CS229: Machine Learning Models for Inverse Power Flow in the Grid

Jennifer Cardona, Thomas Gill, Siobhan Powell

December 16, 2017

1 Introduction

The electricity distribution system is undergoing major changes today with the addition of distributed energy resources (DERs) such as solar and wind, electric vehicles, and energy storage. These new features are challenging traditional control methods, straining grid infrastructure, and adding variability which makes voltage regulation and planning difficult using conventional modeling tools. Without measurements at every node in the system and good estimates of all line parameters, utilities cannot accurately model the flow of electricity to meet this new need. Consequently, new modeling methods have been proposed to address this challenge [1]. The forward power flow mappings represent the physical flow of electricity, giving the real and reactive net power injections, p and q, from the voltage phasor, $v = |v|e^{j\theta}$, at each bus and the line parameters of the system, G_{kj} and B_{kj} [2]:

$$p_k = \sum_{j=1}^{n} |v_k||v_j|(G_{kj}\cos(\theta_k - \theta_j) + B_{kj}\sin(\theta_k - \theta_j)), \tag{1}$$

$$q_k = \sum_{j=1}^{n} |v_k||v_j|(G_{kj}\sin(\theta_k - \theta_j) + B_{kj}\cos(\theta_k - \theta_j)). \tag{2}$$

This project proposes to use machine learning models to approximate the **inverse mapping** at all k buses, which is more useful than the forward map for problems in voltage regulation and control. The inputs to the model are the real and reactive power injections at all the buses, p and q, and the output is the voltage magnitude, $|v|$, at each bus.

$$|v_k| = f_k(p, q) \tag{3}$$

2 Related Work

Machine learning techniques have been applied to power flow problems in many different ways, including for projects in this class [3]. In several studies it has been used to predict optimal decisions for grid operators. For example, based on previous human decisions for remedial actions taken to prevent constraint violations, an algorithm was designed with the goal of improving reliability and limiting damage to equipment [4]. Other algorithms including reinforcement learning and basic classification methods have been used in similar ways to minimize overflows or curtailment, as well as to predict rare failure events [5, 6]. Most studies are most concerned with solving the Optimal Power Flow problem: designing generator decisions under numerous constraints to minimize the cost for the operator. Techniques to predict this optimal solution have included genetic algorithms [7, 3, 8, 9] and particle swarm optimization [10].

Predicting the solution to the optimal power flow problem in terms of operator decisions is useful only for very few problems in the management of the grid. Voltage regulation, for example, plays an important role in protecting distribution equipment and consumer devices, but requires much smaller scale modeling. Only one paper was found to study the forward mapping on this scale [1]. The model presented in this project is unique as a direct representation of the inverse mapping in Equation 1, which has not been explicitly studied in this way. Traditionally that direction of the relation is solved using iterations on the full forward mapping [11].

Currently voltage regulation with reactive power control is designed empirically, or through different methods of optimizing the full power flow equations as shown in Equation 1 over the whole network. Many of these methods are small alterations of traditional optimal power flow solvers which allow the power controls to be rescheduled to meet different constraints on the voltage magnitude [12, 13].

Since traditional methods of voltage regulation are computationally very expensive, relying on large numerical solvers over the full power flow equations, a closed form solution for the inverse mapping would be a great benefit in terms of computational cost and simplicity of analysis. To the best knowledge of the team, this specific model and direction has only been previously studied in ongoing research by one of the team members here at Stanford.

3 Data Set and Features

The system used for testing and data generation in this project was a 16 bus truncation of the IEEE 123 node test feeder [14]. This feeder was designed to provide good examples of voltage drop, and presented minimal problems of convergence during the generation of the data. Since one of the main applications of the model created by this project is voltage control, this was found to be an appropriate test system.

Using this network model with full knowledge of the system line parameters and connectivity, a power systems program called Matpower was used to create ground truth data points by solving the original power flow equations [11, 15]. It solved the inverse mapping using iterative optimization methods on the full power flow model [cite matpower]. To sample the whole system space, the input load values for real and reactive power, p and q, were randomly sampled in $[0, 1]$ and $[-1, 1]$ per unit. The output voltage magnitude and phase angle were collected at all the buses in the system.

A total of 8760 samples were created, representing hourly data collection over a period of one year. Table 1 shows the input and output features of the models created. Combined, p and q give a total of 32 input features.

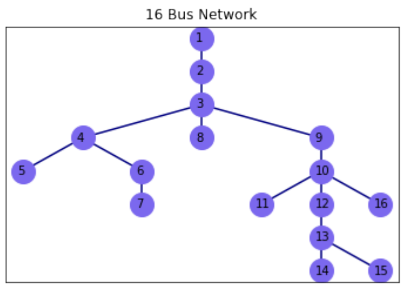

Figure 1: Sketch of network configuration for 16-bus system. Buses and lines are represented by nodes and edges. Bus 1 is the reference or slack bus, and buses 2 - 16 are pq load buses. Represents truncation of the 123-bus radial test feeder [14].

Inputs:	Real power, p, at all 16 buses at time t	$x_t = [p_t^T \quad q_t^T]^T$
	Reactive power, q, at all 16 buses at time t	
Outputs:	Voltage magnitude, $\|v\|$, at each of the 16 buses at time t	$y_t = \|v_t\|$

Table 1: Summary of input and output features of the model.

4 Methods

Weighted linear regression, support vector regression, and regression using a neural network were each implemented and evaluated for the baseline 16 bus system.

4.1 Locally Weighted Linear Regression

Weighted linear regression was chosen for its relative simplicity, making it easy to intuit where the model might work and understand where it might fail, and also for its physical relation to the loss-less case of power flow in an ideal network [16]. Implemented in MATLAB, a Gaussian weighting scheme was implemented on each bus, with highest weighting assigned to the buses numerically closest to the query point. This is described by a weight matrix, W, where the w_{ij} defines the influence of bus j on bus i, and τ is the Gaussian width parameter.

$$w_{ij} = exp(-\frac{(i-j)^2}{2\tau^2}) \tag{4}$$

In a serial network, this would be physically analogous to assigning highest weighting to the buses most closely connected to the bus of interest. Accordingly, the linear coefficient for each of these inputs is determined by solving the weighted normal equations [17]:

$$\theta = (X^T W X)^{-1} X^T W |v|, \tag{5}$$

$$|\hat{v}| = \theta^T X. \tag{6}$$

Here X and $|v|$ are concatenated matrices of the training inputs and outputs, and θ is a vector of the resulting linear coefficients used to make the voltage predictions.

4.2 Neural Network

Neural networks were selected with the insight that they might yield better results for more complex networks that are seen in real world applications, where simpler models studied in this project might fail.

Each hidden unit in the first hidden layer of the network receives a linear combination of the input features plus an intercept term as an input, and outputs a new value calculated using a nonlinear activation function. The units in the next hidden layer each receive a linear combination of the outputs from the previous layer plus an intercept term as its input, and again outputs a new value calculated with an activation function. This continues until the output layer is reached, which uses an output function to make a prediction. Hidden units in all layers were activated using a sigmoid function, and the output activation was an identity function to lead to a regression, as shown in Equation 7 for h hidden layers [17]. Multiple network structures and learning rates were explored using a grid search, given in Table 2. This was implemented in Python using scikit-learn [18].

$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]}; & a^{[1]} &= g_1(z^{[1]}); & g_1(z) &= \frac{1}{1+e^{-z}} \\ &\vdots & &\vdots & &\vdots \\ z^{[h]} &= W^{[h]}a^{[h-1]} + b^{[h]} & a^{[h]} &= g_h(z^{[h]}); & g_h(z) &= \frac{1}{1+e^{-z}} \\ z^{[h+1]} &= W^{[h+1]}a^{[h]} + b^{[h+1]}; & \hat{y} &= g_{h+1}(z^{[h+1]}); & g_{[h+1]}(z) &= z \end{aligned} \tag{7}$$

Hyperparameter	Options Tested in GridSearchCV
Network Architecture	(2), (3), (4), (5), (6), (7), (8), (9), (10), (16), (32), (2, 2), (2, 3), (2, 4), (2, 5), (3, 2), (3, 3), (3, 4), (3, 5), (4, 2), (4, 3), (4, 4), (4, 5), (5, 2), (5, 3), (5, 4), (5, 5), (16, 16), (32, 32), (32, 32, 32), (2, 2, 2, 2, 2, 2, 2, 2, 2, 2)
Learning Rate	0.01, 0.001

Table 2: Hyperparameter options tested using a grid search. Network architecture indicates how many hidden layers were used, and the number of units in each hidden layer (eg. (3, 4) refers to a network with 2 hidden layers where the 1st hidden layer has 3 units and the 2nd hidden layer has 4 units).

4.3 Support Vector Regression

Support vector regression was an obvious candidate to try for the inverse relation based on recent work being done at Stanford studying the forward mapping [1]. It was chosen for its compact formulation, and as a mechanism to use many different models for the relationship in the data through testing with different kernels.

The model for each bus k defines coefficients, $a_{k,t}$, for each vector in the training set, x_t, along with an intercept term, b, to predict the voltage magnitude as

$$|v_k|(x) = \sum_{t=1}^{T} a_{k,t} K(x_t, x) + b_k. \tag{8}$$

For testing in this project four kernel functions were used: linear, polynomial with degree 2, polynomial with degree 3, and radial basis function (gaussian). The are defined as

$$\begin{aligned} K(x_t, x) &= x_t^T x & &\text{Linear kernel,} \\ K(x_t, x) &= \left(1 + x_t^T x\right)^{2 \text{ or } 3} & &\text{Quadratic or Cubic kernel,} \\ K(x_t, x) &= \exp\left(-\gamma ||x - x_t||_2^2\right) & &\text{RBF kernel.} \end{aligned} \tag{9}$$

Originally implemented by hand in MATLAB using CVX and validated through comparison with the sklearn solution in Python, the results presented here were all solved with sklearn [18, 19]. The coefficients and bias term were calculated by solving the following optimization function, where $a_t = \alpha_t - \alpha_t^*$ [20].

$$\begin{aligned} \underset{\alpha,\alpha^*}{\text{minimize}} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(x_i, x_j) + \epsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) - \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0, \quad 0 \le \alpha_i, \alpha_i^* \le C \quad \forall i = 1, \ldots, N \end{aligned} \tag{10}$$

The regularization and error margin terms, C and ϵ, were optimized for each fitting using the GridSearchCV function to find the best possible model in each case [18].

5 Experiments, Results, and Discussion

5.1 Experiments

Each algorithm was implemented in several test cases to test its robustness and efficacy for several common grid study applications. In addition to being trained on clean data, each algorithm was trained on data with varying amounts of random noise (1%, 2%, and 5%) to mimic measurements that would be collected by sensors. Additional experiments were performed by training the algorithms on a case of limited observability where measurements of real and reactive power were only collected from the leaf nodes (buses 1, 5, 7, 8, 11, 14, 15, and 16), which is a realistic case for implementation in residential networks. Finally, each algorithm was tested and trained on data with negative real loads, simulating a network with substantial renewable energy generation which causes power to flow in both directions along the lines. In order to provide direct comparison of accuracy between different algorithms, root mean squared error, J, was used as a metric, where m and N are the number of samples and buses:

$$J = \frac{1}{m}\sum_{i=1}^{m}\sqrt{\frac{\sum_{n=1}^{N}(\hat{v}-v_{test})^2}{N}}. \tag{11}$$

5.1.1 Parameters

For the weighted linear regression models, Gaussian weighting was maintained throughout each experiment. On the base case, the bandwidth parameter was tuned manually between values of $\tau = [.1, 25]$. Ultimately, a bandwidth parameter of 7 was utilized for all cases, as it was found to give weights of > 0.1 to all buses in the network.

In the neural network implementations, based on the results of a grid search using GridSearchCV in sklearn [18], the learning rate was constantly 0.001 and the architectures as presented in Table 3 were used. In the support vector regression tests, the values of C and ϵ as defined in Equation 10 were optimized for each implementation.

	Baseline	Noise ±1%	Noise ±2%	Noise ±5%	Lim Obs	Lim Obs ±1%	Solar	Solar ±1%
Architecture	(2, 3)	(2, 3)	(2, 3)	(5, 3)	(2)	(2)	(3)	(3)

Table 3: Hyper-parameters chosen for each case based on best grid search results

5.2 Results

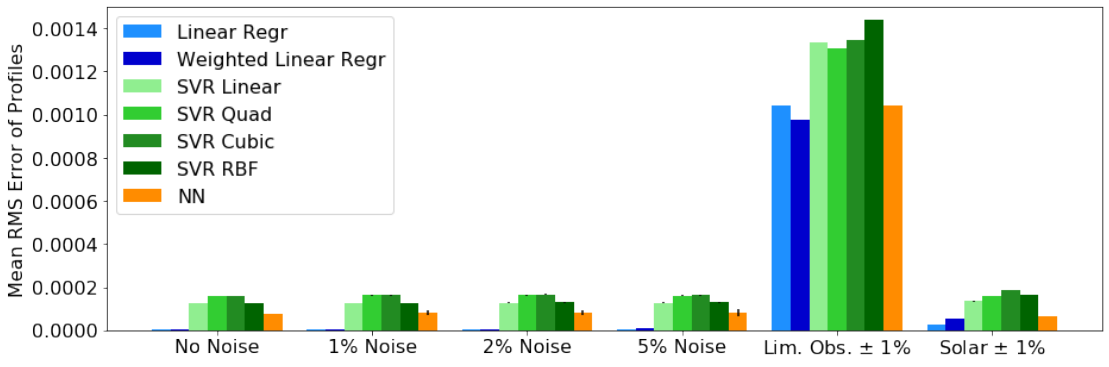

Figure 2: Comparison of mean profile rms errors for the various methods and test cases. Cases with noise were run and averaged over 15 times, and the error bars representing the variability in the results are shown on their bars.

As seen in the results in Figure 2, all models produced consistently robust predictions when noise was added to the data, with deviations in RMS error between the 'No Noise' and '5% Noise' cases on the order of 1×10^{-5}. All models were able to reproduce the original profile very well, as demonstrated in Figure 3. The chosen profile shows that each model provides visibly indistinguishable predictions on a $10^{-}3$ scale.

Linear regression substantially outperformed all other algorithms with an RMS error of 5×10^{-6} in the base case. The solar test with 1% noise yielded higher average error, but showed a similar trend to the noisy base cases, in which the linear regression (RMS 5×10^{-5}) and the neural network (RMS 8×10^{-5}) outperformed all the support vector kernels. The limited observability case proved most difficult to predict, with linear and weighted linear regression performing similarly to the neural network (RMS 1.0×10^{-3}), the first instance in which a more sophisticated algorithm provided comparable performance. In this case the ordering of the SVR kernels was also found to change.

The training and test error converged quickly (2000 examples) for neural networks, with weighted linear regression (3000-4000) and cubic kernel SVR (>4000) taking slightly longer. Each learning curve shows a high degree of convergence between test and training error, as expected since the data are from identical distributions and the models do well.

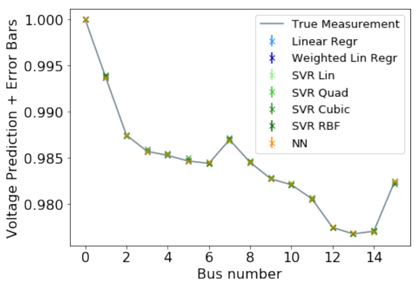

Figure 3: Illustration of the model predictions on a voltage profile chosen from the test set. This profile was the one in the test set with the largest magnitude voltage drop.

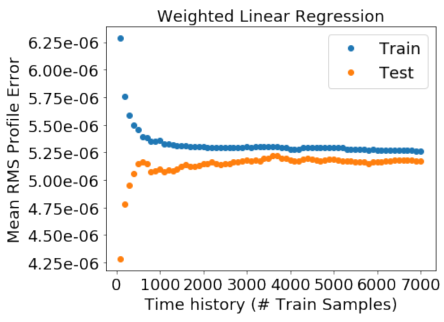

Figure 4: Learning curve for Gaussian weighted linear regression.

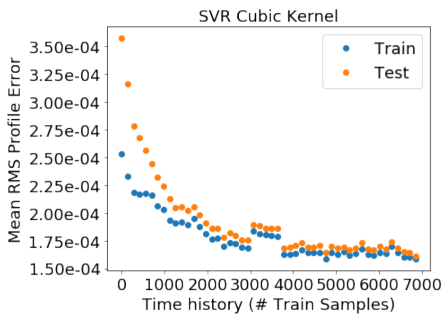

Figure 5: Learning curve for support vector regression using a cubic kernel.

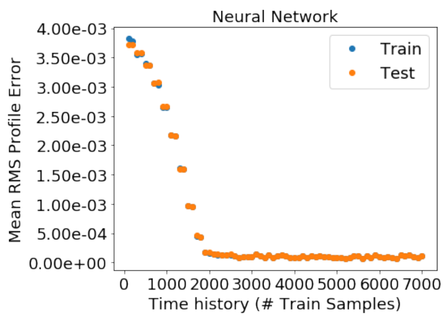

Figure 6: Learning curve for a sample neural network used.

6 Conclusion

6.1 Discussion

In all of the test cases presented in Section 5.2, the linear regression prediction had the highest accuracy. In all but one case the normal linear regression was the best, and only in the case with limited observability of the nodes did the locally-weighted linear regression surpass it.

This is an interesting yet somewhat intuitive result, as the linear map corresponds to the lossless case where there is no power lost moving along the lines. While the drooping voltage profiles found throughout the data set confirm that there are losses in the system, the losses were not large in this small test case.

The goal of the project was to try to capture the loss, the nonlinearity in the mapping, using these more complex models, and it is interesting that none of the models could achieve that since the loss is a direct and predictable function of the system and the inputs used.

The test case with limited observability illustrated several interesting points. First, by proving the most difficult for all of the models to predict, it showed that collecting as many measurement points as possible from the system would contribute more to the accuracy of the prediction than would the choice of model. In addition, in the observability test the neural network results matched the linear regression and the ordering of the SVR kernel models changed. This suggests that networks with limited or unknown observability are better capture with more complicated models.

The main application of this project is in networks where not all the system parameters are known. In that case it is also likely that not all nodes are observed or the data is otherwise incomplete, so the best model choice for that application may most closely match the observability results found in this project.

The results under different levels of measurement noise show that all models, when the parameters are re-optimized, are quite resilient. The neural network shows larger variations in those cases than the others, as illustrated by the error bars in the test error plots, but in all cases they vary by very small amounts.

Beyond accuracy, system operators may want to compare models based on other factors such as simplicity, computational cost, or robustness to noise and change in their specific application. For analysis and the voltage optimization application, for example, the simplest models are preferable.

6.2 Future Work

The first next step for this project will be to test the models on larger networks, where greater variability and loss values are expected to require more complicated models and give different results. It will also be interesting to study the online updating mechanisms of these models more carefully, to develop a fuller understanding of how the models would rank for real time applications. Future work will use the optimal model from this study in other applications, to conduct control, optimization, and other power network analyses.

7 Contributions

The report, poster preparation, and analysis were all equally divided and collaborated on by all three team members. The development and testing of the weighted linear regression model was done by Thomas Gill, the development and testing of the neural network model was done by Jennifer Cardona, and the development and testing of the support vector regression models was done by Siobhan Powell.

References

[1] J. Yu, Y. Weng, and R. Rajagopal, “Mapping Rule Estimation for Power Flow Analysis in Distribution Grids,” 2017.

[2] H. Dommel and W. Tinney, “Optimal Power Flow Solutions,” *IEEE Transactions on Power Apparatus and Systems*, 1968.

[3] T. Navidi, S. Bhooshan, and A. Garg, “Predicting Solutions to the Optimal Power Flow Problem,”

[4] B. Donnot, I. Guyon, M. Schoenauer, P. Panciatici, and A. Marot, “Introducing machine learning for power system operation support,” 2017.

[5] C. Rudin, D. Waltz, R. N. Anderson, A. Boulanger, A. Salleb-Aouissi, M. Chow, H. Dutta, P. N. Gross, B. Huang, S. Ierome, D. F. Isaac, A. Kressner, R. J. Passonneau, A. Radeva, and L. Wu, “Machine Learning for the New York City Power Grid,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[6] J. E. King, S. C. E. Jupe, and P. C. Taylor, “Network State-Based Algorithm Selection for Power Flow Management Using Machine Learning,” *IEEE TRANSACTIONS ON POWER SYSTEMS*, vol. 30, no. 5, 2015.

[7] L. Lai, J. Ma, R. Yokoyama, and M. Zhao, “Improved genetic algorithms for optimal power flow under both normal and contingent operation states,” *International Journal of Electrical Power & Energy Systems*, 1997.

[8] Q. H. Wu and J. T. Ma, “POWER SYSTEM OPTIMAL REACTIVE POWER DISPATCH USING EVOLUTIONARY PROGRAMMING,” *IEEE Transactions on Power System*, vol. 10, no. 3, 1995.

[9] A. G. Bakirtzis, P. N. Biskas, C. E. Zoumas, and V. Petridis, “Optimal Power Flow by Enhanced Genetic Algorithm,” *IEEE TRANSACTIONS ON POWER SYSTEMS*, vol. 17, no. 2, 2002.

[10] C.-R. Wang, H.-J. Yuan, Z.-Q. Huang, J.-W. Zhang, and C.-J. Sun, “A MODIFIED PARTICLE SWARM OPTIMIZATION ALGORITHM AND ITS APPLICATION IN OPTIMAL POWER FLOW PROBLEM,” pp. 18–21, 2005.

[11] R. D. Zimmerman and C. E. Murillo-s, “Matpower 4.1 User’s Manual,” *Power Systems Engineering Research Center (Pserc)*, 2011.

[12] D. S. Kirschen, H. P. Member, and M. Van, “WW/Voltage Control in a Linear Programming Based Optimal Power Flow,” *IEEE Transactions on Power Systems*, vol. 3, no. 2, 1988.

[13] K. R. C. Mamandur, “OPTIMAL CONTROL OF REACTIVE POWER FLOW FOR IMPROVEMENTS IN VOLTAGE PROFILES AND FOR REAL POWER LOSS MINIMIZATION,” *IEEE Transactions on Power Apparatus and Systems*, no. 7, 1981.

[14] W. H. Kersting, “Radial distribution test feeders,” *Proceedings of the IEEE Power Engineering Society Transmission and Distribution Conference*, 2001.

[15] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, “MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education,” *IEEE Trans. Power Syst.*, 2011.

[16] B. Stott, J. Jardim, and O. Alsaç, “DC Power Flow Revisited,” *IEEE TRANSACTIONS ON POWER SYSTEMS*, vol. 24, no. 3, 2009.

[17] A. Ng, “CS229 Lecture notes,” *CS229 Lecture notes*, 2000.

[18] F. Pedregosa and G. Varoquaux, *Scikit-learn: Machine learning in Python.* 2011.

[19] M. Grant and S. Boyd, “CVX: Matlab software for disciplined convex programming,” 2008.

[20] N. Deng, T. Yingjie, and Z. Chunhua, *Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions.* Boca Raton: CRC Press, Taylor & Francis Group, 2013.