Using Neural Networks to Detect Line Outages from PMU Data

Ching-pei Lee and Stephen J. Wright

***Abstract*—We propose an approach based on neural networks and the AC power flow equations to identify single- and double-line outages in a power grid using the information from phasor measurement unit sensors (PMUs) placed on only a subset of the buses. Rather than inferring the outage from the sensor data by inverting the physical model, our approach uses the AC model to simulate sensor responses to all outages of interest under multiple demand and seasonal conditions, and uses the resulting data to train a neural network classifier to recognize and discriminate between different outage events directly from sensor data. After training, real-time deployment of the classifier requires just a few matrix-vector products and simple vector operations. These operations can be executed much more rapidly than inversion of a model based on AC power flow, which consists of nonlinear equations and possibly integer / binary variables representing line outages, as well as the variables representing voltages and power flows. We are motivated to use neural network by its successful application to such areas as computer vision and natural language processing. Neural networks automatically find nonlinear transformations of the raw data that highlight useful features that make the classification task easier. We describe a principled way to choose sensor locations and show that accurate classification of line outages can be achieved from a restricted set of measurements, even over a wide range of demand profiles.**

***Index Terms*—line outage identification, phasor measurement unit, neural network, optimal PMU placement**

I. Introduction

Phasor measurement units (PMUs) have been introduced in recent years as instruments for monitoring power grids in real time. PMUs provide accurate, synchronized, real-time information of the voltage phasor at 30-60 Hz, as well as information about current flows. When processed appropriately, this data has the potential to perform rapid identification of anomalies in operation of the power system. In this paper, we use this data to detect line outage events, discriminating between outages on different lines. This discrimination capability (known in machine learning as "classification") is made possible by the fact that the topological change to the grid resulting from a line outage leads (after a transient period during which currents and voltages fluctuate) to a new steady state of voltage and power values. The pattern of voltage and power changes is somewhat distinctive for different line outages. By gathering or simulating many samples of these changes, under different load conditions, we can train a machine-learning classifier to recognize each type of line outage. Further, given that it is not common in current practice to install PMUs on all buses, we extend our methodology to place a limited number of PMUs in the network in a way that maximizes the performance of outage detection, or to find optimal locations for *additional* PMUs in a network that is already instrumented with some PMUs.

Earlier works on classification of line outages from PMU data are based on a linear (DC) power flow model [1], [2], or make use only of phasor angle changes [3], [4], [5], or design a classifier that depends only linearly on the differences in sensor readings before and after an event [6]. These approaches fail to exploit fully the modeling capabilities provided by the AC power flow equations, the information supplied by PMUs, and the power of modern machine learning techniques. Neural networks have the potential to extract automatically from the observations information that is crucial to distinguishing between outage events, transforming the raw data vectors into a form that makes the classification more accurate and reliable. Although the computational burden of training a neural-network classifier is heavy, this processing can be done "offline." The cost of deploying the trained classifier is low. Outages can be detected and classified quickly, in real time, possibly leading to faster remedial action on the grid, and less damage to the infrastructure and to customers. The idea of using neural networks on PMU data is also studied in [7] to detect multiple simultaneous line outages, in the case that PMU data from all buses are available along with data for power injections at all buses.

The use of neural networks in deep learning is currently the subject of much investigation. Neural networks have yielded significant advances in computer vision and speech recognition, often outperforming human experts, especially when the hidden information in the raw input is not captured well by linear models. The limitations of linear models can sometimes be overcome by means of laborious feature engineering, which requires expert domain knowledge, but this process may nevertheless miss vital information hidden in the data that is not discernible even by an expert. We show below that, in this application to outage detection on power grids, even generic neural network models are effective at classifying outages accurately across wide ranges of demands and seasonal effects. Previous works of data-based methods for outage detection only demonstrated outage-detecting ability of these models for a limited range of demand profiles. We show that neural network models can cope with a wider range of realistic demand scenarios, that incorporate seasonal,

This work was supported by a DOE grant subcontracted through Argonne National Laboratory Award 3F-30222, National Science Foundation Grants IIS-1447449 and CCF-1740707, and AFOSR Award FA9550-13-1-0138.

C. Lee and S. J. Wright are with the Computer Sciences Department, 1210 W. Dayton Street, University of Wisconsin, Madison, WI 53706, USA (e-mails: ching-pei@cs.wisc.edu and swright@cs.wisc.edu).

diurnal, and random fluctuations. Although not explored in this paper, our methodology could incorporate various scenarios for power supply at generation nodes as well. We show too that effective outage detection can be achieved with information from PMUs at a limited set of network locations, and provide methodology for choosing these locations so as to maximize the outage detection performance.

Our approach differs from most approaches to machine learning classification in one important respect. Usually, the data used to train a classifier is *historical* or *streaming*, gathered by passive observation of the system under study. Here, instead, we are able to *generate* the data as required, via a high-fidelity model based on the AC power flow equations. Since we can generate enough instances of each type of line outage to make them clearly recognizable and distinguishable, we have an important advantage over traditional machine learning. The role of machine learning is thus slightly different from the usual setting. The classifier serves as a proxy for the physical model (the AC power flow equations), treating the model as a black box and performing the classification task phenomenologically based on its responses to the "stimuli" of line outages. Though the offline computational cost of training the model to classify outages is high, the neural network proxy can be deployed rapidly, requiring much less online computation than an inversion of the original model.

This work is an extension and generalization of [6], where a linear machine learning model (multiclass logistic regression, or MLR) is used to predict the relation between the PMU readings and the outage event. The neural-network scheme has MLR as its final layer, but the network contains additional "hidden layers" that perform nonlinear transformations of the raw data vectors of PMU readings. We show empirically that the neural network gives superior classification performance to MLR in a setting in which the electricity demands vary over a wider range than that considered in [6]. (The wider range of demands casues the PMU signatures of each outage to be more widely dispersed, and thus harder to classify.) A similar approach to outage detection was discussed in [8], using a linear MLR model, with PMU data gathered during the transient immediately after the outage has occurred, rather than the difference between the steady states before and after the outage, as in [6]. Data is required from all buses in [8], whereas in [6] and in the present paper, we consider too the situation in which data is available from only a subset of PMUs.

Another line of work that uses neural networks for outage detection is reported in [7] (later expanded into the report [9], which appeared after the original version of this paper was submitted). The neural networks used in [7], [9] and in our paper are similar in having a single hidden layer. However, the data used as inputs to the neural networks differs. We use the voltage angles and magnitudes reported by PMUs, whereas [7], [9] use only voltage angles along with power injection data at all buses. Moreover, [7], [9] require PMU data from *all* buses, whereas we focus on identifying a subset of PMU locations that optimizes classification performance. A third difference is that [7], [9] aim to detect multiple, simultaneous line outages using a multilabel classification formulation, while we aim to identify only single- or simultaneous double-line outages. The latter are typically the first events to occur in a large-scale grid failure, and rapid detection enables remedial action to be taken. We note too that PMU data is simulated in [7], [9] by using a DC power flow model, rather than our AC model, and that a variety of power injections are obtained in the PMU data not by varying over a plausible range of seasonal and diurnal demand/generation variations (as we do) but rather by perturbing voltage angles randomly and inferring the effects of these perturbations on power readings at the buses.

This paper is organized as follows. In Section II, we give the mathematical formulation of the neural network model, and the regularized formulation that can be used to determine optimal PMU placement. We then discuss efficient optimization algorithms for training the models in Section III. Computational experiments are described Section IV. A convergence proof for the optimization method is presented in the Appendix.

II. Neural Network and Sparse Modeling

In this section, we discuss our approach of using neural network models to identify line outage events from PMU change data, and extend the formulation to find optimal placements of PMUs in the network. (We avoid a detailed discussion of the AC power flow model.) We use the following notation for outage event.

- y_i denotes the outage represented by event i. It takes a value in the set $\{1, \ldots, K\}$, where K represents the total number of possible outage events (roughly equal to the number of lines in the network that are susceptible to failure).
- $\boldsymbol{x}_i \in \mathbf{R}^d$ is the vector of differences between the pre-outage and post-outage steady-state PMU readings,

In the parlance of machine learning, y_i is known as a *label* and $\boldsymbol{x}_i$ is a *feature vector*. Each i indexes a single item of data; we use n to denote the total number of items, which is a measure of the size of the data set.

A. Neural Network

A neural network is a machine learning model that transforms the data vectors $\boldsymbol{x}_i$ via a series of transformations (typically linear transformations alternating with simple component-wise nonlinear transformations) into another vector to which a standard linear classification operation such as MLR is applied. The transformations can be represented as a network. The nodes in each layer of this network correspond to elements of an intermediate data vector; nonlinear transformations are performed on each of these elements. The arcs between layers correspond to linear transformations, with the weights on each arc representing an element of the matrix that describes the linear transformation. The bottom layer of nodes contains the elements of the raw data vector while a "softmax" operation applied to the outputs of the top layer indicates the probabilities of the vector belonging to each of the K possible classes. The layers / nodes strictly between the

top and bottom layers are called "hidden layers" and "hidden nodes."

A neural network is *trained* by determining values of the parameters representing the linear and nonlinear transformations such that the network performs well in classifying the data objects $(\boldsymbol{x}_i, y_i)$, $i = 1, 2, \ldots, n$. More specifically, we would like the probability assigned to node y_i for input vector $\boldsymbol{x}_i$ to be close to 1, for each $i = 1, 2, \ldots, n$. The linear transformations between layers are learned from the data, allowing complex interactions between individual features to be captured. Although deep learning lacks a satisfying theory, the layered structure of the network is thought to mimic gradual refinement of the information, for highly complicated tasks. In our current application, we expect the relations between the input features — the PMU changes before / after an outage event — to be related to the event in complex ways, making the choice of a neural network model reasonable.

Training of the neural network can be formulated as an optimization problem as follows. Let N be the number of hidden layers in the network, with $d_1, d_2, \ldots, d_N \geq 0$ being the number of hidden nodes in each hidden layer. ($d_0 = d$ denotes the dimension of the raw input vectors, while $d_{N+1} = K$ is the number of classes.) We denote by W_j the matrix of dimensions $d_{j-1} \times d_j$ that represents the linear transformation of output of layer $j-1$ to the input of layer j. The nonlinear transformation that occurs within each layer is represented by the function σ. With some flexibility of notation, we obtain $\sigma(\boldsymbol{x})$ by applying the same transformation to each component of $\boldsymbol{x}$. In our model, we use the $\tanh$ function, which transforms each element $\nu \in \mathbf{R}$ as follows:

$$\nu \to (e^{\nu} - e^{-\nu})/(e^{\nu} + e^{-\nu}). \tag{1}$$

(Other common choices of σ include the sigmoid function $\nu \to 1/(1+e^{-\nu})$ and the rectified linear unit $\nu \to \max(0, \nu)$.) This nonlinear transformation is not applied at the output layer $N+1$; the outputs of this layer are obtained by applying an MLR classifier to the outputs of layer N.

Using this notation, together with $[n] = \{1, 2, \ldots, n\}$ and $[N] = \{1, 2, \ldots, N\}$, we formulate the training problem as:

$$\min_{W_1, W_2, \ldots, W_{N+1}} f(W_1, W_2, \ldots, W_{N+1}), \tag{2}$$

where the objective is defined by

$$f(W_1, \ldots, W_{N+1}) := \sum_{i=1}^{n} \ell(\boldsymbol{x}_i^{N+1}, y_i) + \frac{\epsilon}{2} \sum_{j=1}^{N+1} \|W_j\|_F^2, \tag{3a}$$

$$\text{subject to } \boldsymbol{x}_i^{N+1} = W_{N+1} \boldsymbol{x}_i^N, \quad i \in [n], \tag{3b}$$

$$\boldsymbol{x}_i^j = \sigma(W_j \boldsymbol{x}_i^{j-1}), \; i \in [n], \; j \in [N], \tag{3c}$$

$$\boldsymbol{x}_i^0 = \boldsymbol{x}_i, \qquad i \in [n], \tag{3d}$$

for some given regularization parameter $\epsilon \geq 0$ and Frobenius norm $\|\cdot\|_F$, and nonnegative convex loss function ℓ. [1] We use the constraints in (3) to eliminate intermediate variables $\boldsymbol{x}_i^j$, $j = 1, 2, \ldots, N+1$, so that indeed (2) is an unconstrained optimization problem in $W_1, W_2, \ldots, W_{N+1}$. The loss function ℓ quantifies the accuracy which with the neural network predicts the label y_i for data vector $\boldsymbol{x}_i$. As is common, we use the MLR loss function, which is the negative logarithm of the softmax operation, defined by

$$\ell(\boldsymbol{z}, y_i) := -\log\left(\frac{e^{z_{y_i}}}{\sum_{k=1}^{K} e^{z_i}}\right) = -z_{y_i} + \log\left(\sum\nolimits_{k=1}^{K} e^{z_i}\right), \tag{4}$$

where $\boldsymbol{z} = (z_1, z_2, \ldots, z_K)^T$. Since for a transformed data vector $\boldsymbol{z}$, the neural network assigns a probability proportional to $\exp(z_k)$ for each outcome $k = 1, 2, \ldots, K$, this function is minimized when the neural network assigns zero probabilities to the incorrect labels $k \neq y_i$.

In practice, we add "bias" terms at each layer, so that the transformations actually have the form

$$\boldsymbol{x}_i^{j-1} \to W_j \boldsymbol{x}_i^{j-1} + w_j,$$

for some parameter $w_j \in \mathbf{R}^{d_j}$. We omit this detail from our description, for simplicity of notation.

Despite the convexity of the loss function ℓ as a function of its arguments, the overall objective (3) is generally nonconvex as a function of $W_1, W_2, \ldots, W_{N+1}$, because of the nonlinear transformations σ in (3c), defined by (1).

B. Inducing Sparsity via Group-LASSO Regularization

In current practice, PMU sensors are attached to only a subset of transmission lines, typically near buses. We can modify the formulation of neural network training to determine which PMU locations are most important in detecting line outages. Following [6], we do so with the help of a nonsmooth term in the objective that penalizes the use of each individual sensor, thus allowing the selection of only those sensors which are most important in minimizing the training loss function (3). This penalty takes the form of the sum of Frobenius norms on submatrices of W_1, where each submatrix corresponds to a particular sensor. Suppose that $G_s \subset \{1, 2, \ldots, d\}$ is the subset of features in $\boldsymbol{x}_i$ that are obtained from sensor s. If the columns $j \in G_s$ of the matrix W_1 are zero, then these entries of $\boldsymbol{x}_i$ are ignored — the products $W_1 \boldsymbol{x}_i$ will be independent of the values $(\boldsymbol{x}_i)_j$ for $j \in G_s$ — so the sensor s is not needed. Denoting by I a set of sensors, we define the regularization term as follows:

$$c(W_1, I) := \sum\nolimits_{s \in I} r(W_1, G_s), \text{ where} \tag{5a}$$

$$r(W_1, G_s) := \sqrt{\sum\nolimits_{i=1}^{d_1} \sum_{j \in G_s} (W_1)_{i,j}^2} = \|(W_1)_{\cdot G_s}\|. \tag{5b}$$

(We can take I to be the full set of sensors or some subset, as discussed in Subsection III-B.) This form of regularizer is sometimes known as a group-LASSO [10], [11], [12]. With this regularization term, the objective in (2) is replaced by

$$L_I(W) := f(W_1, \ldots, W_N) + \tau c(W_1, I), \tag{6}$$

[1] We chose a small positive value $\epsilon = 10^{-8}$ for our experiments, as a positive value is required for the convergence theory; see in particular Lemma 1 in the Appendix. The computational results were very similar for $\epsilon = 0$, however.

for some tunable parameter $\tau \geq 0$. A larger τ induces more zero groups (indicating fewer sensors) while a smaller value of τ tends to give lower training error at the cost of using more sensors. Note that no regularization is required on W_i for $i > 1$, since W_1 is the only matrix that operates directly on the vectors of data from the sensors.

We give further details on the use of this regularization in choosing PMU locations in Subsection III-B below. Once the desired subset has been selected, we drop the regularization term and solve a version of (2) in which the columns of W_1 corresponding to the sensors not selected are fixed at zero.

III. Optimization and Selection Algorithms

Here we discuss the choice of optimization algorithms for solving the training problem (2) and its regularized version (6). We also discuss strategies that use the regularized formulation to select PMU locations, when we are only allowed to install PMUs on a pre-specified number of buses.

A. Optimization Frameworks

ALGORITHM 1: Greedy heuristic for feature selection

Given $\epsilon, \tau > 0$, #max_group $\in \mathbf{N}$, set I of possible sensor locations, and disjoint groups $\{G_s\}$ such that $\bigcup_{s\in I} G_s \subset \{1, \ldots, d\}$;
Set $G \leftarrow \emptyset$;
for $k = 1, \ldots,$*#max_group* **do**
  if $k > 1$ **then**
    Let the initial point be the solution from the previous iteration;
  else
    Randomly initialize $W_i \in \mathbf{R}^{d_{i-1}\times d_i}$, $i \in [N+1]$;
  end
  Approximately solve (6) with the given τ and the current I by SpaRSA;
  $\tilde{s} := \arg\max_{s\in I} \quad r(W_1, G_s)$;
  if $r(W_1, G_{\tilde{s}}) = 0$ **then**
    Break;
  end
  $I \leftarrow I \setminus \tilde{s}$, $G \leftarrow G \cup \{\tilde{s}\}$;
end
Output G as the selected buses and terminate;

We solve the problem (2) with the popular L-BFGS algorithm [13]. Other algorithms for smooth nonlinear optimization can also be applied; we choose L-BFGS because it requires only function values and gradients of the objective, and because it has been shown in [14] to be efficient for solving neural network problems. To deal with the nonconvexity of the objective, we made slight changes of the original L-BFGS, following an idea in [15]. Denoting by $\boldsymbol{s}_t$ the difference between the iterates at iterations t and $t+1$, and by $\boldsymbol{y}_t$ the difference between the gradients at these two iterations, the pair $(\boldsymbol{s}_t, \boldsymbol{y}_t)$ is not used in computing subsequent search directions if $\boldsymbol{s}_t^T \boldsymbol{y}_t \ll \boldsymbol{s}_t^T \boldsymbol{s}_t$. This strategy ensures that the Hessian approximation remains positive definite, so the search directions generated by L-BFGS will be descent directions.

We solve the group-regularized problem (6) using SpaRSA [12], a proximal-gradient method that requires only the gradient of f and an efficient proximal solver for the regularization term. As shown in [12], the proximal problem associated with the group-LASSO regularization has a closed form solution that is inexpensive to compute.

In the next section, we discuss details of two bus selection approaches, and how to compute the gradient of f efficiently.

B. Two Approaches for PMU Location

We follow [6] in proposing two approaches for selecting PMU locations. In the first approach, we set I in (6) to be the full set of potential PMU locations, and try different values of the parameter τ until we find a solution that has the desired number of nonzero submatrices $(W_1)_{\cdot j}$ for $j \in I$, which indicate the chosen PMU locations.

The second approach is referred to as the "greedy heuristic" in [6]. We initialize I to be the set of candidate locations for PMUs. (We can exclude from this set locations that are already instrumented with PMUs and those that are not to be considered as possible PMU locations.) We then minimize (6) with this I, and select the index s that satisfies

$$s = \arg\max_{s\in I} \quad r(W_1, G_s)$$

as the next PMU location. This s is removed from I, and we minimize (6) with the reduced I. This process is repeated until the required number of locations has been selected. The process is summarized in Algorithm 1.

C. Computing the Gradient of the Loss Function

In both SpaRSA and the modified L-BFGS algorithm, the gradient and the function value of f defined in (3) are needed at every iteration. We show how to compute these two values efficiently given any iterate $W = (W_1, W_2, \ldots, W_{N+1})$. Function values are computed exactly as suggested by the constraints in (3), by evaluating the intermediate quantities $\boldsymbol{x}_i^j$, $j \in [N+1]$, $i \in [n]$ by these formulas, then finally the summation in (3a). The gradient involves an adjoint calculation. By applying the chain rule to the constraints in (3), treating $\boldsymbol{x}_i^j$, $j \in [N+1]$, as variables alongside $W_1, W_2, \ldots, W_{N+1}$, we obtain

$$\nabla_{W_{N+1}} f = \sum_{i=1}^{n} \nabla_{\boldsymbol{x}_i^{N+1}} \ell(\boldsymbol{x}_i^{N+1}, y_i)(\boldsymbol{x}_i^N)^T + \epsilon W_{N+1}, \quad (7a)$$

$$\nabla_{\boldsymbol{x}_i^N} f = \nabla_{\boldsymbol{x}_i^{N+1}} \ell(\boldsymbol{x}_i^{N+1}, y_i) W_{N+1}^T, \quad (7b)$$

$$\nabla_{\boldsymbol{x}_i^j} f = \nabla_{\boldsymbol{x}_i^{j+1}} f \cdot \sigma'(W_{j+1}\boldsymbol{x}_i^j) W_{j+1}^T, \quad (7c)$$
$$j = N-1, \ldots, 0,$$

$$\nabla_{W_j} f = \sum_{i=1}^{n} \nabla_{\boldsymbol{x}_i^j} f \cdot \sigma'(W_j \boldsymbol{x}_i^{j-1})(\boldsymbol{x}_i^{j-1})^T + \epsilon W_j, \quad (7d)$$
$$j = 1, \ldots, N.$$

Since σ is a pointwise operator that maps $\mathbf{R}^{d_i}$ to $\mathbf{R}^{d_i}$, $\sigma'(\cdot)$ is a diagonal matrix such that $\sigma'(\boldsymbol{z})_{i,i} = \sigma'(z_i)$. The quantities

$\sigma'()$ and $\boldsymbol{x}_i^j$, $j = 1, 2, \ldots, N+1$ are computed and stored during the calculation of the objective. Then, from (7b) and (7c), the quantities $\nabla_{\boldsymbol{x}_i^j} f$ from $j = N, N-1, \ldots, 0$ can be computed in a reverse recursion. Finally, the formulas (7d) and (7a) can be used to compute the required derivatives $\nabla_{W_j} f$, $j = 1, 2, \ldots, N+1$.

D. Training and Validation Procedure

In accordance with usual practice in statistical analysis involving regularization parameters, we divide the available data into a *training set* and a *validation set*. The training set is a randomly selected subset of the available data — the pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, 2, \ldots, n$ in the notation above — that is used to form the objective function whose solution yields the parameters $W_1, W_2, \ldots, W_{N+1}$ in the neural network. The validation set consists of further pairs $(\boldsymbol{x}_i, y_i)$ that aid in the choice of the regularization parameter, which in our case is the parameter τ in the greedy heuristic procedure of Algorithm 1, described in Sections III-A and III-B. We apply the greedy heuristic for $\tau \in \{2^{-8}, 2^{-7}, \ldots, 2^7, 2^8\}$ and deem the optimal value to be the one that achieves the most accurate outage identification on the validation set. We select initial points for the training randomly, so different solutions $W_1, W_2, \ldots, W_{N+1}$ may be obtained even for a single value of τ. To obtain a "score" for each value of τ, we choose the best result from ten random starts. The final model is then obtained by solving (2) over the buses selected on the best of the ten validation runs, that is, fixing the elements of W_1 that correspond to non-selected buses at zero.

Note that validation is not needed to choose the value of τ when we solve the regularized problem (6) directly, because in this procedure, we adjust τ until a predetermined number of buses is selected.

There is also a *testing set* of pairs $(\boldsymbol{x}_i, y_i)$. This is data that is used to evaluate the bus selections produced by the procedures above. In each case, the tuned models obtained on the selected buses are evaluated on the testing set.

IV. Experiments

We perform simulations based on grids from the IEEE test set archive [16]. Many of our studies focus on the IEEE-57bus case. Simulations of grid response to varying demand and outage conditions are performed using MATPOWER [17]. We first show that high accuracy can be achieved easily when PMU readings from all buses are used. We then focus on the more realistic (but more difficult) case in which data from only a limited number of PMUs is used. In both cases, we simulate PMU readings over a wide range of power demand profiles that encompass the profiles that would be seen in practice over different seasons and at different times of day.

A. Data Generation

We use the following procedure from [6] to generate the data points using a stochastic process and MATPOWER.

1. We consider the full grid defined in the IEEE specification, and also the modified grid obtained by removing each transmission line in turn.
2. For each demand node, define a baseline demand value from the IEEE test set archive as the average of the load demand over 24 hours.
3. To simulate different "demand averages" for different seasons, we scale the baseline demand value for each node by the values in $\{0.5, 0.75, 1, 1.25, 1.5\}$, to yield five different baseline demand averages for each node. (Note: In [6], a narrower range of multipliers was used, specifically $\{0.85, 1, 1.15\}$, but each multiplier is considered as a different independent data set.)
4. Simulate a 24-hour fluctuation in demand by an adaptive Ornstein-Uhlenbeck process as suggested in [18], independently and separately on each demand bus.
5. This fluctuation is overlaid on the demand average for each bus to generate a 24-hour load demand profile.
6. Obtain training, validation, and test points from these 24-hour demand profiles for each node by selecting different timepoints from this 24-hour period, as described below.
7. If any combination of line outage and demand profile yields a system for which MATPOWER cannot identify a feasible solution for the AC power flow equations, we do not add this point to the data set. Lines connecting the same pair of buses are considered as a single line; we take them to be all disconnected or all connected.

This procedure was used to generate training, validation, and test data. In each category, we generated equal numbers of training points for each feasible case in each of the five scale factors $\{0.5, 0.75, 1, 1.25, 1.5\}$. For each feasible topology and each combination of parameters above, we generate 20 training points from the first 12 hours of the 24-hour simulation period, and 10 validation points and 50 test points from the second 12-hour period. Summary information about the IEEE power systems we use in the experiments with single line outage is shown in Table I. The column "Feas." shows the number of lines whose removal still result in a feasible topology for at least one scale factor, while the number of lines whose removal result in infeasible topologies for all scale factors or are duplicated is indicated in the column "Infeas./Dup." The next three columns show the number of data points in the training / validation / test sets. As an example: The number of training points for the 14-Bus case (which is 1840) is approximately 19 (number of feasible line removals) times 5 (number of demand scalings) times 20 (number of training points per configurations). The difference between this calculated value of 1900 and the 1840 actually used is from that the numbers of feasible lines under different scaling factors are not identical, and higher scaling factors resulted in more infeasible cases. The last column in Table I shows the number of components in each feature vector $\boldsymbol{x}_i$. There are two features for each bus, being changes in phase angle and voltage magnitude with respect to the original grid under the same demand conditions. There are another two additional features in all cases, one indicating the power generation level (expressed as a fraction of the long-term average), and the other one indicating a bias term manually added to the data.

TABLE I: The systems used in our experiment and statistics of the synthetic data.

System	#lines		#Train	#Val	#Test	#Features
	Feas.	Infeas./Dup.				
14-Bus	19	1	1,840	920	4,600	30
30-Bus	38	3	3,680	1,840	9,200	62
57-Bus	75	5	5,340	2,670	13,350	116
118-Bus	170	16	16,980	8,490	42,450	238

B. Neural Network Design

Configuration and design of the neural network is critical to performance in many applications. In most of our experiments, we opt for a simple design in which there is just a single hidden layer: $N = 1$ in the notation of (2). We assume that the matrices W_1 and W_2 are dense, that is, all nodes in any one layer are connected to all nodes in adjacent layers. It remains to decide how many nodes d_1 should be in the hidden layer. Larger values of d_1 lead to larger matrices W_1 and W_2 and thus more parameters to be chosen in the training process. However, larger d_1 can raise the possibility of overfitting the training data, producing solutions that perform poorly on the other, similar data in the validation and test sets.

We did an experiment to indicate whether overfitting could be an issue in this application. We set $d_1 = 200$, and solved the unregularized training problem (2) using the modified L-BFGS algorithm with $50,000$ iterations. Figure 1 represents the output of each of the 200 nodes in the hidden layer for each of the $13,350$ test examples. Since the output is a result of the $\tanh$ transformation (1) of the input, it lies in the range $[-1, 1]$. We color-code the outputs on a spectrum from red to blue, with red representing 1 and blue representing -1. A significant number of columns are either solid red or solid blue. The hidden-layer nodes that correspond to these columns play essentially no role in distinguishing between different outages; similar results would be obtained if they were simply omitted from the network. The presence of these nodes indicates that the training process avoids using all d_1 nodes in the hidden layer, if fewer than d_1 nodes suffice to attain a good value of the training objective. Note that overfitting is avoided at least partially because we stop the training procedure with a rather small number of iterations, which can be viewed as another type of regularization [19].

In our experiments, we used $d_1 = 200$ for the larger grids (57 and 114 buses) and $d_1 = 100$ for the smaller grids (14 and 30 buses). The maximum number of L-BFGS iterations for all neural networks is set to $50,000$, while for MLR models we terminate it either when the number of iterations reaches $500,000$ or when the gradient is smaller than a pre-specified value (10^{-3} in our experiments), as linear models do not suffer much from overfitting.

C. Results on All Buses

We first compare the results between linear multinomial logistic regression (MLR) (as considered in [6]) and a fully connected neural network with one hidden layer, where the PMUs are placed on all buses. Because we use all the buses, no validation phase is needed, because the parameter τ does not appear in the model. Table II shows error rates on the testing set. We see that in the difficult cases, when the linear model has error rates higher than 1%, the neural network obtains markedly better testing error rates.

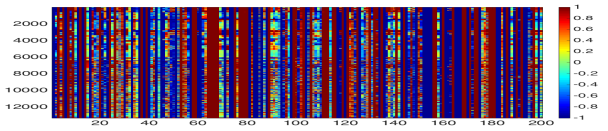


Fig. 1: Output of the hidden layer nodes of a one-layer neural network with 200 hidden nodes applied to the problem of detecting line outages on the IEEE 57-bus grid. Columns with a single color (dark red or dark blue) indicate nodes that output the same value regardless of the feature vector $\boldsymbol{x}_i$ that was input into the neural network. Such nodes play little or no role in discriminating between different line outages.

TABLE II: PMUs on all buses: Test error rates for single-line outage.

Buses	14	30	57	118
Linear MLR	0.00%	1.76%	4.50%	15.19%
Neural network	0.43%	0.03%	0.91%	2.28%

D. Results on Subset of Buses

We now focus on the 57-bus case, and apply the greedy heuristic (Algorithm 1) to select a subset of buses for PMU placement, for the neural network with one hidden layer of 200 nodes. We aim to select 10 locations. Figure 2 shows the locations selected at each run. Values of τ used were $\{2^{-8}, 2^{-7}, \ldots, 2^{8}\}$, with ten runs performed for each value of τ. On some runs, the initial point is close to a bad local optimum (or saddle point) and the optimization procedure terminates early with fewer than 10×2 columns of non-zeros in W_1 (indicating that fewer than 10 buses were selected, as each bus corresponds to 2 columns). The resulting models have poor performance, and we do not include them in the figure.

Even though the random initial points are different on each run, the groups selected for a fixed τ tend to be similar on all runs when $\tau \leq 2$. For larger values of τ, including the value $\tau = 2^4$ which gives the best selection performance, the locations selected on different runs are often different. (For the largest values of τ, fewer than 10 buses are selected.)

Table III shows testing accuracy for the ten PMU locations selected by both the greedy heuristic and regularized optimization with a single well-chosen value of τ. Both the neural network and the linear MLR classifiers were tried. The groups of selected buses are shown for each case. These differ significantly; we chose the "optimal" group from among these to be the one with the best validation score. We note the very specific choice of τ for linear MLR (group-LASSO). In this case, the number of groups selected is extremely sensitive to τ. In a very small range around $\tau = 14.4898999$, the number of buses selected varies between 8 and 12. We report two types of

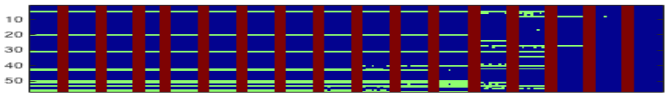


Fig. 2: Groups selected on the 57-bus case for different runs and different values of τ in the greedy heuristic applied on the neural network problem (6). Each row represents a group and each column represents a run. Ten runs are plotted for each value of τ. From left to right (separated by brown vertical lines), these values are $\tau = 2^{-8}, 2^{-7}, \ldots, 2^{8}$. Green indicates selected groups; dark blue are groups not selected.

error rates here. In the column "Err. (top1)" we report the rate at which the outage that was assigned the highest probability by the classifier was not the outage that actually occurred. In "Err. (top2)" we score an error only if the true outage was not assigned either the highest or the second-highest probability by the classifier. We note here that "top1" error rates are much higher than when PMU data from all buses is used, although that the neural network yields significantly better results than the linear classifier. However, "top2" results are excellent for the neural network when the greedy heuristic is used to select bus location.

Table IV repeats the experiment of Table III, but for 14 selected buses rather than 10. Again, we see numerous differences between the subsets of buses selected by the greedy and group-LASSO approaches, for both the linear MLR and neural networks. The neural network again gives significantly better test error rates than the linear MLR classifier, and the "top2" results are excellent for the neural network, for both group-LASSO and greedy heuristics. Possibly the most notable difference with Table III is that the buses selected by the group-LASSO network for the neural network gives much better results for 14 buses than for 10 buses. However, since it still performs worse than the greedy heuristic, the group-LASSO approach is not further considered in later experiments.

E. Why Do Neural Network Models Achieve Better Accuracy?

Reasons for the impressive effectiveness of neural networks in certain applications are poorly understood, and are a major research topic in machine learning. For this specific problem, we compare the distribution of the raw feature vectors with the distribution of feature vectors obtained after transformation by the hidden layer. The goal is to understand whether the transformed vectors are in some sense more clearly separated and thus easier to classify than the original data.

We start with some statistics of the clusters formed by feature vectors of the different classes. For purposes of discussion, we denote $\boldsymbol{x}_i$ as the feature vector, which could be the full set of PMU readings, the reduced set obtained after selection of a subset of PMU locations, or the transformed feature vector obtained as output from the hidden layer, according to the context. For each $j \in \{1, 2, \ldots, K\}$, we gather all those feature vectors $\boldsymbol{x}_i$ with label $y_i = j$, and denote the centroid of this cluster by c_j. We track two statistics: the mean / standard deviation of the distance of feature vectors $\boldsymbol{x}_i$ to their cluster centroids, that is, $\|\boldsymbol{x}_i - c_{y_i}\|$ for $i = 1, 2, \ldots, n$; and the mean / standard deviation of distances between cluster centroids, that is, $\|c_j - c_k\|$ for $j, k \in \{1, 2, \ldots, K\}$. We analyze these statistics for three cases, all based on the IEEE 57-Bus network: first, when $\boldsymbol{x}_i$ are vectors containing full PMU data; second, when $\boldsymbol{x}_i$ are vectors containing the PMU data from the 10 buses selected by the Greedy heuristic; third, the same data vectors as in the second case, but after they have been transformed by the hidden layer of the neural network.

Results are shown in Table V. For the raw data (first and second columns of the table), the distances within clusters are typically smaller than distances between centroids. (This happens because the feature vectors within each class are "strung out" rather than actually clustered, as we see below.) For the transformed data (last column) the clusters are generally tighter and more distinct, making them easier to distinguish.

Visualization of the effects of hidden-layer transformation is difficult because of the high dimensionality of the feature vectors. Nevertheless, we can gain some insight by projecting into two-dimensional subspaces that correspond to some of the leading principal components, which are the vectors obtained from the singular value decomposition of the matrix of all feature vectors $\boldsymbol{x}_i$, $i = 1, 2, \ldots, n$. Figure 3 shows two graphs. Both show training data for the same 5 line outages for the IEEE 57-Bus data set, with each class coded by a particular color and shape. In both graphs, we show data vectors obtained after 10 PMU locations were selected with the Greedy heuristic. In the left graph, we plot the coefficients of the first and fifth principal components of each data vector. The "strung out" nature of the data for each class reflects the nature of the training data. Recall that for each outage / class, we selected 20 points from a 12-hour period of rising demand, at 5 different scalings of overall demand level. For the right graph in Figure 3, we plot the coefficients of the first and third principal components of each data vector *after* transformation by the hidden layer. For both graphs, we have chosen the two principal components to plot to be those for which the separation between classes is most evident. For the left graph (raw data), the data for classes 3, 4, and 5 appear in distinct regions of space, although the border between classes 4 and 5 is thin. For the right graph (after transformation), classes 3, 4, and 5 are somewhat more distinct. Classes 1 and 2 are difficult to separate in both plots, although in the right graph, they no longer overlap with the other three classes. The effects of tighter clustering and cleaner separation after transformation, which we noted in Table V, are evident in the graphs of Figure 3.

F. Double-Line Outage Detection

We now extend our identification methodology to detect not just single-line outages, but also outages on two lines simultaneously. The number of classes that our classifier needs to distinguish between now scales with the *square* of the number of lines in the grid, rather than being approximately

TABLE III: Comparison of different approaches for selecting 10 buses on the IEEE 57-bus case, after $50,000$ iterations for neural networks and $500,000$ iterations for linear MLR models.

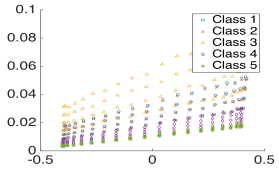
Model	τ	Buses selected	Err. (top1)	Err. (top2)
Linear MLR (greedy)	2	[5 16 20 31 40 43 44 51 53 57]	29.7%	8.4%
Neural Network (greedy)	16	[5 20 31 40 43 50 51 53 54 57]	7.1%	0.1%
Linear MLR (group-LASSO)	14.4898999	[2 4 5 6 7 8 18 27 28 29]	54.4%	39.4%
Neural Network (group-LASSO)	48	[4 5 6 7 8 18 26 27 28 55]	24.1%	12.9%

TABLE IV: Comparison of different approaches for selecting 14 buses on the IEEE 57-bus case, after $50,000$ iterations for neural networks and $500,000$ iterations for linear MLR models.

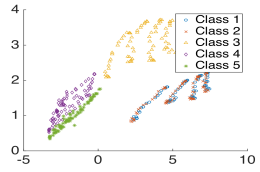
Model	τ	Buses selected	Err. (top1)	Err. (top2)
Linear MLR (greedy)	2	[5 16 17 20 26 31 39 40 43 44 51 53 54 57]	21.8%	3.8%
Neural Network (greedy)	16	[5 6 16 24 27 31 39 40 42 50 51 52 53 54]	5.2%	0.3%
Linear MLR (group-LASSO)	13	[2 4 5 7 8 17 18 27 28 29 31 32 33 34]	42.1%	25.3%
Neural Network (group-LASSO)	44	[4 7 8 18 24 25 26 27 28 31 32 33 39 40]	6.2%	0.6%

TABLE V: Instance distribution before and after neural network transformation for the IEEE 57-bus data set. In the last two columns, 10 buses are selected by the Greedy heuristic.

	Full PMU Data	Selected PMUs	Selected PMUs, after neural network transformation
mean $\pm$ std dev. distance to centroid	0.30 ± 0.14	0.27 ± 0.12	2.30 ± 1.01
mean $\pm$ std dev. between-centroid distance	0.17 ± 0.14	0.08 ± 0.05	3.27 ± 1.10



(a) Original feature space after bus selection (the 1st and 5th principal axes)



(b) The feature space after neural network transformation (the 1st and 3rd principal axes)

Fig. 3: Data representation after dimension reduction to 2D. Different colors/styles represent data points of different labels.

TABLE VI: Statistics of the synthetic data for double lines outage.

System	#classes	#Train	#Val	#Test	#Features
14-Bus	182	16,420	8,210	41,050	30
30-Bus	715	66,160	33,080	165,400	62

equal to the number of lines. For this much larger number of classes, we generate data in the manner described in Section IV-A, again omitting cases where the outage results in an infeasible network. Table VI shows the number of classes for the 14- and 30-bus networks, along with the number of training / validation / test points. Note in particular that there are 182 distinct outage events for the 14-bus system, and 715 distinct events for the 30-bus system.

Table VII shows results of our classification approaches for

TABLE VII: Error rates of placing PMUs on all buses for double lines outage.

	14-bus	30-bus
Linear MLR	26.07%	36.32%
Neural network with one hidden layer	0%	0.65%

the case in which PMU observations are made at all buses. The neural network model has a single hidden layer of 100 nodes. The neural network has dramatically better performance than the linear MLR classifier on these problems, attaining a zero error rate on the 14-bus tests.

We repeat the experiment using a subset of buses chosen with the greedy heuristic described in Section III-B — 3 buses for the 14-bus network and 5 buses for the 30-bus network. Given the low dimensionality of the feature space and the large number of classes, these are difficult problems. (Because it was shown in the previous experiments that the group-LASSO approach has inferior performance to the greedy heuristic, we omit it from this experiment.) As we see in Table VIII, the linear MLR classifiers do not give good results, with "top1" and "top2" error rates all in excess of 71%. Much better results are obtained for neural network with bus selection performed by the greedy heuristic, which obtains "top2" error rates of less than 1% in the 14-bus case and 5.6% in the 30-bus case.

V. Conclusions

This work describes the use of neural networks to detect single- and double-line outages from PMU data on a power grid. We show significant improvements in classification performance over the linear multiclass logistic regression methods described in [6], particularly when data about the PMU signatures of different outage events is gathered over a wide range of demand conditions. By adding regularization to the model, we can determine the locations to place a limited number of PMUs in a way that optimizes classification performance. Our approach uses a high-fidelity AC model of the grid to generate data examples that are used to train the neural-network classifier. Although (as is true in most applications of neural networks) the training process is computationally heavy, the predictions can be obtained with minimal computation, allowing the model to be deployed in real time.

TABLE VIII: Comparison of different approaches for sparse PMU placement for double outages detection.

Case	Number of PMU	Model	τ	Buses selected	Err. (top1)	Err. (top2)
14-bus	3	Linear MLR (greedy)	8	[3 5 14]	83.0%	71.7%
		Neural Network (greedy)	8	[3 12 13]	4.3%	0.9%
30-bus	5	Linear MLR (greedy)	0.5	[4 5 17 23 30]	90.6%	84.5%
		Neural Network (greedy)	8	[5 14 19 29 30]	12.7%	5.6%

REFERENCES

[1] H. Zhu and G. B. Giannakis, "Sparse overcomplete representations for efficient identification of power line outages," *IEEE Transactions on Power Systems*, vol. 27, no. 4, pp. 2215–2224, Nov. 2012.

[2] J.-C. Chen, W.-T. Li, C.-K. Wen, J.-H. Teng, and P. Ting, "Efficient identification method for power line outages in the smart power grid," *IEEE Transactions on Power Systems*, vol. 29, no. 4, pp. 1788–1800, Jul. 2014.

[3] J. E. Tate and T. J. Overbye, "Line outage detection using phasor angle measurements," *IEEE Transactions on Power Systems*, vol. 23, no. 4, pp. 1644–1652, Nov. 2008.

[4] ——, "Double line outage detection using phasor angle measurements," in *2009 IEEE Power & Energy Society General Meeting*, Calgary, AB, Jul. 2009, pp. 1–5.

[5] A. Y. Abdelaziz, S. F. Mekhamer, M. Ezzat, and E. F. El-Saadany, "Line outage detection using Support Vector Machine (SVM) based on the Phasor Measurement Units (PMUs) technology," in *2012 IEEE Power and Energy Society General Meeting*, San Diego, CA, Jul. 2012, pp. 1–8.

[6] T. Kim and S. J. Wright, "PMU placement for line outage identification via multinomial logistic regression," *IEEE Transactions on Smart Grid*, vol. PP, no. 99, 2016.

[7] Y. Zhao, J. Chen, and H. V. Poor, "Efficient neural network architecture for topology identification in smart grid," in *Signal and Information Processing (GlobalSIP), 2016 IEEE Global Conference on*. IEEE, 2016, pp. 811–815.

[8] M. Garcia, T. Catanach, S. Vander Wiel, R. Bent, and E. Lawrence, "Line outage localization using phasor measurement data in transient state," *IEEE Transactions on Power Systems*, vol. 31, no. 4, pp. 3019–3027, 2016.

[9] Y. Zhao, J. Chen, and H. V. Poor, "A learning-to-infer method for real-time power grid topology identification," Tech. Rep., 2017, arXiv:1710.07818.

[10] D. Malioutov, M. Cetin, and A. S. Willsky, "A sparse signal reconstruction perspective for source localization with sensor arrays," *IEEE transactions on signal processing*, vol. 53, no. 8, pp. 3010–3022, 2005.

[11] L. Meier, S. Van De Geer, and P. Bühlmann, "The group LASSO for logistic regression," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 70, no. 1, pp. 53–71, 2008.

[12] S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo, "Sparse reconstruction by separable approximation," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2479–2493, 2009.

[13] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, 1989.

[14] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 265–272.

[15] D.-H. Li and M. Fukushima, "On the global convergence of the BFGS method for nonconvex unconstrained optimization problems," *SIAM Journal on Optimization*, vol. 11, no. 4, pp. 1054–1064, 2001.

[16] "Power systems test case archive," 2014, [Online]. Available: http://www.ee.washington.edu/research/pstca/.

[17] 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 Transactions on power systems*, vol. 26, no. 1, pp. 12–19, 2011.

[18] M. Perninge, V. Knazkins, M. Amelin, and L. Söder, "Modeling the electric power consumption in a multi-area system," *European Transactions on Electrical Power*, vol. 21, no. 1, pp. 413–423, 2011.

[19] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in Neural Information Processing Systems*, 2001, pp. 402–408.

[20] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.

[21] J. D. Pearson, "Variable metric methods of minimisation," *The Computer Journal*, vol. 12, no. 2, pp. 171–178, 1969.

APPENDIX

A. Introduction and Implementation of the SpaRSA Algorithm

We solve the nonsmooth regularized problem (6) by SpaRSA [12], a proximal gradient algorithm. When applied to (6), iteration t of SpaRSA solves the following problem, for some scalar $\alpha_t > 0$:

$$W^{t+1} := \arg\min_W \frac{1}{2}\left\|W - \left(W^t - \frac{1}{\alpha_t}\nabla f(W^t)\right)\right\|_F^2 + \frac{\tau}{\alpha_t}c(W_1^t, I), \tag{8}$$

where $W^t := [W_1^t, \ldots, W_{N+1}^t]$ denotes the tth iterate of W. By utilizing the structure of $c(\cdot, I)$ in (5), we can solve (8) inexpensively, in closed form. For the value of α_t, at any given iteration t, we follow the suggestion in [12] to start at a certain guess, and gradually increase it until the solution of (8) satisfies

$$f(W^{t+1}) < f(W^t) - \frac{\sigma}{2}\alpha_t\|W^{t+1} - W^t\|_F^2, \tag{9}$$

for some small positive value of σ (typically $\sigma = 10^{-3}$).

B. Key Lemmas for Convergence Analysis

We now analyze the convergence guarantee for SpaRSA applied to (6). First, we establish bounds on the gradient and Hessian of f. We do not restrict using (1) as the choice of σ. Instead, we only require that σ is twice-continuously differentiable.

Lemma 1. *Given any initial point W^0, there exists $c_1 \geq 0$ such that*

$$\|\nabla f(W)\| \leq c_1 \tag{10}$$

in the level set $\{W \mid f(W) \leq f(W^0)\}$.

Proof. Because the loss function ℓ defined by (4) is nonnegative, we see from (3) that

$$f(W) \geq \frac{1}{2}\epsilon\|W\|_F^2,$$

and therefore $\{W \mid f(W) \leq f(W^0)\}$ is a subset of

$$B\left(0, \sqrt{\frac{2}{\epsilon}f(W^0)}\right) := \left\{W \mid \|W\|_F \leq \sqrt{\frac{2}{\epsilon}f(W^0)}\right\}, \tag{11}$$

which is a compact set. By the assumption on σ and ℓ, $\|\nabla f(W)\|$ is a continuous function with respect to W. Therefore, we can find $c_1 \geq 0$ such that (10) holds within the set (11). Since the level set is a subset of (11), (10) holds with the same value of c_1 within the level set. □

Lemma 2. *Given any initial point W^0, and any $c_2 \geq 0$, there exists $L_{c_2} > 0$ such that $\|\nabla^2 f(W)\| \leq L_{c_2}$ in the set $\{W + \boldsymbol{p} \mid f(W) \leq f(W^0),\ \|\boldsymbol{p}\| \leq c_2\}$.*

Proof. Clearly, from the argument in the proof for Lemma 1, $\{W + \boldsymbol{p} \mid f(W) \leq f(W^0),\ \|\boldsymbol{p}\| \leq c_2\}$ is a subset of the compact set

$$B\left(0, \sqrt{\frac{2}{\epsilon}f(W^0)} + c_2\right).$$

Therefore, as a continuous function with respect to W, $\|\nabla^2 f(W)\|$ achieves its maximum L_{c_2} in this set. □

Now we provide a convergence guarantee for the SpaRSA algorithm.

Theorem 1. *All accumulation points generated by SpaRSA are stationary points.*

Proof. We will show that the conditions of [12, Theorem 1] are satisfied, and thus the result follows. This theorem states that if the acceptance condition is

$$f(W^{t+1}) \leq \max_{i=\max(t-M,0),\ldots,t} f(W^i) - \frac{\sigma\alpha_t}{2}\|W^{t+1} - W^t\|_F^2 \tag{12}$$

for some nonnegative integer M and some $\sigma \in (0, 1)$, f is Lipschitz continuously differentiable, the regularizer c defined in (5a) is convex and finite-valued, and $L_I(W)$ of (6) is lower-bounded, then all accumulation points are stationary. Clearly, (9) implies the acceptance condition (12), with $M = 0$, and the conditions on $c(W_1, I)$ and $L_I(W)$ are easily verified. It remains only to check Lipschitz continuity of ∇f. Because the condition (9) ensures that it is a descent method, all iterates lie in the set $\{W \mid f(W) \leq f(W^0)\}$. Thus, by Lemma 1, f has Lipschitz continuous gradient within this range. Hence all conditions of Theorem 1 in [12] are satisfied, and the result follows. □

C. Overview of L-BFGS

Before describing our modified L-BFGS algorithm for solving the smooth problem (3) obtained after bus selection, we introduce the original L-BFGS method, following the description from [20, Section 7.2]. Consider the problem

$$\min_{W \in \mathbf{R}^d} \quad f(W),$$

where f is twice-continuously differentiable. At iterate W^t, L-BFGS constructs a symmetric positive definite matrix B_t to approximate $\nabla^2 f(W^t)^{-1}$, and the search direction $\boldsymbol{d}_t$ is obtained as

$$\boldsymbol{d}_t = -B_t \nabla f(W^t). \tag{13}$$

Given an initial estimate B_t^0 at iteration t and a specified integer $m \geq 0$, we define $m(t) = \min(m, t)$ and construct the matrix B_t as follows for $t = 1, 2, \ldots$:

$$B_t := V_{t-1}^T \cdots V_{t-m(t)}^T B_t^0 V_{t-m(t)} \cdots V_{t-1} + \rho_{t-1} s_{t-1} s_{t-1}^T + \sum_{j=t-m(t)}^{t-2} \rho_j V_{t-1}^T \cdots V_{j+1}^T s_j s_j^T V_{j+1} \cdots V_{t-1}, \tag{14}$$

where for $j \geq 0$, we define

$$V_j := I - \rho_j \boldsymbol{y}_j \boldsymbol{s}_j^T, \quad \rho_j := \frac{1}{\boldsymbol{y}_j^T \boldsymbol{s}_j},$$
$$\boldsymbol{s}_j := W^{j+1} - W^j, \quad \boldsymbol{y}_j := \nabla f(W^{j+1}) - \nabla f(W^j). \tag{15}$$

The initial matrix B_t^0, for $t \geq 1$, is commonly chosen to be

$$B_t^0 = \frac{\boldsymbol{y}_{t-1}^T \boldsymbol{s}_{t-1}}{\boldsymbol{y}_{t-1}^T \boldsymbol{y}_{t-1}} I.$$

At the first iteration $t = 0$, one usually takes $B_0 = I$, so that the first search direction $\boldsymbol{d}_0$ is the steepest descent direction $-\nabla f(W^0)$. After obtaining the update direction $\boldsymbol{d}_t$, L-BFGS conducts a line search procedure to obtain a step size η_t satisfying certain conditions, among them the "sufficient decrease" or "Armijo" condition

$$f(W^t + \eta_t \boldsymbol{d}_t) \leq f(W^t) + \eta_t \gamma \nabla f(W^t)^T \boldsymbol{d}_t, \qquad (16)$$

where $\gamma \in (0, 1)$ is a specified parameter. We assume that the steplength η_t satisfying (16) is chosen via a backtracking procedure. That is, we choose a parameter $\beta \in (0, 1)$, and set η_t to the largest value of β^i, $i = 0, 1, \ldots$, such that (16) holds.

Note that we use vector notation for such quantities as $\boldsymbol{d}_t$, $\boldsymbol{y}_j$, $\boldsymbol{s}_j$, although these quantities are actually matrices in our case. Thus, to compute inner products such as $\boldsymbol{y}_t^T \boldsymbol{s}_t$, we first need to reshape these matrices as vectors.

D. A Modified L-BFGS Algorithm

The key to modifying L-BFGS in a way that guarantees convergence to a stationary point at a provable rate lies in designing the modifications so that inequalities of the following form hold, for some positive scalar values of a, b, and $\bar{b}$, and for all vector $\boldsymbol{s}_t$ and $\boldsymbol{y}_t$ defined by (15) that are used in the update of the inverse Hessian approximation B_t:

$$a\|\boldsymbol{s}_t\|^2 \leq \boldsymbol{y}_t^T \boldsymbol{s}_t \leq b\|\boldsymbol{s}_t\|^2, \qquad (17)$$

$$\frac{\boldsymbol{y}_t^T \boldsymbol{y}_t}{\boldsymbol{y}_t^T \boldsymbol{s}_t} \leq \bar{b}. \qquad (18)$$

The average value of the Hessian over the step from W^t to $W^t + \boldsymbol{s}_t$ plays a role in the analysis; this is defined by

$$\bar{H}_t := \int_0^1 \nabla^2 f(W^t + t\boldsymbol{s}_t) dt. \qquad (19)$$

When f is strongly convex and twice continuously differentiable, no modifications are needed: L-BFGS with backtracking line search can be shown to converge to the unique minimal value of f at a global Q-linear rate. In this case, the properties (17) and (18) hold when we set a to be the global (strictly positive) lower bound on the eigenvalues of $\nabla^2 f(W)$ and b and $\bar{b}$ to be the global upper bound on these eigenvalues. Analysis in [13] shows that the eigenvalues of B_t are bounded inside a strictly positive interval, for all t.

In the case of f twice continuously differentiable, but possibly nonconvex, we modify L-BFGS by skipping certain updates, so as to ensure that the conditions (17) and (18) are satisfied. Details are given in the remainder of this section.

We note that conditions (17) and (18) are essential for convergence of L-BFGS not just theoretically but also empirically. Poor convergence behavior was observed when we applied the original L-BFGS procedure directly to the nonconvex 4-layer neural network problem in Section G.

Similar issues regarding poor performance on nonconvex problems are observed when the full BFGS algorithm is used to solve nonconvex problems. (The difference between L-BFGS and BFGS is that for BFGS, in (14), m is always set to t and B_t^0 is a fixed matrix independent of t.) To ensure convergence of BFGS for nonconvex problems, [15] proposed to update the inverse Hessian approximation only when we are certain that its smallest eigenvalue after the update is lower-bounded by a specified positive value. In particular, those pairs $(\boldsymbol{y}_j, \boldsymbol{s}_j)$ for which the following condition holds: $\tilde{\epsilon}\|\boldsymbol{s}_j\|^2 > \boldsymbol{y}_j^T \boldsymbol{s}_j$ (for some fixed $\tilde{\epsilon} > 0$) are not used in the update formula (14).) Here, we adapt this idea to L-BFGS, by replacing the indices $t - m(t), \ldots, t - 1$ used in the update formula (14) by a different set of indices $i_1^t, \ldots, i_{\hat{m}(t)}^t$ such that $0 \leq i_1^t \leq \ldots \leq i_{\hat{m}(t)}^t \leq t - 1$, which are the latest $\hat{m}(t)$ iteration indices (up to and including iteration $t-1$) for which the condition

$$\boldsymbol{s}_j^T \boldsymbol{y}_j \geq \tilde{\epsilon} \boldsymbol{s}_j^T \boldsymbol{s}_j, \qquad (20)$$

is satisfied. (We define $\hat{m}(t)$ to be the minimum between m and the number of pairs that satisfy (20).) Having determined these indices, we define B_t by

$$B_t := V_{i_{\hat{m}(t)}^t}^T \cdots V_{i_1^t}^T B_t^0 V_{i_1^t} \cdots V_{i_{\hat{m}(t)}^t} + \rho_{i_{\hat{m}(t)}^t} \boldsymbol{s}_{i_{\hat{m}(t)}^t} \boldsymbol{s}_{i_{\hat{m}(t)}^t}^T + \sum_{j=1}^{\hat{m}(t)-1} \rho_{i_j^t} V_{i_{\hat{m}(t)}^t}^T \cdots V_{i_{j+1}^t}^T \boldsymbol{s}_{i_j^t} \boldsymbol{s}_{i_j^t}^T V_{i_{j+1}^t} \cdots V_{i_{\hat{m}(t)}^t}, \qquad (21)$$

and

$$B_t^0 = \frac{\boldsymbol{y}_{i_{\hat{m}(t)}^t}^T \boldsymbol{s}_{i_{\hat{m}(t)}^t}}{\boldsymbol{y}_{i_{\hat{m}(t)}^t}^T \boldsymbol{y}_{i_{\hat{m}(t)}^t}} I. \qquad (22)$$

(When $\hat{m}(t) = 0$, we take $B_t = I$.) We show below that, using this rule and the backtracking line search, we have

$$\min_{i=0,1,\ldots,t} \|\nabla f(W^i)\| = O(t^{-1/2}). \qquad (23)$$

With this guarantee, together with compactness of the level set (see the proof of Lemma 1) and that all the algorithm is a descent method so that all iterates stay in this level set, we can prove the following result.

Theorem 2. *Either we have $\nabla f(W^t) = 0$ for some t, or else there exists an accumulation point $\hat{W}$ of the sequence $\{W^t\}$ that is stationary, that is, $\nabla f(\hat{W}) = 0$.*

Proof. Suppose that $\nabla f(W^t) \neq 0$ for all t. We define a subsequence $\mathcal{S}$ of $\{W^t\}$ as follows:

$$\mathcal{S} := \{\hat{t} : \|\nabla f(W^{\hat{t}})\| < \|\nabla f(W^s)\|, \ \ \forall s = 0, 1, \ldots, \hat{t} - 1\}.$$

This subsequence is infinite, since otherwise we would have a strictly positive lower bound on $\|\nabla f(W^t)\|$, which contradicts (23). Moreover, (23) implies that $\lim_{t \in \mathcal{S}} \|\nabla f(W^t)\| = 0$. Since $\{W^t\}_{t \in \mathcal{S}}$ all lie in the compact level set, this subsequence has an accumulation point $\hat{W}$, and clearly $\nabla f(\hat{W}) = 0$, proving the claim. □

E. Proof of the Gradient Bound

We now prove the result (23) for the modified L-BFGS method applied to (2). The proof depends crucially on showing that the bounds (17) and (18) hold for all vector pairs $(\boldsymbol{s}_j, \boldsymbol{y}_j)$ that are used to define B_t in (21).

Theorem 3. *Given any initial point W^0, using the modified L-BFGS algorithm discussed in Section D to optimize* (2)*, then there exists $\delta > 0$ such*

$$1 \geq \frac{-\nabla f(W^t)^T \boldsymbol{d}_t}{\|\nabla f(W^t)\|\|\boldsymbol{d}_t\|} \geq \delta, \quad t = 0, 1, 2, \ldots . \tag{24}$$

Moreover, there exist M_1, M_2 with $M_1 \geq M_2 > 0$ such that

$$M_2\|\nabla f(W^t)\| \leq \|\boldsymbol{d}_t\| \leq M_1\|\nabla f(W^t)\|, \quad t = 0, 1, 2, \ldots . \tag{25}$$

Proof. We first show that for all $t > 0$, the following descent condition holds:

$$f(W^t) \leq f(W^{t-1}), \tag{26}$$

implying that

$$W^t \in \{W \mid f(W) \leq f(W^0)\}. \tag{27}$$

To prove (26), for the case that $t = 0$ or $\hat{m}(t) = 0$, it is clear that $\boldsymbol{d}_t = -\nabla f(W^t)$ and thus the condition (16) guarantees that (26) holds. We now consider the case $\hat{m}(t) > 0$. From (21), since (20) guarantees $\rho_{i_j^t} \geq 0$ for all j, we have that B_t is positive semidefinite. Therefore, (13) gives $\nabla f(W^t)^T \boldsymbol{d}_t \leq 0$, which together with (16) implies (26).

Next, we will show (17) and (18) hold for all pairs $(\boldsymbol{s}_j, \boldsymbol{y}_j)$ with $j = i_1^t, \ldots, i_{\hat{m}(t)}^t$. The left inequality in (17) follows directly from (20), with $a = \tilde{\epsilon}$. We now prove the right inequality of (17), along with (18). Because (27) holds, we have from Lemma 2 that $\bar{H}_t$ defined by (19) satisfies

$$\|\bar{H}_t\| \leq L_{c_2}, \quad t = 0, 1, 2, \ldots . \tag{28}$$

From $\boldsymbol{y}_t = \bar{H}_t \boldsymbol{s}_t$, (28), and (20), we have for all t such that (20) holds that

$$\frac{\|\boldsymbol{y}_t\|^2}{\boldsymbol{y}_t^T \boldsymbol{s}_t} \leq \frac{L_{c_2}^2 \|\boldsymbol{s}_t\|^2}{\boldsymbol{y}_t^T \boldsymbol{s}_t} \leq \frac{L_{c_2}^2}{\tilde{\epsilon}}, \tag{29}$$

which is exactly (18) with $\bar{b} = L_{c_2}^2/\tilde{\epsilon}$.

From $\boldsymbol{y}_t = \bar{H}_t \boldsymbol{s}_t$, the Cauchy-Schwarz inequality, and (28), we get

$$\boldsymbol{y}_t^T \boldsymbol{s}_t \leq \|\boldsymbol{y}_t\|\|\boldsymbol{s}_t\| \leq \|\boldsymbol{s}_t\|\|\bar{H}_t\|\|\boldsymbol{s}_t\| = L_{c_2}\|\boldsymbol{s}_t\|^2,$$

proving the right inequality of (17), with $b = L_{c_2}$.

Now that we have shown that (17) and (18) hold for all indices $i_1^t, \ldots, i_{\hat{m}(t)}^t$, we can follow the proof in [13] to show that there exist $M_1 \geq M_2 > 0$ such that

$$M_1 I \succeq B_t \succeq M_2 I, \quad \text{for all } t. \tag{30}$$

The rest of the proof is devoted to showing that this bound holds. Having proved this bound, the results (25) and (24) (with $\delta = M_2/M_1$) follow directly from the definition (13) of $\boldsymbol{d}_t$.

To prove (30), we first bound B_t^0 defined in (22). This bound will follow if we can prove a bound on $\boldsymbol{y}_t^T \boldsymbol{s}_t / \|\boldsymbol{y}_t\|^2$ for all t satisfying (20). Clearly when $\hat{m}(t) = 0$, we have $B_t^0 = I$, so there are trivial lower and upper bounds. For $\hat{m}(t) > 0$, (18) implies a lower bound of $1/\bar{b} = \tilde{\epsilon}/L_{c_2}^2$. For an upper bound, we have from (20) that

$$\tilde{\epsilon}\|\boldsymbol{s}_j\|^2 \leq \|\boldsymbol{s}_j\|\|\boldsymbol{y}_j\| \quad \Rightarrow \quad \|\boldsymbol{s}_j\| \leq \frac{1}{\tilde{\epsilon}}\|\boldsymbol{y}_j\|.$$

Hence, from (22), we have

$$\left\|B_t^0\right\| = \frac{\left|\boldsymbol{y}_{i_{\hat{m}(t)}^t}^T \boldsymbol{s}_{i_{\hat{m}(t)}^t}\right|}{\left\|\boldsymbol{y}_{i_{\hat{m}(t)}^t}\right\|^2} \leq \frac{\left\|\boldsymbol{s}_{i_{\hat{m}(t)}^t}\right\|}{\left\|\boldsymbol{y}_{i_{\hat{m}(t)}^t}\right\|} \leq \frac{1}{\tilde{\epsilon}}.$$

Now we will prove the results by working on the inverse of B_t. Following [13], the inverse of B_t can be obtained by

$$\begin{aligned} H_t^{(0)} &= (B_t^0)^{-1}, \\ H_t^{(k+1)} &= H_t^{(k)} - \frac{H_t^{(k)} \boldsymbol{s}_{i_k^t} \boldsymbol{s}_{i_k^t}^T H_t^{(k)}}{\boldsymbol{s}_{i_k^t}^T H_t^{(k)} \boldsymbol{s}_{i_k^t}} \\ &\qquad + \frac{\boldsymbol{y}_{i_k^t} \boldsymbol{y}_{i_k^t}^T}{\boldsymbol{y}_{i_k^t}^T \boldsymbol{s}_{i_k^t}}, \quad k = 0, \ldots, \hat{m}(t) - 1, \\ B_t^{-1} &= H_t^{\hat{m}(t)}. \end{aligned} \tag{31}$$

Therefore, we can bound the trace of B_t^{-1} by using (29).

$$\begin{aligned} \operatorname{tr}(B_t^{-1}) &\leq \operatorname{tr}((B_t^0)^{-1}) + \sum_{k=0}^{\hat{m}(t)-1} \frac{\boldsymbol{y}_{i_k^t}^T \boldsymbol{y}_{i_k^t}}{\boldsymbol{y}_{i_k^t}^T \boldsymbol{s}_{i_k^t}} \\ &\leq \operatorname{tr}((B_t^0)^{-1}) + \hat{m}(t)\frac{L_{c_2}^2}{\tilde{\epsilon}}. \end{aligned} \tag{32}$$

This together with the fact that B_t^{-1} is positive-semidefinite and that B_t^0 is bounded imply that there exists $M_2 > 0$ such that

$$\|B_t^{-1}\| \leq \operatorname{tr}(B_t^{-1}) \leq M_2^{-1},$$

which implies that $B_t \succeq M_2 I$, proving the right-hand inequality in (30). (Note that this upper bound for the largest eigenvalue also applies to $H_t^{(k)}$ for all $k = 0, 1, \ldots, \hat{m}(t) - 1$.)

For the left-hand side of (30), we have from the formulation for (31) in [13] (see [21] for a derivation) and the upper bound $\|H_t^{(k)}\| \leq M_2$ that

$$\begin{aligned} \det(B_t^{-1}) &= \det((B_t^0)^{-1}) \prod_{j=0}^{\hat{m}(t)-1} \frac{\boldsymbol{y}_{i_k^t}^T \boldsymbol{s}_{i_k^t}}{\boldsymbol{s}_{i_k^t}^T \boldsymbol{s}_{i_k^t}} \frac{\boldsymbol{s}_{i_k^t}^T \boldsymbol{s}_{i_k^t}}{\boldsymbol{s}_{i_k^t}^T H_t^{(k)} \boldsymbol{s}_{i_k^t}} \\ &\geq \det((B_t^0)^{-1}) \left(\frac{\tilde{\epsilon}}{M_2}\right)^{\hat{m}(t)} \\ &\geq \bar{M}_1^{-1}, \end{aligned}$$

for some $\bar{M}_1 > 0$. Since the eigenvalues of B_t^{-1} are upper-bounded by M_2^{-1}, it follows from the positive lower bound on $\det(B_t^{-1})$ that these eigenvalues are also lower-bounded by a positive number. The left-hand side of (30) follows. □

Corollary 1. *Given any initial point W^0, if we use the algorithm discussed in Section D to solve* (2)*, then the bound* (23) *holds for the norms of the gradients at the iterates $W^0, W^1, \ldots$.*

Proof. First, we lower-bound the step size obtained from the backtracking line search procedure. Consider any iterate W^t and the generated update direction $\boldsymbol{d}_t$ by the algorithm discussed in Section D. From Theorem 3, we have

$$\|\boldsymbol{d}_t\|^2 \leq M_1\|\boldsymbol{d}_t\|\|\nabla f(W^t)\| \leq -\frac{M_1}{\delta}\nabla f(W^t)^T\boldsymbol{d}_t.$$

Thus by using Taylor's theorem, and the uniform upper bound on $\|\nabla^2 f(W)\|$ in the level set defined in Lemma 2, we have for any value of η

$$\begin{aligned} f(W^t+\eta\boldsymbol{d}_t) &\leq f(W^t)+\eta\nabla f(W^t)^T\boldsymbol{d}_t+\frac{L_{c_2}\eta^2}{2}\|\boldsymbol{d}_t\|^2 \\ &\leq f(W^t)+\eta\nabla f(W^t)^T\boldsymbol{d}_t\left(1-\frac{L_{c_2}M_1\eta}{2\delta}\right). \end{aligned}$$

Therefore, since $\nabla f(W^t)^T\boldsymbol{d}_t<0$, (16) holds whenever

$$1-\eta\frac{L_{c_2}M_1}{2\delta}\geq\gamma \quad\Leftrightarrow\quad \eta\leq\bar{\eta}:=\frac{2(1-\gamma)\delta}{L_{c_2}M_1}.$$

Because the backtracking mechanism decreases the candidate stepsize by a factor of $\beta\in(0,1)$ at each attempt, it will "undershoot" $\bar{\eta}$ by at most a factor of β, so we have

$$\eta_t\geq\min(1,\beta\bar{\eta}),\quad \text{for all } t. \tag{33}$$

From (16), Theorem 3, and (33) we have that

$$\begin{aligned} f(W^{t+1}) &\leq f(W^t)+\eta_t\gamma\nabla f(W^t)^T\boldsymbol{d}_t \\ &\leq f(W^t)-\eta_t\gamma\delta\|\nabla f(W^t)\|\|\boldsymbol{d}_t\| \\ &\leq f(W^t)-\eta_t\gamma\delta M_2\|\nabla f(W^t)\|^2 \\ &\leq f(W^t)-\hat{\eta}\|\nabla f(W^t)\|^2. \end{aligned} \tag{34}$$

where $\hat{\eta}:=\min(1,\beta\bar{\eta})\gamma\delta M_2$. Summing (34) over $t=0,1,\ldots,k$, we get

$$\begin{aligned} \min_{0\leq t\leq k}\|\nabla f(W^t)\|^2 &\leq \frac{1}{k+1}\sum_{t=0}^{k}\|\nabla f(W^t)\|^2 \\ &\leq \frac{1}{k+1}\frac{1}{\hat{\eta}}\sum_{t=0}^{k}(f(W^t)-f(W^{t+1})) \\ &\leq \frac{1}{k+1}\frac{1}{\hat{\eta}}(f(W^0)-f(W^{k+1})) \\ &\leq \frac{1}{k+1}\frac{1}{\hat{\eta}}(f(W^0)-f^*)=O(1/k). \end{aligned}$$

where f^* is the optimal function value in (3), which is lower-bounded by zero. By taking square roots of both sides, the claim (23) follows. □

F. Neural Network Initialization

Initialization of the neural network training is not trivial. The obvious initial point of $W_j=0$, $j\in[N+1]$ has $\nabla_{W_j}f=0$, $j\in[N+1]$ (as can be seen via calculations with (3), (1), and (7)), so is likely a saddle point. A gradient-based step will not move away from such a point. Rather, we start from a random point close to the origin. Following a suggestion from a well-known online tutorial,[2] we choose all elements of each W_j uniformly, randomly, and identically distributed from the interval $[-a\sqrt{6}/\sqrt{d_{j-1}+d_j}, a\sqrt{6}/\sqrt{d_{j-1}+d_j}]$, where $a=1$. We experimented with smaller values of a, when setting $a=1$ leads to slow convergence in the training error (which is an indicator that the initial point is not good enough), by starting with $a=10^{-t}$ for some non-negative integer t. We keep trying smaller t until either the convergence is fast enough, or the resulting solution has high training errors and the optimization procedure terminates early. In the latter case, we then set $t\leftarrow t+1$, choose new random points from the interval above for the new value of a, and repeat.

[2] http://deeplearning.net/tutorial/mlp.html#weight-initialization

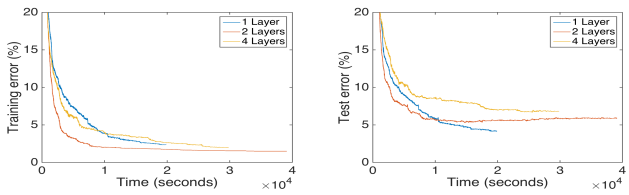
TABLE IX: Performance of different number of layers in the neural network model.

#layers	#variables	Test error	Training error
1	19,675	4.15%	2.36%
2	32,275	5.87%	1.50%
4	12,625	6.83%	2.02%

G. Additional Experiment on Using More Layers in the Neural Networks

We now examine the effects of adding more hidden layers to the neural network. As a test case, we choose the 57-bus case, with ten pre-selected PMU locations, at nodes $[1,2,17,19,26,39,40,45,46,57]$. (These were the PMUs selected by the greedy heuristic in [6, Table III].) We consider three neural network configurations. The first is the single hidden layer of 200 nodes considered above. The second contains two hidden layers, where the layer closer to the input has 200 nodes and the layer closer to the output has 100 nodes. The third configuration contains four hidden layers of 50 nodes each. For this last configuration, when we solved the training problem with L-BFGS, the algorithm frequently required modification to avoid negative-curvature directions. (In that sense, it showed greater evidence of nonconvexity.)

Figure 4 shows the training error and test error rates as a function of training time, for these three configurations. The total number of variables in each model is shown along with the final training and test error rates in Table IX. The training error ultimately achieved is smaller for the multiple-hidden-layer configurations than for the single hidden layer. However, the single hidden layer still has a slightly better test error. This suggests that the multiple-hidden-layer models may have overfit the training data. (A further indication of overfitting is that the test error increases slightly for the four-hidden-layer configuration toward the end of the training interval.) This test is not definitive, however; with a larger set of training data, we may find that the multiple-hidden-layer models give better test errors.



(a) Training error (b) Test error

Fig. 4: Comparison between 1, 2 and 4 layers. We show training and test error v.s. running time.