5-2019

Synchrophasor-based Fault Location Detection and Classification, in Power Systems, using Artificial Intelligence

Hemal Falak
University of Arkansas, Fayetteville

Synchrophasor-based Fault Location Detection and Classification,
in Power Systems, using Artificial Intelligence

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering

by

Hemal Falak
National University of Sciences and Technology
Bachelor of Engineering in Electrical Engineering, 2014

May 2019
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Roy A. McCann, Ph.D.
Dissertation Director:

Juan Balda, Ph.D.
Committee Member

Yue Zhao, Ph.D.
Committee Member

ABSTRACT

With the introduction of sophisticated electronic gadgets which cannot sustain interruption in the provision of electricity, the need to supply uninterrupted and reliable power supply, to the consumers, has become a crucial factor in the present-day world. Therefore, it is customary to correctly identify fault locations in an electrical power network, in order to rectify faults and restore power supply in the minimum possible time. Many automated fault location detection algorithms have been proposed, however, prior art requires topological and physical information of the electrical power network. This thesis presents a new method of detecting fault locations, in transmission as well as distribution networks, using state-of-the-art machine learning algorithms on the real-time synchrophasor measurements obtained from the network. The proposed method first generates a bus admittance matrix from the synchrophasor data and then uses a neural network to identify the faulty buses. It is independent of network-specific data of the electrical power network. The proposed algorithm is evaluated using actual outage data from a real transmission system of Southwest Power Pool, in the year 2015. The results of the system implemented in python shows that the proposed method can detect fault locations with 100% accuracy.

ACKNOWLEDGMENTS

I would like to express my gratitude to my academic advisor Dr. Roy McCann for his extra-ordinary help, support and guidance extended throughout the course of my Masters thesis. I would like to thank him for his expert advice, encouragement and brilliance in his respective field. I would also like to appreciate the perceptive and constructive feedback of my Masters committee members Dr. Juan Balda and Dr. Yue Zhao.

Furthermore, I would like to acknowledge the unparalleled love, care and support of my husband and family members, and the way they inspired me to dig deeper into graver societal problems and come-up with practical solutions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 CHAPTER: INTRODUCTION

In this chapter, the problem statement at hand, its importance, limitations in prior art and the proposed solution method have been introduced, which are as follows:

1.1 Overview

With the introduction of revolutionary and complex electronic equipment which cannot sustain interruption in the supply of electricity, it is customary to provide ceaseless electric power. In the past, permanent faults were a subject of great interest but with the advancement in science & technology and inception of cutting edge yet sensitive electronic appliances, there is a paradigm shift in diverting attention towards temporary faults as well and emphasis has been laid on supplying electric power smoothly. For this reason, various protection schemes, for electric power network, have been introduced in order to minimize duration of a fault as well as the customers affected by a fault. Previously, the staff of an electric utility were mainly informed of faults or outages by phone calls from the consumers. They had to use the location of the caller and send a crew in order to patrol the whole feeder area for detecting fault and restoring electricity supply. This process was cumbersome and time consuming. Numerous automated algorithms have been proposed till date to alleviate this procedure and make it more efficient, each having its own strengths and weaknesses. The coherence of each algorithm depends on how accurately it identifies the network topology and pin points fault locations.

A common impediment in fault location detection is the physical nature of the electrical network topology being implemented in that area, since the algorithms proposed are intrinsic to a special category of transmission/distribution systems. Conventionally, numerous physical constraints are required to be realized before proposing any fault-detection algorithm e.g., What kind of network topology is being implemented? Are the transmission lines under study over-head or underground? Are the transmission lines under study transposed or un-transposed? etc. Besides these aspects, most of the fault location detection algorithms require steady state data for accurate analysis. Furthermore, the accuracy of such algorithms is greatly affected by the assumptions made due to line parameters uncertainty and insufficient or imbalanced data.

Therefore, such customized approaches of automating fault-detection in an electrical power system are specific to the particular power systems they are designed for, and cannot be deployed universally, and any crucial changes in the electrical network topology of those power systems i.e., grid/transmission system expansion or augmentation, would make it obligatory, for such power companies, to reshape their fault detection algorithms as well. Hence, the current approaches for detection of fault currents are limited by various physical constraints. And with the growing demand of energy and an increase in the integration of renewable resources into electric grids, substantial changes in a power system network are inevitable; therefore, the need of hour is to propose such an automated and generic fault-detection algorithm as is void of the aforementioned physical constraints, can efficiently detect faults in transmission or distribution networks irrespective of their nature and construction, and can ensure rapid restoration of electricity to the customers resulting in fewer customer complaints and outage time, as well as decreased loss of revenue due to outages and crew repair costs, in a competitive electric power industry [2].

1.2 Problem Statement

The development of an efficient, precise and accurate real-time fault detection system using artificial intelligence, based on synchrophasor measurements of bus voltages and current injections, in an electrical transmission and distribution network.

1.3 Limitations of Prior Art

The prior at in fault location detection in distribution networks is broadly limited in the following four aspects. Firstly, it is specific to either balanced networks or unbalanced networks in power systems and does not cater for both at the same time. Secondly, it relies on feeders mathematical models and mostly assumes radial topology for distribution networks. Thirdly, it uses network specific data in its computations e.g., network topology, feeder connection schemes and electrical parameters like cable type and geometry etc. The data required for computations is obtained from utility companies database once and is updated occasionally, thus ignoring the rapid changes in distribution networks. And lastly, as explained further in the section of literature review of this thesis, some of the prior art in distribution networks does not address the issue of multiple fault locations.

On the other hand, the conventional prior art in transmission networks is broadly

limited in the following three aspects. Firstly, while using single-ended data measurements, numerous assumptions are made for the lines, which might not reflect the original system. Secondly, while using transmission lines models, lumped models are preferred for computations thus ignoring inter-phase coupling phenomenon in transmission lines. Some solutions do offer compensations for inter-phase coupling in transmission lines but such schemes work for off-line post-fault analysis only. Thirdly, a relatively accurate method of detecting fault current in transmission lines, called as the traveling wave phenomenon, is dependent on frequency and requires very high sampling rate thus increasing the associated costs by manifolds. Therefore, there has been a paradigm shift in using synchrophasor measurements for detecting fault currents in such creative ways as are not constrained by the aforementioned limitations.

In this thesis, a blend of synchrophasor measurements with artificial neural networks has been presented, in an innovative way, to detect fault currents in power systems in real-time. The solution methodology proposed is free from the limitations stated above.

1.4 Proposed Solution Methodology

My proposed methodology is to build upon the work of using synchrophasor measurements of bus voltages and current injections to form the network bus admittance matrix statistically, by constructing and training a neural network to learn different patterns, based on its excellent pattern recognition skills [3], so that the dependency on transmission line models or physical constraints is reduced/omitted. Once the network bus admittance matrix is extracted statistically from the provided synchrophasor data, a neural network can be formed by connecting biologically-inspired elementary neurons in different formations and made to learn patterns about the system admittances. These learned patterns can then be used to detect and locate faults in the system. This pattern recognition algorithm formed, using artificial neural network, has many advantages over other conventional methods devised for fault location detection e.g., parallel- processing and non-linear curve mapping etc., specifically in cases where explicit solution formulation is impossible.

Before going into the details of the algorithms designed using artificial neural network, it is customary to give its general introduction first, followed by the broader concept formulated, which is as follows:

An Overview of Neural Networks

Basic Structure A basic neural structure consists of several layers of neurons e.g., input layer, output layer and hidden layer etc. Each neural layer can receive weighted inputs 'i' from input layer or another neural layer and/or an external source (explained in detail in the section 3.2.1 and 3.2.2 of the thesis) and calculate the sum of weighted inputs, followed by application of an 'activation' function on the weighted sum of the received input signals to produce requisite outputs. This activation function can be of different types e.g.,

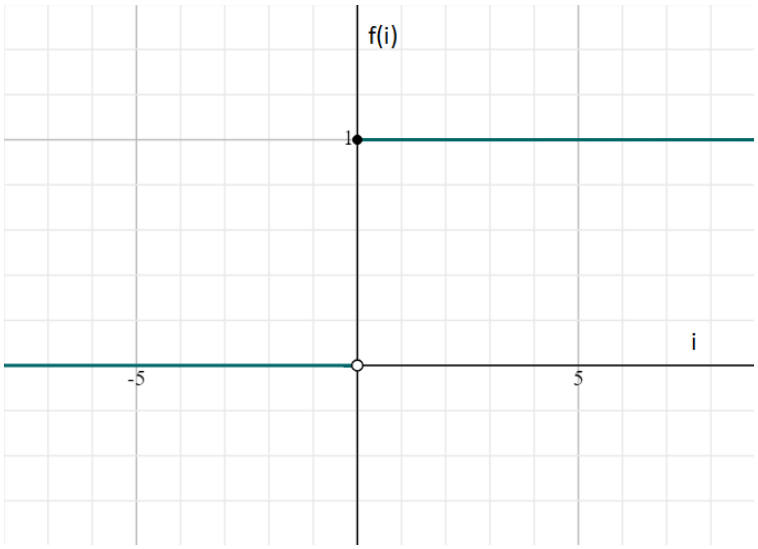


Figure 1.1: Step Function

- Step: $f(i) = \begin{cases} 1, \ if \ i \geq 0 \\ 0, \ if \ i < 0 \end{cases}$, it is illustrated in Fig. 1.1.
- Piece-wise Linear: $f(i) = \begin{cases} 1, \ if \ i > 1 \\ -1, \ if \ i < -1 \\ i, \ if \ |i| < 1 \end{cases}$, it is illustrated in Fig. 1.2.
- Sigmoid Uni-Polar: $f(i) = \frac{1}{1+e^{-\beta i}}$, it is illustrated in Fig. 1.3.
- Sigmoid Bi-Polar: $f(i) = \tanh(\beta i) = \frac{1-e^{-2\beta i}}{1+e^{-2\beta i}}$, it is illustrated in Fig. 1.4.

Types of Neural Networks A neural network can be of two types, trained in three different ways. The two types of neural networks are:

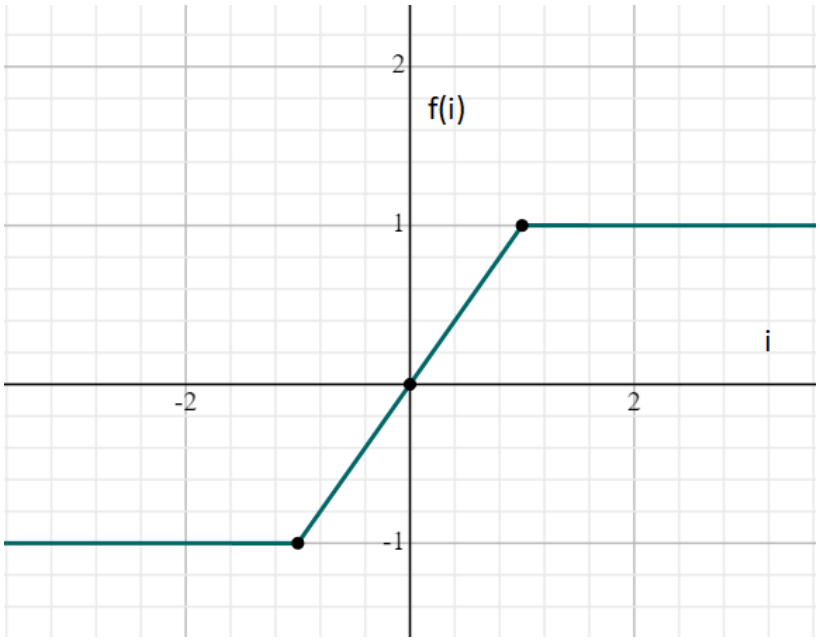


Figure 1.2: Piece-wise Linear Function

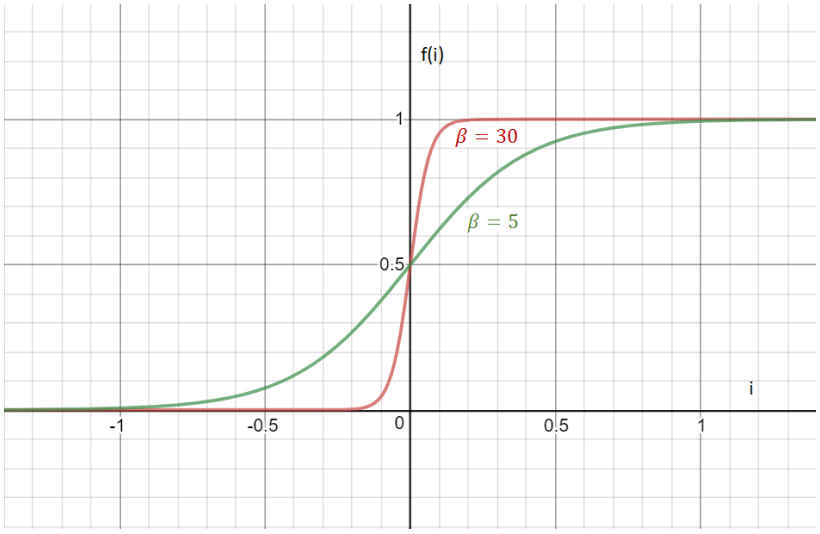


Figure 1.3: Sigmoid Uni-Polar Function

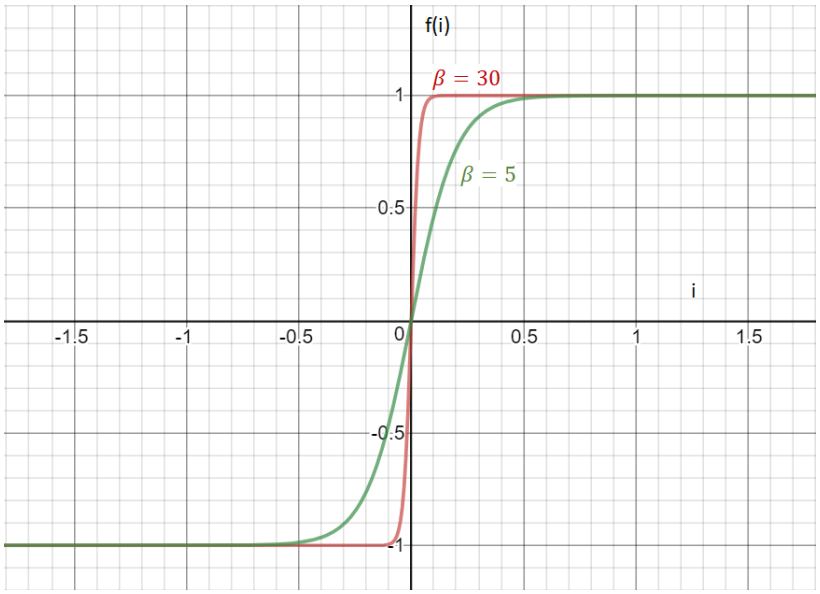


Figure 1.4: Sigmoid Bi-Polar Function

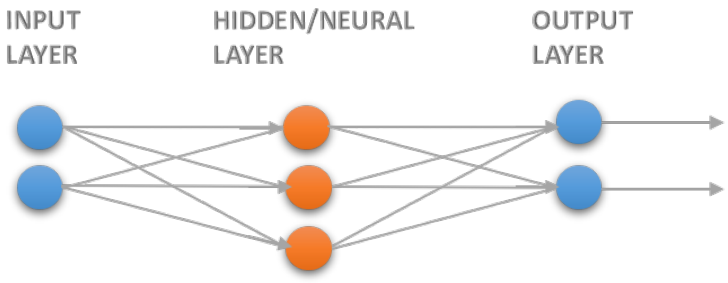


Figure 1.5: Feed-forward Neural Network

1. **Feed-forward Neural Network:** Also known by the name of 'Perceptrons', as the name suggests, feed-forward neural networks work by one-way propagation of weighted input values, modified at each stage. There is no direct feedback path provided between the output and input values as shown in the Fig. 1.5.

   The neurons in a feed-forward neural network compute the weighted sum of the received inputs in terms of elements of either an input vector and external bias vector or output from the preceding neural layer. Thus, the signal processing in the 'a^{th}' layer of a feed-forward neural network is executed as follows:

   Let 'x' be the input vector to the a^{th} layer i.e., $x = [x_1\ x_2\ x_3\ \ldots\ x_n]_{1\times n}$,

   and 'W' be the neural connection weights assigned to the input vector i.e.,

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,m} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,m} \\ w_{n,1} & w_{n,2} & \ldots & w_{n,m} \end{bmatrix}_{n\times m}$$

   where,
   n is the number of elements comprising the input vector.
   m is the number of neurons in the a^{th} neural layer.
   $w_{i,j}$ is the neural connection weight between element 'i' of the input vector and neuron 'j of the a^{th} neural layer.
   $B1 = constant\ \times\ [unity\ matrix]$, showing any external inputs to the a^{th} neural layer.

   Then, the collective output of the neurons of the a^{th} neural layer can be represented as: $Z' = X.W + B1$

$$Z' = \begin{bmatrix} z'_1 \\ z'_2 \\ z'_m \end{bmatrix} = \begin{bmatrix} x_1.w_{1,1} + x_2.w_{1,2} + \ldots + x_n.w_{1,m} \\ x_1.w_{2,1} + x_2.w_{2,2} + \ldots + x_n.w_{2,m} \\ x_1.w_{n,1} + x_2.w_{n,2} + \ldots + x_n.w_{n,m} \end{bmatrix}_{1\times m} + b1 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}_{1\times m}$$

   where, z'_j = output of j^{th} neuron in the a^{th} neural layer.

   Since there is an absence of any direct feedback path between the predicted outputs and provided inputs, therefore, for minimizing error between the predicted output values and requisite output values, the weights of input values are adjusted, during training

of the neural network, through 'back-propagation', by looking at the requisite output values, predicted output values and provided input data-base after each iteration. The neural weights are adjusted by computing and minimizing the difference between requisite (actual) and predicted outputs. This difference is calculated through a loss function, which is a function of internal network parameters.

Different types of loss functions can be used, for calculating the difference between requisite outputs and predicted outputs depending upon the nature and type of the target variables / outputs e.g., **continuous target variables** (continuous outputs) i.e., 'regression' and **discrete target variables** (discrete outputs) i.e., 'classification' etc.

Some of the commonly used loss functions for continuous outputs are as follows [4]:

- Absolute Error: It computes the element-wise difference i.e., $x_0 - x$, where, x_0 is the actual output value and x is the predicted output value.
- Smooth Absolute Error: It is a modified and smoother version of absolute error obtained by including moving average in its computation.

Similarly, some of the commonly used loss functions for discrete outputs are as follows [4]:

- Cross Entropy:

  This function determines the loss in terms of probability of a certain input value to be in a particular output class i.e.,

  $$-\left(y log\left(p\right) + \left(1-y\right) log\left(1-p\right)\right),$$

  in case of two output classes. where,
  y is a binary variable, being assigned a value based on the actual output class of input 'i'. y can either be '0' (indicating the input is not in the first output class) or '1' (affirmative), and,
  p is the predicted probability of the input to be in the first output class. and,

  $$-\sum_{c=1}^{C}\left(y_c log\left(p_c\right) + \left(1-y_c\right) log\left(1-p_c\right)\right),$$

  in case of multiple output classes (in which the binary Cross Entropy for each class is computed separately and summed up for all the output classes to give the

combined Cross Entropy error for multiple output classes). where,
C represents the total number of output classes.
y is a binary variable, being assigned a value based on the actual output class of input 'i'. y can either be '0' (indicating that input 'i' is not in the output class 'c') or '1' (affirmative), and,
p is the predicted probability of input 'i', between 0 and 1, to be in the output class 'c'.

- Negative Loss Likelihood (NLL): This function determines the loss by taking negative loss of the predicted probability of an input class to be in an output class [5] i.e., $L(y) = -log\,(y)$ where, y is the score or probability of an input value 'i' obtained as the output of a neural network to be in a certain output class.
- Margin Classifier: This function computes the loss by associating a distance 'd' to each predicted class from the respective decision boundary of that output class.
- Soft Margin Classifier: This loss function is used in case of non-separable output classes. It works the same way as the Margin Classifier.

Likewise, there are loss functions for embedded type outputs i.e., when output is computed based on the similarity or dissimilarity of the provided inputs and visualized outputs i.e., when input data is processed to visualize an output image. The selection of loss function depends on the type of application or in other words the target output variable in the constructed neural network.

2. **Feedback Neural Network:** Also known as 'Recurrent Networks', as the name suggests, a feedback path is provided between the output and input layers. There can be different types of recurrent networks depending on the nature of the connection between the output layer (either from the output layer or a neural layer/hidden layer) and input layer. A simple example of feedback neural networks is shown in the Fig. 1.6.

   This type of neural network is best suited for dynamic processing, but a major drawback is the probability of the existence of unstable regions in the network.

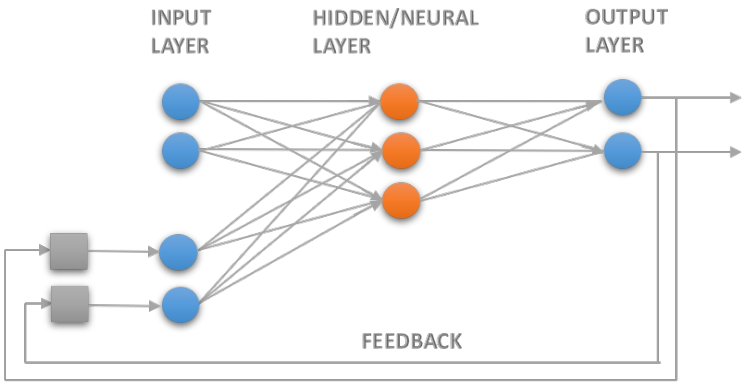


Figure 1.6: Feedback Neural Network

Training Methodologies As stated earlier, the three types of neural networks training ways are:

1. **Supervised Neural Networks:** A neural network when made to learn the patterns of labeled input observations, constitutes supervised learning. This type of learning process is commonly implemented for classification purposes, when the requisite input-output mapping is known.

2. **Unsupervised Neural Networks:** A neural network when made to learn the patterns of unlabeled input observations, constitutes unsupervised learning. This type of learning process is commonly implemented for clustering purposes, when the inputs are required to be clustered in specific groups.

   A chart describing supervised and unsupervised learning strategies is shown in Table. 1.1 [6]:

Table 1.1: Supervised and Unsupervised Neural Networks

	Supervised Learning	**Unsupervised Learning**
Continuous	Regression	Dimensionality Reduction
Discrete	Classification	Clustering

3. **Semi-supervised Neural Networks:** This type of learning falls between supervised and unsupervised learning of neural networks. It is used when some of the input data is labeled and the rest is not labeled.

Proposed Concept Formulation

A supervised feed-forward neural network when constructed and trained would be used to detect and locate faults in the power network, as shown in the Fig. 1.7.

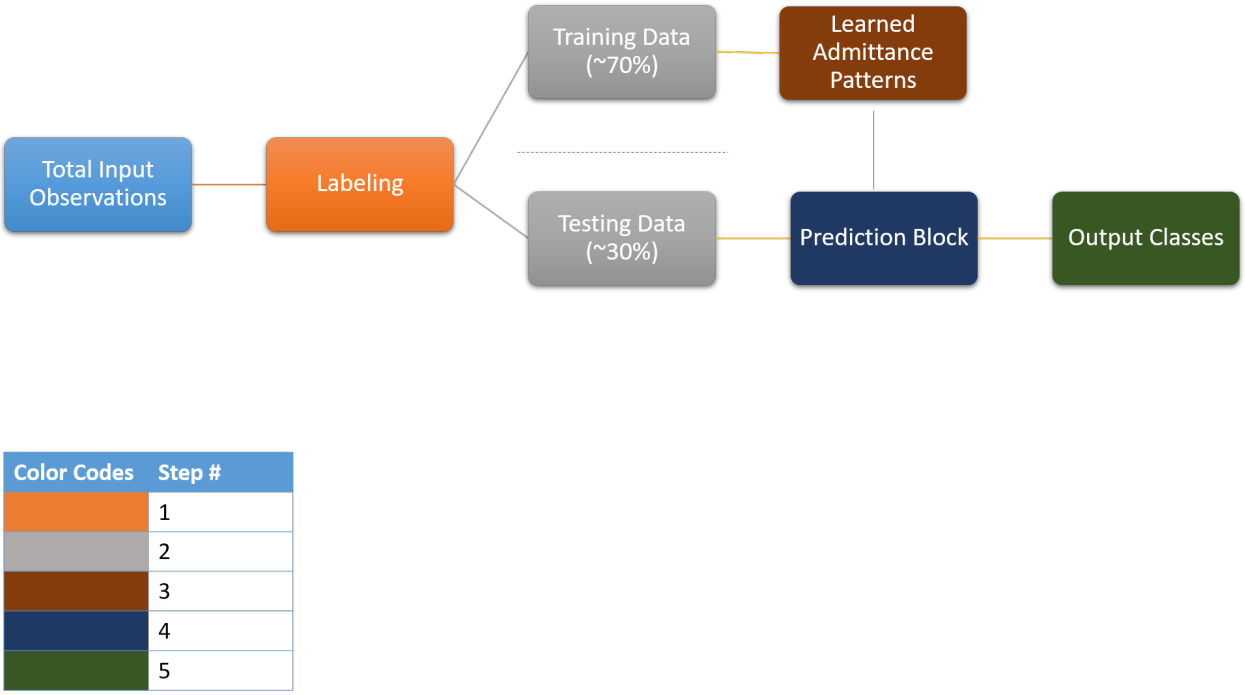


Figure 1.7: Proposed Methodology

The input values i.e., system admittances would be labeled (keeping in mind the requisite input-output mapping) as normal data and faulty data, and divided into two data sets i.e., the training data set and the testing data set. The training data set would then be used to train the neural network and the testing data set to test the accuracy of the trained neural network. The efficacy of the proposed methodology has been illustrated in actual transmission system.

1.5 Thesis Outline

The rest of the thesis explains literature review followed by technical details and results of my proposed methodology, envision and conclusion.

2 CHAPTER: LITERATURE REVIEW

This section discusses the prior art in fault location detection primarily, in both transmission and distribution systems, in electrical power networks. It also analyzes the strengths and weaknesses of each scheme.

2.1 Prior Work in Distribution Systems

Several algorithms have been proposed in distribution systems to accurately locate faults in power lines [7, 8, 9, 10, 11, 12, 13, 14, 15, 16] each having its own merits and demerits.

M.Wache in [7] describes the inevitability of deploying Phasor Measurement Units (PMUs) in distribution networks and proposes acquisition of phasor measurement data through PMUs for accurately studying the dynamic behavior of distribution networks, caused due to power injection from renewable energy resources like solar, wind and biomass etc., since distribution networks are taking over the tasks originally reserved for transmission networks in power systems. The synchrophasor data obtained through these PMUs can be utilized in a variety of crucial applications in distribution networks e.g., wide-area monitoring, fault detection and automatic restoration, angle/frequency monitoring, voltage stability monitoring and improved state estimation etc. Therefore, it is highly likely to have distribution networks with installed PMUs, in the future.

E.C.Senger in [8] proposes an automated system for fault location detection in primary distribution networks. The proposed system uses Intelligent Electronic Devices (IEDs) with built-in oscillography function (installed at each medium voltage feeder level) and information about network topology, network specific electrical parameters e.g., cable type, feeder geometry such as overhead, spacer, twisted and underground etc., nominal power of distribution transformers and different feeder connection schemes, to acquire requisite data for automatically detecting fault locations in the system. This scheme works for both balanced and unbalanced distribution networks and provides probable fault locations; however, there are five limitations to this approach. Firstly, this scheme utilizes network-specific data like network topology, feeder connection schemes and distribution networks electrical parameters e.g., cable type and underlying geometry etc., which is obtained from the electric

utility companys data-base and updated periodically. Due to the frequent modifications in feeder structure, it is highly likely to use outdated data this way for fault location detection. Secondly, this scheme only works for over-current faults detection and doesnt identify high impedance faults (HIF) incurred in the distribution networks. Thirdly, this scheme neglects fault impedance for phase-to-ground faults and works for radial topology of distribution networks only. Fourthly, all the connected segments in the distribution networks are inspected iteratively for detecting fault locations. The probable fault locations are identified and ranked by combining information about the fault type and load rejection (obtained from DSP module) with the feeders topology (assumed to be radial) and electrical parameters (stored in the database which may be outdated) e.g., for permanent faults isolated by breakers, the algorithm ranks detected faulty points protected by fuses, as less probable ones and vice versa. Since the operation of primary/secondary protection schemes is not considered, therefore, it may detect multiple or misleading fault locations. And lastly, while estimating load currents during the faults, it simplifies the problem by assuming that each distribution transformer connected to the feeder supplies load in proportion to its nominal apparent power, which may not always be correct (since it depends on the connected load).

L.Xu in [9, 10] investigates two different classification methods i.e., logistic regression and artificial neural networks (ANN), used in distribution systems, to diagnose faults. Two major causes of faults i.e., animal and tree contact, in distribution systems are used to show the characteristics and the effectiveness of the investigated approaches. In logistic regression's approach, Lu talks about getting recorded faulty data from Dukes Energy Power Distribution Fault Data Collection System, and using six (6) most important features i.e., x1,x2,x3,x4,x5 & x6, out of thirty-three (33) total recorded features of the fault incident, to fit a linear curve ($\alpha + \beta X$ where X = (x1,x2,x3,x4,x5,x6) — x1 $\in$ Circuit ID, x2 $\in$ Weather, x3 $\in$ Season, x4 $\in$ Time of Day, x5 $\in$ Number of Affected Phases, x6 $\in$ Protective Devices Activated) between them, for each fault cause. The probability of each new data point X is calculated by $P(Y = 1) = 1/(1 + e - (\alpha + \beta X))$, for each fault cause, and compared with a predefined threshold value (determined using probabilistic approach) of that particular fault cause, to classify the new fault cause as 'yes/no'. Since only two fault causes are considered here, therefore, each new fault cause is classified as 'animal/non-animal' and 'tree/non-tree'. A fusion mechanism is then used to provide the final decision of the most probable fault cause, by comparing the two probabilities of the new fault cause by their respective threshold values. The farther a probability is from its threshold value, for a particular

fault cause, the most likely the fault is caused by that fault cause. Although this approach gives satisfactory results yet there are two limitations to this approach. Firstly, multiple parallel branches compute probable outputs which are merged at the end to provide the final decisions based on probabilistic approach. The data used for probabilistic approach is fixed and may change in future, due to the frequent modifications in distribution networks, which may give erroneous results. Secondly, linear curve fitting technique is used without providing enough evidence, whereas the data used for curve-fitting may be non-linear. In Artificial Neural Networks (ANN) approach, L.Xu constructs a neural network and trains the neural network once to learn the patterns of different fault causes, through the fault data obtained from utility company. The trained neural network then determines fault causes for new fault incidents based on its own learning. Although the conclusion explained in this work favors neural network for accurate fault cause classification yet one short-coming observed in this technique is that the neural network is trained once using fixed data from the utility company and the data trends may change in future.

D. Thukaram in [11] talks about locating faults in radial distribution networks by using artificial neural networks (ANN) and support vector machine (SVM) approach, using measurements at the sub-station only. Principle component analysis (PCA) technique is used to analyze faults which are then classified using SVM according to their type and short circuit levels (SSC), based on reactances of their paths. An ANN is trained to locate faults based on SVMs outputs as inputs (which are fault type and SSC level). The varying numbers of operational generators in a distribution network are specified by different short circuit levels and the ANN is trained separately for each SSC level and fault type e.g., if 4 fault types and 7 SSC levels (20MVA to 50MVA with a step of 5MVA) are considered, then 28 different neural networks are required to be trained, to accurately determine fault locations. This approach takes its requisite data from SCADA system, installed in the distribution network, through distribution automation (DA) system. Two of the shortcomings of this technique are that it does not address multiple locations problems, and needs a large number of neural networks for addressing wide-area distribution networks with numerous operational generators.

J.Kim in [12] uses PQ monitoring data to locate faulted sections in distribution networks. It first identifies the type of fault by noticing over-currents in the phases and neutral line (from PQ monitoring data) and classifies the fault as line-to-ground (L-G), line-to-line (L-L), line-to-line-to-ground (L-L-G) and three phase (3Φ) faults. After classification, this scheme compares the fault current value (as seen at the feeder) with pre-determined fault-

current values at the feeder, for each fault type. The pre-determined fault current values are calculated theoretically using short-circuit analysis, for the feeder model, and stored in a table for comparison. As stated earlier, the actual fault current values are then compared with the pre-determined fault current values stored in the table and faulted sections are identified. These faulted sections are ranked in descending order of the closeness of actual fault current value with the theoretical table value. It is a relatively simple approach but there are four limitations to this approach. Firstly, this approach doesnt identify high impedance faults (HIFs). Secondly, in case of L-L faults it plots the curves for over-currents in both the phases and takes the flat portions of the curves as the fault current value/level; however, it doesnt provide any solution to the issues faced in assigning a level to the fault current value when the fault is changing i.e., it starts-off as L-L fault and ends as 3Φ since in this case there is no flat portion of the over-currents curves. Thirdly, this scheme determines the faulty sections of the distribution network. And lastly, the feeder model used for performing short-circuit analysis is not shown; therefore, in case of some assumed parameters, the pre-determined theoretical values may not match the actual fault current values and may cause mis-leading rankings of all the faulty sections determined.

A.Meier in [13] talks about developing high-precision Phasor Measurement Units (PMUs) for different applications in distribution systems e.g., unintentional island detection, topology status verification, direct-phase identification and balancing, reverse power flow detection, state estimation and fault detection etc. Thus, the presence of synchrophasor measurements in distribution system works wonders and it is highly likely to deploy PMUs in distribution networks as well, in the future.

J.Ren in [14] uses synchrophasor measurements in distribution networks to accurately locate faults. The proposed scheme can be used for active and passive networks, radial and looped topologies, high impedance faults, and is not limited to a certain pattern of PMU locations. This scheme works in two phases. In the first phase, all the perspective faulted segments are identified by calculating the fault distance based on its apparent impedance and checking if the calculated fault distance lies within the presumed faulty segment. And in the second phase, the actual faulted segment is located by comparing voltage phasors at the branches junctions obtained through synchrophasor measurements obtained from different terminals, where the voltage phasors at succeeding junction nodes are calculated from their respective voltage drops, since voltage phasor measurement at the starting junction point/terminal is known from synchrophasor measurement of that terminal. This scheme

has two limitations. Firstly, the electrical parameters like voltage drops, line impedance etc., and the physical connections in the network are required to be known. Thus accurate and complete prior knowledge of the system is necessary for this scheme to work. Secondly, it is dependent on line models for calculations.

J.Mora-Florez et al., in [16] explains and compares different impedance-based fault location detection techniques in distribution systems. These techniques locate fault in distribution networks by analyzing the deviation between the pre-fault and post-fault impedance of distribution lines. The major difference between these techniques lies in the power system topology used, line and load models used, and the necessity of other power system information required to be known. Though these techniques provide a good estimate of fault location detection in distribution networks but most of them use simplified feeder model, equivalent to voltage divider model. Moreover, these impedance-based methods work only on steady state values of voltages and currents used to estimate apparent impedance, from impedance deviation, which is directly linked with a distance to the fault, and do not account for multi-estimation of probable fault locations.

2.2 Prior Work in Transmission Systems

As seen in distribution systems, several algorithms have been proposed in transmission systems to accurately locate faults in power lines [2, 17, 18, 19, 20, 21, 22] each having its own merits and demerits.

J.D.L.Ree in [17] explains the importance of Phasor Measurement Units (PMUs) in transmission systems, and proclaims that the world-wide frequent blackouts have necessitated and expedited the process of installing PMUs in transmission systems. Hence, it is highly likely to expect an adequate number of PMUs in transmission systems, in near future. Furthermore, the work in [17] provides an overview of the different types of application possibilities viable due to the availability of voltage and current synchrophasor measurements e.g., power system monitoring, protection and control. J. Sykes et al., in [18] talks about the importance and evolution of synchrophasor measurements and informs that the concept of synchrophasor measurements dates back to 1980s. This paper also briefly explains the deployment history of PMUs in United States, Canada and Mexico. Thus implicating the necessity of PMUs in todays' power systems.

J.Jiang in [2, 19, 21] proposes a Phasor Measurement Unit (PMU) based fault location detection technique. While talking about the prior work in fault location detection in transmission systems, the work presented in [2] clearly identifies five major limitations in prior art, and supports using PMU measurements, in fault location detection, in transmission systems. First limitation is the use of single ended data measurements while implementing Fourier and Laplace transforms for locating faults in transmission lines, since several assumptions are required to be made which cause errors in computations due to source impedance variation, fault incidence angle, line asymmetry and loading conditions. Second limitation is the use of time domain transmission line models which require data sampling at a very high rate for accurately approximating the underlying derivatives of computational variables; such a high sampling rate is hard to achieve. Third limitation lies in using lumped models of transmission lines for fault location detection since these models cannot capture the inter-phase coupling phenomenon of transmission lines. Fourth limitation lies in implementing compensation techniques for long transmission lines while using lumped models of transmission lines, since such schemes works for off-line post-fault analysis only. And the last limitation lies in using traveling wave phenomenon for locating faults in transmission lies since this technique is frequency dependent, needs very high sampling rate and is costly. Conclusively using synchrophasor measurements are far most a better way to locate faults within transmission lines. The work in [2] performs online fault detection and location on single-phase or three-phase transposed transmission lines using synchronized phasor measurements of voltage and current at both ends of the transmission line. This scheme can work for series compensated transmission lines as well with slight modification. The basic idea is to write and solve second order partial differential equations (using KVL) in terms of fault distance x, for the transmission lines, where all other measurements are in per unit values i.e., for a single phase transmission line, the partial differential equations would be of the form: $\frac{\partial v}{\partial x} = Ri + L\frac{\partial i}{\partial t}$ and $\frac{\partial i}{\partial x} = Gv + C\frac{\partial v}{\partial t}$ (this concept can be extended to three-phase systems as well). The line parameters are estimated from the synchronized phasor measurement values obtained. This paper also talks about fault location detection in case of un-transposed lines (due to aging), by first estimating line parameters through perturbation method of matrix transformations and then using the same differential equations. The drawback of this method only lies in case of un-transposed lines (due to aging) when the line parameters are required to be estimated since it makes the solution computationally extensive for large networks.

Y.Lin in [20] talks about separating arching faults from permanent faults based on

phasor measurement units. This technique works by comparing magnitude of the arching voltage with a set threshold value. If the magnitude is higher than a set threshold value, the fault is classified as arching fault. It uses distributed parameter model of long transmission lines and implements Extended Discrete Fourier Transform on voltage and current waveform, for determining accurate waveform components, to eliminate errors from computations caused due to DC off-sets. Furthermore, it also detects and locates faults using existing fault detection algorithms. However, a limitation of this work is that it only identifies in-zone and out-of-zone faults in transmission systems and does not precisely locate the faults.

S.M.Brahma in [22] talks about using synchrophasor measurements for locating faults in multi-terminal transmission lines instead of using current measurements from current transformers (which are faulty with as high as 10% tolerance). The proposed scheme first identifies the faulty section first by comparing synchronized voltage phasor measurement at each tap with one another i.e., the section of a multi-terminal transmission line is assumed to be faulty if difference of the terminal voltages is higher than a specified tolerance, and then locates fault. The fault location detection algorithm works by deriving equations from bus admittance matrix in terms of fault current, post-fault bus admittance values, voltage deviation and distances L1 & L2, where L1 and L2 refers to the location of the fault from terminals 1 and 2 of the transmission line, and solving them to find the requisite unknown variables. This scheme assumes that fault resistances varies between 10Ω to 100Ω in case of faults involving ground and 1Ω to 10Ω in case of faults not involving ground and locates faults with 1% error.

The fault location detection method proposed in my thesis, uses the approach shown in [22] for calculating fault distance from the source, after identification of faulty segments through artificial neural networks.

3 CHAPTER: TECHNICAL SOLUTION DESCRIPTION

3.1 Proposed Algorithm: Concept

My Master's thesis focuses on devising a methodology for fault location detection by intelligently constructing upon the perks of the existing schemes to avert the need to provide network specific solutions. The algorithm devised builds on the work stated in [23], which treats bus voltages and current injections, in a power network, as random stochastic signals linked by a cross covariance matrix called as bus-impedance matrix, given by the formula:

$$Z^{bus} = \frac{1}{t_n} \sum_{k=1}^{t_n} V_{\sigma}^{(k)} * \frac{I_{\sigma}^{(k)H}}{\left| I_{\sigma}^{(k)} \right|^2} \tag{3.1}$$

The work in [23] makes use of synchrophasors, of different voltages and currents in the system, for identifying the topology of an electric network by extracting bus impedance matrix statistically using (3.1). Thus, omitting the requirement to know the physical connection of the electric network. The only parameters required for extracting network topology are bus voltage and injected currents synchrophasor measurements, where synchrophasors are phasors of sinusoidal waveforms at different time instants i.e., timed phasors of sinusoidal waveforms, which can be obtained from the Phasor Measurement Units (PMUs) installed separately or within the protective relays in the system (protective relays have been designed with built-in PMU functionality since 2002 and one is only required to enable the PMU modules of these protective relays to acquire the requisite synchrophasor measurements).

Building upon this concept, my research work focuses on forming such a supervised neural network as is trained by 1D input vectors of system admittances, acquired by converting the statistically determined bus admittance matrix – using bus voltage and injected currents' synchrophasors – to one-dimensional arrays. This supervised and trained neural network is then used to detect and pin-point normal admittances as well as admittances under fault conditions in the system. In other words, my research work makes use of artificial intelligence to learn the patterns of system admittances under normal and fault conditions, based on the bus admittance matrices provided in the form of 1D vectors from time to time. This system-specific learned pattern is the key to detecting fault currents in the power system, in real time. Conclusively, this neural network can be trained on any data set provided

from any power network (with training time ranging from a few microseconds to a few minutes depending on the number of buses in the power network and computational capacity of the underlying data center or computer), and used for real time fault-location detection, without having to know the network topology or physical construction of the power system e.g., it liberates the requirement of knowing whether it is a transmission system or a distribution system etc. for deployment purpose, provided that adequate number of synchrophasor voltage and injected current measurements are available for that system. An overview of the proposed methodology is shown in Fig. 3.1.

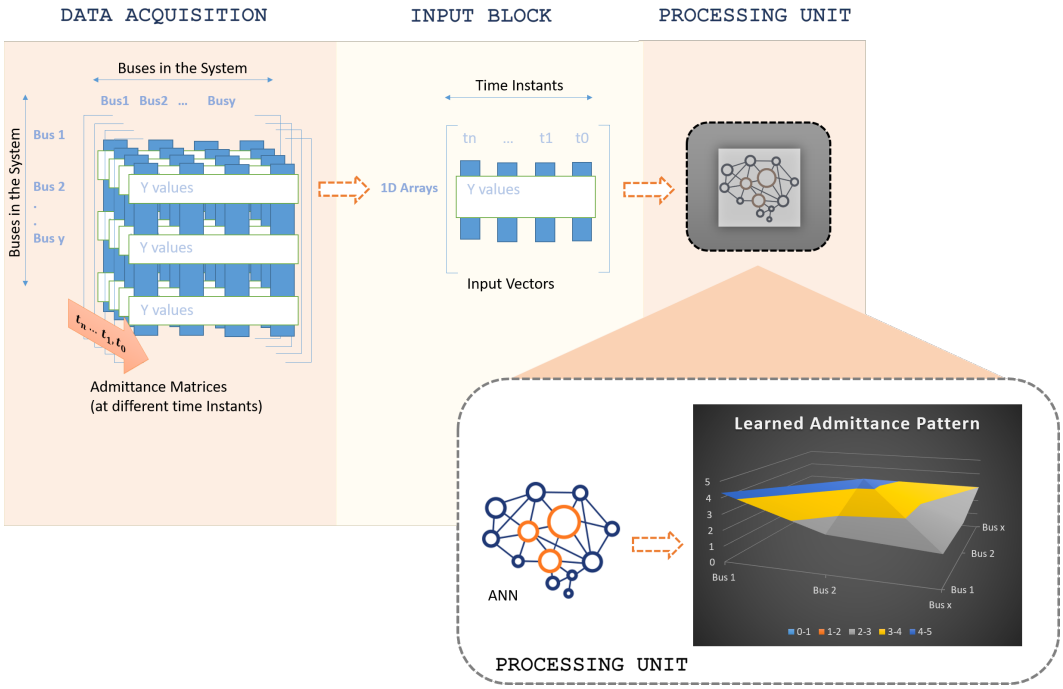


Figure 3.1: Concept of Proposed Methodology

Acquisition of synchrophasor measurements is not of concern in the transmission systems, since they can easily be acquired from Phasor Measurement Units (PMUs) or the protective relays (with built-in synchrophasor functionality) installed in the transmission networks; however, it was of concern in distribution networks. In the past, the use of synchrophasor measurements was not very common in distribution networks, but with the advent of synchrophasor measurements being used for wide area monitoring or stability supervision of transmission systems, efforts are being diverted to incorporate these measurements into distribution networks as well, where they would help us better understand such dynamic requirements of distribution networks as are caused due to inclusion of power

injections from renewable energy resources like wind, solar and biomass etc. With an increase in dynamic processes and power injections from renewable energy resources into the distribution systems, the distribution networks are also handling tasks originally reserved for transmission systems, thus making it a necessity to supervise or monitor distribution systems as well. To this end, Phasor Measurement Units (PMUs) are being deployed in distribution networks as well [1]. And thus, the real-time fault location detection methodology proposed, shows promising results in the present as well as in the future. Moreover, every approach implemented till date to automatically detect fault currents in the power networks, needs some kind of Intelligent Electronic Devices (IEDs) for extraction of information; therefore, making use of built-in synchrophasors measurement option, of the deployed protective relays, is prolific.

3.2 Proposed Algorithm: Prototype

3.2.1 Prototype Design

A supervised feed-forward artificial neural network, to learn the patterns of a power network's admittances under normal and fault conditions, is intended to be prototyped. This learned neural network would then be used to detect fault currents in the power network in real time. Thus, the proposed neural network for automatically detecting fault current locations, in real time, would work in two stages. An overview of these stages is explained as follows:

Stage I

Stage I would be the construction and training phase of the neural network. A neural network can be constructed by integrating multiple layers of computational units called as neurons or nodes. These layers of neurons form the input and output layer, with/without hidden layers which are sandwiched between the input and output layer, as shown in Fig. 3.2.

Introduction of hidden layers in the neural network usually increases the precision of its non-linear pattern learning capabilities i.e., the more the number of hidden layers in a neural network, the precise it is in learning complex non-linear features of a specific data set, but at the cost of increased computational complexity. Thus, one must make a trade-off between precision and computational complexity depending on the nature and type of

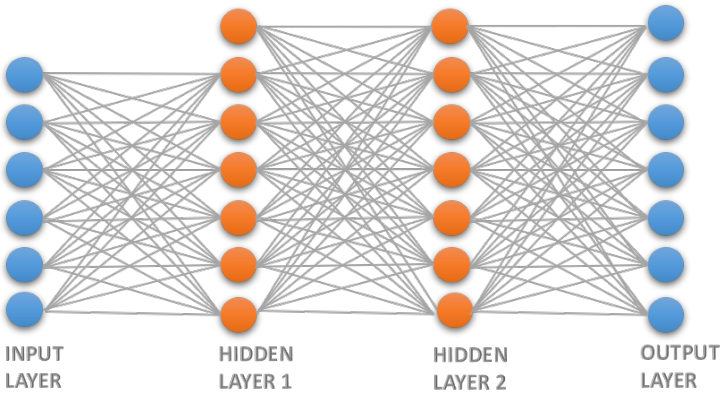


Figure 3.2: Different Layers of Neural Network

application; however, this is not always the case and one needs to experiment with different formations of neural layers, for getting the intended results, for a specific application.

The neurons in the input layer typically store input values whereas, the neurons in the hidden and output layer may perform certain mathematical transformations on these input values for getting the requisite results e.g., each neuron in the hidden and output layer of the neural network may compute its own output by applying a function 'f' to the weighted sum of the received input signals i.e., if a neuron 'a', from a hidden/output layer, receives input-1 with weight-1, input-2 with weight-2 and input-3 with weight-3, its output may be computed as shown in Fig. 3.3.

Remark. *It should be noted here that, in the discussion of the proposed and designed neural network, the input layer is treated as nodes storing values of inputs, whereas the term 'neuron' is being referred to the neurons of hidden and output layer performing certain mathematical computations since there is no hard and fast rule of designing a neural network and a developer may work around those options only as suit his/her application through educated guesses.*

It is pertinent to mention here that the input to a neuron can come from both external sources as well as other nodes in the neural network. Furthermore, the weight to a particular input would initially be assigned randomly and later adjusted in the 'training phase' of the neural network, in such a way as minimizes the difference between the predicted output class and actual output class i.e., the loss function.

In the training phase, the neurons of the first hidden layer would be trained on input vectors along with external input, and each succeeding hidden layer on the outputs from the preceding hidden layer along with external input, to learn the patterns of the

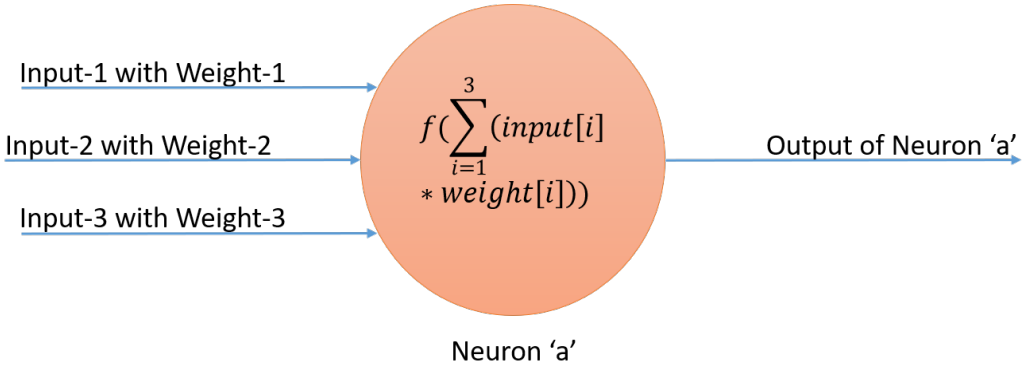


Figure 3.3: Mathematical Computations by a Neuron

Table 3.1: System Admittance Matrix

	Bus 1	**Bus 2**	**Bus x**
Bus 1	4.3 S	2.4 S	2 S
Bus 2	2.4 S	4.4 S	2 S
Bus x	2 S	2 S	3 S

admittances in a power network by adjusting inputs' weights (neural connection weights) in an iterative process for getting the requisite output classes e.g., if a power network has a nominal admittance matrix of the form shown in Table. 3.1:

The learned admittance pattern/output classes (with tolerance), by the neural network, could be of the form shown in Fig. 3.4.

This learning phase of neurons would be supervised, and the neural network would be made to learn certain system admittances as normal admittances and certain other admittances as admittances under fault conditions. Since a branch admittance under fault condition would have different admittance value than its nominal value (by a certain ratio), therefore, the neural network can be made to learn this anomaly, and detect branches with admittance values different than the nominal values by a certain ratio, as the 'faulted' or 'short circuited' branches later in the testing phase (Stage II).

Stage II

Stage II would be the testing phase of the neural network. In this phase, input vectors in real-time would be provided to the trained neural network, which would analyze

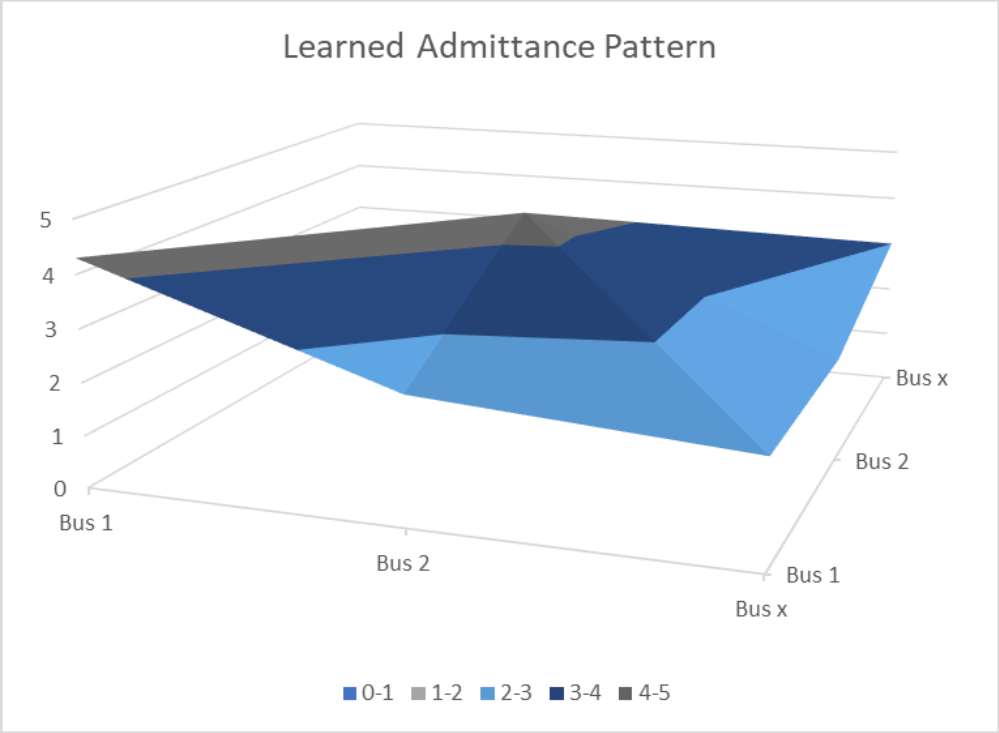


Figure 3.4: Learned Admittance Pattern

and classify them into the learned output classes. These classified outputs would provide information about the locations of faults in the system (if any).

Correspondence b/w Stage I & Stage II

Thus, the proposed solution methodology would work by the sequential correspondence of the afore-mentioned stages. The first stage can be termed as the 'constant stage' since the proposed neural network would require training data for a specific power system, only once. Whereas, the second stage can be termed as the 'variable stage', since it would continuously acquire testing data, every few minutes, from the same power network (on which the neural network has been trained), to locate fault currents in the system (if any), in real time. Fig. 3.5 and Fig. 3.6 provides an overview of the sequential correspondence of the two stages.

Working on the aforementioned concept, the prototype has been implemented as follows.

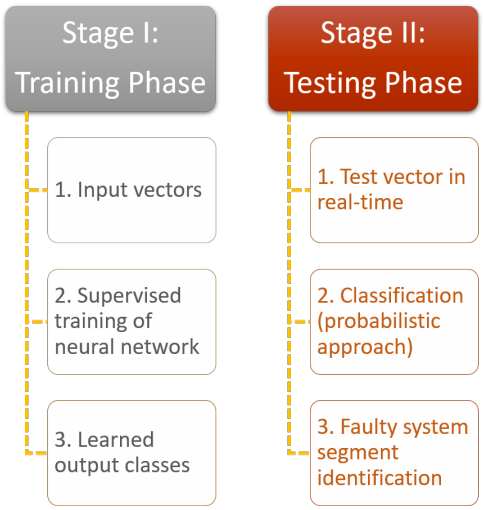


Figure 3.5: Stages of the Proposed Fault Detection Algorithm

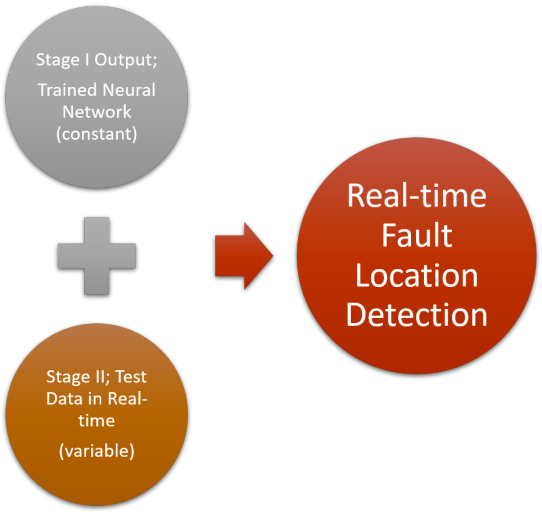


Figure 3.6: Constant and Variable Stages in Fault Location Detection

3.2.2 Prototype Implementation

A three-bus 69kV transmission network has been used as a test case for detecting faults. The data for this network is obtained from [23] in which synchrophasor measurements of bus voltages and current injections are used to extract system Ybus matrix, using parameter estimation technique, as follows:

Extraction of Topology

A summary of the steps followed in [23] for identifying topology of a system are as follows:

1. **Acquisition of Synchrophasors:** The synchrophasor measurements of nodal voltages and branch currents, are obtained from a power transmission company, for a 69kV, three-bus network, the topology of which is required to be determined. These synchrophasor measurements acquired are either obtained from phasor measurement units (PMUs) or the already installed protective relays (with built-in PMU functionality) i.e., current differential relay, distance or impedance relay, overcurrent and earth fault relay etc., in the system, in the form of ".phasor" file format. The ".phasor" files obtained, for an event i.e., a specific duration of time, contains information about the terminal ID (the point where PMU is installed), magnitude and phase angle of corresponding nodal voltage and branch currents, time instant, frequency of the system and rate of change of frequency. This information is converted into ".cvs" file format for utilization purpose.

2. **Calculation of Current Injections:** The current injected into a particular node or bus, in the electrical power transmission system, is calculated from the available synchrophasor measurements of branch currents. For instance, if the bus diagram shown in Fig.3.7 is taken into account, the current injection at node A is equal to branch current at branch 1, however, if PMU is not installed at branch 1 then current injection can be calculated by adding branch currents in branch 2,3 & 4 with reversed polarity. Therefore, one needs to have a PMU installed at every node or most of the branches in the system. However, the presence of protective relays (with built-in PMU functionality), at all the circuits in the system, has exempted one from worrying about the non-availability of a synchrophasor measurement. Moreover, in worst case scenario

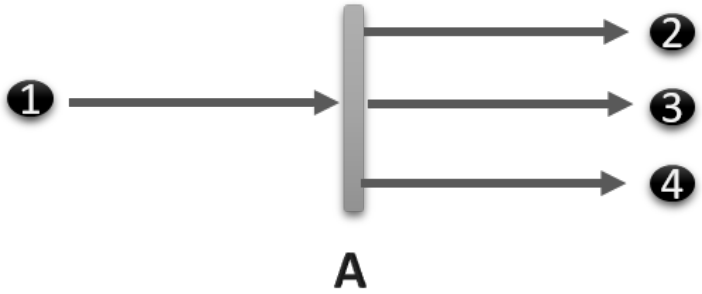


Figure 3.7: Bus 'A' Current Injections

i.e., absence of a synchrophasor measurement for a particular branch, partial current injections can also be calculated and used to extract partial Ybus matrix.

3. **Estimation of Ybus Matrix:** Instead of using an approximation or iterative technique like Newton raphsons method in power flow analysis, this code treats current signals as signals generated from a random process. Since voltage and current elements are related by a cross covariance matrix of impedance, therefore, the voltage signals can also be treated as signals generated from a random process. The cross covariance between the two measurements is given by the formula:

$$Z_{bus} = \frac{1}{M} \sum_{k=1}^{M} V_{\sigma}^{(k)} * \frac{I_{\sigma}^{(k)H}}{\left|I_{\sigma}^{(k)}\right|^2}$$

where,
M denotes the Time Instants Taken
V_{σ} is the Normalized Voltage Matrix
I_{σ} is the Normalized Current Matrix

And Z_{bus} is the respective bus impedance matrix of the power system under study. This Z_{bus} matrix is then used to calculate Y_{bus} matrix at different time instants. The overall process of calculating Ybus matrices statistically at 't_n' time instants, using parametric estimation, is shown in the Fig.3.8.

Using these estimated Ybus matrices, topology of a power network can be extracted, which is explained in the following section.

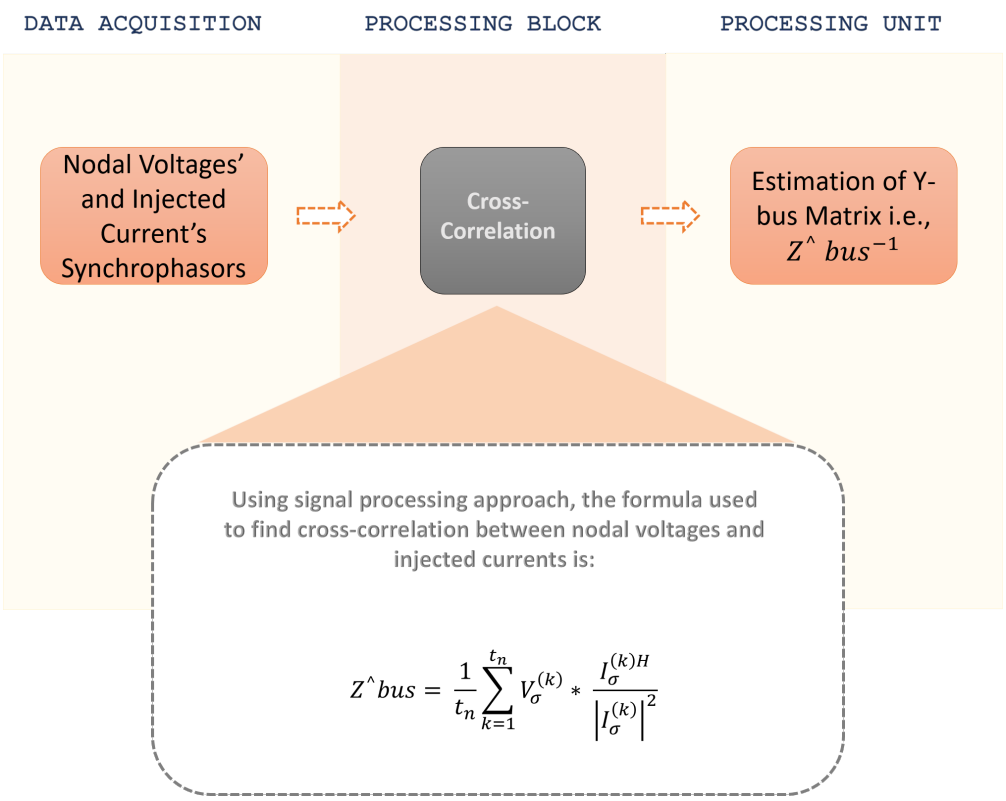


Figure 3.8: Estimation of Ybus Matrix

4. Extraction of Topology:

The Ybus matrices obtained at different time instants i.e., $t_0, t_1, \ldots, t_n$ are shown in Fig.3.9.

The topology of the electrical power network is extracted from these Ybus matrices by analyzing them e.g., if a single Ybus matrix at a time instant t_0 is observed, as shown below (the values of this admittance matrix are taken from the work stated in [23]):

$$Ybus = \begin{bmatrix} 4.087 + 21.010i & 10.249 - 2.381i & -10.309 + 10.723i \\ 10.203 - 2.404i & 2.391 - 0.461i & -3.599 + 2.225i \\ -10.219 + 10.683i & -3.696 + 2.248i & 27.225 + 11.294i \end{bmatrix}$$

It can be seen that all the buses relate to one another through an admittance value, or in other words all the buses are interconnected, in the transmission network being analyzed. Suppose if any of the admittance value i.e., Y12 was '0', it would have shown that bus1 and bus2 are not connected with each other. Thus, the topology of a network can be extracted from its respective Ybus matrix (determined statistically from synchrophasor measurements of bus voltages and injected currents).

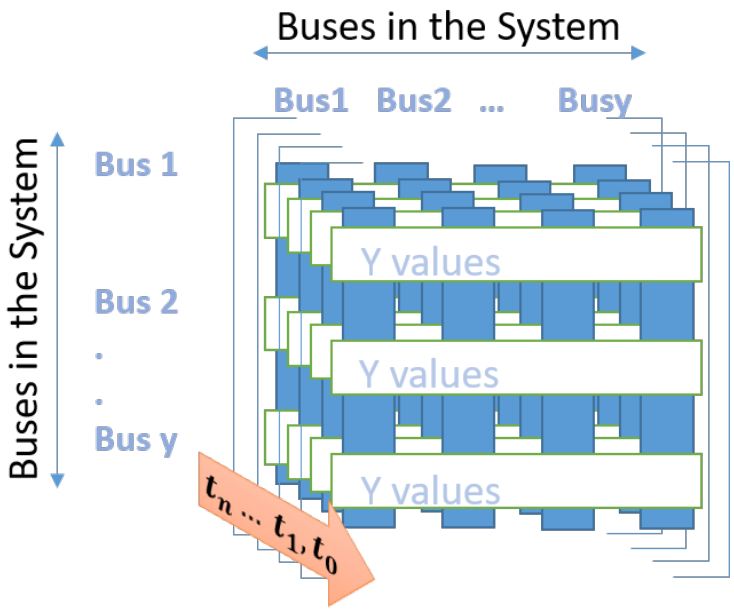


Figure 3.9: Ybus Matrices at Time Instants $t_0, t_1, \ldots, t_n$

Implementation of Artificial Neural Network (ANN)

As stated earlier in this section, the Ybus matrices, shown in Fig.3.9, are converted into one dimensional vectors to be used as inputs to the neural network. This conversion process takes-in Ybus matrices and selects upper triangular elements along with diagonal entries of each Ybus matrix (for simplicity) e.g., if a single Ybus matrix at a particular time instant is taken into account i.e., Y_{t1}, the selected matrix elements to be used, in training the neural network, are as shown in the Fig.3.10.

$$Y_{t1} = \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} \\ Y_{21} & Y_{22} & Y_{23} \\ Y_{31} & Y_{32} & Y_{33} \end{bmatrix}$$

Figure 3.10: System Admittance Matrix at Time Instant t_1

The reason for only selecting upper triangular elements of the admittance matrix is because mutual admittances of the form Y_{ij} are commonly equal to Y_{ji} (where 'i' and 'j' denotes buses in the power network), unless phase shifting transformers are used in the power system (which is rare). In case of phase shifting transformers, all the elements of the bus admittance matrix are required to be considered (for training purpose).

The magnitudes of these upper triangular elements and diagonal elements of each Ybus matrix at a particular time instant, are converted into one dimensional vector as shown in the Fig.3.11. Multiple input vectors are constructed in this way, for multiple time

Figure 3.11: Input Vector to the Neural Network

instants, with each admittance value labeled as either normal '0' or faulty '1'.

The artificial neural network implemented has one input layer, two hidden layers and an output layer for maximized accuracy and precision at minimal computations. There is no specific rule to select the number of hidden layers or the number of neurons in each hidden layer; a programmer is required to play with these parameters for obtaining best results for a particular application, therefore, these values are selected by educated trial and error; furthermore, a minimum of two hidden layers are enough to capture major non-linear features of any dataset. The input layer of the neural network comprises of the aforementioned labeled input vectors. The labeling data is saved in the neural network and each element of the input vector i.e., admittance value, is assigned with a random weight and passed into hidden layer 1 (with seven neurons), along with an external input called as 'bias'. The outputs obtained from hidden layer 1 are passed into hidden layer 2 along with external bias, and the same strategy is adopted for output layer, as illustrated in Fig.3.12.

Each neuron in the hidden layers computes the weighted sum of the received inputs, and doesn't apply any activation function, as shown in the Fig.3.13. The outputs of the neurons in the hidden layers are then propagated in this manner till the final layer i.e., output layer, where an activation function 'f' called as 'Softmax Function' is also applied [24]. The activation function has been applied only once, toward the end, to reduce computational complexity as shown in the Fig.3.14.

Softmax activation function is best suited for multi-class logistic regression. Since there are multiple output classes in my application i.e.,

- 'No Fault'
- 'Fault between bus i and j', where $i, j \in$ buses in the system etc.,

therefore, it is through Softmax activation function applied at the output layer of the neural network which helps the neural network calculate the probabilities of the provided training

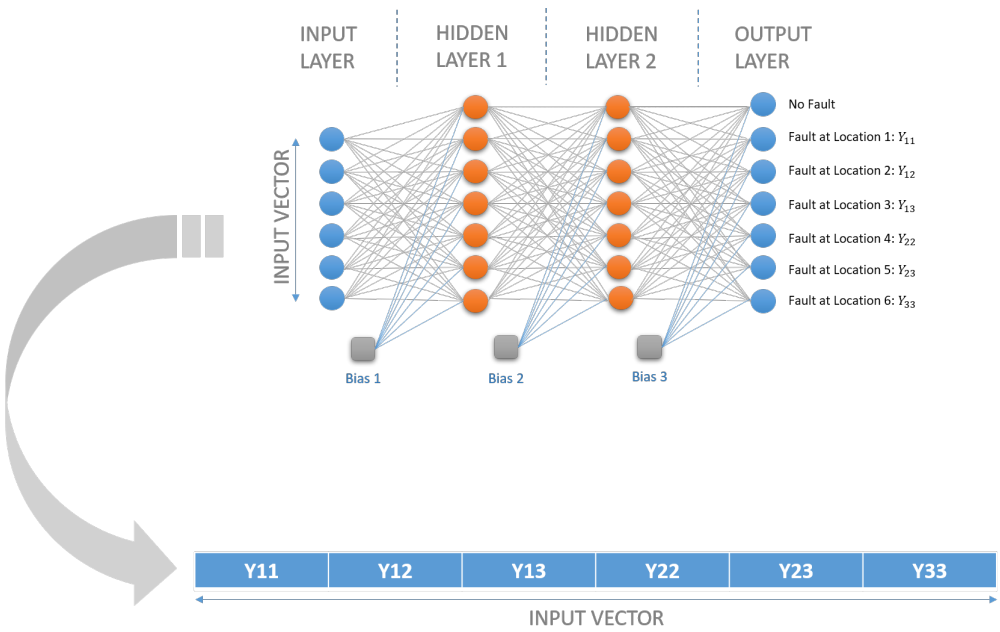


Figure 3.12: Neural Network: Prototype

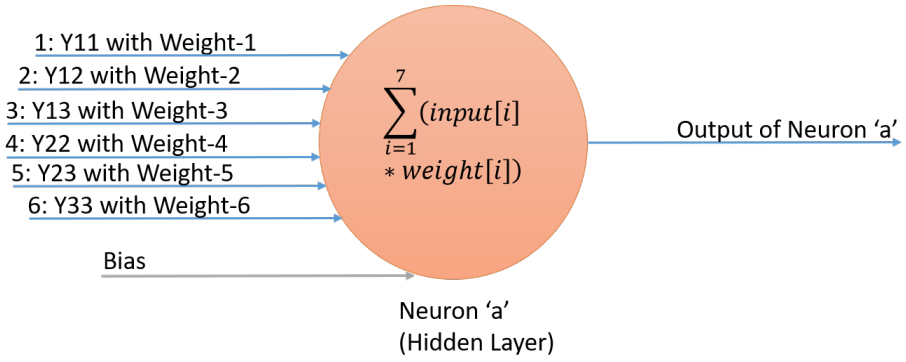


Figure 3.13: Computations by Neurons in the Hidden Layers

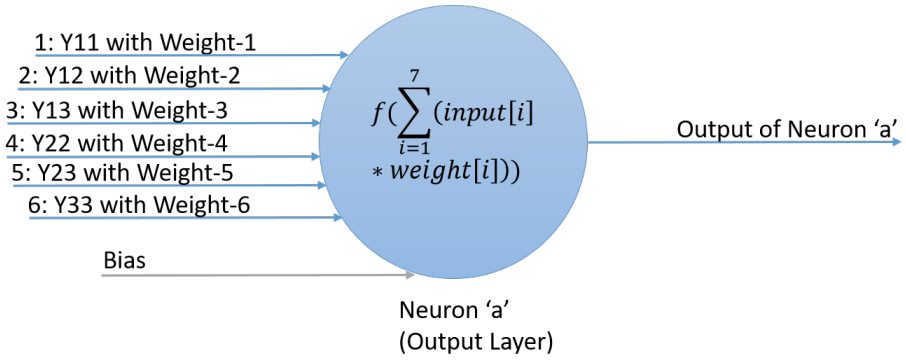


Figure 3.14: Computations by Neurons in the Output Layer

samples to be in a certain output class. The predicted output classes, based on the predicted output probabilities, for the respective input vectors are compared with the requisite output classes of those input vectors, by comparing the predicted probabilities (between 0 and 1) with the saved labeling data (provided earlier). If the predicted results do not match the intended outputs, the network is made to adjust its neural connection weights, through back propagation, in such a way as minimizes the difference between the predicted output classes and requisite or actual output classes. This means that a loss function is computed in terms of internal parameters of the neural network, after each training iteration (or in other words, after providing a batch of input vectors out of the total input vectors available), and tried to be minimized by adjusting these internal parameters of the neural network i.e., weights/bias of the neural connections which were originally assigned randomly.

The loss function used for computing the error between the predicted and actual output classes, in the three-bus network, is 'Cross Entropy'. Cross Entropy loss, also sometimes referred to as 'Log Loss', is best suited for determining the performance of a classification model. Cross Entropy determines the discrepancy between the predicted output classes and the intended or actual output classes, along both axes, in terms of the internal parameters of the neural network i.e., weights and bias and provides a matrix of error values. These error values are transformed into a scalar quantity, denoting loss, by computing their mean (along both the axes) through 'Reduce Mean' function. This computed loss is then tried to be minimized through an optimization technique called as 'Adam Optimization' by adjusting the internal parameters of the neural network. Adam optimization is an extension of classical stochastic gradient descent optimization procedure and usually best suited for large number of data samples and sparse matrices, owing to the following features [25]:

- It is relatively easy to implement and uses a smaller number of computations despite handling large data-sets.
- It is memory efficient and doesn't burden the processor.
- It is not affected by rescaling of computational gradients and works well even in case of noisy or sparse gradients.
- Its parameters can be intuitively adjusted with minimal modifications.
- It combines the perks of two types of optimization techniques i.e., Adaptive Gradient algorithm and Root-Mean-Square propagation.

Remark. *The mathematical formulation and working of Softmax activation function, Cross-Entropy and Adam optimization is described in detail in the section 3.3 of this thesis, describing actual design of the proposed algorithm.*

Besides minimizing the loss value, through optimization, by updating neural network internal parameters i.e., weights and bias, there are some other parameters which are also required to be adjusted for getting precise and accurate results. These different neural network parameters are as follows:

1. **Learning Rate:** It is an important training parameter of the neural network by which the extent of variations in 'weight' and 'bias' can be controlled, after each iteration, in the training phase. It should not be kept too high as to overshoot the ideal weight and bias values or too low as to obstruct algorithm convergence. The typical value chosen for learning rate is around 0.3 and is suggested to be varied by power of 10. For prototyping, this parameter is set to 0.1.

2. **Batch Size:** The number of training vectors (out of the total available training vectors) to be shown to the algorithm at one time, before allowing it to adjust weights of the neural connections, comprise batch size. For prototyping, this parameter is set to 5.

3. **Number of Steps:** Number of steps refers to the number of iterations performed in the neural network to learn system characteristics/patterns etc. It also refers to the number of times, a batch of training data set (with a specified batch size) is provided to the neural network during its training phase. For prototyping, this parameter is set to 100. It is because of the fact that 500 input vectors are available for training the neural network prototype; therefore, 'Number of Steps' should be set to 100 for 'Batch size' of 5, such that:

   $$Batch\ size \times Number\ of\ Steps = 5 \times 100 = 500\ total\ input\ vectors$$

These parameters are adjusted to accurately classify the 500 available input vectors, for 500 time instants, into the seven output classes, for the given three-bus transmission network i.e.,

1. No Fault.

2. Fault at Location 1 = Bus 1 to Gnd.

3. Fault at Location 2 = b/w Bus 1 & Bus 2.

4. Fault at Location 3 = b/w Bus 1 & Bus 3.

5. Fault at Location 4 = Bus 2 to Gnd.

6. Fault at Location 5 = b/w Bus 2 & Bus 3.

7. Fault at Location 6 = b/w Bus 2 & Bus 3.

8. Fault at Location 7 = Bus 3 to Gnd.

This accuracy of the neural network's performance is highly dependent on how the neural network is constructed, for a particular type of application, and the number of samples provided for training the neural network. As stated earlier, in this code, 500 samples have been provided for 500 time-instants and learning rate is set to '0.1', to get maximum accuracy. The code implemented for the prototype detects faults in the transmission system with 99.999% accuracy.

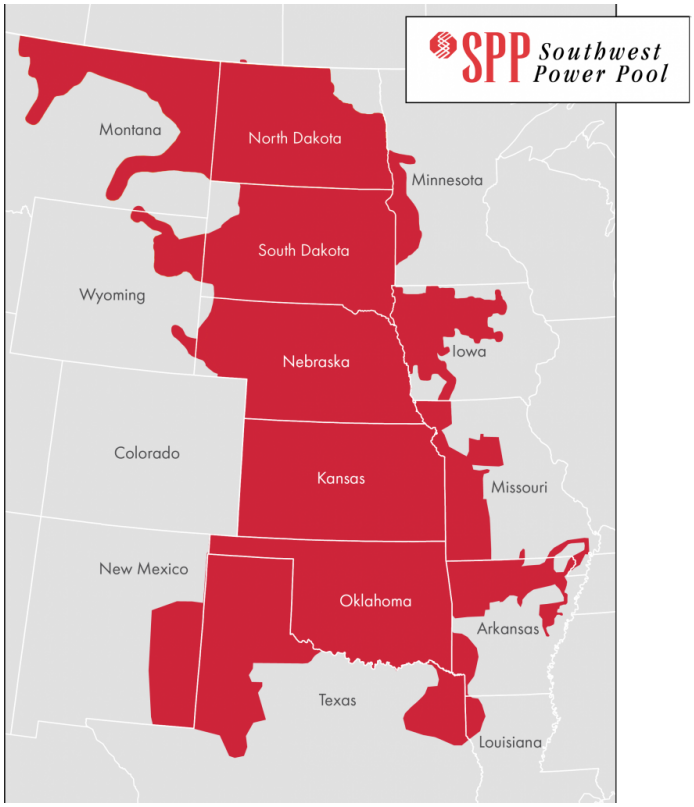


Figure 3.15: Southwest Power Pool [1]

3.3 Proposed Algorithm: Actual Design

The proposed fault location detection algorithm's prototype designed for and tested on three-bus 69kV transmission system earlier, as proof of concept, is now expanded to cover actual 456 bus transmission network under Southwest Power Pool (SPP), following the same approach. The synchrophasor data used for this purpose, is the actual field data of 456 buses obtained from SPP transmission network. Southwest Power Pool (SPP) is one of the nine (9) Regional Transmission Organizations (RTOs) aimed at ensuring reliable supply of electric power at competitive electricity market prices in central states of USA, as shown in the Fig.3.15. The fourteen (14) member states of SPP are Arkansas, Iowa, Kansas, Louisiana, Minnesota, Missouri, Montana, Nebraska, New Mexico, North Dakota, Oklahoma, South Dakota, Texas and Wyoming. The SPP monitors and coordinates power flow among the 14 member states and ensures regional power flow scheduling and network traffic administration at consistent rates, and terms of conditions.

The following paragraphs explain the proposed fault location detection system design process in detail.

3.3.1 Programming Environment

The methodology adopted for implementing the aforementioned proposal involves both matlab and python programming languages. The data pre-processing i.e., acquisition of system admittance matrix in real time and its transformation into neural network input vectors is carried out in matlab programming environment owing to its numerous advantages of handling large matrices over other programming languages. It should be kept in mind here, that the basic data element in matlab is a matrix i.e., a simple integer is treated as a matrix with single row and single column. Moreover, the different mathematical operations on arrays and matrices are built-into matlab environment. Therefore, matlab environment has been used, in the code, for dealing with matrices of complex and intensive electrical power distribution network, to achieve highly optimized and most efficient performance in real-time. Whereas the construction and implementation of artificial neural network for fault location detection in power systems is executed in python environment, since python environment not only has extensive support libraries, user friendly data structures, productivity and agility but also supports third party modules and is open source as well. Conclusively, the output of the code implemented in matlab acts as an input to the code implemented in python

and therefore, a dual combination of the two programming environments serves the requisite purpose.

The same code has also been implemented in python only, to avoid switching between multiple platforms and it has been observed that both of these approaches give the requisite results. While the former approach is programmer friendly, the latter one is user friendly.

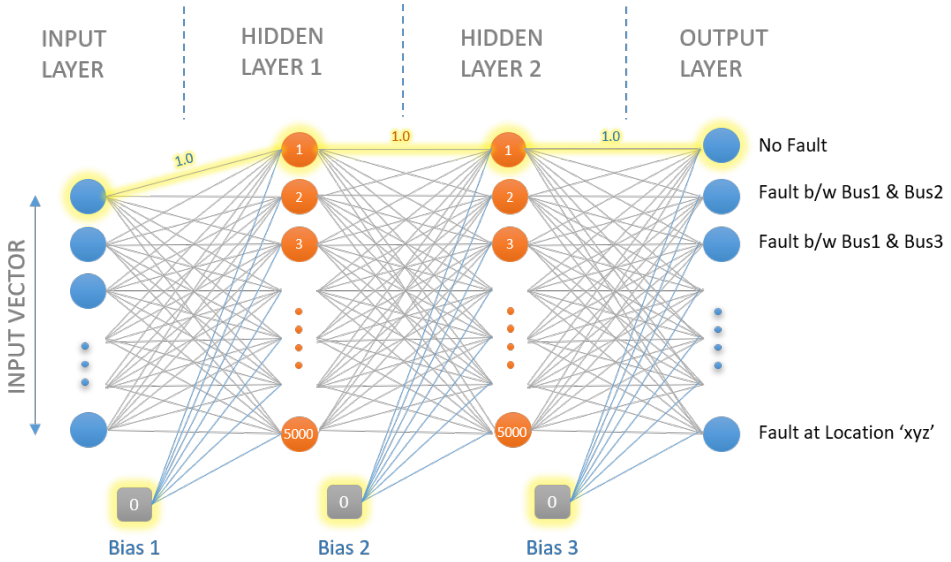
3.3.2 Fault Location Detection

The work presented in [23] is used as a foundation for building upon the concept of fault location detection in power systems using artificial intelligence. Once the power system admittance matrices are extracted statistically, for particular instants of time, from synchrophasor measurements of bus voltages and injected currents (in real time), the proposed fault location detection algorithm's job comes into play. This fault location detection algorithm utilizes artificial intelligence for providing the requisite results. The design and implementation of artificial neural network for detection of faults, in the transmission network of Southwest Power Pool, is presented in the following steps:

Design of Neural Network

Type and Architecture: A supervised feed-forward neural network has been designed for detecting fault locations in the power system. The reason for selecting 'supervised' neural network in place of 'unsupervised' or 'semi-supervised' is to avoid incorrect learned patterns in case of faulty measurement data from the sensors. The neural network designed has only two hidden layers of neurons, sandwiched between the input and output layers. The input layer has neurons equal to the length of input vector, to store each element of the input vector i.e., the admittance value, in a neuron. Since each admittance value of the input vector eventually determines a special output class of fault in the transmission network based on any anomaly in its magnitude, therefore, the output layer has neurons equal to the length of input vector plus one, to include the extra output class denoting 'No Error' in the power transmission network. Similarly, each hidden layer contains 5000 neurons. The number of hidden layers and neurons in each layer is selected through educated hit and trial mechanism since there is no fixed rule to adjust these parameters and one must try different combinations for a particular 'nature' of the problem statement, for coming up with minimized losses and maximized prediction accuracy.

Weights and Bias: The weights of the neural connections between the adjacent layers i.e., input layer, hidden layers and output layer, have been set to '1.0' initially (with '1.0' as the standard deviation and mean value as well), thus giving equal importance to the output of each neuron in the neural network designed. The format of these weights is set to 32 bits floating-point format for ensuring optimal efficiency, fast performance and a memory efficient algorithm. Similarly, each hidden layer and the output layer is assigned with an external bias value of '0.0' (with '0.0' as the standard deviation and mean value as well), to allocate equal importance to all these layers initially. A glimpse of these values is shown in Fig.3.16.



* All of the neural connections have the same properties initially as depicted by the highlighted values

Figure 3.16: Neural Connections' Weights' and Bias' Initialization

Network Parameters: The network parameters are set as follows:

- Learning Rate: The learning parameter of the neural network was initially set to 0.1 (typical value) and since it gave accurate results therefore this value has not been changed.

- Number of Steps & Batch Size: The batch size and number of steps should be chosen in such a way as their multiplication does not exceed the available input vectors for training the neural network i.e.,

  $$batch\ size\ \times\ number\ of\ steps\ \leq\ number\ of\ available\ input\ vectors$$

Therefore, different combinations of number of steps vs batch sizes were tried and tested to achieve maximum accuracy and it was observed that for power transmission systems, the ratio of 2:1 works the best for 'batch size: number of steps', in a neural network, based on the available data or input vectors. Thus, the batch size was set to '100' and number of steps were set to '50' to get optimal results, for the available 5000 input vectors i.e., synchrophasor data of bus voltages and injected currents for 5000 time instants was used to extract 5000 system admittance matrices, for training the neural network, which indicates that the synchrophasor data was obtained for 166.66 seconds or 2.77 minutes (since 30 frames of synchrophasors, or in other words data for 30 instants of time, are provided per second by the Phasor Measurement Units installed in the power system).

The Algorithm: The algorithm has been devised by making each neuron in the neural network calculate weighted sum of the received inputs and only authorizing neurons in the output layer to apply an activation function called as 'Softmax' (once, toward the end). The outputs of the neurons, in the output layer, are evaluated by comparing them with the requisite output classes and the losses in prediction are computed using 'Cross Entropy'. These losses are compiled by taking their mean value and reduced through 'Adam' optimization algorithm.

Working of Neural Network

Data Preprocessing: The system admittance matrices obtained are pre-processed before making the neural network learn admittance pattern. This pre-processing of data is achieved in the following four steps:

1. **Segregation:** The 456x456 system admittance matrices obtained are segregated into upper and lower triangular matrices, and elements of the upper triangular matrices along with diagonal entries are used for training the neural network, assuming absence of phase shifting transformers in the power system under analysis (since in such a scenario an admittance matrix is always symmetric i.e., $Y_{12} = Y_{21}$). In case of phase shifting transformers in the transmission system, all the elements of the admittance matrix are required to be used for training purposes and segregation of data is not required.

2. **Conversion:** The absolute values of the upper triangular elements along with the diagonal entries, of the power system admittance matrices are computed and converted into 1D arrays, with their original indices' numbers preserved in a look up table. This process is further illustrated in the Fig.3.17.

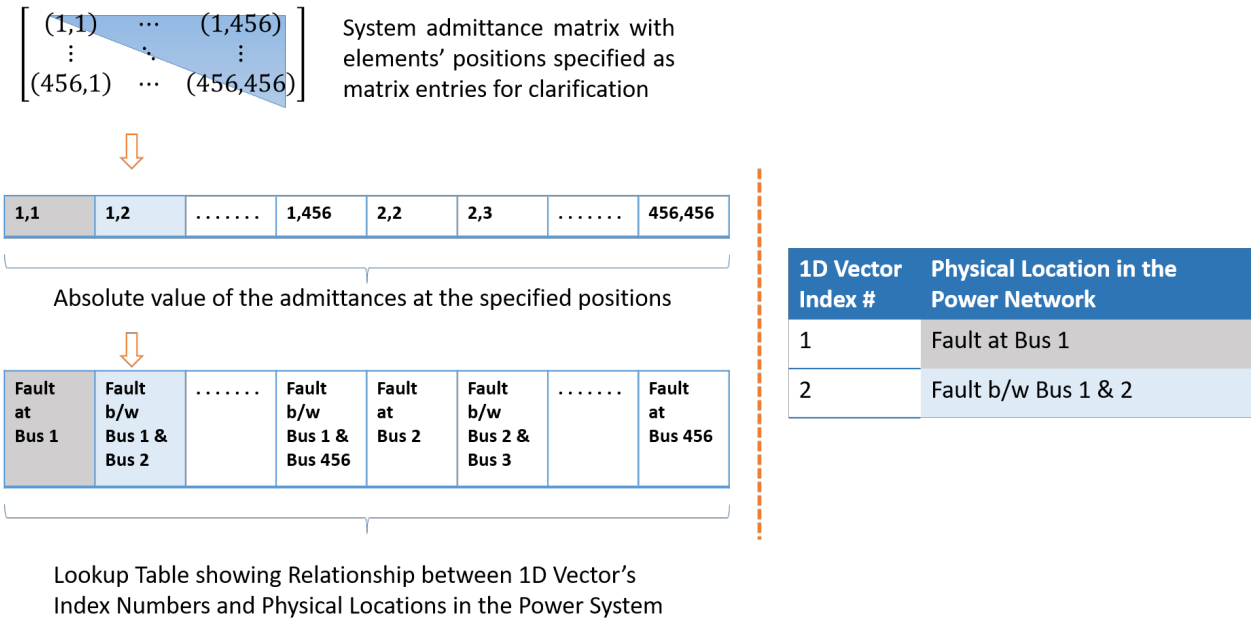


Figure 3.17: Illustration: Conversion of Data

3. **Normalization:** The 1D arrays (called as input vectors in this thesis) obtained from the conversion process are normalized. Each input vector is normalized by dividing it element-wise, with the respective maximum values i.e., since 5000 input vectors are used for training the neural network, therefore, each element of the 1D array or input vector is divided by its respective maximum value out of the 5000 values, along '0' axis (column-wise). For Example if three (3) input vectors i.e., red, blue and green, of four (4) elements each are taken into account, as shown in Table. 3.2:

Table 3.2: Red, Blue and Green Input Vectors with 04 Elements Each

	Col 1: 0 axis	Col 2: 0 axis	Col 3: 0 axis	Col 4: 0 axis
Row 1: 1 axis	2	3	1	2
Row 2: 1 axis	2	4	2	3
Row 3: 1 axis	2	4	3	2

These three vectors would be normalized by dividing elements of 'Col 1: 0 axis' by its maximum entry along '0 axis' i.e., 2, 'Col 2: 0 axis' by its maximum entry along '0 axis' i.e., 4, 'Col 3: 0 axis' by its maximum entry along '0 axis' i.e., 3, and 'Col 4: 0 axis' by its maximum entry along '0 axis' i.e., 3.

After normalization, the input vectors are categorized as follows:

4. **Data Categorization & Labeling:** The pre-processed and normalized 1D arrays, to be used as input vectors, are divided into two categories i.e., training data and testing data in approximately the ratio 0.75:0.25, therefore, out of the 5000 input vectors, 4848 are used as training vectors and the remaining are used as testing vectors as shown in the Fig.3.18.

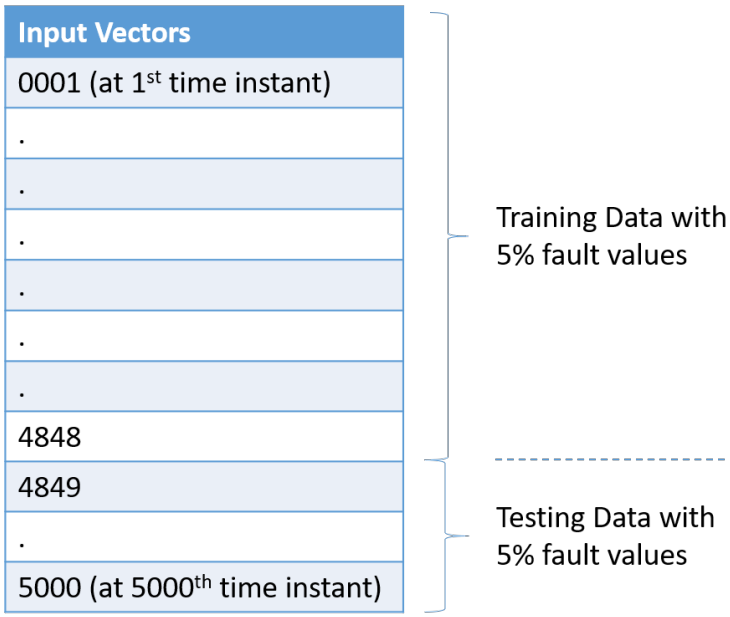


Figure 3.18: Illustration: Data Categorization

The training data is composed of system admittance values under normal conditions as well as fault conditions. Since the neural network designed is supervised to learn certain admittance values as normal admittances and certain as admittances under fault conditions, therefore, input labels are created in python to label each admittance value in the input vector as 'normal entry' or 'faulty entry'. The procedure adopted for labeling training data is that all the admittance values in all of the input vectors are initially labeled as '0.0' (meaning normal), and then faults are introduced in some of the admittance values i.e., approximately 5% data (selected randomly), by

multiplying their magnitudes with a constant number i.e., '3' (imitating over-current between the two buses) and changing their labels to value '1.0' (meaning faulty). As the minimum value of a fault current in a circuit is caused by single phase to ground fault and is approximately 3 times higher than its nominal current value and since $Y \propto I$ $(where\ Y = admittance\ and\ I = Current)$ therefore, any fault or short circuit between any two buses in the transmission system would cause the respective admittance value to go higher than its original magnitude at least by a factor of '3'. The threshold for fault values, set to '3x', can be set to any higher value i.e., 2x, 3x, ..., 10x, since it only helps the neural network better learn about normal data and its variability does not really affect the outcomes.

This labeled data is stored in two separate arrays called as **X** (storing input vectors) and **Y** (storing respective binary labels), provided as input to the neural network, and later used to make the neural network learn and detect the properties of normal and faulty data. The testing data (with fabricated fault values i.e., test case I and actual fault values i.e., test case II) is not labeled and is later being used to test the performance of the neural network designed as shown in Fig.3.19.

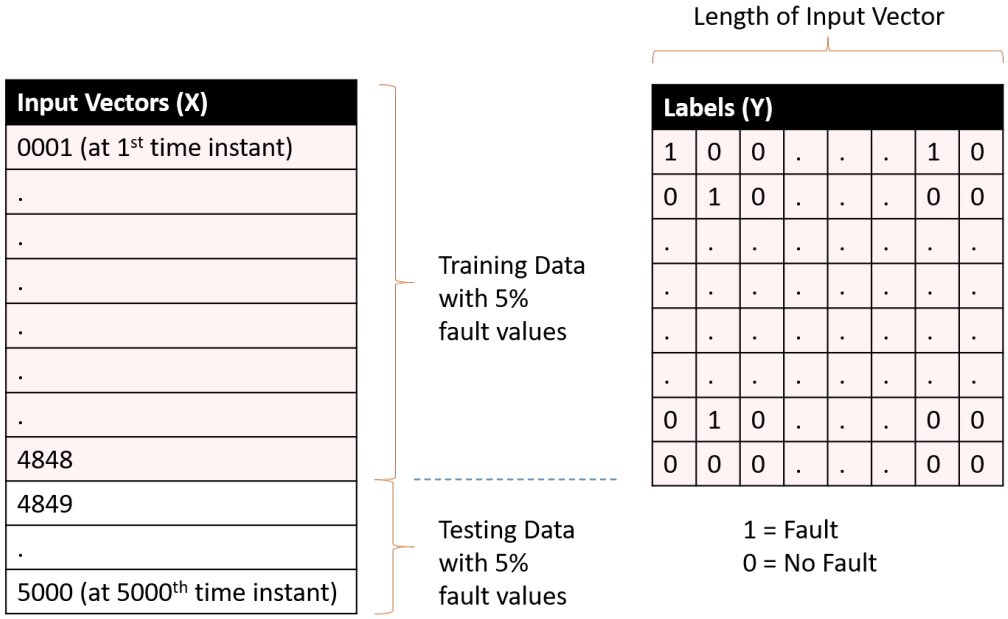


Figure 3.19: Illustration: Input Data **X** and Labels **Y**

.

Learning Process of Neural Network: The labeled training data (**X** and **Y**) is passed as input to the neural network in batch size =100 and number of steps = 50, with 0.1 as the learning parameter 'α' (to be used in optimization algorithm later). The input vectors **X** are made to go through certain computations 'batch-wise' by the neural network whereas, their respective labeling data **Y** is stored as is, by the neural network. These computations are used to determine the error in the predicted outputs and requisite outputs, batch-wise, by comparing the computational results with the already stored respective labels i.e., **Y**. And the total error for a batch, calculated in terms of internal parameters of the neural network, is tried to be minimized by adjusting the internal parameters of neural network i.e., neural connection weights, after each batch's computations (100th iteration in this case), through an optimization algorithm. This process is continued for all the subsequent batches of the input data **X**, till the neural network is trained to provide accurate/requisite results. Thus, the total error while training the neural network is minimized through back propagation method.

In the process of computations performed on X by the neural network, each input vector (from the provided batch of input vectors) is propagated through a series of steps where each neuron in the neural network computes weighted sum of the received input vector along with external input, with the neurons in the output layer applying Softmax activation function as well. These series of steps are as follows:

1. **Step I: Hidden Layers' Computations:** The neural network receives n-dimensional input vector ('x') from the provided batch of '**X**', and its respective n-dimensional labeling data 'Y', and neurons of each hidden layer compute the weighted sum of the received inputs independently, explained as follows:

   Let 'x' be the n-dimensional 1D input vector composed of power system admittances i.e.,

   $x = [x_1 \; x_2 \; x_3 \; \ldots \; x_n]_{1\times n}$

   $$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,m} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,m} \\ w_{n,1} & w_{n,2} & \ldots & w_{n,m} \end{bmatrix}_{n\times m}$$

   where,

   n is the number of elements comprising diagonal and upper triangular elements of

power system admittance matrix.

m is the number of neurons in the first hidden layer.

$w_{i,j}$ is the neural connection weight between element 'i' of the input vector and neuron 'j of the first hidden layer.

$B1 = constant \ \times \ [unity \ matrix]$, since the code implemented uses a constant bias value for each hidden layer.

Then, collective output of the neurons of the first hidden layer can be represented as:
$Z' = (X.W)^T + B1$

$$Z' = \begin{bmatrix} z'_1 \\ z'_2 \\ z'_m \end{bmatrix} = \begin{bmatrix} x_1.w_{1,1} + x_2.w_{1,2} + \ldots + \ x_n.w_{1,m} \\ x_1.w_{2,1} + x_2.w_{2,2} + \ldots + \ x_n.w_{2,m} \\ x_1.w_{n,1} + x_2.w_{n,2} + \ldots + \ x_n.w_{n,m} \end{bmatrix}_{m\times 1} + \ b1 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}_{m\times 1}$$

where, z'_j = output of j^{th} neuron in the first hidden layer.

Similarly, the computed output Z'' of the second hidden layer, with $W\prime$ as the respective matrix of neural connection weights, is: $Z'' = [(Z']^T .W')^T + B2$

$$Z'' = \begin{bmatrix} z''_1 \\ z''_2 \\ z''_o \end{bmatrix} = \begin{bmatrix} z'_1.w'_{1,1} + z'_2.w'_{1,2} + \ldots + \ z'_m.w'_{1,o} \\ z'_1.w'_{2,1} + z'_2.w'_{2,2} + \ldots + \ z'_m.w'_{2,o} \\ z'_1.w'_{m,1} + z'_2.w'_{m,2} + \ldots + \ z'_m.w'_{m,o} \end{bmatrix}_{o\times 1} + b2 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}_{o\times 1}$$

where,

m=number of neurons in the first hidden layer.

o=number of neurons in the second hidden layer.

$w_{k,l}$ = neural connection weight between neuron 'k' in the first hidden layer and neuron 'j' in the second hidden layer.

$B2 = constant \ \times \ [unity \ matrix]$, since the code implemented uses a constant bias value for each hidden layer.

z''_l = output of l^{th} neuron in the second hidden layer e.g., if l = 1 then,

$z''_1 = \ z'_1.w'_{1,1} + z'_2.w'_{1,2} + \ldots + \ z'_m.w'_{1,o} + b2$

$$z''_1 = \left(x_1.w_{1,1} + x_2.w_{1,2} + \ldots + \ x_n.w_{1,m} + b1\right).w'_{1,1}$$
$$+ \left(x_1.w_{2,1} + x_2.w_{2,2} + \ldots + \ x_n.w_{2,m} + b1\right).w'_{1,2} + \ldots$$
$$+ \left(z'_1.w'_{m,1} + z'_2.w'_{m,2} + \ldots + \ z'_m.w'_{m,o} + b1\right).w'_{1,o} + b2$$

2. Step II: Output Layer's Computations:

The outputs of the second hidden layer (Z'') again get multiplied with their respective neural connection weights and are summed by the final layer i.e., output layer, along with the respective bias value for the output layer (the same mathematical computations as that of hidden layer 1 and 2). However, besides calculating weighted sum of received inputs (let it be Z'''), the output layer also applies Softmax activation function on Z''', to calculate probability of the n-dimensional input vector x to be in a certain output class c; explained as follows:

Let Z''' be the weighted sum at the output layer. where, $Z''' = [(Z'']^T .W'')^T + B3$

$$Z''' = \begin{bmatrix} z'''_1 \\ z'''_2 \\ z'''_c \end{bmatrix} = \begin{bmatrix} z''_1.w''_{1,1} + z''_2.w''_{1,2} + \ldots + z''_o.w''_{1,c} \\ z''_1.w''_{2,1} + z''_2.w''_{2,2} + \ldots + z''_o.w''_{2,c} \\ z''_1.w''_{o,1} + z''_2.w''_{o,2} + \ldots + z''_o.w''_{o,c} \end{bmatrix}_{c\times 1} + b3 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}_{c\times 1}$$

where,

c =number of output classes or nodes in the output layer.

o =number of neurons in the second hidden layer.

$w_{q,r}$ = neural connection weight between neuron q in the second hidden layer and neuron r in the output layer.

$B3 = constant \times [unity\ matrix]$, since the code implemented uses a constant bias value for output layer.

z'''_r = output of rth neuron in the output layer e.g., if r = 1 then,

$$z'''_1 = z''_1.w''_{1,1} + z''_2.w''_{1,2} + \ldots + z''_o.w''_{1,c} + b3$$

$$z'''_1 = \left(z'_1.w'_{1,1} + z'_2.w'_{1,2} + \ldots + z'_m.w'_{1,o} + b2\right).w''_{1,1}$$
$$+ \left(z'_1.w'_{2,1} + z'_2.w'_{2,2} + \ldots + z'_m.w'_{2,o} + b2\right).w''_{1,2} + \ldots$$
$$+ \left(z\prime_1.w\prime_{m,1} + z\prime_2.w\prime_{m,2} + \ldots + z'_m.w'_{m,o} + b2\right).w''_{1,c} + b3$$

The Softmax function 'f' takes in this c-dimensional vector Z''' and outputs a c-dimensional vector Z'^v of real values between 0 and 1 indicating probability distribution of Z''' across the 'C' number of output classes.

Softmax activation function is an extension of sigmoid unipolar activation function (mentioned earlier in section 1.4 of this thesis), for more than two output classes. In

sigmoid unipolar activation function, only two output classes are considered i.e., c=0 or c=1, with an aim to correctly predict an output class for the given input Z''' through a probability distribution function described by [24]:

$$Z'^{v} = f\left(Z'''\right) = \frac{1}{1+e^{-Z'''}}$$

$$P\left(0|Z'''\right) = Z'^{v} = f\left(Z'''\right) = \frac{1}{1+e^{-Z'''}}$$

$$P\left(0|Z'''\right) = 1 - Z'^{v} = 1 - f\left(Z'''\right) = 1 - \frac{1}{1+e^{-Z'''}} = \frac{e^{-Z'''}}{1+e^{-Z'''}}$$

The same concept is extended to cover multiple output classes of the type:

- 'No Fault'
- 'Fault at Bus i',where i $\in$ buses in the power system
- 'Fault between Bus i and j',where i,j $\in$ buses in the power system

through Softmax activation function i.e., the value at each index of Z'^{v} refers to the probability of Z''', to be in the output class 'c' represented by that index, explained as follows:

For simplification, let $Z''' = z$ *and* $Z'^{v} = y$, where 'y' is a c-dimensional vector, used in place of Z'^{v} that should not be confused with the capital '$\mathbf{Y}$' (for denoting data labels of $\mathbf{X}$).

The Softmax function 'f' is a normalized exponential function and provides the probability distribution (y) of 'z' across all the classes, as follows:

$$f(z)_c = y_c = \frac{e^{z_c}}{\sum_{i=1}^{C} e^{z_i}}$$

z_c is pre-softmax computed output of the c^{th} neuron or node in the output layer
c $\in$ {1,2,3,...,C} i.e., the output classes in the neural network

The summation factor in the denominator is used for normalization so that the sum of all the probabilities obtained for all the output classes, for a single input vector, is 1 i.e., $\sum_{i=1}^{C} y_i = 1$

The same probability distribution function can also be written as:

$$\begin{bmatrix} P(c=1|z) \\ \cdots \\ P(c=C|z) \end{bmatrix} = \begin{bmatrix} \varsigma(z)_1 \\ \cdots \\ \varsigma(z)_C \end{bmatrix} = \frac{1}{\sum_{i=1}^{C} e^{z_i}} \begin{bmatrix} e_1^z \\ \cdots \\ e_C^z \end{bmatrix}$$

Once the probabilities (y or Z'^{v}) of the input vector 'x', between numbers ranging from 0 to 1, for each output class are obtained at the output layer, they are compared with the respective labeling data 'Y' provided to determine if the probabilities computed for 'x' are in accordance with the provided labels as shown in the Fig.3.20.

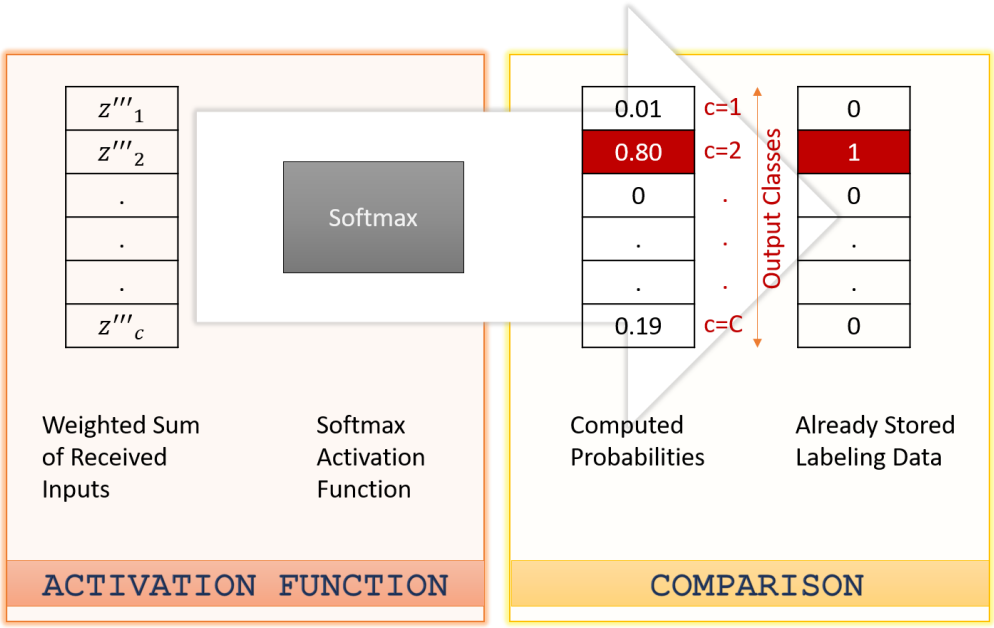


Figure 3.20: Output Layer's Computations

The error in predicted class (shown by the computed probabilities) and actual output class (shown by the provided labels) is determined through a loss function called as 'Cross Entropy'. The Cross Entropy loss is a log function used to determine error in the predicted probabilistic output and actual output, this function decreases with an increase in the accuracy of the predicted probabilistic output.

As mentioned earlier in section 1.4 of this thesis, the Cross Entropy loss determined for multiple, independent, two-class classification is the same as that determined in case of two output classes, except for the fact that for multiple, independent, two-class classification, the total Cross Entropy error is the sum of individual errors determined for each of the independent two-class classification output, given by the formula [26]:

$$-\sum_{c=1}^{C}\left(\mathbf{Y}_c log\left(y_c\right)+\left(1-\mathbf{Y}_c\right) log\left(1-y_c\right)\right)$$

where,

C represents the total number of output classes.

$\mathbf{Y}_c$ is the stored label (already provided to the neural network before its training phase) i.e., '0' (indicating that input 'z' is not in the output class 'c') or '1' (affirmative), and, y_c is the predicted probability of input 'z', between 0 and 1, to be in the output class 'c'.

The aforementioned steps are repeated for each input vector of the batch. The output probabilities obtained for all the input vectors in the batch of $\mathbf{X}$ if not equal to the already stored labeling data '$\mathbf{Y}$', are adjusted through back propagation by changing the neural connection weights, initially assigned as 'unity'. The neural connection weights are adjusted, through an optimization algorithm i.e., Adam optimization, in proportion to the discrepancy between the predicted output classes and actual output classes for the given batch of $\mathbf{X}$ in such a way as minimizes the total error content. This total error content or discrepancy is determined by taking mean of all the errors computed individually through Cross Entropy for all the input vectors in a batch of $\mathbf{X}$. The sequence of these steps is illustrated in the Fig.3.21.

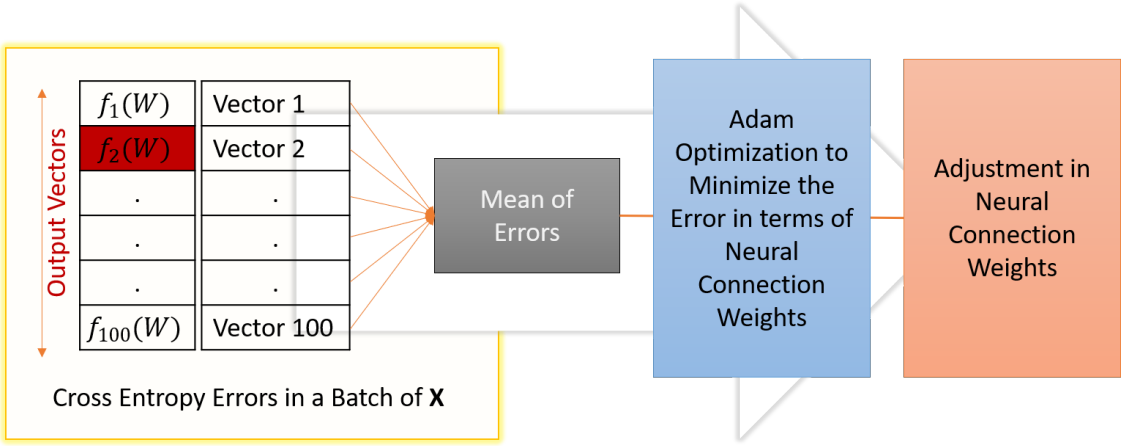


Figure 3.21: Working of Neural Network, taken a Batch of $\mathbf{X}$

3. Step III: Optimization:

The optimization algorithm used to minimize the total error content is called as Adam optimization.

As mentioned earlier in section 3.2.2 of this report, Adam optimization technique is an extension of gradient descent optimization technique but it uses adaptive learning rate, based on the data provided, instead of fixed learning rate for minimizing the total error content with respect to neural connection weights, during the training process. It changes its adaptive learning parameter on run time based on the statistics of the provided data i.e., mean or first moment estimate, and variance or second moment estimate. An overview of the algorithm is as follows [27]:

Algorithm 1 Adam Optimization

while w_t *converges* **do**

$t = t + 1$ ▷ Increment time step

$g_t = \nabla_w f_t(w_{t-1})$ ▷ Compute gradient of the objective function

$m_t = \beta 1.m_{t-1} + (1 - \beta 1)g_t$ ▷ Update biased first moment estimate

$v_t = \beta 2.v_{t-1} + (1 - \beta 2){g^2}_t$ ▷ Update biased second moment estimate

$\hat{m}_t = m_t/(1 - \beta 1^t)$ ▷ Calculate bias-corrected first moment estimate

$\hat{v}_t = v_t/(1 - \beta 2^t)$ ▷ Calculate bias-corrected second moment estimate

$w_t = w_{t-1} - \alpha \hat{m}_t \left(\sqrt{\hat{v}_t} + \varepsilon\right)$ ▷ Update parameters (neural connection weights)

return w_t ▷ Neural connection weights

In this algorithm:

- f(w) is the objective function or the total error function for a batch of **X**, required to be minimized with respect to neural connection weights 'w'.
- gt is the gradient of the objective function i.e., gradient of the total error function for a batch of **X**, with respect to neural connection weights 'w'.
- All vector operations are performed element-wise.
- The default setting of parameters used in Adam optimization for most of the machine learning problems, is kept as is and mentioned as follows:

  $\alpha = 0.1 =$ *Learning rate or step size*

  $\beta 1 = 0.9 =$ *Exponential decay rate for first moment estimate i.e., mean*

  $\beta 2 = 0.999 =$ *Exponential decay rate for second moment estimate i.e., variance*

  $\varepsilon = 10^{-8} =$ *A very small number for preventing division by zero in the algorithm*

- Likewise, the parameter vectors are initialized as follows:

$$w_0 = 1.0 = Initial\ weights\ of\ the\ neural\ connections$$
$$m_0 = 0 = Initial\ value\ of\ first\ moment\ vector$$
$$v_0 = 0 = Initial\ value\ of\ second\ moment\ vector$$
$$t = 0 = Initial\ value\ of\ time\ step$$

The working of the algorithm is explained in algorithm 1.

The aforementioned steps are repeated for every batch of $\mathbf{X}$, with the initial values of neural connection weights for each subsequent batch of $\mathbf{X}$ set to the convergent values obtained for the former batch of $\mathbf{X}$ in Adam optimization, till the last vector of the last batch of $\mathbf{X}$ i.e., the end of the available training data set, as illustrated in the Fig.3.22.

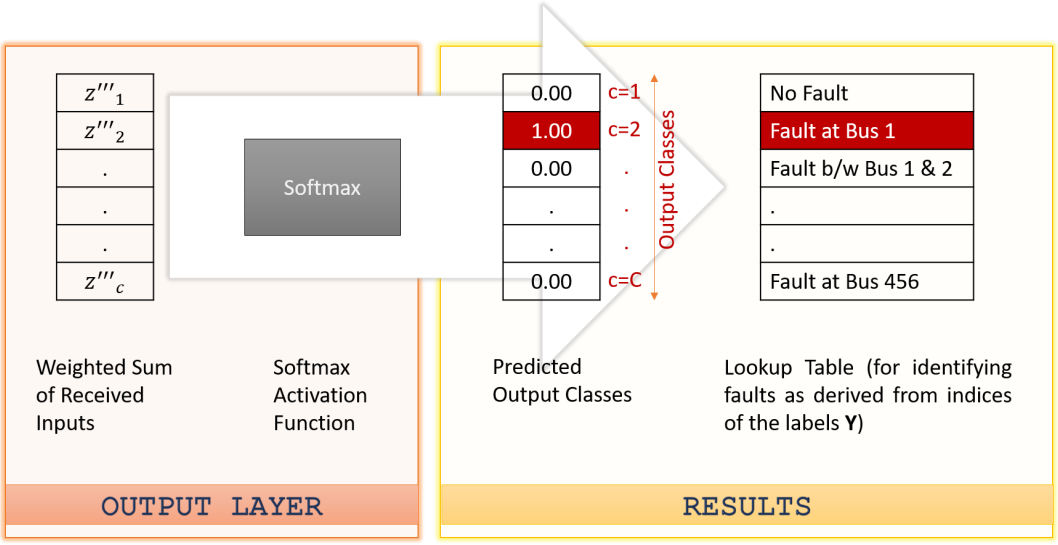


Figure 3.22: Working of Neural Network, taken all Batches of $\mathbf{X}$ for Training

As shown in Fig.3.22, the resulting probabilities of the output classes determines the faulty segment of the transmission network, which is identified by looking at the respective index values of the output vector e.g., the 2^{nd} index of the output vector with fault probability of 100% represents a fault at Bus 1 in the transmission system.

4 CHAPTER: TESTING & RESULTS

The feed-forward supervised neural network once trained on 456 bus SPP transmission network training data is tested using two types of testing data-sets. The first data-set is composed of 152 input vectors having normal admittances and admittances under fabricated fault conditions. Whereas, the second data-set is composed of data (10 input vectors) with actual outage events, which occurred on the transmission lines under Southwest Power Pool (SPP), in the year 2015.

These data sets were acquired by the trained neural network as inputs and were made to go through the hidden layers' computations and the output layer's computations with Softmax activation function. The probabilities obtained, at the output, for these input data-sets to be in certain output classes determine the faulty portions of these data-sets. The system setup with server and software specifications, benchmark data, performance metrics, performance evaluation, analysis of results with effect of certain neural network parameters like learning rate, number of neurons and number of hidden layers, and method for fault classification along with exact distance calculation is as follows:

Table 4.1: Server Specifications

Processor	Intel Core i7 4.00 GHz
Cores	4-Core
RAM	32 GB
Architecture	64-Bit
GPU	NVIDIA GeForce 940M

4.1 System Setup

The specifications of the test-bench used for detecting fault location, in real time, are as follows:

4.1.1 Server Specifications

The server used for training the neural network and detecting fault locations, is a common personal computer with specifications mentioned in Table. 4.1:

4.1.2 Software Toolkit

The specifications of the software toolkit used for designing the algorithm are mentioned in Table. 4.2:

Table 4.2: Software Toolkit

Programming Language	Python
Software Version	Python 3.6.4
Libraries	**Tensor Flow** (Free & open-source library used for machine learning algorithms) **Numpy** (for multi-dimensional arrays and matrices) **Pandas** (for data manipulation and analysis) **Scipy** (for scientific and technical computing) **Matplotlib** (for plotting graphs)

4.2 Benchmark Data

The synchrophasor data for training and testing the neural network, for fault location detection, has been obtained from the 456-bus transmission network of Southwest Power Pool. The data set contains system admittances, acquired statistically as per the work mentioned in [23], for the 456-bus network, at different time instants. The system admittance matrices acquired at different time instants are converted into 1D input vectors and divided into two categories i.e., training data and testing data. The training data thus acquired is used to train the neural network, whereas the testing data is used to test the performance of the trained neural network using different performance metrics.

4.3 Performance Metrics

Five categories, of performance metric, are used to check the validity of the trained machine learning algorithm i.e., neural network, which are as follows:

1. **Accuracy** (total number of correct predictions): It is a performance metric used to determine the accuracy of the number of the predicted output classes, during training phase. Accuracy is determined by equating predicted probabilities of the output classes with the respective labels **Y** to get an array of 0's and 1's, taking sum of the elements of the resulting array and dividing this sum value with the sum of the respective labels **Y**, for each of the last input vector in the provided batch of input vectors **X** i.e.,
   $\%Accuracy = \frac{\left(\sum_{Output\ \ Classes} If\ \ Predicted\ \ Class == \ Respective\ \ Label\right) \times 100}{\sum_{Output\ \ Classes} Respective\ \ Label}$
   Thus, accuracy determines the number of accurately predicted classes.

2. **False Positive** (a label of 1 is predicted for an output class but the true label is 0): Unlike accuracy which determines the validity of the total number of predicted faults, False Positive is used to check the number of times a predicted fault doesn't match its respective label, since it might be possible that the total number of faults and the total number of faulty labels are equal whereas, a predicted fault does not match its respective label. Thus, False Positive generates a flag whenever an output class has been termed faulty (1), but the respective label is not faulty (0). Ideally, the value of False Positive should be zero, indicating no false alarms or class-wise accuracy of the predicted classes.

3. **False Negative** (a label of 0 is predicted for an output class but the true label is 1): Just like False Positive, False Negative works the opposite and generates a flag whenever an output class has been termed normal/without fault (0), but the respective label is faulty (1). Ideally, the value of False Negative should be zero, indicating no false alarms or class-wise accuracy of the predicted classes.

4. **True Positive** (a label of 1 is predicted for an output class and the true label is also 1): Unlike accuracy which determines the validity of the total number of predicted faults, True Positive is used to check the number of times a predicted fault matches its respective label, since it might be possible that the total number of faults and the total number of faulty labels are equal whereas, a predicted fault does not match its

respective label. Thus, True Positive generates a flag whenever an output class has been termed faulty (1), and the respective label is also faulty (1). Ideally, the value of True Positive should be equal to the total number of faults in the testing data, indicating no false alarms or class-wise accuracy of the predicted classes.

5. **True Negative** (a label of 0 is predicted for an output class and the true label is also 0): Just like True Positive, True Negative works the opposite and generates a flag whenever an output class has been termed normal/without fault (0), and the respective label is also normal/without fault (0). Ideally, the value of False Negative should be equal to the total number of non-faulty values in the testing data, indicating no false alarms or class-wise accuracy of the predicted classes.

4.4 Performance Evaluation

The following test cases have been tried to detect faults:

4.4.1 Test Case I

In this test case, the testing data has been acquired from normal system data (1D input vectors i.e., X). One hundred and fifty-two (152) input vectors are taken and fifty (50) faults have been introduced deliberately at random locations. These faults have been introduced by changing the magnitude of certain admittances to a value greater than the nominal value (imitating the case of over-current between the two buses) and the labels from '0' (No Fault) to '1' (Fault). These faulty input vectors are given to the trained neural network as inputs. The neural network performs the same classification operations on these vectors i.e., the testing data, as discussed earlier in section 3.3.2 and correctly identifies all of these 50 fault locations.

The results obtained for performance metrics are shown in Table. 4.3:

The fifty (50) fault locations, displayed on the screen are shown in Fig.4.1, which are exactly in accordance with the introduced fault locations.

Similarly, the graphs plotted between the predicted fault locations and actual fault locations for four (4) of the one hundred and fifty-two (152) provided test data vectors are shown in Fig.4.2, Fig.4.3, Fig.4.4 and Fig.4.5.

```
fault is between Bus [204] & [225]
fault is between Bus [82] & [200]
fault is between Bus [219] & [362]
fault is between Bus [212] & [221]
fault is between Bus [193] & [362]
fault is between Bus [205] & [219]
fault is between Bus [192] & [202]
fault is between Bus [227] & [232]
fault is between Bus [168] & [245]
fault is between Bus [351] & [351]
fault is between Bus [202] & [221]
fault is between Bus [244] & [246]
fault is between Bus [226] & [228]
fault is between Bus [227] & [245]
fault is between Bus [377] & [385]
fault is between Bus [443] & [446]
fault is between Bus [136] & [178]
fault is between Bus [160] & [199]
fault is between Bus [168] & [227]
fault is between Bus [283] & [345]
fault is between Bus [214] & [226]
fault is between Bus [214] & [219]
fault is between Bus [195] & [236]
fault is between Bus [168] & [202]
fault is between Bus [119] & [220]
fault is between Bus [362] & [450]
fault is between Bus [274] & [345]
fault is between Bus [246] & [268]
fault is between Bus [386] & [390]
fault is between Bus [136] & [236]
fault is between Bus [196] & [234]
fault is between Bus [219] & [342]
fault is between Bus [185] & [188]
fault is between Bus [159] & [287]
fault is between Bus [206] & [234]
fault is between Bus [221] & [276]
fault is between Bus [202] & [208]
fault is between Bus [361] & [361]
fault is between Bus [256] & [276]
fault is between Bus [433] & [433]
fault is between Bus [117] & [118]
fault is between Bus [236] & [256]
fault is between Bus [382] & [384]
fault is between Bus [452] & [456]
fault is between Bus [218] & [251]
fault is between Bus [179] & [179]
fault is between Bus [225] & [395]
fault is between Bus [210] & [232]
fault is between Bus [238] & [256]
fault is between Bus [228] & [230]
```

Figure 4.1: 50 faulty Locations

Table 4.3: Performance Metrics Results: Test Case I

Performance Metric	**Results**	**Description**
Accuracy	100%	Accurate
False Positive	0	0 False Alarms
False Negative	0	0 False Alarms
True Positive	50	50 Faulty Locations
True Negative	774998	152 vectors of length 5099, with 50 fault locations: $\implies 50 + 774998 \approx 5099 * 152$

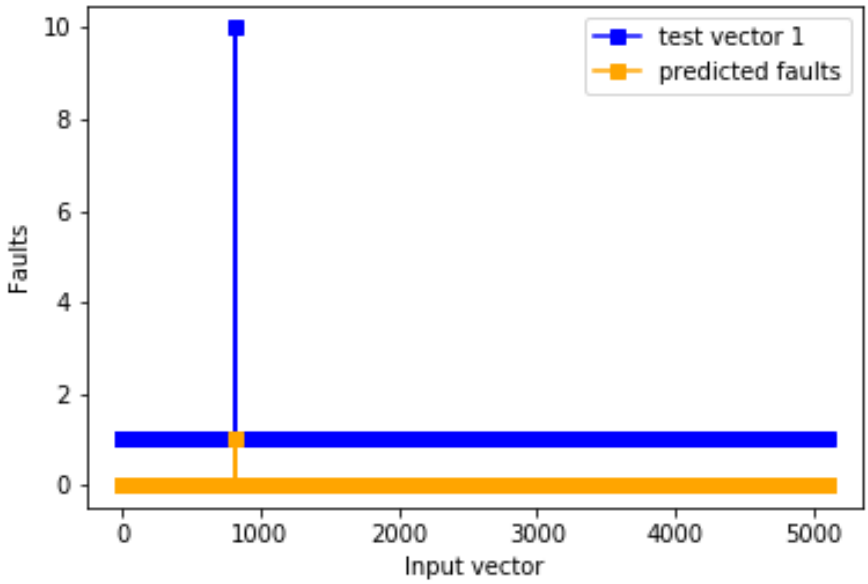


Figure 4.2: Test Vector at 6^{th} Time Instant, Fault at 815^{th} Index Value

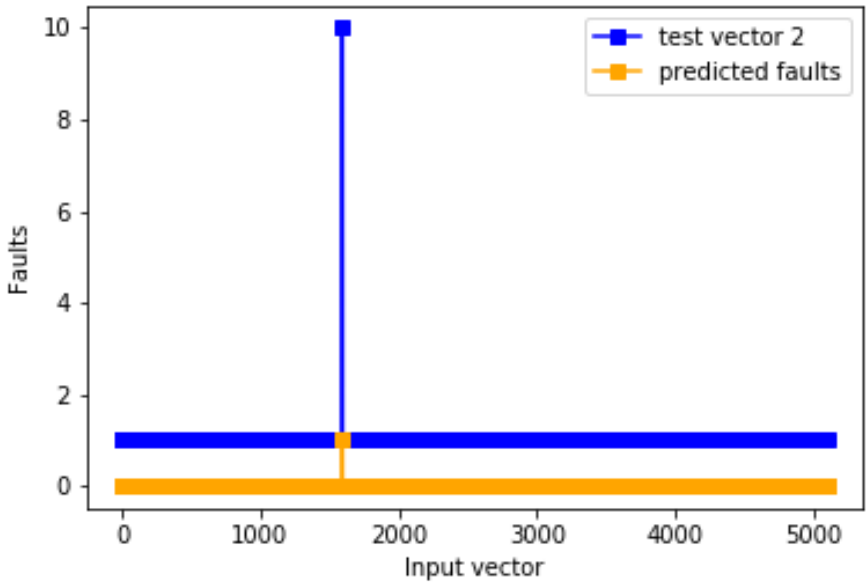


Figure 4.3: Test Vector at 5^{th} Time Instant, Fault at 1593^{th} Index Value

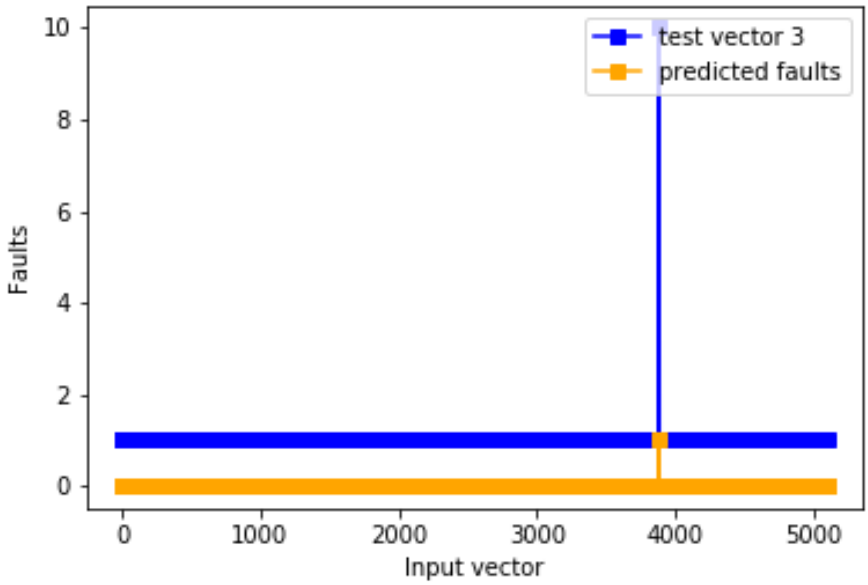


Figure 4.4: Test Vector at 11^{th} Time Instant, Fault at 3885^{th} Index Value

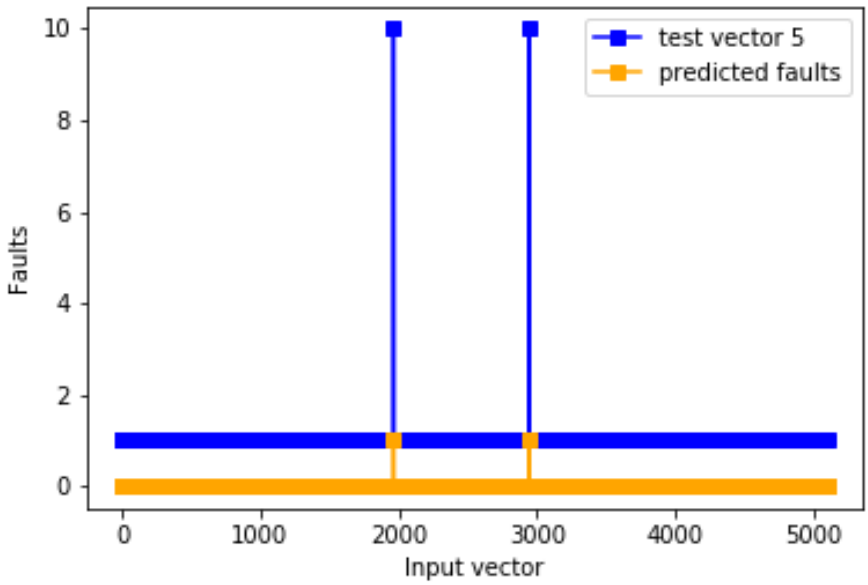


Figure 4.5: Test Vector at 149^{th} Time Instant, Fault at 1999^{th} & 2946^{th} Index Values

4.4.2 Test Case II

In this test case, the testing data has been acquired from actual outage events that occurred in the transmission system of Southwest Power Pool (SPP). Two of the transmission lines i.e., Arcadia-Northwest and Cimmaron-Draper, experienced outages. Testing data for ten (10) post-fault time instants is taken and provided to the trained neural network as 1D input vectors, meaning that the testing data is composed of ten (10) faulty 1D input vectors with two fault locations per vector. The neural network performs the same classification operations on these 10 1D vectors i.e., the testing data, as discussed earlier in section 3.3.2 and correctly identifies the transmission line outages.

The results obtained for performance metrics are shown in Table. 4.4:

The fault locations, per vector, displayed on the screen are shown in Fig.4.6, which are exactly in accordance with the actual fault locations.

```
fault is between Bus Arcadia-Northwest
fault is between Bus Cimarron-Draper
```

Figure 4.6: Two (2) Fault Locations

Similarly, the graph plotted between the predicted fault locations and actual fault locations per vector is shown in Fig.4.7.

Table 4.4: Performance Metrics Results: Test Case II

Performance Metric	**Results**	**Description**
Accuracy	100%	Accurate
False Positive	0	0 False Alarms
False Negative	0	0 False Alarms
True Positive	20	20 Faulty Locations
True Negative	2440	10 vectors of length 246, with 20 fault locations: $\implies 20 + 2440 \approx 246 * 10$

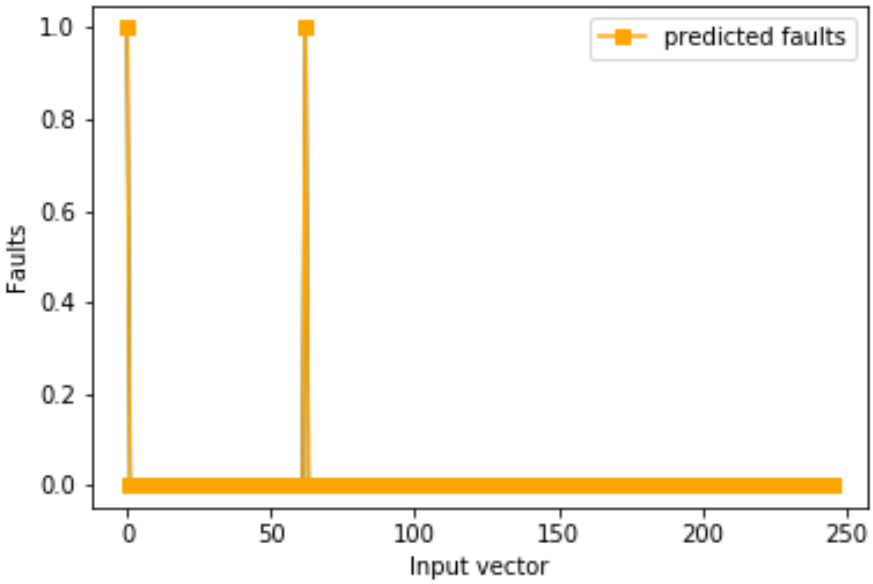


Figure 4.7: Two (2) Fault Locations

4.5 Analysis of Results

The results of the aforementioned experiments have been analyzed as follows:

- Training Results: The total training time required to train the neural network is 0.284 seconds per 10 input vectors.
- Test Case I Results: The output classes predicted, or fault locations identified for test case I are exactly in accordance with the introduced faults. The total time required to detect fifty (50) fault locations for one hundred and fifty-two (152) input vectors is 0.143 seconds.
- Test Case II Results: The output classes predicted, or fault locations identified for test case II are exactly in accordance with the introduced faults. The total time required to detect fifty (20) fault locations for ten (10) input vectors is 0.033 seconds.

4.5.1 Effect of Neural Network Parameters

As discussed earlier in section 3.2.2 and 3.2.3 of this thesis, the neural network parameters have a significant effect on the predicted output classes and one has to try different parameters for a particular type of application through educated trial and error method. Therefore, different neural network parameters were changed to see how the resultant accuracy of the predicted output classes is affected. The resultant effect on accuracy is explained as follows:

Effect of Changing Learning Rate

The learning rate or step size (α) used by the optimization algorithm to change the values of neural connection weights after each batchs iteration is set to 0.1, which is the typical value of used in machine learning algorithms. If is set to a very high value i.e., $\alpha = 1$, the algorithm tends to overshoot the requisite target in each attempt to converge. Similarly, if is set to a very low value i.e., $\alpha = 0.01$, the algorithm does converge but is time inefficient. This behavior of versus algorithms accuracy is shown in Fig.4.8.

As can be seen in Fig.4.8 that a smaller value of $\alpha = 0.01$ results in 100% accurate results, but the convergence time or training time taken by the algorithm for $\alpha = 0.01$ is 0.37 seconds per fifty (50) vectors, as opposed to 0.28 seconds per fifty (50) vectors in case of $\alpha = 0.1$.

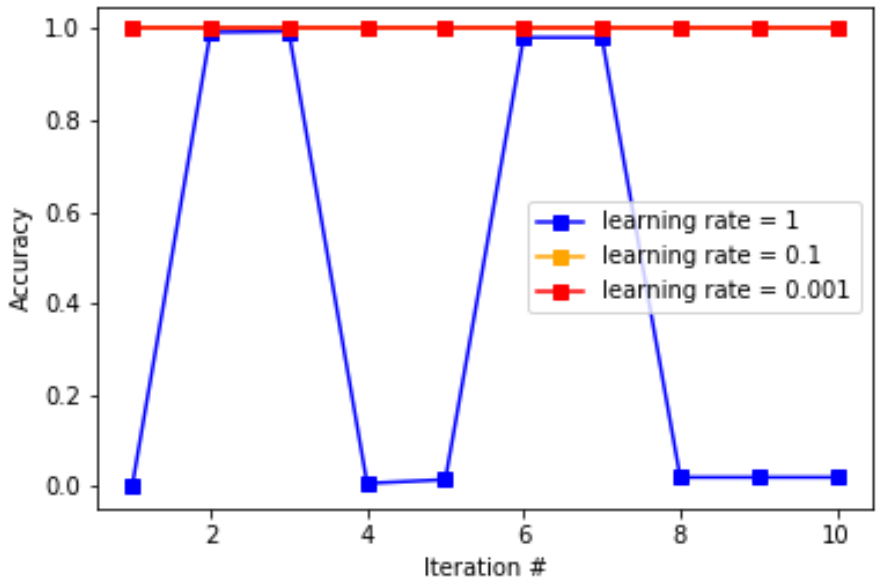


Figure 4.8: Effect of Changing Learning Rate (α) on Accuracy

Effect of Changing Number of Neurons Per Hidden Layer

The number of neurons per hidden layer in the neural network designed is set to 5000, which is an ideal value for a power network having 456 buses. This number was obtained through educated trial and error method. If the number of neurons per hidden layer is set to a lower value than 5000, the resultant accuracy is not 100%. Similarly, the number of neurons per hidden layer exceeds 5000, there is no change in the output accuracy, yet the algorithm becomes computationally extensive. This behavior of number of neurons versus algorithms accuracy is shown in Fig.4.9.

Effect of Changing Number of Hidden Layers

The number of hidden layers in the neural network designed is set to 2, which is an ideal value for capturing the admittance pattern of any power network. This number was obtained through educated trial and error method. If the number of hidden layers is set to a lower value than 2, the resultant accuracy is not 100% for large power systems with more than 456 buses. Similarly, if the number of hidden layers exceeds 2, there is no change in the output accuracy, yet the algorithm becomes computationally extensive.

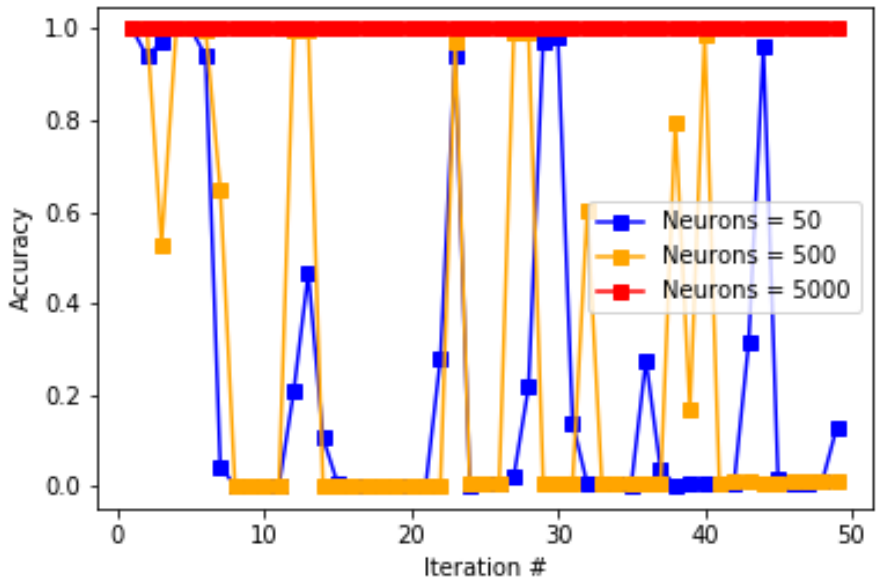


Figure 4.9: Effect of Changing Number of Neurons Per Hidden Layer

4.5.2 Fault Classification & Distance Calculation

Once a faulty transmission segment has been correctly identified by the neural network, the faults can easily be classified into temporary faults and permanent faults depending on the number of post-fault input vectors with fault values, since these post-fault input vectors signify a time period. The rate at which the trained neural network is provided with real-time faulty input vectors is associated with real time-stamps, which can be used to determine if the fault is temporary or permanent e.g., if after detection of fault locations in the first few vectors, lets say 10 vectors fetched at the rate of 1/30 seconds per vector (synchrophasors are fetched at the rate of 30 frames per second), no fault is detected by the neural network than it means that the fault was a temporary fault.

Similarly, the exact distance of fault from the terminals can also be determined in many ways. One of the simplest ways would be to look at the post-fault admittance of the faulty segment and determining the distance of the fault from known 'admittance per unit length' of the transmission line. Another method could be the one mentioned in [22], in which new equations are derived from the network admittance matrix, in terms of distance of the fault from both terminals, and exact fault location is determined on the faulty segment. The procedure adopted in [22] is as follows:

Let the post-fault statistically acquired Ybus matrix of the system, through the pro-

cedure mentioned in [23], for a particular time instant be:

$$Ybus = \begin{bmatrix} y_{1,1} & \cdots & y_{1,n} \\ \vdots & \ddots & \vdots \\ y_{n,1} & \cdots & y_{n,n} \end{bmatrix}_{n \times n}$$

Let the faulty transmission segment identified be segment A-B and the exact fault location on segment A-B be bus (n+1), as shown in Fig.4.10.

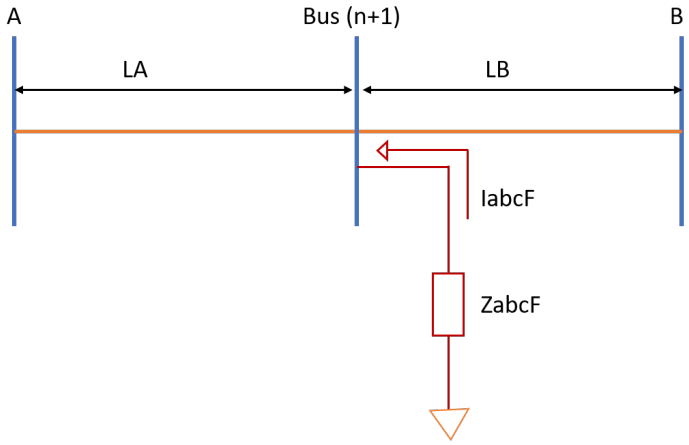


Figure 4.10: Exact Fault Location on Segment A-B

where ZabcF is the fault impedance and IabcF is the fault current. Upon occurrence of a fault, the change in all the bus voltages can be written as:

$$\begin{bmatrix} \Delta V_1 \\ \vdots \\ \Delta V_{(n+1)} \end{bmatrix} = Zbus\_Fault \begin{bmatrix} [0] \\ \vdots \\ IabcF \end{bmatrix} \tag{4.1}$$

or in terms of admittance values, the relationship stated in (4.1) can be re-written as:

$$\begin{bmatrix} [0] \\ \vdots \\ IabcF \end{bmatrix} = Ybus\_Fault \begin{bmatrix} \Delta V_1 \\ \vdots \\ \Delta V_{(n+1)} \end{bmatrix} \tag{4.2}$$

The elements of Ybus matrix mentioned in (4.5.2) are already known; therefore, the elements of the new Ybus matrix i.e., $Ybus\_Fault$, after occurrence of fault would be the same except for the elements related to the faulty segment A-B. These changed elements of $Ybus\_Fault$ are as follows:

$$Ybus\_Fault \;=\; \begin{bmatrix} y_{1,1} & \cdots & y_{1,n+1} \\ \vdots & \ddots & \vdots \\ y_{n+1,1} & \cdots & y_{n+1,n+1} \end{bmatrix}_{n+1\times n+1} \tag{4.3}$$

where,

$$y^{new}_{A,A} \;= y_{A,A} \;-\; (y_{abc})_{AB}/L \;+\; (y_{abc})_{AB}/L_A \tag{4.4}$$

$$y^{new}_{A,B} = \; y^{new}_{B,A} = \; [0] \tag{4.5}$$

$$y_{A,n+1} = \; y_{n+1,A} = -\; (y_{abc})_{AB}\,/L_A \tag{4.6}$$

Similarly,

$$y^{new}_{B,B} \;= y_{B,B} \;-\; (y_{abc})_{AB}/L \;+\; (y_{abc})_{AB}/L_B \tag{4.7}$$

$$y_{B,n+1} = \; y_{n+1,B} = -\; (y_{abc})_{AB}\,/L_B \tag{4.8}$$

$$y_{n+1,n+1} = \; (y_{abc})_{AB}\,/L_A \;+\; (y_{abc})_{AB}\,/L_B \tag{4.9}$$

where,

$(y_{abc})_{AB}$ is the per unit admittance of line segment A-B

L_A and L_B are the distances of the fault point (n+1) from bus A and bus B

L is the total length of the faulty line segment A-B

Since, all the other elements of the $Ybus\_Fault$ are the same as that of Ybus matrix and the newly introduced elements containing y_{n+1} are zero except those mentioned in (4.3)-(4.9); therefore, from (4.2), equations for ΔV_A and ΔV_B can be written as:

$$[0] = y_{A,1}.\Delta V_1 + y_{A,2}.\Delta V_2 + \cdots + y^{new}_{A,A}.\Delta V_A + \cdots + y_{A,n+1}.\Delta V_{(n+1)} \tag{4.10}$$

$$\Rightarrow [0] = y_{A,1}.\Delta V_1 + y_{A,2}.\Delta V_2 + \cdots + (y_{A,A} - (y_{abc})_{AB}/L$$

$$+(y_{abc})_{AB}/L_A).\Delta V_A + \cdots + (-(y_{abc})_{AB}/L_A).\Delta V_{(n+1)} \tag{4.11}$$

and,

$$[0] = y_{B,1}.\Delta V_1 + y_{B,2}.\Delta V_2 + \cdots + y^{new}_{B,B}.\Delta V_B + \cdots + y_{B,n+1}.\Delta V_{(n+1)} \tag{4.12}$$

$$\Rightarrow [0] = y_{B,1}.\Delta V_1 + y_{B,2}.\Delta V_2 + \cdots + (y_{B,B} - (y_{abc})_{AB}/L$$

$$+ (y_{abc})_{AB}/L_B\ ).\Delta V_B + \cdots + (-(y_{abc})_{AB}/L_B).\Delta V_{(n+1)} \tag{4.13}$$

Thus, the equations (4.11) and (4.13) can be solved to get exact locations L_A and L_B of the fault on segment A-B. The different symbols used in these equations are described in Table. 4.5:

Table 4.5: Symbols Used

Symbol	**Description**
L_A, L_B	Length of Fault from Bus A & B
ΔV_n	Voltage at Bus n
$y_{A,A}$	Admittance Value at Index [A,A]
$y_{A,B}$	Admittance Value at Index [A,B]
$y_{B,B}$	Admittance Value at Index [B,B]
y_{abcAB}/L	Admittance per Unit Length of Line Segment A-B

5 ENVISION & CONCLUSION

In this chapter, the scope of the project, in terms of futuristic technology, has been discussed, followed by the conclusions drawn.

5.1 Scope in terms of Futuristic Technology

The scope of this fault location detection algorithm can be envisioned in terms of a futuristic technology as:

1. Each and every piece of information can be extracted from the provided synchrophasor data of bus voltages and injected currents. This synchrophasor data is acquired and processed in real-time, which accurately captures the ever-changing trends in transmission/distribution networks due to uncontrolled factors like aging of transmission lines.

2. It eliminates the dependency on physical models or systems, physical parameters, network-specific data of and interconnections in transmission or distribution networks e.g., connection schemes, distribution networks electrical parameters like cable type and underlying geometry.

3. It is not affected by the different transmission/distribution network topologies i.e., over-head or under-ground, transposed and un-transposed, balanced and un-balanced, and radial/ring/star etc.

4. This algorithm takes a few minutes to learn actual system parameters, does not make any assumptions and therefore, is not affected by changes in transmission or distribution system interconnections as it can always be trained within a few minutes to learn new system behaviors and detect faults accordingly.

5. This algorithm can also be trained for determining normal faults as well as High Impedance Faults (HIF) in distribution networks, without making any significant changes in its code.

6. It gives optimal and fast performance, is deployment friendly since it does not require any changes in system hardware and inexpensive. A control room operator can easily see the faulty locations in the transmission or distribution networks on his/her monitor in real-time within a few seconds of occurrence of the fault.

7. It has significant scope for research and innovation. For the time being, it has only been tested in transmission system networks, but the same algorithm can also be deployed in distribution networks. Furthermore, the physical realization of this algorithm would have a direct societal-impact, liberating major workforce from trivial-issues demanding immense physical/mental exertion and directing it toward the notion of controlling AI. With the future lying in the hands of artificial-intelligence, such automated-systems promise profitable long-term investment.

5.2 Conclusions

Several noteworthy conclusions can be drawn from the research conducted and respective fault location detection algorithm derived.

In terms of parameters of the algorithm, it is concluded that the typical learning rate set for most of the machine learning algorithms works well for fault detection in power systems. It can also be concluded that batch size and number of steps, of the sample synchrophasor data taken, provide the best results when taken in the ratio 2:1 for transmission systems. Another conclusion which can drawn in terms of algorithm parameters, is that synchrophasor data for continuous 2.7 minutes is more than enough for training the algorithm designed for a 456-bus transmission network. Furthermore, the number of neurons per hidden layers should be kept comparable to the size of the 1D input vectors used for training the algorithm. Also, only two number of hidden layers are enough to detect faults in any power system, irrespective of the number of buses in that power system. Last but not the least, the threshold setting required to make the algorithm differentiate between normal and faulty data can be set to any higher or lower value than the nominal value, irrespective of the range of actual fault values in the power system.

Similarly, in terms of performance of the fault location detection algorithm designed, it only needs synchrophasor data of bus voltages and injected currents to accurately detect faulty segments in a transmission or distribution network. And the proposed algorithm requires the admittance per unit length value of the lines for exacting locating the fault on

the faulty line segment. Also, the proposed algorithm has eliminated the need to use line models or make any assumptions in the computations of the fault detection in the power network. And thus, it is one of the most reliable and efficient ways of detecting fault locations in a power network in real time (merely taking a few milliseconds to detect faults in case of 456 buses).

Bibliography

[1] "Southwest power pool," https://www.elp.com/Electric-Light-Power-Newsletter/articles/2018/05/southwest-power-pool-state-of-the-market-2017.html, accessed: 2019-03-30.

[2] J.-A. Jiang, J.-Z. Yang, Y.-H. Lin, C.-W. Liu, and J.-C. Ma, "An adaptive pmu based fault detection/location technique for transmission lines. i. theory and algorithms," *IEEE Transactions on Power Delivery*, vol. 15, no. 2, pp. 486–493, 2000.

[3] M. M. Saha, J. J. Izykowski, and E. Rosolowski, *Fault location on power networks*. Springer Science & Business Media, 2009.

[4] "Loss functions," https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c, accessed: 2019-03-31.

[5] "Negative loss functions," shorturl.at/ryGV2, accessed: 2019-03-31.

[6] "Supervised vs. unsupervised learning," https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d, accessed: 2019-03-30.

[7] M. Wache and D. Murray, "Application of synchrophasor measurements for distribution networks," in *2011 IEEE Power and Energy Society General Meeting*. IEEE, 2011, pp. 1–4.

[8] E. C. Senger, G. Manassero, C. Goldemberg, and E. L. Pellini, "Automated fault location system for primary distribution networks," *IEEE Transactions on Power Delivery*, vol. 20, no. 2, pp. 1332–1340, 2005.

[9] L. Xu and M.-Y. Chow, "Power distribution systems fault cause identification using logistic regression and artificial neural network," in *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*. IEEE, 2005, pp. 6–pp.

[10] L. Xu, M.-C. Chow, and X. Gao, "Comparisons of logistic regression and artificial neural network on power distribution systems fault cause identification," in *Proceedings of the 2005 IEEE Midnight-Summer Workshop on Soft Computing in Industrial Applications, 2005. SMCia/05.* IEEE, 2005, pp. 128–131.

[11] D. Thukaram, H. Khincha, and H. Vijaynarasimha, "Artificial neural network and support vector machine approach for locating faults in radial distribution systems," *IEEE Transactions on Power Delivery*, vol. 20, no. 2, pp. 710–721, 2005.

[12] J. Kim, M. E. Baran, and G. C. Lampley, "Estimation of fault location on distribution feeders using pq monitoring data," in *2007 IEEE Power Engineering Society General Meeting*. IEEE, 2007, pp. 1–4.

[13] A. Von Meier, D. Culler, A. McEachern, and R. Arghandeh, "Micro-synchrophasors for distribution systems," in *ISGT 2014*. IEEE, 2014, pp. 1–5.

[14] J. Ren, S. Venkata, and E. Sortomme, "An accurate synchrophasor based fault location method for emerging distribution systems," *IEEE Transactions on Power Delivery*, vol. 29, no. 1, pp. 297–298, 2014.

[15] J. Zhu, D. L. Lubkeman, and A. A. Girgis, "Automated fault location and diagnosis on electric power distribution feeders," *IEEE Transactions on Power Delivery*, vol. 12, no. 2, pp. 801–809, 1997.

[16] J. Mora-Florez, J. Melendez, and G. Carrillo-Caicedo, "Comparison of impedance based fault location methods for power distribution systems," *Electric power systems research*, vol. 78, no. 4, pp. 657–666, 2008.

[17] J. De La Ree, V. Centeno, J. S. Thorp, and A. G. Phadke, "Synchronized phasor measurement applications in power systems," *IEEE Transactions on smart grid*, vol. 1, no. 1, pp. 20–27, 2010.

[18] J. Sykes, K. Koellner, W. Premerlani, B. Kasztenny, and M. Adamiak, "Synchrophasors: A primer and practical applications," in *2007 Power Systems Conference: Advanced Metering, Protection, Control, Communication, and Distributed Resources*. IEEE, 2007, pp. 213–240.

[19] J.-A. Jiang, Y.-H. Lin, J.-Z. Yang, T.-M. Too, and C.-W. Liu, "An adaptive pmu based fault detection/location technique for transmission lines. ii. pmu implementation and performance evaluation," *IEEE Transactions on Power Delivery*, vol. 15, no. 4, pp. 1136–1146, 2000.

[20] Y.-H. Lin, C.-W. Liu, and C.-S. Chen, "A new pmu-based fault detection/location technique for transmission lines with consideration of arcing fault discrimination-part i: theory and algorithms," *IEEE Transactions on power delivery*, vol. 19, no. 4, pp. 1587–1593, 2004.

[21] C.-S. Yu, C.-W. Liu, S.-L. Yu, and J.-A. Jiang, "A new pmu-based fault location algorithm for series compensated lines," *IEEE Transactions on Power Delivery*, vol. 17, no. 1, pp. 33–46, 2002.

[22] S. M. Brahma, "Fault location scheme for a multi-terminal transmission line using synchronized voltage measurements," *IEEE Transactions on Power Delivery*, vol. 20, no. 2, pp. 1325–1331, 2005.

[23] M. Saadeh, R. McCann, M. Alsarray, and O. Saadeh, "A new approach for evaluation of the bus admittance matrix from synchrophasors:(a statistical ybus estimation approach)," *International Journal of Electrical Power & Energy Systems*, vol. 93, pp. 395–405, 2017.

[24] "Softmax activation function," shorturl.at/lMTY3, accessed: 2019-03-31.

[25] "Adam optimization," https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, accessed: 2019-03-31.

[26] P. Sadowski, "Notes on backpropagation," *homepage: https://www. ics. uci. edu/pjsadows/notes. pdf (online)*, 2016.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.