A Machine Learning Approach for Solving AC Optimal Power Flows

SAMSI–IMSM 2019

Deepjyoti Ghosh[1] Derek Hanely[2] Harley Hanes[3]
Kapila Kottegoda[4] Jenna McDanold[5] Phong Nguyen[6]
Yuqi Su[7]

Industry Scientist: Dr. Adam Attarian[8]
Faculty Mentor: Dr. Hien Tran[9]

July 24, 2019

Abstract

The power grid is currently optimized using a MATLAB program that employs traditional methods for solving optimization problems with nonlinear constraints. These traditional methods are slow, require very close initial guesses, and cannot handle extreme deviations from the norm. We present two types of machine learning models for the IEEE's predefined 30-bus power flow system in this paper. The first is a neural network algorithm with three hidden layers, 100 nodes in each layer, trained on over 20,000 samples with an accuracy rating of 99% that predicts valid generator configurations 80% of the time. The other model is a decision tree regression algorithm (XGBoost) that achieved an 88% fit to the data with 1000 decision trees and a depth of three levels per tree. Future work will include generalizing the models to higher nodal systems, investigating higher estimators for the regression tree model, employing optimality classifications for both models, and analysis of extreme deviations from the norm within the systems.

[1]Department of Mathematics, University of Houston
[2]Department of Mathematics, University of Kentucky
[3]Center for Computational Science, Tulane University
[4]Department of Mathematics, Kansas State University
[5]School of Mathematical Sciences, Rochester Institute of Technology
[6]Department of Civil, Arch. and Env. Engineering, University of Texas at Austin
[7]Operations Research Program, North Carolina State University
[8]Applied Statistics and Computational Modeling Group, Pacific Northwest National Lab
[9]Department of Mathematics, North Carolina State University

Acknowledgements

We would like to thank the following individuals for their tremendous help and guidance with this project:

Chris Oehmen, Pacific Northwest National Lab

Mark Rice, Pacific Northwest National Lab

Yahe Yu, North Carolina State University

Contents

1 Introduction

Electrical power flow networks across the world require regular scrutiny for efficiency and cost-effective production. Millions of people rely on the correct operation of a single grid, e.g., over 500 million people rely on a single grid spanning Europe, Turkey, and parts of North Africa [1]. Finding better solutions for optimizing this network has the potential to save millions of tax payers dollars a year in the U.S. alone [2]. This problem, originally formulated in 1962, is a nonlinearly constrained optimization problem that has traditionally been solved using numerical methods. However, with the changing technology of the last couple decades, new and improved methods for solving this problem have emerged [2]. With the recent surge in machine learning algorithms, machine learning offers a potentially faster method and could make solving this problem significantly easier for grid operators around the world.

Each power system can be modeled as a network of buses—the nodes of the grid topology—and the branches between them—the edges. A subset of those buses comprise the set of generators, which are the locations at which power is generated, stored, or balanced within the system. Each bus serves a particular area and has a particular amount of real and reactive power within the system. The real power is the active power, or the power that is distributed throughout the system. The reactive power moves between the source and the load, balancing the system. Reactive power is not usable power.

MATPOWER is a MATLAB program designed to solve these problems through a variety of numerical methods. The program takes as inputs the network topology, limits for each bus and node, and power requirements for each bus, and outputs the settings for the generators that will optimize the system's power flow and minimize cost. While very useful, this program has limitations for solving this system based on the internal workings of the program. The program relies on linearization methods that depend on convexity and good initial guesses, and doesn't account for extreme cases in which we do not have clear convexity nor do we have an educated guess for initial conditions. As a result, we seek a new approach for these optimization problems, that is, machine learning.

There are two stages within a machine learning algorithm using neural networks. The first stage is the training stage in which we use a training data set to "teach" the algorithm the patterns within the data. The training data set contains solutions for each of the elements in the set so that the algorithm can learn the best configurations for the algorithm to obtain the solutions given. The second stage of this process is the testing stage in which we have a second set of solved elements and we run the program for the inputs for this set and compare the computed solutions to the true solutions to test for the accuracy of the machine learning model.

During the training portion of this process, the algorithm begins with an input layer, which contains nodes or neurons that are each element of the training data set. Then the data passes through a particular number of hidden layers in which each data point goes through all of the nodes within each layer, multiplied by weighting values for each branch with a bias for each hidden layer. In the beginning of this process, the weighting values and biases are arbitrarily chosen. At the end of the process, there is an output layer that will be the solution for the data. This solution is compared to the true solution from the training data set, and then through back propagation, the weighting values and biases are adjusted.

The answer is then run back through the system a given amount of times (referred to as the number of epochs) and the answer is refined; the hope being that the answer will converge to the appropriate value.

This year at ICML (International Conference on Machine Learning), Neel Guha and his colleagues presented a neural network algorithm for tackling this problem [4]. Their work was one of the first attempts to train a neural network to recognize optimal configurations for the AC power flow problem. They used the existing technology to generate data for his training set and then designed a machine learning algorithm to accurately predict generator configurations for particular typical bus systems and their flows.

Our work involved modifying the neural network approach employed in [4] to include optimization of cost per megawatt hour and a more accurate predicting system. We present our findings to the IMSM workshop, July 2019.

2 Methods

2.1 Notation

For clarity in comparing our work to others, we adopt the following notation for the OPF problem.

$\mathbf{N}$	set of buses
$\mathbf{G}$	set of generators
n	total number of simulations
L	total number of branches
m	total number of decision trees in XGBoost
P_i^L	real power demand at bus i
Q_i^L	relative power demand at bus i
P_i^G	generator real power supply at bus i
Q_i^G	generator relative power supply at bus i
V_i^G	generator voltage magnitude at bus i
δ_i	voltage angle at bus i
$\tilde{V}_i$	voltage (complex) at bus i
E_i	real component of $\tilde{V}_i$
F_i	imaginary component of $\tilde{V}_i$
$\tilde{Y_{ik}}$	ik^{th} element (complex) of the bus admittance matrix
G_{ik}	real component of $\tilde{Y_{ik}}$
B_{ik}	imaginary component of $\tilde{Y_{ik}}$

Following the notation above, we have $\tilde{V}_i = E_i + iF_i$, and $\tilde{Y_{ik}} = G_{ik} + jB_{ik}$. Also, $V_i = \sqrt{E_i^2 + F_i^2}$ and $Y_{ik} = \sqrt{G_{ik}^2 + B_{ik}^2}$. Moreover, note that $\mathbf{G} \subseteq \mathbf{N}$.

2.2 Generated Data and Analysis

For this project, we began with two standard IEEE cases: a 30 bus system representing the US power grid in 1961, and a 300 bus system representing the US power grid in 1993.

To find various results, we perturbed the PD (real power output) and QD (reactive power output) values. To create the training set for our machine learning algorithms, we randomly sampled power inputs centered around the standard case according to a normal distribution with mean equal to the nominal value and standard deviation equal to 0.1 of the nominal value. Prior research on this topic used uniform distributions to perturb power demand parameters [4]. We chose a normal distribution because we wanted the probability of a given perturbation to be symmetrical around the base value. We also wanted as diverse a data set as possible and so did not want to set a limit on the maximum possible perturbation as would be required for a uniform distribution. We experimented with standard deviation values up to 50% but as the value increased, the amount of simulations that did not converge also increased. To maximize the amount of simulations that did converge for our training set, we therefore settled on 10%. Note also that since this perturbation is based on a percentage of the original value, any buses that were originally 0 remained at 0. We then evaluated each system with MATPOWER to find the optimal generator configuration. For each sampled input, MATPOWER would either converge to a solution, in which case it was labeled a successful simulation, or fail to converge, in which case it was labeled a failed simulation. The failure of a simulation may be due to the numerical solver being insufficient to find an existent solution or because the set parameters violated the constraints detailed in equation (1) below.

The default solver within the MATPOWER program uses an interior point method, and the two main functions are `runpf` (run power flow) or `runopf` (run optimized power flow). For our purposes, we used `runopf` because it optimizes the generator configurations in terms of the lowest cost whereas `runpf` only identifies the closest solution without regard to cost. The generalized equations and physical constraints of the system are given by equation (1) [4]. The objective function (1) is a polynomial function with variable P_i^G, (1a) and (1b) are power flow equations in polar coordinate system, and (1c)-(1f) represent feasible ranges of real power output, relative power output, voltage magnitude and voltage angle, respectively.

$$
\begin{aligned}
&\underset{P_i^G}{\text{minimize}} \quad \sum_{i \in G} C_i(P_i^G), && && (1)\\
&\text{subject to} \\
&P_i(V,\delta) = P_i^G - P_i^L, && \forall i \in \mathbf{N} && (1a)\\
&Q_i(V,\delta) = Q_i^G - Q_i^L, && \forall i \in \mathbf{N} && (1b)\\
&P_i^{G,\min} \le P_i^G \le P_i^{G,\max}, && \forall i \in \mathbf{G} && (1c)\\
&Q_i^{G,\min} \le Q_i^G \le Q_i^{G,\max}, && \forall i \in \mathbf{G} && (1d)\\
&V_i^{\min} \le V_i \le V_i^{\max}, && \forall i \in \mathbf{N} && (1e)\\
&\delta_i^{\min} \le \delta_i \le \delta_i^{\max}, && \forall i \in \mathbf{N} && (1f)
\end{aligned}
$$

For the AC power flow problem, the grid demand is given according to equation (2), and the corresponding optimal generator supply form corresponds to equation (3)

$$(P_i^L, Q_i^L)_{i=1}^n \quad \text{for bus } 1, 2, ..., N \tag{2}$$

$$(P_j^G, Q_j^G)_{j=1}^n \quad \text{for generator } 1, 2, ..., G \tag{3}$$

which results in a grid demand matrix with dimension $n \times 2N$, and grid control matrix with dimension $n \times 2G$ for each perturbation.

In AC power flow equations, voltage is expressed in polar coordinates and admittance is expressed in rectangular coordinates. Following the notation in §2.1, AC flow equations are as according to equation (4) [3].

$$\begin{aligned} P_i(V,\delta) &= V_i \sum_{i=1}^{N} V_k(G_{ik}\cos(\delta_i - \delta_k) + B_{ik}\sin(\delta_i - \delta_k)) \\ Q_i(V,\delta) &= V_i \sum_{i=1}^{N} V_k(G_{ik}\sin(\delta_i - \delta_k) - B_{ik}\sin(\delta_i - \delta_k)) \end{aligned} \tag{4}$$

2.2.1 30 Bus System

We generated 30,000 samples of IEEE's 30 bus system, with 20,797 being successful and 9,203 flagged as failures. Figure 1 illustrates how all of the PD-QD points are laid out throughout the plane. Using red for failed simulations and blue for successful simulations, we found the relationships between the PD and QD values were insignificant for most buses, but bus 8 appeared to be critical in how the relationships of the values determined the success or failure of the simulation. The circular shape on the top right portion of the graph relates to bus 8. Figures 2 and 3 illustrate how the failures and successes were related to the PD and QD values on each bus for all simulations. The original values for bus 8 were the highest among those within this bus system which we hypothesize is the reason bus 8 had a higher correlation with success or failure of a system. With a larger initial value, the percentage would be larger and therefore the perturbations would also be larger. Although this would account for the drastically different behavior of this bus in comparison with the other buses within the system, we do not have a good idea of why these original values were chosen to begin with, so it is difficult to make conjectures on why bus 8 would have so much more of an effect.

2.2.2 300 Bus System

For the more complex 300 bus system and due to the limited time frame of the workshop and computational power, we created only 25,000 simulated points with 20,434 successful simulations and nearly 5,000 failed simulations. Figure 4 shows the relationships of the PD and QD values for this system. Unlike the 30 bus system, there are negative values for power within this system. This is because the power described is directional and therefore a negative power flow indicates that the power is moving in a direction opposite to the main flow.

Again we see that generally there does not appear to be a correlation between the P and Q values and the success or failure of the optimization. The vertical line at the top left of the figure is for a particular bus that has very small P values and large Q values originally so its perturbation range is more oblong.

We used data from both the uniform perturbation and the normal perturbation within our attempts to make a neural network model for this system. The model did not appear

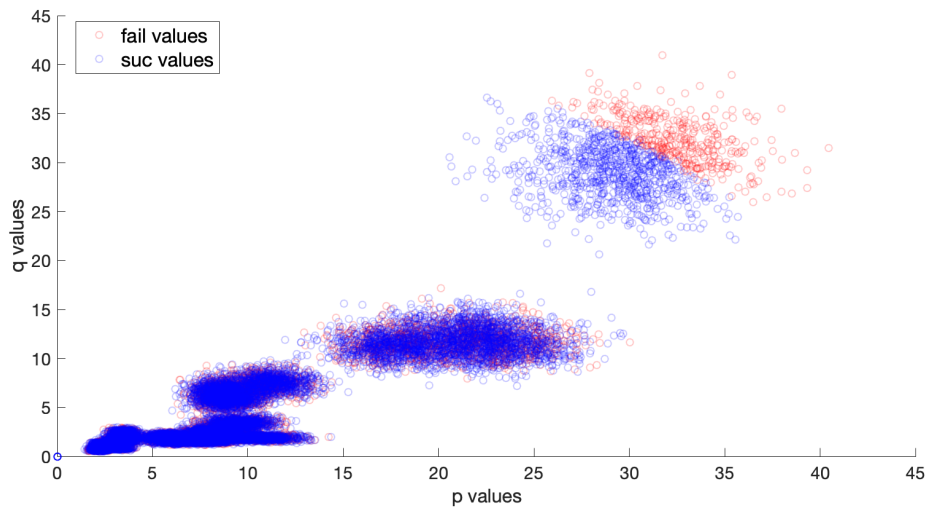


Figure 1: P versus Q values for all buses in the 30 bus system

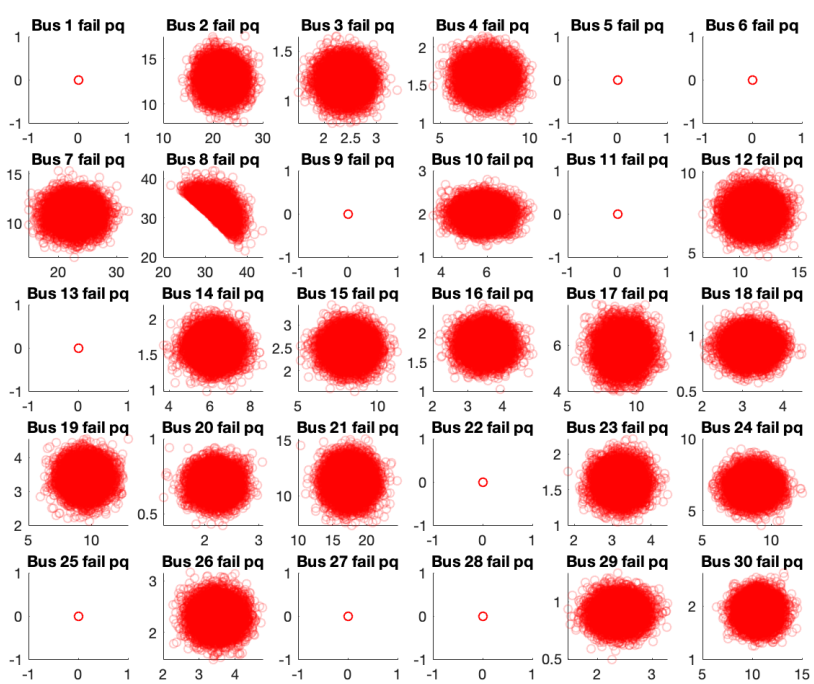


Figure 2: P versus Q values per bus for failed simulations

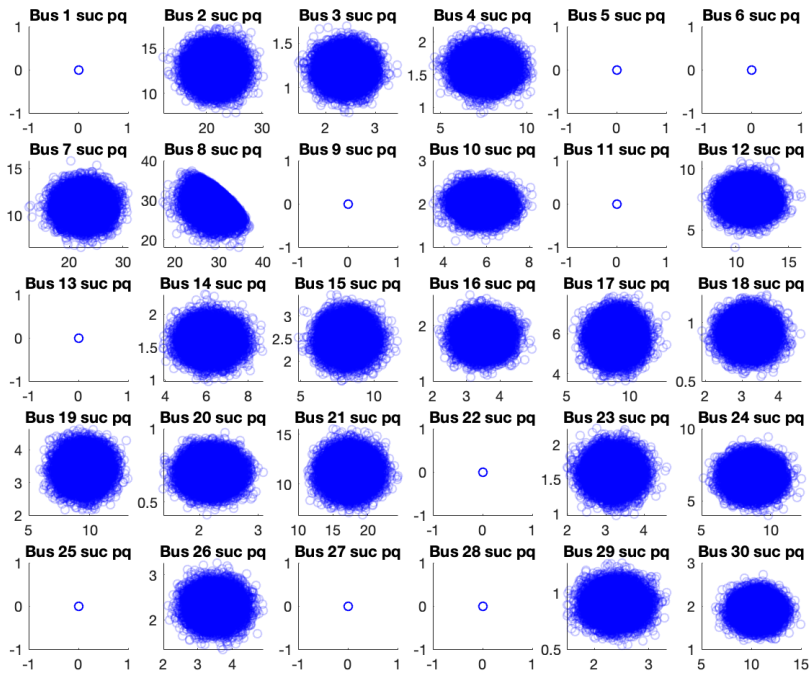


Figure 3: P versus Q values per bus for successful simulations

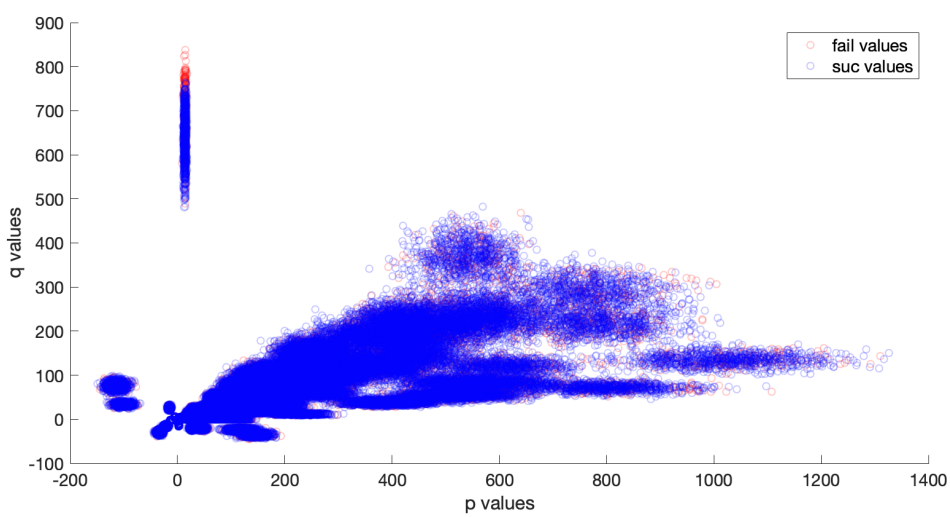


Figure 4: P versus Q values for all buses in the 300 bus system

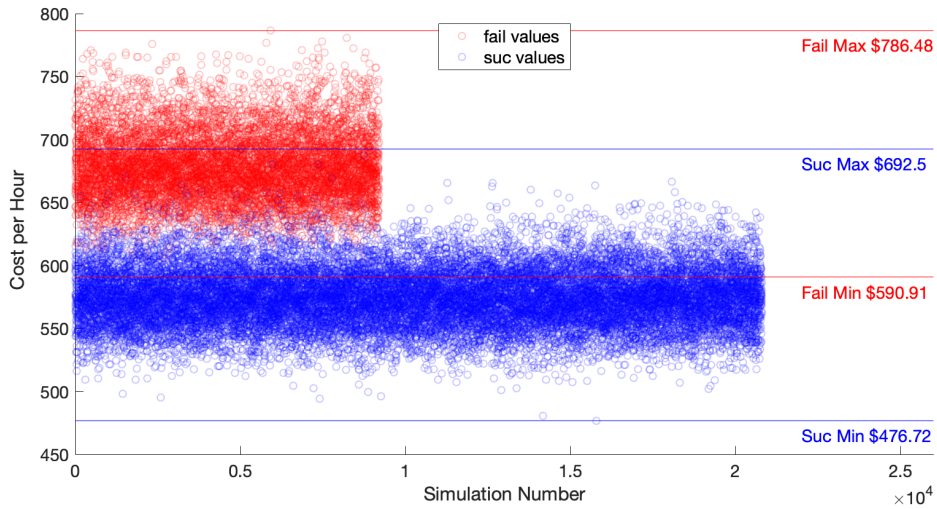


Figure 5: Costs for the 30 bus system per hour

to be learning anything with either dataset and was showing a high success rate within a single epoch, but a stagnant accuracy rating. There are several factors that may have contributed to this anomaly; it may be that the data was not variant enough to show differences between each perturbed system; or it is possible that a training dataset of 20,434 samples was insufficient to train the model.

2.2.3 Costs within the systems

One key output from an OPF solution is the cost per hour for running the optimized system. Figures 5 and 6 show the range of cost values for the solved cases.

It is clear from these graphs that the successful simulations are generally less expensive than those that failed. In `runopf` default setting, the constraint violation tolerance is 5×10^{-6}, and the maximum number of iterations is 150 for primal/dual interior point method [5]. So there are several possible reasons for us getting this result - It only gives us a local minimizer within the feasible set due to the limitation of number of iterations is not big enough, which provides a higher cost; Or, because Newton's method is sensitive to initial points, it makes sense that red and blue points in Figures 5 and 6 have relatively clear differences while they are overlapping with each other in Figures 1 and 4.

2.3 Neural Network Algorithm

Our goal was to design a neural network modeled off of a multilayer perceptron that is able to calculate the optimal generator requirements for a given grid topology, effectively solving the end-to-end OPF problem. This light weight model would then replace the complex, classically solved OPF model for a given topology. To design the network, we built upon the work in [4].

We tried different configurations and determined the optimal number of hidden layers and number of nodes for the neural network model, shown in figure 7. For accuracy, we

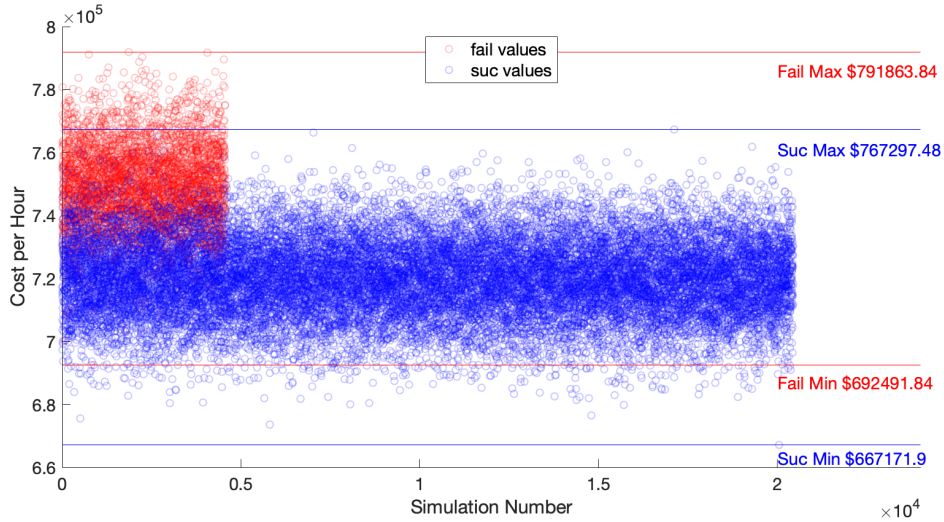


Figure 6: Costs for the 300 bus system per hour

used three metrics: a mean square error, the legality rate as defined in equation (5) and the average cost deviation as defined in equation (6).

$$\text{legality rate} = \frac{\text{number of predicted grids that satisfy (1d) and (1f)}}{\text{number of all predicted grids}} \tag{5}$$

$$\text{avg. cost dev.} = \frac{1}{n}\sum_{i=1}^{n}\left|1 - \frac{\text{pred cost}_i}{\text{true cost}_i}\right| \tag{6}$$

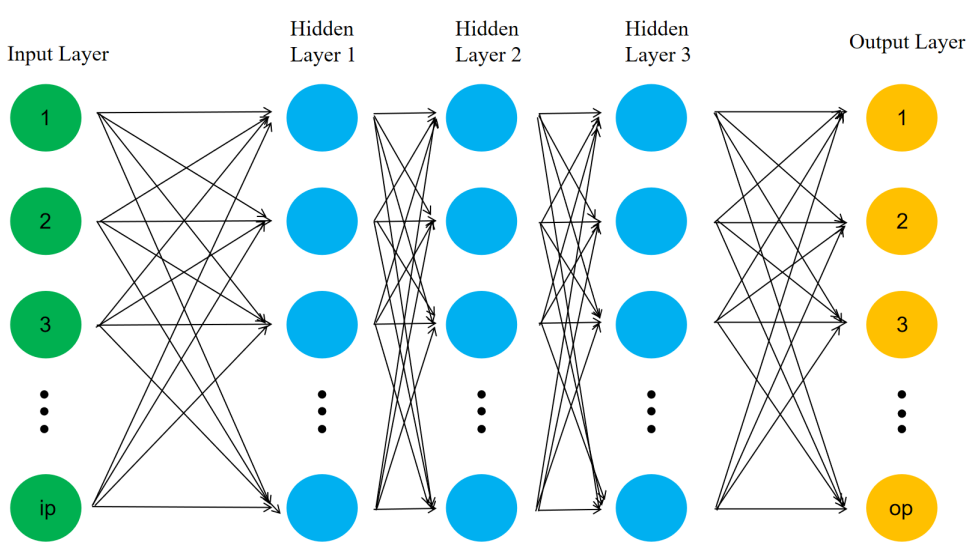


Figure 7: Neural Network Multilayer Perceptron Diagram with 3 Hidden Layers

3 Results

We used an 80/20 training/testing split for the datasets. For the 30 bus case, we generated 20,797 total samples—16,637 cases of which were used to train both 30 bus models, and the remaining 20% were used for testing.

3.1 Neural Network Model

Table 1 shows the results of our investigation into different configurations for the 30 bus neural network algorithm. The first column indicates the number of hidden layers, which we varied as 2, 3, or 5; the second column shows the number of nodes in each of the hidden layers; the third column indicates the activation function, for which we had two choices: ReLU (rectified linear unit) or tanh (hyperbolic tangent); the fourth column shows the validation accuracy; and the fifth column shows the mean cost deviation of the system, determined by equation (6).

There is a third choice for the activation function, the sigmoid function, but it performed significantly worse than the other functions so we did not include it within this table. The learning rate was set as 0.001 for all configurations and the number of epochs was set to 500. For the learning rate, we tried 0.1 initially but found the 0.001 was more effective for the accuracy of the model.

	Hidden Layers	Nodes	Activation Function	Validation Accuracy	Avg. Cost Deviation
1	1	10	ReLU	0.8769	0.0063
2	2	100/100	ReLU	0.9663	0.0042
3	3	5/10/5	ReLU	0.8724	0.0071
4	3	50/50/50	ReLU	0.9820	0.0038
5	3	100/50/100	ReLU/Tanh/ReLU	0.9880	0.0030
6	3	100/100/100	ReLU	0.9911	0.0025
7	3	100/100/100	Tanh	0.9418	0.0127
8	3	100/200/100	ReLU	0.9863	0.0046
9	5	100 - 100	ReLU	0.9932	0.0060

Table 1: Configurations investigated for fitting the 30 bus neural network model

If the constraints from (1d) and (1f) are violated, the system does not function within the engineering specs of the grid. The legality rate indicated in Table 2 is the percentage of legal outcomes predicted by the 30 bus algorithm with the indicated configurations.

We settled on line 6 from Table 1, having three hidden layers, each with 100 nodes, using the ReLU activation function for all three layers. Line 9 appears to have the best validation accuracy, but the accuracy fluctuated significantly with 500 epochs. The configuration we chose does not have that fluctuation, and it also has a higher legality rate and a lower average cost deviation. This configuration employs 27,512 parameters. We also note that the neural network had orders of magnitude faster computation time compared to the `runopf` MATPOWER function, showing that machine learning could in fact significantly reduce computation time for the OPF problem.

	Hidden Layers	Nodes	Activation Function	Legality
1	1	10	ReLU	86%
2	2	100/100	ReLU	74%
3	3	5/10/5	ReLU	98%
4	3	50/50/50	ReLU	76%
5	3	100/50/100	ReLU/Tanh/ReLU	74%
6	3	100/100/100	ReLU	80%
7	3	100/100/100	Tanh	94%
8	3	100/200/100	ReLU	72%
9	5	100 - 100	ReLU	76%

Table 2: Legality for 30 bus model configurations

It should be noted that we made several attempts to fit this algorithm to a 300 bus system, and the results that we found is that this network is not able to be scaled up for use in the 300 bus system.

In the Neural Network Diagram shown in Figure 7, we found the optimal number of hidden layers is 3. ip is the number of input notes, and op is the number of output notes. For bus number $n = 30$, $ip = 60$, and $op = 12$. To improve upon the original model from Neel Guha, we added in a batch normalization program within each hidden layer. This normalizes the data within each layer after the activation function to re-scale the data for less variance and avoid overfitting.

After getting an appropriate model based on training data, we used the three metrics mentioned in section 2.3 to determine its performance and effectiveness. For the 30 bus case, the prediction performance is summarized in Table 3.

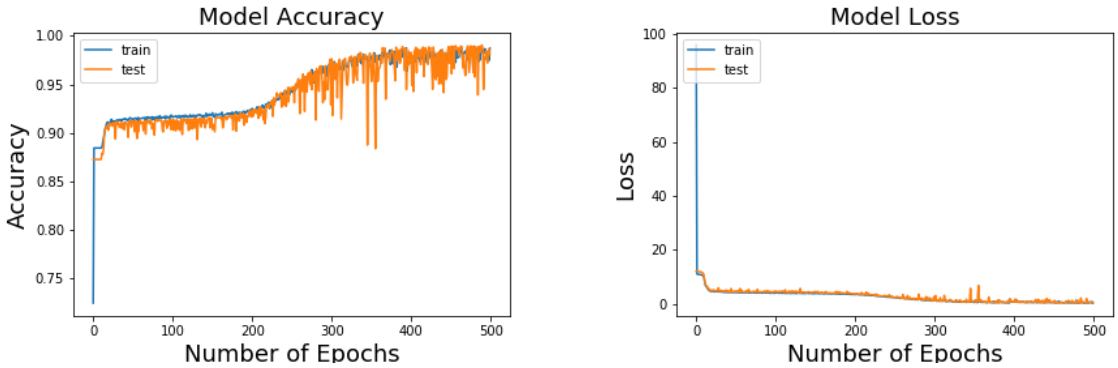
Grid	Accuracy	Legality Rate	Avg. Cost Dev.
case 30	99%	80%	.25 %

Table 3: Prediction performance for neural network 30 bus model

Figure 8 from left to right shows the model accuracy, mean absolute error, and the loss of the model for bus 30 case as the number of epochs increases. The blue line is for training data and the orange line is for testing data.

3.2 The XGBoost Model

XGBoost stands for Extreme Gradient Boosting and is a decision tree classification or regression model. The training method for this algorithm computes and minimizes the residual difference between the predicted value and the true value with each estimator or tree. In a classification algorithm, the first decision tree classifies as many points as possible and the output will be a binary representation of the success of the classifications. Those that were misclassified in the first tree are the focus of the second decision tree. This process perpetuates itself through the total amount of decision trees designated for the model and the output of the classification algorithm is a binary representation of the success of the



(a) NN Model Accuracy for the 30 bus case (b) The loss of the model for the 30 bus case

Figure 8: From left to right: MA and ML for 30 bus neural network model

classifications across all decision trees. In the case of a regression model, the output of each decision tree is approximated parameters and we take the mean of all of the outputs as the output for the entire model.

Table 4 shows the options that we tested for the XGBoost system with the 30 bus case. The first column is the number of decision trees within the algorithm. The R^2 score from the second column is the goodness of fit rating and is calculated using the following formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y_{it} - y_{ip})^2}{\sum_{i=1}^{m}(y_{it} - \overline{y_{it}})^2}, \tag{7}$$

where y_{it} is the true value of ith data point, y_{ip} is the predicted value of ith data point, and $\overline{y_{it}}$ is the mean value of true values. Then the last column is the mean cost deviation of the system, as defined in equation (6). All of these examples have a learning rate of 0.1 and tree depth of 3.

	Num of Trees	R^2 Score	Avg. Cost Dev.	Legality
1	100	0.7890	0.0141	86.13%
2	150	0.8044	0.0118	85.19%
3	200	0.8148	0.0104	84.76%
4	250	0.8226	0.0097	84.25%
5	300	0.8293	0.0093	84.18%
6	400	0.8411	0.0078	84.13%
7	600	0.8659	0.0074	83.82%
8	800	0.8738	0.0072	83.89%
9	1000	0.8809	0.0068	83.77%

Table 4: XGBoost configurations for 30 bus system

From Figure 9 it is clear that as we increase the amount of decision trees our fit rating is increasing at 800 trees, and our cost deviation is decreasing, which is to be expected. To optimize the system, we would want to check the R^2 values for an interval of decision trees and watch for overfitting or a decrease in goodness of fit. For the 30 bus system our

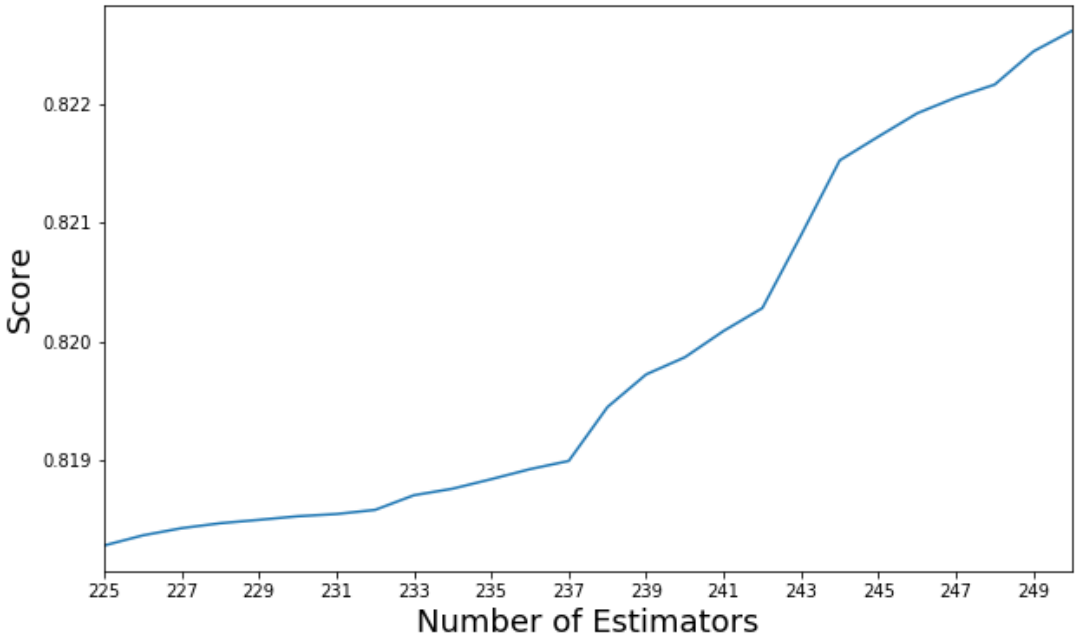


Figure 9: Goodness of fit score for XGBoost model as a function of the number of decision trees (estimators)

computer hardware limited us to testing large increments within the number of trees, and to optimize the system we need the capacity to evaluate the number of trees in much smaller increments.

3.3 Cost Analysis

Our neural network and XGBoost algorithm predictions both maintained a close approximation to the true costs indicated in the datasets. Figure 10 shows the distributions for the predicted costs versus the true costs for the neural network model, and Figure 11 shows the differences in cost for the XGBoost model. The XGBoost model had a bit more deviation from the true costs and appeared to be overestimating the costs, but that may be due to the lack of hardware necessary to fit the model.

3.4 Comparison of the Models

Given its prevalence in the current machine learning literature, we wanted to investigate the usefulness of the decision tree regression models while also experimenting with the more common neural network model. The XGBoost model required a significantly larger amount of computational power than the neural network model, but generally the XGBoost model is known for being more effective for a higher amount of input features. We also found that the legality rate for the XGBoost model was much more stable and remained rather high in

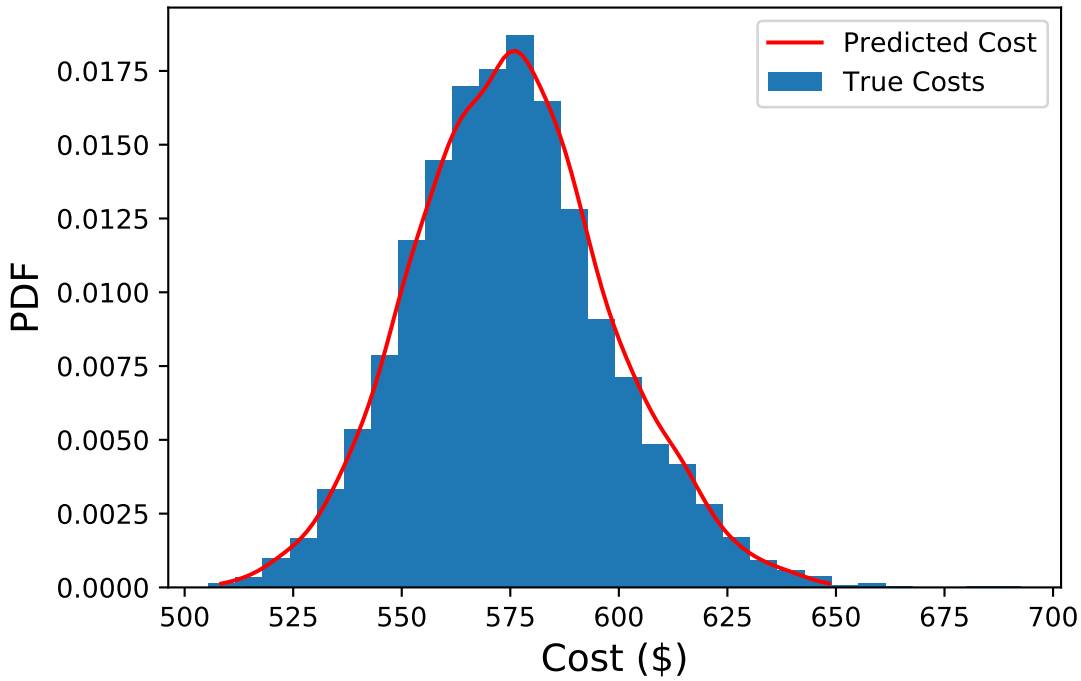


Figure 10: Comparison of predicted costs and true costs for the neural network model

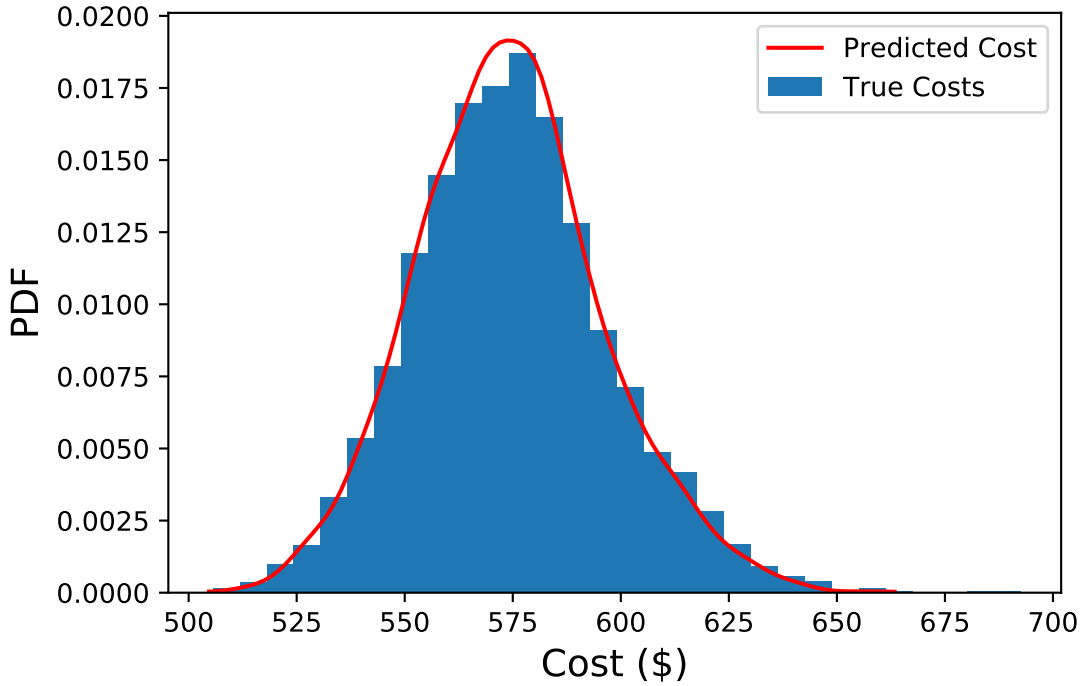


Figure 11: Comparison of predicted costs and true costs for the XGBoost model

comparison to the neural network which fluctuated through a 70-80% range.

4 Conclusions and Future Work

4.1 Conclusions

For this project, we created two machine learning models for solving the OPF for a 30 bus power grid system. The neural network model included three hidden layers with 100 nodes per layer, and with 500 epochs we found an accuracy rating of 99% for the training dataset and an 80% legality rate for the predictions. We were unable to optimize the XGBoost model due to insufficient computer processing capabilities, but found an 88% fit with 1000 decision trees. The neural network model was significantly faster than the XGBoost model in terms of computation time. We were unsuccessful at creating both types of models for the 300 bus system and we conjecture this failure has something to do with the dataset not being variant enough or large enough for the algorithm to recognize any patterns.

4.2 Future Work

This project represents just the beginning of research into using machine learning algorithms for solving the OPF problem. Generalizing the models to be effective for higher nodal systems is the next step, along with optimizing the number of decision trees for the regression tree model. Finding optimality classifications for each of these models and analyzing how they behave within extreme deviations from the norm are both future projects as well.

References

[1] Annual Report 2017. Technical report, European Network of Transmission System Operators for Electricity, 2018.

[2] Mary B Cain, Richard P Oneill, and Anya Castillo. History of optimal power flow and formulations. *Federal Energy Regulatory Commission*, 1:1–36, 2012.

[3] Stephen Frank and Steffen Rebennack. A Primer on Optimal Power Flow: Theory, Formulation, and Practical Examples. Working Papers 2012-14, Colorado School of Mines, Division of Economics and Business, October 2012.

[4] Neel Guha, Zhecheng Wang, Matt Wytock, and Arun Majumdar. Machine learning for AC optimal power flow. Technical report, ICML 2019, 2019.

[5] Ray D Zimmerman and Carlos E Murillo-Sánchez. Matpower 7.0 users manual. *PSERC: Tempe, AZ, USA*, 2019.