Chapter 1

Deep Learning Techniques for Solving Optimal Power Flow Problems

Vassilis Kekatos[1*] and Manish K. Singh[2]

[1] *Purdue University, Elmore Family School of Electrical and Computer Engineering, 47907, Indiana, West Lafayette, 465 Northwestern Ave, United States*
[2] *University of Wisconsin–Madison, Department of Electrical & Computer Engineering, 53706, Wisconsin, Madison, 1415 Engineering Drive, United States*

*Corresponding Author: kekatos@purdue.edu

Abstract: Distribution grids are currently challenged by the rampant integration of distributed energy resources (DERs). Scheduling DERs via an optimal power flow (OPF) problem in real-time and at scale under uncertainty is a formidable task. Prompted by the success of deep neural networks (DNNs) in other fields, this chapter presents two learning-based paradigms for near-optimal DER operation. The first paradigm engages a DNN to predict the solutions of an OPF given the anticipated demands and renewable generation. Different from the generic learning setup, the training dataset here (namely past OPF solutions) features a rich yet largely unexploited structure. To leverage prior information, we train a DNN to match not only the OPF solutions but also their partial derivatives with respect to the input parameters of the OPF. Sensitivity analyses for different OPF

formulations show how such derivatives can be readily computed from the OPF solutions. The proposed sensitivity-informed training of DNNs features sample efficiency improvements at a modest computational increase. Nonetheless, this two-stage OPF-then-learn approach may not be suitable for DER operation when the OPF problem parameters are uncertain. To deal with stochastic OPF setups, we put forth an alternative OPF-and-learn scheme. Here the DNN is not trained to mimic labeled OPF data but is rather organically integrated into a stochastic OPF formulation. The DNN now captures a DER control policy that minimizes average costs subject to average or chance network constraints. A key advantage of this second DNN is that it can be driven by partial OPF inputs or proxies derived from available real-time measurements. Both paradigms apply to power distribution and transmission systems alike, in their exact AC or linearized variants.

Keywords: Optimal power flow (OPF), deep neural networks (DNN), distributed energy resources (DERs), sensitivity analysis, chance-constrained optimization, conic programming, optimality conditions, linear independence constraint qualification (LICQ)

1.1. Introduction

To enable higher integration of renewables and combat climate change, there is an urgent need to advance the efficiency and scalability of the optimal power flow (OPF) problem. Despite advances in numerical optimization, OPF solvers oftentimes fail to respond frequently enough to provide up-to-date solutions in rapidly changing or uncertain environments. Spurred by the field-changing performance of deep learning in various application domains, this chapter puts forth novel physics-aware and application-cognizant learning approaches for the OPF. Operators must routinely solve some rendition of the OPF to opti-

mally dispatch generators in transmission systems or to compute the optimal setpoints of distributed energy resources (DER) in power distribution grids. Assuming the grid topology to remain fixed, each instance of the OPF corresponds to a different set of grid loading conditions, such as solar/wind generation and or load demands. The OPF essentially constitutes a *parametric optimization problem* with the grid conditions being the problem parameters. In turn, the OPF solutions can be seen as a complex mapping from the problem parameters to the space of optimal setpoints. A key promise for deep learning is its ability to capture intricate input-output mappings.

In spite of the commendable advancements in optimization algorithms, capturing the OPF mapping through machine learning models (ML) has been an active research area over the last few years. The primary advantage of such approaches lies in the speed-up during the inference phase by moving most of the computational burden from real-time to offline. For instance, compared to conventional solvers, deep learning-based approaches have offered speed-ups by factors as high as 200 for the so-termed direct current OPF (DC-OPF), and 35 for the AC-OPF [53, 38]. There are two main challenges central to learning for OPF. First, traditional deep learning approaches are not amenable to enforcing constraints even for the training set. Predictions for OPF minimizers may have limited standalone value if the related constraints are violated. Second, power systems undergo frequent topological and operational changes, while problem parameters such as loads and renewable generation are oftentimes uncertain and modeled as random processes.

To cope with the first challenge, a deep neural network (DNN) may be engaged to better initialize existing numerical solvers [49]; or to predict active constraints and hence, result in an OPF model with fewer constraints [20, 14,

16]. Another group of approaches targets feasibility of the DNN predictions by penalizing constraint violations and the related Karush–Kuhn–Tucker (KKT) conditions [39, 38], or explicitly resorting to a Lagrangian dual scheme for DNN training [19, 25, 48]. The third alternative involves post-processing DNN predictions by projecting them using a power flow solver [39, 49]. Although the projected point satisfies the power flow equations, it may still violate engineering limits. Regarding the second challenge, sample efficiency could be improved by judiciously selecting the DNN architecture upon leveraging prior information. For example, by seeking an input-convex DNN when the underlying input-output mapping is convex [51], or using DNNs that *unroll* an iterative optimization algorithm [50], or adopting graph-based priors [37]. Finally, references [52] and [46] have adopted penalty functions from reinforcement learning to account for uncertainties while satisfying constraints.

This chapter reviews two learning-to-OPF methodologies that address the aforementioned challenges in distinct and creative ways. The first methodology falls under the *OPF-then-Learn* category and speeds up training by improving data efficiency for deterministic OPF problems. The second methodology falls under the *OPF-and-Learn* category and waives the need for generating labeled OPF data for stochastic OPF problems. The two categories and our novel methodologies are outlined next.

OPF-then-learn trains a DNN or other ML model to predict the OPF solutions once presented with the OPF parameters at its input (Section 1.2). OPF-then-learn typically involves two steps: *s1)* Generating a training dataset by solving a large number of OPF instances, and *s2)* Training the DNN. Step *s2)* iteratively updates the DNN weights to minimize the distance between the DNN output and the OPF minimizer over all training examples. While *s2)*

is standard in the general ML setup, step *s1)* of generating training data is quite unique to the learning-to-optimize paradigm. Nonetheless, step *s1)* can be time-consuming, especially when the DNN has to be trained afresh due to changes in grid topology or loading statistics. To expedite *s1)*, we have proposed modifying *s2)* so that the DNN can learn the OPF mapping with fewer training examples [41]. Along with the OPF minimizer, we exploit the fact when an OPF instance is solved, one can also compute the *sensitivity* (that is the partial derivatives) of the minimizer with respect to the problem parameters. Then, the DNN can be trained to match not only the OPF minimizer but also its sensitivities to the problem parameters. This sensitivity-informed approach can improve data efficiency by up to an order of magnitude, and thus, reduce the computational cost of *s1)* at a modest increase in the time needed for *s2)*. This allows the DNN to be retrained faster under topological, operational, or distributional shifts. As a byproduct of independent interest, the chapter further explains how OPF sensitivities can be computed by simply solving a set of linear equations. Our findings simplify prior technical conditions on the existence of OPF sensitivities.

Our Learn-then-OPF approach has been applied to different OPF setups:

a) Dispatching DERs to minimize losses while respecting voltage constraints under the linearized distribution flow (LDF) model in distribution grids [40].

b) Dispatching DERs to minimize losses while respecting voltage constraints using an AC-OPF for radial grids posed as a second-order cone program (SOCP) [27].

c) Dispatching generators to minimize the cost of generation while enforcing network and generator limits under the exact AC power flow model

in transmission systems [41]. OPF sensitivities were computed for different formulations of the AC-OPF, namely: *i)* a non-convex quadratically constrained quadratic program (QCQP); *ii)* semidefinite programming (SDP) convex relaxation; and *iii)* the default formulation of MATPOWER.

Sensitivity-informed learning for OPF can be applied to other learning models. To showcase this generality, we trained Gaussian Process (GP)-based models to learn the OPF mapping under setup *b)*. Interestingly, sensitivity information can be neatly incorporated into GP-based learning [27]. Although GPs may have less representation capabilities relative to DNNs, GP training is simpler and GP predictions come with uncertainty quantification in the form of confidence intervals. To elucidate the technical details of sensitivity computation and its inclusion in DNN training, a quadratic programming (QP) formulation for setup *a)* and a QCQP formulation for setup *c)* will be considered under Sections 1.2.1 and 1.2.2, respectively.

OPF-and-Learn. The second part of this chapter utilizes DNNs to tackle OPF problems where grid conditions are only stochastically known (Section 1.3). In such setups, the operator would like to find an optimal dispatch policy instead of a single minimizer. A policy determines the setpoint for a generator or DER once presented with the actual grid conditions experienced in real-time. A policy can be encoded by a DNN to capture the mapping between grid conditions and dispatch decisions as in the deterministic OPF setup discussed earlier. Nonetheless, as grid conditions are not known a priori, the policy must be optimal in a stochastic sense across all anticipated conditions. A policy can be found by solving one OPF involving multiple loading scenarios. The cost function is averaged over all scenarios, while constraints are enforced either

on the average value or with high probability across all scenarios. We suggest solving this scenario-based OPF using our OPF-and-Learn approach. The DNN-based policy is learned not in two steps as in Learn-then-OPF, but in a single step without the need of training OPF labels. The training process updates the DNN weights not to fit training examples, but to minimize a Lagrangian function including the original OPF cost and constraint functions. Primal and dual variables are updated on a per-scenario basis using stochastic primal-dual updates. The OPF-and-Learn approach has been adopted under various setups including a linearized and an AC-OPF for dispatching DERs in distribution grids [25, 22], and an AC-OPF for dispatching generators in transmission systems [21]. Interestingly, if the DNN is designed to have a particular structure, the OPF-and-Learn can be adapted to optimally design Volt/VAR control rules as pursued in [24, 23, 47]. Section 1.3 outlines the general principles of the OPF-and-Learn approach under the AC-OPF setup for dispatching DERs in distribution grids.

1.2. Sensitivity-Informed Learning for OPF

Power system operators have to solve some rendition of the OPF to optimally dispatch generators in transmission systems, or distributed energy resources (DERs) in distribution grids. The OPF is an optimization problem that aims at minimizing a cost function constrained by the engineering limitations imposed by generation units and the power network and subject to power-flow physics. The OPF has to be solved repeatedly every time the *inputs or parameters* to the problem change. The use of the term *parameter* for inputs is motivated by the parametric optimization literature which particularly focuses on settings

where a given optimization problem needs to be solved for changing problem inputs. The term *parameter* in this chapter shall not be confused with known and fixed system quantities, such as generation limits or line impedances.

Such parameters of the OPF could be the load demand for active and reactive power, or the available solar generation at every bus of the power system. The OPF can be thus be interpreted as a *parametric* optimization problem, and be abstracted as follows: Given a vector $\boldsymbol{\theta} \in \mathbb{R}^P$ of problem parameters, find an optimal dispatch $\mathbf{x}(\boldsymbol{\theta}) \in \mathbb{R}^N$ as the minimizer

$$\begin{aligned} \mathbf{x}(\boldsymbol{\theta}) \in \arg\min_{\mathbf{x}} \;\; & f(\mathbf{x};\boldsymbol{\theta}) \\ \text{s.to} \;\; & \mathbf{h}(\mathbf{x};\boldsymbol{\theta}) = \mathbf{0} : \;\; \boldsymbol{\lambda}_{\boldsymbol{\theta}} \\ & \mathbf{g}(\mathbf{x};\boldsymbol{\theta}) \leq \mathbf{0} : \;\; \boldsymbol{\mu}_{\boldsymbol{\theta}} \end{aligned} \tag{$P_{\boldsymbol{\theta}}$}$$

where functions $f(\mathbf{x};\boldsymbol{\theta})$, $\mathbf{h}(\mathbf{x};\boldsymbol{\theta})$, and $\mathbf{g}(\mathbf{x};\boldsymbol{\theta})$ are continuously differentiable with respect to $\mathbf{x}$ and $\boldsymbol{\theta}$; and vectors $(\boldsymbol{\lambda}_{\boldsymbol{\theta}}, \boldsymbol{\mu}_{\boldsymbol{\theta}})$ collect the optimal dual variables or Lagrange multipliers corresponding to the equality and inequality constraints, respectively. The power system network topology is assumed to be known and remains unchanged over time.

To save on running time and computational resources, rather than solving $(P_{\boldsymbol{\theta}})$ for each $\boldsymbol{\theta}$, one can adopt a *learning-to-optimize approach*, according to which a DNN (or other machine learning model) can be trained to predict approximate solutions of $(P_{\boldsymbol{\theta}})$; see e.g., [42]. The DNN can be trained to output a predictor $\hat{\mathbf{x}}(\boldsymbol{\theta};\mathbf{w})$ of $\mathbf{x}(\boldsymbol{\theta})$ when presented with parameter vector $\boldsymbol{\theta}$ at its input. The DNN is parameterized by weights $\mathbf{w}$, which can be selected upon minimizing a suitable distance metric or *loss function* between $\mathbf{x}_{\boldsymbol{\theta}}$ and $\hat{\mathbf{x}}(\boldsymbol{\theta};\mathbf{w})$ over a training dataset.

Given a DNN architecture, the typical approach for learning-to-optimize entails two steps:

s1) Generate a labeled training dataset $\{\boldsymbol{\theta}_s, \mathbf{x}_s\}_{s=1}^S$ by solving $(P_{\boldsymbol{\theta}})$ for S representative scenarios parameterized by $\boldsymbol{\theta}_s$. Here we use the shorthand notation $\mathbf{x}_s := \mathbf{x}(\boldsymbol{\theta}_s)$; and

s2) Learn DNN weights $\mathbf{w}$ by minimizing a data fitting loss over the training dataset such as

$$\min_{\mathbf{w}} \sum_{s=1}^{S} \|\mathbf{x}_s - \hat{\mathbf{x}}_s(\mathbf{w})\|_2^2 . \tag{1.1}$$

We use again the shorthand notation $\hat{\mathbf{x}}_s(\mathbf{w}) := \hat{\mathbf{x}}(\boldsymbol{\theta}_s; \mathbf{w})$ for the DNN output when the DNN is fed with input $\boldsymbol{\theta}_s$ and parameterized by weights $\mathbf{w}$.

We refer to a DNN trained by solving (1.1) as a *plain DNN* or P-DNN for short. The conventional P-DNN approach focuses merely on dataset $\{\boldsymbol{\theta}_s, \mathbf{x}_s\}_{s=1}^S$, and ignores any additional properties the mapping $\boldsymbol{\theta} \to \mathbf{x}(\boldsymbol{\theta})$ induced by problem $(P_{\boldsymbol{\theta}})$ may have. Nonetheless, contrary to the typical ML setting, when learning from OPF data, there is plenty of side information at the learner's disposal, e.g., functions $(f, \mathbf{h}, \mathbf{g})$ are known, solvers usually return optimal dual variables along with the minimizer, and optimal primal/dual variables are known to satisfy optimality conditions.

One way to leverage such rich structure when learning from OPF data is to train the DNN to match not only the OPF minimizer $\mathbf{x}(\boldsymbol{\theta})$, but also its *sensitivities* (partial derivatives) with respect to $\boldsymbol{\theta}$. Sensitivity analysis of the OPF can readily compute the Jacobian matrix $\nabla_{\boldsymbol{\theta}} \mathbf{x}$, i.e., the $N \times P$ matrix carrying the partial derivatives of $\mathbf{x}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, assuming such sensitivities exist. To introduce some notation, suppose we solve $(P_{\boldsymbol{\theta}})$ for parameters $\boldsymbol{\theta}_s$,

and let $(\mathbf{x}_s, \boldsymbol{\lambda}_s, \boldsymbol{\mu}_s)$ denote the corresponding optimal primal/dual variables. We also introduce the shorthand notation $\mathbf{J}_s$ for the Jacobian matrix $\nabla_{\boldsymbol{\theta}_s}\mathbf{x}_s$; the discussion on how to compute $\mathbf{J}_s$ can be computed given $(\mathbf{x}_s, \boldsymbol{\lambda}_s, \boldsymbol{\mu}_s; \boldsymbol{\theta}_s)$ is postponed for later. To incorporate sensitivity information, the key idea here is to extend each training data pair $(\boldsymbol{\theta}_s, \mathbf{x}_s)$ to $(\boldsymbol{\theta}_s, \mathbf{x}_s, \mathbf{J}_s)$, and train the DNN using a loss function augmented by an additional fitting term as

$$\min_{\mathbf{w}} \sum_{s=1}^{S} \|\mathbf{x}_s - \hat{\mathbf{x}}_s(\mathbf{w})\|_2^2 + \rho\|\mathbf{J}_s - \hat{\mathbf{J}}_s(\mathbf{w})\|_F^2 \tag{1.2}$$

where $\rho > 0$ is a scalar weight and $\|\cdot\|_F$ denotes the matrix Frobenius norm. Matrix $\hat{\mathbf{J}}_s(\mathbf{w})$ is the Jacobian of the DNN output $\hat{\mathbf{x}}_s$ with respect to its input $\boldsymbol{\theta}_s$. We name the DNN trained as above a *sensitivity-informed DNN (SI-DNN)*.

When dealing with DNNs, one typically deals with the Jacobian $\nabla_{\mathbf{w}}\hat{\mathbf{x}}$ of the DNN output with respect to DNN weights $\mathbf{w}$, not with respect to its input $\boldsymbol{\theta}$. Such Jacobian can be computed efficiently thanks to automatic differentiation; a process known as *gradient back-propagation* in deep learning. The Jacobian $\nabla_{\mathbf{w}}\hat{\mathbf{x}}$ is used extensively while training the DNN, during which the weights are updated via stochastic gradient descent-type of algorithms and the sensitivities in $\nabla_{\mathbf{w}}\hat{\mathbf{x}}$ are naturally needed. Interestingly enough, contemporary deep learning libraries can compute $\nabla_{\boldsymbol{\theta}}\hat{\mathbf{x}}$ equally efficiently. Therefore, as long as $\{(\mathbf{x}_s, \mathbf{J}_s)\}_{s=1}^{S}$ are provided in the dataset, deep learning packages can readily handle (1.2); see Appendix A of [41] for implementation details.

Could sensitivity-informed learning have an edge over plain learning? To provide some intuition, recall that the mapping $\mathbf{x}(\boldsymbol{\theta})$ can be approximated by its first-order Taylor's series expansion around $\boldsymbol{\theta}$ as

$$\mathbf{x}(\boldsymbol{\theta}) \simeq \mathbf{x}(\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}_0}\mathbf{x} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

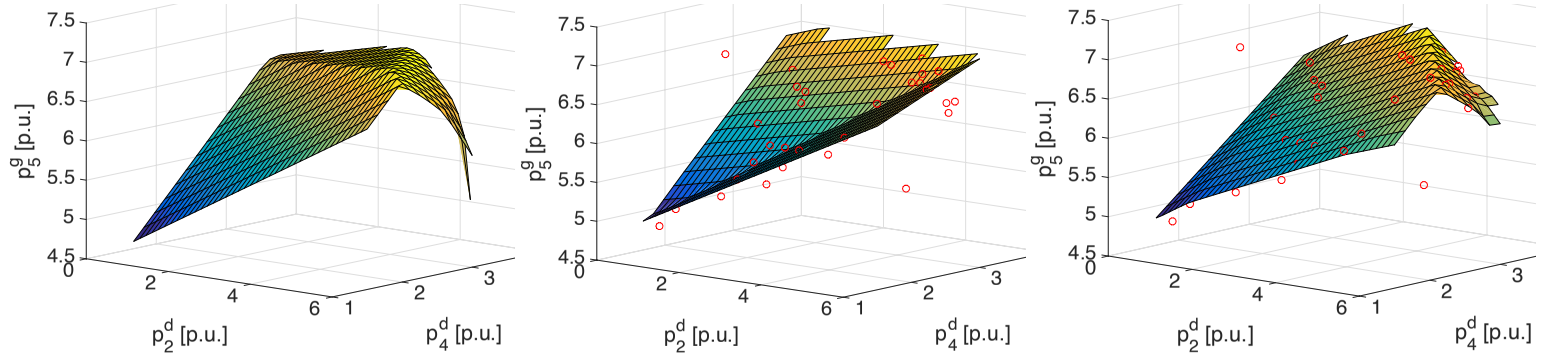

Figure 1.1: The left panel depicts the optimal generation dispatch $p_5^g(\boldsymbol{\theta})$ for bus 5 as a function of load demands at buses 2 and 4, that is $\boldsymbol{\theta} := [p_2^d \; p_4^d]^\top$. The mapping seems to have the shape of a kite. Sampling the parameter space of $\boldsymbol{\theta}$'s provided 523 feasible OPF instances, of which 37 instances constituted the training set. The center and right panels show the dispatches learned by a P-DNN and an SI-DNN. For the two DNNs, we used the same training points (red circles), architecture (two hidden layers with 16 neurons each), optimizer, and learning rates. The P-DNN fails to learn the kite-shaped mapping properly: the front of the kite is bent, while the kite's tail is grossly missed. On the contrary, the SI-DNN succeeds in reproducing a flat front for the kite and the drop in p_5^g for larger load demands.

granted again that $\nabla_{\boldsymbol{\theta}_0}\mathbf{x}$ exists. This approximation becomes more accurate for $\boldsymbol{\theta}$'s close to the linearization point $\boldsymbol{\theta}_0$. The same reasoning holds for the mapping $\hat{\mathbf{x}}(\boldsymbol{\theta})$ modeled by the DNN. Therefore, if the DNN is trained so that it approximates both $\mathbf{x}(\boldsymbol{\theta})$ and $\nabla_{\boldsymbol{\theta}}\mathbf{x}$ for each $\boldsymbol{\theta}_s$, it learns the OPF mapping not only at that particular $\boldsymbol{\theta}_s$, but also in a neighborhood around it; see also [41] for a stochastic interpretation. Obviously, if the learner has a large number of points S in the training dataset, sensitivity information may not be as useful since it involves inaccuracies due to linearization. Phrased differently, sensitivity-informed learning is advantageous when training a DNN using a relatively small dataset. Reference [1] provides an analytic study on how and when sensitivity information can improve learning. Figure 1.1 extracted from [41] provides a visual intuition on how sensitivity-informed learning can outperform plain learning.

Returning to the discussion on solving (1.2), the process is similar to training a standard DNN with the difference that now the learner needs to precompute the Jacobian matrices $\mathbf{J}_s$ that enrich the training dataset. As promised, these matrices can be found using sensitivity analysis of the OPF. We delineate this using two representative instances of the OPF: *i)* A linearized or so-termed DC OPF for a distribution grid yielding a quadratic program (Section 1.2.1); and *ii)* An AC-OPF for a power transmission system yielding a non-convex quadratically-constrained quadratic program (Section 1.2.2).

1.2.1. Learn-to-Optimize for DC-OPF

Several optimization tasks pertaining to power-system operations typically feature linearized versions of the power-flow equations in the constraints to contain

computational complexity. Since the engineering limits are oftentimes linear or can be reasonably linearized, the ensuing OPF formulations have a purely linear set of constraints. The commonly encountered objective functions are convex quadratics, if not linear. Such examples include net power loss, normed voltage deviations, or generator cost curves. Thus, *quadratic programs (QP)* can arguably be considered as the workhorse of OPF in practice.

1.2.1.1. QP-based OPF under a Linearized Grid Model

Suppose a system operator has to regularly solve the ensuing convex QP over the resource vector $\mathbf{x}$:

$$\mathbf{x}_{\boldsymbol{\theta}} := \arg\min_{\mathbf{x}} \ \tfrac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{x}^\top \mathbf{B}\boldsymbol{\theta} \tag{1.3a}$$

$$\text{s.to} \ \ \mathbf{C}\mathbf{x} \leq \mathbf{D}\boldsymbol{\theta} + \mathbf{e} : \quad \boldsymbol{\lambda}_{\boldsymbol{\theta}}. \tag{1.3b}$$

While $(\mathbf{A} \succ \mathbf{0}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{e})$ are kept fixed for some time, parameter vector $\boldsymbol{\theta}$ may be varying frequently. Let $(\mathbf{x}_{\boldsymbol{\theta}}, \boldsymbol{\lambda}_{\boldsymbol{\theta}})$ denote a pair of primal/dual solutions for a particular $\boldsymbol{\theta}$. Problem (1.3) is a *multiparametric QP (MPQP)*, whose solution space features neat properties long exploited by the control community [44, 11, 8]. Among other domains, the properties of MPQPs have been utilized in various power system applications, such as predicting prices [54, 29], and strategic investment [28] in wholesale electricity markets; security-constrained economic dispatch [33]; and hosting capacity analysis of distribution feeders [43].

The OPF application of interest here is dispatching inverters through an approximate OPF that minimizes power losses subject to voltage constraints. The approximation stems from using a linearized in lieu of the exact AC power flow model to capture ohmic losses and voltages [40]. Vector $\mathbf{x}$ comprises the

inverter setpoints for reactive power injection, while $\boldsymbol{\theta}$ carries the loading conditions (active and reactive loads, and active solar generation) across all buses. Problem (1.3) can become infeasible for some values of $\boldsymbol{\theta}$. For example, it may be impossible for the operator to maintain voltages within allowable limits under high solar penetration. To handle infeasible instances, it is customary to convert constraints from *hard* to *soft*. This is accomplished by adding a non-negative slack variable on the right-hand side of inequality constraints, and penalizing that variable heavily in the objective; see [43] for proper penalty functions.

Since $\boldsymbol{\theta}$ may change rapidly, an operator may not be able to solve (1.3) exactly in real-time for multiple feeders hosting hundreds or thousands of buses or inverters each. This motivates the need for a DNN to surrogate (1.3). As it is advantageous to include sensitivity information into the training of this DNN, we next delineate how to compute $\nabla_{\boldsymbol{\theta}}\mathbf{x}$ for the QP-based OPF in (1.3).

1.2.1.2. Sensitivity Analysis of QP-OPF

Consider solving (1.3) for a particular $\boldsymbol{\theta}$. The corresponding optimal primal-dual pair $(\mathbf{x}_{\boldsymbol{\theta}}, \boldsymbol{\lambda}_{\boldsymbol{\theta}})$ should satisfy the Karush-Kuhn-Tucker (KKT) optimality conditions [9]. Among the inequality constraints of (1.3), those satisfied with equality at the optimum are termed *active or binding constraints*. Let $(\tilde{\mathbf{C}}, \tilde{\mathbf{D}}, \tilde{\mathbf{e}}, \tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}})$ be the row partitions of $(\mathbf{C}, \mathbf{D}, \mathbf{e}, \boldsymbol{\lambda}_{\boldsymbol{\theta}})$ associated with active constraints. Likewise, let $(\bar{\mathbf{C}}, \bar{\mathbf{D}}, \bar{\mathbf{e}}, \bar{\boldsymbol{\lambda}}_{\boldsymbol{\theta}})$ denote the row partition related to *inactive or non-binding constraints*. Leveraging complementarity slackness, the KKT conditions can be expressed as [9]

$$\mathbf{A}\mathbf{x}_{\boldsymbol{\theta}} + \tilde{\mathbf{C}}^{\top}\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}} = \mathbf{B}\boldsymbol{\theta} \tag{1.4a}$$

$$\tilde{\mathbf{C}}\mathbf{x}_{\boldsymbol{\theta}} = \tilde{\mathbf{D}}\boldsymbol{\theta} + \tilde{\mathbf{e}} \tag{1.4b}$$

$$\bar{\mathbf{C}}\mathbf{x}_{\boldsymbol{\theta}} < \bar{\mathbf{D}}\boldsymbol{\theta} + \bar{\mathbf{e}} \tag{1.4c}$$

$$\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}} > \mathbf{0} \tag{1.4d}$$

$$\bar{\boldsymbol{\lambda}}_{\boldsymbol{\theta}} = \mathbf{0}. \tag{1.4e}$$

For (1.4d) in particular, it is assumed that binding constraints relate to strictly positive dual variables $\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}} > \mathbf{0}$. The reverse, that is having both the constraint and the Lagrange multiplier, gives rise to *degeneracy*. Degenerate instances of the OPF are unlikely to occur in practice when $\boldsymbol{\theta}$ is drawn at random. If they do occur, the OPF sensitivity for this specific $\boldsymbol{\theta}$ can be ignored and the DNN is trained only to match the OPF minimizer $\mathbf{x}_{\boldsymbol{\theta}}$.

If there are A active constraints, the equalities of (1.4a)–(1.4b) constitute a system of $(N + A)$ linear equations over the $(N + A)$ unknowns $(\mathbf{x}_{\boldsymbol{\theta}}, \tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}})$. Since $\mathbf{A} \succ \mathbf{0}$, the Lagrangian optimality condition of (1.4a) yields

$$\mathbf{x}_{\boldsymbol{\theta}} = \mathbf{A}^{-1}(\mathbf{B}\boldsymbol{\theta} - \tilde{\mathbf{C}}^{\top}\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}}). \tag{1.5}$$

Substituting (1.5) into (1.4b) provides

$$\mathbf{G}\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}} = (\tilde{\mathbf{C}}\mathbf{A}^{-1}\mathbf{B} - \tilde{\mathbf{D}})\boldsymbol{\theta} - \tilde{\mathbf{e}} \tag{1.6}$$

where $\mathbf{G} := \tilde{\mathbf{C}}\mathbf{A}^{-1}\tilde{\mathbf{C}}^{\top}$. If matrix $\tilde{\mathbf{C}}$ has linearly independent rows, then $\mathbf{G} \succ \mathbf{0}$, and hence, (1.6) has a unique solution. Otherwise, there are infinitely many $\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}}$'s satisfying (1.6) within a shift invariance on the nullspace $\text{null}(\tilde{\mathbf{C}}^{\top}) = \text{null}(\mathbf{G})$.

For a general QP, the rows of $\tilde{\mathbf{C}}$ are typically linearly independent, thus satisfying the so-termed *linear independence constraint qualification* (LICQ) [9].

Nevertheless, that is not the case for the inverter dispatch problem of $(P_{\boldsymbol{\theta}})$ where instances of linearly dependent active constraints occur frequently; see [40, 41] for examples with LICQ failing for various renditions of the OPF.

Regardless of whether $\tilde{\mathbf{C}}$ is full row-rank or not, the solution to (1.6) lies in the polyhedron

$$\tilde{\boldsymbol{\Lambda}}_{\boldsymbol{\theta}} := \{\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}} = \mathbf{G}^{\dagger}(\tilde{\mathbf{C}}\mathbf{A}^{-1}\mathbf{B} - \tilde{\mathbf{D}})\boldsymbol{\theta} - \mathbf{G}^{\dagger}\tilde{\mathbf{e}} + \mathbf{u}, \tilde{\mathbf{C}}^{\top}\mathbf{u} = \mathbf{0}\} \tag{1.7}$$

where $\mathbf{G}^{\dagger}$ denotes the pseudo-inverse of $\mathbf{G} = \tilde{\mathbf{C}}\mathbf{A}^{-1}\tilde{\mathbf{C}}^{\top}$. Substituting $\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}}$ back in (1.5) and exploiting $\tilde{\mathbf{C}}^{\top}\mathbf{u} = \mathbf{0}$, we get

$$\mathbf{x}_{\boldsymbol{\theta}} = \mathbf{J}_{\boldsymbol{\theta}}\boldsymbol{\theta} + \mathbf{A}^{-1}\tilde{\mathbf{C}}^{\top}\mathbf{G}^{\dagger}\tilde{\mathbf{e}}. \tag{1.8}$$

where the involved matrix is computed as

$$\mathbf{J}_{\boldsymbol{\theta}} := \mathbf{A}^{-1}\mathbf{B} - \mathbf{A}^{-1}\tilde{\mathbf{C}}^{\top}\mathbf{G}^{\dagger}(\tilde{\mathbf{C}}\mathbf{A}^{-1}\mathbf{B} - \tilde{\mathbf{D}}). \tag{1.9}$$

Although there may be infinitely many $\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}}$'s satisfying the KKT conditions, the optimal primal solution of (1.3) is unique if it exists. This is not surprising since (1.3) has a strictly convex objective. Moreover, the solution can be expressed as an *affine function* of $\boldsymbol{\theta}$. Note that the parameters of this affine function depend on the set of active constraints, and this is indicated by the subscript $\boldsymbol{\theta}$ on $\mathbf{J}_{\boldsymbol{\theta}}$.

Given (1.7), because the triplet $(\boldsymbol{\theta}, \mathbf{x}_{\boldsymbol{\theta}}, \tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}})$ should also satisfy conditions (1.4d) and (1.4c), there exists a $\mathbf{u} \in \text{null}(\tilde{\mathbf{C}}^{\top})$ so that $\boldsymbol{\theta}$ satisfies

$$\mathbf{G}^{\dagger}(\tilde{\mathbf{C}}\mathbf{A}^{-1}\mathbf{B} - \mathbf{D})\boldsymbol{\theta} > \mathbf{G}^{\dagger}\tilde{\mathbf{e}} - \mathbf{u} \tag{1.10a}$$

$$(\bar{\mathbf{C}}\mathbf{J}_{\boldsymbol{\theta}} - \bar{\mathbf{D}})\boldsymbol{\theta} < \bar{\mathbf{e}} - \bar{\mathbf{C}}\mathbf{A}^{-1}\tilde{\mathbf{C}}^{\top}\mathbf{G}^{\dagger}\tilde{\mathbf{e}}. \tag{1.10b}$$

So far, we have characterized the solution to (1.3) for a particular $\boldsymbol{\theta}$. Since we are interested in $\nabla_{\boldsymbol{\theta}}\mathbf{x}$, we ask the natural question whether (1.8) holds for all $\boldsymbol{\theta}'$ within a vicinity of this specific $\boldsymbol{\theta}$. The answer to this question is in the affirmative. To see this, fix $\mathbf{u}$ and perturb $\boldsymbol{\theta}$ to get $\boldsymbol{\theta}'$ so it still satisfies (1.10). Using $\boldsymbol{\theta}'$ in lieu of $\boldsymbol{\theta}$, construct $\mathbf{x}_{\boldsymbol{\theta}'}$ from (1.8), and $\tilde{\boldsymbol{\lambda}}_{\boldsymbol{\theta}'}$ from (1.6). In doing so, we row-partition matrices assuming the same constraints are active as for $\boldsymbol{\theta}$. This means we still use matrix $\mathbf{J}_{\boldsymbol{\theta}}$. We also set $\bar{\boldsymbol{\lambda}}_{\boldsymbol{\theta}'} = \mathbf{0}$. The constructed primal-dual pair $(\mathbf{x}_{\boldsymbol{\theta}'}, \boldsymbol{\lambda}_{\boldsymbol{\theta}'})$ satisfies the KKT conditions of (1.3) for $\boldsymbol{\theta}'$, and hence constitutes an optimal solution for this $\boldsymbol{\theta}'$. The aforesaid process can be repeated for any $\boldsymbol{\theta}'$ in the vicinity of the original $\boldsymbol{\theta}$ because (1.10) are strict inequalities. In other words, formula (1.8) is valid for all $\boldsymbol{\theta}'$ around $\boldsymbol{\theta}$ with matrix $\mathbf{J}_{\boldsymbol{\theta}}$ remaining unaltered. Therefore, matrix $\mathbf{J}_{\boldsymbol{\theta}}$ is indeed the sensitivity matrix $\nabla_{\boldsymbol{\theta}}\mathbf{x}$ evaluated at this particular $\boldsymbol{\theta}$.

The latter reveals that the OPF mapping $\mathbf{x}(\boldsymbol{\theta})$ is differentiable around $\boldsymbol{\theta}$. In addition, its Jacobian matrix is provided in closed form using (1.9). Calculating $\mathbf{J}_{\boldsymbol{\theta}}$ entails: *i)* Knowing the set of active constraints; *ii)* inverting matrix $\mathbf{A}$ once; and *iii)* inverting matrix $\mathbf{G}$. Although step *iii)* is executed once per $\boldsymbol{\theta}$, the computation is lightweight since the number of active constraints A should be smaller than N. In a nutshell, computing $\mathbf{J}_{\boldsymbol{\theta}}$ once problem (1.3) has been solved for a particular $\boldsymbol{\theta}$, is much simpler than solving (1.3) per se. Hence, computing sensitivities add insignificant complexity in the process of constructing the labeled dataset $(\boldsymbol{\theta}, \mathbf{x}_{\boldsymbol{\theta}}, \mathbf{J}_{\boldsymbol{\theta}})$.

Even though MPQP theory has been reviewed here for the sake of computing $\nabla_{\boldsymbol{\theta}}\mathbf{x}$, it is worth pointing out some additional features of MPQPs that may

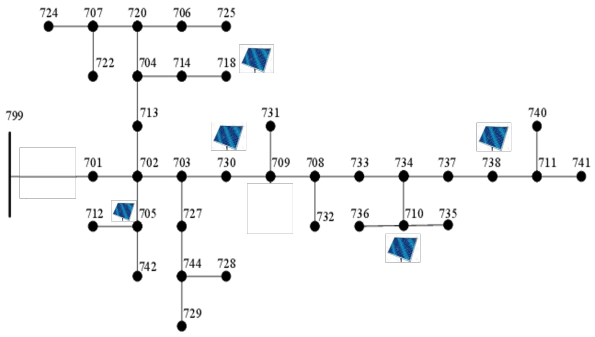

Figure 1.2: A modified IEEE 37-bus feeder showing additional solar generators.

not have been fully appreciated. To simplify the exposition, suppose $\tilde{\mathbf{C}}$ has full row-rank. The previous analysis reveals that the space of $\boldsymbol{\theta}$ for which (1.3) is feasible can be partitioned in polytopes defined by (1.10), which are termed *critical regions*. A critical region $\mathcal{C}$ is defined by (1.10) (with $\mathbf{u} = \mathbf{0}$ if $\tilde{\mathbf{C}}$ is full row-rank), and corresponds to a unique subset of linear inequality constraints being active at optimality. Critically, for all $\boldsymbol{\theta} \in \mathcal{C}$, the optimal primal/dual solutions of (1.3) can be provided in closed form, and are in fact, *affine functions* of $\boldsymbol{\theta}$ as shown in (1.8) and (1.7) for $\mathbf{u} = \mathbf{0}$, respectively. This property has been exploited to provide computational speedups, algorithmic developments, and probabilistic characterizations of MPQP solutions [54, 29, 33].

1.2.1.3. Numerical Tests on QP-OPF

Learning the solutions of the QP-based OPF was numerically evaluated using a P-DNN and an SI-DNN. The numerical tests were on a modified version of the IEEE 37-bus feeder hosting 5 DERs as shown in Fig. 1.2, using minute-based solar generation and load data from the Pecan Street dataset; see [40] for details. The optimal DER setpoints were obtained by solving ($P_{\boldsymbol{\theta}}$) using YALMIP and Sedumi. In solving the 1,440 OPF instances, no constraints were

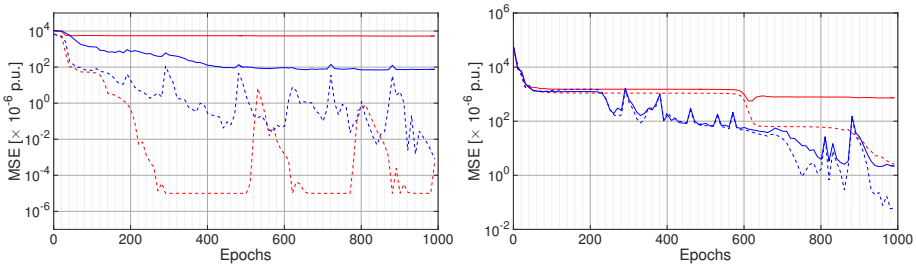

Figure 1.3: Training and testing errors achieved by P-DNN and SI-DNN over epochs in terms of MSE for hours 12 and 20.

active for 914 instances. Out of the remaining 526 instances, the LICQ was not satisfied for 175 instances; thus necessitating the approach pursued here.

We compared the P-DNN and SI-DNN, both trained to predict the minimizer of $(P_{\boldsymbol{\theta}})$. For the first set of tests, the DNNs were assumed to be trained on an hourly basis. To evaluate the potential benefit of integrating sensitivity information, the architecture, optimizer, and training epochs were kept identical for the two DNNs. The architecture was designed to optimize the performance of the P-DNN. It consisted of three layers with 210, 210, and 350 neurons, respectively, all with the rectified linear unit (ReLU) activation. The DNNs were implemented using the TensorFlow library on Google Colab. For each hour, 4 OPF instances were used for training and 56 for testing using the mean square error (MSE) $\|\mathbf{x}_s - \hat{\mathbf{x}}_s\|_2^2$ on the predicted setpoints as a metric. To compare P-DNN and SI-DNN under varying conditions, their training and testing errors were evaluated over different hours of the day. Figure 1.3 some representative results obtained during hours 12 and 20. The tests show that even when P-DNN yields smaller training errors, SI-DNN offers an improvement in testing error by one or two orders of magnitude.

We next compared P-DNN and SI-DNN in terms of learning OPF solutions

Table 1.1: Average Test MSE [in 10^{-6} pu] and training Time [in sec]
after 1,000 Epochs for 10–12 am (840 Min-based Scenarios)

Training	**P-DNN**		**SI-DNN**	
Scenarios	**MSE**	**Time**	**MSE**	**Time**
5%	1407.8	108.9	119.9	118.7
10%	520.2	77.1	72.9	79.7
20%	219.2	100.8	55.1	113.4

for a longer period of time and in terms of the time spent on their training. Specifically, the two DNNs were trained to learn the OPF for an entire day. The training dataset was constructed by sampling 5%, 10%, and 20% of the one-minute data. To explicitly focus on periods of high variability, we excluded the hours from midnight to 10 am from sampling. Table 1.1 summarizes the MSEs obtained after 1,000 epochs and averaged over 100 Monte Carlo draws of the training dataset. The table also reports the average training time for 1,000 epochs for the two methodologies. The runtimes on Google Colab often vary with each session. Thus, these times can only be compared separately for each training scenario; and not across scenarios. Evidently, SI-DNN training time is increased only by 10% or less compared to the training time of P-DNN. For example, the SI-DNN achieves an MSE of $119.9 \cdot 10^{-6}$ pu using 5% of the data, whereas the P-DNN achieves an MSE of $219.2 \cdot 10^{-6}$ pu although it has been trained by using 4 times more data (20%). The results establish that the SI-DNN consistently outperforms the P-DNN without incurring any significant increase in training time and has the ability to generalize using few OPF instances.

1.2.2. Learn-to-Optimize for AC-QCQP-OPF

The previous section demonstrated the advantages of sensitivity-informed training for learn-to-optimize tasks aimed at QPs motivated by the frequent encounter of QPs in power system optimization. However, we must acknowledge that the AC power-flow equations are quadratic equalities which render the originating AC-OPF formulations to be non-convex QCQPs. While the computational complexity of solving the QCQPs restricts their usage in real-time applications, they continue to be the first-principle benchmark of OPFs and there has been a concerted effort in accelerating them [35, 34]. This section illustrates how the sensitivity-informed approach could benefit learn-to-optimize for AC-QCQP-OPF tasks.

1.2.2.1. AC-OPF QCQP formulation

Consider the following non-convex QCQP which is an abstract representation of the classical AC-OPF for generator dispatch minimizing the cost of generation in transmission systems (see [41] for a detailed problem instance):

$$\mathbf{x}_{\boldsymbol{\theta}} := \arg\min_{\mathbf{v},\mathbf{x}_g} \quad \mathbf{a}_0^\top \mathbf{x}_g \tag{1.11a}$$

$$\text{s.to} \quad \mathbf{v}^\top \mathbf{L}_\ell \mathbf{v} = \mathbf{a}_\ell^\top \mathbf{x}_g + \mathbf{b}_\ell^\top \boldsymbol{\theta}, \quad \ell = 1:L \tag{1.11b}$$

$$\mathbf{v}^\top \mathbf{M}_m \mathbf{v} \leq \mathbf{d}_m^\top \boldsymbol{\theta} + f_m, \quad m = 1:M \tag{1.11c}$$

where the real and imaginary components of bus voltages are stacked in $\mathbf{v}$, the generator (re)active power injections are stacked in $\mathbf{x}_g$, and so, the minimizer $\mathbf{x}_{\boldsymbol{\theta}}$ has the optimal values of $\mathbf{v}$ and $\mathbf{x}_g$ stacked together. Further, cost coefficients are collected in $\mathbf{a}_0$, vectors $(\mathbf{a}_\ell, \mathbf{b}_\ell, \mathbf{d}_m)$ are suitable indicator (canonical) vectors that map generation, demands, and limits to the related buses, and

constants f_m relate to generation, voltage, and line limits. Thus, the first L constraints in (1.11b) correspond to the AC power flow equations alongside setting the angle reference, while constraints (1.11c) correspond to the M inequality constraints representing limits on generator power injections, bus voltages, and line power or current flows. Matrices $(\mathbf{L}_\ell, \mathbf{M}_m)$ can be derived from the AC power-flow equations as delineated in [41].

Aiming at computing the sensitivity of a minimizer $\mathbf{x}_{\boldsymbol{\theta}}$ of (1.11) with respect to $\boldsymbol{\theta}$, we explored the related literature. There has indeed been significant interest in computing the sensitivities of OPF minimizers with respect to load [6, 3, 5]. However, the primary motivation for these works was to efficiently compute minimizers and look into binding constraints for a *given trajectory* of load variations. Hence the related OPF was parameterized using a scalar, conveniently varied over a range of interest. Seeking to compute the minimizer sensitivities with respect to vector $\boldsymbol{\theta}$ in a relatively general setting, we explored beyond the power systems literature. Fortunately, there exists a rich corpus of work on perturbation analysis of continuous optimization problems with applications in operation research, economics, mechanics, and optimal control [10]. The first approach applied the implicit function theorem to the related first-order optimality conditions [18]. Thereon, many developments have been made towards relaxing the assumptions of initial works and expanding the scope to conic programs [15, 12, 10, 2]. For several recent applications, however, the early approaches of [18] are well suited due to their simplicity; see for example [7]. Building upon [18], we next compute the sensitivities required for SI-DNN in Section 1.2.2.2; and relax some of the needed assumptions in Section 1.2.2.3.

1.2.2.2. Perturbing Optimal Primal/Dual Solutions

Sensitivity analysis for the QCQP of (1.11) builds on analyzing perturbations in the related optimal primal/dual solutions. Let us denote the optimal dual variables corresponding to (1.11b) and (1.11c) by $\boldsymbol{\lambda}_{\boldsymbol{\theta}}$ and $\boldsymbol{\mu}_{\boldsymbol{\theta}}$, respectively. To keep the notation uncluttered, we will use $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ interchangeably with $(\mathbf{x}_{\boldsymbol{\theta}}, \boldsymbol{\lambda}_{\boldsymbol{\theta}}, \boldsymbol{\mu}_{\boldsymbol{\theta}})$, where the dependence on $\boldsymbol{\theta}$ is implicit. Under mild technical assumptions, a local primal/dual point for (1.11) satisfies the first-order optimality conditions [15]. The goal of sensitivity analysis is to find infinitesimal changes $(\mathrm{d}\mathbf{x}, \mathrm{d}\boldsymbol{\lambda}, \mathrm{d}\boldsymbol{\mu})$, so that the perturbed point $(\mathbf{x} + \mathrm{d}\mathbf{x}, \boldsymbol{\lambda} + \mathrm{d}\boldsymbol{\lambda}, \boldsymbol{\mu} + \mathrm{d}\boldsymbol{\mu})$ still satisfies the first-order optimality conditions when the input parameters change from $\boldsymbol{\theta}$ to $(\boldsymbol{\theta} + \mathrm{d}\boldsymbol{\theta})$ per [18]. To this end, we next review the optimality conditions and then perturb them to compute the sought sensitivities.

The Lagrangian function of (1.11) is defined as

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}; \boldsymbol{\theta}) := \mathbf{a}_0^\top \mathbf{x}_g + \sum_{\ell=1}^{L} \lambda_\ell \left(\mathbf{v}^\top \mathbf{L}_\ell \mathbf{v} - \mathbf{a}_\ell^\top \mathbf{x}_g - \mathbf{b}_\ell^\top \boldsymbol{\theta} \right) + \sum_{m=1}^{M} \mu_m \left(\mathbf{v}^\top \mathbf{M}_m \mathbf{v} - \mathbf{d}_m^\top \boldsymbol{\theta} - f_m \right).$$

With $\mathbf{x} := \{\mathbf{v}, \mathbf{x}_g\}$, Lagrangian optimality $\nabla_{\mathbf{x}} \mathcal{L} = \mathbf{0}$ gives

$$\underbrace{\left(\sum_{\ell=1}^{L} \lambda_\ell \mathbf{L}_\ell + \sum_{m=1}^{M} \mu_m \mathbf{M}_m \right)}_{:=\mathbf{Z}} \mathbf{v} = \mathbf{0}. \tag{1.12a}$$

$$\mathbf{a}_0 = \sum_{\ell=1}^{L} \lambda_\ell \mathbf{a}_\ell \tag{1.12b}$$

In addition to Lagrangian optimality, first-order optimality conditions include primal feasibility [cf. (1.11b)–(1.11c)], as well as complementary slackness and

dual feasibility for all m:

$$\mu_m \underbrace{\left(\mathbf{v}^\top \mathbf{M}_m \mathbf{v} - \mathbf{d}_m^\top \boldsymbol{\theta} - f_m\right)}_{:=g_m} = 0 \tag{1.13a}$$

$$\mu_m \geq 0. \tag{1.13b}$$

From the aforementioned optimality conditions, let us focus on those that take the form of equalities, namely (1.12a)–(1.12b), (1.11b), and (1.13a). For these conditions, we will compute their total differentials. From the first three, we obtain

$$\mathbf{Z}\,\mathrm{d}\mathbf{v} + \mathbf{L}_\lambda\,\mathrm{d}\boldsymbol{\lambda} + \mathbf{M}_\mu\,\mathrm{d}\boldsymbol{\mu} = \mathbf{0} \tag{1.14a}$$

$$\mathbf{A}^\top\,\mathrm{d}\boldsymbol{\lambda} = \mathbf{0} \tag{1.14b}$$

$$2\mathbf{L}_\lambda^\top\,\mathrm{d}\mathbf{v} - \mathbf{A}\,\mathrm{d}\mathbf{x}_g - \mathbf{B}\,\mathrm{d}\boldsymbol{\theta} = \mathbf{0} \tag{1.14c}$$

where $\mathbf{L}_\lambda := \sum_{\ell=1}^{L} \mathbf{L}_\ell \mathbf{v} \mathbf{e}_\ell^\top$ and $\mathbf{M}_\mu := \sum_{m=1}^{M} \mathbf{M}_m \mathbf{v} \mathbf{e}_m^\top$; note $\mathbf{e}_\ell$ and $\mathbf{e}_m$ are the ℓ-th and m-th canonical vectors of length L and M, respectively. Matrices $\mathbf{A}$ and $\mathbf{B}$ stack the vectors $\{\mathbf{a}_\ell\}_{\ell=1}^{L}$ and $\{\mathbf{b}_\ell\}_{\ell=1}^{L}$ as rows.

For (1.13a), the total differential is

$$g_m\,\mathrm{d}\mu_m + \mu_m\,\mathrm{d}g_m = 0 \tag{1.15}$$

where $\mathrm{d}g_m := (\nabla_{\mathbf{v}} g_m)^\top\,\mathrm{d}\mathbf{v} + (\nabla_{\boldsymbol{\theta}} g_m)^\top\,\mathrm{d}\boldsymbol{\theta}$ for all m. We identify three cases:

c1) If $\mu_m = 0$ and $g_m < 0$, then (1.15) implies $\mathrm{d}\mu_m = 0$. It follows that: *i)* $\mu_m + \mathrm{d}\mu_m = 0$; *ii)* $(\mu_m + \mathrm{d}\mu_m)(g_m + \mathrm{d}g_m) = 0$; and *iii)* $g_m + \mathrm{d}g_m < 0$ for an infinitesimally small magnitude $\mathrm{d}g_m$ in any direction. In conclusion, condition (1.15) ensures that the perturbed point satisfies conditions for

optimality, including the inequalities from primal/dual feasibility.

c2) If $\mu_m > 0$ and $g_m = 0$, then (1.15) gives $\mathrm{d}g_m = 0$. It also follows that: *i)* $g_m + \mathrm{d}g_m = 0$; *ii)* $(\mu_m + \mathrm{d}\mu_m)(g_m + \mathrm{d}g_m) = 0$; and *iii)* $\mu_m + \mathrm{d}\mu_m > 0$ for any small $\mathrm{d}\mu_m$. As in case *c1)*, condition (1.15) ensures that the perturbed point satisfies all conditions for optimality.

c3) If $\mu_m = g_m = 0$, then (1.15) is inconclusive on $\mathrm{d}g_m$ and $\mathrm{d}\mu_m$. In this *degenerate* case, for the perturbed point to remain optimal, we need to explicitly impose: *i)* $\mathrm{d}g_m \leq 0$; *ii)* $\mathrm{d}\mu_m \geq 0$; and *iii)* $\mathrm{d}g_m\,\mathrm{d}\mu_m = 0$. Even though the three latter constraints can be handled by the sensitivity analysis of [15, 12], they considerably complicate the treatment. Moreover, such degeneracy is seldom encountered numerically. We henceforth rely on the so called *strict complementarity* assumption, which ignores case *c3)* [18].

Assumption 1: Given a tuple of optimal primal/dual variables $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, constraint $g_m(\mathbf{x}; \boldsymbol{\theta}) = 0$ if and only if $\mu_m > 0$.

Two observations are in order. First, the analysis under *c1)–c2)* reveals that although we perturbed only the equality conditions for optimality, the obtained perturbed point satisfies the inequality conditions for optimality as well. Therefore, under Assumption 1, the point $(\mathbf{x} + \mathrm{d}\mathbf{x}, \boldsymbol{\lambda} + \mathrm{d}\boldsymbol{\lambda}, \boldsymbol{\mu} + \mathrm{d}\boldsymbol{\mu})$ satisfying the perturbed optimality conditions is (locally) optimal for (1.11), when solved for $\boldsymbol{\theta} + \mathrm{d}\boldsymbol{\theta}$. Second, despite Assumption 1, if a degenerate instance of (1.11) does occur for some $\boldsymbol{\theta}_s$ in the training dataset, the particular pair $(\boldsymbol{\theta}_s, \mathbf{x}_{\boldsymbol{\theta}_s})$ can still be used to train the SI-DNN, yet without the additional sensitivity information. In other words, degenerate instances can contribute only to the

first fitting term of (1.2).

Applying (1.15) for all m, the total derivatives for (1.13a) can be compactly expressed as

$$\mathcal{D}(\mathbf{g})\,\mathrm{d}\boldsymbol{\mu} + 2\mathcal{D}(\boldsymbol{\mu})\mathbf{M}_{\mu}^{\top}\,\mathrm{d}\mathbf{v} - \mathbf{D}^{\top}\,\mathrm{d}\boldsymbol{\theta} = \mathbf{0}, \tag{1.16}$$

where $\mathbf{g} := \{g_m\}_{m=1}^{M}$ stacks the inequality constraint values, and matrix $\mathbf{D} := \sum_{m=1}^{M} \mu_m \mathbf{d}_m \mathbf{e}_m^{\top}$. Operator $\mathcal{D}(\mathbf{x})$ returns a diagonal matrix with vector $\mathbf{x}$ on its main diagonal. Conditions (1.14) and (1.16) can be collected in matrix-vector form as

$$\underbrace{\begin{bmatrix} \mathbf{Z} & \mathbf{0} & \mathbf{L}_{\lambda} & \mathbf{M}_{\mu} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}^{\top} & \mathbf{0} \\ 2\mathbf{L}_{\lambda}^{\top} & -\mathbf{A} & \mathbf{0} & \mathbf{0} \\ 2\mathcal{D}(\boldsymbol{\mu})\mathbf{M}_{\mu}^{\top} & \mathbf{0} & \mathbf{0} & \mathcal{D}(\mathbf{g}) \end{bmatrix}}_{:=\mathbf{S}} \begin{bmatrix} \mathrm{d}\mathbf{v} \\ \mathrm{d}\mathbf{x}_g \\ \mathrm{d}\boldsymbol{\lambda} \\ \mathrm{d}\boldsymbol{\mu} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{B} \\ \mathbf{D}^{\top} \end{bmatrix}}_{:=\mathbf{U}} \mathrm{d}\boldsymbol{\theta} \tag{1.17}$$

To compute the sensitivities of primal and dual variables with respect to the p-th entry θ_p of $\boldsymbol{\theta}$, we need to solve the system of linear equations (1.17) for $\mathrm{d}\boldsymbol{\theta} = \mathbf{e}_p$. The size of the system can be reduced by dropping the numerous inactive inequality constraints of (1.11) for which $\mu_m = 0$ and $g_m < 0$, and thus, $\mathrm{d}\mu_m = 0$, as discussed earlier under case *c1)*. Notably, if matrix $\mathbf{S}$ is invertible, the aforementioned sensitivities can all be found at once using the respective blocks of $\mathbf{S}^{-1}\mathbf{U}\mathbf{e}_p$. We next address two relevant questions: *q1) When is* $\mathbf{S}$ *invertible?*; and *q2) What are the implications of a singular* $\mathbf{S}$*?*

1.2.2.3. Existence of Primal Sensitivities

To address *q1)* for an arbitrary optimization problem, say $(P_{\boldsymbol{\theta}})$, the existing literature identifies some assumptions on $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}; \boldsymbol{\theta})$. We first review these as-

sumptions and then assess if they are reasonable for the AC-OPF task at hand. Given an optimal primal $\mathbf{x}$ for some $\boldsymbol{\theta}$, let $\mathcal{A}(\mathbf{x})$ denote the subset of inequality constraints of $\mathbf{g}(\mathbf{x};\boldsymbol{\theta}) \leq \mathbf{0}$ that are *active or binding*, that is $\mathcal{A}(\mathbf{x}) := \{m : g_m(\mathbf{x};\boldsymbol{\theta}) = 0\}$. A primal solution $\mathbf{x}$ is termed *regular* if the next assumption holds.

Assumption 2: The vectors $\{\nabla_{\mathbf{x}} h_\ell\}_{\forall \ell}$ and $\{\nabla_{\mathbf{x}} g_m\}_{m\in\mathcal{A}(\mathbf{x})}$ are linearly independent.

For the OPF in (1.11), the functions h_ℓ and g_m correspond to the (in)equality constraints (1.11b)–(1.11c) written in the standard form as in $(P_{\boldsymbol{\theta}})$. Let us interpret Assumption 2 in the context of the DC-OPF instance (1.3). Since there are no equality constraints in (1.3), Assumption 2 would imply linearly independent rows of $\tilde{\mathbf{C}}$, which encapsulates the active constraints. We introduced the aforementioned requirement as linear independence constraint qualification in Section 1.2.1. Thus, Assumption 2 is the formal general statement for LICQ. If a (locally) optimal $\mathbf{x}$ satisfies the LICQ, the corresponding optimal dual variables $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ are known to be unique [9]. In addition to satisfying first-order optimality conditions, a sufficient condition for $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}; \boldsymbol{\theta})$ to be (locally) optimal is often provided by the following second-order optimality condition.

Assumption 3: For a subspace orthogonal to the subspace spanned by the gradients of active constraints

$$\mathcal{Z} := \Big\{\mathbf{z} : \mathbf{z}^\top \nabla_{\mathbf{x}} h_\ell = 0 \ \forall\ \ell, \mathbf{z}^\top \nabla_{\mathbf{x}} g_m = 0 \ \forall\ m \in \mathcal{A}(\mathbf{x})\Big\}$$

it holds that $\mathbf{z}^\top \nabla^2_{\mathbf{xx}} \mathcal{L} \mathbf{z} > 0$ for all $z \in \mathcal{Z} \setminus \{\mathbf{0}\}$.

Under the strict complementarity, regularity, and second-order optimality conditions, matrix $\mathbf{S}$ is guaranteed to be invertible; see Theorem 2.1 and Corollary 2.1 of [18].

Lemma 1.1 ***([18])*****:** *If Assumptions 1–3 hold, matrix* $\mathbf{S}^{-1}$ *exists.*

Lemma 1.1 implies that under the stated assumptions, the optimal primal and dual variables of $(P_{\boldsymbol{\theta}})$ vary smoothly with changes in parameter $\boldsymbol{\theta}$, and the associated sensitivities can be found via (1.17). Prior works that compute sensitivities of optimal primal and dual variables for scalar-parameterized OPF instances rely on the non-singularity of $\mathbf{S}$; see e.g., [6, 3].

While Assumptions 1–3 seem to be standard in the optimization literature, we discussed in Section 1.2.1 that LICQ (Assumption 2) is violated frequently for various renditions of the OPF [40]. Instances violating LICQ can be conceived for the AC-OPF in (1.11) too [4, 26]. To bring up one such example, consider a power system where a load bus m is connected to the rest of the system through another bus n via a single transmission line (m, n). As bus m is a load bus, it contributes two equality constraints in (1.11b), one each for the active and reactive power balance at bus m. It can be shown that if any of the three following scenarios occurs, LICQ fails: *i)* the limits on voltage magnitude, included in (1.11c) become binding (above or below) for both buses m and n; *ii)* line (m, n) becomes congested, activating a branch current flow limit in (1.11b) and a voltage limit at bus m becomes binding; or *iii)* line (m, n) becomes congested and a voltage limit at bus n becomes binding. Further detailed examples for AC-OPF instances violating LICQ can be found in [4, 26]. Attempting to circumvent LICQ violation via problem reformulations may be futile as their occurrences depend on $\boldsymbol{\theta}$, and are thus, hard to

analyze. However, before tackling the singularity of $\mathbf{S}$ due to LICQ violation, we must answer question *q2)*.

The implications of a singular $\mathbf{S}$ have previously been investigated in [15] and [12]: When LICQ is violated despite strict complementarity, the sensitivities of some primal/dual variables may still exist with respect to a θ_p. In detail, consider the set $\Gamma := \{\boldsymbol{\gamma} : \mathbf{S}\boldsymbol{\gamma} = \mathbf{U}\mathbf{e}_p\}$, which is the solution set of (1.17). If the n-th entry of $\boldsymbol{\gamma}$ remains constant for all $\boldsymbol{\gamma} \in \Gamma$, the sensitivity of the n-th entry of $[\mathbf{x}^\top \ \boldsymbol{\lambda}^\top \ \boldsymbol{\mu}^\top]^\top$ with respect to θ_p does exist; see [15] and [12] for physical interpretation and illustrative examples. While a subset of optimal primal/dual variables may be differentiable under LICQ violation, explicitly identifying the differentiable quantities requires instance-based numerical evaluation in [15] and [12]. Since for training an SI-DNN, we are interested only in the sensitivities $\nabla_{\boldsymbol{\theta}}\mathbf{x}$, we need to ensure that all solutions $\boldsymbol{\gamma} \in \Gamma$ share the same first N entries (recall $\mathbf{x} \in \mathbb{R}^N$). This is equivalent to saying that the first N entries of $\mathbf{n}$ are zero for all $\mathbf{n} \in \text{null}(\mathbf{S})$. The equivalence stems from the fact that if $\mathbf{S}\bar{\boldsymbol{\gamma}} = \mathbf{u}$ for a $\bar{\boldsymbol{\gamma}}$, any other solution to $\mathbf{S}\boldsymbol{\gamma} = \mathbf{u}$ takes the form $\boldsymbol{\gamma} = \bar{\boldsymbol{\gamma}} + \mathbf{n}$ for some $\mathbf{n} \in \text{null}(\mathbf{S})$. The next claim (see [41] for proof) provides sufficient conditions for the first N entries of $\mathbf{n}$ to be zero.

Theorem 1.1: *If Assumptions 1 and 3 hold, then $n_i = 0$ for $i = 1, \ldots, N$ for all $\mathbf{n} \in \text{null}(\mathbf{S})$.*

Thanks to Theorem 1.1, we can proceed with computing $\nabla_{\boldsymbol{\theta}}\mathbf{x}$ by solving (1.17) even if $\mathbf{S}$ is singular. In other words, Theorem 1.1 allows us to compute $\nabla_{\boldsymbol{\theta}}\mathbf{x}$ even if the LICQ (Assumption 2) fails. If $\mathbf{S}^\dagger$ is the pseudo-inverse of $\mathbf{S}$, the Jacobian matrix $\mathbf{J}_{\boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}}\mathbf{x}$ can be computed as the top N rows of $-\mathbf{S}^\dagger\mathbf{U}$. The previous analysis has tacitly presumed the system $\mathbf{S}\boldsymbol{\gamma} = \mathbf{u}$ has at

least one solution for all $\mathbf{u} \in \text{range}(\mathbf{U})$. Numerical tests for different renditions of AC-OPF have demonstrated that the system $\mathbf{S}\boldsymbol{\gamma} = \mathbf{u}$ features a solution indeed [27, 41].

As discussed earlier, we focus on training an SI-DNN for predicting generator voltage magnitudes and active power setpoints. Having solved (1.17) and found the sensitivity of $\mathbf{x}$ with respect to $\boldsymbol{\theta}$, the sensitivity of active power generation can be obtained readily using the corresponding entries of $\mathbf{x}_g$. The sensitivity of voltage magnitudes can be derived from the sensitivities of the real and imaginary components of voltages with respect to $\boldsymbol{\theta}$. Precisely, the voltage magnitude at bus n is given by $v_n = \sqrt{(v_n^r)^2 + (v_n^i)^2}$ and its sensitivity with respect to θ_ℓ can be found through the chain rule

$$\frac{\partial v_n}{\partial \theta_\ell} = \frac{1}{v_n}\left(v_n^r \frac{\partial v_n^r}{\partial \theta_\ell} + v_n^i \frac{\partial v_n^i}{\partial \theta_\ell}\right).$$

Evaluating the above completes the requirements of sensitivities for augmenting the SI-DNN training set.

1.2.2.4. Numerical Tests on AC-QCQP-OPF

This section illustrates the performance of SI-DNN approach in predicting AC-QCQP-OPF solutions for the IEEE 39-bus system. Further, elaborate empirical performance evaluations and insights are included in [41]. The DNN input $\boldsymbol{\theta}$ consists of the (re)active power demands at load buses. The DNN output is the setpoints for active power and voltage magnitude at all generators excluding the slack bus. We collect these output quantities in $\check{\mathbf{x}}_{\boldsymbol{\theta}}$, a subvector of $\mathbf{x}_{\boldsymbol{\theta}}$.

Both for P-DNN and SI-DNN, we chose a feed-forward fully connected architecture. The output layer uses tanh as its activation function (to explic-

itly enforce generator voltage and power limits via scaling), while all other layers use ReLU. We built all DNNs using the `TensorFlow 2.0` python platform alongside `Keras` libraries. For all tests, optimizer Adam was used with an exponential decay reducing the rate to 85% every 250 epochs. The initial learning rate will be reported later. DNNs were compiled using Jupyter Notebook on a 2.7 GHz Intel Core i5 computer with 8 GB RAM. We first trained DNNs towards predicting MATPOWER AC-OPF minimizers. We contrasted SI-DNN with P-DNN in terms of the MSE and the related training times. With the primary goal of improving sample efficiency, the numerical tests emphasize performance evaluation for relatively small training datasets; see [41] for tests with large training datasets.

The network parameters and nominal loads for the IEEE 39-bus system were fetched from MATPOWER `casefile` [55]. A dataset $\{(\boldsymbol{\theta}_s, \mathbf{J}_{\boldsymbol{\theta}_s}, \check{\mathbf{x}}_{\boldsymbol{\theta}_s})\}_{s=1}^{1000}$ was created by randomly sampling 1000 instances of $\boldsymbol{\theta}$ by scaling the benchmark demands entry-wise by a scalar drawn independently and uniformly within $[0.8, 1.2]$. For the aforementioned sampling, all 1000 OPF instances were solved using MATPOWER and were found to be feasible. Since the generator cost functions are identical in the benchmark system, a uniform active power cost was used for all generators. The default OPF formulation of MATPOWER deviates from the QCQP in (1.11). These differences introduce some nuances in building the linear system of (1.17) for computing sensitivities; see [41] for details. Based on preliminary tests, identical architectures were chosen for P-DNN and SI-DNN with 4 hidden layers each featuring 256 neurons.

The evaluation of DNNs was performed as follows. First, for a *training size* of 10, we created 20 different training sets by sampling 10 OPF instances from the dataset without replacement. For each of these 20 times or *runs*,

the OPF instances not sampled for training consisted of the testing sets. We then separately trained P-DNN and SI-DNN on these 20 sets. For the training sizes of $(50, 100, 250)$, we had $(20, 10, 4)$ runs, respectively. For training sizes $(10, 50, 100)$, the entire training set was used for gradient computation at each step, with the total epochs being 5000. When the training size was 250, the batch size was fixed to 100, and total epochs to 2000. The training and testing MSE loss for all training sizes and runs are shown in Fig. 1.4 *(top)*. For the tests with training size 10, the evolution of DNN errors is shown in Fig. 1.4 *(bottom)*. The average test MSE and training times for the two DNNs are shown in Table 1.2. From Fig. 1.4 *(top)*, we observe as anticipated, that for both DNNs, the gap between training and testing loss decreases for larger training size. Further, the errors for different *runs* are well clustered, indicating a numerically stable DNN implementation. From Table 1.2, it is fascinating to note that the test loss attained by SI-DNN is much lower than P-DNN, especially at smaller training sets. For instance, the P-DNN requires 100 samples to roughly attain the average test MSE which the SI-DNN attains with 10 samples. The lower MSE for P-DNN with training size 250 is a repercussion of not updating ρ for varying training sizes, which was avoided for simplicity. It is worth stressing that the improvement in sample efficiency comes at a modest increase in training time.

1.2.3. Sensitivity Analysis with Convex Relaxation

Towards tractably solving AC-OPF without resorting to linearizations of the power-flow constraints, there have been tremendous advancements in obtaining

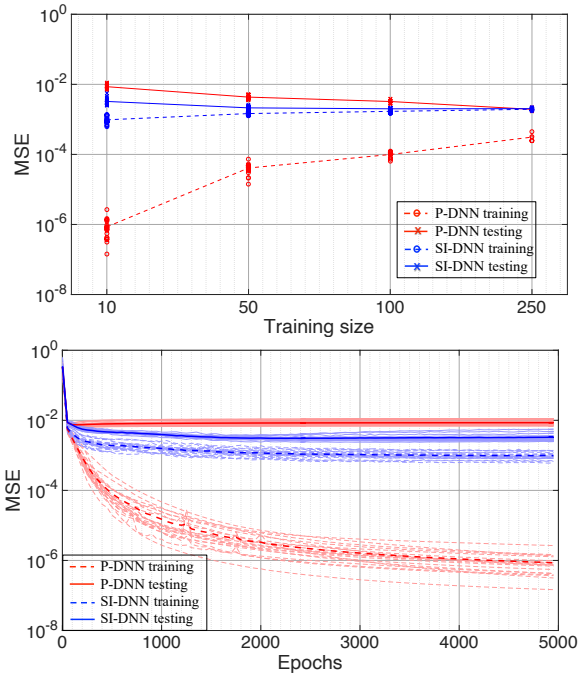

Figure 1.4: Average training and testing errors for different training sizes (*top*); and errors across epochs for different runs with training size of 10 (*bottom*).

convex relaxations with exactness guarantees [34]. Among the prominent relaxations are semidefinite programming and second-order cone programs [32]. Sensitivity analysis for these conic programs, although possible, could be perplexing [2]. Interestingly, one can resort to SDP- or SOCP-based convex relaxations to solve (1.11) efficiently and then use the optimal primal and dual variables of the relaxed problems to readily recover the corresponding optimal primal/dual pair for the originating QCQP instance (1.11). As an implication, the analysis approach of Section 1.2.2 can still be applied to obtain sensitivities of relaxed problems; see [41] and [27] for further details on SDP and SOCP relaxations, accordingly.

Table 1.2: Average Test MSE [$\times 10^{-3}$] and training Time [in sec] for predicting MATPOWER solution on IEEE 39-bus system

Training Size	**P-DNN**		**SI-DNN**	
	MSE	Time	MSE	Time
10	8.6	738	3.3	746
50	4.3	739	2.1	756
100	3.2	747	2.0	776
250	1.9	302	2.0	332

1.3. Deep Learning for Stochastic OPF

The previous section presented what we term an *OPF-then-Learn* training procedure, according to which the system operator has to solve S instances of the OPF to generate a labeled dataset prior to training the DNN. Moreover, the goal of that DNN was to learn the minimizer of a deterministic OPF. Changing gears, this section puts forth an *OPF-and-Learn* training procedure, wherein the DNN is trained by substituting the data-fitting cost of (1.1) or (1.2) with the OPF cost and constraints, thus sparing the need for labeled OPF data. The goal for this *OPF-and-Learn* training procedure is to train a DNN to learn the optimal policy of a *stochastic OPF*, rather than the minimizer of a *deterministic OPF*. Reference [17] addressed a similar task in the context of wireless communications and trained a DNN to find near-optimal stochastic policies in a wireless communication context. Here, we adopt that line of work to the OPF context and extend it to chance-constrained formulations [22].

Let us consider a specific variant of the OPF in ($P_{\boldsymbol{\theta}}$). The task is for the utility operator to coordinate DERs. As in (1.3), the operator would like to find the reactive power setpoints for DERs by minimizing ohmic losses on

distribution lines while maintaining voltage magnitudes within allowable limits:

$$\begin{aligned} \min_{\mathbf{x}\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad & \ell(\mathbf{x};\boldsymbol{\theta}) \\ \text{s.to} \quad & \underline{\mathbf{v}} \leq \mathbf{v}(\mathbf{x};\boldsymbol{\theta}) \leq \overline{\mathbf{v}}. \end{aligned} \tag{1.18}$$

Vector $\mathbf{v}$ carries voltage magnitudes at all buses; we will henceforth refer to voltage magnitudes as voltages unless stated otherwise. Functions $\ell(\mathbf{x};\boldsymbol{\theta})$ and $\mathbf{v}(\mathbf{x};\boldsymbol{\theta})$ capture the dependence of losses and voltages on the reactive setpoints of DERs $\mathbf{x}$ under grid conditions $\boldsymbol{\theta}$. DER setpoints are required to lie within feasible set $\mathcal{Q}_{\boldsymbol{\theta}}$. Because it is oftentimes easy to project onto $\mathcal{Q}_{\boldsymbol{\theta}}$, those constraints are left implicit. It is assumed that the feeder model and the participating inverters are known and remain fixed throughout the control period.

Solving (1.18) can be computationally and communication-wise taxing if $\boldsymbol{\theta}$ changes frequently. Moreover, parameters $\boldsymbol{\theta}$ may not be precisely known due to measurement noise and lack of state observability in distribution grids. Moreover, by the time ($P_{\boldsymbol{\theta}}$) is solved and optimal setpoints are downloaded to DERs, grid conditions $\boldsymbol{\theta}$ may have changed rendering the computed setpoints obsolete [30, 45]. To account for uncertainty in $\boldsymbol{\theta}$, the operator may resort to a stochastic OPF formulation, such as

$$\begin{aligned} \min_{\mathbf{x}\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad & \mathbb{E}[\ell(\mathbf{x};\boldsymbol{\theta})] \\ \text{s.to} \quad & \underline{\mathbf{v}} \leq \mathbb{E}[\mathbf{v}(\mathbf{x};\boldsymbol{\theta})] \leq \overline{\mathbf{v}} \end{aligned} \tag{1.19}$$

where the expectation operator $\mathbb{E}$ applies over $\boldsymbol{\theta}$, treated as a random vector here. We refer to (1.3) as the *averaged formulation* as it considers the average losses and constraints. While the averaged formulation accounts for uncertain-

ties in $\boldsymbol{\theta}$, the obtained setpoints may violate voltage limits quite frequently. This undesirable behavior results from the fact that constraining the average value of a particular bus voltage $\mathbb{E}[v_n(\boldsymbol{\theta})]$ does not provide strong guarantees on its per-instance values $v_n(\boldsymbol{\theta})$. A more conservative approach is possible through a *probabilistic or chance-constrained formulation*:

$$\min_{\mathbf{x}\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad \mathbb{E}[\ell(\mathbf{x};\boldsymbol{\theta})] \tag{1.20a}$$

$$\text{s.to} \quad \Pr\left[\underline{v}_n \leq v_n(\mathbf{x};\boldsymbol{\theta}) \leq \overline{v}_n\right] \geq 1-\alpha, \ \forall n \tag{1.20b}$$

where constraint (1.20b) ensures each bus voltage remains within limits with probability at least $1-\alpha$. Here $\alpha \in (0,1)$ is a small *violation probability*. In contrast to (1.3), the formulation in (1.20) focuses on restricting the *frequency* of voltage violations. Heed that both (1.3) and (1.20) yield *one-size-fits-all* solution in the sense that they seek a single vector of DER setpoints $\mathbf{x}$ to be applied under all grid conditions $\boldsymbol{\theta}$ drawn from a probability distribution function (PDF).

A more flexible approach would be to replace $\mathbf{x}$ in (1.19) or (1.20) with a *policy* $\mathbf{x}(\boldsymbol{\theta})$ that adapts to the $\boldsymbol{\theta}$ experienced each time. *How does a policy differ from the OPF mapping learned under Section 1.2?* The OPF mapping stems from a deterministic OPF, whereas the policy is designed to solve a stochastic OPF. Although they both take $\boldsymbol{\theta}$ as input and output DER setpoints, the setpoints computed by the policy aim at minimizing the *average* losses and satisfying voltage constraints on the average or in probability. In other words, a control policy accounts for uncertainties in $\boldsymbol{\theta}$ and focuses on the *expected* performance of the grid. This resonates well with industry standards that typically constrain voltages and powers in terms of time-averaged rather

than instantaneous values. The policy can be computed once, centrally by the operator, and executed by DERs in a decentralized manner in real time. Furthermore, the policy could be designed to be driven by purely local data. For example, the setpoint x_n for inverter n may depend solely on a subvector of $\boldsymbol{\theta}$, collected locally at bus n.

Similar to reinforcement learning, the control policy $\mathbf{x}(\boldsymbol{\theta})$ can be modeled as a DNN. If the DNN is parameterized by a weight vector $\mathbf{w}$, let us denote the control policy as $\mathbf{x}(\boldsymbol{\theta}; \mathbf{w})$. The sought DNN weights can be found via a stochastic OPF to be presented later in Section 1.3.2, after a quick detour to discuss chance constraints in Section 1.3.1.

1.3.1. Chance Constraints

Before delving into the details of finding a stochastic OPF policy, let us reformulate the chance constraint in (1.20b). The chance constraint for each bus can be first converted from a double-sided to a single-sided constraint. For example, enforcing $\underline{v}_n \leq v_n \leq \overline{v}_n$ for $\underline{v}_n = 0.97$ and $\overline{v}_n = 1.03$ per unit (pu), can be expressed as $(v_n - 1)^2 \leq 0.03^2$, or $0.03^2 - (v_n - 1)^2 \geq 0$. Therefore, the violation probability can be upper bounded as:

$$\Pr\left[(v_n - 1)^2 \geq 0.03^2\right] \leq \alpha, \quad \forall n.$$

The probability appearing on the previous constraint can be expressed in a different way using the expectation operator and the *unit step function* denoted by $\mathrm{u}(\cdot)$. In general, if $\boldsymbol{\theta}$ is any random vector and $\mathbf{x}$ is a variable, it is easy to verify that $\Pr\left[f_{\boldsymbol{\theta}}(\mathbf{x}) \geq 0\right] = \mathbb{E}\left[\mathrm{u}(f_{\boldsymbol{\theta}}(\mathbf{x}))\right]$. Therefore, the voltage chance

constraint per bus can be equivalently expressed as

$$\mathbb{E}\left[\mathrm{u}\left((v_n-1)^2-0.03^2\right)\right]\leq\alpha.$$

Nonetheless, the step function $\mathrm{u}(x)$ is discontinuous at $x=0$ and has zero gradients elsewhere. Hence, it is not a convenient option for gradient-based solvers. Standard literature on chance-constrained programs surrogates the step function with a parameterized convex approximation [36]. If $\mathrm{r}(z)=[z]_+=\max\{0,z\}$ denotes the *unit ramp function*, we get that

$$\mathrm{u}(z)\leq\mathrm{r}\left(\frac{z}{t}+1\right),\quad\text{for all } z \text{ and } t>0.$$

The inequality can be easily verified by examining the two cases of negative and non-negative z. Therefore, enforcing

$$\mathrm{r}\left(\frac{z}{t}+1\right)\leq\alpha\quad\text{or}\quad\mathrm{r}(z+t)\leq\alpha t\ \text{ for some }\ t>0$$

ensures $\mathrm{u}(x)\leq\alpha$. Consequently, the chance constraint of (1.20) can be inner approximated by constraint

$$\mathbb{E}\left[\mathrm{r}\left((v_n-1)^2-0.03^2+t\right)\right]\leq\alpha t \tag{1.21}$$

for some t. If the argument of the ramp function in (1.21) is a convex function with respect to optimization variable $\mathbf{x}$, then (1.21) is a convex constraint in both $\mathbf{x}$ and t. Moreover, it is an inner approximation or restriction of the original non-convex chance constraint. Being an inner approximation means that solving (1.20) with constraint (1.20b) replaced by (1.21) would find a *safe* policy, i.e., a policy that definitely satisfies (1.20b).

Nevertheless, when optimizing over DNN weights, convexity is lost and the benefit of inner approximation may be overshadowed by yielding over-conservative designs. To overcome such conservatism, the step function can be approximated by the (mirrored) logistic function. The logistic function and its derivative are [21]:

$$\sigma(x) := \frac{1}{1+e^{-x/\gamma}} \quad \text{and} \quad \frac{\mathrm{d}\sigma}{\mathrm{d}x} = \frac{1}{\gamma}\sigma(x)\cdot(1-\sigma(x))$$

with parameter $\gamma > 0$ controlling the approximation accuracy. Using the logistic function, chance constraints can be approximated as [47]

$$\mathbb{E}\left[\sigma\left((v_n-1)^2-0.03^2\right)\right] \leq \alpha. \tag{1.22}$$

1.3.2. DNN Training via a Stochastic OPF

Having presented different ways to deal with average and chance constraints, we next proceed on how to train the DNN so it learns a near-optimal stochastic OPF policy. This can be accomplished by solving the ensuing stochastic OPF over DNN weights $\mathbf{w}$:

$$\begin{aligned} \min_{\mathbf{w}:\mathbf{x}(\boldsymbol{\theta};\mathbf{w})\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad & \mathbb{E}[\ell(\mathbf{x}(\boldsymbol{\theta};\mathbf{w}),\boldsymbol{\theta})] \\ \text{s.to} \quad & \mathbb{E}[\mathbf{g}(\mathbf{x}(\boldsymbol{\theta};\mathbf{w}),\boldsymbol{\theta})] \leq \mathbf{0}. \end{aligned} \tag{1.23}$$

Here the vector mapping $\mathbf{g}(\mathbf{x}(\boldsymbol{\theta};\mathbf{w}),\boldsymbol{\theta})$ abstracts the average or chance constraints, with the latter being approximated by the ramp or the sigmoid function. Note that cost and constraint functions depend on the policy $\mathbf{x}(\boldsymbol{\theta};\mathbf{w})$ as well as grid conditions $\boldsymbol{\theta}$. The policy itself depends on $\boldsymbol{\theta}$ too, and also on

weights $\mathbf{w}$.

Solving (1.23) is challenging because of the expectation in the objective and constraints. Computing the needed expectations requires knowing the PDF of $\boldsymbol{\theta}$. Even if this PDF is known, computing the expectations is still non-trivial granted the policies $\mathbf{x}(\boldsymbol{\theta};\mathbf{w})$ are non-linear in $\boldsymbol{\theta}$. These complications promulgate a stochastic approximation approach towards solving (1.23). In the conventional ML setup, DNN weights are found by solving (1.1) via stochastic gradient descent-type algorithms. To accommodate constraints that depend on data such as those in (1.23), we adopt the stochastic primal/dual updates of [17] as presented next.

Consider the Lagrangian function of the problem in (1.23)

$$L(\mathbf{w};\boldsymbol{\lambda}) := \mathbb{E}[\ell(\mathbf{x}(\boldsymbol{\theta};\mathbf{w}),\boldsymbol{\theta})] + \boldsymbol{\lambda}^\top \mathbb{E}[\mathbf{g}(\mathbf{x}(\boldsymbol{\theta};\mathbf{w}),\boldsymbol{\theta})] \tag{1.24}$$

where $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers corresponding to the constraints in (1.23). A stationary point for the related dual problem

$$D^* := \max_{\boldsymbol{\lambda}\geq\mathbf{0}} \min_{\mathbf{w}:\mathbf{x}(\boldsymbol{\theta};\mathbf{w})\in\mathcal{Q}_{\boldsymbol{\theta}}} L(\mathbf{w};\boldsymbol{\lambda}) \tag{1.25}$$

can be obtained iteratively using the primal/dual updates indexed by k:

$$\mathbf{w}^{k+1} := \Big[\mathbf{w}^k - \mu_w \nabla_{\mathbf{w}} L(\mathbf{w}^k;\boldsymbol{\lambda}^k)\Big]_{\mathcal{Q}_{\boldsymbol{\theta}}} \tag{1.26a}$$

$$\boldsymbol{\lambda}^{k+1} := \Big[\boldsymbol{\lambda}^k + \mu_\lambda \nabla_{\boldsymbol{\lambda}} L(\mathbf{w}^k;\boldsymbol{\lambda}^k)\Big]_{+} \tag{1.26b}$$

where μ_w and μ_λ are positive step sizes. Here primal variables are updated through projected gradient descent steps on the Lagrangian function. Dual variables are updated through projected gradient ascent steps on the La-

grangian function. The operator $[x]_+ = \max\{x, 0\}$ is applied entry-wise and ensures $\boldsymbol{\lambda} \geq \mathbf{0}$ at all times. Note that the gradient $\nabla_{\boldsymbol{\lambda}} L(\mathbf{w}; \boldsymbol{\lambda})$ in the dual variable update (1.26b) can be substituted as

$$\nabla_{\boldsymbol{\lambda}} L(\mathbf{w}; \boldsymbol{\lambda}) = \mathbb{E}[\mathbf{g}(\mathbf{x}(\boldsymbol{\theta}; \mathbf{w}), \boldsymbol{\theta})].$$

The operator $[\cdot]_{\mathcal{Q}_{\boldsymbol{\theta}}}$ projects $\mathbf{w}^{k+1}$ such that $\mathbf{x}(\boldsymbol{\theta}; \mathbf{w}^{k+1}) \in \mathcal{Q}_{\boldsymbol{\theta}}$ for all $\boldsymbol{\theta}$. In general, this is not hard to implement. The implementation varies with the particular form of the feasible set $\mathcal{Q}_{\boldsymbol{\theta}}$. For example, if $\mathcal{Q}_{\boldsymbol{\theta}}$ consists of box constraints on individual entries of $\mathbf{x}$, it can be readily enforced using the hyperbolic tangent (tanh) as the output activation function.

Adopting a stochastic approximation approach, the expectation operators in (1.24) are first replaced by sample averages computed over a set of S scenarios $\{\boldsymbol{\theta}_s\}_{s=1}^S$. Scenarios $\boldsymbol{\theta}_s$ will be interchangeably termed *training data* or *grid condition scenarios*. The average ohmic losses for example can be approximated as

$$\mathbb{E}[\ell(\mathbf{x}(\boldsymbol{\theta}; \mathbf{w}), \boldsymbol{\theta})] \simeq \frac{1}{S} \sum_{s=1}^{S} \ell(\mathbf{x}(\boldsymbol{\theta}_s; \mathbf{w}), \boldsymbol{\theta}_s).$$

Even with this sample approximation, computing the gradients needed in (1.26) remains computationally expensive as one needs to compute gradients for each one of the S training examples. Fortunately, stochastic approximation alleviates this burden by approximating the gradients needed in (1.26) using a *single* scenario s per iteration k. In other words, gradients are approximated as

$$\mathbb{E}[\nabla_{\mathbf{w}} \ell(\mathbf{x}(\boldsymbol{\theta}; \mathbf{w}^k), \boldsymbol{\theta})] \simeq \nabla_{\mathbf{w}} \ell(\mathbf{x}(\boldsymbol{\theta}_s; \mathbf{w}^k), \boldsymbol{\theta}_s). \tag{1.27}$$

Therefore, at iteration k, stochastic approximation selects a scenario s to com-

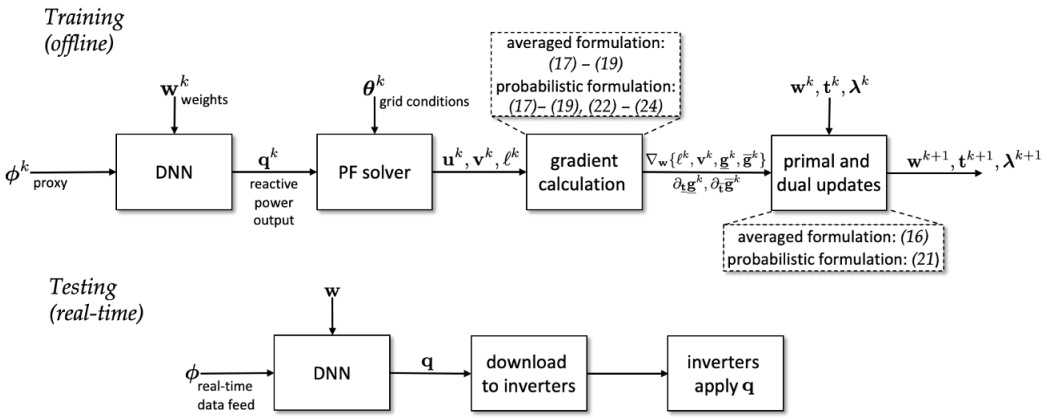

Figure 1.5: Workflow for the training and testing (operation) phases of the proposed DNN-based DER control strategy.

pute all gradients needed in (1.26). Scenarios can be selected at random or sequentially. Either way, since each iteration k ends up using only a single scenario s, we will henceforth use the symbol k to index both iterations and scenarios. *Mini-batch* versions of stochastic approximation do exist that use B with $1 < B < S$ scenarios per iteration.

Thanks to the single-scenario stochastic approximation, the gradients in (1.26) can be surrogated as [22, 25]

$$\mathbf{w}^{k+1} := \mathbf{w}^k - \mu_w \left(\nabla_{\mathbf{w}} \ell^k + \left(\nabla_{\mathbf{w}} \mathbf{g}^k \right)^\top \boldsymbol{\lambda}^k \right) \tag{1.28a}$$

$$\boldsymbol{\lambda}^{k+1} := \left[\boldsymbol{\lambda}^k + \mu_\lambda \mathbf{g}(\mathbf{x}(\boldsymbol{\theta}^k); \mathbf{w}^k), \boldsymbol{\theta}^k) \right]_+ \tag{1.28b}$$

where the shorthand notation $\nabla_{\mathbf{w}} \ell^k$ denotes the gradient of ℓ and $\nabla_{\mathbf{w}} \mathbf{g}^k$ the Jacobian matrix of $\mathbf{g}$, both with respect to $\mathbf{w}$ and both evaluated at $(\mathbf{w}^k, \boldsymbol{\theta}^k)$. The notation $\boldsymbol{\theta}^k$ denotes the scenario $\boldsymbol{\theta}_s$ selected at iteration k.

What are the practical steps to implement (1.28)*?* First, feed grid condition scenario vector $\boldsymbol{\theta}^k$ into the DNN parameterized with weights $\mathbf{w}^k$. The

DNN outputs the DER setpoints $\mathbf{x}(\boldsymbol{\theta}^k)$. These could be the reactive power injections by DERs. Given $\boldsymbol{\theta}^k$ (remaining of power injections) and $\mathbf{x}(\boldsymbol{\theta}^k)$, compute the complex voltages at all buses using a power flow (PF) solver. Knowing the complex bus voltages, compute bus voltage magnitudes and evaluate the mapping $\mathbf{g}(\mathbf{x}(\boldsymbol{\theta}^k); \mathbf{w}^k), \boldsymbol{\theta}^k)$ appearing in (1.28b). The n-th entry of $\mathbf{g}$ could be $v_n - \bar{v}_n$ or $\underline{v}_n - v_n$, if dealing with the average formulation of (1.3). If dealing with the probabilistic formulation of (1.20), one could use either the ramp or the sigmoid approximation. For the ramp approximation, use $g_n = \left[(v_n - 1)^2 - 0.03^2 + t\right]_+ - \alpha t$, where t is the auxiliary optimization variable. For the sigmoid approximation, use $g_n = \sigma\left((v_n - 1)^2 - 0.03^2\right) - \alpha$. When using the ramp approximation, the stochastic OPF has variable t as an additional auxiliary primal variable, which can be updated using a stochastic gradient descent step similar to (1.28a). To complete the primal update step of (1.28a), we also need to compute the gradients of losses ℓ and constraint functions $\mathbf{g}$ with respect to the DNN weights. The constraints are functions (linear, ramp, or sigmoid) of voltage magnitudes at all buses, and thus, to compute $\nabla_{\mathbf{w}}\mathbf{g}$, it suffices to compute $\nabla_{\mathbf{w}}\mathbf{v}$. Figure 1.5 summarizes the workflow for the training and testing (operational) phases for both formulations, while the next section describes how to compute the needed gradients.

1.3.3. Gradient Computations

We next explain how to compute the gradient vector $\nabla_{\mathbf{w}}\ell$ and the Jacobian matrix $\nabla_{\mathbf{w}}\mathbf{v}$ needed in (1.28a). We commence with $\nabla_{\mathbf{w}}\mathbf{v}$. Recall that vector $\mathbf{v} \in \mathbb{R}^N$ collects the voltage magnitudes at all N buses of a distribution grid (excluding the substation bus), and that the DNN output $\mathbf{x}(\boldsymbol{\theta}; \mathbf{w})$ corresponds

to a component of reactive power injections at a subset of buses. To keep the notation uncluttered, we can substitute $\mathbf{x} = \mathbf{q}$. Let vector $\mathbf{u} \in \mathbb{R}^{2N}$ carry the real and imaginary parts of complex voltages at all N buses, excluding again the substation bus. Using the chain rule of differentiation, the sought Jacobian matrix can be expressed as the product of Jacobian matrices:

$$\nabla_{\mathbf{w}}\mathbf{v} = \nabla_{\mathbf{q}}\mathbf{v} \cdot \nabla_{\mathbf{w}}\mathbf{q}. \tag{1.29}$$

The second matrix $\nabla_{\mathbf{w}}\mathbf{q}$ is the Jacobian of the DNN output with respect to its weights, and can be readily computed using gradient back-propagation using standard deep-learning libraries. Focusing on the first matrix $\nabla_{\mathbf{q}}\mathbf{v}$, we can apply the chain rule yet one more time to get:

$$\nabla_{\mathbf{q}}\mathbf{v} = \nabla_{\mathbf{u}}\mathbf{v} \cdot \nabla_{\mathbf{q}}\mathbf{u}. \tag{1.30}$$

Matrix $\nabla_{\mathbf{u}}\mathbf{v}$ can be easily computed and is block diagonal. The non-zero entries are the partial derivatives of voltage magnitude v_n with respect to the real and imaginary parts of the complex voltage at bus n.

Matrix $\nabla_{\mathbf{q}}\mathbf{u}$ cannot be computed directly because there is no analytic expression of voltage magnitudes as functions of (reactive) power injections. We bypass this predicament by leveraging the inverse function theorem. Let $\mathbf{s} \in \mathbb{R}^{2N}$ carry the active and reactive power injections at all buses, modulo the substation. Despite the opposite, power injections $\mathbf{s}$ can be expressed analytically in terms of complex voltages $\mathbf{u}$ through the power flow equations. Therefore, the Jacobian $\nabla_{\mathbf{u}}\mathbf{s}$ can be computed upon differentiating the power flow equations with respect to $\mathbf{u}$. Since $\mathbf{u}(\mathbf{s})$ is the inverse function of $\mathbf{s}(\mathbf{u})$, the

inverse function theorem dictates that

$$\nabla_{\mathbf{s}}\mathbf{u} = (\nabla_{\mathbf{u}}\mathbf{s})^{-1} \tag{1.31}$$

if the inverse exists. Matrix $\nabla_{\mathbf{q}}\mathbf{u}$ is a submatrix of $\nabla_{\mathbf{s}}\mathbf{u}$.

Evaluating $\nabla_{\mathbf{u}}\mathbf{s}$ requires knowing $\mathbf{u}$, which means that we first need to solve the power flow equations for a particular $\boldsymbol{\theta}^k$ and reactive injections by DERs $\mathbf{x}(\boldsymbol{\theta}^k; \mathbf{w}^k)$, to compute the corresponding complex voltages. Note that the expression for $\nabla_{\mathbf{u}}\mathbf{s}$ involves also the substation voltage. Nonetheless, the substation voltage is held at a constant and known value, and we do not differentiate over it.

The gradient $\nabla_{\mathbf{w}}\ell$ of losses with respect to DNN weights can be computed similarly as

$$(\nabla_{\mathbf{w}}\ell)^\top = (\nabla_{\mathbf{q}}\ell)^\top \cdot \nabla_{\mathbf{w}}\mathbf{q}. \tag{1.32}$$

Losses can be expressed as the summation of the active powers injected at all buses, including the substation. Because active power injections are quadratic functions of complex voltages (including the fixed voltage at the substation), we can easily compute $\nabla_{\mathbf{u}}\ell$. We can subsequently compute

$$(\nabla_{\mathbf{q}}\ell)^\top = (\nabla_{\mathbf{u}}\ell)^\top \cdot \nabla_{\mathbf{q}}\mathbf{u} \tag{1.33}$$

with the latter Jacobian computed as explained earlier.

1.3.4. Control Policies using Proxies

Ideally, the control policy is driven by the vector of grid conditions $\boldsymbol{\theta}$. Nevertheless, during real-time operation, the operator controlling the DERs may

not be able to observe the complete data $\boldsymbol{\theta}$. Instead, it may have to act upon a *proxy* $\boldsymbol{\phi}$ of the actual $\boldsymbol{\theta}$. The DER control policy driven by $\boldsymbol{\phi}$ can then be found by solving the constrained stochastic minimization

$$\begin{aligned} \min_{\mathbf{w}:\mathbf{x}(\boldsymbol{\phi};\mathbf{w})\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad & \mathbb{E}[\ell(\mathbf{x}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})] \\ \text{s.to} \quad & \mathbb{E}[\mathbf{g}(\mathbf{x}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})] \leq \mathbf{0}. \end{aligned} \tag{1.34}$$

The DER control policies found through (1.34) are adaptive to the proxy vector $\boldsymbol{\phi}$ and the optimization is over the parameters $\mathbf{w}$. The notation $\ell(\mathbf{x}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})$ captures the fact that the control policy is fed by proxy $\boldsymbol{\phi}$ to determine $\mathbf{q}$, but of course, ohmic losses depend on the actual grid conditions $\boldsymbol{\theta}$.

The proxy vector $\boldsymbol{\phi}$ can be chosen to represent the operational setup for which the control policies are being designed. In the absence of real-time measurements from all buses, and/or to save on communication overhead, vector $\boldsymbol{\phi}$ can consist of active line flows from distribution lines [25]. Meteorological data such as solar irradiance and ambient temperature, which serve as surrogates for $\mathbf{p}$, can also be included in $\boldsymbol{\phi}$. One can also explore convolutional neural networks (CNNs)-based policies that accept sky images in place of solar irradiance measurements as inputs to be included in $\boldsymbol{\phi}$. Similarly, the proxy vector $\boldsymbol{\phi}$ can also represent partial, delayed, or noisy data on the grid conditions or even aggregate versions of them. While training the DNN, the operator uses both actual and proxy data. In other words, the training dataset consists of the pairs $\{\boldsymbol{\theta}_s, \boldsymbol{\phi}_s\}_{s=1}^{S}$: Proxy data are used as inputs to the DNN, while actual data will be used to solve the power flow equations and evaluate the effect of DNN-based policies on grid losses and voltages. During operation, the DNN is fed by proxy data.

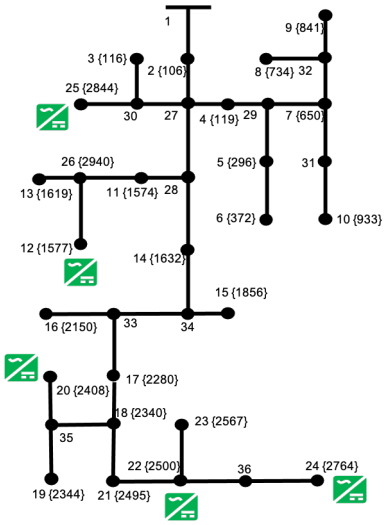

Figure 1.6: The IEEE 37-bus feeder used for the numerical tests. Node numbering follows the format `node number {panel ID}`. The inverters at nodes $\{12, 20, 22, 24, 25\}$ provide reactive power control, whereas the rest operate at unit power factor.

1.3.5. Numerical Tests

The proposed DNN-based control strategy was evaluated using a single-phase version of the IEEE 37-bus feeder. Real-world one-minute active load and solar generation data were extracted for April 2, 2011 from the Smart* project [13], and pre-processed as described in [22]. All tests were conducted on a 2.4 GHz 8-Core Intel Core i9 processor laptop computer with 64 GB RAM. Simulation scripts were written in Python and TensorFlow libraries to implement and train the DNNs. For the tests presented, four-layered fully connected DNNs were employed. The grid conditions vector $\boldsymbol{\theta} := [\mathbf{p}^g;\ \mathbf{p}^c;\ \mathbf{q}^c]$ were fed as inputs to the DNNs. Therefore, the input layers were chosen to have $3N$ neurons. The two subsequent hidden layers were fixed to having $3N$ and $2N$ neurons, respectively. Finally, the output layers had 5 neurons corresponding to the 5

inverters. All but the final layers of the DNNs employed the ReLU (rectified linear unit) activation with the final layers using a scaled *tanh* activation to ensure the inverter limits $\mathbf{q}^g \in \mathcal{Q}^t$. The weights for the DNN layers were initialized from a Gaussian distribution with zero mean and a unit standard deviation. The biases for the DNN layers, the dual variables, and the auxiliary variables were all initialized at zero.

The DNN was fed with the complete $\boldsymbol{\theta}$ obtained from measurements collected at all buses. DNN weights were updated using the DNN optimization algorithm Adam with a learning rate of 0.001. Dual variables were updated using SGD with a learning rate of 10 that decayed with the square root of the iteration index [31]. The model was then trained for 15 epochs over the training scenarios.

To demonstrate the efficacy of the proposed approach, the results are compared against a no-compensation scenario (DERs provide no reactive power support), and a deterministic optimal approach that solves the problem in (1.18) on per-minute minute. Fig. 1.7 compares the average losses and bus voltages under the three scenarios over the training set and during the high solar period of 12–4 pm. Without any reactive power compensation, buses $\{18, 19, 20, 21, 22, 33, 34\}$ experience over-voltages. The proposed DNN-based approach behaves as expected by lowering the average voltages at these buses down to the acceptable range. The deterministic optimal approach also achieves the same objective but by bringing all instantaneous voltages to the desired range whenever feasible. Note that both the DNN-based approach and the deterministic OPF incur higher losses when compared to the no-compensation scenario. This is a result of the increase in the magnitude of line currents on account of reactive power withdrawals. Since the deterministic optimal approach

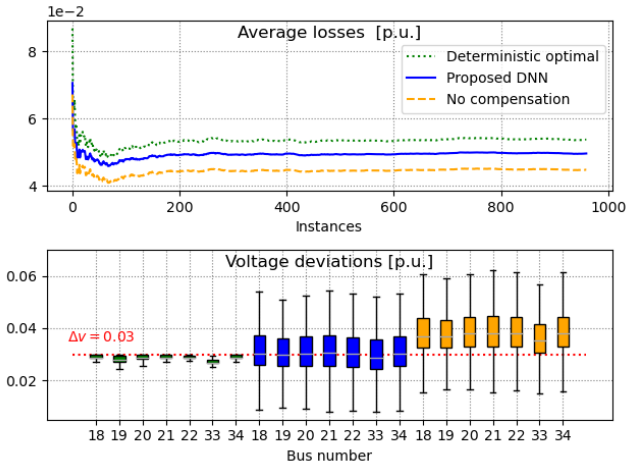

Figure 1.7: Evaluation using training data. *Top:* Time-averaged losses during the 12–4 pm training interval attained by the deterministic optimal control strategy of ($P_{\boldsymbol{\theta}}$); the proposed DNN-based inverter control; and no reactive power compensation by inverters. *Bottom:* Box plots showing the first and third quantiles of the voltage deviations experienced across buses under the three control strategies. Due to high solar generation, the feeder experiences lower ohmic losses at the expense of severe over-voltages if there is no reactive power control by inverters. The deterministic optimal inverter control strategy regulates voltages by absorbing reactive power, which increases line currents and consequently losses. The proposed strategy achieves lower average losses over deterministic optimal inverter control as voltages are not constrained within ±3% at all times.

focuses on instantaneous voltage values rather than their averages, it incurs higher losses when compared to the DNN-based approach. The trained DNN was then evaluated over unseen scenarios of the testing set. As can be seen in Fig. 1.8, the proposed approach performed remarkably well in maintaining voltages within limits and lowering average losses over the testing set.

The bottom panels of Figs. 1.7 and 1.8 demonstrate that although voltages remain within limits on the average, instantaneous voltages can deviate widely. To remedy this, we also tested the probabilistic formulation upon ap-

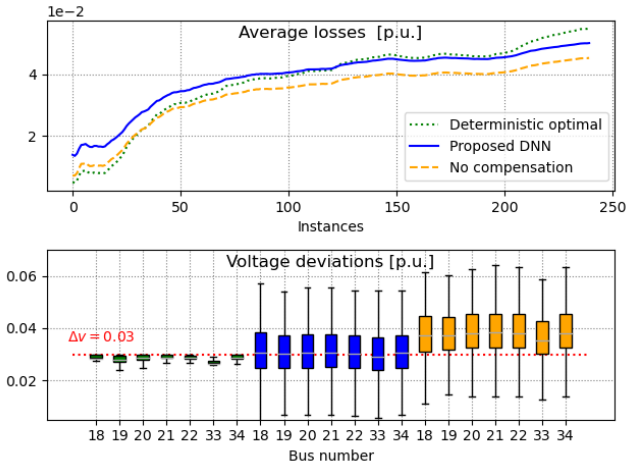

Figure 1.8: Results for averaged formulation over *testing data* during the interval 12–4 pm: Average losses under the deterministic optimal strategy, the proposed DNN-based approach; and no reactive power compensation are depicted on the top panel. Voltage deviations across buses under the three strategies are shown at the bottom panel.

proximating the step function appearing in the chance constraint using the ramp function. The experiments were conducted for the same time period of 12–4 pm, and for three different values of $\alpha = \{0.7, 0.5, 0.3\}$. The radar plots for the resulting sample probabilities of voltage violations are shown in Fig. 1.9. As desired, when compared to the averaged formulation, the occurrences of voltage violations under the probabilistic formulation were found to be drastically less for lower values of α. Since the calculated sample probabilities came out to be less than the selected α, the results in Fig. 1.9 confirm the conservative nature, being an inner approximation of the actual chance constraints. Approximating the step function of the chance constraint using a sigmoid function can alleviate such conservatism [21].

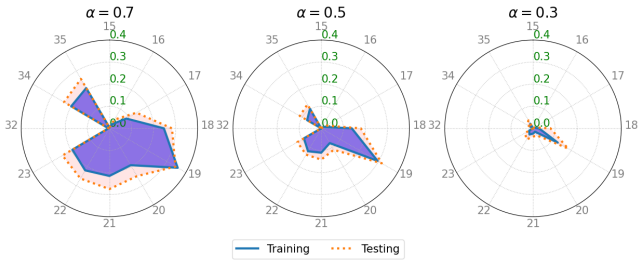

Figure 1.9: Results for probabilistic formulations for the 12–4 pm window. Voltage profiles for different values of $\alpha = \{0.7, 0.5, 0.3\}$ are depicted. Angular markings correspond to bus numbers whereas radial markings are sampled probabilities of voltage limits violation.

1.4. Conclusions

To conclude, this chapter has presented two novel ways for deep learning to expedite OPF tasks under deterministic and stochastic environments. Such solutions align well with the need to offload some of the heavy computational load from real-time to offline, and thus enhance scalability, safety, and efficiency of power-system operation. The OPF-then-Learn methodology has exploited the partial derivatives of OPF solutions to train a DNN using fewer data samples. Incorporating sensitivity information into a DNN inadvertently improves feasibility and enables retraining a DNN to cater to topological, operational, or distributional changes in the OPF. The OPF-and-Learn methodology has demonstrated how a DNN can be trained to output optimal policies that solve a stochastic OPF and satisfy network constraints on average or in probability. The two methodologies are quite general as they cover linearized and exact AC renditions of the OPF, for power transmission and distribution networks alike. The methodologies can be creatively combined with other ideas, such as graph neural networks or input-convex DNNs.

Bibliography

[1] Z. Abdeen, R. Jia, V. Kekatos, and M. Jin. A theoretical analysis of using gradient data for Sobolev training in RKHS. In *IFAC World Congress*, Yokohama, Japan, July 2023.

[2] Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M. Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 1(2):107–115, 2020.

[3] Venkataramana Ajjarapu and Nirlesh Jain. Optimal continuation power flow. *Electric Power Systems Research*, 35(1):17–24, October 1995.

[4] Katia C. Almeida and Aline Kocholik. Solving ill-posed optimal power flow problems via Fritz-John optimality conditions. *IEEE Trans. Power Syst.*, 31(6):4913–4922, November 2016.

[5] K.C. Almeida and R. Salgado. Optimal power flow solutions under variable load conditions. *IEEE Trans. Power Syst.*, 15(4):1204–1211, November 2000.

[6] K.C. Almeida, F.D. Galiana, and S. Soares. A general parametric optimal power flow. *IEEE Trans. Power Syst.*, 9(1):540–547, February 1994.

[7] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Intl. Conf. on Machine Learning*, page 136–145, Sydney, NSW, Australia, 2017.

[8] A. Bemporad, F. Borrelli, and M. Morari. Model predictive control based on linear programming – the explicit solution. *IEEE Trans. Automat. Contr.*, 47(12):1974–1985, December 2002.

[9] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition, 1999.

[10] J Frederic Bonnans and Alexander Shapiro. *Perturbation Analysis of Optimization Problems*. Springer Science & Business Media, New York, NY, 2000.

[11] F. Borrelli, A. Bemporad, and M. Morari. Geometric algorithm for multiparametric linear programming. *Journal of Optimization Theory and Applications*, 118(3):515–540, September 2003.

[12] E. Castillo, A. J. Conejo, C. Castillo, R. Minguez, and D. Ortigosa. Perturbation approach to sensitivity analysis in mathematical programming. *Journal of Optimization Theory and Applications*, 128(1):49–74, January 2006.

[13] Dong Chen, Srinivasan Iyengar, David Irwin, and Prashant Shenoy. Sunspot: Exposing the location of anonymous solar-powered homes. In *ACM International Conference on Systems for Energy-Efficient Built Environments*, page 85–94, Palo Alto, CA, Nov 2016.

[14] Yize Chen and Baosen Zhang. Learning to solve network flow problems via neural decoding. preprint, 2020. URL https://arxiv.org/abs/2002.04091.

[15] A. J. Conejo, E. Castillo, R. Minguez, and R. Garcia-Bertrand. *Decomposition Techniques in Mathematical Programming*. Springer, 2006.

[16] D. Deka and S. Misra. Learning for DC-OPF: Classifying active sets using neural nets. In *IEEE PowerTech*, pages 1–6, Milan, Italy, June 2019.

[17] M. Eisen, C. Zhang, L. F. O. Chamon, D. D. Lee, and A. Ribeiro. Learning optimal resource allocations in wireless systems. *IEEE Trans. Signal Processing*, 67(10):2775–2790, May 2019.

[18] Anthony V Fiacco. Sensitivity analysis for nonlinear programming using penalty methods. *Mathematical Programming*, 10(1):287–311, December 1976.

[19] Ferdinando Fioretto, Terrence W.K. Mak, and Pascal Van Hentenryck. Predicting AC optimal power flows: Combining deep learning and Lagrangian dual methods. In *AAAI Conf. on Artificial Intelligence*, New York, NY, February 2020.

[20] Neel Guha, Zhecheng Wang, Matt Wytock, and Arun Majumdar. Machine learning for AC optimal power flow. Climate Change Workshop at ICML 2019, 2019. URL https://arxiv.org/abs/1910.08842.

[21] S. Gupta, S. Misra, D. Deka, and V. Kekatos. DNN-based policies for stochastic AC-OPF. In *Proc. Power Syst. Comput. Conf.*, Porto, Portugal, June 2021.

[22] S. Gupta, V. Kekatos, and M. Jin. Controlling smart inverters using proxies: A chance-constrained DNN-based approach. *IEEE Trans. Smart Grid*, 13(2):1310–1321, March 2022.

[23] S. Gupta, S. Chatzivasileiadis, and V. Kekatos. Deep learning for optimal Volt/VAR control using distributed energy resources. *IEEE Trans. Smart Grid*, 2023. URL https://arxiv.org/abs/2210.12805. (under review).

[24] S. Gupta, A. Mehrizi-Sani, S. Chatzivasileiadis, and V. Kekatos. Scalable optimal design of incremental Volt/VAR control using deep neural networks. *IEEE Contr. Syst. Lett.*, 7:1957–1962, June 2023.

[25] Sarthak Gupta, Vassilis Kekatos, and Ming Jin. Deep learning for reactive power control of smart inverters under communication constraints. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Tempe, AZ, November 2020.

[26] Adrian Hauswirth, Saverio Bolognani, Gabriela Hug, and Florian Dorfler. Generic existence of unique Lagrange multipliers in AC optimal power flow. *IEEE Contr. Syst. Lett.*, 2(4):791–796, October 2018.

[27] M. Jalali, M. K. Singh, V. Kekatos, G. B. Giannakis, and C. C. Liu. Fast inverter control by learning the OPF mapping using sensitivity-informed Gaussian processes. *IEEE Trans. Smart Grid*, 14(3):2432–2445, May 2022.

[28] M. Jalali, S. Taheri, and V. Kekatos. Strategic investment in energy markets using bayesian optimization. In *Proc. IEEE PES General Meeting*, Orlando, FL, July 2023.

[29] Y. Ji, R. J. Thomas, and L. Tong. Probabilistic forecasting of real-time LMP and network congestion. *IEEE Trans. Power Syst.*, 32(2):831–841, March 2017.

[30] V. Kekatos, G. Wang, A. J. Conejo, and G. B. Giannakis. Stochastic reactive power management in microgrids with renewables. *IEEE Trans. Power Syst.*, 30(6):3386–3395, November 2015.

[31] L. M. Lopez-Ramos, V. Kekatos, A. G. Marques, and G. B. Giannakis.

Two-timescale stochastic dispatch of smart distribution grids. *IEEE Trans. Smart Grid*, 9(5):4282–4292, September 2018.

[32] Steven Low. Convex relaxation of optimal power flow—-Part I: Formulations and equivalence. *IEEE Trans. Control of Network Systems*, 1(1): 15–27, June 2014.

[33] A. N. Madavan, S. Bose, Y. Guo, and L. Tong. Risk-sensitive security-constrained economic dispatch via critical region exploration. In *Proc. IEEE PES General Meeting*, Atlanta, GA, USA, August 2019.

[34] Daniel K. Molzahn and Ian A. Hiskens. A survey of relaxations and approximations of the power flow equations. *Foundations and Trends® in Electric Energy Systems*, 4(1-2):1–221, February 2019. ISSN 2332-6557.

[35] Daniel K. Molzahn, Florian Dörfler, Henrik Sandberg, Steven H. Low, Sambuddha Chakrabarti, Ross Baldick, and Javad Lavaei. A survey of distributed optimization and control algorithms for electric power systems. *IEEE Trans. Power Syst.*, 8(6):2941–2962, November 2017.

[36] Arkadi Nemirovski and Alexander Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2007.

[37] Damian Owerko, Fernando Gama, and Alejandro Ribeiro. Optimal power flow using graph neural networks. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Process.*, pages 5930–5934, Barcelona, Spain, May 2020.

[38] Xiang Pan, Minghua Chen, Tianyu Zhao, and Steven H Low. Deepopf: A

feasibility-optimized deep neural network approach for AC optimal power flow problems. (preprint), 2020. URL https://arxiv.org/abs/2007.01002.

[39] Xiang Pan, Minghua Chen, Tianyu Zhao, and Steven H. Low. DeepOPF: a feasibility-optimized deep neural network approach for AC optimal power flow problems. *IEEE Systems Journal*, 17(1):673–683, March 2023.

[40] M. K. Singh, S. Gupta, V. Kekatos, G. Cavraro, and A. Bernstein. Learning to optimize power distribution grids using sensitivity-informed deep neural networks. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Tempe, AZ, November 2020.

[41] M. K. Singh, V. Kekatos, and G. B. Giannakis. Learning to solve the AC-OPF using sensitivity-informed deep neural networks. *IEEE Trans. Power Syst.*, 37(4):2833–2846, July 2022.

[42] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos. Learning to optimize: Training deep neural networks for interference management. *IEEE Trans. Signal Processing*, 66(20):5438–5453, October 2018.

[43] S. Taheri, M. Jalali, V. Kekatos, and L. Tong. Fast probabilistic hosting capacity analysis for active distribution systems. *IEEE Trans. Smart Grid*, 12(3):2000–2012, May 2021.

[44] P. Tondel, T. A. Johansen, and A. Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, 39(3):489 – 497, 2003.

[45] G. Wang, V. Kekatos, A.-J. Conejo, and G. B. Giannakis. Ergodic energy management leveraging resource variability in distribution grids. *IEEE Trans. Power Syst.*, 31(6), November 2016.

[46] W. Wang, N. Yu, Y. Gao, and J. Shi. Safe off-policy deep reinforcement learning algorithm for Volt-VAR control in power distribution systems. *IEEE Trans. Smart Grid*, pages 1–1, 2019.

[47] J. Wei, S. Gupta, D. C. Aliprantis, and V. Kekatos. A chance-constrained optimal design of Volt/VAR control rules for distributed energy resources. In *Proc. North American Power Symposium*, Ashville, NC, October 2023.

[48] Mausam Yatin Nandwani, Abhishek Pathak and Parag Singla. A primal dual formulation for deep learning with constraints. In *Proc. of Adv. Neural Inf. Process. Syst.*, pages 12157–12168, Vancouver, Canada, December 2019.

[49] Ahmed Zamzam and Kyri Baker. Learning optimal solutions for extremely fast AC optimal power flow. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Tempe, AZ, November 2020.

[50] L. Zhang, G. Wang, and G. B. Giannakis. Real-time power system state estimation and forecasting via deep unrolled neural networks. *IEEE Trans. Signal Processing*, 67(15):4069–4077, August 2019.

[51] Ling Zhang, Yize Chen, and Baosen Zhang. A convex neural network solver for DCOPF with generalization guarantees. *IEEE Trans. Control of Network Systems*, 9(2):719–730, June 2022.

[52] Qianzhi Zhang, Kaveh Dehghanpour, Zhaoyu Wang, Feng Qiu, and Dongbo Zhao. Multi-agent safe policy learning for power management of networked microgrids. *IEEE Trans. Smart Grid*, 12(2):1048–1062, March 2021.

[53] Tianyu Zhao, Xiang Pan, Minghua Chen, Andreas Venzke, and Steven H. Low. Deepopf+: A deep neural network approach for DC optimal power flow for ensuring feasibility. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Tempe, AZ, November 2020.

[54] Q. Zhou, L. Tesfatsion, and C.-C. Liu. Short-term congestion forecasting in wholesale power markets. *IEEE Trans. Power Syst.*, 26(4):2185–2196, November 2011.

[55] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas. MATPOWER: steady-state operations, planning and analysis tools for power systems research and education. *IEEE Trans. Power Syst.*, 26(1):12–19, February 2011.