called Adam optimizer which is very broadly adopted on Machine Learning applications. Adam is fast and converges rapidly. Learning rate controls how big of a change in weights occur in every parsing. Having a smaller learning rate increases the chances of finding minima but are slower whereas large learning rate can miss minima and is not recommended.

3.3.2 Data Processing

Raw data should not be directly fed into the model. First, data should be checked for any null and outliers. Then, either delete the entire data sample or come up some simple predicting measure like replacing it with the average value. Data are generally divided into Training, Validation, and Testing groups. As the name suggests, Training data is used for updating the model weights, while Validation data is used to keep a check on the generalization ability of the model learnt but not to update the weights. Test data are solely used to validate the model's operation. The partition of the data is generally done as 70-15-15 % of the entire data. But before data division, following few steps are recommended to be carried out to ensure the better working of any ML model.

1. **Scaling and normalization:** Normalizing both input and output data is known to help the training and learning process. Since all neural networks assume that no two input variable is dependent. So, it is helpful to normalize them. The basic form of normalization used is bringing the data to zero mean and one standard deviation. In our case, the measurements recorded are in per units (pu), so it is found that there is no necessity of normalization for input variables. But the output variables are the parameters which are on different scales. So, output variables are normalized using

Equations 3.3-3.5.

$$Mean(\bar{x}) = \frac{1}{N}\sum_{i=1}^{N} x_i^n \tag{3.3}$$

$$Variance(\sigma_n^2) = \frac{1}{N-1}\sum_{i=1}^{N}(x_i^n - \bar{x_n})^2 \tag{3.4}$$

$$Normalization(\mathbf{x}_i^n) = \frac{x_i^n - \bar{x_n}}{\sigma_n} \tag{3.5}$$

2. **Feature extraction:** There is a popular concept in ML called the curse of dimensionality. This means a large number of attributes in the data space which ironically reduce the performance of the NN. There are several methods to combat this curse. One of the popular is called Principal component analysis(PCA). As the name suggests this extracts the principal components from the data space and therefore reduces the dimension. PCA is fit only on training data and the fitted PCA model is applied to both training and testing data.This helps to keep the testing data untouched and helps to realize true generalization ability of the model. Also, real time field data from PMU can now be transformed with this fitted PCA model before feeding to the ML network. Fig 3.7 gives a general illustration on its working- a 3d data is reduced to 2d data with pc1 and pc2 as the principal component.*

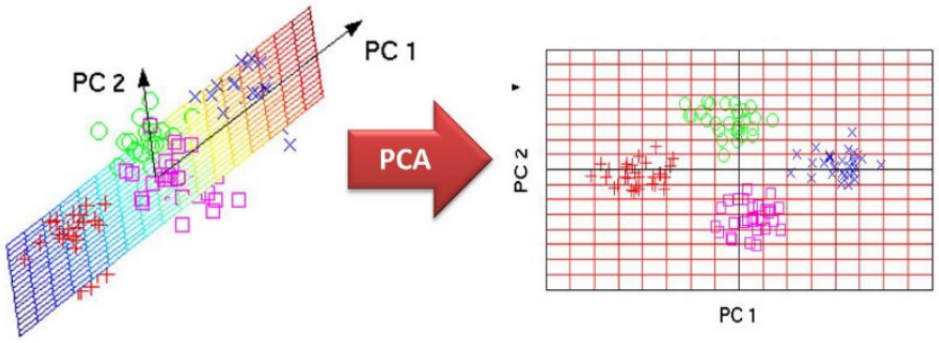


Figure 3.7: Working of PCA

3.3.3 Bagging ensemble

Bias and variance are two keywords that come up with predictive models. Bias is the error due to false assumptions during the training. High bias means the algorithm missed important relations and hence underfitting. Whereas variance is the error causing overfitting because of being sensitive towards irrelevant information during training. In most of the cases with the use of high epochs, algorithms tend to be high variance- low bias. To solve the

problem of high variance, an ensemble method called bagging is used. Bagging is the method of training multiple similar models with slightly different sets of input data and averaging output for the result. Fig 3.8 illustrates this concept. *

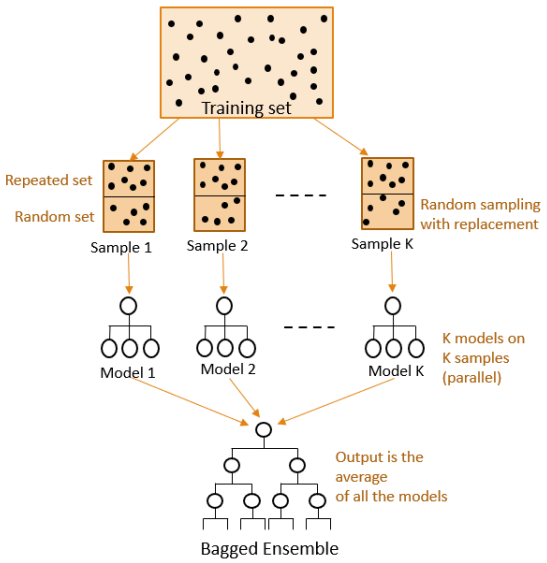


Figure 3.8: Illustration on bagging

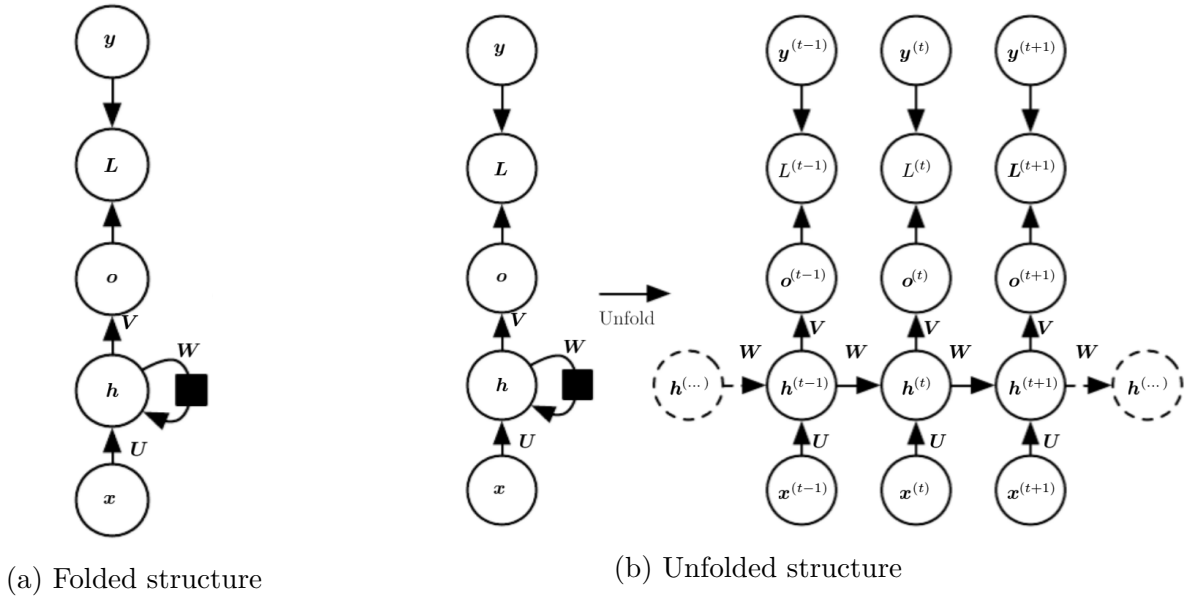
3.3.4 Many to One approach

Generally, for any worthwhile predictive models, the input is multivariate. Similarly, the output layer can also be predicting multiple variables at ones. This is a multivariate input multi-output approach. But what if each the output variable to be predicted have their own respective model i.e. weights and biases learned with just one output variable at a time. In this manner, every output variable to be predicted will have a dedicated model. This approach is called multivariate input single output or many to one approach.

3.4 Long-Short Term Memory Network

Recurrent Neural Networks(RNN) are separate from Feed-forward neural networks (like ANN) in that they possess a memory state with an ability to remember context like information about past input. This comes in handy during pattern recognition such as working with numerical time series data. In ANN, all the input attributes are considered as Independent whereas, with RNN, dependency on time is achieved. Figure 3.9(a) is a typical basic

structure for RNN. It looks like a regular ANN with single hidden layer except for a flow link connecting the hidden layer to itself. Figure 3.9(b) presents an unfolded RNN, which illustrates the meaning of self connecting link.



(a) Folded structure

(b) Unfolded structure

Figure 3.9: Basic structure of RNN

RNN can be looked as many copies of the same network, each advancing new information to the next-in-line. At each time instant (t), the RNN neuron gets two information from two separate sources: the regular input $x^{(t)}$ and its output from the earlier instant $h^{(t-1)}$. The way the network work is by using $o^{(t)}$ in every fold as output and comparing it with the real output variable $y^{(t)}$ to get error rate $L^{(t)}$ similar to ANN. After computing error, a technique called Back Propagation Through Time (BPTT), a modified version of regular Back Propagation, backtraces through the whole network and updates the weight depending on the error found earlier. In this manner, the network is adjusted to better itself during learning. Equation 3.6 represents vector of current input and the input seen before. The amount of information retained from previous time steps is represented by W vector. U represents the weight vector for the input neural links. And there is bias b_u. Equation 3.7 represents the non-linear activation function (like ReLU, tahn) for modeling complex data. It is important to note that $h^{(t)}$ is the same vector as $h^{(t-1)}$, its value is just tweaked over time steps. Whereas each x's are different and together they represent a sequence of data. Equation 3.8- 3.9 is the output at time t, with V being the weight vector for output link. Equation 3.10 represents loss at each time steps resulting from comparing prediction 'o' with actual output 'y'. Equation 3.11 is summation of all the losses. Mathematically speaking, learning with RNN becomes estimating V, W and U matrices with forward propagation and

BPTT.

$$a^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b_u \tag{3.6}$$
$$h^{(t)} = Activation[a^{(t)}] \tag{3.7}$$
$$c^{(t)} = Vh^{(t)} + b_v \tag{3.8}$$
$$o^{(t)} = Activation[c^{(t)}] \tag{3.9}$$
$$L^{(t)} = Loss(o^{(t)}, y^{(t)}) \tag{3.10}$$
$$L = \sum_t L^{(t)} \tag{3.11}$$

The concept of RNN is very promising but RNN itself suffers from vanishing gradient problem; which means they struggle in learning long-range relationships or simply put, they cannot remember much from few tens of steps before. Consider a simple recurrent network with no hidden units but with a recurrence on some scalar $x^{(0)}$ as in Figure 3.10. Equations 3.12- 3.15 both carefully illustrates how this problem may arise. To solve this matter, a special kind of RNN known as Long Short-Term Memory (LSTM) cell is used [28][29]. Adding LSTM is equivalent to adding memory to aid the network to remember from the instances long while before.

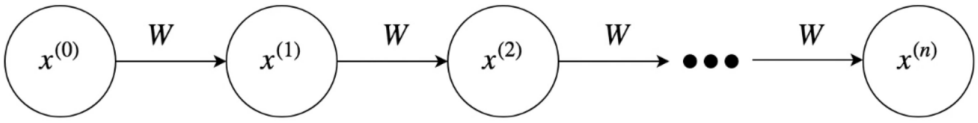


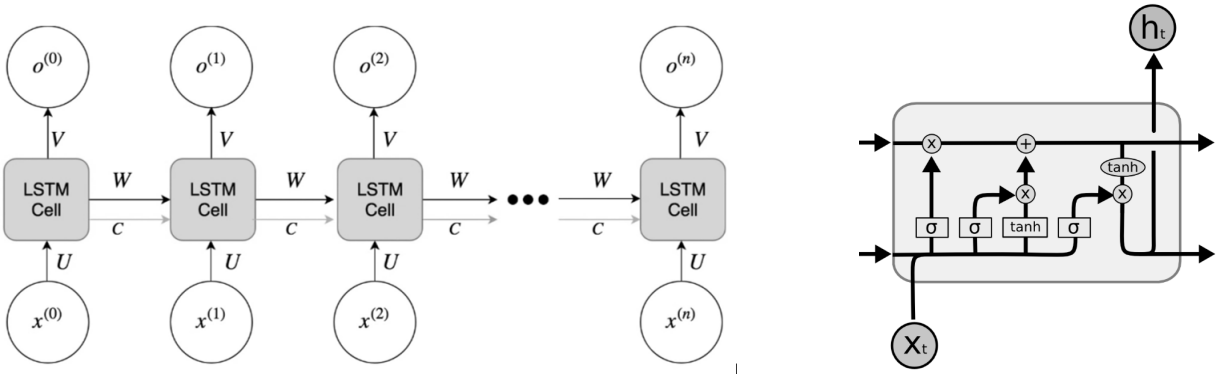
Figure 3.10: Illustration on RNN problem with long sequences

$$x^{(n)} = W^n x^{(0)} \tag{3.12}$$
$$x^{(i)}, W \in \mathbb{R} \; \forall \; i \in [0, n] \tag{3.13}$$
$$W^n x^{(0)} \rightarrow \begin{cases} \infty; \; W > 1 \\ 0; \; W < 1 \end{cases} \tag{3.14}$$
$$\frac{\delta W^n x^{(0)}}{\delta W} \rightarrow \begin{cases} \infty; \; W > 1 \\ 0; \; W < 1 \end{cases} \tag{3.15}$$

3.4.1 LSTM structure:

If hidden units in Figure 3.9 (b) are replaced with LSTM cell and added another connection from every cell is called Cell State (C), this new structure is LSTM as shown in 3.11. LSTM

is designed to mitigate the vanishing and exploding gradient problem. Apart from hidden state vector, each LSTM cells maintains the Cell state vector and at each time step, The next LSTM cell can read from it, write to it or reset the cell using an explicit gating mechanism. Each unit has three gates of the same shape.



(a) LSTM Structure

(b) LSTM Cell

Figure 3.11: Basic structure to LSTM

1. **Input Gate:** Input gate controls whether the memory cell is updated. Figure 3.12 (a) and Equation 3.16 illustrates its working. This gate has sigmoid activation. It's because sigmoid constitutes in the range 0-1 and remains differentiable.

$$i^{(t)} = \sigma(W^i[h^{(t-1)}, x^{(t)}] + b^i) \tag{3.16}$$

2. **Forget Gate:** Forget gate controls if the memory cell is reset to 0. Figure 3.12 (b) and Equation 3.17 illustrates its working. It also has sigmoid activation for the same reason.

$$f^{(t)} = \sigma(W^f[h^{(t-1)}, x^{(t)}] + b^f) \tag{3.17}$$

3. **Output Gate:** Output gate controls whether the information of the current cell state is visible. Figure 3.12 (c) and Equation 4.13 illustrates its working. It also has sigmoid activation for the same reason. Apart from these gates, there is another vector $\bar{C}$ that modifies the cell state as in Equation 3.19. It has *tanh* activation because with a zero centred range, a long sum operation will distribute the gradients pretty well. This allows the cell state information to flow longer without vanishing or exploding.

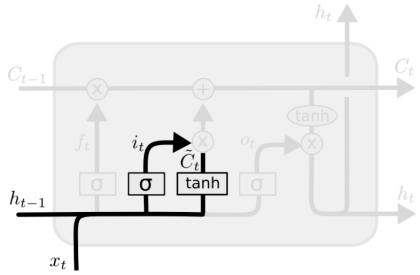
$$o^{(t)} = \sigma(W^o[h^{(t-1)}, x^{(t)}] + b^o) \tag{3.18}$$

$$\bar{C}^{(t)} = tanh(W^C[h^{(t-1)}, x^{(t)}] + b^C) \tag{3.19}$$

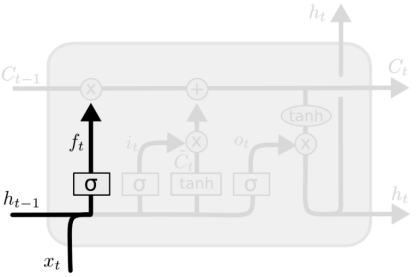
$$C^{(t)} = f^{(t)}C^{(t-1)} + i^{(t)}\bar{C}^{(t)} \tag{3.20}$$

$$h^{(t)} = tanh(C^{(t)}) * o^{(t)} \tag{3.21}$$

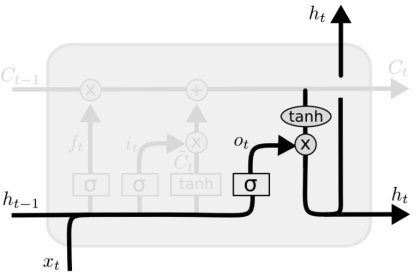
Equations 3.20 and 3.21 represent overall working equations for all gates. Hence, total parameters to learn is $\{b^i, W^i, b^f, W^f, b^C, W^C, b^o, W^o\}$



(a) Input Gate



(b) Forget Gate



(c) Output Gate

Figure 3.12: LSTM Gates

4. **Dropout:** Dropout is a regularization technique popular for giving a huge performance boost for how straightforward the method is. Basically what dropout does is to randomly turn off a percentage of neurons at each layer, for each training step. This helps to make the network more robust as the network cannot rely on a defined set of input neurons for decision making. Hence, the chances of overfitting are reduced. Usually, the dropout rate is best taken between 0.1 to 0.5 for LSTMs. With the increase in layers, the dropout rate should be increased.

5. **Batchnorm:** Batchnormalization or batchnorm is performing regular normalization in between layers i.e. making the output from hidden layer zero mean and one standard

deviation. With this, we can use larger learning rates and hence the algorithms tend to converge faster. It also reduces the vanishing gradient problem by zero centering the data which helps to map a very low value to a somewhat significant one.

3.4.2 Data Processing:

1. **Normalization:** We repeated our scaling and normalization process from the ANN using Equations 3.3-3.5. This helps LSTM to learn much easier and faster.

2. **Downsampling:** While in ANN, we used PCA to reduce the dimension of the data. With LSTM, it is necessary to save the trend of data. PCA removes the timestamps and provides only the principal components from the data. Our simulated data was 1.09 seconds long with 720 frames per seconds recording speed which gives a total of 793 instances. Since LSTM is sequential input network, feeding 793 instances with 3 inputs (Voltage, Current, and Frequency) per instance is a long process. Hence, we opted to use downsampling method called Piecewise Approximate Aggregation (PAA). PAA downsampled the original time-series to 60 frames per second which eased the computational burden on LSTM network. PAA primarily represents a piece of the time-series by its arithmetic mean. Appendix C has its code.*

3.4.3 Stacked LSTM

Although LSTMs are innately deep in time, as their hidden state acts as a memory for many previous time instances. But, what if LSTMs are also made deep in respect to space i.e. stacking multiple LSTMs [30] layers on top of each other similar to how ANN stacks its hidden layers. Figure 3.13 represents an illustration of stacked LSTM. Front LSTMs can provide sequence output to the succeeding LSTMs even though the final output is a single vector. In this manner, earlier LSTMs can be used to learn the intricate pattern in the data.

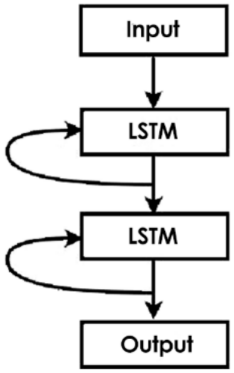


Figure 3.13: Illustration of Stacked LSTM architecture

3.5 Test System and simulation

ANN and LSTM described in the previous sections in this chapter will be applied to measurement data collected from the IEEE 118 bus system. IEEE 118 bus system is chosen because it is large enough system close to a real one but not too big to complicate the primary work in this research- dynamic load modelling. There are 118 buses, 186 branches, 54 generators, and, 91 loads. We chose PSSE as our simulation platform. We started with raw data for IEEE 118 bus system which contained system information like bus information, and, line/shunt information. In order to perform a dynamic simulation, another set of data file called dynamic (.dyr) file was built. This file contained information on transient equipment like excitor, governor, load and generator. Fig 3.14 shows IEEE 118 bus system ready for simulation with the additional circuit at bus 11.

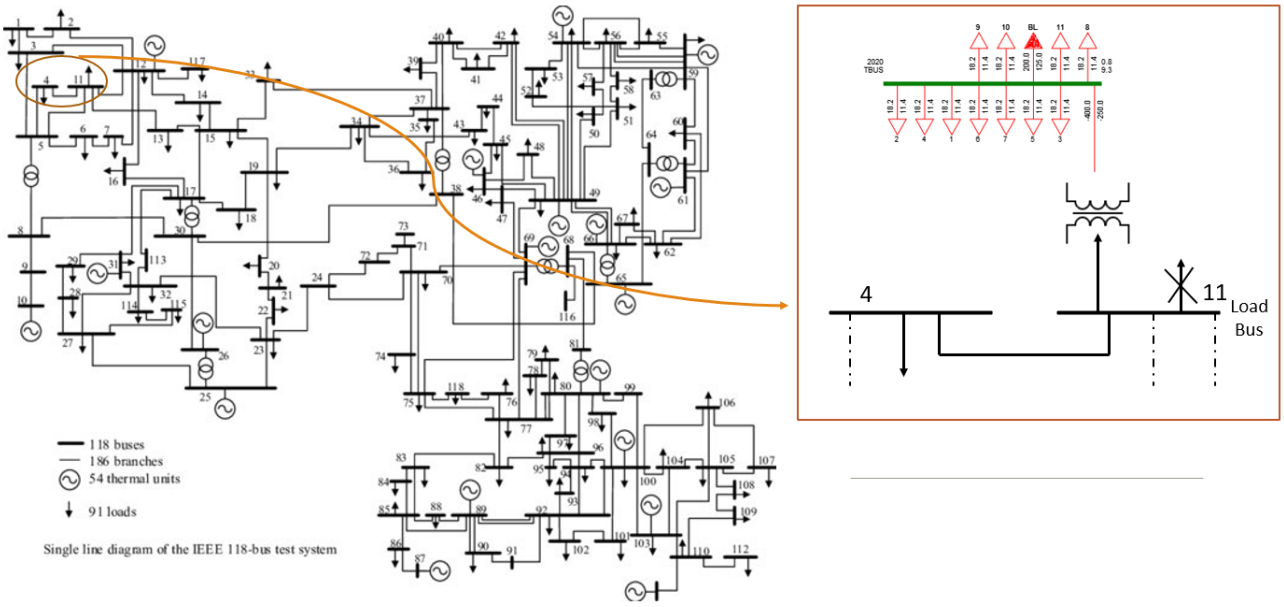


Figure 3.14: IEEE 118 bus test system

An additional circuit here is the composite load with constant power and induction motor side by side. But unlike static load, induction motor load is not recommended to be placed at high voltage levels. Hence, a new bus '2020' is placed on the other end of the step-down transformer and induction motor load is split into smaller individual loads with the exact same characteristics. This setup is to replicate a single large composite load. The original load at bus '11' is disconnected and the equivalent composite load was built.

Simulation setup:
Power System Simulator for Engineering (PSSE) is used as a simulating environment. It is a software tool developed by Siemens mostly used to simulate electric power transmission networks. PSSE has many useful features, we mostly used it to do a steady state (load

flows) and dynamic studies (faults application). In our case, the IEEE 118 bus system is first solved in steady-state to see its stability. This system will be a revised system with mew composite load connected at bus '11' with load composition and dynamic parameters at every run. Then dynamic simulation is set up starting with a set of pre-initialization processes listed below.

1. NETG : Netting of all the generations
2. CONL : Converting all loads
3. CONG : Converting all generators
4. ORDR : Ordering Buses for easy matrix manipulation
5. FACT : Factorizing the Network Admittance matrix
6. TYSL : Solving the converted case

After these steps are fulfilled without errors, dynamic simulation is run. Every simulation is of 1.09 seconds and data capture rate is 720 frames per second. The recorded data for the use in this chapter are Bus voltage, Bus current and change of bus frequency. Both the voltage and current had magnitude and angle data recorded but upon application, we found that adding angle data does not add any additional information, so only voltage magnitude, current magnitude and change in frequency data are eventually used. ML algorithms learn from the variation in system reaction to different load parameters. Therefore, 3 phase to ground fault is placed on the neighbouring line (line 11-line 4). The simulated events are listed here with Fig 3.15 illustrating a sample plot.

1. Steady state Run from 0 to 0.2 seconds
2. 3 phase fault along transmission line between buses 11 and 4 at circuit '1', at t= 0.2 seconds
3. Run the faulted system till t = 0.29 seconds
4. Trip breaker and line 11-4 is disconnected at t=0.29 seconds
5. Run with disconnected line until t = 0.67 seconds
6. close the breaker, transmission line gets reconnected at t = 0.67 seconds
7. Fault is still on transmission line which runs till t= 0.77 seconds
8. Trip breaker and disconnect the faulted line at t =0.77 seconds
9. Run without fault and line 11-4 disconnected till t =1.09 seconds

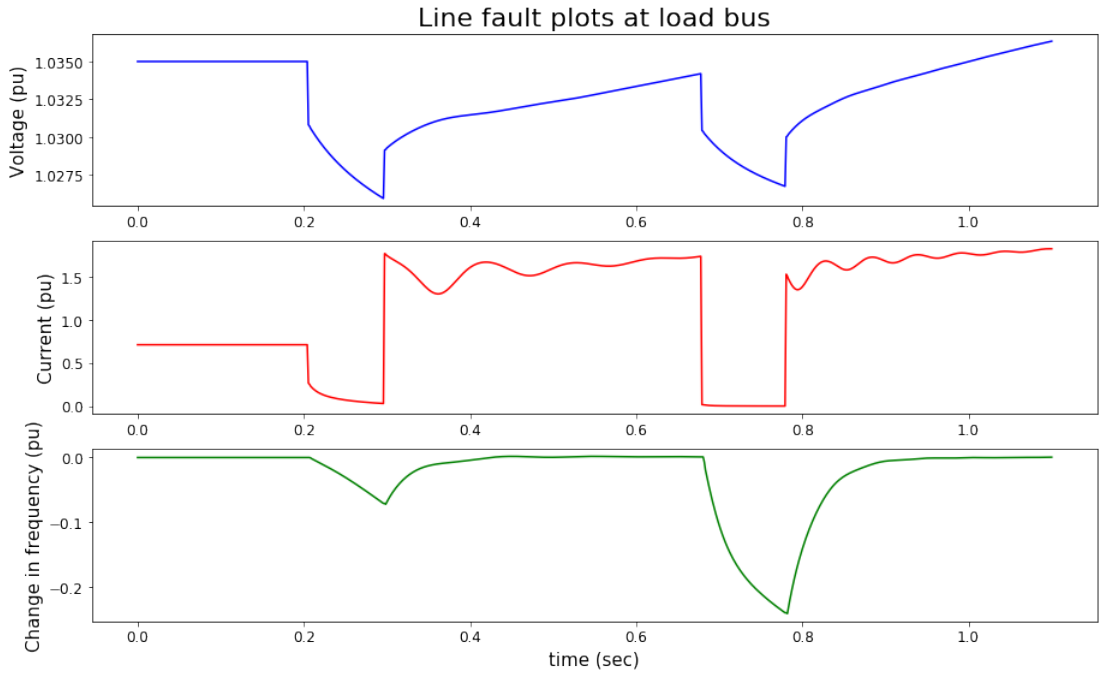


Figure 3.15: A sample Voltage, Current and change of Frequency plot at load bus

This process is repeated for each of the combinations of the parameter vector. This process is automated using python script and can be found in appendix A.4. In the end, a DAT file is generated with recorded data for each simulation run.

3.6 Parameter Estimation

The initial parameter estimation algorithm was built on a small scale to estimate only front four parameters to find the optimum architecture and hyperparameters. To do this, a python program as in appendix A.2 was used. It generated combinations of parameters and saved them in a text file with each line representing a unique combination. Table 3.2 shows the range for 4 parameters to be simulated and constant values for the rest. Similarly Table 3.3 shows the range for 9 parameters.

Figure 3.16 shows the flowchart for how training and predicting work will be taken care of.

3.6.1 ANN estimation

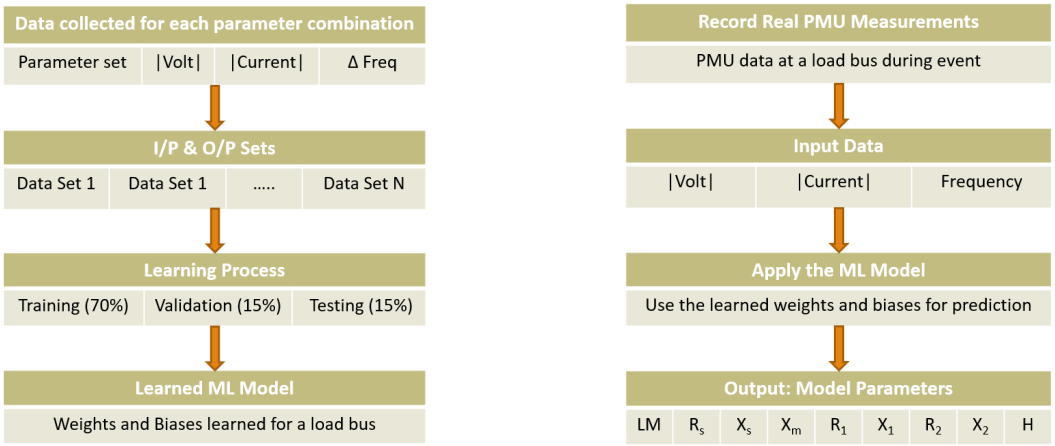
1. **4 param estimation:** Using the range in Table 3.2, 750 datasets were simulated i.e. 750 different combinations of parameters with the corresponding dataset to represent them. We normalized the target variables and went to feature extraction using PCA.

Estimated Parameters	Constant Parameters
$0 \leq LM \leq 100$	$R_r = 0.013$
$0.012 \leq R_s \leq 0.024$	$X_r = 0.067$
$0.05 \leq X_s \leq 0.2$	$R_2 = 0.009$
$3.5 \leq X_m \leq 6.5$	$X_2 = 0.17$
	H = 1

Table 3.2: 4 parameters simulation range

Estimated Parameters	Constant Parameters
$0 \leq LM \leq 100$	None
$0.012 \leq R_s \leq 0.024$	
$0.05 \leq X_s \leq 0.2$	
$3.5 \leq X_m \leq 6.5$	
$0.008 \leq R_r \leq 0.02$	
$0.05 \leq X_r \leq 0.2$	
$0.005 \leq R_2 \leq 0.015$	
$0.1 \leq X_2 \leq 0.2$	
$0.4 \leq H \leq 1.2$	

Table 3.3: 9 parameters simulation range



(a) Training and Validation Phase

(b) Testing and Predicting Phase

Figure 3.16: Flowchart on training and testing

The original input dimension was 793x3: 793 data points for each voltage magnitude, current magnitude, and change in frequency. Figure 3.17 is the feature extraction graph for voltage magnitude. The way to read the graph is to find the number of

components large enough to sufficiently represent the data or with a total coefficient of variance equal to 1. It was found that reducing down to 7 components was sufficient for the study. Likewise, Current magnitude and change in frequency followed a similar path. Hence, the new input dimension was reduced to 7x3 or 21 total dimensions.

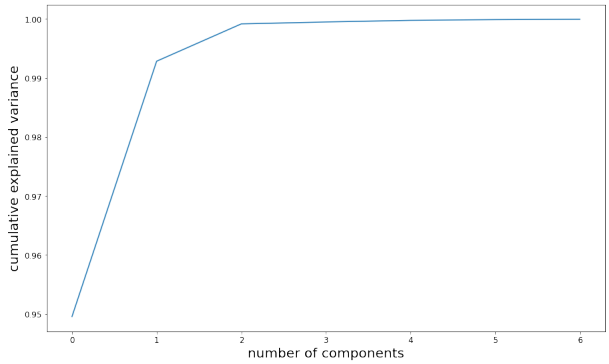


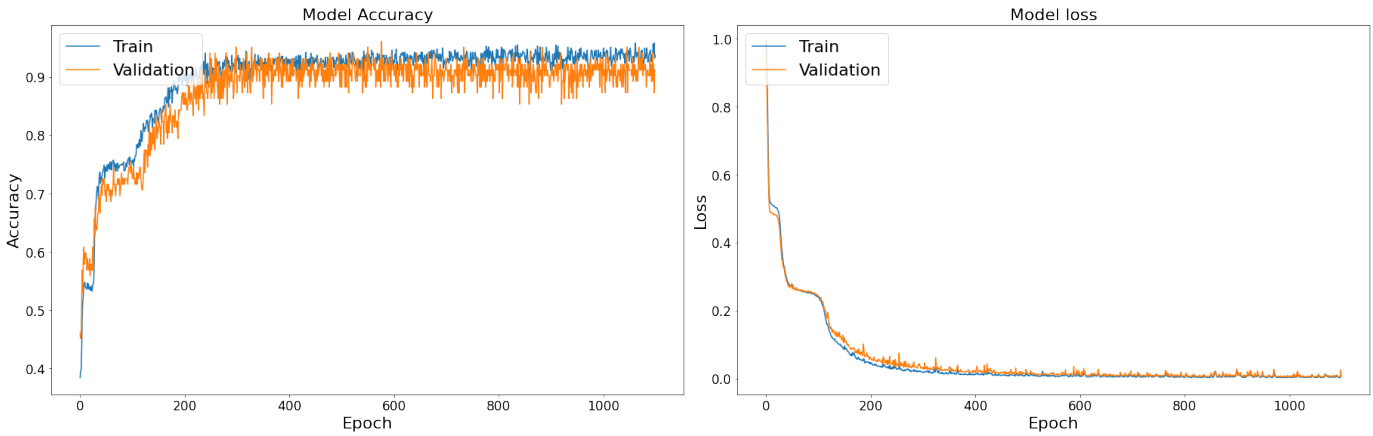
Figure 3.17: Principle Component Analysis on 4 parameter estimation data

The next step as shown in the flowchart is to divide the data into training, validation and testing sets and then start training. Table 3.4 shows the optimum architecture found. We started with a single hidden layer and 20 neurons architecture and trained to see the training/validation loss and accuracy. The training accuracy was not sufficient and consequently, we started adding new neurons. It is common knowledge in Neural Networks that adding a new hidden layer is more beneficial than adding too many neurons to existing layer. We iterated the process of adding components, with total neurons cap of 60, until a satisfactory accuracy was reached. Figure 3.18 shows the training loss and accuracy. The strategy used to decide the number of epochs is to watch for lowering validation loss and training loss closely and stop the training as soon as validation loss gets stuck or starts raising while training loss continues lowering.

Input Dimension	21
Output Dimension	4
Hidden layers	3
Number of neurons	56, 45 and 45
Activation function	ReLU
Optimizer	Adam
cost function	Mean Squared Error (mse)
Epochs	1100
Batch size	50

Table 3.4: ANN architecture for 4 parameter estimation

Bagging method: We decided to utilize the bagging ensemble to see the benefit over



(a) Training accuracy

(b) Training loss

Figure 3.18: ANN 4 parameter model accuracy and loss graph

regular ANN. Figure 3.19 represent the increase in r^2 score due to bagging. and Table 3.5 lists with-bagging and without-bagging scores.

	Training	Testing
Loss	0.0224	0.03
Accuracy	0.9126	0.93

(a) Loss and Accuracy score without ensemble

	Training	Testing
R^2 score	0.977	0.96

(b) R square score after ensemble

Table 3.5: (a) Loss and accuracy (b)Coefficient of determination

Validation Result Table 3.6(b) shows the error rate on the estimated parameters. Equation 3.22 defines the error rate for each parameter i, in our study; y_{max} and y_{min} represents the maximum and minimum values of each parameters respectively. Figure 3.20 compares the estimated response to the real response. They seem quite close to each other.

$$E^i = \frac{|\hat{y} - y^i|}{(y^i_{max} - y^i_{min})} \tag{3.22}$$

2. **9 parameter estimation:** Using the range in Table 3.3, 82000 data sets were simulated i.e. 82000 different combinations of parameters with the corresponding dataset to represent them. We normalized the target variables and went to feature extraction using PCA. The original input dimension was 793x3: 793 data points for each voltage magnitude, current magnitude, and change in frequency. Figure 3.21 is the feature

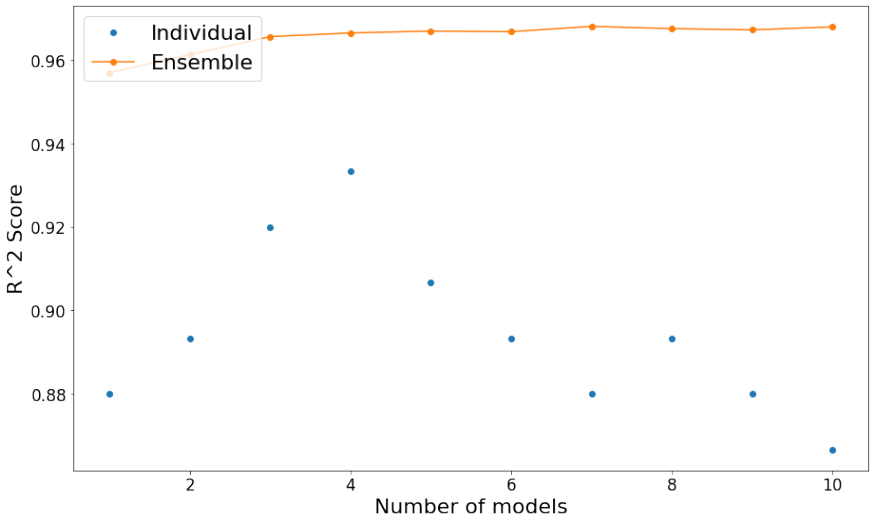


Figure 3.19: Result from ensemble approach

543 training data set
95 validation data set
112 testing data set

(a) Data division for 4 parameters estimation

Parameters	Error (%)
LM	0.944
R_S	1.03
X_s	0.82
X_m	5.154

(b) Estimated parameters error rates

Table 3.6: (a) Data division (b) Estimated parameters error rates

extraction graph for voltage magnitude. The way to read the graph is to find the number of components large enough to sufficiently represent the data or with a total coefficient of variance equal to 1. It was found that reducing down to 25 components was sufficient for the study. Likewise, Current magnitude with 35 components and change in frequency with 35 components followed a similar path. Hence, the new input dimension was reduced to 95 total dimensions.

The next step as shown in the flowchart is to divide the data into training, validation and testing sets and then start training. Table 3.7 shows the optimum architecture found. We set the optimum architecture from 4 parameter estimation as base case and used aforementioned strategy to figure out the optimum structure for this problem. Figure 3.22 shows the training error and accuracy.

Bagging method: We decided to utilize the bagging ensemble to see the benefit over regular ANN. Figure 3.23 represent the increase in r^2 score due to bagging and Table 3.8 lists with-bagging and without-bagging scores.

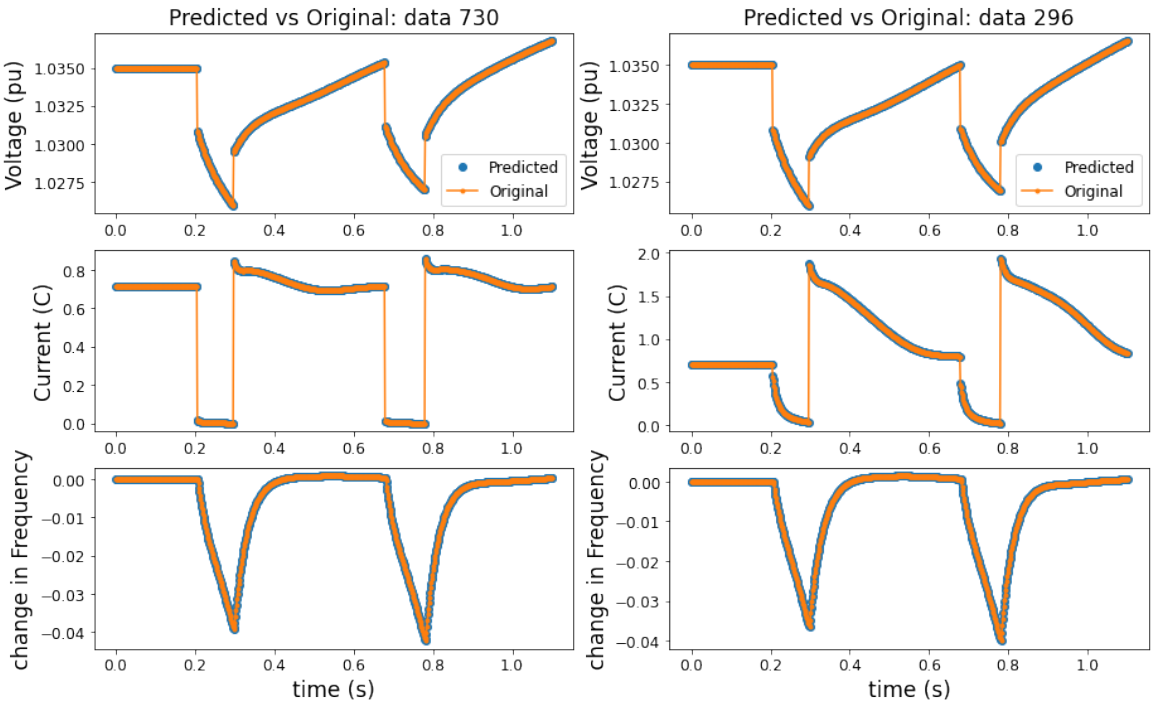


Figure 3.20: Validation graph

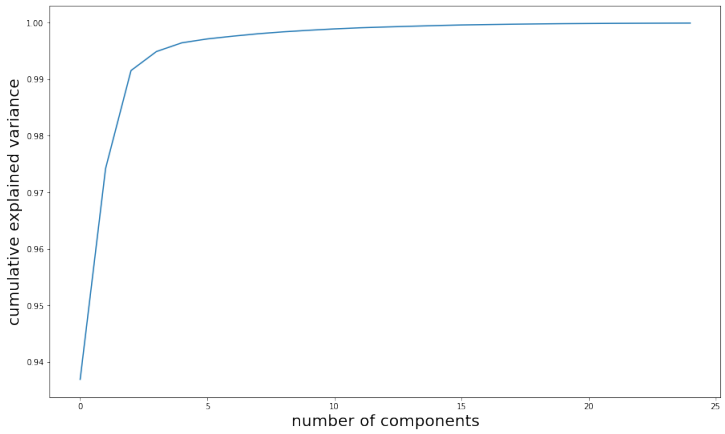
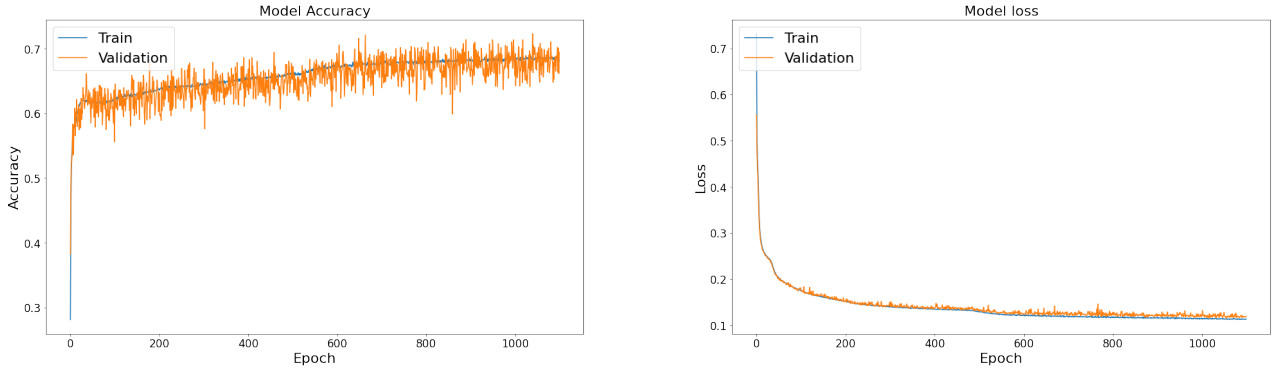


Figure 3.21: Principle Component Analysis on 9 parameter estimation data

Many to one approach and validation result: There is seen a little improvement in prediction from regular ANN to bagging ANN. However, there is a way utilized quite widely and accepted to work much better than ensembling methods. Instead of a single model predicting all nine output variables, we can do multivariate input to single variable outputs. This makes 9 different ANN model; each designated to solely work on to predict on output variable. Hence, weights and biases learnt will be customized just for one output variable as opposed to nine different variables like in regular ANN.

Input Dimension	95
Output Dimension	9
Hidden layers	3
Number of neurons	100, 55 and 30
Activation function	ReLU
Optimizer	Adam
cost function	Mean Squared Error (mse)
Epochs	1100
Batch size	450

Table 3.7: ANN architecture for 9 parameter estimation



(a) Training accuracy

(b) Training loss

Figure 3.22: ANN 9 parameter model accuracy and loss graph

	Training	Testing
Loss	0.118	0.13
Accuracy	0.6714	0.663

(a) Loss and Accuracy score without ensemble

	Training	Testing
R^2 score	0.887	0.874

(b) R square score after ensemble

Table 3.8: (a) Loss and accuracy (b) Coefficient of determination

Table 3.9(b) shows the error rate on the estimated parameters from both of these methods. Figure 3.24 shows loss function for model R_S and X_S respectively. Since the validation loss function for the R_S is reduced to almost 0, the error rate is very low for it whereas the validation loss is stuck at around 0.2 for model X_S and that's why its error rate is high. Figure 3.25 and 3.26 compares the estimated response to the real response. The estimated response seems to follow the real response much better with the many to one approach as opposed to the ensemble approach.

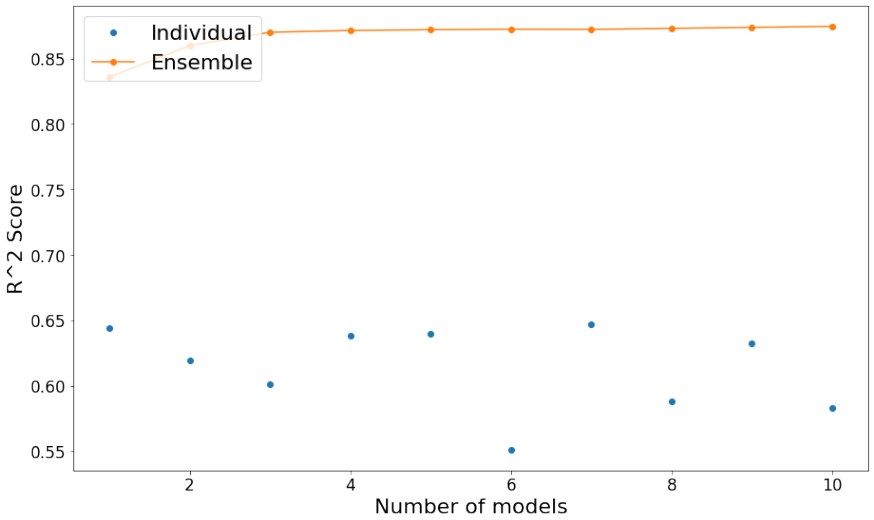


Figure 3.23: Result from ensemble approach

59245 training data set
10455 validation data set
12300 testing data set

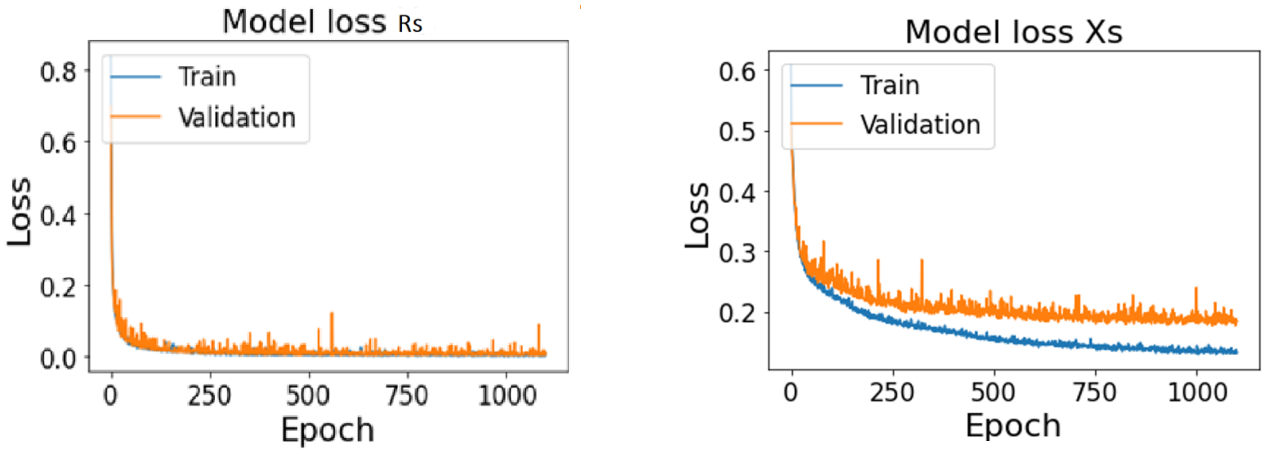
(a) Data division for 9 parameters estimation

Parameters	Err(%) Ensemble	Err(%) Many to one
LM	0.85	0.18
R_S	3.33	0.87
X_s	14.36	11.98
X_m	27.6	25.75
R_1	2.01	0.12
X_1	14.43	12.59
R_2	1.85	0.0117
X_2	2.94	0.212
H	0.93	0.089

(b) Estimated parameters error rate

Table 3.9: (a) Data division (b) Estimated parameters error rates

3. **Investigating Error percentages:** The main objective of this research is to successfully train ML algorithms to estimate all dynamic load parameters. Dynamic load parameters are defined as parameters that affect the dynamic behaviour of the power system. We started with four parameters of complex load model and went on to estimate nine parameters and Table 3.9(b) reflects the error rate, which denotes how far is the estimation from the real value. Anything around 10 per cent is generally acceptable for these kinds of study. As seen in the table, the majority of the parameters have an error rate below 1 per cent. Parameters X_s and X_1 are slightly higher than 10 percent but shockingly X_m has almost 26 %. However, validation graph in Figure 3.26 shows the predicted dynamic response has almost perfectly captured the real response.



(a) Model loss for Rs

(b) Model loss for Xs

Figure 3.24: Model loss for two dedicated model

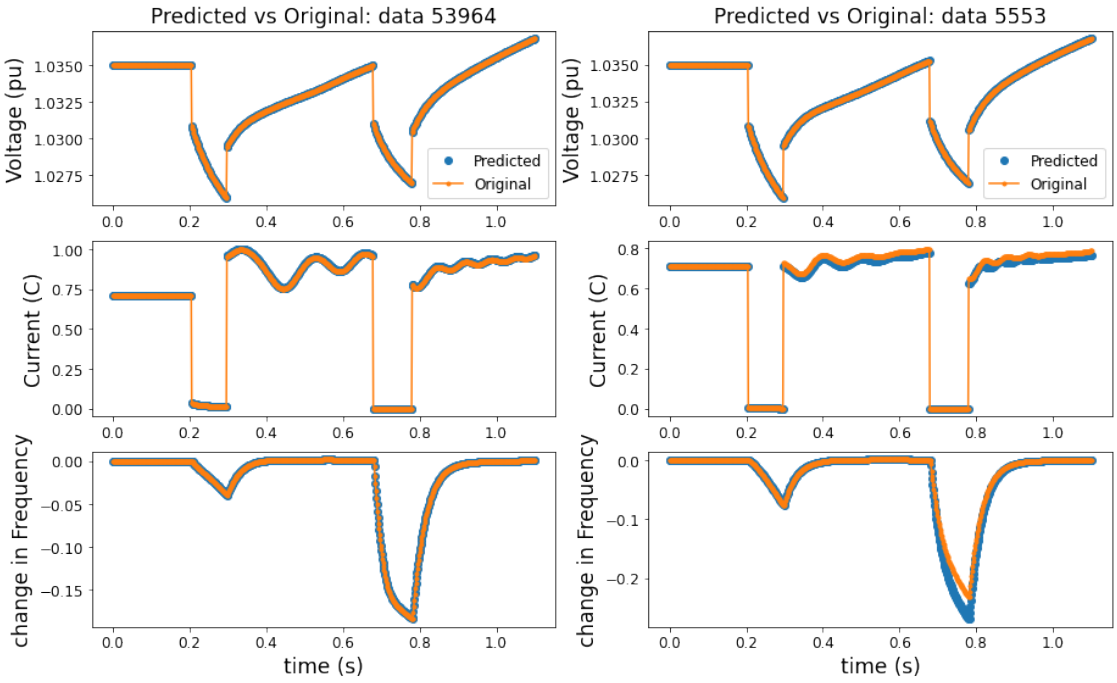


Figure 3.25: Validation graph with many to many bagging

All parameters must create enough variation in the dynamic data for the algorithm to properly learn their behaviour. So we checked the variation in dynamic response due to each parameter by comparing the minimum and maximum value of that parameter while every other parameter is at their minimum value. Voltage and current magnitude response along with the change in frequency at the load bus are recorded. Figure 3.27 shows the comparison graphs for LM and X_m(plots for rest of the parameters are in

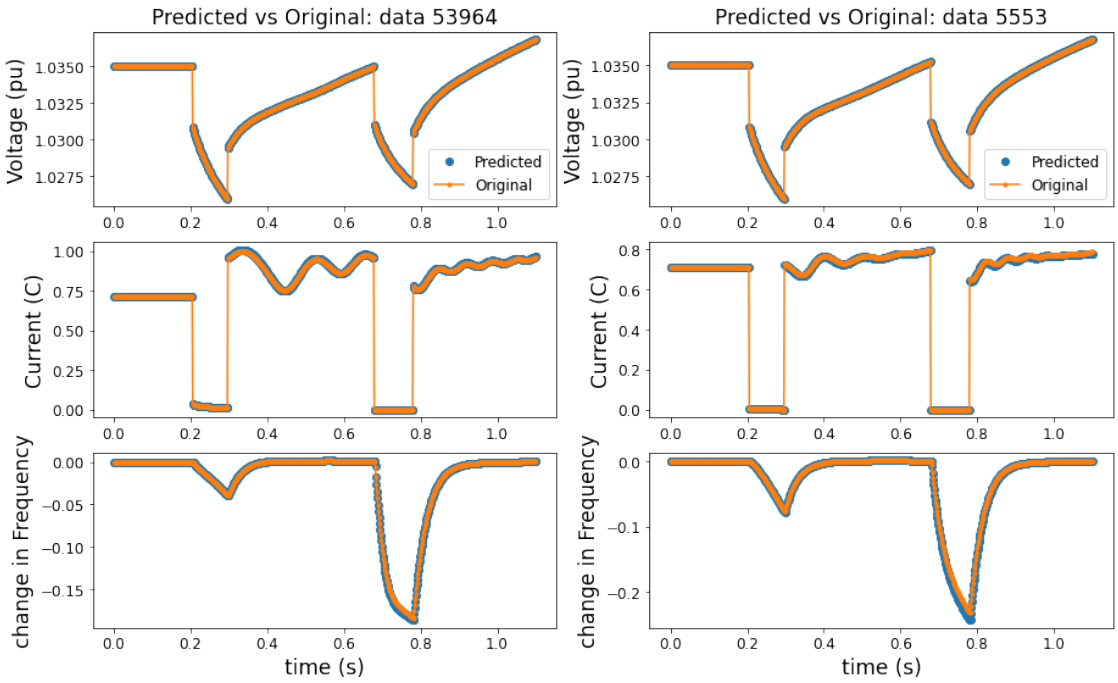


Figure 3.26: Validation graph with many to one

Appendix B). It is apparent that the range of X_m is not creating any sort of variation on the data. This infers that our choice of range for X_m is not sufficiently participating to create variation. Changing the range and utilizing similar methodology should in-fact work as seen with every other parameter here. Also now onward, all the work on this chapter is done keeping X_m error rate insignificant.

3.6.2 LSTM Estimation

LSTM can be looked as similar to the normal neural network but instead of increasing the number of input dimensions and input layer's neurons. The data enters sequentially unlike other machine learning algorithms. LSTMs are a type of recurrent network, and as such are designed to take sequence data as input, unlike other models where lag observations must be presented as input features.Using the setup of 9 parameter estimation from ANN, the experiment was run with LSTM network.

We will develop a model with a two hidden LSTM layer with 128 units each. The number of units in the hidden layer is unrelated to the number of time steps in the input sequences. The LSTM layer is followed by a fully-connected layer with 32 nodes that will interpret the features learned by the LSTM layer. Finally, an output layer will directly predict a vector with one element; in this manner, there will be nine models dedicated to each parameter.

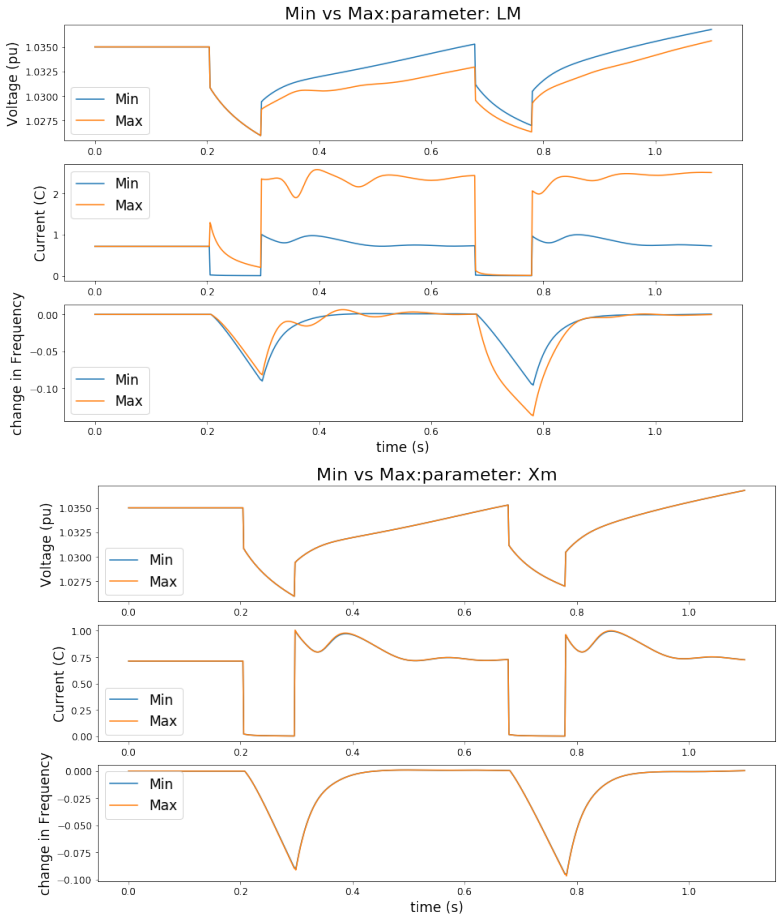


Figure 3.27: Validation graph with many to one

The next step as shown in the flowchart is to divide the data into training, validation and testing sets and then start training. Table 3.10 shows the optimum architecture found. Figure 3.28 shows the training error and proximity.

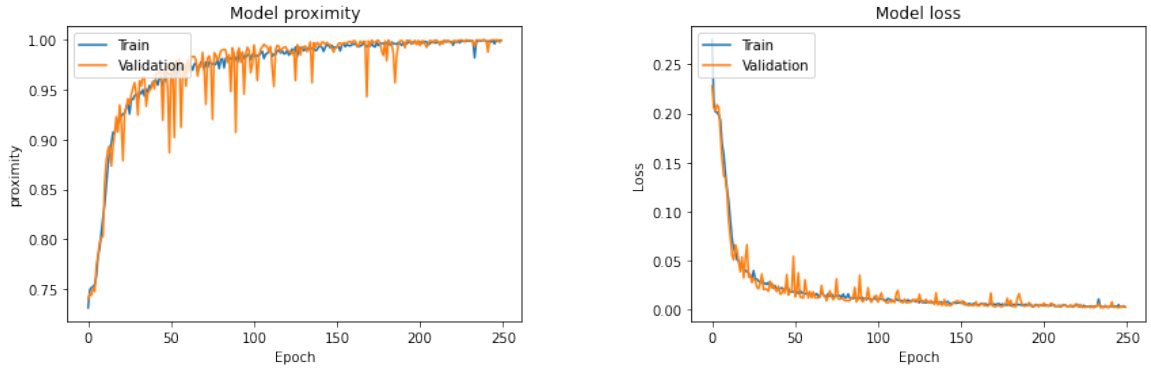
Validation Result Table 3.11(b) shows the error rate on the estimated parameters. Figure 3.29 compares the estimated response to the real response. Estimated response didn't seem to follow the original one that closely.

3.6.3 Parameter Randomization subroutine:

Until now, parameter generator algorithm as in Appendix A.2 generates combinations of parameters from finite choices. This is good for a trail example just to see the methodology working. But in real life, parameters do not be in finite choices. Fig 3.30 shows a subroutine on how randomizing the parameter generator process. Appendix A.3 includes code for the same.

Input Dimension	66*3
Output Dimension	1 per model
LSTM layers	2
Dense layers	1
Number of units	128, 128 and 32
Activation function	ReLU
Optimizer	Adam
cost function	Mean Squared Error (mse)
Epochs	1100
Batch size	450

Table 3.10: LSTM architecture for 9 parameter estimation



(a) Training proximity

(b) Training loss

Figure 3.28: LSTM model accuracy and loss graph

Algorithm testing 5 parameter estimation: Using this randomization algorithm, the dataset is generated from simulation for 5 parameter estimation. The upper and lower limit and constants are listed in Table 3.12. 2078 dataset was simulated i.e. 2078 different combinations of parameters with the corresponding dataset to represent them. We normalized the target variables and went to feature extraction using PCA. The original input dimension was 793x3: 793 data points for each voltage magnitude, current magnitude, and change in frequency. Figure 3.31 is the feature extraction graph for voltage magnitude. The way to read the graph is to find the number of components large enough to sufficiently represent the data or with a total coefficient of variance equal to 1. It was found that reducing down to 10 components was sufficient for the study. Likewise, Current magnitude and change in frequency followed a similar path. Hence, the new input dimension was reduced to 10x3 or 30 total dimensions.

The next step as shown in the flowchart is to divide the data into training, validation and testing sets and then start training. We utilized the optimum architecture found for 9 parameter estimation as in Table 3.7. Combined that with many to one approach, Figure

59245 training data set
10455 validation data set
12300 testing data set

(a) Data division for 9 parameters LSTM estimation

Parameters	Err(%) Many to one	Err(%) LSTM
LM	0.18	1.18
R_S	0.87	5.47
X_s	11.98	21.04
X_m	–	–
R_1	0.12	3.14
X_1	12.59	14.84
R_2	0.0117	3.967
X_2	0.212	3.37
H	0.089	0.384

(b) Estimated parameters error rate

Table 3.11: (a) Data division (b) Estimated parameters error rate

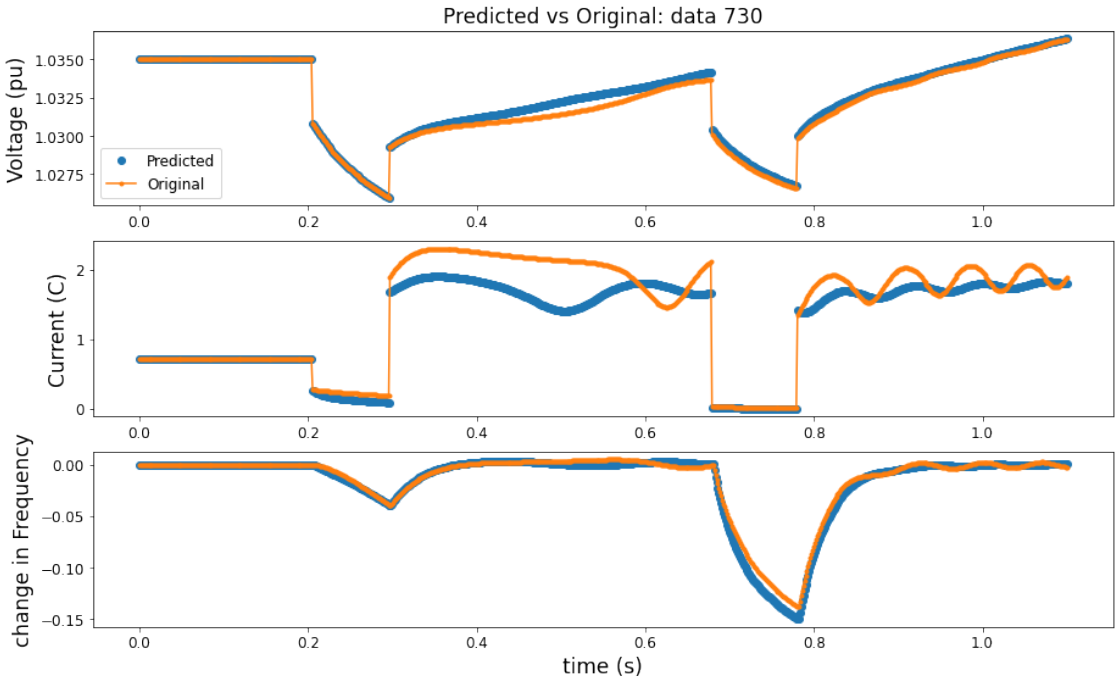


Figure 3.29: Validation graph

3.32 shows the training loss and accuracy of the model.

Validation Result Table 3.13(b) shows the error rate on the estimated parameters. All the error rates are well below 10 per cent except for Magnetizing Reactance X_m. As explained above, X_m didn't push variation into the response on its own and hence our algorithms couldn't learn it's behaviour. Figure 3.33 compares the estimated response to the real re-

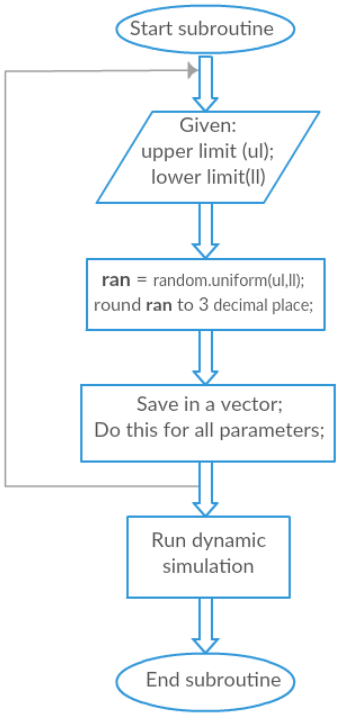


Figure 3.30: Parameter Randomization algorithm

Estimated Parameters	Constant Parameters
LM (0,100)	$X_r = 0.067$
R_s (0.012,0.024)	$R_2 = 0.009$
X_s (0.05,0.2)	$X_2 = 0.17$
X_m (3.5, 6.5)	H = 1.2
R_r (0.008,0.02)	

Table 3.12: 5 parameters simulation range

sponse. They seem quite close to each other.

3.7 Validation with a scenario

An example scenario is created where system response is recorded on a load bus with real parameters sets (Y). Let's assume that the previously estimated dynamic load parameters are noisily represented by y which for our experiment are derived by utilizing an algorithm in Appendix C.3. Utilizing the noisy data as our best option, we simulated a disturbance scenario. And, our real response is also recorded on the load bus which corresponds to the

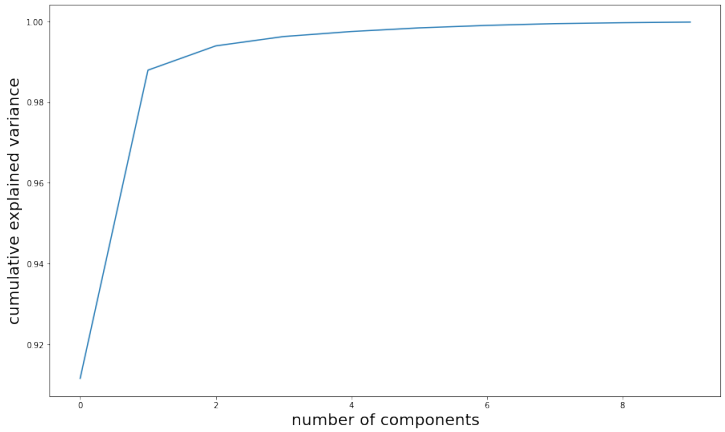
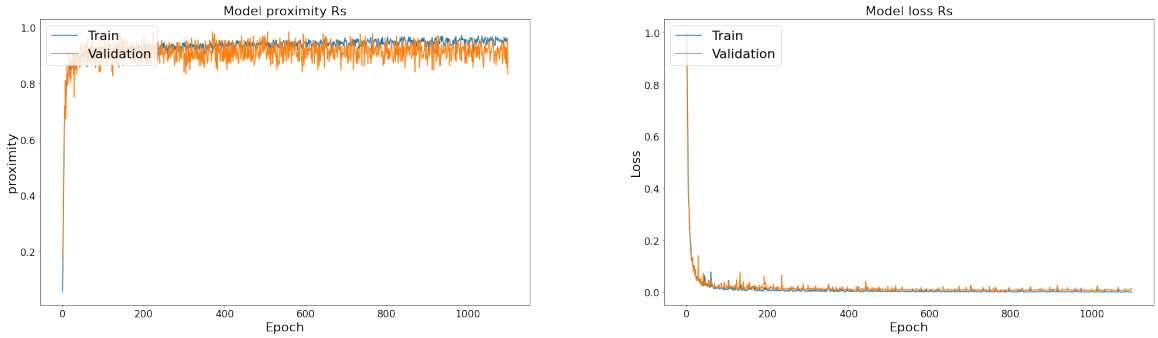


Figure 3.31: Principle Component Analysis on 5 parameter estimation data



(a) Training proximity

(b) Training loss

Figure 3.32: ANN model proximity and loss graph

1501 training data set
265 validation data set
312 testing data set

(a) Data division for 5 parameter estimation

Parameters	Error (%)
LM	0.134
R_S	1.64
X_s	0.52
X_m	12.64
R_1	0.76

(b) Estimated parameters error rate

Table 3.13: (a) Data division (b) Estimated parameters error rate

actual parameter combination.

Now, the real response recorded at the load bus is pre-processed and fed as an input to our best-learnt parameter estimation model which predicts a new parameter set ($\hat{Y}$). Putting

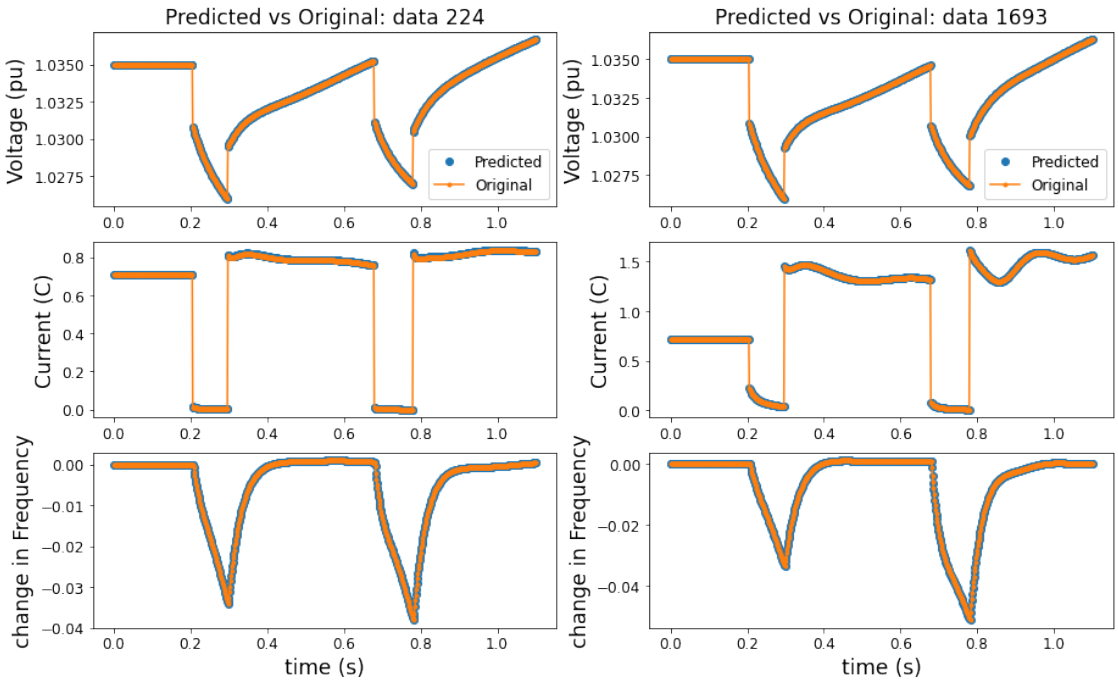


Figure 3.33: Validation graph

back these newly predicted parameter set in the simulation with the same disturbance, a predicted response is recorded. Figure 3.34 presents all three responses in comparison. As we can see, the predicted graph and the real graph are close to each other whereas the noisy response does not seem to capture the system response. This can be thought of as real PMU recording real response and feeding that response back to the model to generate predicted parameters which can closely represent the real system.

3.8 Summary and Conclusion

On comparing the results from different variations of ANN and LSTM, ANN with many to one approach came out better. Even though LSTM introduced an extra dimension of time, it can be deduced that for parameter estimation problem, the variation in dataset is more valuable than sequence information.

Even though the major part of the research dealt with data generated from a combination of narrow choices of parameters, an example replication with randomized continuous parameter generator algorithm, is tested and the methodology is verified. The only obstacle to repeating the whole experiment with randomized parameter vectors was that the time it takes to simulate the huge amount of data. To simulate 82000 combinations, it took my 8 GB RAM, Intel(R) Core(TM) i7- 7500U CPU @ 2.70 GHz system almost 8 whole days. Those

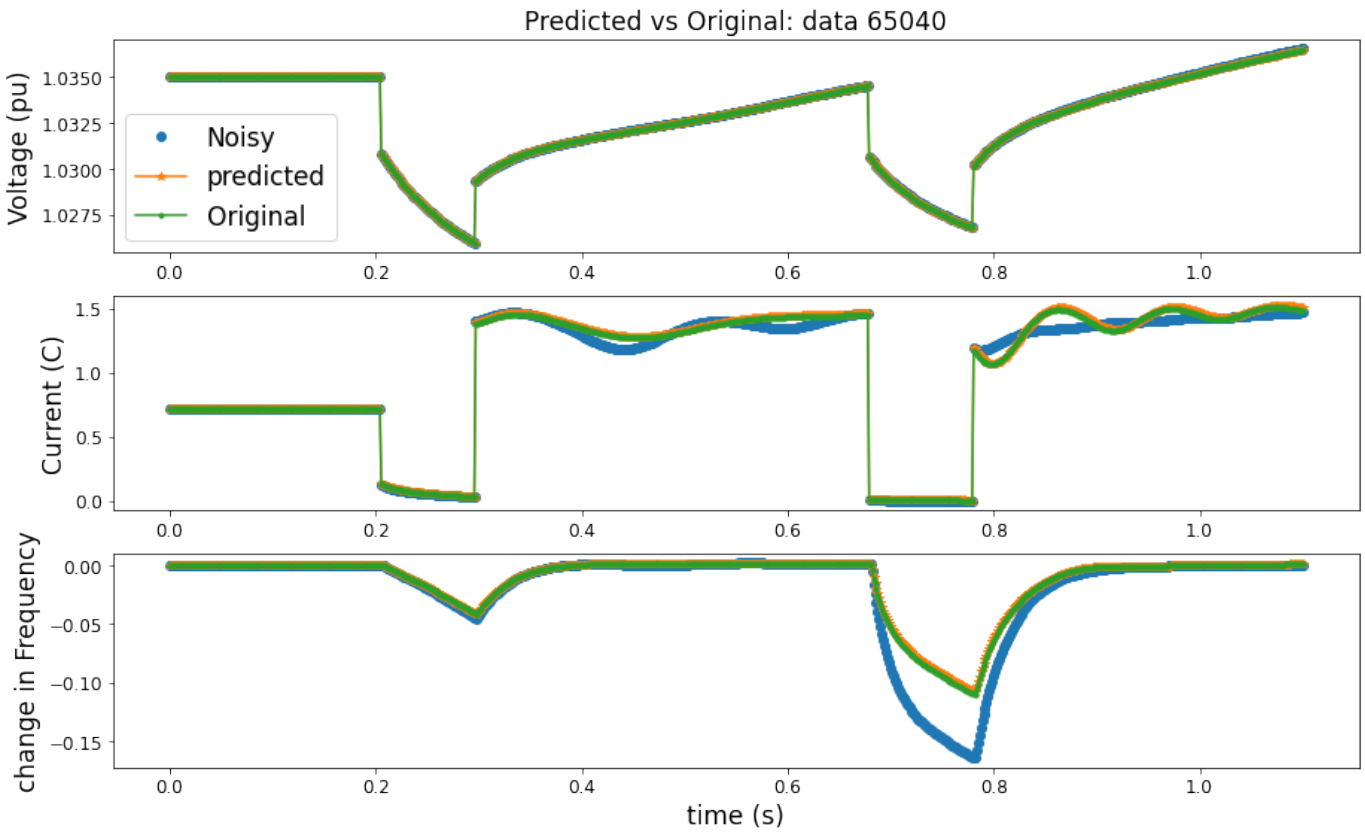


Figure 3.34: Noisy vs Predicted vs Real Response

82000 combinations were drawn from the limited choices of the parameter set. If they are too randomized, the amount of data required to successfully learn the pattern will expand drastically. With an appropriate system in hand, this methodology can be checked and confirmed.

Chapter 4

Differential equation based dynamic load

In bulk power system studies, a load is represented as a collective power need at a particular substation. In this representation, downstream of the substation is not directly presented but served as a single lump load connected to a bus. Load model at that bus is anything that matches the behaviour of that downstream load. One of the ways to model such load is to estimate parameters of the induction motor as done in the previous chapter. This chapter takes a different approach to load modelling. Here, the load model is defined as a mathematical relationship between a bus voltage, bus frequency and the active or reactive power flowing. The mathematical equation for dynamic load model, which is necessary during system stability and protection study, exhibits active or reactive power at the current instant of time as the function of active or reactive power at previous instances and voltage and frequency at current and previous instances. Differential equations are utilized to represent these expressions as in Equations 4.1- 4.2 for active and reactive power respectively.

$$\begin{cases} \Delta P(k) & = \sum_{i=1}^{n} d_i \Delta P(k-i) + \sum_{j=1}^{n+1} [e_j \Delta u(k-j+1) + g_j \Delta f(k-j+1)] \\ P(k) = P_0 + \Delta P(k) & \end{cases} \tag{4.1}$$

$$\begin{cases} \Delta Q(k) & = \sum_{i=1}^{n} d'_i \Delta Q(k-i) + \sum_{j=1}^{n+1} [e'_j \Delta u(k-j+1) + g'_j \Delta f(k-j+1)] \\ Q(k) = Q_0 + \Delta Q(k) & \end{cases} \tag{4.2}$$

Equations 4.1- 4.2 are nth order difference equations and P_0 and Q_0 are the initial powers. Considering these equations as system with current Δ powers as output, the inputs to the system can be expressed as Equations 4.3- 4.4 and the parameters to be solved can be expressed as Equations 4.5- 4.6.

$$X^p(k) = \begin{Bmatrix} \Delta P(k-1), \Delta P(k-2), \Delta P(k-3), ..., \Delta P(k-n), \\ \Delta u(k), \Delta u(k-1), \Delta u(k-2), \Delta u(k-3), ..., \Delta u(k-n), \\ \Delta f(k), \Delta f(k-1), \Delta f(k-2), \Delta f(k-3), ..., \Delta f(k-n) \\ \end{Bmatrix} \tag{4.3}$$

$$X^q(k) = \left\{ \begin{array}{c} \Delta Q(k-1), \Delta Q(k-2), \Delta Q(k-3), ..., \Delta Q(k-n), \\ \Delta u(k), \Delta u(k-1), \Delta u(k-2), \Delta u(k-3), ..., \Delta u(k-n), \\ \Delta f(k), \Delta f(k-1), \Delta f(k-2), \Delta f(k-3), ..., \Delta f(k-n) \\ \end{array} \right\} \tag{4.4}$$

$$\Theta = (d_1, d_2, d_3, ..., d_n, e_1, e_2, e_3, ..., e_{n+1}, g_1, g_2, g_3, ..., g_{n+1}) \tag{4.5}$$

$$\Theta' = (d'_1, d'_2, d'_3, ..., d'_n, e'_1, e'_2, e'_3, ..., e'_{n+1}, g'_1, g'_2, g'_3, ..., g'_{n+1}) \tag{4.6}$$

The active power in Equation 4.1 can be rephrased as a vector inner product form as in Equation 4.7.

$$\Delta P(k) =< X^p(k), \Theta > \tag{4.7}$$

The reactive power in Equation 4.2 can be rephrased as a vector inner product form as in Equation 4.8.

$$\Delta Q(k) =< X^q(k), \Theta' > \tag{4.8}$$

Our job now is to utilize these equations and do equation fitting to estimate parameter Θ and Θ'. The best method for this task is to utilize a measurement-based approach, which collects a group of directly measured voltage, frequency and flows (current or power) at a designated bus. And fit them to the above equations to estimate the parameters, the process of which is called parameter identification. These measurements can be obtained from data acquisition devices like Phasor Measurement Units. PMUs are becoming widely popular with utilities and their capabilities have also been enhanced; they can now capture data at the rate of 60 samples per second. So In this chapter, both parameter identification with Support Vector Regression(SVR) and Artificial-Intelligent(AI) based ML with LSTM is tested and compared for accurately predicting the response and their generalization ability. SVR uses the differential equation-based input to output relationship whereas neural nets like LSTM do their best work without a predefined relationship between input and output. Figure 4.1 presents a flowchart on the basic route to the first approach (with SVR). For this study, instead of using field-based measurements, we conveniently used simulated data from PSSE. Differential equation-based model is chosen as the load model structure. The process of parameter estimation and validation follows.

4.1 Support Vector Machines

Support Vector Machines make use of kernel trick to model non-linear decision boundaries [31]. Let's say we have a set of points we want to be classified, each point represented by

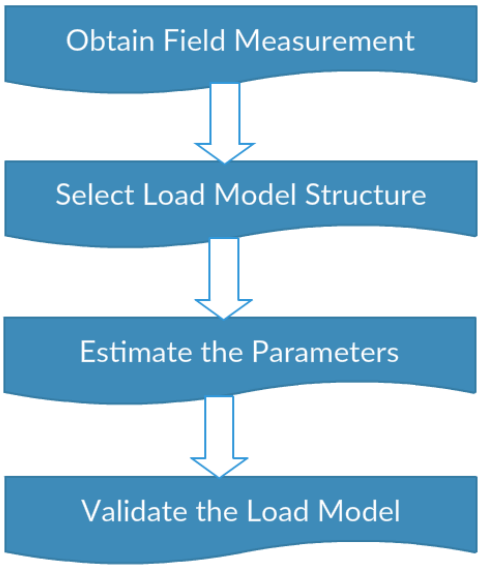


Figure 4.1: General Procedures on Load Modeling with Measurement based approach

feature vector x. But, this may be too simple for many applications. So, it is better to map it to a complex non-linear feature space Φ(x). Thus, we have transformed the feature space where each of these feature space x is mapped to a basis vector Φ(x). A decision boundary is the main separator that divides these points to respective classes. This hyperplane is represented by Equation 4.11 and in d-dimensional space, the hyperplane is a (d-1) - dimensional space.

$$x \in \mathbb{R}^D \tag{4.9}$$

$$\Phi : \mathbb{R}^D \rightarrow \mathbb{R}^M \quad \Phi(x) \in \mathbb{R}^M \tag{4.10}$$

$$H : w^T \Phi(x) + b = 0 \tag{4.11}$$

We can write the equation of the distance of a hyperplane from a point vector. Considering the case with absolute separation between data classes. There will exist at least one hyperplane which separates the training data group with 100 % accuracy. Let's consider a binary data as in Figure 4.2. Multiple hyperplanes are giving full accuracy. The goal of SVM is to find the hyperplane with maximum margin from the closest point or to maximize the minimum distance as in Equation 4.12.

$$w^* = \underset{w}{argmax}[\underset{n}{min}\ d_H(\Phi(x_n))] \tag{4.12}$$

$$y_n \in \{-1, +1\} \tag{4.13}$$

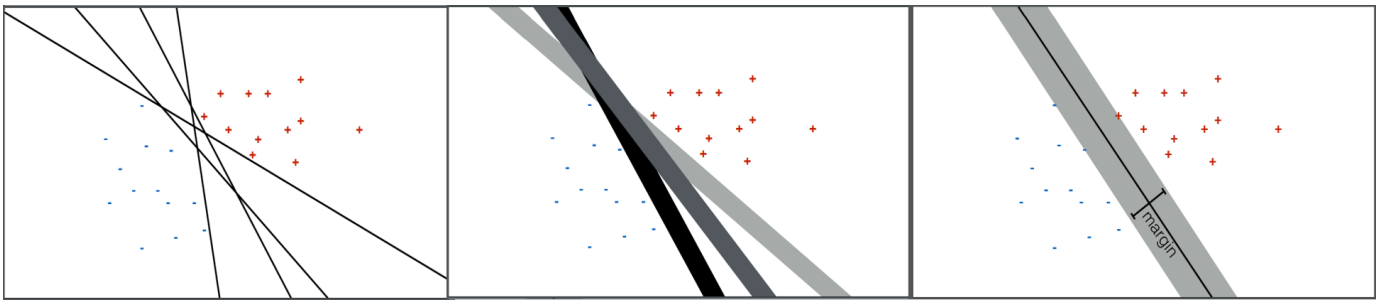


Figure 4.2: Binary class SVM Classifier

Since we have training data, our labels will be +1 for the positive group and -1 for the negative group as in Equation 4.13. From here, Equations 4.14- 4.17 are self explanatory and are towards the achievement of objective of SVM i.e. finding weights and biases.

$$y_n[w^T\Phi(x)+b] = \begin{cases} \geq 0, correct \\ < 0, incorrect \end{cases} \tag{4.14}$$

$$w^* = \underset{w}{argmax}[\underset{n}{min}\ \frac{|w^T\Phi(x_n)+b|}{\|w\|_2}] \tag{4.15}$$

$$w^* = \underset{w}{argmax}[\underset{n}{min}\ \frac{y_n[w^T\Phi(x_n)+b]}{\|w\|_2}] \quad \because perfect\ separation; \tag{4.16}$$

$$w^* = \underset{w}{argmax}\frac{1}{\|w\|_2}[[\underset{n}{min}\ y_n[w^T\Phi(x_n)+b]] \quad Distance\ of\ closest\ point\ to\ H \tag{4.17}$$

Let's normalize Equation 4.17 by making closest distance, $\underset{n}{min}\ y_n[w^T\Phi(x_n)+b] = 1$. So, Equation 4.17 is reduced to:

$$\begin{cases} w^* & = \underset{w}{argmax}\frac{1}{\|w\|_2} \\ s.t. \quad \underset{n}{min}\ y_n[w^T\Phi(x_n)_b] & = 1 \end{cases} \tag{4.18}$$

$$\begin{cases} \underset{w\in\mathbb{R}}{min}\ \frac{1}{2}w^Tw \\ s.t. \quad y_n[w^T(x_n)+b] \geq 1\ \forall\ n \end{cases} \tag{4.19}$$

Equation 4.19 is the Primal Form of SVM. This constrained optimization problem contains the original variables of the problem. A Quadratic optimization programming can be used to converge on result w and b. Since weight and bias define SVM, after their estimation,

SVM is considered solved. With X as input vector, W weight vector, <,> indicating scalar product, Φ as mapping function, estimated y for SVM can be written as Equation 4.20.

$$y(x) =< W, \Phi(X) > +b \tag{4.20}$$

$$g(X) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) K(X_i, X) + b \tag{4.21}$$

4.1.1 SVM structure

1. **Regularization parameter (C):**

$$\begin{cases} \underset{w,b,\xi \in \mathbb{R}}{min} \ \frac{1}{2} w^T w + C \sum_n \xi_n \\ s.t. \quad y_n[w^T(x_n) + b] \geq 1 - \xi_n \ \forall \ n \\ \xi_n \geq 0 \ \forall \ n \end{cases} \tag{4.22}$$

   Equation 4.22 is the new primal formula for SVM with an added term for soft margin. In the above scenario, all the data points were linearly separable into two groups. But, the real-world problems most definitely aren't. Instead of trying to make the classifier classify every single training point correctly and overfitting, the classifier is allowed to make mistake. This is done by the introduction of the slack variable (ξ) for each data point, this is called a margin classifier. But the slack should be regulated and kept as low as possible. This regulated by C, regularization parameter. If C is 0, the classifier is not penalized by slack, even large misclassification is allowed, the decision boundary is generally linear and there is a good chance of underfitting. If C is very big (∞), the classifier is harshly penalized even for a small misclassification. The decision boundary is a very complex and high chance of overfitting.

2. **kernels:** Equation 4.21 represents the SVR form after optimization. Xs are the support vectors, α are Lagrange multipliers. $K(X_i, X)$ is Kernel Function. It gives the dot product of feature space vector X_i with input vector X. There are four types of kernels. Figure 4.3 shows the types.

4.1.2 SVM Regression

Support vector machine is the name of the machine learning algorithm which can be used to perform two forms of pattern recognition: classification and regression. SVM classifier learns to classify the new data into one of the predefined groups like learning handwritten digit to classify it as either of the ten digits [0-9]. As in classification, SVM regression or in short SVR [32] is used in real number function estimation. It is characterized by the utilization of kernels, several support vectors and sparse solution. As with other supervised-learning

Kernel function types	Kernel function formula
Linear kernel	$K(X_i, X) = \langle X_i, X \rangle$
Polynomial kernel	$K(X_i, X) = (\langle X_i, X \rangle + p)^d$ $d \in N, p > 0$
Gaussian radial basis function (RBF) kernel	$K(X_i, X) = \exp\left(-\frac{\|X_i - X\|^2}{2\sigma^2}\right)$
Hyperbolic tangent kernel	$K(X_i, X) = \tanh(c\langle X_i, X \rangle + d)$ $c > 0, d > 0$

Figure 4.3: Common types of Kernels

methods, SVR learns using a symmetrical loss function, which evenly penalizes both over and under misestimates.

4.2 Test System and simulation

The test setup is very similar to Chapter 3. IEEE 118 bus system is still chosen because it is large enough system close to a real one but not too big to complicate the primary work in this research- dynamic load modelling. Fig 3.14 shows IEEE 118 bus system ready for simulation with the additional circuit at bus 11. As with the previous setup, the additional circuit here is the composite load with constant power and induction motor side by side. A new bus '2020' is placed on the other end of the step-down transformer and induction motor load is split into smaller individual loads with the exact same characteristics. This setup is to replicate a single large composite load. The original load at the bus '11' is disconnected and the equivalent composite load was built.

Simulation setup:
Power System Simulator for Engineering (PSSE) is used as a simulating environment. Initialization of the software is repeated from Chapter 3. After these steps are fulfilled without errors, dynamic simulation is run. Every simulation is of 2 seconds and the data capture rate is 720 frames per second. The recorded data for the use in this chapter is bus voltage, change of bus frequency, active and reactive power flows. ML algorithms learn from the variation in system reaction to different load parameters. Therefore, 3 phase to ground fault is placed on the neighbouring line (line 11-line 4). The simulated events are listed here with Fig 4.4 illustrating a sample plot.

1. Steady state Run from 0 to 1 seconds

2. 3 phase fault along transmission line between buses 11 and 4 at circuit '1', at t= 1

seconds

3. Run the faulted system till t = 1.2 seconds
4. Trip breaker and line 11-4 is disconnected at t=1.2 seconds
5. Run with disconnected line until t = 2 seconds

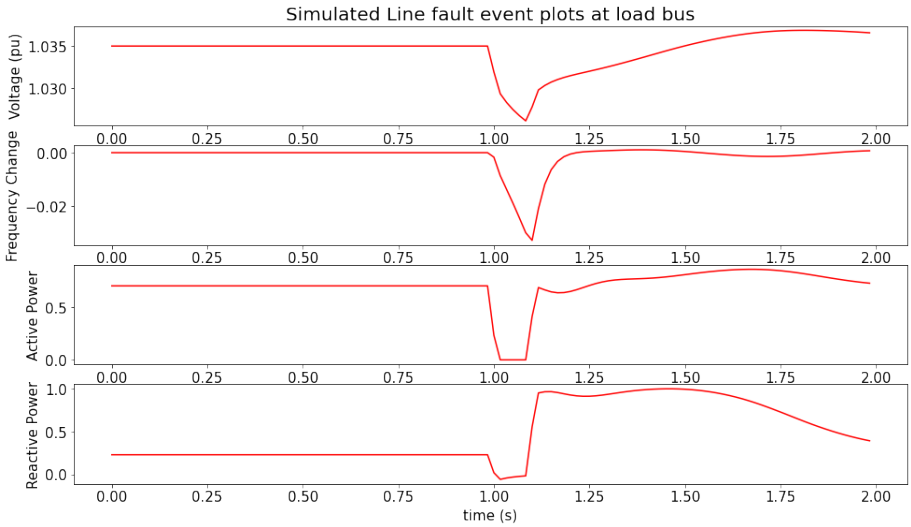


Figure 4.4: A sample Voltage, change of Frequency, Active and Reactive power at load bus

This process is repeated for each of the combinations of the parameter vector. This process is automated using python script and can be found in appendix A.4. In the end, a DAT file is generated with recorded data for each simulation run.

4.3 Equation fitting and LSTM learning

Data was first downsampled to 60 frames per second and for 2 seconds of total time, there were 120 instances per data sample. As the previous study suggested and also tested here, going beyond the 4th order of differential equation does not help to better predict the current states.

4.3.1 SVR fitting

Hyperparameter selection: There are just 2 hyperparameters to select here: types of the kernel, and Regularization. Based on the R^2 score from Figure 4.5 and 4.6, 'RBF' Kernel with C= 69 is chosen.

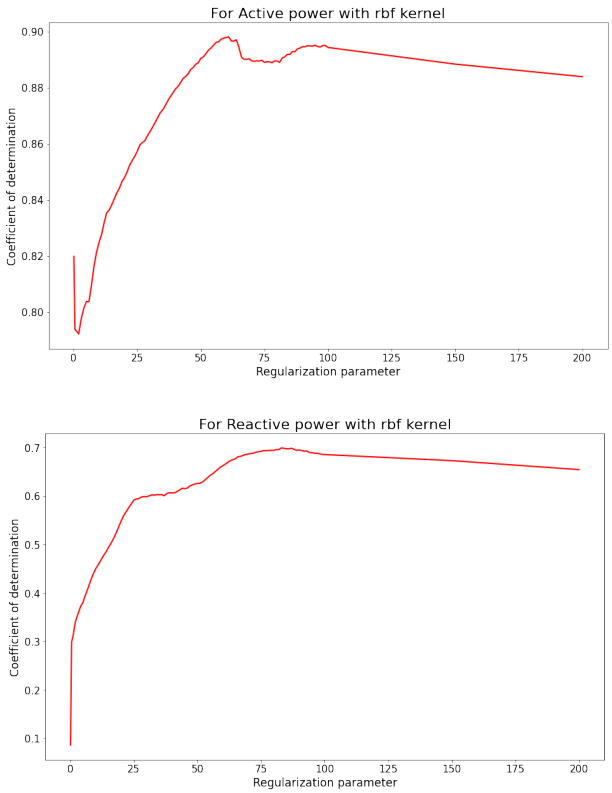


Figure 4.5: Hyperparameter selection: C with constant kernel: rbf

4.3.2 LSTM learning

The same architecture from chapter 3 is used here. The result from both SVR and LSTM are presented in Figures 4.7- 4.9. These are graphs comparing predicted active power versus actual active power. It is evident that LSTM, even though it wasn't told explicitly about the relationship between input and output variables, have much better generalization ability.

4.4 Summary and Conclusion

The advantage of the ML method is the capacity of estimated parameters to work with entirely new sets of scenario also called generalization ability. In this chapter, two different ML algorithms were experimented together. LSTM is seen to have a good scope in the power system as all the data collected will almost always have time-stamped. LSTM has good generalization ability and is surprisingly fast for a one input at a time algorithm. LSTM should do a good job with forecasting, component modelling, etc in power systems.

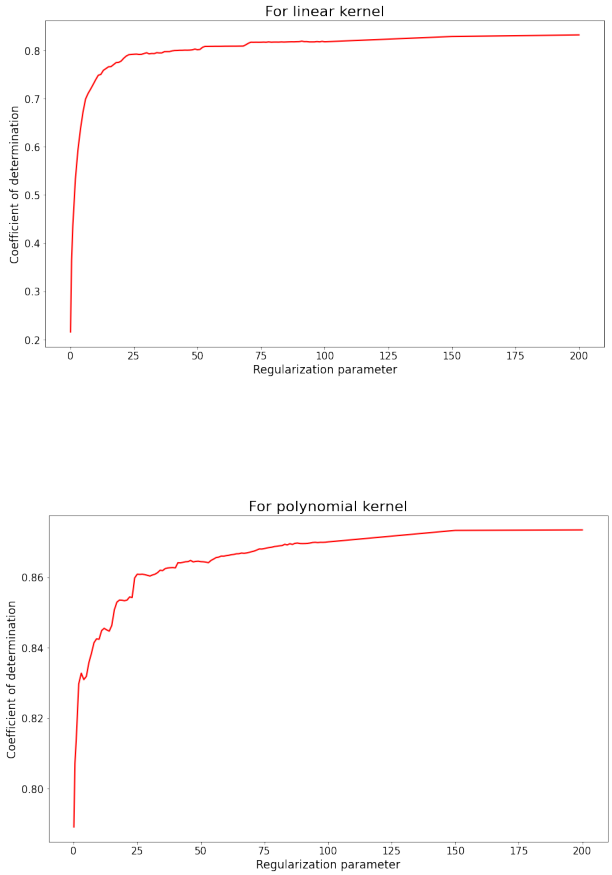


Figure 4.6: Hyperparameter selection: kernel

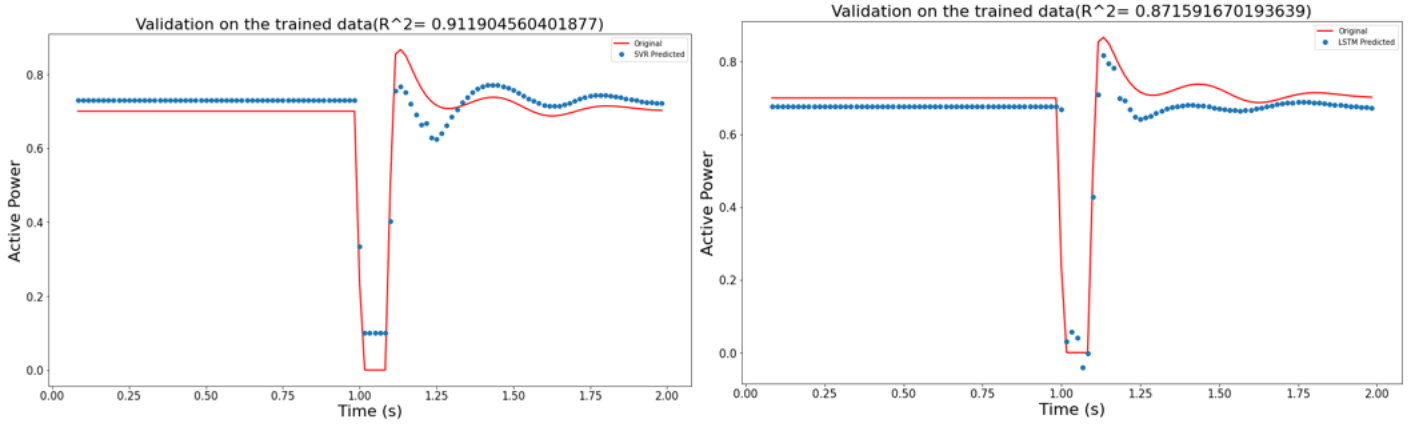


Figure 4.7: Validation on Train data: SVM (left)

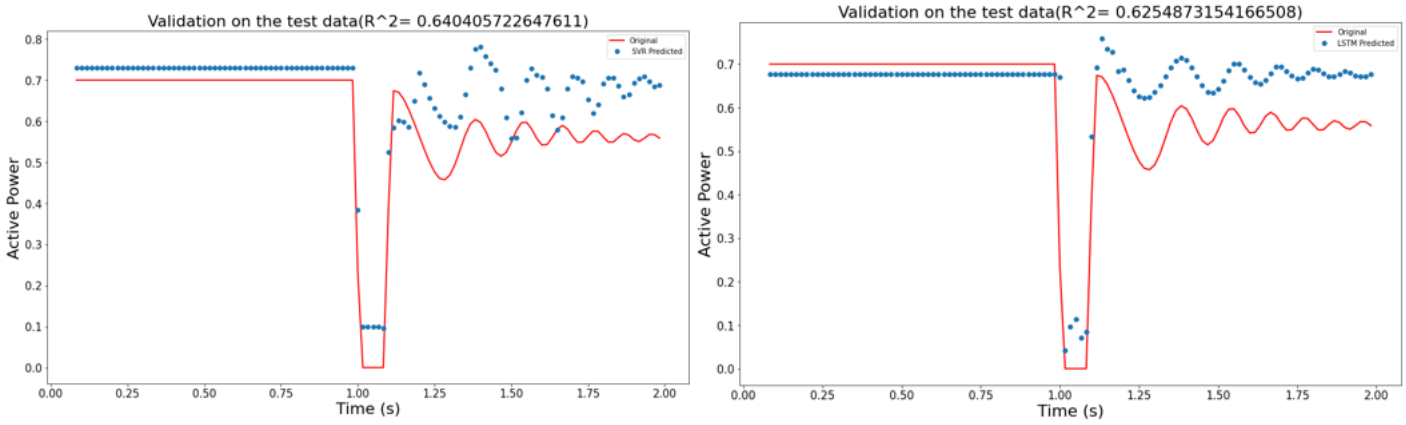


Figure 4.8: Validation on Test data: SVM (left)

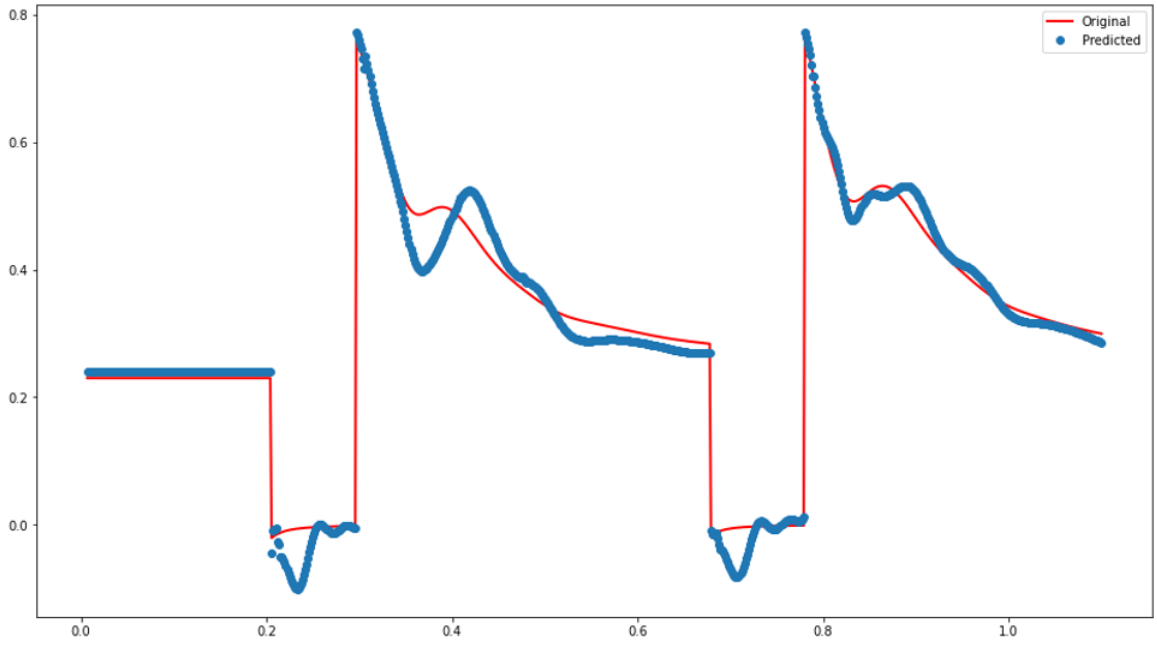


Figure 4.9: Cross Validation on chapter 3 active power data: LSTM

Chapter 5

Conclusions and Future Work

One of the largest potential areas for furthering this research would be to extrapolate dynamic load model estimation methods to include more than one load bus. Ideally, the algorithm is complete only when the parameters for every single load bus can be estimated accurately. As the number of load buses increases, the number of parameters increases, therefore requiring a much larger data set. The complexity of the neural network model will have to increase the dynamic model estimation.

Also, the future work can be using differential equation and synchrophasor data to explore different form of load model structure to achieve improved accuracy and model parameter updating for dynamic load models. Adding multiple types of faults can add complexity and need more data for training.

Bibliography

[1] Computer Methods in Power System Analysis by G.W. Stagg & a.H. El-Abiad | Matrix (Mathematics) | Determinant.

[2] M. A. Pai, P. W. Sauer, B. C. Lesieutre, and R. Adapa. Structural stability in power systems - effect of load models. *IEEE Transactions on Power Systems*, 10(2):609–615, 1995.

[3] Xiong Hua Shi, Gen Jun Chen, Ping Ju, and Dao Nong Zhang. A new generalized load model of power systems and its applications. In *APAP 2011 - Proceedings: 2011 International Conference on Advanced Power System Automation and Protection*, volume 3, pages 2268–2271, 2011.

[4] Christopher G Mertz, Jaime De La, Ree Lopez, Arun G Phadke, and Virgilio A Centeno. Utilization of Genetic Algorithms and Constrained Multivariable Function Minimization to Estimate Load Model Parameters from Disturbance Data. Technical report, 7 2013.

[5] Jian Ma, Yang Dong Zhao, and Pei Zhang. Using a support vector machine (SVM) to improve generalization ability of load model parameters. In *2009 IEEE/PES Power Systems Conference and Exposition, PSCE 2009*, 2009.

[6] NAERC; FERC. ferc-azscoutages-2012 - Arizona-Southern California Outages on September 8, 2011: Causes and Recommendations. April 2012 - GCIS.

[7] Edgar C. Portante, Stephen F. Folga, James A. Kavicky, and Leah Talaber Malone. Simulation of the September 8, 2011, San Diego blackout. In *Proceedings - Winter Simulation Conference*, volume 2015-Janua, pages 1527–1538. Institute of Electrical and Electronics Engineers Inc., 1 2015.

[8] Hema A Retty, Virgilio A Centeno, James S Thorp, Jaime De La, Ree Lopez, Dhruv Batra, and Anil Vullikanti. Load Modeling using Synchrophasor Data for Improved Contingency Analysis. Technical report, 1 2015.

[9] WECC MVWG. Composite Load Model for Dynamic Simulations, 2012.

[10] Hong Tao Ma and Badrul H. Chowdhury. Dynamic simulations of cascading failures. In *2006 38th Annual North American Power Symposium, NAPS-2006 Proceedings*, pages 619–623, 2006.

[11] Seimens PTI PSS/E. PSS/E 34.0 Model Library, 2019.

[12] William W. Price, Kim A. Uirgau, Alexander Murdoch, James V. Nitsche, Ebrahim Vaahedi, and Moe A. El-Kady. Load modeling for power flow and transient stability computer studies. *IEEE Transactions on Power Systems*, 3(1):180–187, 1988.

[13] Ma Da-Qiang and Ju Ping. A Novel Approach to Dynamic Load Modelling, 1989.

[14] Dmitry Bliznyuk, Alexander Berdin, and Ilya Romanov. PMU data analysis for load characteristics estimation. In *2016 57th International Scientific Conference on Power and Electrical Engineering of Riga Technical University, RTUCON 2016*. Institute of Electrical and Electronics Engineers Inc., 11 2016.

[15] Byoung Kon Choi and Hsiao Dong Chiang. On the local identifiability of load model parameters in measurement-based approach. *Journal of Electrical Engineering and Technology*, 4(2):149–158, 2009.

[16] W. H. Kersting. Distribution System Modeling and Analysis. *IEEE Power and Energy Magazine*, 11(3):106–108, 5 2013.

[17] WECC Dynamic Composite Load Model (CMPLDW) Specifications. Technical report, 2015.

[18] Luis Rodriguez-Garcia, Sandra Perez-Londono, and Juan Mora-Florez. Particle swarm optimization applied in power system measurement-based load modeling. In *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, pages 2368–2375, 2013.

[19] Zhang Pei and Bai Hua. Derivation of load model parameters using improved genetic algorithm. In *3rd International Conference on Deregulation and Restructuring and Power Technologies, DRPT 2008*, pages 970–977, 2008.

[20] Jinyu Wen, Shaorong Wang, Shijie Cheng, Q. H. Wu, and D. W. Shimmin. Measurement based power system load modeling using a population diversity genetic algorithm. In *POWERCON 1998 - 1998 International Conference on Power System Technology, Proceedings*, volume 1, pages 771–775. Institute of Electrical and Electronics Engineers Inc., 1998.

[21] Xiaodong Liang. Linearization approach for modeling power electronics devices in power systems. In *IEEE Journal of Emerging and Selected Topics in Power Electronics*, volume 2, pages 1003–1012. Institute of Electrical and Electronics Engineers Inc., 12 2014.

[22] Xiaodong Liang, Yi He, Massimo Mitolo, and Weixing Li. Support vector machine based dynamic load model using synchrophasor data. In *Conference Record - Industrial and Commercial Power Systems Technical Conference*, volume 2018-May, pages 1–11. Institute of Electrical and Electronics Engineers Inc., 5 2018.

[23] W. W. Price, C. W. Taylor, G. J. Rogers, K. Srinivasan, C. Concordia, M. K. Pal, Bess, P. Kundur, B. L. Agrawal, J. F. Luini, E. Vaahedi, and B. K. Johnson. Standard Load Models for Power Flow and Dynamic Performance Simulation. *IEEE Transactions on Power Systems*, 10(3):1302–1313, 1995.

[24] Anon. Load representation for dynamic performance analysis. *IEEE Transactions on Power Systems*, 8(2):472–482, 5 1993.

[25] Pinky Sodhi, Naman Awasthi, and Vishal Sharma. Introduction to Machine Learning and Its Basic Application in Python. *SSRN Electronic Journal*, 2 2019.

[26] Vidushi Sharma, Sachin Rai, and Anurag Dev. A Comprehensive Study of Artificial Neural Networks. Technical Report 10, 2012.

[27] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to Construct Deep Recurrent Neural Networks. 12 2013.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[29] Felix Gers. Long Short-Term Memory in Recurrent Neural Networks. Technical report, 2001.

[30] Urminder Singh, Sucheta Chauhan, A. Krishnamachari, and Lovekesh Vig. Ensemble of deep long short term memory networks for labelling origin of replication sequences. In *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015*. Institute of Electrical and Electronics Engineers Inc., 12 2015.

[31] Yongli Zhang. Support vector machine classification algorithm and its application. In *Communications in Computer and Information Science*, volume 308 CCIS, pages 179–186. Springer, Berlin, Heidelberg, 2012.

[32] Harris Drucker · , Chris J C Burges, Linda Kaufman, Alex Smola · · , and Vladimir Vapoik. Support Vector Regression Machines. Technical report.

Appendices

Appendix A

Parameters Generator and Simulation

A.1 CIM5BL Induction motor parameters

CONs	Value	Description
J		R_A
J+1		X_A
J+2		$X_m > 0$
J+3		$R_1 > 0$
J+4		$X_1 > 0$
J+5		R_2 (0 for single cage)[a]
J+6		X_2 (0 for single cage)
J+7		$E_1 \geq 0$
J+8		$S(E_1)$
J+9		E_2
J+10		$S(E_2)$
J+11		MBASE[b]
J+12		PMULT
J+13		H (inertia, per unit motor base)
J+14		V_l (pu)[c]
J+15		T_l (cycles)[d]
J+16		T_B (cycles)
J+17		D (load damping factor)
J+18		T_{nom}, Load torque at 1 pu speed (used for motor starting only) (≥ 0)

[a]To model single cage motor: set $R_2 = X_2 = 0$.

[b]When MBASE = 0, motor MVA base = PMULT x MW load. When MBASE > 0, motor MVA base = MBASE.

[c]V_l is the per unit voltage level below which the relay to trip the motor will begin timing. To disable relay, set $V_l = 0$.

[d]T_l is the time in cycles for which the voltage must remain below the threshold for the relay to trip. T_B is the breaker delay time cycles.

A.2 Parameter Vector Generator(fixed sets)

```
# Parameter Vector Generator
# With upper and lower limit and certain increments.


import numpy
import os
from functools import reduce


lm = list(range(10,101,10))
rs = [x/1000 for x in range(24,11,-6)]
xs = [x/1000 for x in range(200,49,-75)]
xm = [x/10 for x in range(35,66,15)]
rr = [x/1000 for x in range(20,7,-4)]
xr = [x/1000 for x in range(200,49,-75)]
r2 = [x/1000 for x in range(5,16,5)]
x2 = [x/100 for x in range(10,21,5)]
H =[x/10 for x in range(4,13,4)]


cparam1 = [1, 0.06, 1.2, 0.6, 0, 1.2]
cparam2 = [0, 0, 0, 1, 0]
combination = numpy.array(numpy.meshgrid(lm, rs, xs, xm, rr, xr, r2, x2, H)).T.reshape(-
1,9).tolist()
filename ='parameters.txt'
with open(filename, 'w') as f1:
for _ in range(len(combination)):
f1.write(' '.join(str(i) for i in reduce(lambda acc_l, sl: acc_l.extend(sl) or acc_l, [combina-
tion[_][:8], cparam1, [combination[_][8]], cparam2])))
f1.write('\n')
f1.close()
```

A.3 Parameter Vector Generator(random)

```
# Parameter Vector Generator
# with lower and upperlimit and randomization


import numpy
```

```
import os
from functools import reduce
import random


random.seed(7)


def parameterRandomization(ll,ul, Float = 3):
return round(random.uniform(ll,ul), Float)


lm = 0,100
rs = 0.012,0.024
xs = 0.05,0.2
xm = 3.5,6.5
rr = 0.008, 0.02
xr = 0.067
r2 = 0.009
x2 = 0.17
H = 1.2


Vectors = [lm, rs, xs, xm, rr, xr, r2, x2, H]
cparam1 = [1, 0.06, 1.2, 0.6, 0, 1.2]
cparam2 = [0, 0, 0, 1, 0]


combination = []
for i in range(89):
temp =[]
for j in range(len(Vectors)):
try:
len(Vectors[j])
temp.append(parameterRandomization(Vectors[j][0],Vectors[j][1]))
except:
temp.append(Vectors[j])
combination.append(temp)
filename = "parameters.txt"
with open(filename, 'w') as f1:
for _ in range(len(combination)):
f1.write(' '.join(str(i) for i in reduce(lambda acc_l, sl: acc_l.extend(sl) or acc_l, [combina-
tion[_][:8], cparam1, [combination[_][8]], cparam2])))
f1.write('\n')
f1.close()
```

A.4 PSSE automation

```
# Collect Fault Data for Complex Load - Ind Motor and ZIP
#transformer at load bus 11- modelled at LV
# Sanij Gyawali inspired from Hema Retty
#created on 05/15/2020


from __future__ import with_statement
from contextlib import contextmanager
from copy import deepcopy
import os,sys
import numpy as np
import itertools


os_path_PSSE =r"C:\ProgramFiles(x86)\PTI\PSSE34\PSSBIN"
sys_path_PSSE = r"C:\Program Files (x86)\PTI\PSSE34\PSSPY37"
sys.path.append(sys_path_PSSE) #This is where python27 is
os.environ['PATH'] += ';' + os_path_PSSE #This is where the psse is installed

import psse34
import psspy
psspy.psseinit(2000)
_i = psspy.getdefaultint()
_f = psspy.getdefaultreal()
_s = psspy.getdefaultchar()


import redirect
redirect.psse2py()
import dyntools


#----------------
# Read parameters from the file- [%LM, %R1, %x1, %Xm, %R2, %X2]
f= open("validateparam.txt")
param3 = f.readlines()
param3 = np.loadtxt(param3)
param3 = np.array(param3)
f.close()
```

```
paramLength = len(param3)


#Start simulation
nLM = 50 # number of Large motors/ total numbers of small load models
# I think this is actually finding the no. of load models, let's see
for pid in range(1):
print(param3[pid][0])
if param3[pid][0]> 0 and param3[pid][0]<100:
nLDs = nLM +1
elif param3[pid][0] == 0:
nLDs = 1
elif param3[pid][0] == 100:
nLDs = nLM
LID = []


#PSSE saved case
CASE = r"E:\Sanij\with118\IEEE 118 bus\ieee118bus_PSSE.sav"
psspy.case(CASE)
psspy.fnsl() #full static load flow


#Variables
loadbus = 11
tbus = 2020 #Transformer bus at load
refbus = 69 #Reference bus
bfault = [3,5] #[11, 4] #Fault Branch
param = [[100.0, 0.0, 0.0, 100.0], [0.0, 0.0, 0.0, 0.0], [100.0, 0.0, 100.0, 0.0], [0.0, 100.0, 0.0,
100.0]]


#Add Transformer at Load Bus
ierr = psspy.bus_data_2(tbus, name = "tbus")
ierr, psspy.realaro = psspy.two_winding_data_3(loadbus, tbus, '1', [], [0.0, 0.037])


#Split Load into two IDS
ierr, cmpval = psspy.loddt2(loadbus, "BL", "MVA", "ACT")


ierr = psspy.load_data_3(loadbus, "BL", intgar1= 0)
Pinit = cmpval.real #Xfmr ratio
Qinit = cmpval.imag
Pind = Pinit*param3[pid][0]/100
```

```
Qind = Qinit*param3[pid][0]/100
Pzip = Pinit - Pind
Qzip = Qinit - Qind


tempZIP = np.array([param[0][1]/100, param[0][0]/100, (100-param[0][0]-param[0][1])/100,
param[0][3]/100, param[0][2]/100, (100 -param[0][2]-param[0][3])/100])


tempmach = list(param3[pid][1:20])
if param3[pid][0]>0:
for iLM in range(nLM):
LID.append('%d' %(iLM+1))
ierrl1 = psspy.load_data_3(tbus, LID[iLM], [], [Pind/nLM, Qind/nLM, 0, 0, 0, 0])
if param3[pid][0]<100:
ierrl2 = psspy.load_data_3(tbus, 'BL', [], [Pzip, Qzip, 0, 0, 0, 0])
psspy.fnsl()


#Create lists of branches and 2 winding transformers and load buses
ierr3, totbrn = psspy.abrncount(-1, 1, 1, 1)
ierr4, brnlist = psspy.abrnint(-1, 1, 1, 1, 1, ['FROMNUMBER', 'TONUMBER'])
ierr, totgen = psspy.agenbuscount(-1, 1)
ierr2, genlist = psspy.agenbusint(-1, 1, 'NUMBER')
ierr5, loadlist = psspy.alodbusint(-1, 1, 'NUMBER')
loadlist = loadlist[0]
ierr6, otherloads = psspy.alodbusint(-1, 1, 'NUMBER')
otherloads = otherloads[0]
if tbus in otherloads:
otherloads.remove(tbus)
loadnum = len(loadlist)
othernum = len(otherloads)


outfile = r'E:\Sanij\with118\code and result\data\data18bus_' + 'load' + loadbus.___str___()
+ '_lines' + bfault[0].___str___() + '-' + bfault[1].___str___() + '_' + param3[pid][0].___str___()
+ '_' + param3[pid][1].___str___() + '_' + param3[pid][2].___str___() + '_' + param3[pid][3].___str___()
+ '_' + param3[pid][4].___str___() + '_' + param3[pid][5].___str___() + '_' + param3[pid][6].___str___()
+ '_' + param3[pid][7].___str___() + '_' + param3[pid][14].___str___() +'.out

datfile = r'E:\Sanij\with118\code and result\data\data18bus_' + 'load' + loadbus.___str___()
+ '_lines' + bfault[0].___str___() + '-' + bfault[1].___str___() + '_' + param3[pid][0].___str___()
+ '_' + param3[pid][1].___str___() + '_' + param3[pid][2].___str___() + '_' + param3[pid][3].___str___()
+ '_' + param3[pid][4].___str___() + '_' + param3[pid][5].___str___() + '_' + param3[pid][6].___str___()
+ '_' + param3[pid][7].___str___() + '_' + param3[pid][14].___str___() +'.dat'
```

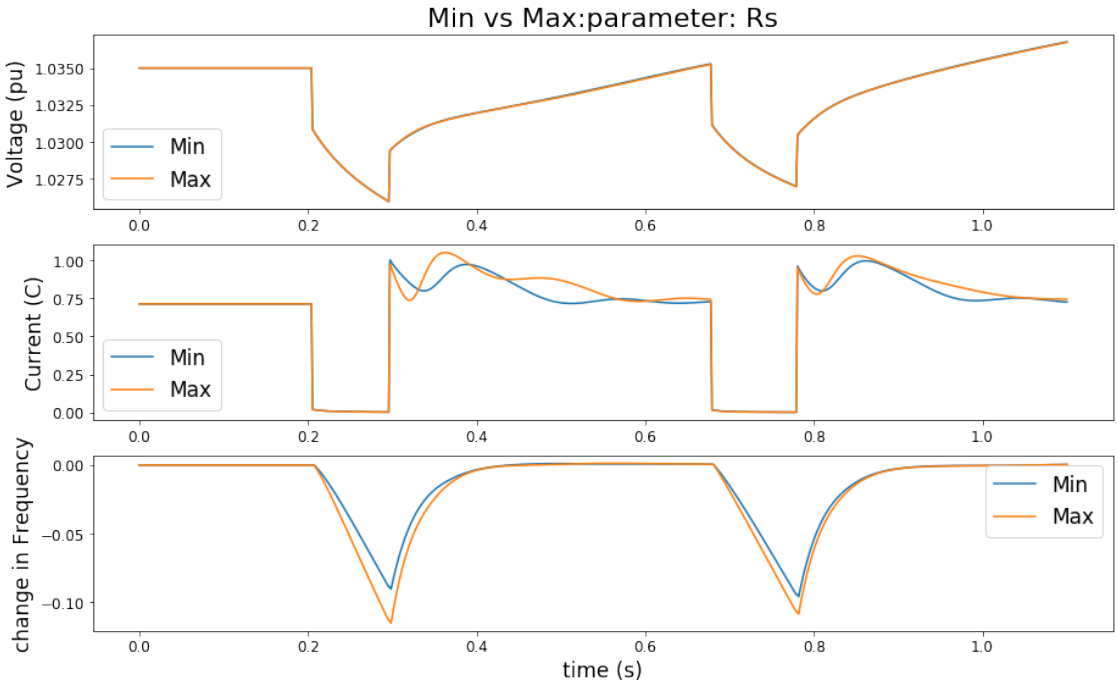
```
#Create SIDs (Subsystem ID)
psspy.bsys(1, 0, [0, 0], 0, [], loadnum, loadlist)
psspy.bsys(2, 0, [0, 0], 0, [], 1, tbus)
psspy.bsys(3, 0, [0, 0], 0, [], 2, [tbus, refbus])
psspy.bsys(4, 0, [0, 0], 0, [], othernum, otherloads)

#PowerFlow
psspy.fnsl()

#Dynamic Simulation Setup
CASE2 = r"E:\Sanij\with118\IEEE 118 bus\ieee118.dyr"
ierrdyn = psspy.dyre_new([1, 1, 1, 1], CASE2)
psspy.dynamics_solution_param_2(1000)
psspy.dynamics_solution_param_2(realar3=0.00138889)
psspy.chsb(3, 0, [1, -1, -1, 1, 14, 0]) #Bus Voltage Mag and Angle; Choose output variables and channel index
psspy.chsb(2, 0, [5, -1, -1, 1, 12, 0]) #Bus frequency
psspy.chsb(2, 0, [6, -1, -1, 1, 25, 0]) #Load Power - P
psspy.chsb(2, 0, [6+nLDs, -1, -1, 1, 26, 0]) #Load Power - Q

#Add Load Model(s)
if param3[pid][0]<100: psspy.add_load_model(tbus, 'BL', 0, 1, 'IEELBL', 0, [], [], 14, [tempZIP[0], tempZIP[1], tempZIP[2], tempZIP[3], tempZIP[4], tempZIP[5], 0, 0, 2, 1, 0, 2, 1, 0])
if param3[pid][0]>0: for iLM in range(nLM): psspy.add_load_model(tbus, LID[iLM], 0, 1, 'CIM5BL', 1, 1, [] , 19, tempmach[:])

#Convert Load at all Load Buses
psspy.conl(1, 1, 1, [0, 0])
ierr, rlods = psspy.conl(1, 1, 2, [0, 0], param[0]) #change 1st param to 2 and 2nd param to 0 for load bus or both params to 1 for all buses
psspy.conl(1, 1, 3)

#PSSE Dyanmic Simulation Pre-Initialization
psspy.cong()
psspy.ordr(1)
psspy.fact()
psspy.tysl(0)
```

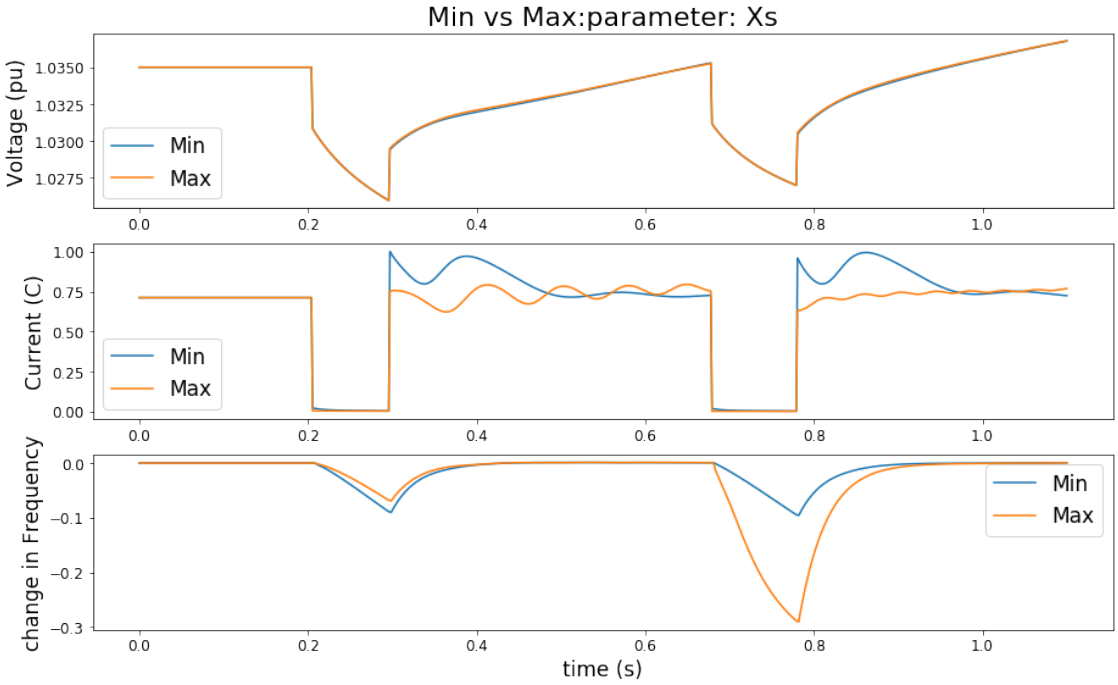
```
ierr1 = psspy.strt(0, outfile) #Check if load model parameters are correct
try:
ierr2 = psspy.okstrt()
except psspy.OkstrtError:
ierr2 = 1
if (ierr1+ierr2)==0:
try:
psspy.run(0, 0)
psspy.run(0, 0.2)
psspy.dist_branch_fault(bfault[0], bfault[1], '1')
psspy.run(0, 0.29)
psspy.dist_branch_trip(bfault[0], bfault[1], '1')
psspy.run(0, 0.67)
psspy.dist_branch_close(bfault[0], bfault[1], '1')
psspy.run(0, 0.77)
psspy.dist_branch_trip(bfault[0], bfault[1], '1')
psspy.run(0, 1.09)


#Convert Output (.out) file to MATLAB (.dat) input.
chan = dyntools.CHNF(outfile)
data = chan.get_data()
psspy.close_powerflow()


# Sum Load Power
sumP = []
sumQ = []
for subL in range(nLDs):
sumP = np.sum([data[2][6+subL], sumP], axis=0) #sumP and sumQ are still vectors
sumQ = np.sum([data[2][6+nLDs+subL], sumQ], axis=0)


# Calculate Current from P, Q and V
curm = []
cura = []
langle = []
for pitem,qitem,vmitem,vaitem,raitem in zip(sumP,sumQ,data[2][1],data[2][2],data[2][4]):
# for pitem,qitem,vmitem,vaitem,raitem in itertools.izip(data[2][6],data[2][7],data[2][1],data[2][2],data[2][4]
loadangle = vaitem - raitem
langle.append(loadangle)
current = (pitem + 1j*qitem)/(vmitem*np.exp(1j*np.radians(loadangle)))
```

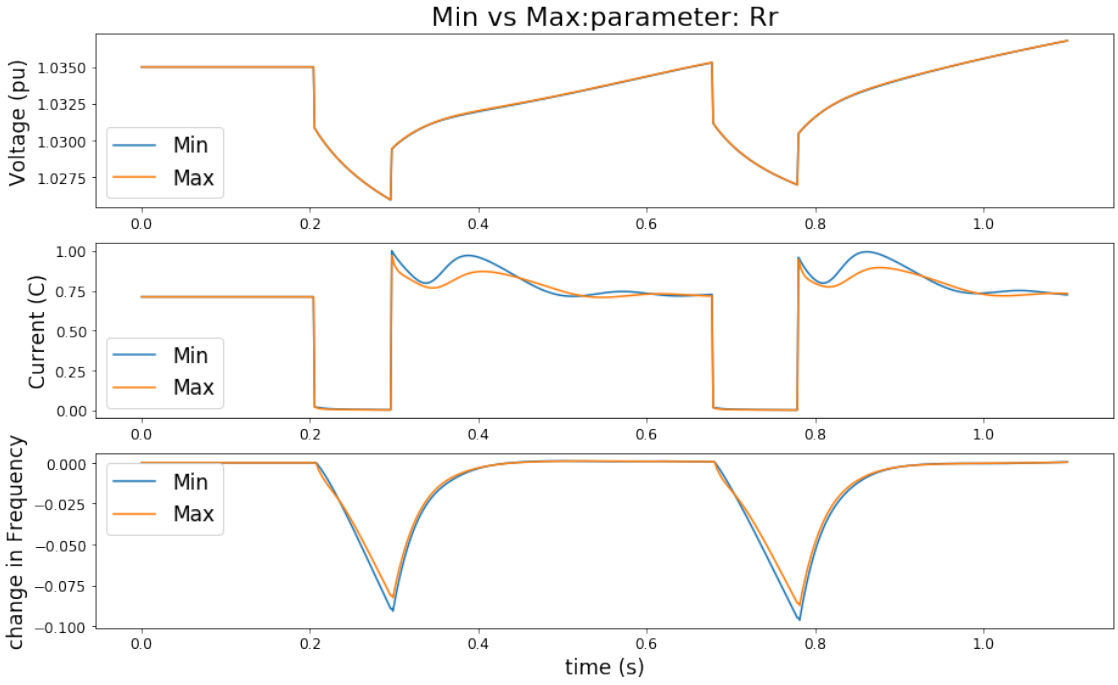
```
currentm = np.abs(current)
currenta = -1*np.angle(current, deg=True) #Complex Conjugate in degrees
curm.append(currentm)
cura.append(currenta)


fid = open(datfile, "w")
fid.writelines("%s\n" % item for item in data[2][1]) #Load volt mag
fid.write('EOF\n')
fid.writelines("%s\n" % item for item in data[2][2]) #Load volt ang
fid.write('EOF\n')
fid.writelines("%s\n" % item for item in data[2][3]) #Ref volt mag
fid.write('EOF\n')
fid.writelines("%s\n" % item for item in data[2][4]) #Ref volt ang
fid.write('EOF\n')
fid.writelines("%s\n" % item for item in data[2][5]) #Load Freq
fid.write('EOF\n')
fid.writelines("%s\n" % item for item in curm) #Load Current mag
fid.write('EOF\n')
fid.writelines("%s\n" % item for item in cura) #Load Current Ang
fid.write('EOF\n')


#Probably should record power somewhere here fid.writelines("%s\n" % item for item in
sumP) #Total active power at load bus
fid.write('EOF\n')
fid.writelines("%s\n" % item for item in sumQ) #Total reactive power at load bus
fid.close()


except (RuntimeError, TypeError, NameError) as e:
print(e)
print('Error on line ',sys.exc_info()[-1].tb_lineno)
psspy.close_powerflow()
else:
psspy.close_powerflow()
else:
print("Start Error")
psspy.close_powerflow()
os.remove(outfile)
print("END OF PROGRAM")
```

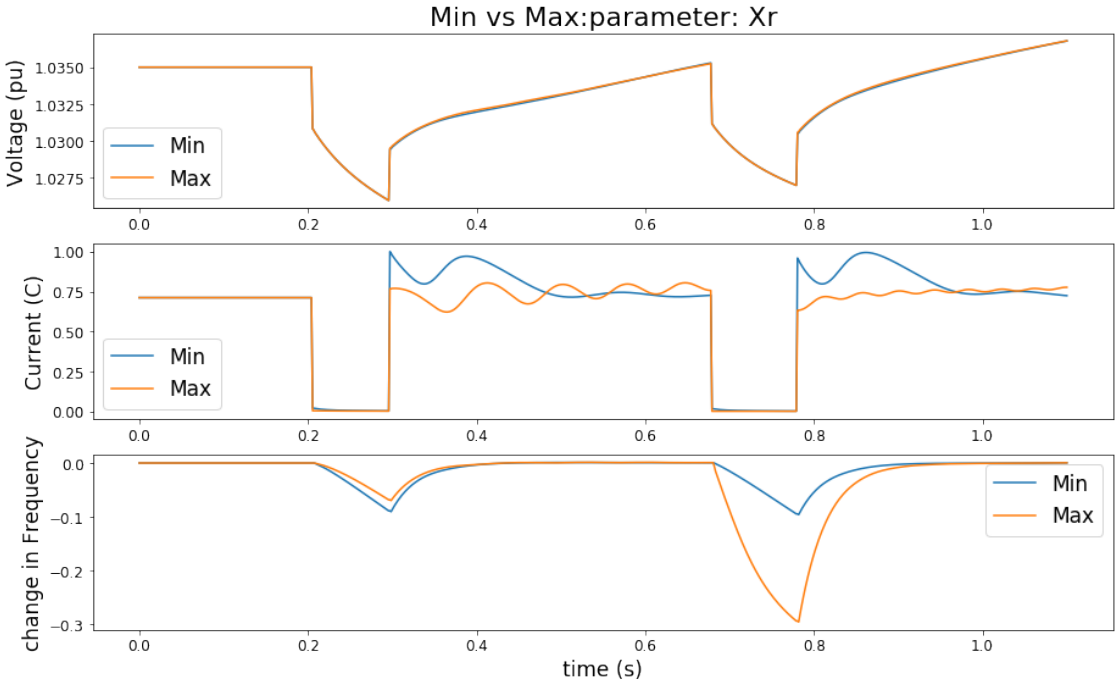
Appendix B

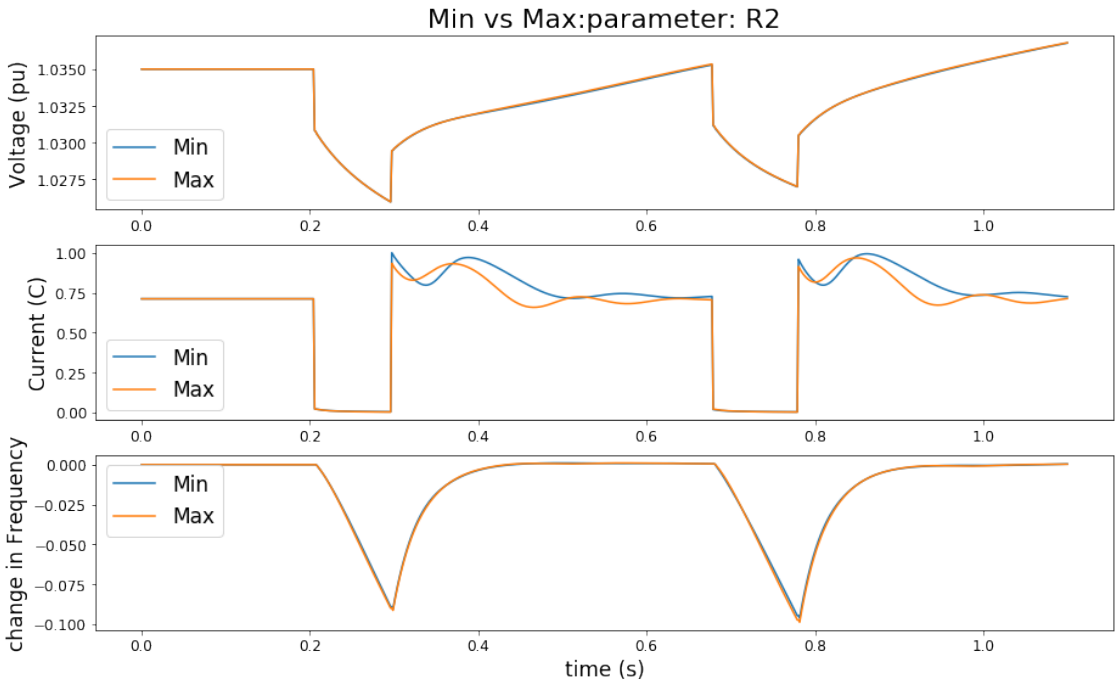
Min Max Investigation




Min vs Max:parameter: Xs
Voltage (pu)
1.0350
1.0325
1.0300
1.0275
Min
Max
0.0
0.2
0.4
0.6
0.8
1.0
Current (C)
1.00
0.75
0.50
0.25
0.00
Min
Max
change in Frequency
0.0
−0.1
−0.2
−0.3
Min
Max
time (s)


Min vs Max:parameter: Rr
Voltage (pu)
1.0350
1.0325
1.0300
1.0275
Min
Max
0.0
0.2
0.4
0.6
0.8
1.0
Current (C)
1.00
0.75
0.50
0.25
0.00
Min
Max
change in Frequency
0.000
−0.025
−0.050
−0.075
−0.100
Min
Max
time (s)


Min vs Max:parameter: Xr
Voltage (pu)
1.0350
1.0325
1.0300
1.0275
Min
Max
0.0
0.2
0.4
0.6
0.8
1.0
Current (C)
1.00
0.75
0.50
0.25
0.00
Min
Max
change in Frequency
0.0
−0.1
−0.2
−0.3
Min
Max
time (s)


Min vs Max:parameter: R2
Voltage (pu)
1.0350
1.0325
1.0300
1.0275
Min
Max
0.0
0.2
0.4
0.6
0.8
1.0
Current (C)
1.00
0.75
0.50
0.25
0.00
Min
Max
change in Frequency
0.000
−0.025
−0.050
−0.075
−0.100
Min
Max
time (s)

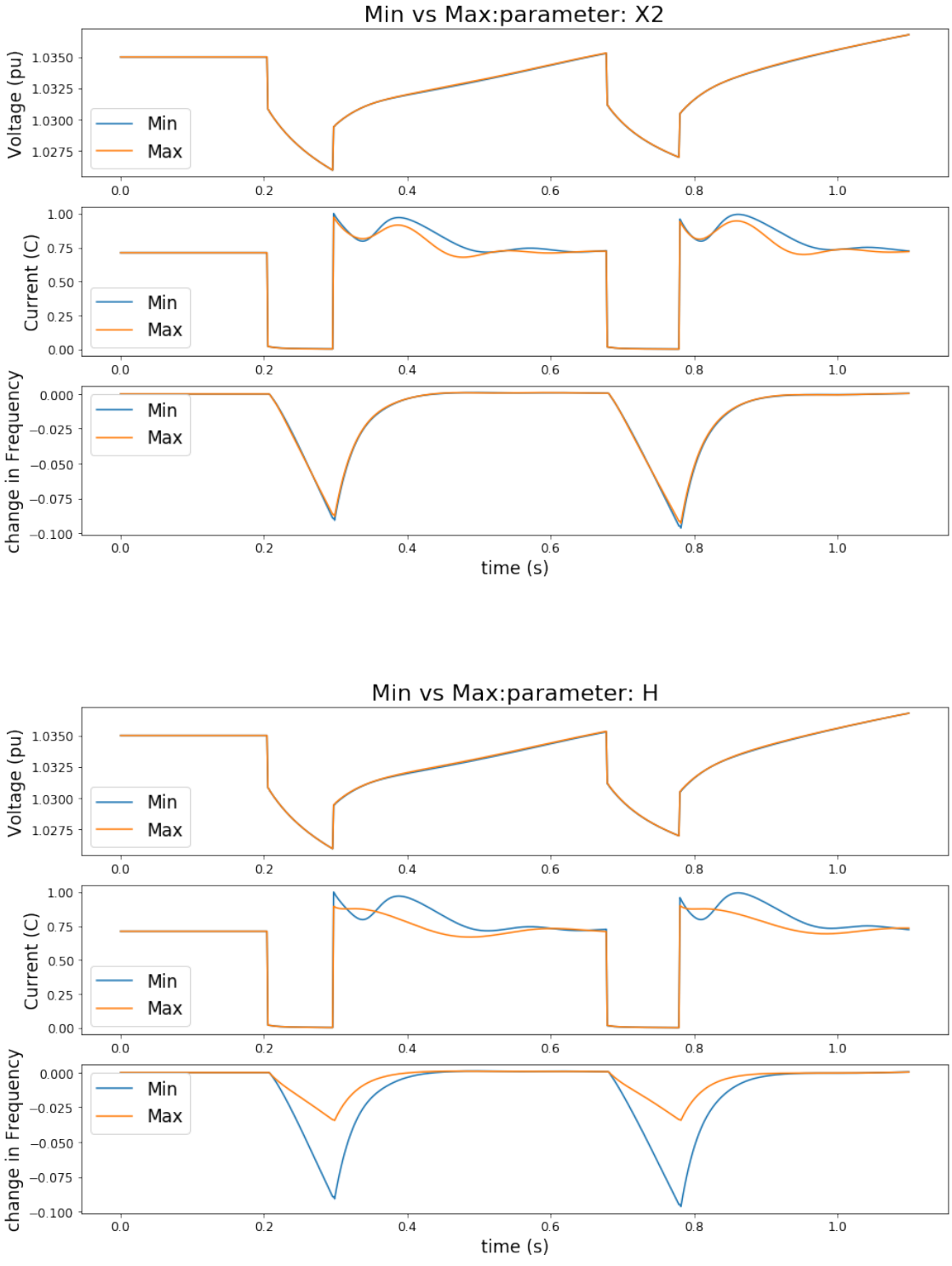


Figure B.1: Minimum vs Maximum component system response

Appendix C

Machine Learning

C.1 Artificial Neural Network algorithm

```
import os
import re
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pickle
#Loading saved data
Dependent=pd.read_pickle('/content/drive/My Drive/Thesis/9paramonemodelperparameter/Target9.pkl
infile = open('/content/drive/My Drive/Thesis/9paramonemodelperparameter/Independent9.pkl','rb')
Independent = pickle.load(infile)


period = 793
Voltage = Independent.iloc[:,:period]
Current = Independent.iloc[:,period*2:period*3].T.reset_index(drop = True).T
Frequency = Independent.iloc[:,period:period*2].T.reset_index(drop= True).T
#PCA feature extraction
from sklearn.decomposition import PCA
pca = PCA(n_components= 25)
pca_V = pca.fit_transform(Voltage)
pca = PCA(n_components= 35)
pca_C = pca.fit_transform(Current)
pca = PCA(n_components= 35)
pca_F = pca.fit_transform(Frequency)
pca_V = pd.DataFrame(pca_V)
pca_C = pd.DataFrame(pca_C)
pca_F = pd.DataFrame(pca_F)
pcaData = pd.concat([pca_V, pca_C, pca_F], ignore_index=True, sort =False, axis =1)


#Now something to split up for test and training cases
```

```
X_train, X_test, y_train, y_test = train_test_split(pcaData, Dependent, test_size=0.15,
random_state=7)


# function to fit model on dataset
import numpy as np
import keras
import matplotlib.pyplot as plt
from keras.models import Sequential #ANN architecture
from keras.layers import Dense #The layers in the ANN


#Function definitions
def fit_model(trianX, triany):
#define model
model = Sequential()
model.add( Dense(100, kernel_initializer='normal', activation='relu', input_dim = 95))
model.add( Dense(55, kernel_initializer='normal', activation='relu'))
model.add( Dense(30, kernel_initializer='normal', activation='relu'))
model.add(Dense(1))


#compile model
model.compile(
optimizer = 'adam',
loss = 'mean_squared_error',
metrics = ['cosine_proximity']
)


#fit model
history =model.fit(
trianX,
triany,
validation_split = 0.15,
epochs = 1100, #The number of iterations over the entire dataset to train on
batch_size = 450, #the number of samples per gradient update for training
)
return model, history.history


#Now train them
for i in range(9):
print(str(i)+"th is going!!!!")
Return = fit_model(X_train, np.array(y_train[i].tolist()))
```

```
member = (Return[0])
member.save("/content/drive/My Drive/Thesis/9paramonemodelperparameter/model_"+str(i)+".h5")
with open('/content/drive/My Drive/Thesis/9paramonemodelperparameter/history'+str(i)+'.pkl',
mode = 'wb') as f:
pickle.dump(Return[1], f)
print("Saved model to drive")
```

C.2 Bagging Subroutine

```
# function to fit model on dataset
def fit_model(trianX, triany):
#define model
model = Sequential()
model.add( Dense(100, kernel_initializer='normal', activation='relu', input_dim = 95))
model.add( Dense(55, kernel_initializer='normal', activation='relu'))
model.add( Dense(30, kernel_initializer='normal', activation='relu'))
model.add(Dense(9,kernel_initializer='normal', activation = 'linear'))


#compile model
model.compile(
optimizer = 'adam',
loss = 'mean_squared_error',
metrics = ['accuracy']
)


#fit model
history =model.fit(
trianX,
triany,
validation_split = 0.15,
epochs = 1100, #The number of iterations over the entire dataset to train on
batch_size = 450, #the number of samples per gradient update for training
)
return model


# make an ensemble prediction for multi-class classification
def ensemble_predictions(members, testX):
#make predictions
```

```
yhats = [model.predict(testX) for model in members]
#member is trained collection of models
yhats = np.array(yhats)
#Sum across ensemble members
summed = np.sum(yhats, axis = 0)
#Averaged out
result = summed/len(members)
return result


# evaluate a specific number of members in an ensemble
#coefficient of determination, a.k.a. R2
from sklearn.metrics import r2_score
def evaluate_n_members(members, n_members, testX, testy):
# select a subset of members
subset = members[:n_members]
# make prediction
yhat = ensemble_predictions(subset, testX)
# calculate accuracy
return r2_score(testy, yhat)


X_remain, X_repeated, y_remain, y_repeated= train_test_split(X_train, y_train, test_size=0.2,
random_state=7)

# Create a random subsamples from the dataset with replacement
# One random subsample of size P(540) and a repeated subsample of size Q(135) for total
input sample of 675
#repeatedsample = (X_repeated, y_repeated)
from numpy import random
baggingsamplesize = 10
trainx, trainy = dict(), dict() # this will be the Dictionary of Dataframes as value
INDEX = X_remain.index
for _ in range(baggingsamplesize):
randindex = random.choice(INDEX,45473)
trainx[_] = pd.concat([X_remain.loc[randindex],X_repeated])
trainy[_] = pd.concat([y_remain.loc[randindex], y_repeated])


#Now train them
members = []
for _ in range(baggingsamplesize):
print(str(_)+"th is going!!!!")
members.append(fit_model(trainx[_], trainy[_]))
```

```
members[_].save("/content/drive/My Drive/Thesis/model_"+str(_)+".h5")
print("Saved model to drive")
```

C.3 Random noise addition algorithms

```
#random noise addition code : noise should/could be negative too
# Given a limit and the original, the algorithm return a noisy data


def noiseadd(original,limits):
random.seed(7)
Return = []
for _ in range(len(original)):
ll, ul = limits[_][0], limits[_][1]
noise = round(random.uniform(ll/10,ul/10), 3)
sign= 1 if random.random() < 0.5 else -1
Return.append(original[_]+sign*noise)
return Return
```

C.4 Support vector machine algorithm

```
#Import libraries
import os
import re
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split


#Loading saved data
infile = open('/content/drive/My Drive/Thesis/9paramonemodelperparameter/Independentsvm.pkl','rb')
Independent = pickle.load(infile)


period = 1446
Voltage_all = Independent.iloc[:,:period]
```

```
Frequency_all = Independent.iloc[:,period:period*2].T.reset_index(drop = True).T
Active_all = Independent.iloc[:,period*2:period*3].T.reset_index(drop= True).T
Reactive_all = Independent.iloc[:,period*3:period*4].T.reset_index(drop= True).T
time = [i/720 for i in range(1446)]


#Downsampling PAA
def subsample(data, sample_size):
samples = list(zip(*[iter(data)]*sample_size))  use 3 for triplets, etc.
return list(map(lambda x:sum(x)/float(len(x)), samples))


headers = [Voltage_all, Frequency_all, Active_all, Reactive_all]
for i in range(len(headers)):
temp = []
for j in range(len(headers[i])):
temp.append(subsample(headers[i].iloc[j], 12))
if i==0:
Voltage_all1 = pd.DataFrame(temp)
elif i==1:
Frequency_all1 = pd.DataFrame(temp)
elif i==2:
Active_all1 = pd.DataFrame(temp)
elif i==3:
Reactive_all1 = pd.DataFrame(temp)


#SVM Implementation
# Initialize
X1_train = pd.DataFrame(columns=['dp(k-1)', 'dp(k-2)', 'dp(k-3)', 'dp(k-4)', 'du(k)', 'du(k-
1)', 'du(k-2)', 'du(k-3)', 'du(k-4)', 'df(k)', 'df(k-1)', 'df(k-2)', 'df(k-3)', 'df(k-4)'])
X1_test = pd.DataFrame(columns=['dp(k-1)', 'dp(k-2)', 'dp(k-3)', 'dp(k-4)', 'du(k)', 'du(k-
1)', 'du(k-2)', 'du(k-3)', 'du(k-4)', 'df(k)', 'df(k-1)', 'df(k-2)', 'df(k-3)', 'df(k-4)'])
y1_train = pd.Series()
y1_test = pd.Series()


from sklearn.svm import SVR # "Support vector regression"
modelsvr = SVR(kernel='rbf', C= 61, epsilon=0.1)


# Training process

one = np.random.choice(indexes) #indexes is collection random index from training data
for i in indexes:
```

```
Voltage = Voltage_all1.iloc[i]
Frequency = Frequency_all1.iloc[i]
ActiveP = Reactive_all1.iloc[i]


#Now find the per instant of time change in values
#Going for fourth order


#Let's make a DataFrame with each of them in it.
DATA = pd.DataFrame(data=[ActiveP, Voltage, Frequency]).T
Columns = ['ActiveP', 'Voltage', 'Frequency']
DATA.columns = Columns


#Let's find previous state values
DATA['ActiveP_PreviousState']= DATA['ActiveP'].shift(1)
DATA['Voltage_PreviousState']= DATA['Voltage'].shift(1)


DATA=DATA.fillna(0)


#Finding deltas
DATA['delta_ActiveP']=DATA['ActiveP']-DATA['ActiveP_PreviousState']
DATA['delta_Voltage']=DATA['Voltage']-DATA['Voltage_PreviousState']


svmInput = DATA[['delta_ActiveP', 'delta_Voltage', 'Frequency']]
Y = pd.DataFrame(columns=['dp(k-1)', 'dp(k-2)', 'dp(k-3)', 'dp(k-4)', 'du(k)', 'du(k-1)',
'du(k-2)', 'du(k-3)', 'du(k-4)', 'df(k)', 'df(k-1)', 'df(k-2)', 'df(k-3)', 'df(k-4)'])
Y.empty


dpmi1= svmInput['delta_ActiveP'][4:-1].reset_index(drop = True)
dpmi2= svmInput['delta_ActiveP'][3:-2].reset_index(drop = True)
dpmi3= svmInput['delta_ActiveP'][2:-3].reset_index(drop = True)
dpmi4= svmInput['delta_ActiveP'][1:-4].reset_index(drop = True)


du = svmInput['delta_Voltage'][5:].reset_index(drop = True)
du1 = svmInput['delta_Voltage'][4:-1].reset_index(drop = True)
du2 = svmInput['delta_Voltage'][3:-2].reset_index(drop = True)
du3 = svmInput['delta_Voltage'][2:-3].reset_index(drop = True)
du4 = svmInput['delta_Voltage'][1:-4].reset_index(drop = True)
```

```
df = svmInput['Frequency'][5:].reset_index(drop = True)
df1 = svmInput['Frequency'][4:-1].reset_index(drop = True)
df2 = svmInput['Frequency'][3:-2].reset_index(drop = True)
df3 = svmInput['Frequency'][2:-3].reset_index(drop = True)
df4 = svmInput['Frequency'][1:-4].reset_index(drop = True)


Y['dp(k-1)']= dpmi1
Y['dp(k-2)']= dpmi2
Y['dp(k-3)']= dpmi3
Y['dp(k-4)']= dpmi4
Y['du(k)']= du
Y['du(k-1)']= du1
Y['du(k-2)']= du2
Y['du(k-3)']= du3
Y['du(k-4)']= du4
Y['df(k)']= df
Y['df(k-1)']= df1
Y['df(k-2)']= df2
Y['df(k-3)']= df3
Y['df(k-4)'] = df4
temptarget = DATA['ActiveP'][5:]


if i==one:
X1_test =X1_test.append(Y,ignore_index=True)
y1_test =y1_test.append(temptarget,ignore_index=True)
else:
X1_train =X1_train.append(Y,ignore_index=True)
y1_train =y1_train.append(temptarget,ignore_index=True)


modelsvr.fit(X1_train, y1_train)
```