2023

Application of Conventional Feedforward and Deep Neural Networks to Power Distribution System State Estimation and State Forecasting

James Paul Carmichael
University of Kentucky, jamespcarmichael@gmail.com
Author ORCID Identifier:
https://orcid.org/0009-0005-2824-3391
Digital Object Identifier: https://doi.org/10.13023.etd.2023.054

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

James Paul Carmichael, Student

Dr. Yuan Liao, Major Professor

Dr. Daniel L. Lau, Director of Graduate Studies

2023

Application of Conventional Feedforward and Deep Neural Networks to Power Distribution System State Estimation and State Forecasting

James Paul Carmichael
University of Kentucky, jamespcarmichael@gmail.com
Author ORCID Identifier:
https://orcid.org/0009-0005-2824-3391
Digital Object Identifier: https://doi.org/10.13023.etd.2023.054

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

James Paul Carmichael, Student

Dr. Yuan Liao, Major Professor

Dr. Daniel L. Lau, Director of Graduate Studies

Application of Conventional Feedforward and Deep Neural Networks to Power Distribution System State Estimation and State Forecasting

---

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
Department of Electrical and Computer Engineering
at the University of Kentucky

By
James P. Carmichael, PE, PMP
Lexington, Kentucky
Director: Dr. Yuan Liao, Professor of Electrical and Computer Engineering
Lexington, Kentucky
2023

https://orcid.org/0009-0005-2824-3391

ABSTRACT OF DISSERTATION

Application of Conventional Feedforward and Deep Neural Networks to Power Distribution System State Estimation and Forecasting

Classical neural networks such as feedforward multilayer perceptron models (MLPs) are well established as universal approximators and as such, show promise in applications such as static state estimation in power transmission systems. This research investigates the application of conventional neural networks (MLPs) and deep learning based models such as convolutional neural networks (CNNs) and long short-term memory networks (LSTMs) to mitigate challenges in power distribution system state estimation and forecasting based upon conventional analytic methods. The ability of MLPs to perform regression to perform power system state estimation will be investigated. MLPs are considered based upon their promise to learn complex functional mapping between datasets with many features. CNNs and LSTMs are considered based upon their promise to perform time-series forecasting by learning the autocorrelation of the dataset being predicted. The performance of MLPs will be presented in terms of root-mean-square error (RMSE) between actual and predicted voltage magnitude and voltage phase angles and training execution time for distribution system state estimation (DSSE). The performance of CNNs, and LSTMs will be presented in terms of RMSE between actual and predicted real power demand and execution time when performing distribution system state forecasting (DSSF). Additionally, Bayesian Optimization with Gaussian Processes are used to optimize MLPs for regression. An IEEE standard 34-bus test system is used to illustrate the proposed conventional neural network and deep learning methods and their effectiveness to perform power system state estimation and power system state forecasting respectively.

KEYWORDS: artificial neural networks (ANNs), multilayer perceptron networks (MLPs), convolutional neural networks (CNNs), long short-term memory networks (LSTMs), distribution system state estimation (DSSE)

Application of Conventional Feedforward and Deep Neural Networks to Power Distribution System State Estimation and Forecasting

By
James Paul Carmichael

Dr. Yuan Liao
Director of Dissertation

Dr. Daniel L. Lau
Director of Graduate Studies

04/07/2023
Date

DEDICATION

This dissertation and related PhD research is dedicated to my mother, Carol M. Self.

ACKNOWLEDGMENTS

I would like to express my appreciation and gratitude to those who supported and guided me through my PhD studies, dissertation defense and completion of my dissertation. First, I would like to thank my advisor Dr. Yuan Liao. He guided my studies through some very challenging life events and served a constant supporter. I look forward to collaborating with him and his future students on research as I move forward in my career as a research scientist.

I would also like to thank my PhD committee members: Dr. Zongming Fei, Dr. Joseph Sottile, and Dr. Peng Wang. I would also like to thank Dr. Miroslaw Truszcynski for serving as an outside examiner. I would also like to thank electrical and computer engineering Director of Graduate Studies, Dr. Daniel Lau. His timely communications and availability to address questions concerning key milestones and requirements for completion of my PhD studies is greatly appreciated.

I would also like to thank and acknowledge the U. S. Department of Defense Science, Mathematics, and Research for Transformation (SMART) Program for helping to fund portions of my research and for providing opportunities beyond graduation.

Last but not least I would like to thank Kristie L. Wideman for her love and support through the most challenging periods of my research. It made all the difference in the world.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. PURPOSE AND SIGNIFICANCE OF THE RESEARCH

1.1 Research Purpose Statement

In power systems an essential requirement is resiliency. In general, resiliency includes the ability of a power system to withstand and recover quickly from events that may be considered low-frequency, yet high-impact events or adverse conditions. Examples of such events or adverse conditions relate to but are not limited to the following: Extreme weather, Natural disasters, man-made outages (physical, cyber, coordinated), lack of observability, topology errors, and false data injection attacks (FDIAs) [12].

Ensuring robust state estimation in the presence of noisy environments and following a cyber-attack to the grid is critical [1]. The state estimation process seeks to provide an optimal estimation of the true values of bus voltages and angles and power flows across the power system [2]. Thus, state estimation provides the basis or enhancement for other power system applications such as system planning, optimization, fault analysis, protection, and fault location [3], [4], [5], [6].

Artificial neural networks have been used in power distribution system state estimation. However, there is a lack of analysis and study of which types of ANNs and what structures including parameters are most suitable for state estimation applications. When designing an ANN for a state estimator, trial and error approach has been common and there is to the knowledge of the author of this dissertation no systematic method available to guide the process.

This research aims to perform a comprehensive study of the performance of various types of ANNs with different structures and provides a possible optimization method to determine the optimal parameters for desired performance metrics. These parameters are

referred to as *hyperparameters*, including *model parameters* such as number of hidden layers and number of neurons in a layer, and *algorithm parameters* such as adjustable learning rate. This research will improve the training efficiency for large power systems.

This research focuses on application of classical artificial neural networks and deep learning networks to distribution system state estimation (DSSE) and distribution system state forecasting (DSSF). There are various types of networks such as conventional feedforward multilayer perceptron networks (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs)/long short-term memory networks (LSTMs), and hybrid-neural networks utilizing a combination of network types. Preliminary results based on MLPs, CNNs, and LSTMs are presented in this research.

Hyperparameters may be obtained using optimization methods such as but not limited to grid search, genetic algorithms, and Bayesian Optimization with Gaussian Processes. Bayesian Optimization with Gaussian Processes was selected for this research.

1.2 Dissertation Outline

CHAPTER 2 presents the background and related work. The objective of this chapter is to provide a background in state estimation and strengthen the case for application of deep learning to mitigate challenges of applying classical (analytical) techniques to modern power distribution systems. It starts with a review of conventional power system state estimation, observability, and resiliency. It then considers state estimation applied to smart distributions systems. Next, artificial neural networks and deep learning applied to power system state estimation, observability, topology errors and false data injection attacks are presented. Next, distribution system state estimation is presented. Included are key characteristics of transmission and distribution systems,

functional block diagrams of state estimation and an overview of a state estimator and related processes. The chapter then presents challenges of applying conventional state estimation utilizing weighted least square method to distribution systems. It is noted that data driven approaches utilizing deep learning can directly mitigate the challenges of applying conventional methods. Then, a background and definition of challenges to all distribution systems such as lack of observability, topology errors and false data injection attacks is presented. It is noted that these challenges exist in all power distribution systems and that deep learning may serve as part of solutions to address each. It should also be noted that the research presented in this dissertation does not directly address these challenges and mitigation of the challenges is left for future research.

Next, the chapter presents an introduction to deep learning models utilized in the remainder of the research on which this dissertation is based. Conventional feedforward multilayer perceptron models (MLPs) are presented first. This introduction of MLPs includes a visual representation of a perceptron building block of MLP networks and a multilayer perceptron model functional representation. Next, this chapter presents an overview of convolutional neural networks and includes a functional representation of this model type. The chapter concludes with a presentation of recurrent neural networks (RNNs) in general and long short-term memory networks (LSTMs) in particular.

CHAPTER 3 presents distribution system state estimation (DSSE) with MLP models. The objective of this chapter is to describe the process of training, testing and validating unoptimized MLP models and to introduce hyperparameter optimization based upon Bayesian Optimization with Gaussian Processes. It starts with a description of DSSE without hyperparameter optimization. Next, the test distribution utilized in the

research on which this dissertation is based, is described. The IEEE 34 Node base test distribution is presented. Then, the chapter continues with a description of the power flow simulation measurement points and quantities. This description includes details of the power monitors and voltage measurement locations and the quantities captured at each location in the test distribution system. The chapter continues with a description of the training, testing and validation data to be used with MLP models. Next, the network model parameters and network hyperparameters for unoptimized conventional feedforward MLPs are presented. Next, the chapter includes a description, details, and summary of the actual implementation of the unoptimized MLP model in Python. Trial results for MLP models without hyperparameter optimization are presented. It should be noted that the methodology described in this chapter is considered the "original" approach that involves "partial state estimation" owing to the limited monitor locations in the test base distribution system. CHAPTER 4 will expand upon this methodology to yield full state estimation. CHAPTER 3 also presents state estimation and forecasting based upon time series physics aware models. Hourly temperature is considered to predict real power demand at the substation bus of the test base feeder system. The average RMSE of real power, reactive power, voltage magnitude, and voltage phase is presented for forecast horizons of 24, 168, and 672 hours. Next, the process of DSSE with MLPs with hyperparameter optimization is presented. Bayesian Optimization with Gaussian Processes is introduced along with the supporting theory and equations on which the Python implementation is based. Next, an example of a one dimensional test objective function is presented to illustrate all steps of the optimization process. The chapter then continues with the practical implementation of Bayesian Optimization with

Gaussian Processes applied to previously unoptimized MLP models. Trial results for the baseline MLP model without hyperparameter optimization are presented along with the hyperparameters selected via Bayesian Optimization. The chapter ends with a presentation of the final optimized model configurations.

CHAPTER 4 expands upon CHAPTER 3 to present full distribution system state estimation with optimized MLP models. The main objective of CHAPTER 4 is to present an improved workflow and methodology over the methodology introduced in the early phases of the research on which this dissertation is based. The chapter starts by summarizing the elements of the original workflow and data pipeline. It presents details of the load profiles required, the distribution system simulator, the output of the simulator, data pre-processing required to establish the data sets required for training, testing and validating machine learning models and a description of the machine learning ecosystem that made application of MLPs to power distribution system state estimation described in CHAPTER 3 possible. Next, an improved workflow and data pipeline is presented. The load profile, distribution simulator and machine learning ecosystem are described. Central to the improved workflow, is the introduction of a Python dynamic link library (DLL) that enables configuration and control of the distribution system simulator. CHAPTER 4 continues with an explanation of how the updated methodology enables full system state estimation by considering all lines and resulting bi-directional power flow along with all node voltages and phase angles. This contrasts with the previous methodology that utilized a limited number of lines and buses corresponding to the monitors placed in the test feeder system. Next, the process of gathering training and testing data is presented. Then, the process of randomly selecting lines is presented. The

objective of using percentages of lines (10, 25, 50, 75 and 100) and associated bi-directional power flow to predict complex voltages at all bus locations is presented. Next, unoptimized MLP models are compared with optimized MLP models to perform state estimation. The optimized MLPs utilize Bayesian Optimization with Gaussian Processes. Finally, the hyperparameters selected via Bayesian Optimization are presented. The main takeaway from CHAPTER 4 is that an improved workflow and data pipeline enables full system state estimation, which dis not limited by the available monitors placed within the power simulator software. The improved workflow enables the possibility of state estimation being applied to much larger distribution systems. Application and validation of the improved workflow to larger distributions systems is left to future research.

CHAPTER 5 presents distribution system state forecasting with convolutional neural network models. The objective of CHAPTER 5 is to demonstrate the ability of CNNs to be utilized in time-series forecasting. The chapter starts with an introduction to time-series forecasting and then presents a structured time series forecasting process. Next, data preparation for time-series forecasting with deep learning models is presented. Following data preparation, CHAPTER 5 presents hyperparameters selected for unoptimized CNN models. Next, the practical implementation of an unoptimized CNN model for time series forecasting in Python is presented. A specific location in the test feeder system was selected to demonstrate the forecasting process. Plots for the actual and predicted real power, reactive power, voltage magnitude and voltage phase angle for an unoptimized CNN model are presented. Next, the effect of adjusting the learning rate

manually is presented along with a plot of the actual and predicted reactive power at a specific location.

CHAPTER 6 presents distribution system state forecasting with long short-term memory models. The objective of CHAPTER 6 is to demonstrate the ability of LSTMs to be utilized in time-series forecasting. The chapter starts with data preparation for LSTMs and it is noted that the data preparation for time-series data is the same as that for CNNs. Next, practical implementation of an unoptimized LSTM model for time series forecasting in Python are presented. A specific location in the test feeder system was selected to demonstrate the forecasting process. Plots for the actual and predicted real power, reactive power, voltage magnitude and voltage phase angle for an unoptimized LSTM model is presented.

CHAPTER 7 presents a comparison of auto regressive models and convolutional neural networks for power distribution system time-series forecasting. The objective is to demonstrate that for univariate time-series forecasting, classical and deep learning models should be considered as options. The chapter starts with an introduction to a "classical" machine learning model used for univariate time-series forecasting (ARIMA). Next, CHAPTER 5 is referenced for a review of CNN models and related hyperparameters as they will be utilized throughout the comparison. Then a practical ARIMA model implementation in Python is presented along with the locations in the test feeder on which time-series forecasting will be performed with this model type. Then CHAPTER 7 presents a practical CNN model implementation in Python along with the locations in the test feeder on which time-series forecasting will be performed with this model type. Next, the chapter presents plots of actual versus predicted real power,

reactive power, voltage magnitude and voltage phase angle at three separate locations in the test feeder for both ARIMA and CNN model types. Finally, the root-mean squared errors (RMSE) for predicted values at each location is presented.

CHAPTER 2. BACKGROUND AND RELATED WORK

2.1 Introduction

State estimation research and application has historically been largely focused on transmission systems as opposed to distribution systems. With increasing developments of the "smart grid", increased utilization of phasor measurement units (PMUs) and improvements in monitoring and communications, distribution system state estimation (DSSE) interest and research has greatly increased in recent years.

The inherent challenges of application of "conventional" state estimation techniques to power distribution systems based upon weighted least squares is well established in the literature. Application of MLPs, featuring feedforward architecture, to mitigate the challenges of applying weighted least squares, is also well established in the literature. Additionally, literature review has found research in fault identification on electrical transmissions using feedforward neural networks [63] and [65], modular neural networks for single transmission lines [64], and transmission fault location techniques using traveling wave method and discrete wavelet transform [66].

In recent years, "deep learning neural networks" have gained increasing interest not only in being able to improve state estimation over conventional techniques utilizing weighted least squares, but also in the possibility of being able to address what may be considered as "extreme" or "adverse" conditions such as, but not limited to lack of observability, topology errors, false data injection attacks, network outages due to weather or malicious attack, and variances in weather that may affect distributed power generation from solar and wind sources.

The literature review supporting this research includes the following major topics:

- Conventional (Analytical) Power System State Estimation, Resiliency and Observability
- State Estimation Applied to Smart Distribution Systems
- Artificial Neural Networks and Deep Learning Applied to Power System State Estimation, Observability, Topology Errors, and False Data Injection Attacks

2.2 Review of Conventional (Analytical) Power System State Estimation, Resiliency and Observability

Conventional state estimation was introduced in 1970 via a series of papers authored by Fred C. Shweppe and J. Wildes [2], [7], [8]. The overall problem, mathematical modeling and general algorithm for state estimation, error detection and identification are presented in [2]. The key assumption of the classical approach presented is that the state estimation vector consisting of the voltage magnitude and phase angles at all generation and load buses is static or quasi-static.

Further assumptions are that the system is balanced, linear and can be accurately approximated via an iterative algorithm utilizing weighted least squares as the estimator. While these assumptions are reasonable when applied to transmission systems, they are not considered reasonable when applied directly to distribution systems. An approximate model and the resulting simplifications in state estimation, bad data detection and identification are presented in [7]. This model is based on a DC load flow yielding linear equations with the following four basic assumptions. The first assumption is $X/R \gg 1$ for all lines, where X represents the line impedance and R represents the line resistance. The second assumption is that $V \approx 1$ for all buses, where V represents the line to neutral voltage

magnitude. The third assumption is $\delta_i - \delta_k \approx 0$ for all lines, where δ_i represents the phase angle at node i and δ_k represents the phase angle at node k. The fourth assumption is that the real power measurement errors are uncorrelated with voltage and reactive power measurement errors.

The resulting approximate model, while enabling potential application to distribution systems is not readily applicable to state estimation in general for practical transmission or distribution networks. Thus, reference [8] addresses implementation problems associated with dimensionality, computational efficiency, data storage and the time-varying nature of actual power systems.

The time-variation inherent in power systems is addressed in [9]. This paper is a review of dynamic state estimation (DSE) methods as opposed to static state estimation (SSE). These methods are based primarily on Kalman Filtering (KF) techniques, M-estimation, and the Square Root Filter (SRF) technique which is an alternative implementation of KF that is numerically more stable.

The authors summarize the review by stating that the Kalman Filtering technique, while being the most popular, is not necessarily the most accurate. They also promote further research into the application of artificial neural networks (ANNs) in general and/or fuzzy logic networks to the dynamic state estimation problem. The actual architecture of either of these two network types were not discussed as part of the review.

Paper [10] discusses the essential role of power system observability to the state estimation problem and presents a theoretical basis for an algorithm to determine observability. The authors emphasize the requirement that conventional or classical state estimation methods be applied only to systems that are observable and thus establish that an observability test be conducted prior to performing state estimation.

The algorithm presented is based upon a graph theoretical or topological approach. Specifically, the algorithm seeks to determine if the Jacobian of the system parameter

network is of full rank. If so, the power system network is considered observable. The challenge facing practical implementation of the proposed algorithm is that it can be difficult to provide a "correct" answer of rank in every case. In other words, most techniques provide a floating point computation of rank.

The references [69], [70], [71] represent performance considerations of the weighted least squares state estimation method in the case of topology errors and propose potential ways of identifying topology errors for improved state estimation results.

The challenges to state estimation due to lack of observability are further discussed in [11]. The authors reiterate the essential observability criteria needed to perform classical state estimation and further surmise that the first step to controllability is observability. The definition of observability is generalized from the numerical rank definition proposed in [10] to that of "obtaining accurate knowledge about relevant parameters of a system."

The authors in [11] suggest that future work involve research into the impacts of systems with a larger number of buses than was considered in their simulations and consideration of the robustness of the proposed distribution system state estimation algorithm. In the paper being referenced, *robustness* refers to the insensitivity of the state estimation algorithm to major deviations in a limited number of redundant measurements.

The authors in [12] provide an in depth discussion of the growing threats to modern power system resiliency that applies to all aspects of the grid (i.e. generation, transmission, distribution, distributed generation, micro-grids, etc.) According to the authors in [12], investment in the modernization of the power grid must be done so with a "No Regrets Strategy". This strategy is based upon the cornerstones of resiliency, flexibility and connectivity. The paper defines resiliency as resistance to high-impact, low frequency events such as extreme weather, earthquakes, tsunamis and outages (physical, cyber, coordinated). Flexibility is defined as adaptability to uncertainties such as fuel prices, power market prices/incentives, variable generation, consumer behavior, regulation and policy. Connectivity is defined as enhanced interoperability across the electricity

enterprise. Connectivity includes advanced sensors, mobile devices, grid modernization and two-way flow.

2.3 State Estimation Applied to Smart Distribution Systems

Since the introduction and formalization of state estimation applied to power systems in the early 1970s, most of the attention has been on application to transmission systems. This is understandable given the challenges presented to this point and those expanded upon in section 2.6 of this dissertation. The authors in [13] provide a survey on state estimation techniques and challenges in so-called "smart distribution systems".

This survey summarizes most of the essential concepts considered to this point. Among these concepts are conventional mathematical formulation based upon an iterative algorithm utilizing weighted least squares or similar estimator; application of pseudo-measurements to mitigate lack of sufficient metering to enable system observability; consideration of optimal meter placement given the relatively limited metering availability in distribution systems; network topology issues and effects on system modeling required for accurate distribution state estimation; impacts of renewable penetration; and cybersecurity concerns.

The paper goes further to make a distinction between "conventional" state estimation that is considered analytical and deterministic and "modern" state estimation that is considered *data driven* and *probabilistic.* Regarding conventional state estimation, various "robust state estimators" are presented in Table 1, along with pros and cons of each.

Table 1 - Robust State Estimators

Robust State Estimators	Pros	Cons
Weighted Least Squares (WLS)	Fast, simple, widely used	Sensitive to bad data
Least Median of Squares (LMS)	Robust against bad data	High computational cost, high measurement redundancy requirements
Least Trimmed Squares (LTS)	Robust against bad data	High computational cost and memory requirement
Least Absolute Value (LAV)	Robust against bad data, small sensitivity to line impedance uncertainty	High computational cost, sensitivity to leverage points and measurement redundancy
Generalized Maximum-likelihood (GM)	Robust against bad data	Parameter selection sensitivity

The authors in [13] identify two major categories of data driven approaches as alternatives to conventional state estimation based upon the previous list of estimators. Table 2 presents the data driven categories, key characteristics, and applications.

Table 2 - Data Driven Approaches to State Estimation

Data Driven Categories	Key Characteristics	Applications
Probabilistic and Statistical Approaches	Employ spatial/temporal correlation and historical probability distributions Used widely for pseudo-measurement generation and uncertainty assessment.	Empirical Studies Gaussian Mixture Models (GMM) Expectation Maximization (EM) Time-Varying Variation and Mean Modeling Correlation Analysis (between total and individual consumption) Nodal Active-Reactive Correlation Analysis Internodal and Intranodal Correlation Modeling Intertemporal Correlation Analysis Multivariate Complex Gaussian Modeling Constrained Optimization
Learning-Based Approaches	Based on machine learning algorithms Addresses problem of active/reactive power pseudo-measurement generation and uncertainty assessment.	Probabilistic Neural Networks (PNNs) Artificial Neural Networks (ANNs) Clustering Algorithms Parallel Distributed Processing Networks (PDPs)

Related to the recommendations of notable research directions, the paper in [14] presents previous work in the area of state estimation for real-time monitoring of distribution systems. While the work presented in this paper is based upon weighted least squares estimation, it shows the close correlation of state estimation accuracy to the initial starting point selected and accuracy of the forecasted loads.

Thus, an important takeaway from the work presented in [13] and [14] collectively is the idea of establishing a *hybrid process* involving classical state estimation algorithms and data-driven forecasting. The data-driven portions would support the classical state estimation algorithm by providing a better starting point than a typical "flat start", higher probability of convergence and produce more accurate pseudo-measurements than those queried from large historical data repositories.

The design of an off-line planning method to enable real-time monitoring and control in systems with limited observability is considered in [15] through consideration of robust measurement placement for distribution system state estimation. The authors in this paper propose a robust measurement placement model to maximize estimation accuracy for DSSE over a wide-range of worst case operating conditions.

The problem is formulated as a *mixed-integer semi-definite programming problem* (MISDP). The authors seek to avoid combinatorial complexity through a convex relaxation, followed by a local optimization method. The approach in [15] demonstrates that accuracy of DSSE can be enhanced significantly by placing a limited number of measurements in optimal locations. Again, the approach taken, can be considered a hybrid approach of classical state estimation with updated probabilistic and statistical components that seek to minimize the effect of lack of observability on the weighted least squares estimator.

The paper presented in [16], provides a linear state estimation formulation for smart distribution systems. The authors assume the availability of synchro phasors which yield

direct voltage phasors at bus locations. Line power flows and current magnitudes are then able to be ascertained via the direct quantities available.

Reference [16] shows that availability of direct voltage phasors effectively linearizes the *h(x)* coefficient matrix used in classical state estimation so that the result is a linear, non-iterative state estimation solution. Results shown in [16] confirm low computational burden, accommodation of meshed networks and avoidance of convergence issues which may occur in dealing with practical distribution systems with high r/x ratios. It should be noted, however that to achieve the results presented in [16], two requirements are necessary. The first requirement is a resolution of +/- 1 μS corresponding to 0.0216 degree error in a 60-Hz system. The second requirement is a maximum allowable total vector error (TVE) of 1.0% when maximum phase error is 0.57 degrees. Literature review also revealed the IEC/IEEE 60255-118 standard that defines synchro phasor frequency, and rate of change of frequency (ROCOF) measurements that ensure interoperability between PMU manufacturers [67].

The authors in [17] present a branch-estimation-based state estimation method for radial distribution systems. While this approach utilizes many of the conventional or classical state estimation techniques, it has the ability to handle most kinds of real-time measurements by decomposing the weighted least squares problem into a series of weighted least squares problems such that each sub-problem deals with single-branch estimation.

The establishment of "zones" is a novel idea such that the entire distribution system can be comprised of much simpler single-branches and each zone will then correspond to a weighted least squares sub problem. The authors in [17] propose two main parts: *load allocation and state estimation*. The load allocation portion is a real-time load modeling

technique that incorporates use of customer class curves and provides a measure of the uncertainty (statistics) in the estimates.

The purpose of the load allocation portion is to produce pseudo-measurements with a higher level of accuracy in real-time than historical data that must be retrieved from a large data repository. The state estimation portion then utilizes the pseudo-measurements that ensure observability and follows a traditional weighted least squares technique that is applied to each "zone". The authors propose that a forward/backward sweep scheme based upon this method would allow state estimation to be performed accurately for large-scale practical distribution systems while not requiring sparse matrix techniques.

2.4 Artificial Neural Networks and Deep Learning Applied to Power System State Estimation, Observability, Topology Errors and False Data Injection Attacks

The authors in [18] and [19] present *Bayesian state estimation* for unobservable distribution systems via deep learning. The paper presented in [18] is an introduction and [19] provides greater depth and simulation results. The authors in these papers propose a novel state estimation scheme that combines *Bayesian inference* with *deep neural networks* to minimize the mean squared error of network states in real-time.

Bayesian inference is used to learn the probability distributions of the net power injection from historical measurements. Samples are then generated from the learned distributions and passed to a deep neural network to approximate the minimum mean squared error (MMSE) estimate of system states.

The authors contend that this hybrid approach outperforms classical pseudo-measurement techniques that utilize averaging of historical data and conventional weighted least squares estimation. It also outperforms Bayesian state estimation alone or feedforward neural network state estimation alone. It should be noted that in [18] and [19], "deep neural

networks" are considered to have a relatively larger number of hidden layers than a "flat neural network" involving one or two layers.

For example, the authors in [18] and [19] show that for the tested 85 and 141 bus networks, a feedforward neural network of 10 layers or more achieved a mean square error (MSE) per bus at a level of 10-5 to 10-6 *p.u.* on test data, while classical methods using weighted least squares had errors several orders of magnitudes or higher. The authors point out that the proposed *Hybrid Bayesian-Deep Neural Network* approach is less capable of adapting to changes such as line and generation outages. *Additionally, the training of the deep neural network was largely ad hoc and offered little guarantee of performance.*

It was also recommended by the authors that it may prove beneficial to exploit temporal dependencies and other types of deep learning networks. Other possible deep learning networks include convolutional neural networks (CNNs), recurrent neural networks (RNNs) in general and long short-term memory networks (LSTMs) in particular. The authors in [20] also utilize a hybrid Bayesian-Deep Neural Network to achieve what is referred to as "high-resolution" and "high-fidelity" state estimation for systems that are PMU-unobservable. By *high-resolution*, the authors mean that states are estimated at the PMU-timescale. This is achieved by a Bayesian inference approach utilized in [18] and [19].

By *high-fidelity*, the authors are referring to the ability of the state estimation algorithm to mitigate bad and malicious data. The main contributions of this paper are a proposal of a hybrid Bayesian-Deep Neural Network that performs bad-data detection/cleansing and state estimation; development of a generative adversary network (GAN) to enable Bayesian state estimation and bad data detection; and an introduction of a novel deep learning approach that enables universal bad data detection (UBD) [20].

The authors in [20] refer to *universal bad data detection* as the ability to detect and cleanse bad-data without knowledge of whether the source data distributions originated

from either regular or abnormal conditions. Some historical data is assumed to be available under regular operating conditions and no abnormal data samples are available.

The Bayesian inference solution to bad-data detection is particularly advantageous for bad-data originating from false data injection attacks given that such detection is considered to be a seemingly intractable statistical inference problem. Scalability of any state estimation approach to large distribution systems is of practical significance, both from the standpoint of convergence and computational efficiency. *Thus, even methods that converge are not considered practical if they cannot ensure execution time that approaches real-time.*

The authors in [21] present a scalable distribution system state estimation approach using *surrogate modeling* with long short-term memory (LSTM) networks. Surrogate modeling in this paper refers to the use of LSTM networks that are built to take previous states and output a rough estimate of the current states which is called the surrogate.

The surrogate is a course but very fast estimate of the state variables. The surrogate therefore replaces the unit vector (flat start) traditionally used to initialize the state vector. The authors in [21] show that this approach while being suboptimal to the original problem, involves minimal computational cost and greatly reduces the cost of iterations and thus results in faster convergence.

Auto encoders are utilized to compress the input to the LSTMs and thus greatly increase the scalability. The authors demonstrate the proposed framework on IEEE-123 bus and 8500-node test feeders. The authors in [21] note that frequent topology changes often occurring in distribution systems was not considered in their study. They propose two potential solutions. The first solution involves training multiple LSTM networks responsible for each specific topology. The second solution proposes the treatment of topology status as categorical input of a model with one LSTM network to account for all topologies.

The authors in [22] investigate a method for detecting false data injection attacks in smart grids based upon deep learning. The proposed method utilizes a combination of auto encoders and *Generative Adversarial Networks* (GANs) and can be described as employing a "semi-supervised adversarial auto encoder" algorithm. Semi-supervised networks are used to deal with input datasets that are not fully labeled. The authors in [22] show that the GAN framework is effective in detection of unobservable FDIAs that would otherwise bypass conventional bad data detection methods. As seen in [21], auto encoders enable input compression and thus increase the scalability to larger distribution systems.

GANs are considered deep learning based networks that establish a min-max adversarial game between two neural networks. One network is referred to as a *generator* (G) and the other network is referred to as the *discriminator* (D). Fake samples are produced by the generator that follows the distribution of the original real samples. The discriminator distinguishes between the generated data samples and the real samples. Thus, the training of GANs is performed in two stages. The first stage involves an update to the discriminator using fixed generator parameters so that real samples may be distinguished from the generated (fake) samples. The second stage involves an update to the generator with fixed discriminator parameters to "fool" the discriminator with the generated (fake) samples.

The authors in [22] contend that the two-player game is globally optimal. The use of GANs is commonly employed in practical systems when there are limited labeled measurements available for training. Similar to the suggestion of future research in [21], the authors in [22] acknowledge that the proposed method requires collecting data from a known topology structure. Thus, when topology changes occur, it is necessary to store corresponding measurements and states along with new topology labels for subsequent data analysis.

The authors leave research related to varying topologies and varying DER penetration to future work. The author in [23] presents a study of distribution system state

estimation with LSTMs. As promoted in [22], the author in [23] proposes the use of surrogate modeling of distribution system state estimation. The author describes surrogate modeling as a means of solving problems that are challenging to solve or that exhibit low computational efficiency. The author in [23] formulates the DSSE surrogate model as a deep neural network (DNN) model to approach the problem as a regression problem and a LSTM network to approach the problem as a time-series problem.

The DNN model is a fully-connected feedforward neural network with multiple layers to differentiate it from a single or double layer neural network that is considered a "shallow neural network". As was the case in previous literature that employed LSTMs, the author in [23] also utilizes auto encoders to compress the input and thus minimize the number of LSTM blocks required for larger distribution systems.

Comparisons are made between the DNN and the LSTM in terms of RMSE and computational efficiency for both IEEE-123 bus and 8500 node systems. The author in [23] also considers the performance of the LSTM network with and without surrogate modeling. A suggestion for future research includes the effect of load profile changes on surrogate modeling due to electric vehicle (EV) charging and photovoltaic (PV) generation.

The author in [24] presents *dynamic distribution state estimation* using synchro phasor data. The proposed DSSE formulation, although assuming the availability of synchro phasor data, is novel in its linearized model and time-varying nature to account for dynamic states and data without requiring an iterative method.

The author in [24] presents a *first-order prediction-correction* (FOPC) method that relies on the Hessian of the cost function. In other words, the requirements of conventional methods of inverse computations are removed and thus, the method presented is very attractive for DSSE problems where measurements are collected at high frequency by distribution level PMUs.

The merits of the proposed FOPC method are two-fold. The first includes computational savings by estimating the state before new measurements are collected and

processed. This contrasts with conventional computationally expensive iterative algorithms without prediction steps. The second is that FOPC can perform state prediction while waiting for measurements to be transferred from PMUs. Thus, once the measurement is received, the correction step may begin immediately.

While the method presented in [24] is based upon Kalman Filtering techniques as opposed to neural networks, it is considered a modern fully data driven approach in that its prediction phase attempts to approach an optimal solution of the next time period without new observations by exploiting the temporal correlations of the cost function. The predicted vector is then corrected using the last measurement in the correction phase. The author in [24] suggests that future work explore diverse types of distribution system measurements and large-scale system validations using real data.

The authors in [25] present a deep neural network model for short-term load forecasting based on long short-term memory networks (LSTMs) and convolutional neural networks (CNNs). By short-term load forecast, the authors in [25] are referring to forecasts of loads from 24 hours to one week. The study compares the performance of models that are based upon convolutional neural networks (CNNs) only, long short-term memory networks (LSTMs) only, and hybrid-CNN-LSTM models.

CNNs extract local trends and capture patterns and they are typically used for speech recognition, image processing and other tasks on which patterns can be determined. LSTMs learn relationships from the data itself when the data is framed as a time-series dataset. In the proposed hybrid-CNN-LSTM model, there is a CNN module that captures local trends, a LSTM module that is utilized to learn long-term dependency and a feature-fusion module that concatenates the output of the CNN and LSTM modules to formulate a final prediction. The combined output of the CNN and LSTM is based upon “hidden features”. This refers to predictions and classifications that the deep learning models generate that are learned from the data itself as opposed to known categories and features being given as input.

The authors in [25] contend that hidden feature extraction results in predictions of higher accuracy than those obtained through conventional probabilistic and statistical inference. Considering the literature discussed in this section, it is reasonable to conceive a scenario in which systems with sufficient PMU availability could utilize a FOPC method as presented in [24] as the primary state estimation scheme and deep learning methods such as the LSTM based surrogate model presented in [23] and/or the hybrid-CNN-LSTM model presented in [25] as the secondary or backup method for state estimation, bad data detection and load forecasting.

2.5 Distribution System State Estimation

State estimation as it relates to power systems is defined as the vector of the voltage magnitudes and angles at all network buses [2]. Essentially, state estimation algorithms provide for a means of reducing the impacts of measurement errors, system parameter error or topology errors.

Figure 1 presents an example of both types of power networks and some of the key characteristics.

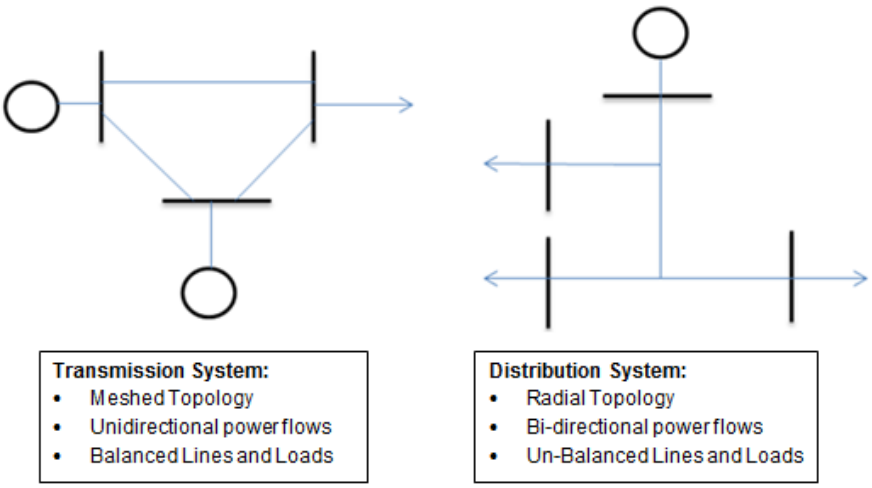


Figure 1 - Transmission and Distribution System Key Characteristics

To appreciate the challenges that the emerging smart distribution grid pose to the direct application of conventional state estimation, it is essential to first understand the inputs and functional blocks that enable state estimation. Figure 2 provides an overview of the inputs and main functional blocks.

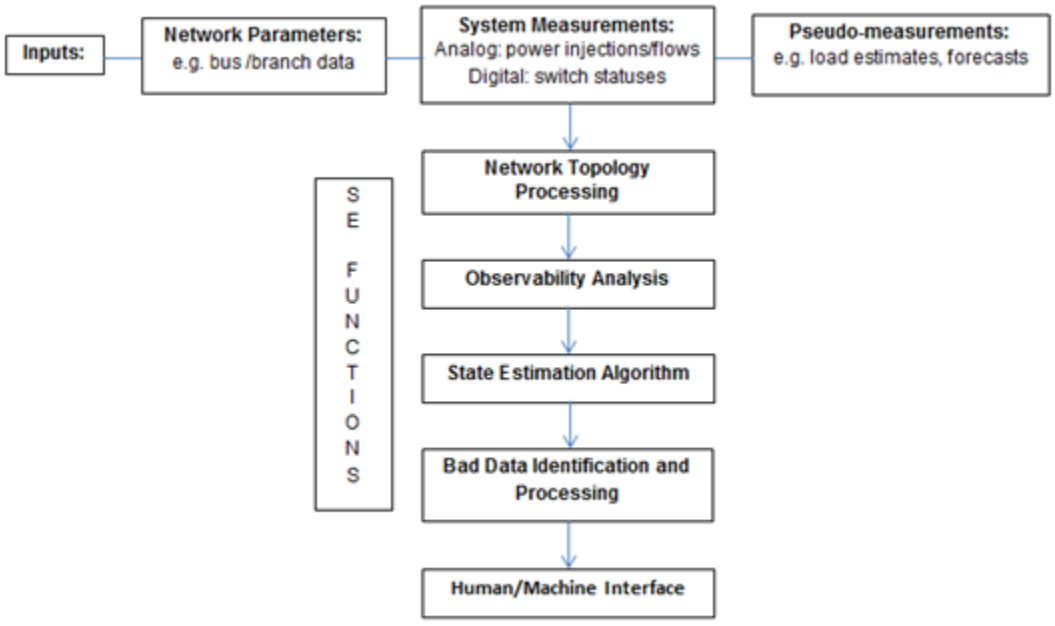


Figure 2 - Functional Block Diagram of State Estimation

Note that the *Network Topology Processing* functional block verifies the accuracy of the network parameters included as Inputs. The *Observability Analysis* functional block establishes that there is sufficient data available for the *State Estimation Algorithm* functional block. As discussed earlier, the relative lack of metering in distribution networks reduces the "observability" of the system. The ability to meet this challenge, while being improved through the implementation of "smart meters" such as PMUs (phasor measurement units), will continue to be an inherent challenge in distribution networks as opposed to transmission networks.

The *State Estimation Algorithm* functional block then seeks to determine a unique solution or system state. Also, critical to the overall state estimation functionality and final determination of the system state is the *Bad Data Identification and Processing* functional block that uses statistical techniques (i.e. Chi-square Test) to identify and filter out "noise" which may be related to inaccuracies in measurement meters and/or communication system failures. Finally, the *Human/Machine Interface* functional block relates to the software and hardware utilized to visualize and otherwise monitor and control the power system. Further challenges beyond lack of metering, are those associated with topology errors and false data injection attacks. The terms and consequences of lack of observability, topology errors and false data injection attacks will be explained in later sections.

Figure 3 summarizes the key characteristics of the "conventional" state estimator based upon weighted least squares.

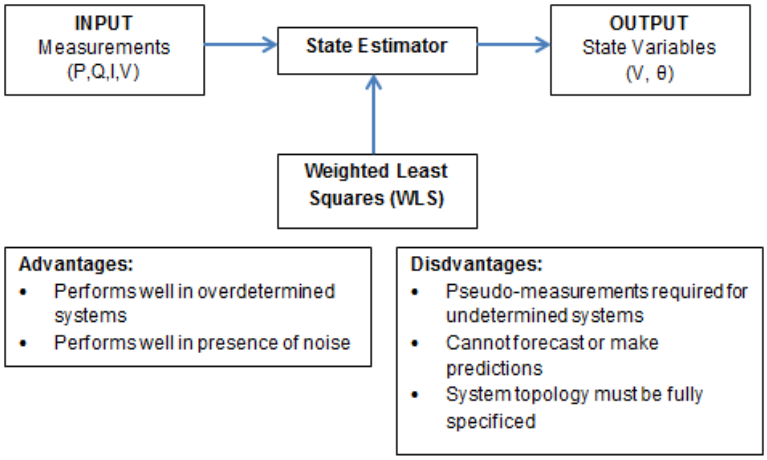


Figure 3 - State Estimator Overview

Note that the INPUT are typically measurements of P (Real Power), Q (Reactive Power), I (Current Flows), and Voltage Magnitudes. The OUTPUT state variables are typically voltage magnitudes and voltage phase angles at all buses. With these two state variables, it is then possible to determine the remaining parameters such as Real and Reactive Power Injections and Current flow.

Note that one of the buses can be established as the *reference bus* or *slack bus*. Thus, if Bus 1 is established as the reference bus, then the phase angle for Bus 1 can be removed from the vector representation. Therefore, if there are n buses in the network, the total number of states is given as $2n - 1$. It is important to note that conventional state estimation applies only to *overdetermined systems*.

Overdetermined systems are those in which the number of measurements exceeds the number of states. This critical and limiting requirement for application of conventional state estimation can be summarized in two criteria. The first criteria states that if the number of measurements is m, and the number of states is $2n-1$, then in state estimation, $m > 2n - 1$. The second criteria states that if $m = 2n - 1$, the problem reduces to a power flow solution.

Thus, as stated previously, distribution systems with limited measurement devices are inherently not overdetermined systems. For such underdetermined systems that may be either transmission or distribution networks lacking sufficient metering, observability is reduced and as indicated in Figure 3, the state estimation algorithm must rely upon pseudo-measurements for the solution to converge.

The research being presented in this dissertation seeks to contribute to the body of knowledge to mitigate this problem by incorporating neural networks and deep learning methods.

2.6 Challenges of Applying Conventional State Estimation Utilizing Weighted Least Squares to Distribution Systems

The most common conventional state estimation algorithm is based upon the Weighted Least Squares (WLS) algorithm. There are fundamental characteristics of distribution systems that pose major challenges to the direct application of conventional state estimation based upon weighted least squares. Radial topology yields bi-directional power flow. Reduced observability results from lack of adequate quality and quantity of measurement devices resulting in underdetermined systems (number of measurements less than number of system states). Unbalanced lines and loads result in the need to consider all phases in the state estimator algorithm. Unpredictability exists in energy sources injecting power back onto the grid (i.e. intermittent sunlight and wind, electric vehicles, etc.). Variability exists in the timing of power utilization throughout the day. Low X/R ratios which do not allow for neglecting resistances due to dominant inductive terms as is permissible in transmission systems. Substantial number of nodes that combined with the need to consider all phases, results in the need for acquisition, storage, and processing of substantial amounts of data. Excessive noise resulting from the variety and lack of standardization of communication schemes between metering devices and the central control stations.

It should be noted that the limitations listed above are considered "normal conditions" inherent in all distribution systems. The addition of "adverse conditions" noted previously further strengthens the case for needed research of methods such as artificial neural networks to maintain data integrity of distribution system state estimation and thus the overall resiliency of the modern power grid.

2.7 Lack of Observability in Distribution Systems

Lack of observability is directly related to the inability to accurately measure and store system values (power, voltage magnitude, voltage phase angles and current flow) of a distribution system due to lack of measurement devices, failures in devices, communication failures and/or malicious attacks that would also fall into the category of false data injection attacks.

While there are increasing advances in application of Phasor Management Units (PMUs) and so-called "smart-meters", in the research on which this dissertation is based, there will not be an assumption that these devices are available at every bus location of a practical distribution system. Thus, distribution system state estimation is fundamentally challenged by lack of observability.

2.8 Topology Errors in Distribution Systems

Topology errors are directly related to errors in determination of system state values due to inaccurate determination of system breaker status. More generally, these errors could relate to incorrect determination of any device that involves switching or tap positioning. The false status of system breakers could result from failures in devices, communication failures and/or malicious attacks that would also fall into the category of false data injection attacks. Thus, distribution system state estimation is also fundamentally challenged by topology errors.

2.9 False Data Injection Attacks in Distribution Systems

False data injection attacks refer to malicious attempts to alter data within distribution systems such that the true system state is made inaccurate. The goal of such attacks could be financial, such as controlling aspects of the power market or sabotage to the security of the power system resulting in power outages. It should be noted that with advances in smart grid metering and reliance on digital communications, the susceptibility of the power grid to false data injection attacks will continue to be a growing security concern. Thus, distribution system state estimation is also fundamentally challenged (even threatened), by false data injection attacks.

It should be noted that identification and mitigation of lack of observability, topology errors and false data injection attacks will be left to future research. These topics are mentioned in this dissertation to strengthen the case that classical methods are not sufficient to address their existence in distribution systems and data driven solutions proposed in this research may serve as a basis for addressing them directly in the future.

2.10 Conventional Feedforward Multilayer Perceptron Networks (MLPs)

This type of network is considered the "conventional" or "classical neural network model". Figure 4 shows a "perceptron", which is the fundamental building block of neural networks, where x_i through x_m represent inputs, w_i through w_m represent the associated weights, b represents a bias term and w_{ebs} is its associated weight. Input features are multiplied by weights and the resulting values are summed. The bias term allows for the adjustment of the decision boundary allowing for a better fit of the training data and more accurate predictions. The output of the summation is passed to an activation function to yield a final output y. A learning algorithm (i.e. stochastic gradient

descent) is then utilized to find the set of weights and bias that minimizes the error between the predicted output and the actual output for every training example.

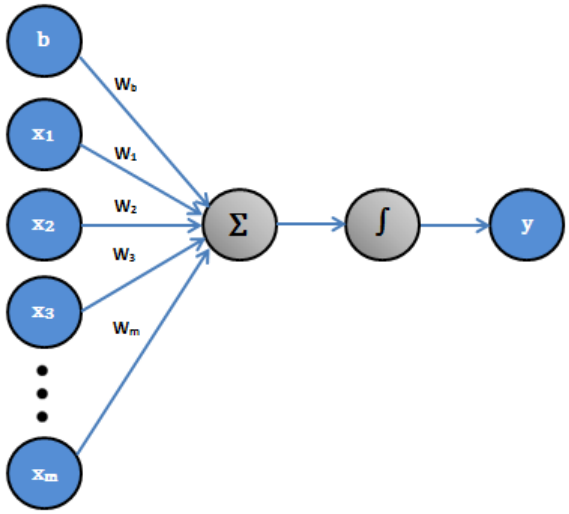


Figure 4 - Perceptron Building Block of MLP Networks

Figure 5 provides a visualization of the functional blocks of a MLP network model.

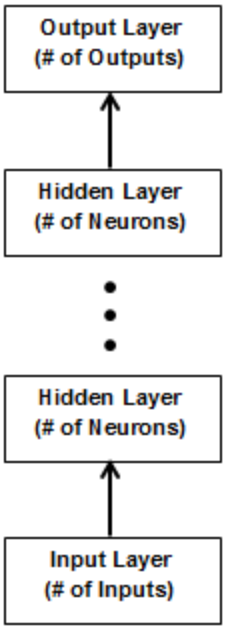


Figure 5 - Multilayer Perceptron (MLP) Model Functional Representation

This type of network is considered a reasonable model for *regression* and *classification* problems, however, it is limited in terms of its ability to predict or forecast sequence or time-series data as it does not maintain and share features between layers. This type of neural network is also limited to how "deep" they can be in terms of number of layers that would otherwise enable them to solve more complex problems with greater accuracy. Even with the noted limitations, this network type shows promise in its ability to overcome many of the limitations of weighted least squares in state estimation. The principal advantage of this network type as confirmed in this research is the promise of being able to accurately learn the mapping of inputs to outputs for a regression problem without the requirement of complex and/or large number of equations that would be necessary to perform non-linear regression on very large distribution systems.

2.11 Convolutional Neural Networks (CNNs)

This type of network is considered to be an improvement upon the classical MLP architecture in that it learns directly from the input data and thus does not require a target dataset during training. Figure 6 below, shows the general structure for a CNN model.

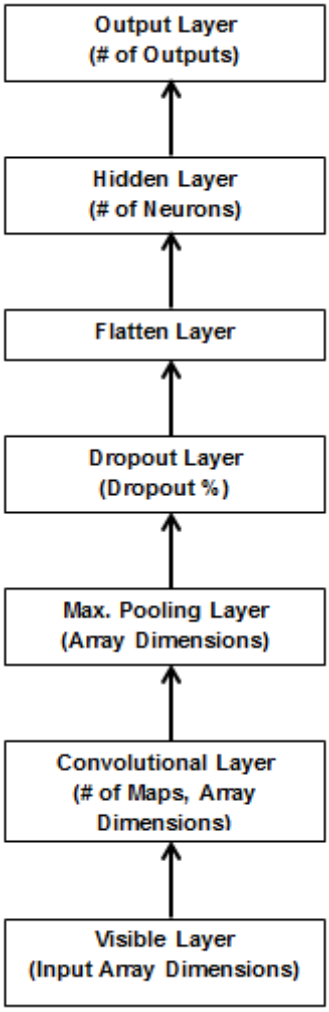


Figure 6 - Convolutional Neural Network (CNN) Model Functional Representation

Table 3 presents the fundamental CNN layer types and associated key characteristics of each.

Table 3 - CNN Layer Types and Key Characteristics

Layer Type	Key Characteristics
Convolutional Layers	Comprised of Filters and Feature Maps Filters correspond to neurons of the layer Filters have weighted inputs and produce outputs like a neuron Filters input size is fixed and is a "window" for convolution Feature Maps contain current values within the moving filter window
Pooling Layers	Down-sample and consolidate features learned from previous feature maps Serve to generalize or compress features selected and reduce overfitting of model training Simple functionality – selection of either maximum or average of input value to establish a new compressed feature map
Dropout Layers	Used between other layers to further reduce overfitting not completely eliminated by pooling layers by randomly excluding neurons Specified by a Dropout Percentage
Flatten Layers	Converts multidimensional arrays to vectors that can be sent to fully connected layers for final processing by activation functions
Fully Connected Layers	Normal flat feedforward neural network layer Contain a 'softmax' or nonlinear activation function to output probabilities of predicted classes

	Utilized at the end of network to create combinations of nonlinear features used for predictions

While primarily used in image/object detection and classification, computer vision and natural language processing, this network has also gained interest in its ability to automatically learn and generalize features from time-series data.

2.12 Recurrent Neural Networks (RNNs)

This type of network is also considered to be an improvement upon the classical MLP architecture in that it maintains an internal state (memory). Table 4 presents three primary variants of RNNs and key characteristics of each.

Table 4 - RNN Variants and Key Characteristics

RNN Variant	Key Characteristics
Bidirectional Recurrent Neural Networks (BRNN)	RNNs that utilize future data along with data from previous inputs to improve accuracy
Long Short-Term Memory Networks (LSTM)	Discussed in more detail in the section 2.13
Gated Recurrent Units (GRUs)	Like LSTMs, overcome short-term memory limitations of the basic RNN model Uses hidden states instead of "cell state" utilized by LSTMs Contains reset and update gates to control what information is retained and how much of this information to use for making predictions

This network has also gained interest due to its ability to automatically learn and generalize features from time-series data. Additionally, it maintains and passes features between layers and thus very deep structures can be developed without the negative effects of exploding or vanishing gradients.

2.13 Long Short-Term Memory Networks (LSTMs)

This network is a type of RNN that can learn long-term dependencies between time steps of input sequence data by "remembering" the state between predictions. In the research being presented, LSTMs will represent the more general RNN network and other RNN variants will not be considered. Table 5 provides more detail on the internal architecture of the LSTM unit.

Table 5 - LSTM Operations and Purpose

LSTM Operations	Purpose
Step 1 – "Forget Gate"	Determines and eliminates previous information deemed as irrelevant and thus not useful
Step 2 – "Store Gate"	Determines what new information to maintain as new candidate values
Step 3 – "Update Gate"	Updates old cell state to new cell state Then scales new candidate values by how much it was decided to update each state value
Step 4 – "Output Gate"	Determines what is to be output for the next step Output will be based on cell state, but will be a filtered version First run a *sigmoid* layer to decide what parts of a cell state to output Then put the cell state through a *tanh* activation function to push values between -1 and 1 and multiply it by the output of the sigmoid gate so that only the desired parts are output

CHAPTER 3. DISTRIBUTION SYSTEM STATE ESTIMATION (DSSE) WITH MULTILAYER PERCEPTRON (MLP) MODELS

3.1 DSSE with MLPs without Hyperparameter Optimization

Conventional MLP Models were utilized to perform regression to map power data (real and reactive power) as inputs to voltage (voltage magnitudes and phase angles) as outputs. Utilization of MLPs for this purpose is reasonable given the ability of a suitable MLP to perform as a "universal function approximator". *The Universal Approximation*

Theorem states that a neural network with a single hidden layer can approximate any arbitrary continuous function given a sufficient number of neurons. This approach, however, does not take into consideration *time-series features* of data such as trend, seasonality, and residual noise. For purposes of training a *supervised* MLP neural network to perform regression, it was decided that the power (real and reactive) at each bus for all 3 phases would be measured and deemed the "input" dataset. The voltage and phase angle at each bus for all 3 phases was selected to be measured and deemed the "target" dataset. Note that the voltage dataset will sometimes be referred to as "voltage_output" as this dataset is an output of a power flow simulation. Table 6 presents a summary of the supervised learning datasets and general structure of power flow, training, testing and validation datasets.

Table 6 - Supervised Learning Datasets and General Structure

Supervised Learning Data Sets	General Structure: (#samples, #features)
Initial Power Flow Data	Input Power Dimensions: (8760, 56) Target Voltage Dimensions: (8760, 56)
Training Data (70% Split)	Input Power Training Dimensions: (6132, 56) Output Voltage Training Dimensions: (6132, 56)
Testing Data (30% Split)	Input Power Testing Dimensions: (2628, 56) Output Voltage Testing Dimensions: (2628, 56)
Validation Data (New Power Flow with different load profile)	Input Power Validation Dimensions: (8760, 56) Output Voltage Validation Dimensions: (8760, 56) – Predicted

3.2 Test Distribution System

An IEEE 34 Bus Test Feeder radial distribution system was selected as the base test distribution system [13]. It is shown below in Figure 7.

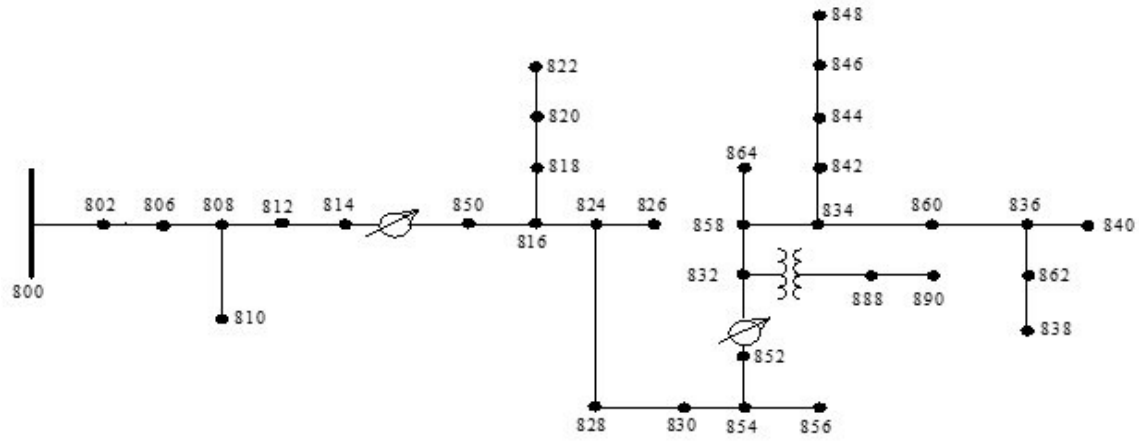


Figure 7 - IEEE 34 Node Test Base Distribution System

3.3 Power Flow Simulation Measurement Points and Quantities

The selected measurement points and quantities are shown in Figure 8. The labels corresponding to the "Key:" represent either a power or voltage "monitor", which is similar to a physical meter and will be discussed in more detail later in this dissertation.

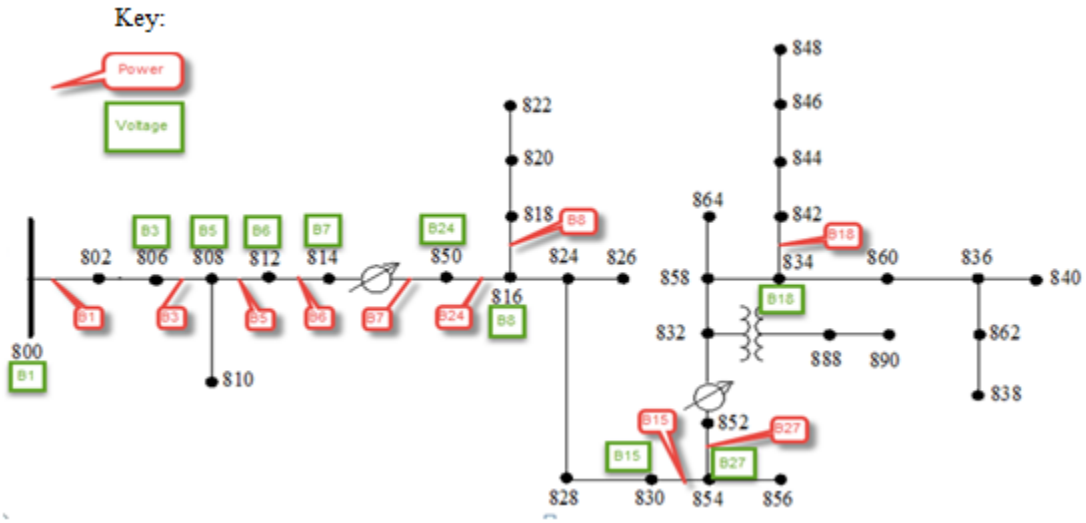


Figure 8 - IEEE 34 Node Test Base Distribution System Measurement Points

Table 7 and Table 8 provide a description of the monitors, their locations in the test distribution systems and the quantities they measure. Note that power monitors capture

the real and reactive power flow along the lines between specific nodes as indicated in Table 7. Likewise, voltage monitors capture the voltage magnitude and voltage phase angle at specific nodes as indicated in Table 8.

Table 7 - Power Monitor Descriptions and Locations

Monitor:	Line Element:	From Node:	To Node:	Quantities:
B1_power	L1	800	802	Phase A,B,C P (kW) and Q(kVAR)
B3_power	L3	806	808	Phase A,B,C P (kW) and Q(kVAR)
B5_power	L5	808	812	Phase A,B,C P (kW) and Q(kVAR)
B6_power	L6	812	814	Phase A,B,C P (kW) and Q(kVAR)
B7_power	L7	814	850	Phase A,B,C P (kW) and Q(kVAR)
B24_power	L24	850	816	Phase A,B,C P (kW) and Q(kVAR)
B8_power	L8	816	818	Phase A P(kW) and Q(kVAR)
B15_power	L15	830	854	Phase A,B,C P (kW) and Q(kVAR)
B18_power	L18	834	842	Phase A,B,C P (kW) and Q(kVAR)
B27_power	L27	854	852	Phase A,B,C P (kW) and Q(kVAR)

Table 8 - Voltage Monitor Descriptions and Locations

Monitor:	Line Element:	Node:	Quantities:
B1_voltage	L1	800	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
B3_voltage	L3	806	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
B5_voltage	L5	808	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
B6_voltage	L6	812	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
B7_voltage	L7	814	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
B24_voltage	L24	850	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
B8_voltage	L8	816	Phase A to Neutral Voltage Mag. (V) and Phase Angle (degrees)
B15_voltage	L15	830	Phase A,B,C to Neutral

			Voltage Mag. (V) and Phase Angle (degrees)
B18_voltage	**L18**	**834**	**Phase A,B,C to Neutral** **Voltage Mag. (V) and Phase Angle (degrees)**
B27_voltage	**L27**	**854**	**Phase A,B,C to Neutral** **Voltage Mag. (V) and Phase Angle (degrees)**

3.4 Training and Testing Data

For purposes of performing a power flow simulation of the test feeder system to gather the power and voltage at each monitor location, Upends (Open Distribution System Simulator version 8.5.9.1 64-bit build) from Electric Power Research Institute, Inc. (EPRI) was chosen [43]. Note that the convention in OpenDSS is that Phase-1, Phase-2 and Phase-3 represent phases a, b, and c respectively. It was decided that the loads within the test distribution feeder would be varied over a time period of a year (8760 hours) to yield a multivariate time-series dataset corresponding to the power and voltage as discussed previously.

To vary the base loads in a realistic manner, historical data from the Electric Reliability Council of Texas (ERCOT) was obtained [26]. The load data for the entire ERCOT grid for every hour of the entire year of 2018 was selected. The ERCOT load dataset was then used to realistically scale the power (P and Q values) at each node that contains a load to establish the needed variation over a period of a year. Note that "ERCOT" is used as the

baseline load profile and all references to ERCOT datasets have their origin from the baseline power flow simulation just described.

OpenDSS was then utilized to perform a power flow simulation of the test feeder distribution system and the resulting power and voltage datasets were exported. This exported data serves as the input and target datasets from the test system under *normal conditions*. Normal conditions mean that the data collected is not subjected to excessive noise and/or other conditions such as false data injection attacks. Training and testing of the neural network models described later in this dissertation are based upon this simulation data.

3.5 Validation Data

It is essential that a trained neural network model be *validated with data never seen by the network*. Again historical data from the Electric Reliability Council of Texas (ERCOT) was obtained. The new data is sample data from the same year as the training and testing datasets described previously. However it originates from data selected from another region within the ERCOT grid. The previous steps related to performing a power flow simulation with OpenDSS were repeated with a new load profile to establish previously unseen data for validating the various neural network types. Note that "COAST" will be used in descriptions of datasets that have their origin from the power flow simulation of the test distribution system performed with varying loads according to this load profile.

3.6 Unoptimized Conventional Feedforward Multilayer Perceptron Network (MLP)

3.6.1 Network Model Parameters

- *Number of neurons in visible layer (input layer): 56 (Held constant for Trials 1 – 11)*
    - *Power Monitors:*
        - *B1_power: 6 features (P and Q values for 3 phases)*
        - *B3_power: 6 features (P and Q values for 3 phases)*
        - *B5_power: 6 features (P and Q values for 3 phases)*
        - *B6_power: 6 features (P and Q values for 3 phases)*
        - *B7_power: 6 features (P and Q values for 3 phases)*
        - *B24_power: 6 features (P and Q values for 3 phases)*
        - *B8_power: 2 features (P and Q values for 1 phase)*
        - *B15_power: 6 features (P and Q values for 3 phases)*
        - *B18_power: 6 features (P and Q values for 3 phases)*
        - *B27_power: 6 features (P and Q values for 3 phases)*
- *Number of hidden layers and number of neurons per hidden layer:*
    - *Adjusted for each trial according to* Table 10
- *Number of neurons in output layer: 56 (Held constant for Trials 1 – 11)*
    - *Voltage Monitors:*
        - *B1_voltage: 6 features (Mag. and Phase values for 3 phases)*
        - *B3_power: 6 features (Mag. and Phase values for 3 phases)*
        - *B5_power: 6 features (Mag. and Phase values for 3 phases)*
        - *B6_power: 6 features (Mag. and Phase values for 3 phases)*
        - *B7_power: 6 features (Mag. and Phase values for 3 phases)*
        - *B24_power: 6 features (Mag. and Phase values for 3 phases)*
        - *B8_power: 2 features (Mag. and Phase values for 2 phases)*
        - *B15_power: 6 features (Mag. and Phase values for 3 phases)*
        - *B18_power: 6 features (Mag. and Phase values for 3 phases)*
        - *B27_power: 6 features (Mag. and Phase values for 3 phases)*

3.6.2 Network Hyperparameters

- *Activation Function per Layer: Rectified Linear Unit (ReLU)*
- *Loss Function: Mean Squared Error (MSE)*
- *Optimizer: Stochastic Gradient Descent (SGD)*
- *Learning Rate: 0.001*
- *Batch Size = 10*

- *Epochs = 200*

3.7 Implementation of Unoptimized MLP Models for DSSE

The following section covers the essential elements required to perform distribution system state estimation with an unoptimized MLP model. For brevity, only Trial 1 will be shown.

Code Listing 1 - Unoptimized MLP Model for DSSE

```
000 # MLP_Trial_01_Unoptimized.py
001 # Input Layer: 56 Neurons; 1 Hidden Layer: 56 Neurons; Output
Layer: 56 Neurons
002
003
004 import time
005 start_time = time.time()
006
007 # Import Required Python Libraries
008 from pandas import read_csv
009 import math
010 import numpy
011 from keras.models import Sequential
012 from keras.layers import Dense
013 from sklearn.model_selection import train_test_split
014 from sklearn.metrics import mean_squared_error
015 from sklearn.preprocessing import MinMaxScaler
016
017 # Data Normalization
018 scaler_power_ercot = MinMaxScaler(feature_range=(-1,1),copy=True)
019 scaler_voltage_ercot = MinMaxScaler(feature_range=(-1,1),copy=True)
020
021 scaler_power_coast = MinMaxScaler(feature_range=(-1,1),copy=True)
022 scaler_voltage_coast = MinMaxScaler(feature_range=(-1,1),copy=True)
023
024 # Load Datasets
025
026 # EROCT Data used to Train and Test MLP
027 input_power_ercot_dataframe =
read_csv('C:/Python/input_power_ercot.csv', header=None)
028 input_power_ercot = input_power_ercot_dataframe.values
029 input_power_ercot_normalized =
scaler_power_ercot.fit_transform(input_power_ercot)
030
031 output_voltage_ercot_dataframe =
read_csv('C:/Python/output_voltage_ercot.csv', header=None)
032 output_voltage_ercot = output_voltage_ercot_dataframe.values
033 output_voltage_ercot_normalized =
scaler_voltage_ercot.fit_transform(output_voltage_ercot)
```

```
034
035 # Coast Data used to Validate MLP
036 input_power_coast_dataframe =
read_csv('C:/Python/input_power_coast.csv', header=None)
037 input_power_coast = input_power_coast_dataframe.values
038 input_power_coast_normalized =
scaler_power_coast.fit_transform(input_power_coast)
039
040 output_voltage_coast_dataframe =
read_csv('C:/Python/output_voltage_coast.csv', header=None)
041 output_voltage_coast = output_voltage_coast_dataframe.values
042 output_voltage_coast_normalized =
scaler_voltage_coast.fit_transform(output_voltage_coast)
043
044 # Split Data Into 70% for Train and 30% for Test
045 power_train, power_test, voltage_train, voltage_test =
train_test_split(
046     input_power_ercot_normalized, output_voltage_ercot_normalized,
test_size=0.30)
047
048 # Define the MLP Model
049 model = Sequential()
050 model.add(Dense(56, input_dim=56, activation='relu'))
051 model.add(Dense(56))
052
053 # Compile the MLP Model
054 model.compile(loss='mean_squared_error', optimizer='sgd')
055
056 # Fit the MLP Model
057 model.fit(power_train, voltage_train,
validation_data=(power_test,voltage_test),
058           epochs=200, batch_size=10, verbose=0)
059
060 # Evaluate the MLP Model
061 trainScore = model.evaluate(power_train, voltage_train, verbose=0)
062 print('Train Score: %.6f MSE (%.6f RMSE)' % (trainScore,
math.sqrt(trainScore)))
063 testScore = model.evaluate(power_test, voltage_test, verbose=0)
064 print('Test Score: %.6f MSE (%.6f RMSE)' % (testScore,
math.sqrt(testScore)))
065
066 print(' ')
067
068 # Make Predictions with the Model
069 coastScore =  model.evaluate(input_power_coast_normalized,
output_voltage_coast_normalized,
070                              verbose=0)
071 print('Coast Score: %.6f MSE (%.6f RMSE)' % (abs(coastScore),
math.sqrt(abs(coastScore))))
072
073 print(' ')
074
075 print("--- %s seconds ---" % (time.time() - start_time))
```

Table 9 presents a summary of Code Listing 1.

Table 9 - Summary of Code Listing 1 for Unoptimized MLP Model for DSSE

Line #:	Description:
007 - 015	Import essential Python libraries for implementation of an unoptimized MLP model for DSSE
017 - 022	Define "scalar" objects to perform data normalization on source data to aid in training performance.
026 - 033	Import ERCOT data used to train and test the MLP. Perform data normalization.
035 - 042	Import COAST data used to validate MLP. Perform normalization.
044 - 046	Split input data into 70% for training and 30% for testing.
048 - 051	Define unoptimized MLP model
054	Compile unoptimized MLP model
056 – 058	Fit MLP with unoptimized hyperparameters
060 - 064	Evaluate the unoptimized MLP on training and testing data in terms of RMSE.
068 - 071	Perform DSSE with unoptimized MLP on COAST validation data and report RMSE.

The following results are for MLP models trained and tested on ERCOT data and validated on COAST data.

Table 10 presents training, testing and validation root-mean squared errors for eleven MLP model architectures. Note that each "Trial" represents a separate MLP model. As indicated in Table 10, the number of hidden layers and number of hidden layer neurons were varied. The number of input and output layer neurons was held constant at 56 neurons to correspond to the number of input and output features.

As indicated in this table, 70% of the ERCOT data was used for training and 30% was held out for testing. These columns represent the "training error" and "testing error" respectively for each model. The "COAST Act. vs. Est" column shows results for the various architectures of the MLP when predicting output voltages and phase angles for COAST data that has never been seen by the neural network. This column represents the "validation error" for each model. Note that RMSE (root mean squared error) throughout this research is calculated based upon the following equation.

$$RMSE = \sqrt{\frac{1}{n}\sum\left(y_{true} - y_{predicted}\right)^2} \quad (1)$$

Where y_{true} is an array of true target values, $y_{predicted}$ is an array of predicted target values and n is the total number of samples.

It should also be noted that input data to all neural networks in this research were *standardized* to be in the range of -1 to +1 prior to training, testing and validation. This transformation was performed by the MinMaxScaler function available in Python sklearn library [68].

The default transformation utilizing this function is a range of 0 to +1, however the following equation is used when scaling data to be in the range of -1 to +1:

$$X_{scaled} = \left(\frac{X - X_{min}}{X_{max} - X_{min}}\right) * 2 - 1 \quad (2)$$

Where X is the original value, X_{scaled} is the scaled value, X_{min} is minimum value of the dataset, and X_{max} is the maximum value of X in the dataset.

This formula first scales the feature to be in a range between 0 and +1 using the original *MinMaxScaler* formula, then it multiplies the result by 2 and subtracts 1 to scale the range to -1 to +1. Given that the input data is standardized according to equation 2, all RMSE values in Table 10 and Table 17 do not carry a physical unit. The RMSE values presented in CHAPTER 5, CHAPTER 6 and CHAPTER 7 that consider time-series forecasting of a univariate electrical quantity (Real Power, Reactive Power, Voltage Magnitude and Voltage Phase Angle) carry the associated units (kWatts, kVARs, Volts and Degrees respectively).

Table 10 - Performance Results for MLP Models without Hyperparameter Optimization

Trial	Input Layer	Hidden Layers	Output Layer	Train RMSE (70%)	Test RMSE (30%)	COAST Act. vs. Est. RMSE
1	56 Neurons	1 Layer 56 Neurons	56 Neurons	0.140927	0.142162	0.323075
2	56 Neurons	1 Layer 112 Neurons	56 Neurons	0.140486	0.136711	0.323293
3	56 Neurons	1 Layer 224 Neurons	56 Neurons	0.137222	0.137328	0.322124
4	56 Neurons	10 Layers 56 Neurons	56 Neurons	0.092110	0.092036	0.318610
5	56 Neurons	10 Layers 112 Neurons	56 Neurons	0.091231	0.090394	0.317231

6	56 Neurons	10 Layers 224 Neurons	56 Neurons	0.089581	0.089328	0.333226
7	56 Neurons	20 Layers 56 Neurons	56 Neurons	0.085845	0.087380	0.309943
8	56 Neurons	20 Layers 112 Neurons	56 Neurons	0.082363	0.082748	0.314640
9	56 Neurons	20 Layers 224 Neurons	56 Neurons	0.077807	0.078516	0.314956
10	56 Neurons	50 Layers 224 Neurons	56 Neurons	0.076469	0.078396	0.313638
11	56 Neurons	100 Layers 224 Neurons	56 Neurons	0.079144	0.079668	0.307243

3.8 State Estimation and Forecasting Based Upon Time Series Physics Aware Models

To add "awareness" of temporal dynamics and physics inherent in power systems, weather data (hourly temperatures at Dallas/Fort Worth International Airport for the entire 2018 year) was obtained [27]. The time frame was selected to correspond with the original ERCOT datasets discussed previously. Figure 9 provides an example of the hourly temperature and real power demand for Dallas/Fort Worth for January 1, 2018.

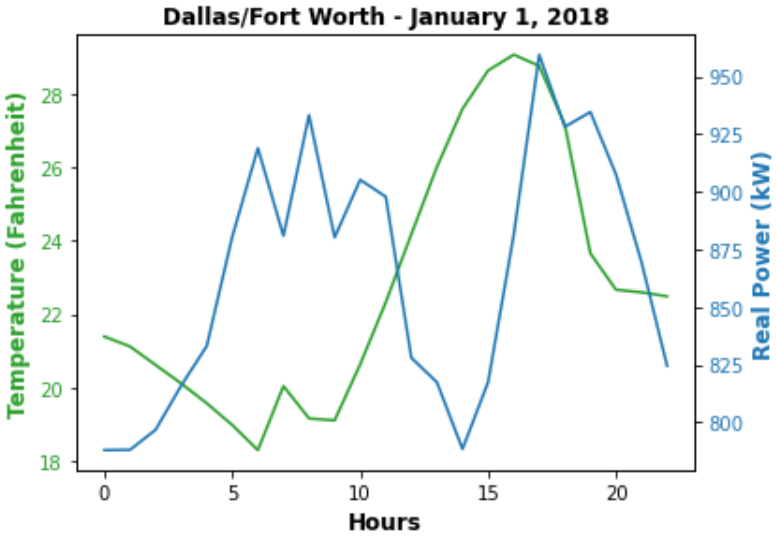


Figure 9 - Hourly Temperature and Real Power Demand

Figure 9 shows that on January 1, 2018, in Dallas/ Fort Worth there appears to be a "negative correlation" between temperature and real power demand. Thus there exists a relationship in which the changing temperature throughout the day results in an increase or decrease in the real power demand owing to energy needed for maintaining heating resources. Thus, it was decided that temperature could be used to aid in prediction of real and reactive power flow, and subsequently voltage magnitude and voltage phase angle for various forecasting time horizons.

The datasets utilized in this research were restructured such that each "row" of data would correspond to a time element (hour) and each "column" of data would represent a unique time-series of measurement quantities or "features". To simplify the preliminary predictive model datasets, it was decided that only the power and voltage data associated with the substation bus would be considered. This corresponds to the power and voltage data collected at Bus 800 with OpenDSS monitor B1 shown in Figure 8 and is utilized for training and testing. Weather data was utilized to train this network type to predict power

demand. Thus, an MLP model was used to learn the mapping of weather data as input and power as output through regression. Then power data (real and reactive powers) were then utilized to predict the voltage data (voltage magnitudes and phase angles). The results of an unoptimized MLP model utilized to forecast the real power (P), reactive power (Q), voltage magnitude (V_mag), and voltage phase angle (V_phase) at the substation bus of the test feeder distribution system shown in Figure 8 are presented in Table 11, Table 12, and Table 13 for various forecast horizons. These tables also include the results of utilizing an unoptimized CNN and unoptimized LSTM model for the same purpose and will be discussed in more detail in CHAPTER 5 and CHAPTER 6 that cover time-series forecasting with CNNs and LSTMs respectively.

Table 11 - 24 Hour Forecast (Average RMSE) at Substation Bus

Model	Real Power – 24 Hour Forecast Avg. RMSE (kWatts)	Reactive Power – 24 Hour Forecast Avg. RMSE (kVARs)	Voltage Magnitude – 24 Hour Forecast Avg. RMSE (Volts)	Phase Angle – 24 Hour Forecast Avg. RMSE (Degrees)	Execution Time (Seconds)
MLP	142.165531	173.737340	0.042655	0.000119	180
CNN	51.852689	174.998276	70.833594	0.000324	60
LSTM	76.419820	172.469473	25.603404	0.000109	7200

Table 11 shows that for a 24 hour forecast horizon, that the unoptimized MLP model yielded a significantly higher RMSE for real power than the CNN and LSTM models. All three models performed similarly for reactive power. The MLP model yielded a

significantly lower RMSE for voltage magnitude than the CNN and LSTM models and all three models performed similarly for phase angle in terms of RMSE. Note that for a 24 hour forecast horizon, the CNN model resulted in the lowest execution time followed by the MLP and LSTM model. Note that *execution time* is the total time for the models to import training, testing and validation data, perform training and testing and perform predictions for the given forecast horizon.

Extending the forecast horizon to 168 hours and 672 hours, the data in Table 12 and Table 13 respectively show relatively better performance of the MLP model when forecasting the voltage magnitude and voltage phase angle than the CNN and LSTM models considered. It should also be noted that extending the forecast horizon resulted in longer execution times for the CNN model and an inability of the LSTM model to converge. The MLP model did not show an increase in execution time as the forecast horizon was extended.

Table 12 - 168 Hour Forecast (Average RMSE) at Substation Bus

Model	Real Power – 168 Hour Forecast Avg. RMSE (Watts)	Reactive Power – 168 Hour Forecast Avg. RMSE (VARs)	Voltage Magnitude – 168 Hour Avg. RMSE (Volts)	Phase Angle – 168 Hour Forecast Avg. RMSE (Degrees)	Execution Time (Seconds)
MLP	181.882021	176.532732	0.046121	0.000144	180
CNN	92.732562	172.081075	40.681005	0.000139	120
LSTM	----	----	----	----	No Convergence

Table 13 - 672 Hour Forecast (Average RMSE) at Substation Bus

Model	Real Power – 672 Hour Forecast Avg. RMSE (Watts)	Reactive Power – 672 Hour Forecast Avg. RMSE (VARs)	Voltage Magnitude – 672 Hour Avg. RMSE (Volts)	Phase Angle – 672 Hour Forecast Avg. RMSE (Degrees)	Execution Time (Seconds)
MLP	134.544553	174.901558	0.043063	0.000112	180
CNN	165.820596	172.774166	45.946654	0.000123	240
LSTM	----	----	----	----	No Convergence

3.9 DSSE with MLPs with Hyperparameter Optimization

The exact model, approximate model and practical implementation of power system state estimation based upon weighted least squares is presented in [2], [7], and [8] respectively. The proposed methods are considered "classical" or "analytical" approaches and have found application mainly in power transmission systems.

The authors in [44] consider the benefits of machine learning to mitigate challenges of applying analytical approaches to power distribution system state estimation. While machine learning algorithms show great promise in a vast array of power system applications, widespread acceptance and application to mission critical functions will require stronger performance guarantees, certifications and elimination of vulnerabilities inherent in systems built upon data driven models [45].

The emerging smart grid will require increased situational awareness and the necessity to utilize vast amounts of data in real-time. To this end, phasor management units (PMUs)

will play an increasing role in ensuring that data is collected in a timely and accurate manner. An overview of the history, benefits of application and initiatives to increase PMU penetration in the smart grid can be found in [46] and [47].

Widespread penetration of PMUs will enable increased accuracy of classical state estimation methods by effectively linearizing highly non-linear relationships that make numerical and iterative solutions necessary for power flow and state estimation equations. This is especially true for power distribution systems. Given the early stages of PMU penetration initiatives and prohibitive costs [48], the research on which this dissertation is based does not assume widespread availability of PMUs in distribution systems.

The authors in [44] consider the application of feedforward multilayer perceptron models (MLPs) to power distribution system state estimation (DSSE). These models are considered "unoptimized" in the sense that model hyperparameters such as number of neurons per hidden layer, learning rate, number of training epochs, and training batch size were selected by an ad-hoc approach.

A key observation from [44] is that for the MLP models considered without hyperparameter optimization, there appears to be no significant improvement in performing power distribution system state estimation by utilizing more complicated networks in terms of number of layers and number of neurons per hidden layer. Further, the ad-hoc approach to hyperparameter selection, lacks constraints on the ranges of hyperparameters that could be chosen, thus increasing uncertainty regarding convergence, training execution time and hindering a suitable performance guarantee required for mission critical functions such as state estimation.

To avoid the shortcomings of selecting hyperparameters on an ad-hoc basis, the current research seeks to incorporate a suitable optimization technique. The author's in [49] present a comparison of three common approaches to hyperparameter selection. *Grid search* is considered a traditional method that involves a complete search over a given subset of the available hyperparameter space. *Random search* overrides the complete selection of all combinations by their random selection. Genetic algorithms are used to solve optimization and modeling problems by sequentially selecting, combining and varying parameters using mechanisms that resemble biological evolution.

Grid search involves evaluation of all options, even those considered "bad" options. Random search, while considered a better option than grid search, does not learn from previous samples selected and thus is not considered an optimization approach in the sense of converging on an optimal solution with each iteration. Genetic algorithms involve excessive training time and are better suited for very large hyperparameter spaces.

The author in [10] presents a method of Bayesian Optimization with Gaussian Processes that could be considered an improvement upon both grid search and random search methods in terms of providing a structured approach to hyperparameter optimization that considers prior selections and avoids combinations already shown to result in poor performance.

3.9.1 Bayesian Optimization with Gaussian Processes

Bayesian Optimization is based in part on the work of English clergyman Thomas Bayes and in particular his theory of probability given in 1764 [32]. Bayes Theorem establishes a methodical way of calculating conditional probability [33].

Bayes Theorem is given by the following relationship:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad 3$$

Where P(A|B) is the probability of an event A occurring, given that event B has already occurred, P(B|A) is the probability of an event B occurring, given that event A has already occurred, P(A) is the probability of an event A occurring, and P(B) is the probability of an event B occurring.

Often this equation is stated qualitatively as *"Posterior = Prior multiplied by Likelihood over Marginal Probability)"*. Thus, Bayes Theorem provides a way to calculate the probability of a hypothesis ("belief") based on its prior probability, the probabilities of observing various data given the hypothesis ("belief") and the set of observed data itself. [35].

Bayes Theorem is used in machine learning optimization techniques to make use of "prior" selections to improve on the selections of future values. This results in establishing regions of higher probability of maximizing (minimizing) an objective function and avoiding excess sampling in regions already deemed less likely to result in a global maximum (minimum).

Bayesian Optimization with Gaussian Processes is a technique that has gained significant interest in applications on which objective functions are considered "expensive" in terms of risk for reward or are otherwise intractable by other optimization techniques.

By *intractable*, candidate objective functions are those in which first and second order derivatives are not easily evaluated. Thus, techniques built upon variations of Newton's

method and gradient descent are not considered feasible options. Bayesian Optimization with Gaussian Processes is among a class of optimization techniques referred to as "derivative free" [34]. Thus, the function itself is considered and not first and second order derivatives [28]. It should be noted that there is research into the application of Bayesian Optimization that does take advantage of derivative information when available and is worthy of future research to compare the performance of the method proposed by the authors in [41] to gradient based methods such as stochastic gradient descent utilized in MLP to optimize weights and biases during the training process.

Applications on which Bayesian Optimization with Gaussian Processes (derivative free) may be advantageous are oil drilling (very expensive to probe, but potentially high return); evaluation of drug candidate efficacy (possible danger in its initial use, but potentially helpful to society); and hyperparameter optimization (computationally expensive to perform simulations).

The technique has the added benefit of being tolerant to *stochastic noise* during the optimization process. Thus, there is promise that applying Bayesian Optimization will reduce the inherent stochastic performance of MLP models and move toward the goal of establishing stronger *performance guarantees* and/or *certifications* needed for more widespread acceptance of machine learning as applied to mission critical applications such as power system state estimation and state forecasting. The process involves first building a “surrogate” for the original objective function often called the "acquisition function". The *acquisition function* is more tractable than the original objective function and an algorithm based upon Bayes Theorem and Gaussian process regression is employed to estimate the next search point based upon a score of the most probable

region of the parameter space given prior scores of previously sampled regions. It should be noted that this optimization technique is not considered appropriate for applications on which curve fitting and linear programming methods are sufficient.

The goal of the optimization technique is to seek a global maximum (or global minimum). Figure 10 presents basic pseudo-code for the Bayesian Optimization algorithm utilized in this research [28].

Algorithm 1 Basic pseudo-code for Bayesian optimization

Place a Gaussian process prior on f
Observe f at n_0 points according to an initial space-filling experimental design. Set $n = n_0$.
while $n \leq N$ **do**
  Update the posterior probability distribution on f using all available data
  Let x_n be a maximizer of the acquisition function over x, where the acquisition function is computed using the current posterior distribution.
  Observe $y_n = f(x_n)$.
  Increment n
end while
Return a solution: either the point evaluated with the largest $f(x)$, or the point with the largest posterior mean.

Figure 10 - Bayesian Optimization Algorithm

As shown in Figure 10, central to Bayesian optimization is the construction of a *posterior distribution of functions* also known as a *Gaussian process prior* that best describes the objective function to be optimized. As the number of observations grows, the posterior distribution improves, and the algorithm becomes more certain of which regions in the parameter space are worth exploring.

Mathematically, a gaussian process will be defined by a *mean* and *covariance* function, *m* and *k* respectively [42]:

$$f(x) \sim GP\big(m(x), k(x, x')\big) \qquad 4$$

Where, f(x) is an arbitrary function, GP represents a gaussian process or distribution over functions completely specified by its mean function m(x) and covariance function, k(x, x'). For any arbitrary x, the GP returns the mean and variance of a normal distribution.

For simplicity, the prior *m(x)* is often set to zero and the *squared exponential function* is often selected as the covariance function *k*.

$$k(x_i, x_j) = \exp\left(-\frac{1}{2}\left\|x_i - x_j\right\|^2\right) \quad (5)$$

Where x_i and x_j represent two independent samples. Points close in proximity are expected to have much larger influence on each other than those that are further apart. As values get closer together, the function approaches 1 and approaches 0 as the values get further apart.

Note that this function has the property that points close in proximity have a much larger influence over each other than those that are more distant thus aiding in convergence. Also note that function k is often referred to as the *kernel* [42]. Equation 5 provides a very simple version of the *squared exponential function* and there are many other more sophisticated alternatives that include hyperparameters for controlling "smoothness" and variance [40].

Note in the Bayesian Algorithm presented in Figure 10 that the *acquisition function* is key to determining the *next region or point to sample*. Several approaches to the selection of the next point to be explored are as follows:

- Upper Confidence Bound (UCB)

- Lower Confidence Bound (LCB)
- Probability of Improvement (PI)
- Expected Improvement (EI)

Upper Confidence Bound (UCB) concerns maximization and is described mathematically as the following:

$$UCB(x) = \mu(x) + \kappa\sigma(x) \ \ where\ \kappa \geq 0 \qquad 6$$

Where μ(x) represents the "mean function", σ(x) represents the variance function and κ is a scale factor to establish the width of the confidence interval. Note that the "+" operation establishes the upper bound of the confidence interval.

Lower Confidence Bound (LCB) concerns minimization and is described mathematically as the following:

$$LCB(x) = \mu(x) - \kappa\sigma(x) \ \ where\ \kappa \geq 0 \qquad 7$$

Where μ(x) represents the "mean function", σ(x) represents the variance function and κ is a scale factor to establish the width of the confidence interval. Note that the "-" operation establishes the lower bound of the confidence interval.

Probability of Improvement (PI) is described mathematically as the following:

$$PI(x) = P(f(x) \geq f(x^+) + \xi \qquad 8$$

$$PI(x) = \Phi\left(\frac{\mu(x) - f(x^+) - \xi}{\sigma(x)}\right) \qquad 9$$

Where Φ is the normal cumulative distribution function, $f(x^+)$ is the incumbent maximum value and ξ is a "trade-off parameter" intended to reduce selection of points yielding larger gains, but less certainty [42].

Expected Improvement is described mathematically as the following [42]:

$$EI(x) = \mathbb{E}[\max((f(x^*) - f(x^+), 0)] \quad 10$$

Where x^* represents the candidate parameters, and x^+ represents the current "best guess" among the previously evaluated parameters. The advantage of this function is that unlike the original objective function, there is a closed form solution given by equations 11 and 12.

$$EI(x) = \begin{cases} (\mu(x) - f(x^+) - \xi)\Phi(z) + \sigma(x)\phi(Z) & if\ \sigma(x) \geq 0 \\ 0 & if\ \sigma(x) = 0 \end{cases} \quad 11$$

$$Z = \frac{\mu(x) - f(x^+) - \xi}{\sigma(x)} \quad 12$$

In equations 11 and 12, Φ and ϕ represent the cumulative distribution function (CDF) and probability density function (PDF) respectively of a unit Gaussian distribution (zero mean and variance equal to 1). Note that the trade-off parameter ξ is included to maintain a balance of "exploration" and "exploitation". When *exploring*, regions where the surrogate variance is large is desirable. When *exploiting*, regions where the surrogate mean is high is desirable. Thus, the Expected Improvement acquisition function considers the probability and magnitude of improvement that a sample may provide and thus maintains a balance between global search and local optimization

(exploration/exploitation) [42]. According to the literature reviewed as part of the research being described in this dissertation, Expected Improvement is by far the most prevalent acquisition function used to establish the "next best guess".

To implement Bayesian Optimization with Gaussian Processes for power system state estimation, it is necessary to establish the following requirements [42].

- **Minimization - Maximization Transformation**: The state estimation problem is framed as the minimization of the RMSE of the estimated states while common implementation of the Bayesian Optimization algorithm is framed as the maximization of an objective function. Thus a transformation to a maximization problem can be made by noting that the maximization of a real-valued function $x^* = argmax_x f(x)$ can be considered as a minimization problem through a transformed function $y(x) = -f(x)$. Thus, while optimization algorithms are often implemented as a maximization problem, the transformation is straightforward.
- **Continuity of the Objective Function**: There will be an assumption of *Lipschitz-Continuity*. Thus a constant C exists, although not generally known such that:

$$||f(x_1) - f(x_2)|| \leq C||x_1 - x_2|| \quad (13)$$

  for $x_1, x_2 \in A$ (compact set of observations) and C intuitively represents how fast a function may change.

- **Global Optimization**: For local maximization problems it is sufficient to find an x^* such that:

$$f(x^*) \geq f(x), \forall x \; s.t. \; ||x^* - x|| < \varepsilon \quad (14)$$

For *-f(x)* that is convex, a local maximum will also be considered a global maximum. In the research being discussed in this dissertation, there will be no assumption that *-f(x)* is convex. Thus, the advantage of the optimization technique being considered is that it will avoid being confined to regions of local maxima (minima).

- **Black Box Optimization:** There will be no assumption that the objective function has known derivatives. Thus, the MLP model itself will be treated as a *black box* on which bounds will be made on the hyperparameters on which it is being optimized over. In optimization theory, this assumption is stated as an assumption that the bounds are all *axis-aligned* such that the search space is *hyperrectangle and of dimension d.*
- **Application of an Acquisition Function:** This function, with a closed form solution, will be considered more tractable than the objective function being optimized however it too will not be assumed to be convex.

The implementation of Bayesian Optimization with Gaussian Processes was done with the *BayesOpt* Python library [28]. This library is built upon the algorithm presented in Figure 10 and the mathematical theory presented in equations 3 – 12. There are two primary functions. The functions are *BayesianOptimization( )* and *maximize( )*. *BayesianOptimization* has the following general structure:

```
BayesianOptimization(
  f=black_box_function,
  pbounds=pbounds,
  random_state=1,
)
```

Note in the *BayesianOptimization* function that there are three essential parameters. Parameter *"f"* is the objective function to be optimized, *"pbounds"* is the range of hyperparameters to be searched over and *"random_state"* is set with a seed to establish repeatability. The *maximize* function has the following general structure:

```
maximize(
    init_points,
    n_iter
)
```

The parameter *"init_points"* establishes how many initial samples to search over before executing the Bayesian Optimization algorithm. The minimum number for this parameter is 2. The parameter *"n_iter"* is the number of iterations to execute the Bayesian Optimization algorithm search process.

To illustrate the Bayesian Optimization with Gaussian Processes, a relatively simple 1-dimensional function is considered that has an easily identifiable global maximum and minimum. The function also includes local maxima and local minima to illustrate the ability of the optimization technique to avoid being confined to points that are not globally maximum.

Thus, let the test objective function be defined as follows:

$$f(x) = \sin(x)\, e^{\frac{-(x-10)^2}{10}} + 2\cos(x) e^{\frac{-(x-2)^2}{10}} \qquad 15$$

Figure 11 is a plot of the test objective function. It should be noted, that in general Bayesian Optimization is applied to objective functions that are not easily expressed analytically.

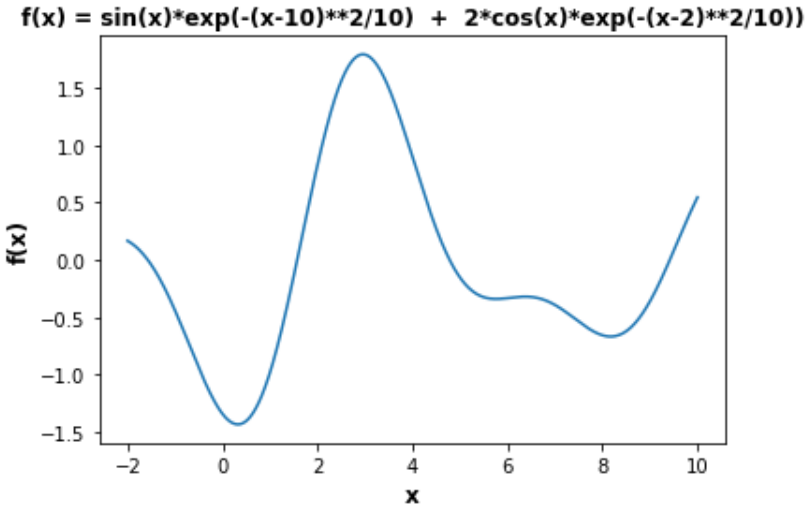


Figure 11 - Plot of Test Objective Function

Code Listing 2 – Visualization of Bayesian Optimization with Gaussian Processes

```
000 # Bayesian_Optimization_Visualization.py
001 from bayes_opt import BayesianOptimization
002 from bayes_opt import UtilityFunction
003 import numpy as np
004
005 import matplotlib.pyplot as plt
006 from matplotlib import pyplot
007 from matplotlib import gridspec
008 #%matplotlib inline
009
010 def target(x):
011     #return -np.sin(x)*np.exp(-x**2)
012     return (-np.sin(x)*np.exp(-(x-10)**2/10) - 2*np.cos(x)*np.exp(-
(x-2)**2/10))
013 x = np.linspace(-2, 10, 10000).reshape(-1, 1)
014 y = target(x)
015
016 pyplot.xlabel('x',fontweight = 'bold', fontsize = 12.0,
color='black')
017 pyplot.ylabel('f(x)', fontweight = 'bold', fontsize = 12.0,
color='black')
018 pyplot.title('f(x) = sin(x)*exp(-(x-10)**2/10)  +  2*cos(x)*exp(-
(x-2)**2/10))', fontweight = 'bold', fontsize = 12.0,)
019
020 plt.plot(x, y);
021
022 optimizer = BayesianOptimization(target, {'x': (-2, 10)},
random_state=27)
023 optimizer.maximize(init_points=2, n_iter=0, kappa=5)
024
```

```
025 def posterior(optimizer, x_obs, y_obs, grid):
026     optimizer._gp.fit(x_obs, y_obs)
027 
028     mu, sigma = optimizer._gp.predict(grid, return_std=True)
029     return mu, sigma
030 
031 
032 def plot_gp(optimizer, x, y):
033     fig = plt.figure(figsize=(16, 10))
034     steps = len(optimizer.space)
035     fig.suptitle(
036         'Step {}: Gaussian Process and Acquisition
Function'.format(steps),
037         fontdict={'size':30}
038     )
039 
040     gs = gridspec.GridSpec(2, 1, height_ratios=[3, 1])
041     axis = plt.subplot(gs[0])
042     acq = plt.subplot(gs[1])
043 
044     x_obs = np.array([[res["params"]["x"]] for res in
optimizer.res])
045     y_obs = np.array([res["target"] for res in optimizer.res])
046 
047     mu, sigma = posterior(optimizer, x_obs, y_obs, x)
048     axis.plot(x, y, linewidth=3, label='Objective Function')
049     axis.plot(x_obs.flatten(), y_obs, 'D', markersize=8,
label=u'Sample', color='r')
050     axis.plot(x, mu, '--', color='k', label='Prediction')
051 
052     axis.fill(np.concatenate([x, x[::-1]]),
053               np.concatenate([mu - 1.9600 * sigma, (mu + 1.9600 *
sigma)[::-1]]),
054         alpha=.6, fc='c', ec='None', label='Confidence Interval
(95%)')
055 
056     axis.set_xlim((-2, 10))
057     axis.set_ylim((None, None))
058     axis.set_ylabel('Objective Function', fontdict={'size':20})
059     axis.set_xlabel('x', fontdict={'size':20})
060 
061     utility_function = UtilityFunction(kind="ei", kappa=5, xi=0)
062     utility = utility_function.utility(x, optimizer._gp, 0)
063     acq.plot(x, utility, label='Acquisition Function',
color='green')
064     acq.plot(x[np.argmax(utility)], np.max(utility), '*',
markersize=15,
065              label=u'Next Sample', markerfacecolor='yellow',
markeredgecolor='k', markeredgewidth=1)
066 
067     acq.set_xlim((-2, 10))
068     acq.set_ylim((0, np.max(utility) + 0.5))
069     acq.set_ylabel('Acqusition Function', fontdict={'size':20})
070     acq.set_xlabel('x', fontdict={'size':20})
071 
072     axis.legend(loc=2, bbox_to_anchor=(1.01, 1), borderaxespad=0.)
073     acq.legend(loc=2, bbox_to_anchor=(1.01, 1), borderaxespad=0.)
```

```
074
075     print()
076
077     print('Next Best Guess: x = %.6f , f(x) = %.6f' %
(x[np.argmax(utility)], np.max(utility)))
078
079 plot_gp(optimizer, x, y)
080
081 print(' ')
082 print('After one step of GP (and two random points):')
083 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
084 plot_gp(optimizer, x, y)
085
086 print(' ')
087 print('After two steps of GP (and two random points)')
088 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
089 plot_gp(optimizer, x, y)
090
091 print(' ')
092 print('After three steps of GP (and two random points)')
093 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
094 plot_gp(optimizer, x, y)
095
096 print(' ')
097 print('After four steps of GP (and two random points)')
098 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
099 plot_gp(optimizer, x, y)
100
101 print(' ')
102 print('After five steps of GP (and two random points)')
103 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
104 plot_gp(optimizer, x, y)
105
106 print(' ')
107 print('After six steps of GP (and two random points)')
108 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
109 plot_gp(optimizer, x, y)
110
111 print(' ')
112 print('After seven steps of GP (and two random points)')
113 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
114 plot_gp(optimizer, x, y)
115
116 print(' ')
117 print('After eight steps of GP (and two random points)')
118 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
119 plot_gp(optimizer, x, y)
120
121 print(' ')
122 print('After nine steps of GP (and two random points)')
123 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
124 plot_gp(optimizer, x, y)
125
126 print(' ')
127 print('After ten steps of GP (and two random points)')
128 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
129 plot_gp(optimizer, x, y)
```

```
130
131 print(' ')
132 print('After eleven steps of GP (and two random points)')
133 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
134 plot_gp(optimizer, x, y)
135
136 print(' ')
137 print('After twelve steps of GP (and two random points)')
138 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
139 plot_gp(optimizer, x, y)
140
141 print(' ')
142 print('After thirteen steps of GP (and two random points)')
143 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
144 plot_gp(optimizer, x, y)
145
146 print(' ')
147 print('After fourteen steps of GP (and two random points)')
148 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
149 plot_gp(optimizer, x, y)
150
151 print(' ')
152 print('After fifteen steps of GP (and two random points)')
153 optimizer.maximize(init_points=0, n_iter=1, kappa=5)
154 plot_gp(optimizer, x, y)
```

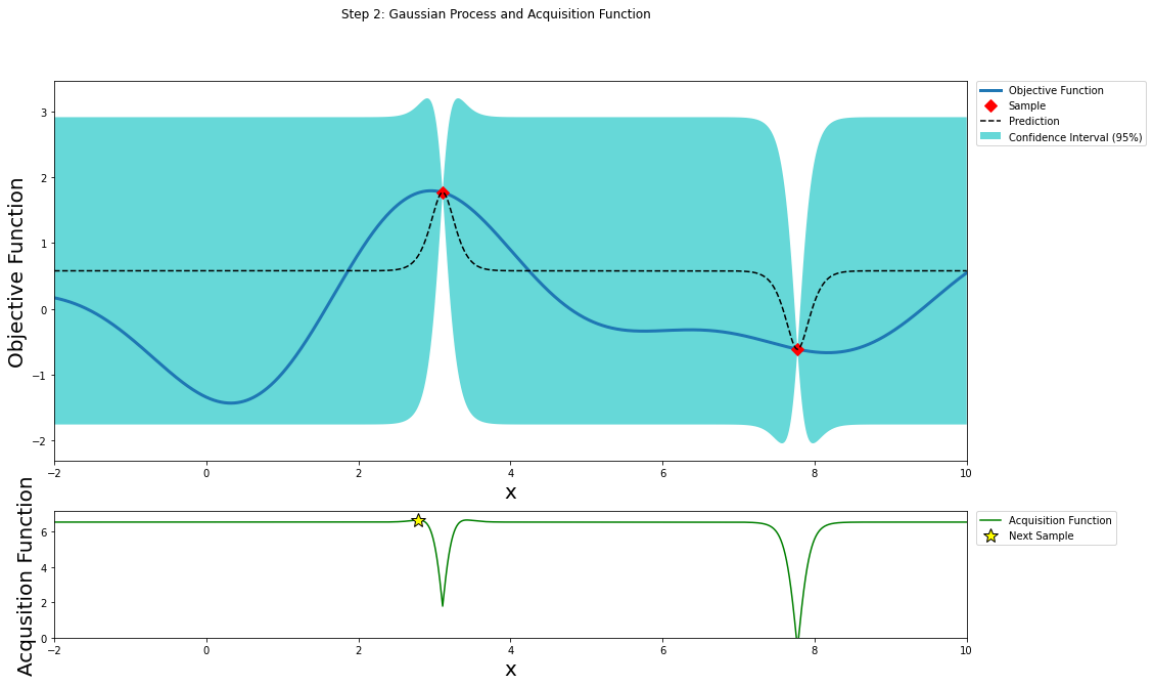
Table 14 presents a summary of Code Listing 2 for visualization of Bayesian Optimization with Gaussian Processes.

Table 14 - Summary of Code Listing 2 for Visualization of Bayesian Optimization with Gaussian Processes

Line #:	Description:
001 - 002	Import Python libraries for Bayesian Optimization
010 - 012	Define a "target" objective function
022	Define an "optimizer" and pass the objective function to the BayesianOptimization function
023	Initiate the Bayesian Optimization algorithm via the "optimizer.maximize" function. Two initial random points are searched. Two points are the minimum number of random points to initiate the optimization process
061	Define a "UtilityFunction" that takes as an argument, the type of acquisition function to use (i.e., "ucb", "pi", "ei"). Acquisition function "ei" represents the Expected Improvement function presented in equations 8 - 10
077	Print "Next Best Guess" result from the Bayesian Optimization with Gaussian Process
083, 088, 093, 098, 103, 108, 113, 118, 123, 128, 133, 138, 143, 148, 153	Initiates steps of Bayesian Optimization with Gaussian Processes shown in Figures 12-18

The following steps illustrate the process. Some of the steps are not shown to maintain brevity. Also note that "Step 1 of Optimization" is the initial random sample of the first point and is not shown.

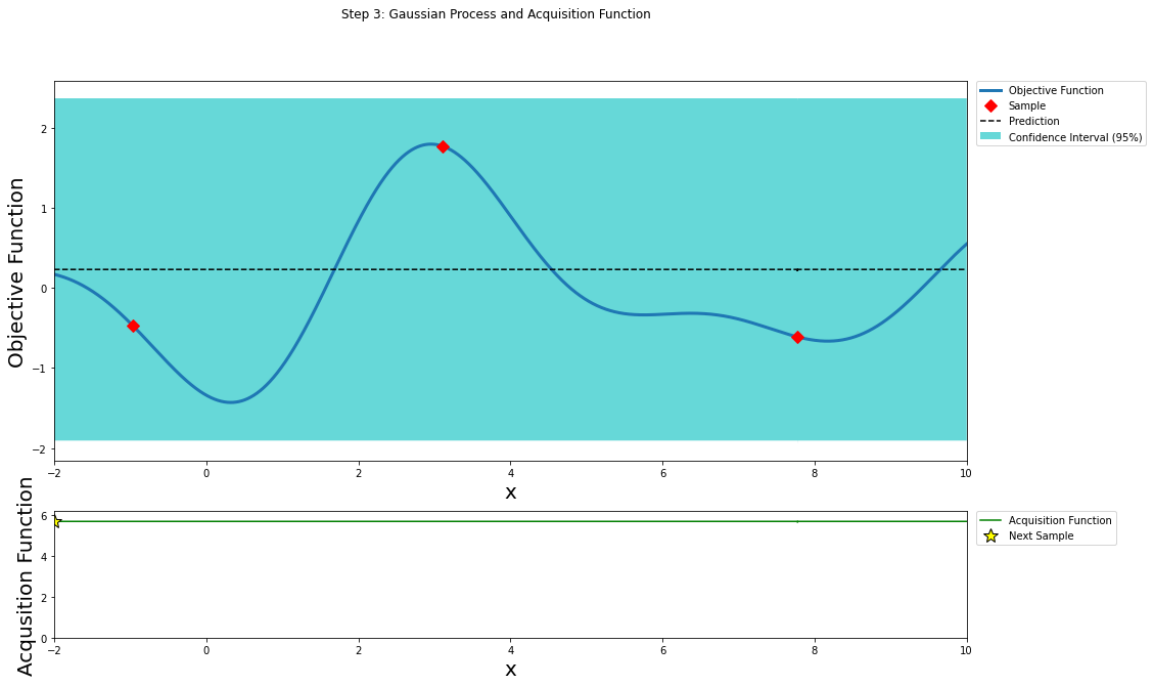
In the upper portion of Figure 12 the objective function, two samples, the prediction, and the confidence interval (95%) are notated by the legend. Note that the "prediction" is established by the mean function of the confidence interval. The two samples shown were selected randomly from regions where the variance is large. This is the "exploration" phase of Bayesian Optimization. The lower portion of the figure shows the "Acquisition Function" along with the next sample selected by the Bayesian optimizer. In this case, the acquisition function is established by the Expected Improvement function given in equations 10, 11 and 12 presented earlier in this section. Note that the "Next Best Guess" will be selected where the mean of the acquisition function is high.



iter	target	x
1	1.767	3.109
2	-0.6133	7.775

Next Best Guess: x = 2.789679 , f(x) = 6.646690

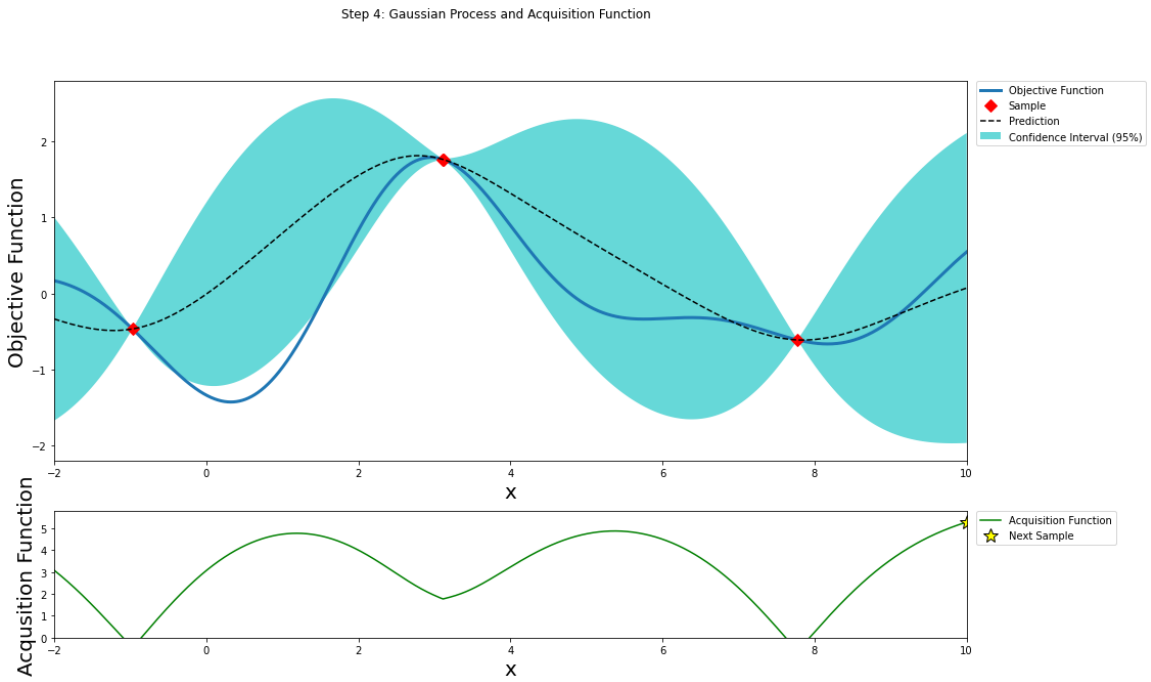
Figure 12 - Step 2 of Bayesian Optimization with Gaussian Processes




```
After one step of GP (and two random points):
|   iter    |  target   |     x     |
-------------------------------------
|  3        | -0.4685   | -0.9694   ||
=====================================

Next Best Guess: x = -2.000000 , f(x) = 5.677356
```

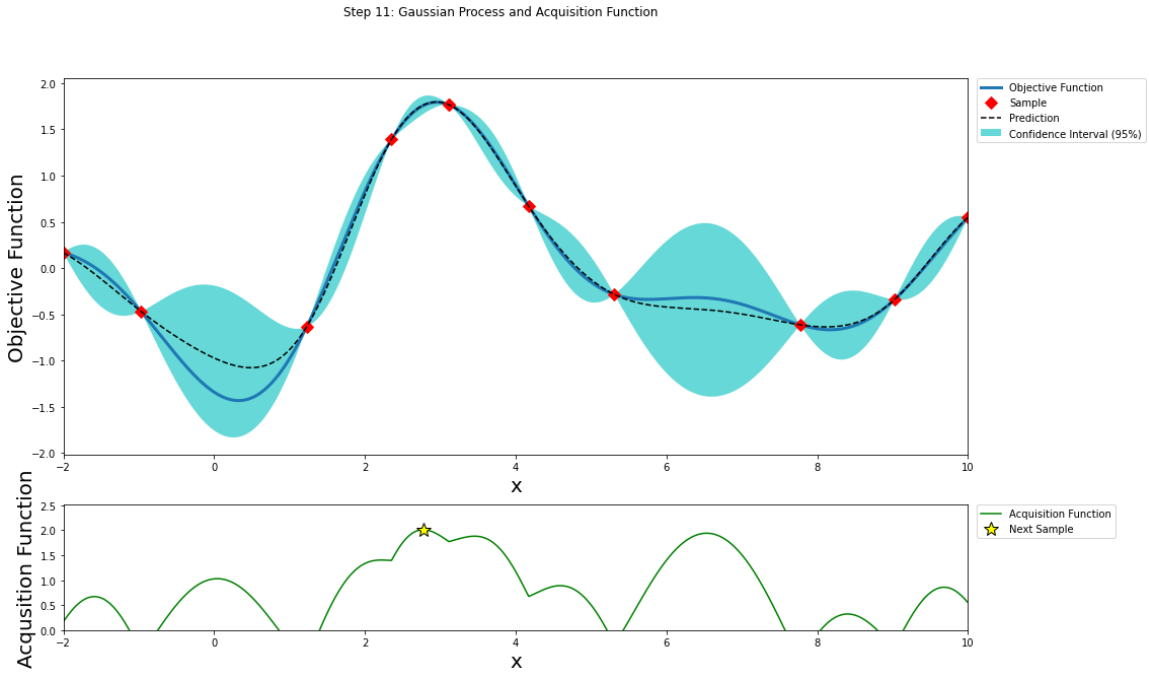
Figure 13 - Step 3 of Bayesian Optimization with Gaussian Processes




```
After two steps of GP (and two random points)
|   iter    |  target   |     x     |
-------------------------------------
|  4        |  1.765    |  3.115    |
=====================================

Next Best Guess: x = 10.000000 , f(x) = 5.282425
```

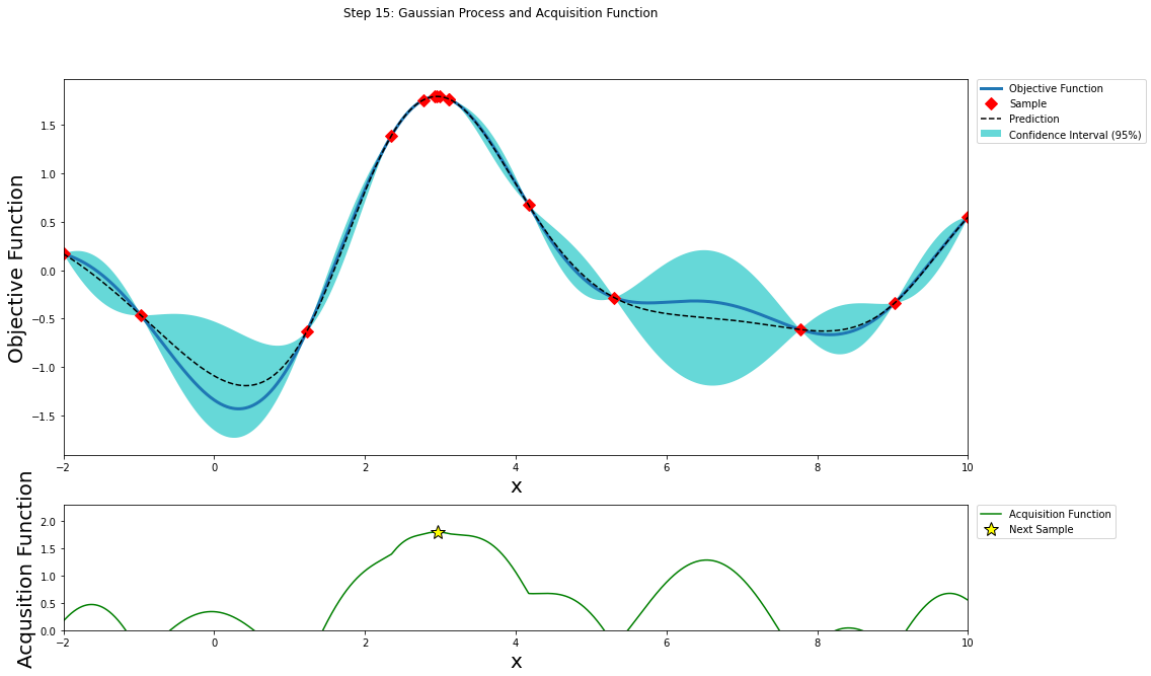
Figure 14 - Step 4 of Bayesian Optimization with Gaussian Processes




```
After nine steps of GP (and two random points)
|   iter    |  target   |     x     |
-------------------------------------
|  11       |  1.39     |  2.353    |
=====================================

Next Best Guess: x = 2.774077 , f(x) = 2.007187
```

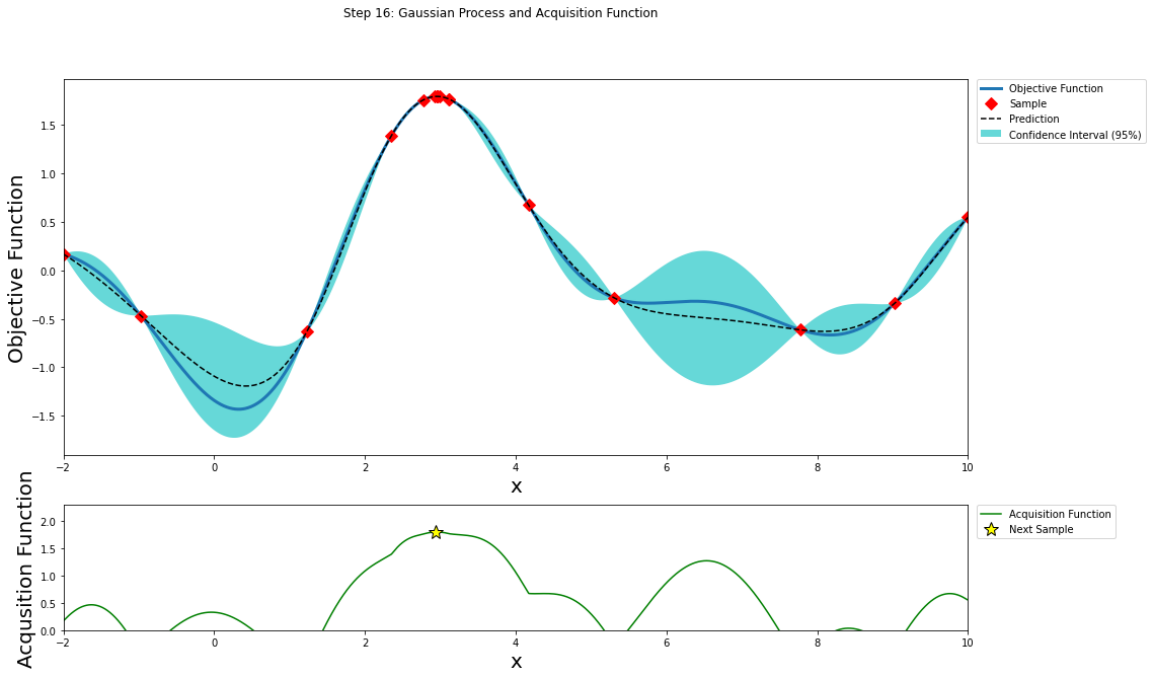
Figure 15 - Step 11 of Bayesian Optimization with Gaussian Processes




```
After thirteen steps of GP (and two random points)
|   iter    |  target   |     x     |
-------------------------------------
|  15       |  1.791    |  2.923    |
=====================================

Next Best Guess: x = 2.962496 , f(x) = 1.795703
```

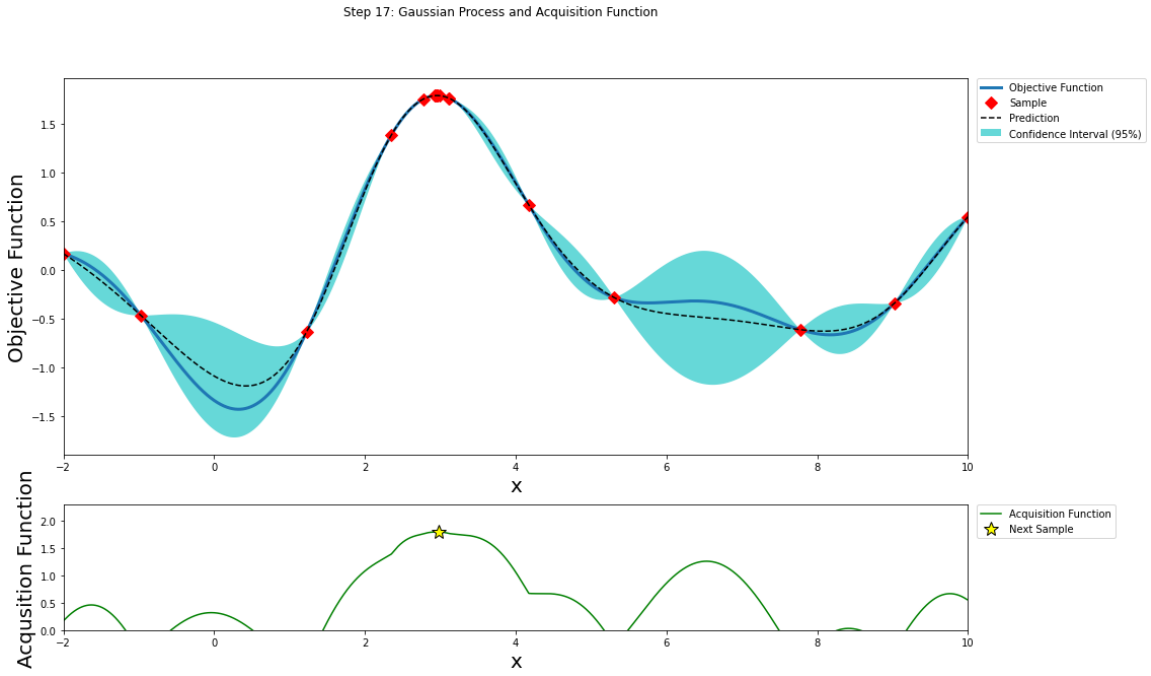
Figure 16 - Step 15 of Bayesian Optimization with Gaussian Processes




```
After fourteen steps of GP (and two random points)
|   iter    |  target   |     x     |
-------------------------------------
|  16       |  1.793    |  2.963    |
=====================================

Next Best Guess: x = 2.943294 , f(x) = 1.795271
```

Figure 17 - Step 16 of Bayesian Optimization with Gaussian Processes




```
After fifteen steps of GP (and two random points)
|   iter    |  target   |     x     |
-------------------------------------
|  17       |  1.793    |  2.944    |
=====================================

Next Best Guess: x = 2.973297 , f(x) = 1.795055
```

Figure 18 - Step 17 of Bayesian Optimization with Gaussian Processes

Note that additional steps of the process will continue to converge on the "global maximum". Also note that in regions where there is a greater likelihood of a maximum, there are a higher concentration of points as the "Next Best Guess" will be chosen to remain in these regions. Likewise, regions that did not contain values of sufficient probability of "expected improvement" were not sampled again. Thus, Bayesian Optimization with Gaussian Processes will greatly reduce the probability of selecting values that will not produce the maximum of the original objective function.

3.9.2 Implementation of Bayesian Optimization with Gaussian Processes on MLPs for DSSE

The authors in [51] discuss practical Bayesian Optimization of machine learning algorithms and make a distinction between optimization of "learning parameters" such as learning rate and batch size and optimization of "model parameters" such as the number of layers and number of neurons per hidden layer. The research on which this dissertation is based will consider a combination of both types to be referred to as "hyperparameters".

Section 3.7 presented the performance of MLPs without hyperparameter optimization for DSSE. The model architectures for each trial shown in Table 10 involved selecting the number of hidden layers, number of neurons per hidden layer, and other hyperparameters in an "ad-hoc" fashion. The data show that by holding the following hyperparameters constant while randomly adjusting the number of layers and neurons did not result in significant improvement in training, testing, and validation RMSE with more complex networks.

Table 15 presents the unoptimized hyperparameters held constant for Trials 1-11 in Table 10.

Table 15 - Unoptimized MLP Hyperparameter Settings

MLP Hyperparameters	Settings
Activation Function per Layer	Rectified Linear Unit (ReLU)
Batch Size	10
Epochs	200
Learning Rate	0.001
Loss Function	Mean Squared Error
Input Layer Neurons	56
Output Layer Neurons	56
Optimizer	Stochastic Gradient Descent (SGD)

Note the *Input Layer Neurons* and *Output Layer Neurons* are established by the number of features in the complex power (input) and complex voltage (output) gathered from the fixed number of monitors placed in the power distribution test system. In general the number of neurons in each of these layers is independent and are not required to be equal. Bayesian Optimization was performed on some of the previous trials in addition to new random configurations. The new configurations involved consideration of models with 2 hidden layers given that the original trials did not contain this configuration. It was decided that the following hyperparameters would be held constant given their prevailing use in modern machine learning:

- *Activation Function per Layer: ReLU*
- *Loss Function: Mean Squared Error*
- *Optimizer: Adam optimizer with adjustable learning rate*

The following hyperparameters were configured to have the following ranges:

- *Hidden Layer Neurons: 1 - 100*
- *Learning Rate: 0.0001 - 1.0*
- *Batch Size: 1 - 100*
- *Epochs: 1 - 300*

Bayesian Optimization with Gaussian Processes was used to optimize the MLP network corresponding to "Trial 1" in Table 10. The optimized MLP is labeled as "Trial 1Opt". The results of optimization for this trial and remaining trials for other MLP architectures considered are shown in Table 17. For brevity, the steps below show the critical steps required to enable Bayesian Optimization with Gaussian Processes for "Trial 1Opt" only:

Code Listing 3 - MLP Model Optimized with Bayesian Optimization with Gaussian Processes:

```
000 # MLP_Trial_01_Optimized.py
001
002 import time
003 start_time = time.time()
004
005 # !pip install keras.optimizers
006 # !pip install bayesian-optimization
007
008 # from google.colab import drive
009 # drive.mount('/content/drive')
010
011 import os
012 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
013
014 import warnings
015 warnings.filterwarnings('ignore')
016
017 import numpy as np
018 import numpy
019 import pandas as pd
020 from pandas import read_csv
021 import math
022 from math import floor
023 import matplotlib.pyplot as plt
024 import seaborn as sns
025 import pickle
026 from keras.models import Sequential
027 from keras.layers import Dense, BatchNormalization, Dropout
```

```
028 from keras.callbacks import EarlyStopping, ModelCheckpoint
029 from keras.wrappers.scikit_learn import KerasRegressor
030 from keras.layers import LeakyReLU
031 from sklearn.model_selection import train_test_split
032 from sklearn.metrics import mean_squared_error
033 from sklearn.preprocessing import MinMaxScaler
034 from sklearn.model_selection import train_test_split
035 from sklearn.model_selection import cross_val_score
036 from sklearn.metrics import make_scorer, accuracy_score
037 from sklearn.model_selection import StratifiedKFold
038 from sklearn.model_selection import KFold
039 from tensorflow.keras.optimizers import Adam
040
041 # Import BayesianOptimization Library
042 from bayes_opt import BayesianOptimization
043
044 pd.set_option("display.max_columns", None)
045 LeakyReLU = LeakyReLU(alpha=0.1)
046
047 # Make scorer accuracy
048 score_acc = make_scorer('neg_mean_squared_error')
049
050 # Data Normalization
051 scaler_power_ercot = MinMaxScaler(feature_range=(-1,1),copy=True)
052 scaler_voltage_ercot = MinMaxScaler(feature_range=(-1,1),copy=True)
053
054 scaler_power_coast = MinMaxScaler(feature_range=(-1,1),copy=True)
055 scaler_voltage_coast = MinMaxScaler(feature_range=(-1,1),copy=True)
056
057 # Load Datasets
058
059 # EROCT Data used to Train and Test MLP
060 #input_power_ercot_dataframe =
read_csv('/content/input_power_ercot.csv', header=None)
061 input_power_ercot_dataframe =
read_csv('C:/Python/input_power_ercot.csv', header=None)
062 input_power_ercot = input_power_ercot_dataframe.values
063 input_power_ercot_normalized =
scaler_power_ercot.fit_transform(input_power_ercot)
064
065 #input_power_ercot_normalized = input_power_ercot[0:24]
066
067 #output_voltage_ercot_dataframe =
read_csv('/content/output_voltage_ercot.csv', header=None)
068 output_voltage_ercot_dataframe =
read_csv('C:/Python/output_voltage_ercot.csv', header=None)
069 output_voltage_ercot = output_voltage_ercot_dataframe.values
070 output_voltage_ercot_normalized =
scaler_voltage_ercot.fit_transform(output_voltage_ercot)
071
072 #output_voltage_ercot_normalized = output_voltage_ercot[0:24]
073
074 # Coast Data used to Validate MLP
075 #input_power_coast_dataframe =
read_csv('/content/input_power_coast.csv', header=None)
076 input_power_coast_dataframe =
read_csv('C:/Python/input_power_coast.csv', header=None)
```

```
077 input_power_coast = input_power_coast_dataframe.values
078 input_power_coast_normalized =
scaler_power_coast.fit_transform(input_power_coast)
079
080 #output_voltage_coast_dataframe =
read_csv('/content/output_voltage_coast.csv', header=None)
081 output_voltage_coast_dataframe =
read_csv('C:/Python/output_voltage_coast.csv', header=None)
082 output_voltage_coast = output_voltage_coast_dataframe.values
083 output_voltage_coast_normalized =
scaler_voltage_coast.fit_transform(output_voltage_coast)
084
085 # Split Data Into 70% for Train and 30% for Test
086 power_train_normalized, power_test_normalized,
voltage_train_normalized, voltage_test_normalized =
train_test_split(input_power_ercot_normalized,
output_voltage_ercot_normalized, test_size=0.30)
087
088 # Set seed
089 from numpy.random import seed
090 seed(123)
091
092 import os
093 os.environ['PYTHONHASHSEED']=str(123)
094
095 import random
096 random.seed(123)
097
098 import tensorflow as tf
099 tf.random.set_seed(123)
100
101 # Define an Objective Function with Hyperparameters as Arguments
102 def nn_bo(neurons, learning_rate, batch_size, epochs):
103     neurons = round(neurons)
104     batch_size = round(batch_size)
105     epochs = round(epochs)
106
107     def nn_fun():
108         opt = Adam(lr = learning_rate)
109
110         nn = Sequential()
111         nn.add(Dense(neurons, input_dim=56, activation='relu'))
112         nn.add(Dense(56))
113         nn.compile(loss='mean_squared_error', optimizer=opt,
metrics=['mse'])
114         return nn
115
116     es = EarlyStopping(monitor='mean_squared_error', mode='max',
verbose=0, patience=20)
117     nn = KerasRegressor(build_fn=nn_fun, epochs=epochs,
batch_size=batch_size,
118                         verbose=0)
119     kfold = KFold(n_splits=5, shuffle=True, random_state=123)
120     score = cross_val_score(nn, power_train_normalized,
voltage_train_normalized, cv=kfold).mean()
121
122     return score
```

```
123
124 # Define the Hyperparameter Space to Optimize Over
125 params_nn ={
126     'neurons': (1, 1000),
127     'learning_rate':(0.0001, 1),
128     'batch_size':(1, 10),
129     'epochs':(1,10)
130 }
131
132 # Run Bayesian Optimization with Gaussian Processes
133 nn_bo = BayesianOptimization(nn_bo, params_nn, random_state=111)
134
135 nn_bo.maximize(init_points=2, n_iter=2)
136
137 # Save and Print Hyperparameters Selected via Bayesian Optimization
138 params_nn_ = nn_bo.max['params']
139
140 params_nn_['neurons'] = round(params_nn_['neurons'])
141 params_nn_['learning_rate'] = round(params_nn_['learning_rate'], 2)
142 params_nn_['batch_size'] = round(params_nn_['batch_size'])
143 params_nn_['epochs'] = round(params_nn_['epochs'])
144
145 params_nn_
146
147 print(params_nn_)
148
149 # Fit the  Neural Network with Optimized Hyperparameters
150 def nn_fun():
151     opt = Adam(lr = params_nn_['learning_rate'])
152     nn = Sequential()
153     nn.add(Dense(params_nn_['neurons'], input_dim=56,
activation='relu'))
154     nn.add(Dense(56))
155     nn.compile(loss='mean_squared_error', optimizer=opt,
metrics=['mse'])
156     return nn
157
158 es = EarlyStopping(monitor='mean_squared_error', mode='max',
verbose=0, patience=20)
159 nn = KerasRegressor(build_fn=nn_fun, epochs=params_nn_['epochs'],
batch_size=params_nn_['batch_size'],
160                             verbose=0)
161 kfold = KFold(n_splits=5, shuffle=True, random_state=123)
162 nn.fit(power_train_normalized, voltage_train_normalized,
validation_data=(power_test_normalized,
163        voltage_test_normalized), verbose=0)
164
165 # Evaluate the MLP Model with Optimized Hyperparameters
166 trainScore = cross_val_score(nn, power_train_normalized,
voltage_train_normalized, cv=kfold).mean()
167 print('ERCOT Train Score: %.6f MSE (%.6f RMSE)' % (abs(trainScore),
math.sqrt(abs(trainScore))))
168
169 testScore = cross_val_score(nn, power_test_normalized,
voltage_test_normalized, cv=kfold).mean()
170 print('ERCOT Test Score: %.6f MSE (%.6f RMSE)' % (abs(testScore),
math.sqrt(abs(testScore))))
```

```
171
172 print(' ')
173 # Make Prediction with the Optimimized MLP on Validation Data
174 coastScore =  cross_val_score(nn, input_power_coast_normalized,
output_voltage_coast_normalized,
175                               cv=kfold).mean()
176
177 print('Coast Validation Score: %.6f MSE (%.6f RMSE)' %
(abs(coastScore),
178
math.sqrt(abs(coastScore))))
179 print(' ')
180
181 print("--- %s seconds ---" % (time.time() - start_time))
```

Execution of the code listing presented above produced the following results:

```
|   iter    |  target   | batch_... |  epochs   | learni... |  neurons  |
-------------------------------------------------------------------------
|  1        | -1.032    |  6.51     |  2.522    |  0.4417   |  769.5    |
|  2        | -0.03407  |  3.658    |  2.342    |  0.03225  |  420.8    |
|  3        | -0.09807  |  3.074    |  3.462    |  0.1861   |  420.5    |
|  4        | -0.02204  |  9.955    |  2.796    |  0.05658  |  434.1    |
=========================================================================
{'batch_size': 10, 'epochs': 3, 'learning_rate': 0.06, 'neurons': 434}
ERCOT Train Score: 0.021157 MSE (0.145455 RMSE)
ERCOT Test Score: 0.023658 MSE (0.153813 RMSE)

Coast Validation Score: 0.019329 MSE (0.139029 RMSE)

--- 92.21797847747803 seconds ---
```

Table 16 presents a summary of Code Listing 3 for an MLP model optimized with Bayesian Optimization with Gaussian Processes.

Table 16 - Summary of Code Listing 3 for MLP Model Optimized with Bayesian Optimization with Gaussian Processes

Line #:	Description:
017 - 042	Import essential Python libraries for implementation of MLP model for DSSE and Bayesian Optimization with Gaussian Process for hyperparameter selection
050 - 055	Define "scalar" objects to perform data normalization on source data to aid in training performance.
059 - 070	Import ERCOT data used to train and test the MLP. Perform data normalization.
074 - 083	Import COAST data used to validate MLP. Perform normalization.
085 - 086	Split input data into 70% for training and 30% for testing.
101 - 122	Define objective function with hyperparameters as arguments. Hyperparameters to be optimized over are the # of neurons in the hidden layer, learning rate for the ADAM optimizer, batch size and # of epochs for training.
107 - 114	Define a parameterized MLP model that is "wrapped" within the objective function.
124 - 129	Define the hyperparameter space to optimize over. Establish ranges for each hyperparameters.
133	Pass the objective function and parameter space to the BayesianOptimization() function.
135	Initialize Bayesian Optimization with Gaussian Process with the maximize() function. Two initial points will be sampled and the Bayesian Optimization algorithm will perform 2 iterations.
137 - 147	Save and print hyperparameters selected via Bayesian Optimization.
149 - 156	Fit the MLP with the optimized hyperparameters.
165 - 170	Evaluate the optimized MLP on training and testing data in terms of RMSE.

173 - 178	Perform DSSE with optimized MLP on COAST validation data and report RMSE.

Table 17 is an extension of Table 10 to include trials (models) with and without hyperparameter optimization to compare the performance in training, testing, and validation scores (RMSE) as well as execution time on a system with the following specifications:

Processor: *Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz 2.81 GHz*
Installed: *RAM 16.0 GB (15.9 GB usable)*
System type: *64-bit operating system, x64-based processor*

Table 17 – Performance Results for MLP Models with Hyperparameter Optimization

Trial	Input Layer	Hidden Layers	Output Layer	Train RMSE (70%)	Test RMSE (30%)	COAST Act. vs. Est. RMSE	Execution Time (Seconds)
1	56 Neurons	1 Layer 56 Neurons	56 Neurons	0.140927	0.142162	0.323075	135.453
1Opt	56 Neurons	1 Layer 72 Neurons	56 Neurons	0.128002	0.141306	0.118630	969.183
2	56 Neurons	1 Layer 112 Neurons	56 Neurons	0.140486	0.136711	0.323293	138.883
2Opt	56 Neurons	1 Layer 72 Neurons	56 Neurons	0.128002	0.141306	0.118630	969.183
3	56 Neurons	1 Layer 224 Neurons	56 Neurons	0.137222	0.137328	0.322124	143.491
3Opt	56 Neurons	1 Layer 72 Neurons	56 Neurons	0.128002	0.141306	0.118630	969.183
3a	56 Neurons	2 Layers 56 Neurons	56 Neurons	0.133488	0.132823	0.329049	140.123

3aOpt	56 Neurons	2 Layers 39 Neurons	56 Neurons	0.072172	0.079611	0.063347	2079.529
3b	56 Neurons	2 Layers 224 Neurons	56 Neurons	0.128535	0.128073	0.335831	176.820
3bOpt	56 Neurons	2 Layers 39 Neurons	56 Neurons	0.072172	0.079611	0.063347	2079.529
3c	56 Neurons	3 Layers 56 Neurons	56 Neurons	0.131211	0.130995	0.336122	142.877
3cOpt	56 Neurons	3 Layers 31 Neurons	56 Neurons	0.077014	0.093673	0.072809	996.437
3d	56 Neurons	3 Layers 224 Neurons	56 Neurons	0.127834	0.127370	0.334619	217.837
3dOpt	56 Neurons	3 Layers 31 Neurons	56 Neurons	0.077014	0.093673	0.072809	996.437
3e	56 Neurons	4 Layers 56 Neurons	56 Neurons	0.130180	0.129626	0.337249	144.387
3eOpt	56 Neurons	4 Layers 91 Neurons	56 Neurons	0.072290	0.092400	0.062920	1168.186
3f	56 Neurons	4 Layers 224 Neurons	56 Neurons	0.125555	0.125256	0.334843	249.738
3fOpt	56 Neurons	4 Layers 91 Neurons	56 Neurons	0.072290	0.092400	0.062920	1168.186
3g	56 Neurons	5 Layers 56 Neurons	56 Neurons	0.130120	0.129770	0.338243	151.583
3gOpt	56 Neurons	5 Layers 26 Neurons	56 Neurons	0.077375	0.105210	0.074014	2090.218

Note that the number of hidden neurons per layer for the trials with the “xxOpt” designation was determined by the Bayesian Optimization with Gaussian Processes described in section 3.9.1.

Note in Table 17 that the column "COAST Act. vs. Est. RMSE" represents the "validation error" for data not previously seen by each MLP model. The Bayesian Optimizer selected hyperparameters that reduced validation error over the "unoptimized" model configuration for each trial. The "Execution Time" in Table 17 confirms the "exploration/exploitation tradeoff" by the Expected Improvement (EI) acquisition function discussed in section 3.9.1. While improvements were made in the reduction of validation error enabled by hyperparameter optimization, the execution time for each trial was increased due to the time required to sample regions of the hyperparameter space and select the "next best guess" for each subsequent step of optimization as illustrated in the example presented in section 3.9.1. Table 18 provides the hyperparameters selected by the Bayesian Optimization process for the trials shown in Table 17.

Table 18 - Hyperparameters Selected via Bayesian Optimization with Gaussian Processes

Trial	Neurons per Hidden Layer	Learning Rate	Epochs	Batch Size
1Opt	72	0.023	29	85
2Opt	72	0.023	29	85
3Opt	72	0.023	29	85
3aOpt	39	0.821	220	93
3bOpt	39	0.821	220	93
3cOpt	31	0.010	109	82
3dOpt	31	0.010	109	82
3eOpt	91	0.321	87	75
3fOpt	91	0.321	87	75
3gOpt	26	0.013	82	33

Table 19 presents a summary of the optimized model configurations MLPs optimized via Bayesian Optimization with Gaussian Processes. The table shows for models with a given number of hidden layers, the configuration established by utilizing the hyperparameters selected by the optimization process.

Table 19 - Optimized Model Configurations

Number of Hidden Layers	Model Configuration
1	Input Layer Neurons: 56 Hidden Layer Neurons: 72 Output Layer Neurons: 56 Activation Function per Layer: Rectified Linear Unit (ReLU) Learning Rate: 0.023 Epochs = 29 Batch Size = 85 Loss Function: Mean Squared Error Optimizer: Adam Optimizer
2	Input Layer Neurons: 56 Hidden Layer Neurons: 39 Output Layer Neurons: 56 Activation Function per Layer: Rectified Linear Unit (ReLU) Learning Rate: 0.821 Epochs = 220 Batch Size = 93 Loss Function: Mean Squared Error Optimizer: Adam Optimizer
3	Input Layer Neurons: 56 Hidden Layer Neurons: 31 Output Layer Neurons: 56 Activation Function per Layer: Rectified Linear Unit (ReLU) Learning Rate: 0.010 Epochs = 109 Batch Size = 82 Loss Function: Mean Squared Error Optimizer: Adam Optimizer
4	Input Layer Neurons: 56 Hidden Layer Neurons: 91 Output Layer Neurons: 56 Activation Function per Layer: Rectified Linear Unit (ReLU) Learning Rate: 0.321 Epochs = 87 Batch Size = 75 Loss Function: Mean Squared Error Optimizer: Adam Optimizer
5	Input Layer Neurons: 56

	Hidden Layer Neurons: 26 Output Layer Neurons: 56 Activation Function per Layer: Rectified Linear Unit (ReLU) Learning Rate: 0.013 Epochs = 82 Batch Size = 33 Loss Function: Mean Squared Error Optimizer: Adam Optimizer

The following observations can be made regarding MLPs considered in this research for DSSE:

- For unoptimized MLPs considered in this current research, there is no notable benefit in utilizing increasingly complex models in terms of number of hidden layers and number of neurons per hidden layer. In some cases, there is even diminishing performance in terms of the validation error RMSE with more complex models.
- For unoptimized MLPs considered in this current research, the RMSE for training and testing is of a similar scale, however this error metric increases significantly when making predictions for validation on data never seen by the network. This is verified in the column "Coast Act. vs. Est. RMSE" in Table 17 . Thus, the unoptimized MLPs considered, exhibit in machine learning terms the condition of “overfitting” in which the MLP learns the training dataset quite well, however, fails to generalize to new data.
- For MLPs optimized by the Bayesian Optimization method considered in this research, a parameter space of $30x10^9$ combinations can be searched to arrive at a combination of hyperparameters to yield a reduction in validation error.
- For MLPs optimized by the Bayesian Optimization method considered in this research, the RMSE for training and testing is of a similar scale to that of unoptimized

MLPs, however this error metric is significantly lower when making predictions on data never seen by the network. This is verified in the column "Coast Act. vs. Est. RMSE" in Table 17. Thus Bayesian Optimization applied to MLPs enabled the resulting models to better generalize to new data.

- Bayesian Optimization seeks a “global” solution however there is a trade-off between exploration and exploitation of subsets of the hyperparameter space.

Possible research opportunities may include the following:

- Investigation of full-state estimation of complex voltages at all buses and nodes given power flow through a subset of lines in a test feeder circuit.
- Investigation of Gaussian Processes with various mean and covariance functions.
- Investigation of various acquisition functions and related parameters illustrated in section 3.9.1. Thus, the configuration of the parameters of the Gaussian Process can be approached as another optimization problem to improve performance.
- Investigation of the exploration/exploitation tradeoff inherent in Bayesian Optimization with Gaussian Process to further improve its performance.

CHAPTER 4. FULL DISTRIBUTION SYSTEM STATE ESTIMATION WITH OPTIMIZED MLP MODELS

Preliminary research on which this dissertation is based involved the placement of power and voltage monitors at specific locations in an IEEE-34 Node Test Distribution System as shown in Figure 8. Power monitors captured complex power flowing through specific nodes and voltage monitors captured complex voltage at the same nodes. This data was then used to train MLPs to predict complex voltage at the same nodes given new complex

power. An improved methodology captures bi-directional power flow on every line and complex voltage at every node to establish training data. The current research investigates the effect of utilizing specific percentages of complex power flow with randomly selected lines to train MLPs to predict complex voltage at all nodes. The effect of Bayesian Optimization with Gaussian Processes is investigated in terms of reducing validation error. The performance of training, testing, and validation is presented in terms of root-mean-squared-error (RMSE) and execution time.

Figure 8 in section 3.3 presents the IEEE test distribution system and measurement points considered in the early stages of the research on which this dissertation is based. The number of power and voltage monitors and locations was selected randomly and the MLP model hyperparameters such as number of hidden layer neurons, learning rate, batch size and number of epochs were selected via an ad hoc approach. Thus, the MLP models were considered "unoptimized".

As noted in section 3.7, a key observation is that MLP models *without hyperparameter optimization* do not offer significant improvement in RMSE when performing regression by utilizing increasingly complicated networks in terms of number of layers and number of neurons per layer.

It was found that improvement in RMSE could be achieved through Bayesian Optimization and that a hyperparameter space of 30×10^9 could be searched at the expense of additional training time over that of unoptimized MLP models.

Additionally, it was found that "optimized" MLP models were able to avoid *over-training* and thus generalize better to data never seen by the networks than models with the same structure without hyperparameter optimization.

The goals of the latest iteration of research described in this chapter are as follows:

1. Establish an improved methodology and data pipeline [52] for training, testing and validation through the integration of a high level programming language (Python) and a distribution system simulator (OpenDSS) [53].
2. Investigate the effect of utilizing specific percentages of complex power flow with randomly selected lines to train MLPs to predict the complex voltage at all nodes of the distribution system.
3. Identify opportunities for improved state estimation given larger power distribution systems.

4.1 Original Workflow and Data Pipeline

Figure 19 shows a functional diagram of the original workflow and data pipeline for the preliminary research on which this dissertation is based.

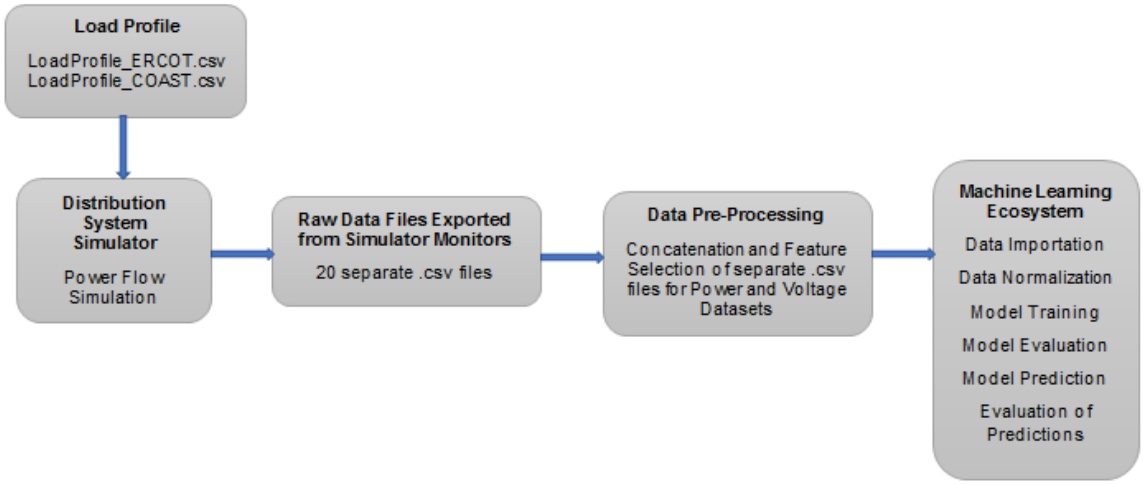


Figure 19 - Original Workflow and Data Pipeline

4.1.1 Load Profile

A load profile provides a means of varying the loads in the distribution system during a power flow simulation. As presented in section 3.4 and 3.5, *LoadProfile_ERCOT.csv* and *LoadProfile_COAST.csv* represent load profiles associated with two regions within the *Electric Reliability Council of Texas* [10]. The choice of load profiles used in the current research is based upon the accessibility of the data and prevalence in power system literature and has no other significance.

4.1.2 Distribution System Simulator

An open source distribution system simulator (OpenDSS) is used for performing power flow simulations to gather training, testing and validation datasets.

Power and voltage monitors are placed at specific locations in the test feeder system. The descriptions, locations and quantities measured were presented previously in section 3.3 of

this dissertation. Power flow simulations were performed based on load profiles to establish the datasets for training, testing and validation.

4.1.3 Raw Data Files Exported from Simulator Monitors

Each OpenDSS monitor provides the capability of exporting raw data files in .csv format. Specifically, for the monitors placed in the test feeder system, there are 10 .csv files representing the complex power flow through each monitor location and 10 .csv files representing the complex voltage at each monitor location.

These datasets are considered "raw" in the sense that they contain additional information not conducive to the machine learning models selected for performing regression.

For the complex power and voltage datasets, each file contains non-numeric column headers describing the numeric data and the corresponding units (i.e. P(kW), Q(kvar), V(volts), Vangle(degrees). Additionally, columns representing the "hour" and "sec" of the power flow simulation were included and deemed unnecessary.

For the complex voltage datasets, current flow per phase was included and deemed unnecessary. It should be noted that for state estimation, current flow could be considered an option for future research.

4.1.4 Data Pre-processing

In order to prepare and reshape the datasets for processing by multilayer-perceptron models considered in this research, it is necessary to establish a single dataset for *supervised learning* comprised of complex power (PQ values) and a single dataset comprised of complex voltage (V_mag and V_angle) from all of the separate .csv files.

This task is performed by a high-level programming language (Python) capable of looping through a file directory to load and concatenate the separate datasets into the required final datasets. Additionally during this process, column headings and extraneous columns are removed.

The decision of which columns to maintain and which to remove is in the parlance of machine learning, referred to as “feature selection” or “feature engineering” [52]. Additional research may consider the effect of selection of different features for use in state estimation. For example, the authors in [54] consider branch-current based state estimation.

4.1.5 Machine Learning Ecosystem

The machine learning ecosystem selected in the previous research includes the following: Python [62], Sklearn [55],Pandas [56], TensorFlow [57], Keras [58].

Once preprocessing is complete, the MLP models could be trained, tested and validated for performing state estimation for the limited locations within the test feeder system corresponding to the location of the monitors.

4.2 Improved Workflow and Data Pipeline

Figure 20 shows a functional diagram of an improved workflow and data pipeline.

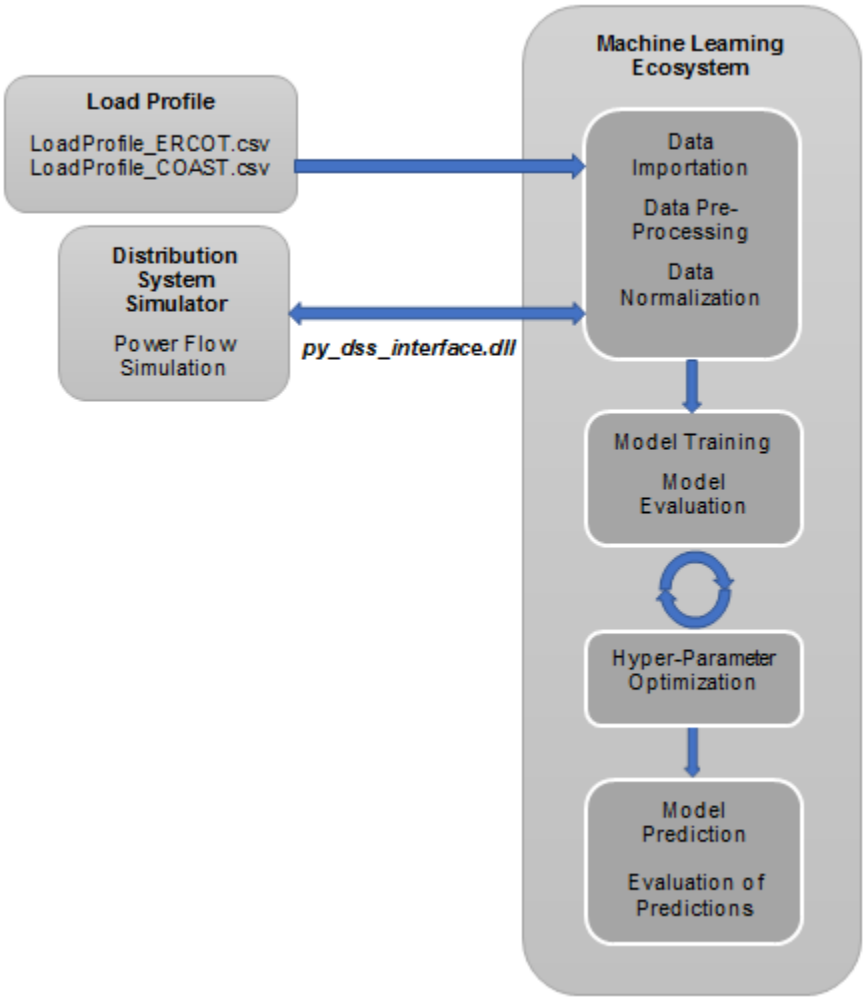


Figure 20 - Improved Workflow and Data Pipeline

4.2.1 Load Profile

The load profile .csv files are loaded directly to the Machine Learning Ecosystem.

4.2.2 Distribution System Simulator

As shown in Figure 20, a *dynamic link library (py_dss_interface.dll)* [60] enables bidirectional communication between the Machine Learning Ecosystem and the distribution system simulator. This .dll allows for control and configuration of the

distribution system simulator to be performed within the Machine Learning Ecosystem. This configuration thus simplifies the workflow by establishing the distribution system simulator as a dedicated *power flow engine*. Additionally, the need to place power and voltage monitors within the distribution system is removed along with the external data processing required in the original workflow. Finally, ability of the distribution system simulator to capture bi-directional power flow throughout the entire system and thus state estimation of much larger circuits is enabled.

4.2.3 Machine Learning Ecosystem

The Machine Learning Ecosystem is again based upon Python supported by machine learning libraries of *sklearn, Pandas, TensorFlow*, and *Keras*. Additionally, the *bayes_opt* [59] library is utilized for hyperparameter optimization based upon Bayesian Optimization with Gaussian Processes. The theory and application of this library to the research on which this dissertation is based can be found in sections 3.9.1 and 3.9.2.

As shown in Figure 20, the data pipeline is organized into *data processing*, *model training/evaluation*, *hyperparameter optimization* and *final evaluation* units.

The profile files are loaded directly to the data processing unit. A bi-directional path is established between the power flow simulation "engine" and the data processing unit enabled by the *dynamic link library (py_dss_interface.dll)*.

Initiation and configuration of the power flow simulation and related parameters such as total hours are performed within the machine learning ecosystem.

The output from the distribution system simulator is retrieved by the data processing unit and external processing is not required to concatenate files and eliminate non-numeric data. Complex power through all lines and complex voltage at all nodes are accessible

directly via the dynamic link library and read into Python data structures (i.e. Numpy Arrays [61] and Pandas Data Frames [56]). A feedback loop is established between the model training/evaluation and hyperparameter optimization units. The Multilayer Perceptron (MLP) model is passed to the hyperparameter optimization unit as an argument and the final selection of the MLP hyperparameters is performed via Bayesian Optimization with Gaussian Processes. Finally, the "optimized" MLP model is used to perform state estimation by minimizing the root-mean-squared error (RMSE) between actual and predicted values of complex voltage given validation data (new complex power data).

Specifically, these improvements enabled the following:

1. An automated method to perform power flow simulations of a test feeder with any number of nodes and lines.
2. Integration of Python and OpenDSS through the dynamic link library "py_dss_interface.dll".
3. Ability to automatically acquire complex power through all lines and complex voltage at every node for training, testing, and validation thus enabling "full-state estimation".

4.2.4 Experimental Methodology

The IEEE 34 Node Test Distribution System that was considered in previous research was also used in the updated research methodology discussed in this chapter.

4.2.5 Measurement Points and Locations

The current research features full-state estimation in the sense that all lines and all nodes of the distribution feeder are employed in gathering training, testing and validation data.

Figure 20 provides the line elements and power quantities for the IEEE 34 Node Test Distribution system considered in the current research.

Table 20 - Line Elements and Power Flow

Line Element	Node	Node	Quantities
L1	800	802	Phase A,B,C P (kW) and Q(kVAR)
L2	802	806	Phase A,B,C P (kW) and Q(kVAR)
L3	806	808	Phase A,B,C P (kW) and Q(kVAR)
L4	808	810	Phase B P (kW) and Q(kVAR)
L5	808	812	Phase A,B,C P (kW) and Q(kVAR)
L6	812	814	Phase A,B,C P (kW) and Q(kVAR)
L7	814	850	Phase A,B,C P (kW) and Q(kVAR)
L8	816	818	Phase A P(kW) and Q(kVAR)
L9	816	824	Phase A,B,C P (kW) and Q(kVAR)
L10	818	820	Phase A P(kW) and Q(kVAR)
L11	820	822	Phase A P (kW) and Q(kVAR)
L12	824	826	Phase B P (kW) and Q(kVAR)

L13	824	828	Phase A,B,C P (kW) and Q(kVAR)
L14	828	830	Phase A,B,C P (kW) and Q(kVAR)
L15	830	854	Phase A,B,C P (kW) and Q(kVAR)
L16	832	858	Phase A,B,C P (kW) and Q(kVAR)
L17	834	860	Phase A,B,C P (kW) and Q(kVAR)
L18	834	842	Phase A,B,C P (kW) and Q(kVAR)
L19	836	840	Phase A,B,C P (kW) and Q(kVAR)
L20	836	862	Phase A,B,C P (kW) and Q(kVAR)
L21	842	844	Phase A,B,C P (kW) and Q(kVAR)
L22	844	846	Phase A,B,C P (kW) and Q(kVAR)
L23	846	848	Phase A,B,C P (kW) and Q(kVAR)
L24	850	816	Phase A,B,C P (kW) and Q(kVAR)
L25	852	832	Phase A,B,C P (kW) and Q(kVAR)
L26	854	856	Phase B P (kW) and Q(kVAR)
L27	854	852	Phase A,B,C P (kW) and Q(kVAR)
L28	858	864	Phase A

			P (kW) and Q(kVAR)
L29	858	834	Phase A,B,C P (kW) and Q(kVAR)
L30	860	836	Phase A,B,C P (kW) and Q(kVAR)
L31	862	838	Phase B P (kW) and Q(kVAR)
L32	888	890	Phase A,B,C P (kW) and Q(kVAR)

It should be noted that all lines in the test feeder are being accounted for and the training data contains bi-directional power flow between each "Node" pair presented in Table 20. Table 21 provides the expanded set of node voltages and phase angles for the IEEE 34 Node Test Distribution system considered in the current research.

Table 21 - Node Voltages and Phase Angles

Node	Quantities
800	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
802	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
806	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
808	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
810	Phase B to Neutral Voltage Mag. (V) and Phase Angle (degrees)
812	Phase A,B,C to Neutral

	Voltage Mag. (V) and Phase Angle (degrees)
814	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
850	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
816	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
818	Phase A to Neutral Voltage Mag. (V) and Phase Angle (degrees)
820	Phase A to Neutral Voltage Mag. (V) and Phase Angle (degrees)
822	Phase A to Neutral Voltage Mag. (V) and Phase Angle (degrees)
824	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
826	Phase B to Neutral Voltage Mag. (V) and Phase Angle (degrees)
828	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
830	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
854	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)

852	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
832	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
858	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
834	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
842	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
844	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
846	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
848	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
860	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
836	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
840	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
862	Phase A,B,C to Neutral

	Voltage Mag. (V) and Phase Angle (degrees)
838	Phase B to Neutral Voltage Mag. (V) and Phase Angle (degrees)
864	Phase A to Neutral Voltage Mag. (V) and Phase Angle (degrees)
888	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
890	Phase A,B,C to Neutral Voltage Mag. (V) and Phase Angle (degrees)
856	Phase B to Neutral Voltage Mag. (V) and Phase Angle (degrees)

Note from Table 21 that the complex voltage at *every node* of the test distribution system are accounted for.

4.2.6 Gather Training/Testing Data

OpenDSS is used to perform a power flow simulation by utilizing a load profile for the ERCOT region. Execution and control of the OpenDSS power flow simulation is performed within the Python ecosystem utilizing *py_dss_interface.dll*. The power flow is solved for a specified number of hours. The Python ecosystem allows for efficient gathering and data processing capability to prepare the complex power flow through all lines and complex voltages at all nodes for training and testing MLP models.

4.2.7 Perform Random Selection of Lines

From the entire set of power flow lines, random selection of lines is gathered to establish datasets representing complex power flow for 10, 25, 75, and 100% of the lines. These datasets and the specific names of the lines are used as inputs to the next steps to gather the validation data and perform predictions. Table 22 provides the lines that were selected for each random trial.

4.2.8 Gather Validation Data

OpenDSS is used to perform a power flow simulation by utilizing a load profile for the COAST region. Execution and control of the OpenDSS power flow simulation is performed with the Python ecosystem by utilizing *py_dss_interface.dll*. The power flow is solved for a specified number of hours.

The line names established in 4.2.7 were used to select the same lines for the validation data sets.

4.2.9 Unoptimized MLP Models

The datasets acquired in sections 4.2.6 and 4.2.8 were utilized as input to unoptimized MLP models.

4.2.10 Optimized MLP Models

The datasets acquired in sections 4.2.6 and 4.2.8 were utilized as inputs to MLP models with hyperparameters selected by Bayesian Optimization with Gaussian Processes.

Table 24, Table 25, and Table 26 provide the results for unoptimized and optimized models in terms of training, testing, and validation RMSE and execution time.

Table 27 provides the hyperparameters selected via Bayesian Optimization.

Table 22, provides the lines selected from the available lines in Table 20 at percentages of 10, 25, 50, 75 and 100%.

Table 22 - Randomly Selected Lines

Random Selection %	R1 Lines:	R2 Lines:	R3 Lines:	R4 Lines:
10	Line.l4 Line.l17 Line.l21	Line.l6 Line.l8 Line.l32	Line.l16 Line.l28 Line.l30	Line.l4 Line.l17 Line.l21
25	Line.l4 Line.l5 Line.l15 Line.l19 Line.l24 Line.l25 Line.l31 Line.l32	Line.l5 Line.l14 Line.l15 Line.l18 Line.l20 Line.l27 Line.l28 Line.l29	Line.l9 Line.l17 Line.l18 Line.l20 Line.l25 Line.l26 Line.l29 Line.l32	Line.l3 Line.l5 Line.l8 Line.l11 Line.l15 Line.l18 Line.l24 Line.l32
50	Line.l4 Line.l5 Line.l11 Line.l12 Line.l14 Line.l15 Line.l16 Line.l17 Line.l19 Line.l20 Line.l21 Line.l22 Line.l23 Line.l28	Line.l1 Line.l3 Line.l4 Line.l5 Line.l9 Line.l10 Line.l11 Line.l16 Line.l19 Line.l22 Line.l26 Line.l28 Line.l29 Line.l30	Line.l3 Line.l7 Line.l8 Line.l9 Line.l10 Line.l17 Line.l20 Line.l22 Line.l23 Line.l24 Line.l25 Line.l27 Line.l28 Line.l29	Line.l1 Line.l3 Line.l5 Line.l7 Line.l9 Line.l11 Line.l13 Line.l14 Line.l16 Line.l17 Line.l18 Line.l19 Line.l23 Line.l24

	Line.l30 Line.l31	Line.l31 Line.l32	Line.l31 Line.l32	Line.l28 Line.l32
75	Line.l2 Line.l6 Line.l7 Line.l8 Line.l9 Line.l10 Line.l11 Line.l12 Line.l13 Line.l14 Line.l15 Line.l16 Line.l17 Line.l19 Line.l20 Line.l23 Line.l24 Line.l26 Line.l27 Line.l28 Line.l29 Line.l30 Line.l31 Line.l32	Line.l1 Line.l2 Line.l5 Line.l6 Line.l7 Line.l9 Line.l11 Line.l13 Line.l14 Line.l15 Line.l16 Line.l17 Line.l18 Line.l19 Line.l21 Line.l22 Line.l23 Line.l24 Line.l26 Line.l27 Line.l28 Line.l30 Line.l31 Line.l32	Line.l2 Line.l4 Line.l5 Line.l6 Line.l7 Line.l9 Line.l11 Line.l12 Line.l13 Line.l14 Line.l15 Line.l17 Line.l18 Line.l19 Line.l20 Line.l21 Line.l22 Line.l23 Line.l24 Line.l25 Line.l26 Line.l27 Line.l29 Line.l30	Line.l1 Line.l3 Line.l4 Line.l5 Line.l6 Line.l7 Line.l8 Line.l9 Line.l10 Line.l12 Line.l15 Line.l16 Line.l17 Line.l20 Line.l21 Line.l22 Line.l23 Line.l24 Line.l25 Line.l27 Line.l28 Line.l29 Line.l30 Line.l31
100	Line.l1 Line.l2 Line.l3 Line.l4 Line.l5 Line.l6 Line.l7 Line.l8 Line.l9 Line.l10 Line.l11 Line.l12 Line.l13 Line.l14 Line.l15 Line.l16 Line.l17 Line.l18 Line.l19 Line.l20 Line.l21 Line.l22 Line.l23 Line.l24	Line.l1 Line.l2 Line.l3 Line.l4 Line.l5 Line.l6 Line.l7 Line.l8 Line.l9 Line.l10 Line.l11 Line.l12 Line.l13 Line.l14 Line.l15 Line.l16 Line.l17 Line.l18 Line.l19 Line.l20 Line.l21 Line.l22 Line.l23 Line.l24	Line.l1 Line.l2 Line.l3 Line.l4 Line.l5 Line.l6 Line.l7 Line.l8 Line.l9 Line.l10 Line.l11 Line.l12 Line.l13 Line.l14 Line.l15 Line.l16 Line.l17 Line.l18 Line.l19 Line.l20 Line.l21 Line.l22 Line.l23 Line.l24	Line.l1 Line.l2 Line.l3 Line.l4 Line.l5 Line.l6 Line.l7 Line.l8 Line.l9 Line.l10 Line.l11 Line.l12 Line.l13 Line.l14 Line.l15 Line.l16 Line.l17 Line.l18 Line.l19 Line.l20 Line.l21 Line.l22 Line.l23 Line.l24

	Line.l25 Line.l26 Line.l27 Line.l28 Line.l29 Line.l30 Line.l31 Line.l32	Line.l25 Line.l26 Line.l27 Line.l28 Line.l29 Line.l30 Line.l31 Line.l32	Line.l25 Line.l26 Line.l27 Line.l28 Line.l29 Line.l30 Line.l31 Line.l32	Line.l25 Line.l26 Line.l27 Line.l28 Line.l29 Line.l30 Line.l31 Line.l32

Table 23 shows the results for random selection of lines corresponding to the column of "R1_Lines" in Table 22. Results for unoptimized and optimized MLP models with 1 hidden layer utilizing output data from power flow simulations of 720 hours are presented.

Note that for unoptimized models, the size of the *Input Layer* and *Output Layer* in terms of number of neurons is determined by the number of features of the training data. For example, for trial R1_10, with 10 percent of the total lines selected there are 36 features corresponding to the PQ values of each phase of each line. The *Output Layer* size of 190 neurons is fixed owing to the number of features (voltage magnitudes and voltage phase angles) comprising the target complex voltage set being predicted. This voltage dataset includes the complex voltage at all nodes and phases of the test distribution system.

The number of *Hidden Layer* neurons for the unoptimized models is set equal to the number of *Input Layer* neurons as a "reasonable" starting point and could be set to any other number in a scenario of selecting the number of neurons ad-hoc.

The choice of setting the two sizes equal is to provide some consistency in the methodology used in the current research as there are to the knowledge of the author of this dissertation no analytical methods to determine the number of neurons in the hidden layer.

The hyperparameters of the unoptimized models are presented in section 3.6.2 and repeated below for convenience.

- *Activation Function per Layer: Rectified Linear Unit (ReLU)*
- *Batch Size = 10*
- *Epochs = 200*
- *Learning Rate: 0.001*
- *Loss Function: Mean Squared Error*
- *Input Layer Neurons: 56*
- *Output Layer Neurons: 56*
- *Optimizer: Stochastic Gradient Descent*

For each "unoptimized" model in Table 23, there is a corresponding optimized model with hyperparameters determined by Bayesian Optimization with Gaussian Processes. To clarify, "R1_10" corresponds to an unoptimized MLP trained on 10% of the available lines in the distribution system to predict the complex voltage at all nodes of the distribution system. Likewise, "R1_10_Opt" corresponds to an optimized version of "R1_10" performing the same task with the same 10% selection of lines. Note that the number of *Input Layer* and *Output Layer* neurons for the optimized model matches that of the unoptimized model in each case given that the same 10% of lines is used for training and the same number of complex node voltages are being predicted.
The *Hidden Layer* neurons is determined by Bayesian Optimization. Additionally, for each optimized model the *Learning Rate, Epochs and Batch Size* are also determined by Bayesian Optimization. The result for optimized models is found in Table 23.

The columns "Train RMSE (70%)" and "Test RMSE (30%)" correspond to the training and testing data RMSE. The train/test split is set at 70/30. The column "COAST Act. vs. Est. RMSE" corresponds to the validation data RMSE and is the figure of merit for comparison of how well the MLP models generalize to new data. The column

"Execution Time (Seconds)" represents the total time of each model to train/test and perform predictions. The data validation error shown in the column "COAST Act. vs. Est. RMSE" in Table 23 indicate the trials on which Bayesian Optimization resulted in a reduction in error. It should be noted that most of the trials show improvement through optimization at the expense of increased execution time required by the Bayesian Optimization with Gaussian Processes. As discussed in section 3.9.2, the Bayesian Optimization process investigated in the current research enables a systematic approach to search over a hyperparameter space of $30x10^9$ possibilities. Table 24, Table 25, and Table 26 show the results for the other 3 trials of random selection (R2, R3, and R4). Note that the highlighted values are those on which Bayesian Optimization did not result in a reduction in RMSE for the validation dataset.

Table 23 – Performance Comparison of Unoptimized and Optimized MLP Networks Under Random Selection (R1) of Lines (Total Hours = 720)

Line %	Input Layer	Hidden Layers	Output Layer	Train RMSE (70%)	Test RMSE (30%)	COAST Act. vs. Est. RMSE	Execution Time (Seconds)
R1_10	36 Neurons	1 Layer 36 Neurons	190 Neurons	0.048141	0.051349	0.113290	29.247
R1_10_Opt	36 Neurons	1 Layer 74 Neurons	190 Neurons	0.062044	0.095299	0.055176	175.182
R1_25	80 Neurons	1 Layer 80 Neurons	190 Neurons	0.045688	0.046384	0.099254	28.090
R1_25_Opt	80 Neurons	1 Layer 30 Neurons	190 Neurons	0.180729	0.322345	0.143314	366.843
R1_50	152 Neurons	1 Layer 152 Neurons	190 Neurons	0.041447	0.051087	0.061411	25.912
R1_50_Opt	152 Neurons	1 Layer 55 Neurons	190 Neurons	0.049035	0.057948	0.038316	183.641
R1_75	336 Neurons	1 Layer 336 Neurons	190 Neurons	0.044454	0.048146	0.055020	39.162
R1_75_Opt	336 Neurons	1 Layer 74 Neurons	190 Neurons	0.047537	0.051982	0.033706	265.670
R1_100	548 Neurons	1 Layer 548 Neurons	190 Neurons	0.043145	0.047000	0.052913	48.736
R1_100_Opt	548 Neurons	1 Layer 92 Neurons	190 Neurons	0.056025	0.053020	0.035677	296.960

Table 24 - Performance Comparison of Unoptimized and Optimized MLP Networks Under Random Selection (R2) of Lines (Total Hours = 720)

Line %	Input Layer	Hidden Layers	Output Layer	Train RMSE (70%)	Test RMSE (30%)	COAST Act. vs. Est. RMSE	Execution Time (Seconds)
R2_10	28 Neurons	1 Layer 28 Neurons	190 Neurons	0.050066	0.051851	0.031051	23.463
R2_10_Opt	28 Neurons	1 Layer 87 Neurons	190 Neurons	0.066081	0.068677	0.056953	270.520
R2_25	88 Neurons	1 Layer 88 Neurons	190 Neurons	0.042833	0.051507	0.030713	26.742
R2_25_Opt	88 Neurons	1 Layer 33 Neurons	190 Neurons	0.047930	0.057454	0.048139	212.906
R2_50	268 Neurons	1 Layer 268 Neurons	190 Neurons	0.051145	0.053782	0.042308	32.392
R2_50_Opt	268 Neurons	1 Layer 100 Neurons	190 Neurons	0.056522	0.054557	0.051480	1988.112
R2_75	456 Neurons	1 Layer 456 Neurons	190 Neurons	0.044707	0.049357	0.034962	42.663
R2_75_Opt	456 Neurons	1 Layer 11 Neurons	190 Neurons	0.051553	0.061514	0.047830	164.720
R2_100	548 Neurons	1 Layer 548 Neurons	190 Neurons	0.042781	0.049381	0.030084	46.130
R2_100_Opt	548 Neurons	1 Layer 38 Neurons	190 Neurons	0.047130	0.055054	0.045304	525.401

Table 25 - Performance Comparison of Unoptimized and Optimized MLP Networks Under Random Selection (R3) of Lines (Total Hours = 720)

Line %	Input Layer	Hidden Layers	Output Layer	Train RMSE (70%)	Test RMSE (30%)	COAST Act. vs. Est. RMSE	Execution Time (Seconds)
R3_10	28 Neurons	1 Layer 28 Neurons	190 Neurons	0.048646	0.049470	0.079072	28.562
R3_10_Opt	28 Neurons	1 Layer 30 Neurons	190 Neurons	0.058915	0.064127	0.043230	473.312
R3_25	88 Neurons	1 Layer 88 Neurons	190 Neurons	0.037615	0.046131	0.061209	33.262
R3_25_Opt	88 Neurons	1 Layer 69 Neurons	190 Neurons	0.044697	0.050339	0.028499	360.209
R3_50	188 Neurons	1 Layer 188 Neurons	190 Neurons	0.040473	0.048051	0.058768	36.045
R3_50_Opt	188 Neurons	1 Layer 77 Neurons	190 Neurons	0.048241	0.055870	0.039581	306.078
R3_75	360 Neurons	1 Layer 360 Neurons	190 Neurons	0.039080	0.043416	0.053810	39.184
R3_75_Opt	360 Neurons	1 Layer 72 Neurons	190 Neurons	0.047118	0.056091	0.034844	1011.391
R3_100	548 Neurons	1 Layer 548 Neurons	190 Neurons	0.042287	0.045293	0.055810	51.044
R3_100_Opt	548 Neurons	1 Layer 77 Neurons	190 Neurons	0.050010	0.057300	0.038648	767.584

Table 26 - Performance Comparison of Unoptimized and Optimized MLP Networks Under Random Selection (R4) of Lines (Total Hours = 720)

Line %	Input Layer	Hidden Layers	Output Layer	Train RMSE (70%)	Test RMSE (30%)	COAST Act. vs. Est. RMSE	Execution Time (Seconds)
R4_10	28 Neurons	1 Layer 28 Neurons	190 Neurons	0.046668	0.048348	0.069602	23.480
R4_10_Opt	28 Neurons	1 Layer 8 Neurons	190 Neurons	0.057249	0.068387	0.044056	162.001
R4_25	108 Neurons	1 Layer 108 Neurons	190 Neurons	0.045170	0.048479	0.072729	37.063
R4_25_Opt	108 Neurons	1 Layer 94 Neurons	190 Neurons	0.071175	0.077030	0.052773	183.495
R4_50	300 Neurons	1 Layer 300 Neurons	190 Neurons	0.047717	0.050853	0.061553	37.291
R4_50_Opt	300 Neurons	1 Layer 100 Neurons	190 Neurons	0.049017	0.056798	0.039105	1111.410
R4_75	364 Neurons	1 Layer 364 Neurons	190 Neurons	0.043033	0.051618	0.060404	43.922
R4_75_Opt	364 Neurons	1 Layer 41 Neurons	190 Neurons	0.049714	0.064112	0.040147	265.596
R4_100	548 Neurons	1 Layer 548 Neurons	190 Neurons	0.043685	0.047946	0.057841	55.095
R4_100_Opt	100 Neurons	1 Layer 100 Neurons	190 Neurons	0.051265	0.057406	0.037380	1395.088

Table 27 presents the hyperparameters selected via the Bayesian Optimization with Gaussian Process. The optimization process was applied to the following hyperparameter space:

- *Hidden Layer Neurons: 1 - 100*
- *Learning Rate: 0.0001 - 1.0*
- *Batch Size: 1 - 100*

- *Epochs: 1 – 300*

Note that trial "R1_10_Opt" corresponds to the trial specified by "R1_10_Opt" in Table 23. Likewise, the remaining trials in Table 27 correspond to the trials in the column "Line %" in Table 23, Table 24, Table 25 and Table 26. The results presented in this dissertation were selected by executing the Bayesian Optimization process 15 times for each "Trial" and utilizing the result that corresponds to the best validation error score (lowest RMSE).

Table 27 - Hyperparameters Selected via Bayesian Optimization

Trial	Neurons per Hidden Layer	Learning Rate	Epochs	Batch Size
R1_10_Opt	74	0.0210	71	27
R1_25_Opt	30	0.4181	291	33
R1_50_Opt	55	0.0010	178	50
R1_75_Opt	74	0.0010	198	38
R1_100_Opt	92	0.0010	260	100
R2_10_Opt	87	0.0310	181	36
R2_25_Opt	33	0.0010	161	24
R2_50_Opt	100	0.0010	164	1
R2_75_Opt	11	0.0020	228	73
R2_100_Opt	38	0.0010	243	37
R3_10_Opt	30	0.0091	127	66
R3_25_Opt	69	0.0010	300	33
R3_50_Opt	77	0.0010	241	77
R3_75_Opt	72	0.0010	200	42
R3_100_Opt	77	0.0010	251	23
R4_10_Opt	8	0.0132	195	74
R4_25_Opt	94	0.0306	85	49
R4_50_Opt	100	0.0010	244	74
R4_75_Opt	41	0.0010	189	65
R4_100_Opt	100	0.0010	224	99

Notable items are as follows from the data:

- Bayesian Optimization with Gaussian Processes enabled an efficient search of hyperparameter space of $30x10^9$ combinations.

- A reduction in validation error was achieved for 70% of the trials via Bayesian Optimization.
- In all trials involving Bayesian Optimization with Gaussian Processes, a trade off existed in searching for "optimal" hyperparameters and an increase in execution time.

The improved methodology presented in this chapter accomplished the following:

- Established an improved workflow and data pipeline for training, testing and validation through the integration of a high-level programming language (Python) and a distribution system simulator (OpenDSS).
- Demonstrated the ability of an improved workflow and data pipeline to automate the acquisition of complex power through all lines and complex voltages at every node for training, testing and validation thus enabling "full-state estimation".
- Enabled investigation of the effect of utilizing specific percentages of complex power flow with randomly selected lines to train MLPs to predict the complex voltage at all the nodes.
- Enabled investigation of the application of Bayesian Optimization with Gaussian processes to aid in the improvement (reduction) of validation error through the selection of hyperparameters for MLP models (neurons per hidden layer, learning rate, epochs, and batch size).

Items for consideration in future work are as follows:

- Application of the workflow and data pipeline presented in this dissertation to larger test feeder systems (i.e. IEEE 8500 Node Test Feeder).

- Investigation of the significance of the lines selected to perform state estimation of complex voltage.
- Investigation of state estimation based upon branch-current flow. Specifically, utilize branch current flow to predict complex voltage utilizing optimized MLP models.
- Investigate Bayesian Optimization with Gaussian Processes with a hyperparameter space including number of hidden layers.

CHAPTER 5. DISTRIBUTION SYSTEM STATE FORECASTING (DSSF) WITH CONVOLUTIONAL NEURAL NETWORK (CNN) MODELS

5.1 Time-Series Forecasting

In this research, a time-series will be defined as an ordered sequence of values that are equally spaced. A typical time-series for real power flow between two buses in a power distribution system is shown below in Figure 21. In particular, this time-series is the real power flow between nodes 800 and 802 for phase-a captured each hour of the day (10/20/2018) for the test distribution system shown in Figure 7. Note that this time-series is considered "univariate" in that it represents a single value for each time step.

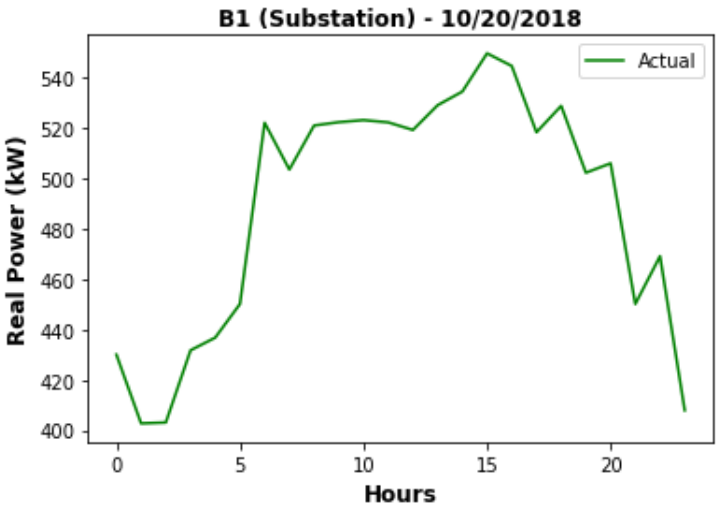


Figure 21 - Example Univariate Time Series

Time-series forecasting will be defined as training a statistical, machine learning or deep learning model to learn the autocorrelation of a given time series and to perform a prediction for a specified prediction horizon. Thus, framing a problem as a time-series forecasting problem is in contrast to framing a problem as a *functional regression problem* in which the mapping of inputs to outputs is learned and new data is predicted based upon the input of new previously unseen data. It should be noted that distribution system state estimation (DSSE) with multilayer perceptron (MLP) models presented in CHAPTER 3 was framed as a *functional regression problem*. Time-series forecasting utilizing deep learning techniques does not require and is thus not limited to having to be trained on existing historical "input" and "target" datasets. Forecasting models are said to therefore "extract" features from the input data itself and perform prediction by learning from *trend*, *seasonality*, and *residual noise* within the data itself (autocorrelation**).** A major motivation for the research based upon machine learning for distribution system state forecasting is that predictions of system state can be made without the requirement

to train offline with vast amounts of historical data that must be retrieved from a database or knowledge of distribution system topology beyond the available measurement devices capturing the time-series data. In order to move toward a higher level of performance guarantee required for greater acceptance of machine learning techniques in the power industry, the current research being discussed in this dissertation is built upon the idea of enhancement of data driven methods with constraints, repeatable processes and a structured approach to model design.

5.1.1 Time-Series Forecasting Process

Moving towards a repeatable process for time-series forecasting, the methods promoted by Hyndman and Athanasopoulos [41] were employed in the research. This process can be summarized as follows:

1. **Define the Problem:** Determination of who will be utilizing the forecast and in what capacity.
2. **Gather Information and Datasets:** Selection of historical data and insights from domain experts to aid interpretation of data features and the results of the forecasts.
3. **Exploratory Analysis:** Utilization of summary statistics to understand trends, seasonality, outliers, missing data and anomalies.
4. **Model Selection and Fitting:** Consideration of classical statistical models, machine learning, deep learning, etc. and initial fitting of models to make final selection of the model type best suited for forecasting given a specific dataset.
5. **Evaluation and Validation of the Chosen Forecasting Model:** Forecasts are made with the selected model and relative skill of the model is estimated through back-testing with historical data.

These steps are implemented as follows:

1. **Define the Problem:**
    - Given sufficient historical real power flow, reactive power flow, voltage magnitude and voltage phase angle data up to a specified point during 2018, forecast these quantities for the next 24 hours as a time-series.
2. **Gather Information and Datasets:**
    - ERCOT data discussed in section 3.4 will represent the input data.

- Real and reactive power flows on phase-a between nodes 854 and 852 as measured by monitor B27 in Figure 7 are to be forecasted.
- Voltage magnitude and voltage phase angles at node 854 as measured by monitor B27 in Figure 7 are to be forecasted.

3. **Exploratory Analysis:**
    - Plots of the actual time-series were made to understand the nature of the dynamics (changes over time) for each quantity under consideration (i.e. real power, reactive power, voltage magnitude and voltage phase angle). Utilization of deep learning models eliminates the need to identify and remove seasonality and trend. All time-series considered in this research were continuous over the time-periods on which the data was gathered.
4. **Model Selection and Fitting:**
    - MLPs were considered based upon their simple structure, however not deemed strong candidates given their limitations of utilizing data in early portions of the time series for forecasting owing to vanishing gradients.
    - CNNs and LSTMs were considered based on their ability to "learn" and "remember" features of the source data and thus overcome some of the limitations of vanishing gradients.
5. **Evaluation and Validation of the Chosen Forecasting Model:**
    - CNN and LSTM models were evaluated in terms of their ability to perform time-series forecasting over a 24-hour forecast horizon.
    - Actual data was available for the forecast horizon for comparison of each model type.
    - The average RMSE of each model type was determined and each forecasted time series was plotted along with the actual time series for visual comparison.

5.2 Data Preparation for Time-Series Forecasting with Deep Learning

Two essential considerations for use of deep learning models such as CNNs and LSTMs for time-series forecasting are as follows:

- Both models require data to be 3-dimensional
- Performance is greatly affected by number of time-steps, so that sequences with more than 400 time steps are split into samples.
- Order must be maintained with splitting time-series into training and testing

The term often used in machine learning is that of "reshaping" original time-series data into the general format of [# samples, # time-steps, # features]. In this research the following steps were taken to reshape the ERCOT dataset to prepare it for time-series forecasting:

- ERCOT Dataset General Structure: (#time-steps, #features)
    - Rows of the dataset correspond to each hour of the year of 2018
    - Columns of the dataset correspond to the real powers, reactive powers, voltage magnitudes and voltage phase angles for each bus and each phase
    - Dimensions: (8760, 112)
- Maintaining the temporal ordering of the time-series dataset:
    - Splitting the dataset into 80% for Training and 20% for Testing
    - First 7008 hours – Training Dataset Dimensions: (7008, 112)
    - Remaining 1752 hours – Testing Dataset Dimensions: (1752, 112)
- Reshaping the data into 3D-Arrays: (#samples, #time-steps, #features)
    - Reshaped Training Dataset Dimensions: (292, 24, 112)
    - Reshaped Testing Dataset Dimensions: (73, 24, 112)

The following should be noted from the reshaping process:

- The first 7008 hours of the data represent the first 292 days of 2018 in the ERCOT region under consideration.
- The remaining 1752 hours of the data represent the remaining 73 days of 2018 in the ERCOT region under consideration. Of the 73 days available for testing, only the first day of this period was used as the forecast horizon. This forecast horizon was selected arbitrarily and could have been extended for predictions beyond 24 hours.
- Time-series forecasting considered in this research involved "training" on the first 292 days and then using deep learning models to forecast the next 24 hours beyond this training period. In this case, a forecast would be made for the real power flow, reactive power flow, voltage magnitude and voltage phase angle at the monitor location B27 for day 293 of 2018 (October 20, 2018).
- The forecasts would then be compared to the known data for this same day.

5.3 Hyperparameter Selection for Unoptimized CNN Model

Convolutional Neural Networks are among a class of machine learning models that are termed as "deep learning" models to differentiate them from traditional feedforward multilayer perceptron models. While their design and initial inception was largely for use in applications such as object recognition, CNNs are being investigated and applied to applications related to computer vision, natural language processing and time-series forecasting. The three fundamental building blocks of CNNs are convolutional layers, pooling layers, and fully-connected layers [29]. CNNs can be designed to perform auto-regression and classification on time-series data.

Table 28 presents the hyperparameters considered for the CNN models in this dissertation.

Table 28 - CNN Hyperparameters

CNN Hyperparameter	Description
n_input	Number of prior inputs for the model (e.g. 24 hours)
n_filters	Number of filter maps in the convolutional layer(s) (e.g. 100)
n_kernel	Kernel size in the convolutional layer (e.g. 3)
n_fc	Number of neurons in the output fully connected layer (s) (e.g. 100)
n_epochs	Number of training epochs (e.g. 10)
n_batch	Number of samples to include in each mini-batch (e.g. 10).
n_diff	Difference order to remove seasonality and trends (e.g. 0 or 12)
activation	Activation function (e.g. relu)
loss	Loss function to minimize (e.g. mse)
opt	Algorithm to adjust weights, biases and learning rate to reduce losses (e.g. adam, sgd)
lr	Learning rate. Step size at each iteration while moving toward a minimum of a loss function

Table 29 presents the unoptimized CNN hyperparameters settings selected ad hoc.

Table 29 - Unoptimized CNN Hyperparameter Settings

CNN Hyperparameter	Settings
n_input	24
n_filters	32
n_kernel	3
n_fc	112
n_epochs	70
n_batch	16
n_diff	0
activation	relu
loss	mse
opt	adam
lr	0.001

5.4 Implementation of Unoptimized CNN Model for Time-Series Forecasting

The following covers the essential elements required to perform time-series forecasting with an unoptimized CNN model. For brevity, the forecast for real power flow described in section 5.2 is shown.

Code Listing 4 - Unoptimized CNN Model for DSSF

```
000 # Unoptimized CNN Model for DSSF
001 import time
002 start_time = time.time()
003
004 # Import Required Python Libraries
005 import numpy
006 import numpy as np
007 import pandas as pd
008 from math import sqrt
009 from numpy import split
010 from numpy import array
011 from pandas import read_csv
012 from matplotlib import pyplot
013 from keras.models import Sequential
014 from keras.layers import Dense
015 from keras.layers import Flatten
```

```
016 from keras.layers.convolutional import Conv1D
017 from keras.layers.convolutional import MaxPooling1D
018 from keras.layers.convolutional import AveragePooling1D
019 from sklearn.metrics import mean_squared_error
020 from sklearn.preprocessing import MinMaxScaler
021 from sklearn.model_selection import train_test_split
022
023 # Evaluate one or more daily forecasts against expected values
024 def evaluate_forecasts(actual, predicted):
025     scores = list()
026     # calculate an RMSE score for each day
027     for i in range(actual.shape[1]):
028         # calculate mse
029         mse = mean_squared_error(actual[:, i], predicted[:, i])
030         # calculate rmse
031         rmse = sqrt(mse)
032         # store
033         scores.append(rmse)
034     # calculate overall RMSE
035     s = 0
036     for row in range(actual.shape[0]):
037         for col in range(actual.shape[1]):
038             s += (actual[row, col] - predicted[row, col])**2
039     score = sqrt(s / (actual.shape[0] * actual.shape[1]))
040     return score, scores
041
042 # Summarize Scores
043 def summarize_scores(name, score, scores):
044     s_scores = ', '.join(['%.6f' % s for s in scores])
045     print('%s: [%.6f] %s' % (name, score, s_scores))
046
047 # Convert History into Inputs and Outputs
048 def to_supervised(train, n_input, n_out=24):
049     # flatten data
050     data = train.reshape((train.shape[0]*train.shape[1],
train.shape[2]))
051     X, y = list(), list()
052     in_start = 0
053     # step over the entire history one time step at a time
054     for _ in range(len(data)):
055         # define the end of the input sequence
056         in_end = in_start + n_input
057         out_end = in_end + n_out
058         # ensure we have enough data for this instance
059         if out_end <= len(data):
060             X.append(data[in_start:in_end, :])
061             y.append(data[in_end:out_end, 50])
062         # move along one time step
063         in_start += 1
064     return array(X), array(y)
065
066 # Train the Model
067 def build_model(train, n_input):
068     # prepare data
069     train_x, train_y = to_supervised(train, n_input)
070     # define parameters
071     verbose, epochs, batch_size = 0, 70, 16
```

```
072     n_timesteps, n_features, n_outputs = train_x.shape[1],
train_x.shape[2],train_y.shape[1]
073     # define model
074     model = Sequential()
075     model.add(Conv1D(32,3, activation='relu',
input_shape=(n_timesteps,n_features)))
076     model.add(MaxPooling1D())
077     model.add(Flatten())
078     #model.add(Dense(32,3, activation='relu'))
079     model.add(Dense(n_outputs))
080     model.compile(loss='mse', optimizer='adam')
081     # fit network
082     model.fit(train_x, train_y, epochs=epochs,
batch_size=batch_size, verbose=verbose)
083     #model.summary()
084     return model
085
086 # Make a Forecast
087 def forecast(model, history, n_input):
088     # flatten data
089     data = array(history)
090     data = data.reshape((data.shape[0]*data.shape[1],
data.shape[2]))
091     # retrieve last observations for input data
092     input_x = data[-n_input:, :]
093     # reshape into [1, n_input, n]
094     input_x = input_x.reshape((1, input_x.shape[0],
input_x.shape[1]))
095     # forecast the next week
096     yhat = model.predict(input_x, verbose=0)
097     # we only want the vector forecast
098     yhat = yhat[0]
099     return yhat
100
101 # Evaluate a Single Model
102 def evaluate_model(train, test, n_input, k):
103     # fit model
104     model = build_model(train, n_input)
105     # history is a list of daily data
106     history = [x for x in train]
107     # walk-forward validation over each day
108     predictions = list()
109     for i in range(len(test)):
110         # predict the week
111         yhat_sequence = forecast(model, history, n_input)
112         # store the predictions
113         predictions.append(yhat_sequence)
114         # get real observation and add to history for predicting
the next day
115         history.append(test[i, :])
116     # evaluate predictions days for each day
117     predictions = array(predictions)
118     score, scores = evaluate_forecasts(test[:, :, 50], predictions)
119     #history_array = array(history)
120     return score, scores, predictions, history, test
121
122 # Load the PV dataset file
```

```
123 pv_dataframe = pd.read_csv('C:/Python/power_voltage_time-
series_ercot.csv',header=None)
124 pv = pv_dataframe.values
125
126 # Normalize the datasets
127 scaler_pv = MinMaxScaler(feature_range=(-1,1),copy=True)
128 pv = scaler_pv.fit_transform(pv)
129
130 # Split Dataset into 80% Train and 20% Test
131 pv_train, pv_test = pv[0:7008], pv[7008:8760]
132
133 # First 7008 hours for Training
134 # Reshape Training time series data into [samples, timesteps,
features]
135 # Dividing by 24 hours yields: pv_train 24 dimensions of (292, 24,
112)
136 pv_train_24 = array(split(pv_train, len(pv_train)/24))
137
138
139 # Remaining 1752 hours for Testing
140 # Reshape Testing time series data into [samples, timesteps,
features]
141 # Dividing by 24 hours yields: pv_test 24 dimensions of (73, 24,
112)
142 pv_test_24 = array(split(pv_test, len(pv_test)/24))
143
144 # Evaluate Model and Get Scores
145 n_input = 24
146 k = 3
147 score, scores, predictions, history, test =
evaluate_model(pv_train_24, pv_test_24,
148
n_input,k)
149 history_array = array(history)
150
151 # Summarize Scores
152 summarize_scores('cnn', score, scores)
153
154 # Create datasets to plot
155 p_10_20_2018_actual = pv_test[0:24,50]
156 p_10_20_2018_predicted = predictions[0,:]
157
158
159 p_10_20_2018_actual_avg = test[:,:,50].mean(axis=0)
160 p_10_20_2018_predicted_avg = predictions.mean(axis=0)
161
162 # Plot Actual versus Predicted
163 pyplot.xlabel('Hours',fontweight = 'bold', fontsize = 12.0,
color='black')
164 pyplot.ylabel('Real Power (kW)', fontweight = 'bold', fontsize =
12.0, color='black')
165
166 pyplot.plot(p_10_20_2018_actual,'g') # plotting t, a separately
167 pyplot.plot(p_10_20_2018_predicted, 'r') # plotting t, b separately
168
169 pyplot.title('CNN - B27 - 10/20/2018', fontweight = 'bold',
fontsize = 12.0,)
```

```
170 pyplot.legend(["Actual", "CNN Pred."])
171 pyplot.show()
172
173 # Export predictions
174 numpy.savetxt('CNN_B27_Trial_01_24_p.csv', p_10_20_2018_predicted,
delimiter=",")
175
176 print("--- %s seconds ---" % (time.time() - start_time))
```

Table 30 presents a summary of Code Listing 4 for an unoptimized CNN model for distribution system state forecasting.

Table 30 - Summary of Code Listing 4 for an Unoptimized CNN Model for DSSF

Line #:	Description:
004 – 021	Import essential Python libraries for implementation of unoptimized CNN model for DSSF
023 - 040	Define a function to evaluate one or more daily forecasts against expected values
043 – 045	Define a function summarize RMSE scores
048 – 064	Define a function to convert 3 dimensional data into inputs and outputs, thus establishing "supervised" learning
067 – 084	Define a function to build a unoptimized CNN to perform DSSF
087 – 099	Define a function to perform a forecast over a specified forecast horizon (i.e., 24 hours)
102 - 120	Define a function to evaluate the forecast based upon RMSE
123 - 124	Import time-series data
127 - 128	Perform normalization on input data to improve training
131	Split time-series data into 80% for training and 20% for testing. Note that the order of the time-series data must be maintained.
133 – 142	Reshape training and testing data into 3 dimensional format required by CNNs.
144 - 147	Call evaluate_model() function to create a "history" of daily scores
152	Call summarize_scores() function to return an average hourly RMSE for the forecast.
155 - 171	Plot actual versus predicted real power for a 24 hour forecast horizon.
174	Export the time-series prediction data.

Figure 22 presents the actual and predicted real power at location B27 (Figure 7) for a 24 hour period utilizing an "unoptimized" CNN model. Note that the predicted time-series has a noticeable "delay" or "lag" relative to the actual time-series in the early portion of the prediction. Later portions of the prediction follow the actual dynamics reasonably well.

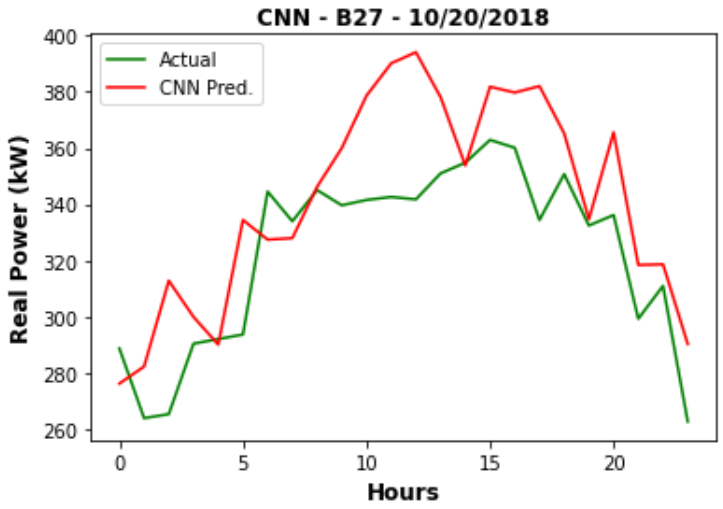


Figure 22 - Actual and Predicted Real Power at Location at B27 (Unoptimized CNN)

Figure 23 presents the actual and predicted reactive power at location B27 (Figure 7) for a 24 hour period utilizing an "unoptimized" CNN model. Note that the predicted time-series has a "flat" or "constant" waveform. This result is indicative of "underfitting" in which the model does not improve during the training process and maintains either a constant or increasing error.

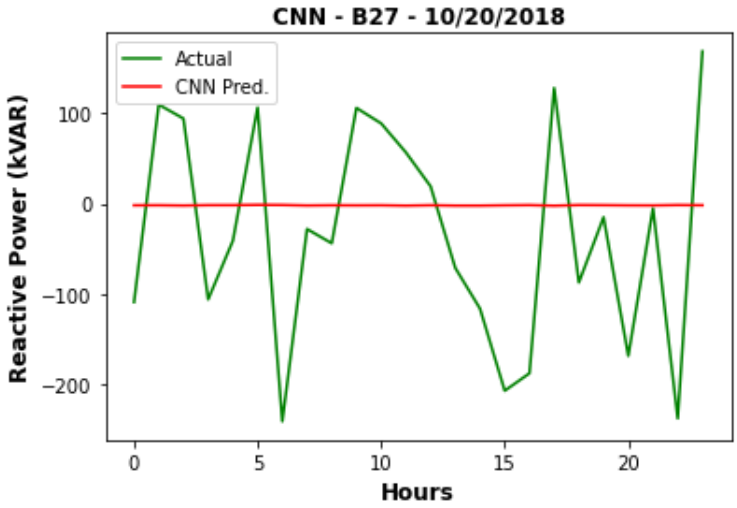


Figure 23 - Actual and Predicted Reactive Power at Location B27 (Unoptimized CNN)

Figure 24 presents the actual and predicted voltage magnitude at location B27 (Figure 7) for a 24 hour period utilizing an "unoptimized" CNN model. The predicted time-series appears to be delayed and has a "reciprocal" relationship to the actual time-series.

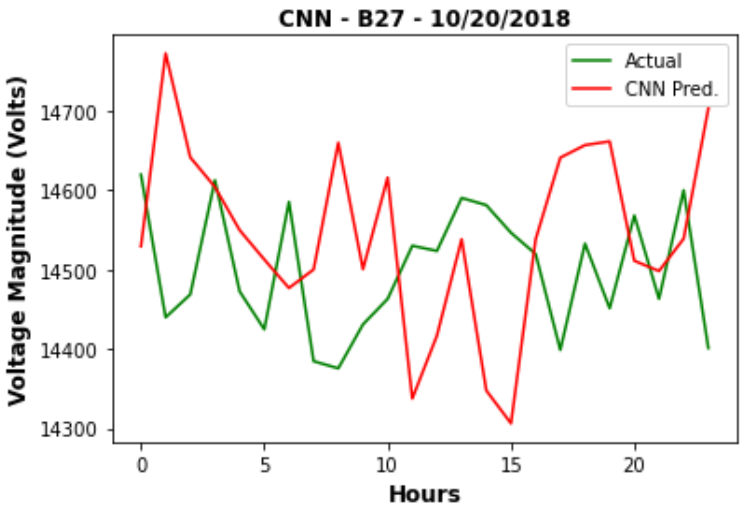


Figure 24 - Actual and Predicted Voltage Magnitude at Location B27 (Unoptimized CNN)

Figure 25 presents the actual and predicted voltage phase angle at location B27 (Figure 7) for a 24 hour period utilizing a "unoptimized" CNN model. The predicted time-series appears to track the average value of the actual time-series. This is also indicative of underfitting.

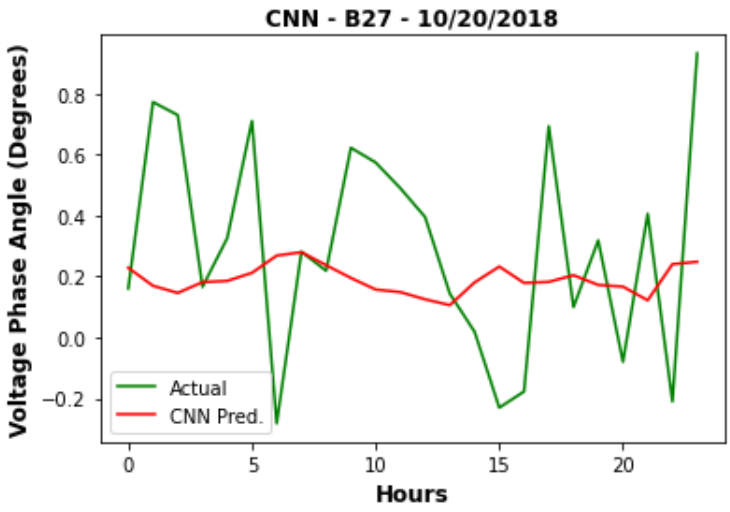


Figure 25 - Actual and Predicted Voltage Phase Angle at Location B27 (Unoptimized CNN)

A possible remedy for the "flat" waveform noted for the prediction of reactive power shown in Figure 23 is to adjust the learning rate of the Adam optimizer. The learning rate is the proportion that weights are updated. A larger value (i.e. 0.5) results in faster initial learning during training. A smaller value (i.e. 1e-06) results in a slower initial learning during training. By default, the *learning rate* established by the Keras Python library used in the implementation of the unoptimized CNN model is 0.001.

The Adam optimization algorithm has gained considerable attention in recent years as an improvement over the classical stochastic gradient descent algorithm that is used in machine learning to iteratively adjust network weights during training [37].

This method arose to prominence based upon the collaboration of OpenAI and the University of Toronto in 2015 [39].

The Adam optimizer offers advantages over classical stochastic gradient descent. The Adam optimizer is ideal for non-convex optimization problems, is computationally efficient, has low memory requirements, is suitable for large datasets, is well suited for

non-stationary objective functions and is tolerant of objective functions with noisy gradients.

The following result was observed for a reduction in the learning rate to a value of 1e-06 in the forecast of reactive power. As shown in Figure 26, the reduction in the learning rate enabled the model to improve the forecast for reactive power.

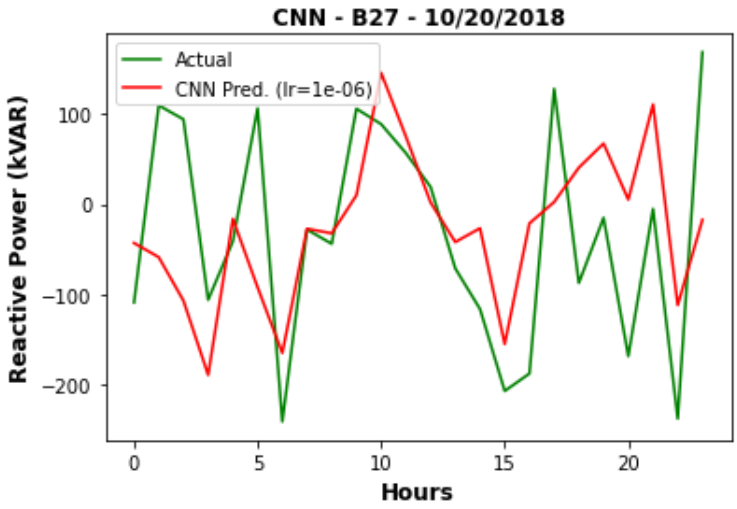


Figure 26 - Actual and Predicted Reactive Power at Location B27 (Unoptimized CNN – Learning Rate = 1e-06)

CHAPTER 6. DISTRIBUTION SYSTEM STATE FORECASTING (DSSF) WITH LONG SHORT-TERM MEMORY (LSTM) MODELS

6.1 Data Preparation for LSTMs

Data preparation for LSTMs for time-series forecasting follows the same process as that for CNNs discussed in section 5.2.

6.2 Hyperparameter Selection for Unoptimized LSTM Model

Table 31 presents the hyperparameters considered for the LSTM models in this dissertation.

Table 31 - LSTM Hyperparameters

LSTM Hyperparameter	Description
n_input	Number of prior inputs for the model (e.g. 24 hours)
n_lstm	Number of lstm blocks (e.g. 100)
n_fc	Number of neurons in the output fully connected layer (s) (e.g. 100)
n_epochs	Number of training epochs (e.g. 10)
n_batch	Number of samples to include in each mini-batch (e.g. 10).
n_diff	Difference order to remove seasonality and trends (e.g. 0 or 12)
activation	Activation function (e.g. relu)
loss	Loss function to minimize (e.g. mse)
opt	Algorithm to adjust weights, biases and learning rate to reduce losses (e.g. adam, sgd)
lr	Learning rate. Step size at each iteration while moving toward a minimum of a loss function

Table 32 presents the unoptimized LSTM hyperparameters settings selected ad hoc.

Table 32 - Unoptimized LSTM Hyperparameter Settings

LSTM Hyperparameter	Settings
n_input	24
n_lstm	150
n_fc	112
n_epochs	70
n_batch	16
n_diff	0
activation	relu
loss	mse
opt	adam
lr	0.001

6.3 Implementation of Unoptimized LSTM Model for Time-Series Forecasting

The following covers the essential elements required to perform time-series forecasting with an unoptimized LSTM model. For brevity, the forecast for real power flow described section 5.2 is shown.

Code Listing 5 - Unoptimized LSTM Model for DSSF

```
000 # Unoptimized LSTM Model for DSSF
001 import time
002 start_time = time.time()
003
004 # print("--- %s seconds ---" % (time.time() - start_time)
005
006 # Import Required Python Libraries
007 import numpy
008 import numpy as np
009 import pandas as pd
010 from math import sqrt
011 from numpy import split
012 from numpy import array
013 from pandas import read_csv
014 from matplotlib import pyplot
015 from keras.models import Sequential
016 from keras.layers import Dense
017 from keras.layers import LSTM
018 from keras.layers import RepeatVector
019 from keras.layers import TimeDistributed
020 from sklearn.metrics import mean_squared_error
021 from sklearn.preprocessing import StandardScaler, MinMaxScaler
022 from sklearn.model_selection import train_test_split
023 from sklearn.metrics import mean_squared_error
024
025
026 # Evaluate one or more daily forecasts against expected values
027 def evaluate_forecasts(actual, predicted):
028     scores = list()
029     # calculate an RMSE score for each day
030     for i in range(actual.shape[1]):
031         # calculate mse
032         mse = mean_squared_error(actual[:, i], predicted[:, i])
033         # calculate rmse
034         rmse = sqrt(mse)
035         # store
036         scores.append(rmse)
037     # calculate overall RMSE
038     s = 0
039     for row in range(actual.shape[0]):
```

```
040         for col in range(actual.shape[1]):
041             s += (actual[row, col] - predicted[row, col])**2
042     score = sqrt(s / (actual.shape[0] * actual.shape[1]))
043     return score, scores
044
045 # Summarize Scores
046 def summarize_scores(name, score, scores):
047     s_scores = ', '.join(['%.6f' % s for s in scores])
048     print('%s: [%.6f] %s' % (name, score, s_scores))
049
050 # Convert History into Inputs and Outputs
051 def to_supervised(train, n_input, n_out=24):
052     # flatten data
053     data = train.reshape((train.shape[0]*train.shape[1],
train.shape[2]))
054     X, y = list(), list()
055     in_start = 0
056     # step over the entire history one time step at a time
057     for _ in range(len(data)):
058         # define the end of the input sequence
059         in_end = in_start + n_input
060         out_end = in_end + n_out
061         # ensure we have enough data for this instance
062         if out_end <= len(data):
063             X.append(data[in_start:in_end, :])
064             y.append(data[in_end:out_end, 50])
065         # move along one time step
066         in_start += 1
067     return array(X), array(y)
068
069 # Train the Model
070 def build_model(train, n_input):
071     # prepare data
072     train_x, train_y = to_supervised(train, n_input)
073     # define parameters
074     verbose, epochs, batch_size = 0, 70, 16
075     n_timesteps, n_features, n_outputs = train_x.shape[1],
train_x.shape[2], train_y.shape[1]
076     # define model
077     model = Sequential()
078     model.add(LSTM(150, activation='relu',
input_shape=(n_timesteps, n_features)))
079     #model.add(Dense(100, activation='relu'))
080     model.add(Dense(n_outputs))
081     model.compile(loss='mse', optimizer='adam')
082     # fit network
083     model.fit(train_x, train_y, epochs=epochs,
batch_size=batch_size, verbose=verbose)
084     return model
085
086 # Make a Forecast
087 def forecast(model, history, n_input):
088   # flatten data
089   data = array(history)
090   data = data.reshape((data.shape[0]*data.shape[1], data.shape[2]))
091   # retrieve last observations for input data
092   input_x = data[-n_input:, :]
```

```
093   # reshape into [1, n_input, n]
094   input_x = input_x.reshape((1, input_x.shape[0],
input_x.shape[1]))
095   # forecast the next week
096   yhat = model.predict(input_x, verbose=0)
097   # we only want the vector forecast
098   yhat = yhat[0]
099   return yhat
100
101 # Evaluate a Single Model
102 def evaluate_model(train, test, n_input, k):
103     # fit model
104     model = build_model(train, n_input)
105     # history is a list of daily data
106     history = [x for x in train]
107     # walk-forward validation over each day
108     predictions = list()
109     for i in range(len(test)):
110         # predict the week
111         yhat_sequence = forecast(model, history, n_input)
112         # store the predictions
113         predictions.append(yhat_sequence)
114         # get real observation and add to history for predicting
the next day
115         history.append(test[i, :])
116     # evaluate predictions days for each day
117     predictions = array(predictions)
118     score, scores = evaluate_forecasts(test[:, :, 50], predictions)
119     #history_array = array(history)
120     return score, scores, predictions, history, test
121
122 # Load the PV dataset file
123 pv_dataframe = pd.read_csv('C:/Python/power_voltage_time-
series_ercot.csv',header=None)
124 pv = pv_dataframe.values
125
126 # Normalize the datasets
127 scaler_pv = MinMaxScaler(feature_range=(-1,1),copy=True)
128 pv = scaler_pv.fit_transform(pv)
129
130 # Split dataset into 80% Train and 20% Test
131 pv_train, pv_test = pv[0:7008], pv[7008:8760]
132
133 # First 7008 hours for Training
134 # Reshape Training time series data into [samples, timesteps,
features]
135 # Dividing by 24 hours yields: pv_train 24 dimensions of (292, 24,
112)
136 pv_train_24 = array(split(pv_train, len(pv_train)/24))
137
138 # Remaining 1752 hours for Testing
139 # Reshape Testing time series data into [samples, timesteps,
features]
140 # Dividing by 24 hours yields: pv_test 24 dimensions of (73, 24,
112)
141 pv_test_24 = array(split(pv_test, len(pv_test)/24))
142
```

```
143 # Evaluate Model and Get Scores
144 n_input = 24
145 k = 3
146 score, scores, predictions, history, test =
evaluate_model(pv_train_24, pv_test_24,
147
n_input,k)
148 history_array = array(history)
149 # summarize scores
150 summarize_scores('lstm', score, scores)
151
152 # Create datasets to plot
153 p_10_20_2018_actual = pv_test[0:24,50]
154 p_10_20_2018_predicted = predictions[0,:]
155
156 p_10_20_2018_actual_avg = test[:,:,50].mean(axis=0)
157 p_10_20_2018_predicted_avg = predictions.mean(axis=0)
158
159 # Plot Actual versus Predicted
160 pyplot.xlabel('Hours',fontweight = 'bold', fontsize = 12.0,
color='black')
161 pyplot.ylabel('Real Power (kWatts)', fontweight = 'bold', fontsize
= 12.0, color='black')
162
163 pyplot.plot(p_10_20_2018_actual,'g') # plotting t, a separately
164 pyplot.plot(p_10_20_2018_predicted, 'r') # plotting t, b separately
165
166 pyplot.title('LSTM - B27 - 10/20/2018', fontweight = 'bold',
fontsize = 12.0,)
167 pyplot.legend(["Actual", "LSTM Pred."])
168 pyplot.show()
169
170 # Export predictions
171 numpy.savetxt('LSTM_B27_Trial_01_24_q.csv', p_10_20_2018_predicted,
delimiter=",")
172
173 print("--- %s seconds ---" % (time.time() - start_time))
```

Table 33 presents a summary of Code Listing 5 for an unoptimized LSTM model for distribution system state forecasting.

Table 33 - Summary of Code Listing 5 for Unoptimized LSTM Model for DSSF

Line #:	Description:
006 – 023	Import essential Python libraries for implementation of unoptimized LSTM model for DSSF
027 - 043	Define a function to evaluate one or more daily forecasts against expected values
045 – 048	Define a function summarize RMSE scores
051 – 067	Define a function to convert 3 dimensional data into inputs and outputs, thus establishing "supervised" learning
070 – 084	Define a function to build a unoptimized LSTM to perform DSSF
087 – 099	Define a function to perform a forecast over a specified forecast horizon (i.e., 24 hours)
102 - 120	Define a function to evaluate the forecast based upon RMSE
123 - 124	Import time-series data
127 - 128	Perform normalization on input data to improve training
131	Split time-series data into 80% for training and 20% for testing. Note that the order of the time-series data must be maintained.
134 – 141	Reshape training and testing data into 3 dimensional format required by LSTMs.
144 - 147	Call evaluate_model() function to create a "history" of daily scores
150	Call summarize_scores() function to return an average hourly RMSE for the forecast.
153 - 168	Plot actual versus predicted real power for a 24 hour forecast horizon.
171	Export the time-series prediction data.

Figure 27 presents the actual and predicted real power at location B27 (Figure 7) for a 24 hour period utilizing an “unoptimized” LSTM model. Note that, like the same prediction made by an unoptimized CNN in Figure 22, the predicted time-series has a noticeable “delay” or “lag” relative to the actual time-series in the early portion of the prediction. Later portions of the prediction follow the actual dynamics reasonably well.

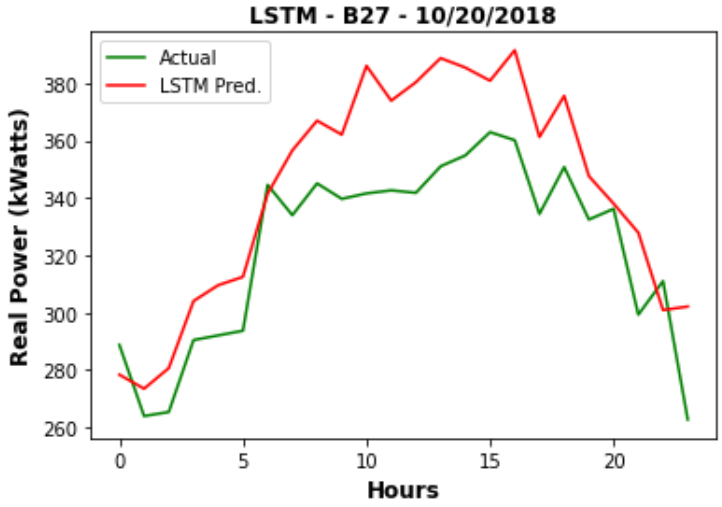


Figure 27 - Actual and Predicted Real Power at Location B27 (Unoptimized LSTM)

Figure 28 presents the actual and predicted reactive power at location B27 (Figure 7) for a 24hr period utilizing an “unoptimized” LSTM model. Note that, unlike the unoptimized CNN shown in Figure 23, the prediction of reactive power follows the dynamics of the actual time-series better with a default learning rate of 0.001.

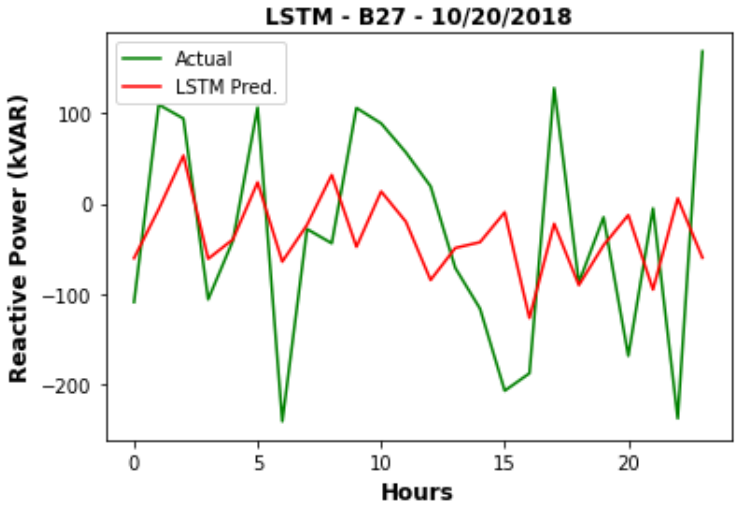


Figure 28 - Actual and Predicted Reactive Power at Location B27 (Unoptimized LSTM)

Figure 29 presents the actual and predicted voltage magnitude at location B27 (Figure 7) for a 24 hour period utilizing an "unoptimized" LSTM model. Note that the predicted time-series appears to be delayed and has a "reciprocal" relationship to the actual time-series. This is similar to the prediction of voltage magnitude made with an unoptimized CNN in Figure 24.

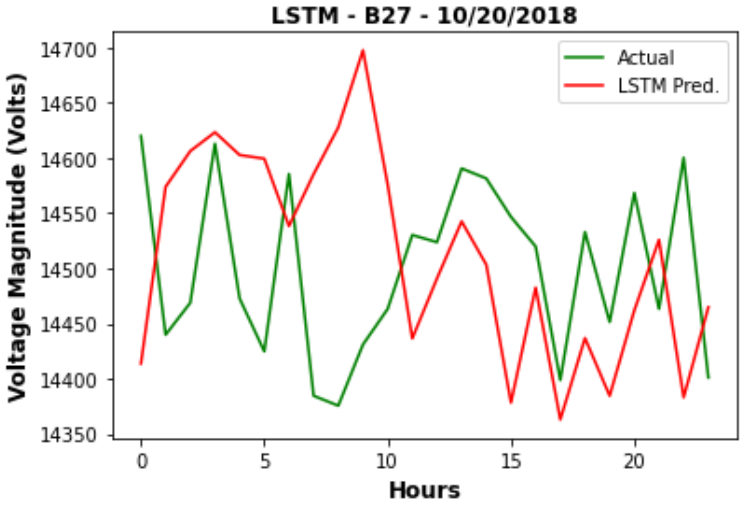


Figure 29 - Actual and Predicted Voltage Magnitude at Location B27 (Unoptimized LSTM)

Figure 30 presents the actual and predicted voltage phase angle at location B27 (Figure 7) for a 24 hour period utilizing an "unoptimized" LSTM model. Note that the prediction of voltage phase angle made by the unoptimized LSTM follows they dynamics of the actual time series better than the unoptimized CNN in Figure 25.

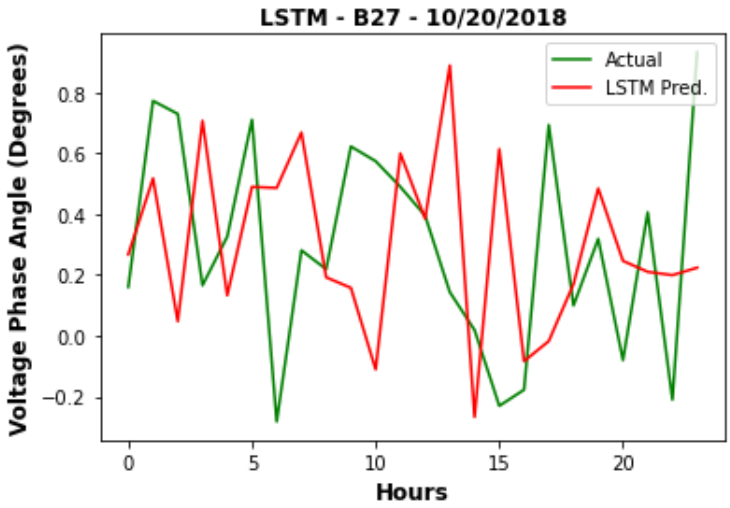


Figure 30 - Actual and Predicted Voltage Phase Angle at Location B27 (Unoptimized LSTM)

The following observations can be made regarding the unoptimized LSTM models applied to DSSF with a 24 hour forecast horizon:

- Real Power: The predicted time-series has a noticeable "delay" or "lag" relative to the actual time-series.
- Reactive Power: The predicted time-series does not exhibit the "flat" or "constant" waveform noted for the CNN model.

- Voltage Magnitude: The predicted time-series exhibits some of the same characteristics as the CNN model for the given forecast horizon.
- Voltage Phase Angle: The predicted time-series appears to perform slightly better than the unoptimized CNN model for the given forecast horizon.

LSTM models have gained considerable attention in the literature for their ability to "learn" and "remember" key "features" of time-series data. Thus, there is promise that for time-series forecasts of increasing length that LSTM models will be less susceptible to vanishing and exploding gradients exhibited by MLPs and CNNs used for the same purpose.

CHAPTER 7. A COMPARISON OF AUTO-REGRESSIVE MODELS AND CONVOLUTIONAL NEURAL NETWORKS FOR POWER DISTRIBUTION SYSTEM TIME-SERIES FORECASTING

This chapter presents a subset of the continued research into the application of deep learning methods to power system distribution state forecasting. An investigation of power distribution time series forecasting utilizing a "classical statistical" method as well as a fully data-driven "deep learning" method will be presented. In particular, autoregressive integrated moving average (ARIMA) will be considered as well as convolutional neural networks (CNN) for time-series forecasting. The IEEE standard 34-bus test system considered previously is used to demonstrate the proposed statistical and deep learning methods and their effectiveness for potential application to demand forecasting, distribution system state estimation (DSSE), and distribution system state forecasting (DSSF). The methods and results presented in this chapter are based on an ad-hoc selection of hyperparameters in order to demonstrate what is possible in terms of

readily applying both conventional and deep learning techniques to time series forecasting. Optimization will be left to subsequent research. Results of the performance of the ARIMA and CNN models will include plots of actual versus predicted real power, reactive power, voltage magnitude and voltage phase angle as well as the average root mean square error of the time series predictions of these quantities at various locations of the test distribution system over a 24-hour forecast horizon. The results presented in this chapter will contribute to the ongoing research into the application of machine learning in general and deep learning in particular to power distribution system state estimation and forecasting.

7.1 ARIMA Model

ARIMA stands for *Auto-Regressive Integrated Moving Average*. The model is statistically based and is defined by the following components:

- AR: Auto-regression. Determination of the relation of the data to itself. The relationship of observations to previous observations.
- I: Integrated. Utilization of differencing to make the time series stationary.
- MA: Moving Average. Application of a moving average to lagged observations to determine the dependency between an observation and residual error.

The standard notation used for this model type is *ARIMA(p,d,q)* where each of the hyperparameters are defined as follows:

- p: The number of lag observations included in the model, also called the lag order.

- d: The number of times that the raw observations are differenced, also called the degree of differencing.
- q: The size of the moving average window, also called the order of moving average.

7.2 CNN Model

An introduction to CNN model architecture and hyperparameters is presented in section 5.3.

7.3 ARIMA Model Implementation

The implementation of the ARIMA model was done in Python 3.9. A multi-variate time series consisting of all of the power and voltage measurements discussed in CHAPTER 3 was available for the ARIMA model. For purposes of simplicity, only the measurement points at locations B1 (substation), B18 and B27 in Figure 7 were considered. Also, only a single phase (phase a) was considered for each measurement point. Thus, separate univariate time series for the real power (P), reactive power (Q), voltage magnitude (V_mag) and voltage phase angle (V_phase) were considered.

Each of these time series was split into 80% for training and 20% for testing. An ARIMA model with hyperparameters p = 1, d = 1 and q = 0 was selected ad hoc. Thus, the model featured first order auto regressive terms, first order nonseasonal differencing and no lagged forecast errors were used in the prediction equation.

The model was then used in a walk forward validation loop to forecast the next 24 hours beyond the endpoint selected for the training dataset for each univariate time series (P, Q, V_mag, and V_phase) at each bus location (B1, B18, and B27).

7.4 CNN Model Implementation

The implementation of the CNN model was done in Python 3.9. The same datasets, measurement points, and training-testing split for the ARIMA Model in section 7.3 were also used for the CNN model.

The unoptimized CNN hyperparameter settings are the same as given in Table 29

7.5 Actual Versus Predicted P, Q, V_mag, V_phase Plots at B1

Figure 31 presents the actual and predicted real power at the substation (B1 in Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. Note that both models produce a time series prediction that follow the dynamics of the actual time-series, however the ARIMA model appears to outperform the CNN model over the forecast horizon considered.

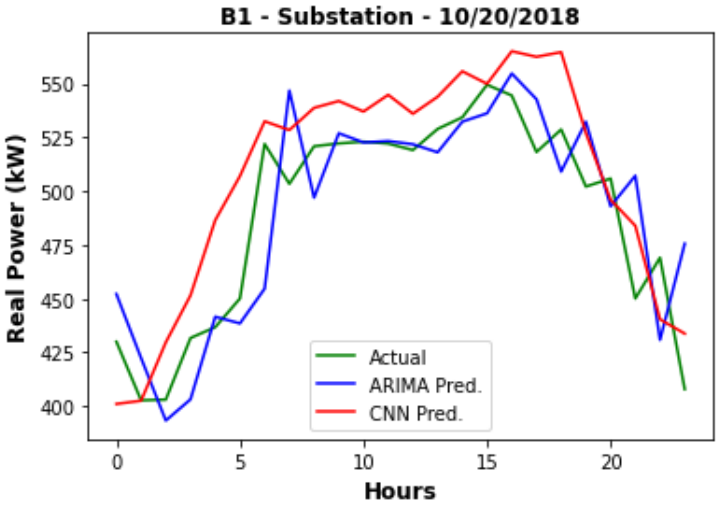


Figure 31 - Actual versus Predicted Real Power at B1

Figure 32 presents the actual and predicted reactive power at the substation (B1 in Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. Note that the unoptimized CNN model exhibits a "flat" response indicative of underfitting. The ARIMA appears to outperform the CNN model over the forecast horizon considered, however, as shown in Table 34, the RMSE of the CNN model is less than that of the ARIMA model. This scenario suggests that evaluation of overall "performance" of time-series predictions include multiple figures of merit.

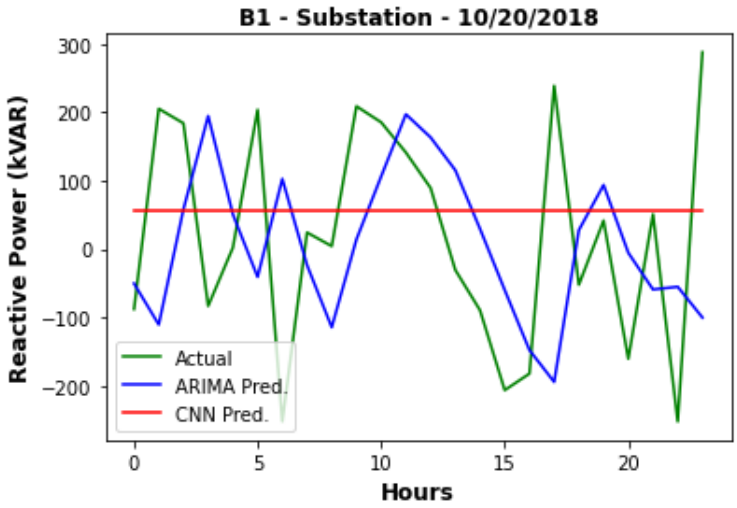


Figure 32 - Actual versus Predicted Reactive Power at B1

Figure 33 presents the actual and predicted voltage magnitude at the substation (B1 in Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. Note that the ARIMA model follows the actual time series to such a close degree that there appears to be no difference in the ARIMA prediction and the actual time series based upon the scale presented.

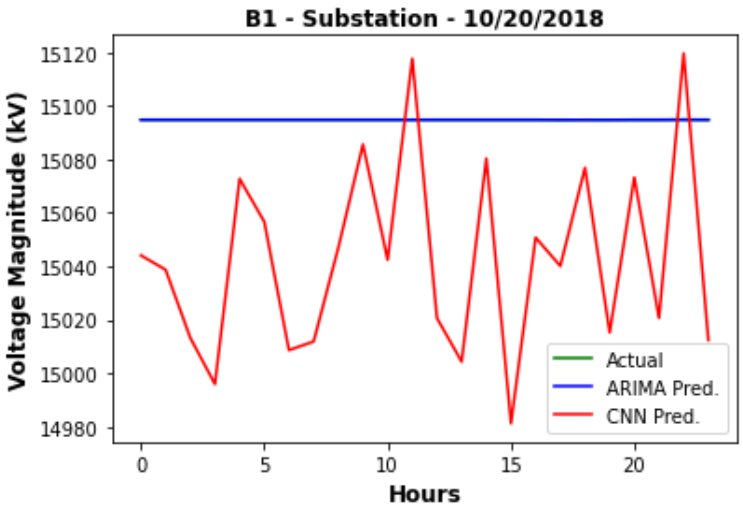


Figure 33 - Actual versus Predicted Voltage Magnitude at B1

Figure 34 presents on a different scale, the actual and predicted voltage magnitude at the substation (B1 in Figure 7) for a 24 hour period using an unoptimized ARIMA model. Note that the actual voltage magnitude at the substation changes very little over the forecast horizon and the ARIMA model can track the dynamics much better than the CNN model. The RMSE values for the ARIMA and CNN models in Table 34 this result.

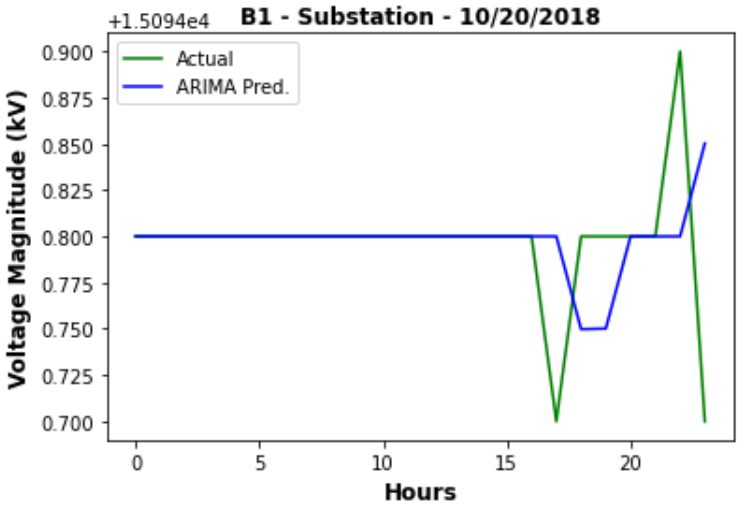


Figure 34 - Actual versus ARIMA Prediction of Voltage Magnitude at B1

Figure 35 presents the actual and predicted voltage phase angle at the substation (B1 in Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. Note that the actual voltage phase angle at the substation changes very little over the forecast horizon. Table 34 shows that the RMSE for the CNN model is significantly lower than the ARIMA model.

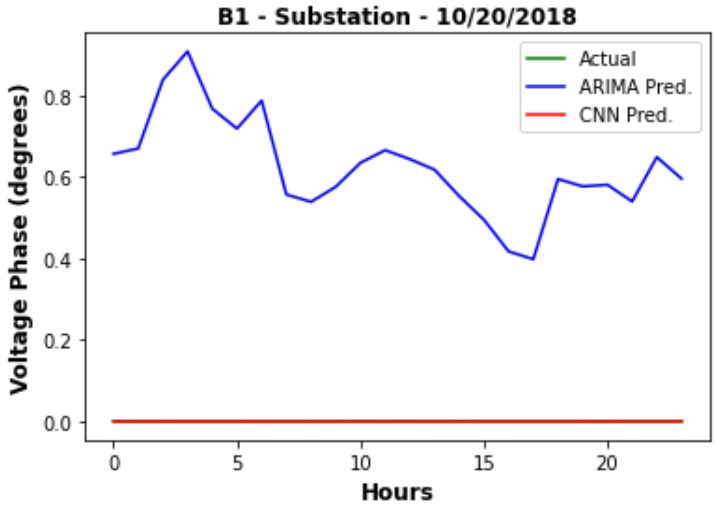


Figure 35 - Actual versus Predicted Voltage Phase Angle at B1

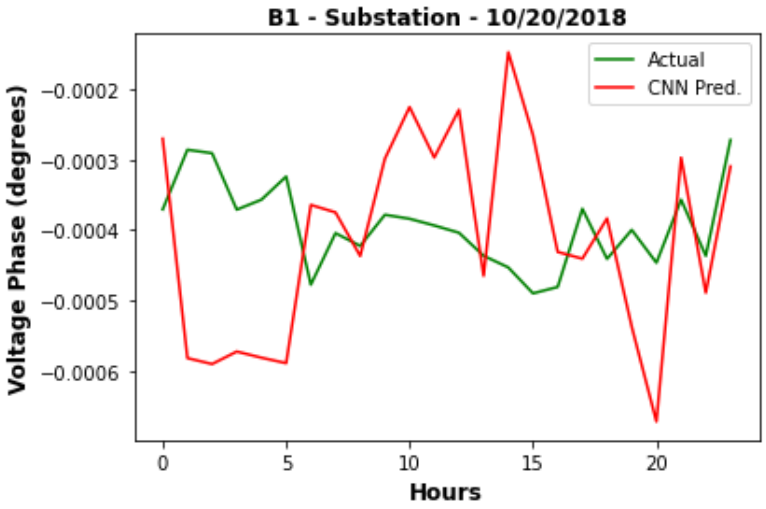


Figure 36 – Actual versus CNN Predicted Voltage Phase Angle at B1

7.6 Actual Versus Predicted P, Q, V_mag, V_phase Plots at B18

Figure 37 presents the actual and predicted real power at the substation (Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. Note that both models produce

a time series prediction that follow the dynamics of the actual time-series, however the ARIMA model appears to outperform the CNN model over the forecast horizon considered.

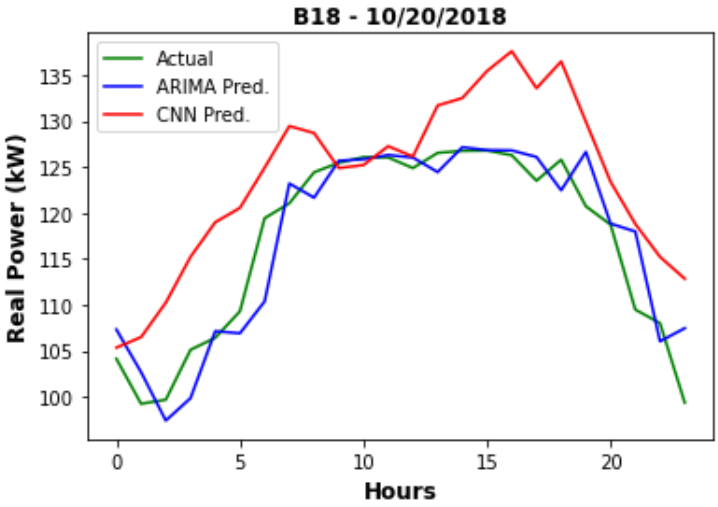


Figure 37 - Actual versus Predicted Real Power at B18

Figure 38 presents the actual and predicted reactive power at location B18 (Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. Note that the unoptimized CNN model exhibits a prediction that aligns with the approximate average of the actual time series. At this location, the ARIMA model does not follow the actual time series as well as the CNN model.

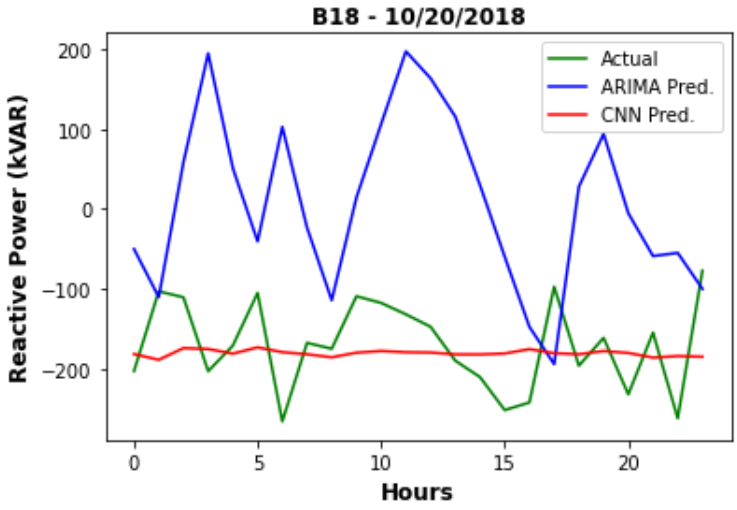


Figure 38 - Actual versus Predicted Reactive Power at B18

Figure 39 presents the actual and predicted voltage magnitude at B18 (Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. In this case, both models are responsive to the dynamics of the actual time series, however both models would benefit from hyperparameter optimization.

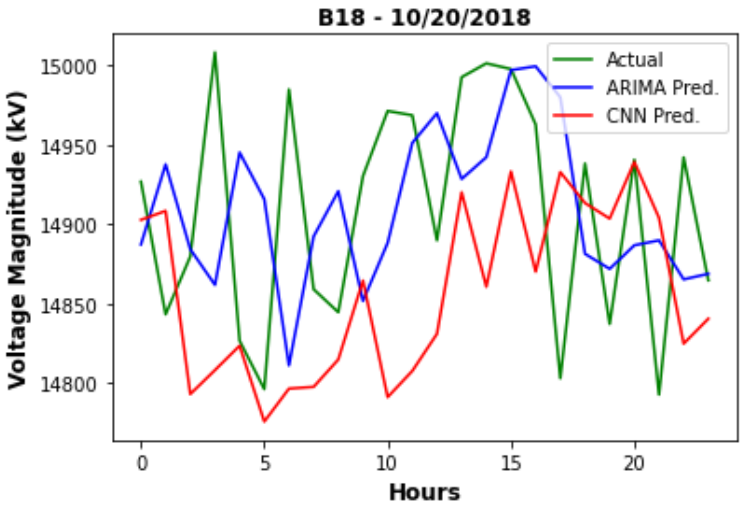


Figure 39 - Actual versus Predicted Voltage Magnitude at B18

Figure 40 presents the actual and predicted voltage phase angle at location B18 (Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. In this case, the unoptimized CNN model again follows the approximate average of the actual time series. According to Table 35, the unoptimized CNN model presents a slightly lower RMSE than the ARIMA model.

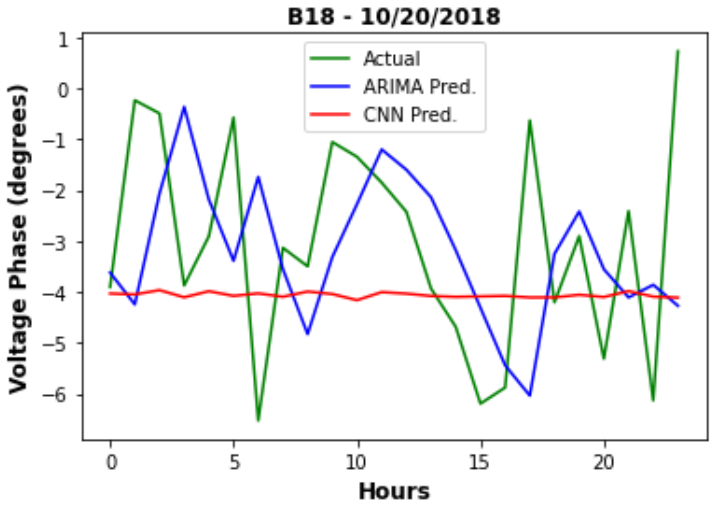


Figure 40 - Actual versus Predicted Voltage Phase Angle at B18

7.7 Actual Versus Predicted P, Q, V_mag, V_phase Plots at B27

Figure 41 presents the actual and predicted real power at location B27 (Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. Note that both models produce a time series prediction that follow the dynamics of the actual time-series.

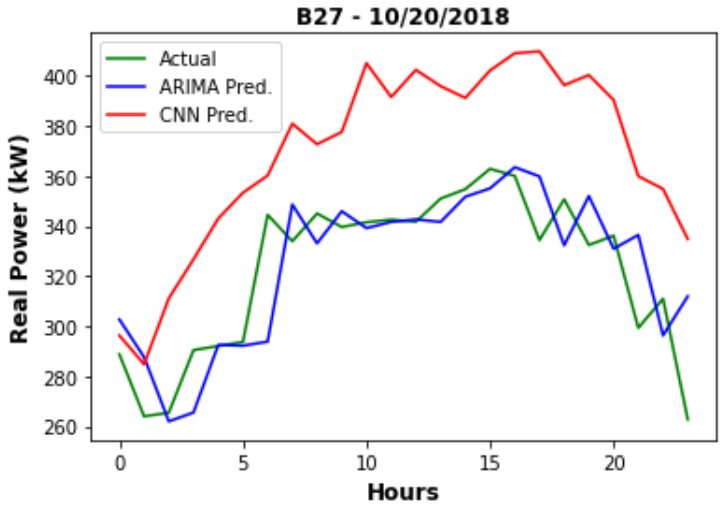


Figure 41 - Actual versus Predicted Real Power at B27

Figure 42 presents the actual and predicted reactive power location B27 (Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models. Note that the unoptimized CNN model exhibits a "flat" response indicative of underfitting. The ARIMA model outperforms the CNN model over the forecast horizon considered.

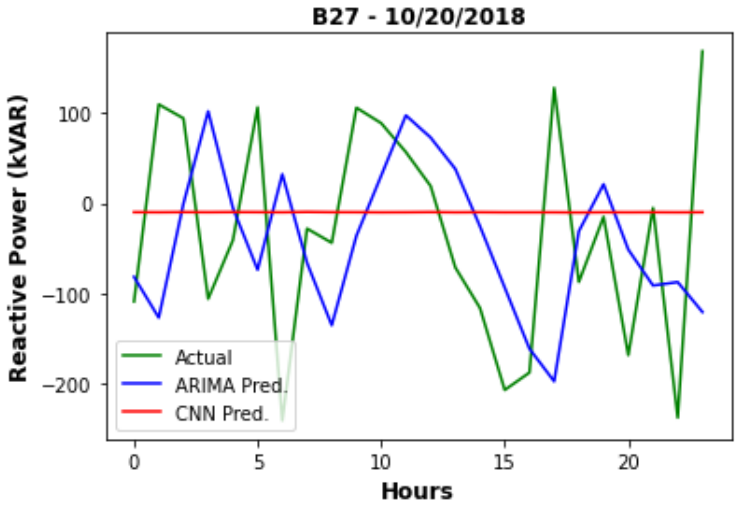


Figure 42 - Actual versus Predicted Reactive Power at B27

Figure 43 presents the actual and predicted voltage magnitude location B27 (Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models.

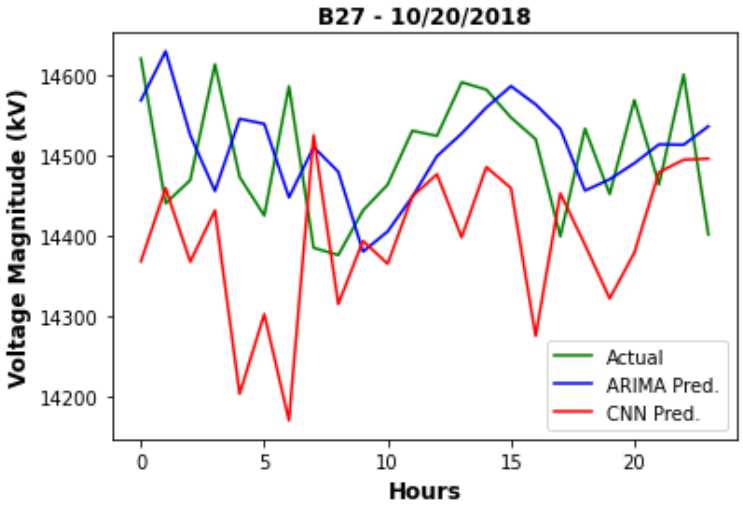


Figure 43 - Actual versus Predicted Voltage Magnitude at B27

Figure 44 presents the actual and predicted voltage phase angle at location B27 (Figure 7) for a 24 hour period using unoptimized ARIMA and CNN models.

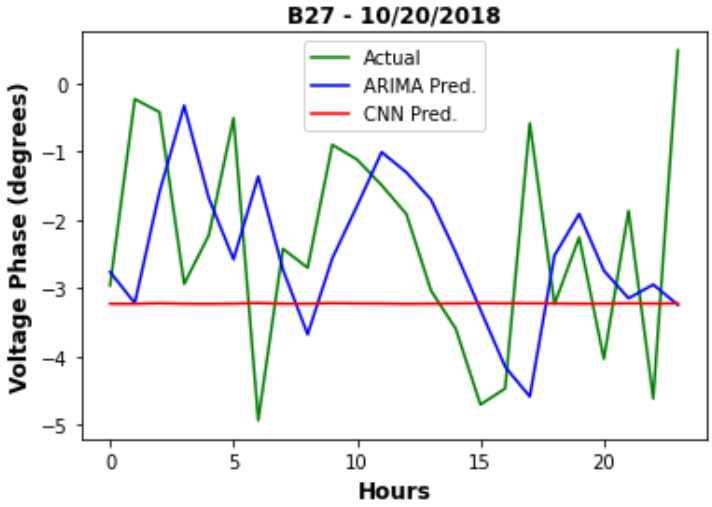


Figure 44 - Actual versus Predicted Voltage Phase Angle at B27

Table 34 - RMSE for Predicted Values at B1 for ARIMA and CNN Models

	P (kW)	Q (kVAR)	V_mag (V)	V_phase (degrees)
ARIMA	29.3809441	196.915513	0.044514	0.635800
CNN	78.454014	164.142076	63.174731	0.000169

Table 35 - RMSE for Predicted Values at B18 for ARIMA and CNN Models

	P (kW)	Q (kVAR)	V_mag (V)	V_phase (degrees)
ARIMA	3.849720	215.928072	86.691458	2.474801
CNN	8.112244	164.142076	101.198798	2.277744

Table 36 - RMSE for Predicted Values at B27 for ARIMA and CNN Models

	P (kW)	Q (kVAR)	V_mag (V)	V_phase (degrees)
ARIMA	20.360220	148.037981	93.571318	2.474801
CNN	49.279813	122.617277	161.039947	1.751593

Table 37 presents the RMSE and execution time for predicted values at B27 for the unoptimized ARIMA, CNN and LSTMs models presented in this research. Note that the RMSE values for each model type are comparable, however the execution time for the time-series prediction is lowest for the CNN model and highest for the LSTM models.

Table 37 - RMSE and Execution Time for Predicted Values at B27 for ARIMA, CNN and LSTM Models

		P (kW)	Q (kVAR)	V_mag (volts)	V_phase (degrees)
ARIMA	**RMSE**	20.360220	148.037981	93.571318	2.474801
	Execution Time (seconds)	275.665	495.134	477.426	486.234
CNN	**RMSE**	49.279813	122.617277	161.039947	1.751593
	Execution Time (seconds)	86.670	86.637	87.372	79.328
LSTM	**RMSE**	61.211237	131.694421	119.390719	6.791523
	Execution Time (seconds)	1059.255	1051.643	918.478	1167.424

CHAPTER 8. RESEARCH CONCLUSION

Chapter 1 of this dissertation introduces the purpose and significance of the research. Chapter 2 presents the background and related work. A key takeaway from this chapter is that conventional (analytical) approaches are not sufficient for power distribution system state estimation and state forecasting and that application of data driven approaches involving deep learning models may help overcome limitations of earlier methods. Chapter 3 presents distribution system state estimation (DSSE) with multilayer perceptron models (MLPs). The main takeaway from the chapter is a demonstration of the process of applying MLP models to power distribution system state estimation along with a structured methodology of selecting hyperparameters for the same models as an improvement over an ad-hoc approach. Additional layers may also prove

counterproductive in terms of higher root mean square error and additional training execution time. Another key result is that Bayesian Optimization with Gaussian Processes will greatly reduce the probability of selecting values that will not produce the maximum of the original objective function. Chapter 4 presents full distribution system state estimation with optimized MLP models. The main takeaway from Chapter 4 is that an improved workflow and data pipeline enables full system state estimation, that is not limited by the available monitors placed within the power simulator software. The improved workflow enables the possibility of state estimation being applied to much larger distribution systems. Application and validation of the improved workflow to larger distributions systems is left to future research.

Chapter 5 presented distribution system state forecasting (DSSF) with convolutional neural network (CNN) models. The main takeaway from Chapter 5 is that CNNs may be applied to areas other than object recognition and vision systems to perform power system state forecasting by learning patterns, seasonality, and trends of a time-series.

Chapter 6 presents distribution system state forecasting (DSSF) with long short-term memory models (LSTMs). The main takeaway from Chapter 6 is that LSTMs may be applied to areas other than speech recognition, language translation and image captioning to perform power system state forecasting by learning the autocorrelation of long sequence time-series data.

Chapter 7 presents a comparison of auto-regressive models and convolutional neural networks for power distribution system time-series forecasting. The main takeaway from Chapter 7 is that for univariate time-series prediction, classical and deep learning methods should be considered as viable options. Although left for future research, for

multivariate time-series forecasting, classical methods such as ARIMA are quite limited while deep learning models such as CNNs and LSTMs are viable options.

REFERENCES

[1] Soltan, S., Mittal, P., and Poor, H. V. (2018). Bayesian regression for robust power grid state estimation following a cyber-physical attack. In: 2018 IEEE Power and Energy Society General Meeting (PESGM), p. 1–5. doi: 10.1109/PESGM.2018.8586142

[2] Schweppe, Fred C, and Wildes, J. Power System Static-State Estimation, Part I: Exact Model. IEEE Transactions on Power Apparatus and Systems PAS-89.1 (1970): 120-25.

[3] Fan, W., and Liao, Y. (2018). Fault Identification and Location for Distribution Network with Distributed Generations. Int. J. Emer. Electr. Power Syst. 19, 1–13. doi: 10.1515/ijeeps-2018-0048

[4] Fan, W., and Liao, Y. (2019). Microgrid Operation Optimization Considering Storage Devices, Electricity Transactions and Reserve. Int. J. Emerging Electr. Power Syst. 20, 1–17. doi: 10.1515/ijeeps-2019-0003

[5] Fan, W. (2019). Advanced Fault Area Identification and Fault Location for Transmission and Distribution Systems, Dissertation. USA.

[6] Fan,W., Hossan,M. S., Zheng, H., Cook, A., Zaid, S., and Fard, S. A. (2021). A CVR On/Off Status Detection Algorithm for Measurement and Verification. In: 2021 IEEE Power and Energy Society Innovative Smart Grid Technologies Conference (ISGT),Washington, DC. USA. p. 1–5. doi: 10.1109/ISGT49243.2021.9372257

[7] Schweppe, Fred C, and Rom, Douglas B. "Power System Static-State Estimation, Part II: Approximate Model." IEEE Transactions on Power Apparatus and Systems PAS-89.1 (1970): 125-30.

[8] Schweppe, Fred C. "Power System Static-State Estimation, Part III: Implementation." IEEE Transactions on Power Apparatus and Systems PAS-89.1 (1970): 130-35.

[9] Shivakumar, N.R, and Jain, A. "A Review of Power System Dynamic State Estimation Techniques." 2008 Joint International Conference on Power System Technology and IEEE Power India Conference (2008): 1-6.

[10] Krumpholz, G. R, Clements, K. A, and Davis, P. W. "Power System Observability: A Practical Algorithm Using Network Topology." IEEE Transactions on Power Apparatus and Systems PAS-99.4 (1980): 1534-542.

[11] Urban Kuhar, Gregor Kosec, and Ales Svigelj, Observability in Distribution Systems, January 2020 – Full Textbook

[12] Electric Power Research Institute (EPRI), Electrical Power System Resiliency: Challenges and Opportunities, EPRI White Papers, February 2016

[13] Kaveh Dehghanpour, Zhaoyu Wang , Jianhui Wang ,Yuxuan Yuan and Fankun Bu, A Survey on State Estimation Techniques and Challenges in Smart Distribution Systems, IEEE Transactions on Smart Grid, Vol. 10, No. 2, March 2019

[14] Baran, M.E, and Kelley, A.W. "State Estimation for Real-time Monitoring of Distribution Systems." IEEE Transactions on Power Systems 9.3 (1994): 1601-609.

[15] Yao, Yiyun, Liu, Xuan, and Li, Zuyi. "Robust Measurement Placement for Distribution System State Estimation." IEEE Transactions on Sustainable Energy 10.1 (2019): 364-74. Web.

[16] Haughton, Daniel A, and Heydt, G. T. "A Linear State Estimation Formulation for Smart Distribution Systems." IEEE Transactions on Power Systems 28.2 (2013): 1187-195.

[17] Youman Deng, Ying He, and Boming Zhang. "A Branch-estimation-based State Estimation Method for Radial Distribution Systems." IEEE Transactions on Power Delivery 17.4 (2002): 1057-062.

[18] Mestav, Kursat Rasim, Luengo-Rozas, Jaime, and Tong, Lang. "State Estimation for Unobservable Distribution Systems via Deep Neural Networks." 2018 IEEE Power & Energy Society General Meeting (PESGM) (2018): 1-5.

[19] K. R. Mestav, J. Luengo-Rozas and L. Tong, "Bayesian State Estimation for Unobservable Distribution Systems via Deep Learning," in IEEE Transactions on Power Systems, vol. 34, no. 6, pp. 4910-4920, Nov. 2019, doi: 10.1109/TPWRS.2019.2919157.

[20] K. R. Mestav, L. Tong, " Learning the Unobservable: High-Resolution State Estimation via Deep Learning," 2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton) Allerton Park and Retreat Center Monticello, IL, USA, September 24-27, 2019

[21] Cao, Zhiyuan, Wang, Yubo, Chu, Chi-Cheng, and Gadh, Rajit. "Scalable Distribution Systems State Estimation Using Long Short-Term Memory Networks as Surrogates." IEEE Access 8 (2020): 23359-3368.

[22] Zhang, Ying, Wang, Jianhui, and Chen, Bo. "Detecting False Data Injection Attacks in Smart Grids: A Semi-Supervised Deep Learning Approach."
IEEE Transactions on Smart Grid 12.1 (2021): 623-34.

[23] Cao, Z. (2020). Data Driven State Estimation in Distribution Systems. UCLA. ProQuest ID: Cao_ucla_0031D_19094. Merritt ID: ark:/13030/m5n358fb.
Retrieved from https://escholarship.org/uc/item/2p33m7wg

[24] Song, Jianhan, Dall'Anese, Emiliano, Simonetto, Andrea, and Zhu, Hao. "Dynamic Distribution State Estimation Using Synchrophasor Data." IEEE
Transactions on Smart Grid 11.1 (2020): 821-31.

[25] Tian C, Ma J, Zhang C, Zhan P. A Deep Neural Network Model for Short-Term Load Forecast Based on Long Short-Term Memory Network and Convolutional
Neural Network. Energies. 2018; 11(12):3493. https://doi.org/10.3390/en11123493

[26] ERCOT Historical Load Data, https://www.ercot.com/gridinfo/load/load_hist

[27] Dallas/Fort Worth International Airport Historical Weather,
https://www.meteoblue.com/en/weather/archive/export/dallas%2ffort-worth-international-airport_united-states_4684943

[28] Frazier, Peter I. A Tutorial on Bayesian Optimization, arXiv:1807.02811v1
July 10, 2018

[29] Brownlee, Jason. Deep Learning with Python, Machine Learning Mastery, 2021, pp. 107-108

[30] Adams, Ryan P. A Tutorial on Bayesian Optimization. School of Engineering and Applied Sciences Harvard University,
https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf, 2018

[31] Dugan, Roger. IEEE 34 Node Test Feeder. IEEE Power Engineering Society, (2004): pg. 2

[32] O'Connor, J.J. and Robertson, E.F., Thomas Bayes, Available from https://mathshistory.st-andrews.ac.uk/Biographies/Bayes/, accessed April 03, 2022.

[33] Brownlee, Jason. A Gentle Introduction to Bayes Theorem for Machine Learning, Machine Learning Mastery, Available from https://machinelearningmastery.com/bayes-theorem-for-machine-learning/, accessed April 03, 2022.

[34] Larson, J., Menickelly, M., & Wild, S. M. (2019). Derivative-free optimization methods. ArXiv. https://doi.org/10.1017/S0962492919000060

[35] Vink, Ritchie. Algorithm Breakdown: Bayesian Optimization, Available from https://www.ritchievink.com/blog/2019/08/25/algorithm-breakdown-bayesian-optimization/, accessed April 03, 2022.

[36] Brownlee, Jason. How to Work Through a Time Series Forecast Project, Machine Learning Mastery, Available from https://machinelearningmastery.com/work-time-series-forecast-project/, accessed April 03, 2022.

[37] ADAM, https://keras.io/api/optimizers/adam/, Retrieved on 01/02/2023

[38] Brownlee, Jason. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, Machine Learning Mastery, Available from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, accessed April 03, 2022.

[39] Kingma, Diederik and Ba, Jimmy. Adam: A Method for Stochastic Optimization. Presented at 3rd International Conference for Learning Representations, San Diego, 2015, https://arxiv.org/abs/1412.6980

[40] C. E. Rasmussen & C. K. I. Williams, Gaussian Processes for Machine Learning, the MIT Press, 2006,
ISBN 026218253X. Massachusetts Institute of Technology.
www.GaussianProcess.org/gpml

[41] Wu, J., Poloczek, M., Wilson, A. G., Frazier, Peter I., Bayesian Optimization with Gradients, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

[42] Brochu, Eric et al. “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning.” ArXiv abs/1012.2599 (2010)

[43] OpenDSS, https://www.epri.com/pages/sa/opendss, Retrieved on 02/01/2023

[44] James Carmichael, Yuan Liao, Application of Deep Neural Networks to Distribution System State Estimation and Forecasting, Front. Sustain. Cities, https://doi.org/10.3389/frsc.2021.814037, January 07, 2022

[45] Y. Chen, Y. Tan and D. Deka, Is Machine Learning in Power Systems Vulnerable?, 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), 2018, pp. 1-6, doi: 10.1109/SmartGridComm.2018.8587547

[46] Synchrophasor System Benefits Fact Sheet, North American Synchrophasor Initiative (NASPI), https://www.naspi.org/, Retrieved on May 05, 2022

[47] Synchrophasor Monitoring for Distribution Systems: Technical Foundations and Applications, NASPI Distribution Task Team, January 2018

[48] Factors Affecting PMU Installation Costs, U.S. Department of Energy, https://www.energy.gov/oe/articles/factors-affecting-pmu-installation-costs-october-2014, Retrieved on 02/01/2023

[49] Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS
Petro Liashchynskyi, Pavlo Liashchynskyi, https://doi.org/10.48550/arXiv.1912.06059, Submitted on 12 Dec 2019

[50] Frazier, Peter I. A Tutorial on Bayesian Optimization, https://doi.org/10.48550/arXiv.1807.02811, July 10, 2018

[51] Snoek, J., Larochelle, H., Adams, Ryan P., Practical Bayesian Optimization of Machine Learning Algorithms, https://doi.org/10.48550/arXiv.1206.2944, Submitted on 13 Jun 2012

[52] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay; Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research 12(85):2825−2830, 2011.

[53] OpenDSS, https://www.epri.com/pages/sa/opendss, Retrieved on 02/01/2023

[54] M. E. Baran and A. W. Kelley, "A branch-current-based state estimation method for distribution systems," in IEEE Transactions on Power Systems, vol. 10, no. 1, pp. 483-491, Feb. 1995, doi: 10.1109/59.373974.

[55] sklearn https://scikit-learn.org/stable/, Retrieved on 02/01/2023

[56] Pandas Dataframes, https://pandas.pydata.org/docs/getting_started/index.html, Retrieved on 02/01/2023

[57] Tensorflow, https://www.tensorflow.org/, Retrieved on 02/01/2023

[58] Keras, https://keras.io/, Retrieved on 02/01/2023

[59] Ruben Martinez-Cantin, BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits. Journal of Machine Learning Research, 15(Nov):3735--3739, 2014.

[60] Paulo Radatz, Enio Viana, Rodolfo Pilar Londero. py_dss_interface, https://py-dss-interface.readthedocs.io/en/latest/readme.html, Retrieved on 02/01/2023

[61] Numpy, https://numpy.org/, Retrieved on 02/01/2023

[62] Python, https://www.python.org/, Retrieved on 02/01/2023

[63] Chris Asbery and Yuan Liao, "Fault identification on electrical transmission lines using artificial neural networks," Electric Power Components and Systems, vol. 49, no. 13-14, 2021.

[64] Chris Asbery, Yuan Liao, "Electric transmission system fault identification using modular artificial neural networks for single transmission lines," Power System Conference, Clemson, SC, March 10-13, 2020.

[65] Chris Asbery, Yuan Liao, "Electric transmission system fault identification using artificial neural networks," International Energy & Sustainability Conference 2019, Farmingdale, NY, USA, October 17 - 18, 2019.

[66] Wesley Fluty, Yuan Liao, "Electric transmission fault location techniques using traveling wave method and discrete wavelet transform," Power System Conference, Clemson, SC, March 10-13, 2020.

[67] "IEEE/IEC International Standard - Measuring relays and protection equipment - Part 118-1: Synchrophasor for power systems - Measurements," in IEC/IEEE 60255-118-1:2018 , vol., no., pp.1-78, 19 Dec. 2018, doi: 10.1109/IEEESTD.2018.8577045.

[68] sklearn.preproccessing.MinMaxScaler, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html , Retrieved on 03/28/2023

[69] Jiaxiong Chen, 2013, Dissertation, University of Kentucky, Power System State Estimation Using Phasor Measurement Units.

[70] Jiaxiong Chen and Yuan Liao, "Investigation of WLS state estimation convergence under topology errors and load increment," International Journal of Automation and Logistics, vol. 1, no. 1, pp. 47-60, 2013.

[71] Jiaxiong Chen, Yuan Liao, Bei Gou, “Study of WLS state estimation convergence characteristics under topology errors,” SoutheastCon, Jacksonville, Florida, April 04-07, 2013.

VITA

1. Educational Institutions
    - August 2014 – Present
        - PhD. Student
        - Department of Electrical and Computer Engineering, University of Kentucky, Lexington, Kentucky, USA
    - December 1997 – August 1999
        - Master of Engineering in Electrical Engineering
        - Department of Electrical Engineering, University of Louisville, Louisville, Kentucky, USA
    - August 1991 – December 1997
        - Bachelor of Science in Electrical Engineering
        - Department of Electrical Engineering, University of Louisville, Louisville, Kentucky, USA
2. Professional Positions
    - January 2014 – Present
        - Adjunct Online Faculty
        - School of Science, Technology, Engineering, and Mathematics
    - February 2018 – January 2020
        - Senior Data Scientist
        - Humana, Inc., Louisville, Kentucky, USA
    - August 2007 – February 2018
        - Program Manager
        - Humana, Inc., Louisville, Kentucky, USA
3. Scholastic and professional honors
    - SMART Scholarship – April 2022
    - Tennessee Valley Authority Fellowship – August 2014
    - Project Management Professional Certification (PMP) – March 2010
    - Professional Engineering License, State of Kentucky, December 2003
4. Professional publications
    - James Carmichael and Yuan Liao, "Application of Deep Neural Networks to Distribution System State Estimation and State Forecasting", Frontiers In Sustainable Cities - Smart Technologies and Cities · Jan 7, 2022, https://doi.org/10.3389/frsc.2021.814037
5. Student: James P. Carmichael