DNN-based Policies for Stochastic AC OPF

Sarthak Gupta*†, Sidhant Misra†, Deepjyoti Deka†, Vassilis Kekatos*
*ECE Dept., Virginia Tech, Blacksburg, VA 24061, USA
{gsarthak,kekatos}@vt.edu
†Theory Division, Los Alamos National Laboratory, Los Alamos, NM, USA
{sidhant,deepjyoti}@lanl.gov

***Abstract*—A prominent challenge to the safe and optimal operation of the modern power grid arises due to growing uncertainties in loads and renewables. Stochastic optimal power flow (SOPF) formulations provide a mechanism to handle these uncertainties by computing dispatch decisions and control policies that maintain feasibility under uncertainty. Most SOPF formulations consider simple control policies such as affine policies that are mathematically simple and resemble many policies used in current practice. Motivated by the efficacy of machine learning (ML) algorithms and the potential benefits of general control policies for cost and constraint enforcement, we put forth a deep neural network (DNN)-based policy that predicts the generator dispatch decisions in real time in response to uncertainty. The weights of the DNN are learnt using stochastic primal-dual updates that solve the SOPF without the need for prior generation of training labels and can explicitly account for the feasibility constraints in the SOPF. The advantages of the DNN policy over simpler policies and their efficacy in enforcing safety limits and producing near optimal solutions are demonstrated in the context of a chance constrained formulation on a number of test cases.**

***Index Terms*—Chance constraints; constrained learning; deep neural networks; primal-dual updates; optimal power flow; stochastic optimization.**

I. Introduction

In current intra-day power systems operations, the optimal power flow (OPF) is solved at a time scale of 5-15 minutes using load forecasts to obtain economic generation dispatch, whereas affine/linear generation control policies are used within each OPF time period to account for real-time fluctuations in generation and demand. With growing levels of uncertainty, the design of the real-time control policies have become increasingly important for secure and economic grid operation. Such control design is the primary focus of this paper.

Motivated by the need for better uncertainty management, the stochastic AC-OPF (SOPF) [1], [2] problem that accounts for uncertainty, has received significant attention from the research community. Being a stochastic extension of the non-convex AC-OPF, the stochastic AC-OPF is highly computationally challenging. Standard stochastic optimization methods such as stochastic programming using sample average approximation, and scenario-based approaches quickly grow in size rendering them intractable. Hence, much effort has been devoted towards obtaining tractable formulations and designing efficient algorithms for obtaining fast and reliable solutions to this problem. The design of control polices however is relatively under-studied and most SOPF formulations consider *affine* functions to represent control policies. These linearly parameterized policies are algorithmically easier to handle, and serve as reasonable mathematical models for the automatic generation control and local voltage control employed in current practice. However, affine control polices are restrictive and possible sub-optimal. As such, more general control policies need to be studied as they have the potential to handle a much larger class of real-time fluctuations, possibly with limited information.

Design of general control policies poses significant technical/computational challenges and the choice of parameterization for their mathematical representation plays a pivotal role in their design. First, the expressive power of the parameterization dictates how general of a control policy it can represent. Second, the parameterization chosen must be compatible with the corresponding SOPF formulation to ensure computational tractability. In what follows, we provide a broad classification of approaches in the literature to this coupled control-parameterization and algorithm-design problem:

i) Affine policy. Here the control is restricted to be a linear function typically of the net system uncertainty, which makes it easy to model within OPF. Such affine policies have been used for convex SOPF with the DC power flow model [1] as well as for scenario-based approaches [3], and SOPF with nonlinear AC power flow model [4], [5].

ii) Non-affine policies include non-linear and hence more general policies that can ensure better feasibility and optimality enforcement over the affine ones. However they are computationally harder to incorporate within the SOPF. Examples include polynomial chaos based policies [6], [7], as well as non-liner policies with generator saturation [8].

iii) Nonlinear policies using robust optimization. This approach uses a fully optimized recourse for each uncertainty realization using a min-max-min formulation [9]. The problem is highly computationally challenging and approximation and relaxations are employed that can lead to conservative solutions. Similar robust formulations with affine control have also been investigated in the literature [10].

iv) Nonlinear policies using data-driven methods. This ap-

Submitted to the 22nd Power Systems Computation Conference (PSCC 2022). This work was supported by the U.S. National Science Foundation under grant 2034137 and by the U.S. Department of Energy under the Advanced Grid Modeling (AGM) program.

proach uses historical data or simulated OPF solutions to devise efficient non-linear control policies as well as to improve the computational speed of the SOPF. Examples of such methods include kernel-based control policies for voltage control [11] and active set learning approaches [12], [13] for OPF. Owing to the advancements in deep learning over the past decade, DNN-based OPF solution schemes have also been explored by the power systems community. Taking the conventional supervised learning route, DNN-based OPF solvers have been trained to match the optimal labels [14], [15]. However, solving a large number of OPFs to generate optimal labels is itself a computationally overwhelming task. Sensitivity-informed deep learning [16], [17], which matches not only OPF minimizers but also partial derivatives with respect to the inputs, partially alleviates the above concern by improving upon the data efficiency. Nonetheless, all these DNN-based solvers entail a label generation stage, which contributes significantly to training overheads.

In this paper, we consider a DNN-based control policy design for SOPF. The DNN parameterization serves as a generalization to non-linear policies, but also provides backward compatibility with previous policies since it can easily accommodate communication constraints and restricted features (e.g., control based on total/net uncertainty and/or local control). Different from the aforementioned works, our approach does not involve an offline data generation phase where a large number of OPF or OPF-like problems are solved. Instead, similar to [18], [19], the training phase of the DNN itself serves as the SOPF solver, which if deployed within current practices will replace the OPF solved at the 5-15 minutes time scale. Crucially, we also model the *stochastic* non-linear power system constraints during training to ensure feasibility of the control actions. Thus our approach uses a hybrid model that has the advantages of the NNs such as high representation power and the potential for parallelization and GPU implementation, without sacrificing feasibility like black-box data-driven efforts.

Prior works on constrained unsupervised learning for DNNs focus on deterministic OPFs. Penalty terms that grow with constraint violations have been added to the training loss function in [20]. A new set of hyperparameters corresponding to coefficients of the penalty terms are introduced and need to be pre-tuned for achieving a desirable performance. Alternatively, a constraint completion and correction strategy is discussed in [21], which improves constraint satisfaction via post-processing on DNN predictions. In contrast, our work aims to design DNN-based solvers for SOPFs where constraint violations are permissible for some samples. Instead of introducing hyperparameters, a primal-dual learning procedure is implemented which finds the DNN weights and dual variables while solving the stochastic OPFs. Lastly, during the implementation phase, no post-processing is required and DNN predictions are implemented straightaway. To the best of our knowledge, this is the first approach to explicitly model stochastic constraints inside a DNN-based OPF formulation.

The rest of the paper is organized as follows. Section II presents the mathematical formulation for SOPF with system and control constraints. Section III describes in detail our control policy design, in particular the modeling and training of our neural network with system constraints. Section IV includes experimental validation of our approach and comparison with existing methods on IEEE test cases. Finally, Section V concludes the paper.

II. Problem Formulation

A. Grid Modeling

Consider a power transmission network modeled as a directed connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. The nodes $n \in \mathcal{N} := \{1, \ldots, N\}$ of the graph $\mathcal{G}$ correspond to network buses, while the directed edges $e_{nk} \in \mathcal{E}$ to transmission lines. The complex voltage at bus n is expressed in polar coordinates as $v_n e^{j\theta_n}$, and the complex power injection at the same bus as $p_n + jq_n$. The power injections at node n are described by the AC power flow equations (AC-PF) as

$$p_n = v_n \sum_{k=1}^{N} v_k \left(G_{nk} \cos\theta_{nk} + B_{nk} \sin\theta_{nk}\right) \tag{1a}$$

$$q_n = v_n \sum_{k=1}^{N} v_k \left(G_{nk} \sin\theta_{nk} - B_{nk} \cos\theta_{nk}\right) \tag{1b}$$

where $\theta_{nk} := \theta_n - \theta_k$, and G_{nk} and B_{nk} are the entries of the real and imaginary parts of the bus admittance matrix $\mathbf{Y} = \mathbf{G} + j\mathbf{B}$. Power injections can be decomposed into their dispatchable (p_n^g, q_n^g) and inflexible (p_n^d, q_n^d) parts as

$$p_n = p_n^g - p_n^d \quad \text{and} \quad q_n = q_n^g - q_n^d.$$

The former corresponds to generators and flexible loads; the latter to inelastic loads hosted per bus n. The buses hosting dispatchable injections form set $\mathcal{N}_g \subset \mathcal{N}$, and its complement is defined as $\mathcal{N}_\ell$. Set $\mathcal{N}_g$ will be henceforth referred to as the set of *generator* buses, and $\mathcal{N}_\ell$ as the set of *load* buses. To simplify the exposition, each generator bus is assumed to host exactly one dispatchable unit. The bus indexed by $n = 1$ is designated as the slack and reference bus with $\theta_1 = 0$.

The complex power $P_{nk} + jQ_{nk}$ flowing from bus n to bus k over line e_{nk} can be expressed as

$$P_{nk} = v_n v_k \left(G_{nk} \cos\theta_{nk} + B_{nk} \sin\theta_{nk}\right) - v_n^2 G_{nk} \tag{2a}$$

$$Q_{nk} = v_n v_k \left(G_{nk} \sin\theta_{nk} - B_{nk} \cos\theta_{nk}\right) + v_n^2 \left(B_{nk} - B_{nk}^{\text{sh}}\right) \tag{2b}$$

where B_{nk}^{sh} is the shunt susceptance of line e_{nk}. The apparent power flow on the line is

$$f_{n,k} := \sqrt{P_{n,k}^2 + Q_{n,k}^2}. \tag{3}$$

B. Optimal Power Flow (OPF)

Let $c_n(p_n^g)$ denote the cost of generation at bus n. Given inflexible loads $\{p_n^d, q_n^d\}_{n\in\mathcal{N}}$, the OPF problem aims at minimizing the total cost of generation while operating the transmission network within limits. The OPF is posed as

$$\min \quad \sum_{n\in\mathcal{N}_g} c_n(p_n^g) \tag{4a}$$

$$
\begin{aligned}
&\text{over } \{v_n, \theta_n\}_{n\in\mathcal{N}}, \{p_n^g, q_n^g\}_{n\in\mathcal{N}_g} \\
&\text{s.to } (1), (2) && \forall n \in \mathcal{N}, \forall e_{nk} \in \mathcal{E} \\
&\quad p_n = p_n^g - p_n^d && \forall n \in \mathcal{N}_g && \text{(4b)} \\
&\quad q_n = q_n^g - q_n^d && \forall n \in \mathcal{N}_g && \text{(4c)} \\
&\quad p_n = -p_n^d && \forall n \in \mathcal{N}_\ell && \text{(4d)} \\
&\quad q_n = -q_n^d && \forall n \in \mathcal{N}_\ell && \text{(4e)} \\
&\quad \underline{p}_n^g \le p_n^g \le \bar{p}_n^g && \forall n \in \mathcal{N}_g && \text{(4f)} \\
&\quad \underline{q}_n^g \le q_n^g \le \bar{q}_n^g && \forall n \in \mathcal{N}_g && \text{(4g)} \\
&\quad \underline{v}_n \le v_n \le \bar{v}_n && \forall n \in \mathcal{N} && \text{(4h)} \\
&\quad f_{n,k} \le \bar{f}_{n,k} && \forall e_{n,k} \in \mathcal{E} && \text{(4i)} \\
&\quad \theta_1 = 0 && && \text{(4j)}
\end{aligned}
$$

where (1) and (4b)–(4d) enforce the power flow equations for power injections. Constraints (4f)-(4g) represent generation limits. Constraint (4h) confines voltage magnitudes within given limits. Constraints (2) and (4i) ensure that apparent power flows remain within line ratings. Lastly, constraint (4j) fixes the phase angle at the reference bus.

Given load demands stored in vector $\boldsymbol{\phi} := \{p_n^d, q_n^d\}_{n\in\mathcal{N}}$, the system operator solves (4) to find the optimal voltage and active power set-points for generators, which can be collected in vector $\mathbf{x} := \{\{v_n\}_{n\in\mathcal{N}_g}, \{p_n^g\}_{n\in\mathcal{N}_g\setminus\{1\}}\}$. Given $(\boldsymbol{\phi}, \mathbf{x})$, all other grid quantities involved in the OPF can be expressed as functions of $(\boldsymbol{\phi}, \mathbf{x})$ implicitly via the AC-PF equations (1) and (2). These quantities include the bus generations, and (p_1^g, q_g^1) which we collectively represent using the variable $\mathbf{y}$. The OPF of (4) can be abstracted as a parametric optimization solely over variable $\mathbf{x}$ given parameters $\boldsymbol{\phi}$:

$$
\begin{aligned}
\min_{\mathbf{x}} \quad & \sum_{n\in\mathcal{N}_g} c_n\left(\mathbf{x}, \boldsymbol{\phi}\right) && \text{(5a)} \\
\text{s.to} \quad & \underline{\mathbf{x}} \le \mathbf{x} \le \bar{\mathbf{x}} && \text{(5b)} \\
& \mathbf{y}\left(\mathbf{x}, \boldsymbol{\phi}\right) \le \bar{\mathbf{y}}. && \text{(5c)}
\end{aligned}
$$

Here (5b) captures the constraints on generator setpoints, that is (4f) and (4h) for $n \in \mathcal{N}_g$, while (5c) captures the constraints in (4g), (4h), and (4i).

Heed there is no need to explicitly enforce the constraints on loads as in (4b)–(4e). This is because active and reactive load demands have been included in $\boldsymbol{\phi}$, and their effect on all other grid quantities of interest is captured by the power flow equations, which are still taken into account in (5) indirectly through the constraint function $\mathbf{y}(\mathbf{x}, \boldsymbol{\phi})$. Note also that even though the mapping $\mathbf{y}(\mathbf{x}, \boldsymbol{\phi})$ does not feature an analytical form, we will be able to compute its partial derivatives with respect to $\mathbf{x}$ via the chain rule and the inverse function theorem later in Section III-C. For now, let us explain how the OPF in (5) can be modified to cope with uncertainty in $\boldsymbol{\phi}$.

C. Chance-Constrained Optimal Power Flow

Solving the non-convex problem of (5) can be computationally taxing if $\boldsymbol{\phi}$ changes frequently. Also by the time (5) is solved and the optimal setpoints are communicated to generators, demands $\boldsymbol{\phi}$ may have changed rendering $\mathbf{x}$ obsolete. The above concerns can be accounted for by considering a stochastic version of the OPF [22], [23]. We consider the chance constrained OPF (CC-OPF), that treats $\boldsymbol{\phi}$ as a random variable and seeks a control policy $\mathbf{x}(\boldsymbol{\phi})$ that reacts to the realization of $\boldsymbol{\phi}$ to minimize the expected generation cost while satisfying network constraints with high probability

$$
\begin{aligned}
\min_{\mathbf{x}(\boldsymbol{\phi})\in\mathcal{X}} \quad & \mathbb{E}\left[\sum_{n\in\mathcal{N}_g} c_n\left(\mathbf{x}, \boldsymbol{\phi}\right)\right] && \text{(6a)} \\
& \Pr\left[y_i\left(\mathbf{x}, \boldsymbol{\phi}\right) \le \bar{y}_i\right] \ge 1-\alpha, \quad i = 1:M && \text{(6b)}
\end{aligned}
$$

where the expectation $\mathbb{E}$ and the probability $\Pr$ are with respect to $\boldsymbol{\phi}$; and parameter $\alpha \in (0, 1)$ controls the probability of constraint violation. The deterministic constraint (5b) has been abstracted as $\mathbf{x} \in \mathcal{X}$. Using the indicator function, the chance constraints in (6b) can be recast as

$$
\mathbb{E}\left[\mathbb{1}\left(y_i\left(\mathbf{x}, \boldsymbol{\phi}\right) - \bar{y}_i\right)\right] \ge 1-\alpha,\ i = 1:M \tag{7}
$$

where the indicator function $\mathbb{1}(x)$ takes the values of 1 and 0 for $x \le 0$ and $x > 0$, respectively. The problem (6) is a variational optimization problem over the control policy $\mathbf{x}(\boldsymbol{\phi})$ and is generally intractable in its native form. Instead, one can parameterize the control policy as $\mathbf{x}(\boldsymbol{\phi}) = \boldsymbol{\pi}\left(\boldsymbol{\phi}; \mathbf{w}\right)$, where $\boldsymbol{\pi}(.)$ is a chosen function of $\boldsymbol{\phi}$ determined by the parameters $\mathbf{w}$. Restricting the policy to take this parameterized form, (6) can be written as the constrained stochastic minimization

$$
\begin{aligned}
\min_{\mathbf{w}:\boldsymbol{\pi}_\mathbf{w}\in\mathcal{X}} \quad & \mathbb{E}\left[\sum_{n\in\mathcal{N}_g} c_n\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}\right)\right] && \text{(8a)} \\
\text{s.to} \quad & \mathbb{E}\left[\mathbb{1}\left(y_i\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}\right) - \bar{y}_i\right)\right] \ge 1-\alpha, \quad i = 1:M
\end{aligned}
$$

where the optimization is now over the policy parameters $\mathbf{w}$, and the notation has been slightly abused by simplifying $\boldsymbol{\pi}\left(\boldsymbol{\phi}; \mathbf{w}\right)$ to $\boldsymbol{\pi}_\mathbf{w}$. In general (8) is an inner approximation of (6) due to the restriction imposed on the policy.Given their universal approximation capabilities [24], DNNs constitute great candidates for modeling the policy $\boldsymbol{\pi}_\mathbf{w}$ and narrowing down the gap between (6) and (8). In the case of DNNs, the parameter vector $\mathbf{w}$ collects the weighs and biases across all layers of the DNN.

We next delineate how a DNN can be trained to solve the CC-OPF in (8). Unlike standard feed-forward DNNs, our approach involves an iterative primal-dual scheme that inherently satisfies the non-convex stochastic constraints involved in (8).

III. Finding Optimal Policies

This section approximates the indicator function in (8) with a logistic function; adopts a stochastic primal-dual scheme to learn a DNN to solve the CC-OPF. It explains how the required gradients can be computed, and provides an overview of the training and operational phase of the DNN.

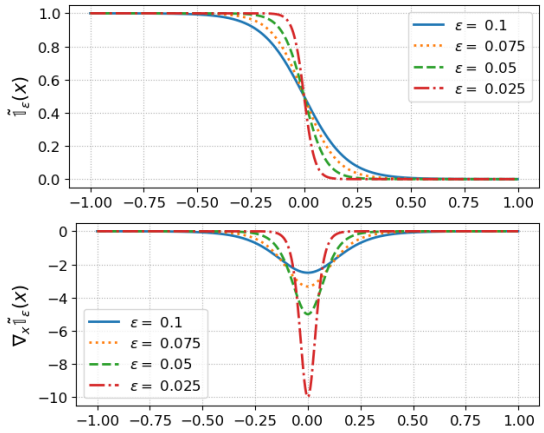

Fig. 1. Logistic function approximation of the indicator function *(top)*, and its derivative *(bottom)* for different values of parameter ϵ.

A. Approximating the Indicator Function

The weights $\mathbf{w}$ for a DNN-based policy can be learned by solving (8). However, computing the expectations in (8) is challenging even if the pdf of $\boldsymbol{\phi}$ is known, given the non-linearities in the objective and constraints. Standard DNN training alleviates this issue by using stochastic gradient descent (SGD) updates over a training dataset generated by drawing random samples with respect to the distribution of $\boldsymbol{\phi}$ to learn the DNN weights. In the presence of constraints, stochastic primal-dual updates (SPD) can be used to yield a similar training process [25], [19]. Unfortunately, the indicator function $\mathbb{1}(x)$ in (8) prevents the application of any gradient-based approach. This is because $\mathbb{1}(x)$ is discontinuous at $x = 0$, and its derivative becomes 0 at $x \neq 0$. Hence, no useful gradients are obtained while applying SPD.

The non-differentiability of $\mathbb{1}(x)$ has been addressed in the literature by substituting it with a differentiable approximation. Depending on the objective and constraints, the convex approximations of $\mathbb{1}(x)$ proposed in [26] may yield overall convex CC-OPF formulations [4]. Nonetheless, the computational advantage gained by introducing convexity is balanced off by the sub-optimal, conservative nature of these formulations. It is worth stressing that the problem in (8) has sources of non-convexity other than $\mathbb{1}(x)$, namely the policy $\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w})$ and the underlying non-linear power flow equations. Therefore, aiming for a more accurate and differentiable approximation of $\mathbb{1}(x)$ is pertinent, even if this approximation is non-convex. Fortunately, the logistic function, which is well known in the ML community, can serve as a smooth surrogate of $\mathbb{1}(x)$ [27]. The logistic function and its derivative are

$$\tilde{\mathbb{1}}_\epsilon(x) := \frac{e^{-x/\epsilon}}{1+e^{-x/\epsilon}}, \quad \nabla_x \tilde{\mathbb{1}}_\epsilon(x) = \frac{-\tilde{\mathbb{1}}_\epsilon(x)\left(1-\tilde{\mathbb{1}}_\epsilon(x)\right)}{\epsilon} \tag{9}$$

where parameter ϵ controls the accuracy of the approximation. It is easy to verify that $\tilde{\mathbb{1}}_\epsilon(x)$ tends to $\mathbb{1}_\epsilon(x)$ as $\epsilon \to 0^+$ as demonstrated in Figure 1. The same figure points towards a possible trade-off between approximation error and rate of convergence for SPD: As ϵ decreases, the range of values of x over which $\tilde{\mathbb{1}}_\epsilon(x)$ has non-diminishing derivative decreases as well. It is hence anticipated that for smaller values of ϵ, a larger number of SPD updates may be needed.

B. Stochastic Primal-Dual Updates

Dualizing (8) with $\mathbb{1}(x)$ being replaced by $\tilde{\mathbb{1}}_\epsilon(x)$ provides the Lagrangian function

$$L(\mathbf{w};\boldsymbol{\lambda}) := \mathbb{E}\left[\sum_{n\in\mathcal{N}_g} c_n\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}\right)\right] + \boldsymbol{\lambda}^\top\left[(1-\alpha)\mathbf{1} - \mathbb{E}\left[\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}\right) - \bar{\mathbf{y}}\right)\right]\right] \tag{10}$$

where $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers associated with the constraints in (8a). Observe that $\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}\right) - \bar{\mathbf{y}}\right)$ are vector quantities obtained by applying $\tilde{\mathbb{1}}_\epsilon(x)$ element-wise for all constraints in (8a). The corresponding dual problem is

$$D^* := \max_{\boldsymbol{\lambda}\geq\mathbf{0}} \min_{\mathbf{w}:\boldsymbol{\pi}_\mathbf{w}\in\mathcal{X}} L(\mathbf{w};\boldsymbol{\lambda}) \tag{11}$$

A stationary point of this min/max problem can be reached via the projected primal-dual iterations indexed by k

$$\mathbf{w}_{k+1} := \left[\mathbf{w}_k - \mu\nabla_\mathbf{w} L(\mathbf{w}_k\boldsymbol{\lambda}_k)\right]_{\boldsymbol{\pi}_\mathbf{w}\in\mathcal{X}} \tag{12a}$$

$$\boldsymbol{\lambda}_{k+1} := \left[\boldsymbol{\lambda}_k + \nu\nabla_{\boldsymbol{\lambda}} L(\mathbf{w}_k;\boldsymbol{\lambda}_k)\right]_+ \tag{12b}$$

with positive step sizes μ and ν. The primal update (12a) involves a projection of $\boldsymbol{\pi}_\mathbf{w}$ into the feasible set $\mathcal{X}$. With $\boldsymbol{\pi}_\mathbf{w}$ being modeled by a DNN, projection onto $\mathcal{X}$ can be easily accomplished by using appropriately scaled hyperbolic tangent functions (tanh). Dual variables are projected on the non-negative orthant via the operator $[x]_+ := \max\{x, 0\}$.

The expectation operator in (10) can be handled using stochastic approximation. This entails first surrogating each of the expected values by their sample averages over K uncertainty realizations $\{\boldsymbol{\phi}^k\}_{k=1}^K$. For the cost function for example, this approximation yields

$$\mathbb{E}\left[\sum_{n\in\mathcal{N}_g} c_n\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}\right)\right] \simeq \frac{1}{K}\sum_{k=1}^{K}\sum_{n\in\mathcal{N}_g} c_n\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}^k\right).$$

The *stochastic* primal-dual (SPD) updates further reduces the computational effort by using one uncertainty realization per iteration to obtain:

$$\mathbf{w}_{k+1} := \left[\mathbf{w}_k - \mu\sum_{n\in\mathcal{N}_g}\nabla_\mathbf{w} c_n^k + \mu\boldsymbol{\lambda}_k^\top\nabla_\mathbf{w}\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}^k - \bar{\mathbf{y}}\right)\right]_{\boldsymbol{\pi}_\mathbf{w}\in\mathcal{X}} \tag{13a}$$

$$\boldsymbol{\lambda}_{k+1} := \left[\boldsymbol{\lambda}_k + \nu(1-\alpha)\mathbf{1} - \nu\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}^k - \bar{\mathbf{y}}\right)\right]_+, \tag{13b}$$

where $c_n^k := c_n\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}^k\right)$ and $\mathbf{y}^k := \mathbf{y}\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}^k\right)$.

C. Computing Gradients for Primal Updates

The stochastic dual update simply requires evaluating the indicator approximation $\tilde{\mathbb{1}}(x)$ for the constraint functions $\mathbf{y}^k$. The stochastic primal update of (13a) however is more involved as it requires evaluating the gradients $\{\nabla_\mathbf{w} c_n^k\}$ and the Jacobian matrix $\nabla_\mathbf{w}\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}^k - \bar{\mathbf{y}}\right)$. We next explain how

these gradients can be computed via the chain rule and the inverse function theorem.

We commence with gradient $\nabla_{\mathbf{w}} c_n$ of the generation cost functions for all but the generator at the reference bus:

$$(\nabla_{\mathbf{w}} c_n)^\top = \frac{\mathrm{d}c_n}{\mathrm{d}p_n^g} \cdot (\nabla_{\mathbf{x}} p_n^g)^\top \cdot \nabla_{\mathbf{w}} \mathbf{x}, \quad \forall n \in \mathcal{N}_g \setminus \{1\}. \tag{14}$$

From the definition of $\mathbf{x}$, the gradient $\nabla_{\mathbf{x}} p_g^n$ is simply a canonical vector. The Jacobian matrix $\nabla_{\mathbf{w}} \mathbf{x}$ contains the partial derivatives of the DNN outputs with respect to the DNN inputs, and can be computed by most deep learning platforms such as TensorFlow using gradient *back-propagation* in a computationally efficient manner.

Computing $\nabla_{\mathbf{w}} c_1$ for the generation cost at the reference bus is less direct, yet still computationally efficient. It is less direct because p_1^g is not part of the setpoints, and hence, the DNN output $\mathbf{x}$. Nonetheless, it does depend on all other generator setpoints and in turn $\mathbf{x}$ indirectly, through the power flow equations. If we introduce the vector of voltages in polar coordinates $\mathbf{u} := [v_1 \ \ldots \ v_N \ \theta_2 \ \ldots \ \theta_N]^\top$ excluding the fixed angle $\theta_1 = 0$, the sought gradient can be computed as

$$(\nabla_{\mathbf{w}} c_1)^\top = \frac{\mathrm{d}c_1}{\mathrm{d}p_1^g} \cdot (\nabla_{\mathbf{u}} p_1^g)^\top \cdot \nabla_{\mathbf{x}} \mathbf{u} \cdot \nabla_{\mathbf{w}} \mathbf{x} \tag{15}$$

The Jacobian matrix $\nabla_{\mathbf{x}} \mathbf{u}$ required in (15) can be found by solving a system of linear equations. Let vector $\mathbf{z}$ collect the active and reactive power demands at all load buses. Then, using the power flow equations in (1), vectors $\mathbf{x}$ and $\mathbf{z}$ can be abstractly expressed as functions of $\mathbf{u}$ as

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{g}(\mathbf{u}) \\ \boldsymbol{\ell}(\mathbf{u}) \end{bmatrix}. \tag{16}$$

Regarding the mapping $\mathbf{g}(\mathbf{u})$, recall that $\mathbf{x} := \{\{v_n\}_{n\in\mathcal{N}_g}, \{p_n^g\}_{n\in\mathcal{N}_g\setminus\{1\}}\}$. Then, the first $|\mathcal{N}_g|$ entries of $\mathbf{g}(\mathbf{u})$ are simply $\mathbf{e}_n^\top \mathbf{u}$, where $\mathbf{e}_n$ is the n-th canonical vector. The second $|\mathcal{N}_g|$ entries of $\mathbf{g}(\mathbf{u})$ follow from the power flow equations plus any possible load p_n^ℓ at the corresponding bus. Since p_1^g is a dependent variable, it has not been included in $\mathbf{x}$ or $\mathbf{z}$. Differentiating either sides of (16) with respect to $\mathbf{x}$ yields:

$$\begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \nabla_{\mathbf{u}} \mathbf{g} \\ \nabla_{\mathbf{u}} \boldsymbol{\ell} \end{bmatrix} \nabla_{\mathbf{x}} \mathbf{u} \tag{17}$$

where $\mathbf{I}$ is an identity matrix of dimensions $2|\mathcal{N}_g| - 1$ and $\mathbf{0}$ is a matrix of all zeroes of dimension $2|\mathcal{N}_\ell| \times (2|\mathcal{N}_g| - 1)$. From (17), we can now compute the Jacobian matrix $\nabla_{\mathbf{x}} \mathbf{u}$ as

$$\nabla_{\mathbf{x}} \mathbf{u} = \begin{bmatrix} \nabla_{\mathbf{u}} \mathbf{g}(\mathbf{u}) \\ \nabla_{\mathbf{u}} \boldsymbol{\ell}(\mathbf{u}) \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \tag{18}$$

where the matrix to be inverted here is $(2N-1) \times (2N-1)$.

The Jacobian matrix $\nabla_{\mathbf{w}} \tilde{\mathbb{1}}_\epsilon (\mathbf{y} - \bar{\mathbf{y}})$ appearing in the primal update of (13a) can be found in a similar application of the chain rule and the inverse function theorem as

$$\nabla_{\mathbf{w}} \tilde{\mathbb{1}}_\epsilon (\mathbf{y} - \bar{\mathbf{y}}) = \nabla_{\mathbf{y}} \tilde{\mathbb{1}}_\epsilon (\mathbf{y} - \bar{\mathbf{y}}) \cdot \nabla_{\mathbf{u}} \mathbf{y} \cdot \nabla_{\mathbf{x}} \mathbf{u} \cdot \nabla_{\mathbf{w}} \mathbf{x}. \tag{19}$$

Matrix $\nabla_{\mathbf{y}} \tilde{\mathbb{1}}_\epsilon (\mathbf{y} - \bar{\mathbf{y}})$ is diagonal with diagonal entries provided by (9). Matrix $\nabla_{\mathbf{u}} \mathbf{y}$ contains the partial derivatives of all constraint functions of interest with respect to voltage magnitudes and angles. Such derivatives can be readily computed from (1)–(3).

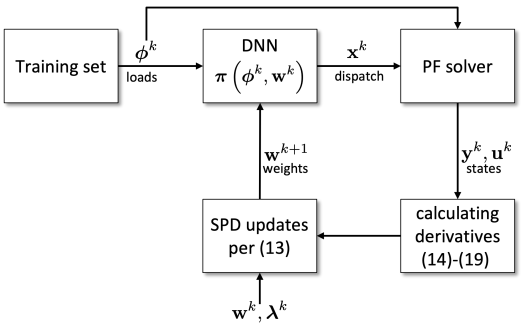

Fig. 2. Overview of the training phase for the DNN-based policy.

Remark 1. *In a typical OPF formulation, one would select complex voltages in rectangular or polar coordinates* $\mathbf{u}$ *as the optimization variables and then express each quantity in cost or constraints (injections, line flows, voltage magnitudes) as an analytic function (e.g., quadratic) of complex voltages. In the proposed formulation on the other hand, the optimization variables are the generator PV setpoints stored in* $\mathbf{x}$*; the PQ setpoints are known problem parameters collected in* $\mathbf{z}$*. Cost and constraints can be expressed as functions of the PV/PQ point* $(\mathbf{x}, \mathbf{z})$*. Although such functions may be implicit (no analytic form), one can still compute their values and derivatives with respect to* $\mathbf{x}$ *at some* $(\mathbf{x}, \mathbf{z})$*. This is possible upon solving the power flow (PF) equations at* $(\mathbf{x}, \mathbf{z})$ *and using the inverse function theorem as detailed between* (16)–(19)*, and under the tacit assumption that the PF equations are solvable at* $(\mathbf{x}, \mathbf{z})$*. The proposed method does not explicitly ensure PF solvability for the entire sequence of iterates* $\mathbf{x}_k = \boldsymbol{\pi}(\mathbf{z}; \mathbf{w}_k)$ *and across all scenarios. It is anticipated that if the iterates are initialized at a solvable PV/PQ point* $(\mathbf{x}, \mathbf{z})$ *and the step size* μ *is sufficiently small, it would be unlikely for the iterates in* (13) *to land at a PV/PQ point with no PF solution. This is intuitively justified by the continuity of the PF equations and the fact that when a PF trajectory is approaching insolvability, the PF solutions would become infeasible (e.g., voltages being well outside the desirable range). Practically, it should be noted that during our numerical tests, we did not encounter any scenario where a PV/PQ point would have no PF solution during an iteration.*

D. Deployment Workflow

Deploying the DNN-based strategy consists of two phases; (i) *solving the SOPF*, which is achieved by the training phase of the DNN, and (ii) *real time computation of control actions*, which is done by simply evaluating the trained DNN and is similar to the testing phase of DNNs. For the training phase, the system operator samples K grid conditions $\{\boldsymbol{\phi}^k\}_{k=1}^K$ that reflect the real-time conditions to which the policy will be applied. The training samples could be sourced from historically recorded loads, simulations, or predictions. Depending

on the generators that have been committed for the upcoming time-period, the operator identifies generator and load buses in sets $\mathcal{N}_g$ and $\mathcal{N}_\ell$, respectively, and defines vectors $\mathbf{x}$ and $\mathbf{y}$. The input and output layers of the DNN match the dimensions of $\boldsymbol{\phi}$ and $\mathbf{x}$, accordingly. The dimensions and the number of hidden layers are hyper-parameters to be determined for the specific CC-OPF setting. The DNN is then trained using SPD as per Section III. This entails: *i)* Sampling a $\boldsymbol{\phi}^k$; *ii)* Performing a forward pass through the DNN to obtain $\mathbf{x}^k$; *iii)* Obtaining $\mathbf{y}^k$ and $\mathbf{u}^k$ from a power flow solver; *iv)* Using $\{\theta^k, \mathbf{x}^k, \mathbf{y}^k, \mathbf{u}^k\}$ to calculate the gradients (14), (15), and (19); and *v)* Performing updates (13a)–(13b). These steps are repeated over the training samples for multiple epochs. Figure 2 depicts a block diagram of the training process. In real-time, the operator simply feeds the DNN with the real-time realizations of the uncertainty $\boldsymbol{\phi}$ to obtain the dispatch $\mathbf{x}$ at the output.

IV. Numerical Tests

A. Experimental Setup

The performance of the proposed DNN-based policy is evaluated using transmission networks of varying sizes. The simulation scripts are written in Python and run on a 2.4 GHz 8-Core Intel Core i9 processor laptop computer with 64 GB RAM. Compatibility with the commonly used MATPOWER [28] models is achieved by interfacing the Python script with the open-source Octave engine [29]. The communication between the two platforms is enabled via the Oct2py library [30], which allows seamless calling of M-files and Octave functions from Python. Leveraging upon this functionality, in-built MATPOWER functions are called to read the networks, solve the power flow equations, and calculate the derivatives needed for the primal updates. The implementation advantages of such synergy come at the cost of the communication overhead between the two platforms. For the networks tested, this overhead is within $0.02 - 0.03$ seconds per iteration. Because these delays can be avoided by porting the required MATPOWER functions to Python, they have been eliminated from the training times reported here.

TensorFlow libraries are employed to model and train the DNN. For each of the experiments, a five-layer DNN is considered. The five layers are: *1)* input layer matching the dimensions of $\boldsymbol{\phi}$; *2)* two hidden layers, each with half the dimensions of the input layer; *3)* output layer with the same dimensions as $\mathbf{x}$; and *4)* custom designed activation layer based on $\tanh$ ensuring $\mathbf{x} \in \mathcal{X}$. All DNN weights are initialized by randomly drawing from a standard normal distribution. DNN biases and dual variables are initialized at zero. The primal updates are performed using the Adam optimizer and the dual variables are updated using SGD. The primal step size μ decays exponentially at the rate of 0.5 per epoch, whereas the dual step size ν decays with the square-root of the iteration index [31]. The values for hyper-parameters $\{\epsilon, E, \mu_0, \nu_0\}$ are identified for each network via cross-validation, and are reported with results from the experiment.

For training and testing the DNN, we generated $1,000$ samples of $\boldsymbol{\phi}$ by adding zero-mean uniformly distributed noise to the nominal loads of the MATPOWER models. Recall that training corresponds to solving the SOPF formulation in (8) and testing corresponds to using the computed policy for real time prediction of control actions in response to uncertainty. To ensure that the problem in (8) is feasible, the generated demands were truncated within a range of $[-R, R]$ around the nominal point, where R was selected per network to ensure that the corresponding deterministic formulation of the OPF in (5) is feasible. The $1,000$ generated samples were then randomly grouped into training and testing sets of 800 and 200 examples, respectively. Training was conducted over E epochs of the training set. The performance of the DNN-based policy was bench-marked against the *OPF-policy* that solves the OPF in (5) deterministically per $\boldsymbol{\phi}_k$.

TABLE I
Training and testing details for the 14-bus system for different values of α.

α	ν_0	Train time (sec)	Maximum violation [%]	Test time (sec)	Average cost [$]
0.05	$3 \cdot 10^{-4}$	97.4	3.5	0.30	2180.53
0.10	$1.5 \cdot 10^{-4}$	98.5	8.5	0.33	2180.48
0.15	$1 \cdot 10^{-4}$	100.6	16.5	0.34	2180.45
0.20	$1.8 \cdot 10^{-4}$	100.6	17.0	0.34	2180.44

B. Accuracy of the Logistic Function Approximation

We investigate the accuracy and performance of the logistic function approximation using $\tilde{\mathbb{1}}_\epsilon(x)$ to the chance constraints in (7) using the 14-bus "pglib_opf_case14_ieee" network. For $\alpha \in \{0.05, 0.10, 0.15, 0.20\}$ and $\{\epsilon, E, \mu_0, R\} = \{0.01, 5, 10^{-3}, 0.1\}$, Table I shows the values of the remaining hyperparameters and the training time. The performance of the DNN-based policy on unseen test samples is also divulged via three metrics: maximum sampled probability of constraint violations (abbreviated in Table I as *maximum violation* [%]), test time, and the average cost.

Table I demonstrates the advantages of the DNN policy. First, the evaluated violation probabilities closely follows the prescribed α showing that the approximation $\tilde{\mathbb{1}}_\epsilon(x)$ is sufficiently accurate while facilitating use of the efficient SPD algorithm. This is an improvement over the conservative convex approximation of the indicator function in [19], where the observed violation probabilities were considerably less than α. Second, during the real-time evaluation of the policy, the DNN-based policy is able to predict the dispatch for the 200 test samples in around 0.3 seconds. Comparing this to the OPF-policy, which has an evaluation time of 31.3 seconds and cost of $\$2180.16$, the DNN policy is approximately 100 times faster without sacrificing optimality.

C. Enforcing Communication and Complexity Constraints

While our presentation thus far has been focused on the unconstrained DNN policy where all controllable variables are allowed to respond to uncertainty realizations, the framework also allows for seamless encoding of communication constraints where only part of $\boldsymbol{\phi}$'s is observed and communicated, and complexity constraints where only a subset of the control variables respond to uncertainty. This flexibility allows us to account for any limitations of the available infrastructure.

TABLE II
TEST RESULTS COMPARING THE FULL POLICY WITH AN AGC-TYPE POLICY FOR $\alpha = 0.10$, AND $E = 20$.

Policy	Maximum violation [%]		Average cost [$]	
	$R = 0.1$	$R = 0.2$	$R = 0.1$	$R = 0.2$
Full	8.5	10.5	2180.45	2184.67
AGC-type	9.0	26.5	2185.52	2189.23

As an example, we consider an automatic generation control (AGC) type policy, where only the active power generation of the participating generators responds to the total active load deviation in the system. This policy closely resembles the usual affine policy model in terms of input-output dependencies but allows for more general non-linear functions. This policy can be implemented within the DNN framework by modifying the input layer to have only 1 neuron that receives the signal $\sum_{n\in\mathcal{N}} p_n^d$, and making the neurons for $\{v_n^g\}_{n\in\mathcal{N}^g}$ insensitive to the input by setting the corresponding weights to 0. Note that the modified DNN still produces the average nominal values for $\{v_n^g\}_{n\in\mathcal{N}^g}$ at the output, because the biases in the output layer are learnt during training.

The AGC-type policy along with the full policy are evaluated on the 14 bus system for $\alpha = 0.1$ and two values of $R = \{0.1, 0.2\}$ corresponding to a *moderate* case with smaller uncertainty and a *stressed* case with larger uncertainty respectively. The results are presented in Table II. For the moderate case, both policies are able to enforce the chance constraints with the full policy marginally out-performing the AGC-type policy in terms of cost. For the stressed case however, the AGC-type policy could not converge to anywhere near the desired $\alpha = 0.10$ even after 20 epochs and saturated at a constraint violation probability of around 26.5%. The full policy on the other hand was able to decrease the probability of constraint violations to 10.5%. As a baseline comparison, the OPF policy was evaluated and found to attain an average cost of $ 2183.14. The stressed case demonstrates that general control policies can significantly improve cost of control actions and their ability to enforce constraints in real time.

D. Scalability

We test the DNN framework on a number of test networks of varying sizes. The values for hyper-parameters, load variation parameter R, and the training times of these networks are compiled in Table III and Table IV reports the performance of the trained DNNs. The testing times and the average costs attained by the proposed strategy are compared against the OPF policy. For all networks, the DNN policy is consistently able to enforce the chance constraints with the specified α with the costs remaining close to that of the OPF policy indicating that the DNN policy is near-optimal. The training times in Table III show a moderate increase with size of the test networks indicating that the proposed approach is highly scalable. We remark that the exact training times reported are less indicative than the trend. The exact times are highly dependent on implementation details where parallelized implementations and GPU deployment can drastically improve these numbers. The evaluation times in Table IV are orders of magnitude faster than the OPF policy, making them highly suitable for real-time computation of control actions.

TABLE III
THE VALUES FOR HYPER-PARAMETERS $\{\epsilon, E, \mu_0, \nu_0\}$, SETTING R, AND THE TRAINING TIMES FOR DIFFERENT NETWORKS FOR $\alpha = 0.05$

Network	ϵ	E	μ_0	ν_0	R	Train time (sec)
case6ww	0.005	5	10^{-3}	$4 \cdot 10^{-2}$	0.05	73.45
case69	0.01	5	10^{-3}	10^{-2}	0.1	79.85
case118	0.07	5	10^{-3}	1	0.01	308.4
case141	0.01	5	10^{-3}	10^{-3}	0.1	104.25

TABLE IV
TEST RESULTS FOR THE FULL DNN-BASED POLICY (FULL) AND OPF POLICY (OPF) FOR DIFFERENT NETWORKS FOR $\alpha = 0.05$.

Network	Maximum violation [%]	Time (sec)		Average cost [$]	
		Prop.	Opt.	Prop.	Opt.
case6ww	5	0.22	24.09	$3.17 \cdot 10^3$	$3.15 \cdot 10^3$
case69	0	0.25	26.35	80.58	80.58
case118	0.5	0.39	61.10	$1.30 \cdot 10^5$	$1.30 \cdot 10^5$
case141	0.0	0.28	40.34	251.89	251.89

V. CONCLUSIONS

We presented a DNN-based approach for solving the SOPF, where DNNs are used to parameterize the control policies required for real-time power balancing in response to uncertainty. Our approach does not require previously generated training labels and instead used the training phase to solve the SOPF. Stochastic primal-dual updates are employed to learn the DNN weights such that generation costs are minimized while respecting the power system constraints. Numerical tests on a variety of benchmark networks confirm that the generalized policy is able to provide high quality feasible solutions to chance constrained AC-OPF problem over a range of operating conditions, with significant improvements over an AGC-type policy in terms of cost and constraint enforcement. Comparison with the OPF policy where an OPF is solved in response to each uncertainty realization shows that the DNN policy is able to attain similar levels of feasibility and optimality, while facilitating near-instant computation of real-time control actions. In future, we plan to extend our approach to joint chance-constrained OPF problems and research scenario selection policies to aid in faster DNN training.

REFERENCES

[1] D. Bienstock, M. Chertkov, and S. Harnett, "Chance-constrained optimal power flow: Risk-aware network control under uncertainty," *Siam Review*, vol. 56, no. 3, pp. 461–495, 2014.

[2] L. Roald, F. Oldewurtel, T. Krause, and G. Andersson, "Analytical reformulation of security constrained optimal power flow with probabilistic constraints," in *2013 IEEE Grenoble Conference*. IEEE, 2013, pp. 1–6.

[3] M. Vrakopoulou, K. Margellos, J. Lygeros, and G. Andersson, "A probabilistic framework for reserve scheduling and security assessment of systems with high wind power penetration," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 3885–3896, 2013.

[4] E. Dall'Anese, K. Baker, and T. Summers, "Chance-constrained AC optimal power flow for distribution systems with renewables," *IEEE Trans. Power Syst.*, vol. 32, no. 5, pp. 3427–3438, 2017.

[5] I. Mezghani, S. Misra, and D. Deka, "Stochastic AC optimal power flow: A data-driven approach," *Electric Power Systems Research*, vol. 189, Dec 2020.

[6] T. Mühlpfordt, L. Roald, V. Hagenmeyer, T. Faulwasser, and S. Misra, "Chance-constrained AC optimal power flow: A polynomial chaos approach," *IEEE Trans. Power Syst.*, vol. 34, no. 6, 2019.

[7] D. Métivier, M. Vuffray, and S. Misra, "Efficient polynomial chaos expansion for uncertainty quantification in power systems," *Electric Power Systems Research*, vol. 189, p. 106791, 2020.

[8] L. Roald, S. Misra, M. Chertkov, and G. Andersson, "Optimal power flow with weighted chance constraints and general policies for generation control," in *Proc. IEEE Conf. on Decision and Control*. IEEE, 2015, pp. 6927–6933.

[9] A. Lorca and X. Sun, "The Adaptive Robust Multi-Period Alternating Current Optimal Power Flow Problem," *IEEE Trans. Power Syst.*, vol. 33, no. 2, pp. 1993–2003, 2018.

[10] D. Lee, K. Turitsyn, D. K. Molzahn, and L. Roald, "Robust AC optimal power flow with robust convex restriction," *IEEE Trans. Power Syst.*, 2021.

[11] M. Jalali, V. Kekatos, N. Gatsis, and D. Deka, "Designing reactive power control rules for smart inverters using support vector machines," *IEEE Trans. Smart Grid*, vol. 11, no. 2, pp. 1759–1770, Mar. 2020.

[12] Y. Ng, S. Misra, L. A. Roald, and S. Backhaus, "Statistical learning for dc optimal power flow," in *2018 Power Systems Computation Conference (PSCC)*. IEEE, 2018, pp. 1–7.

[13] D. Deka and S. Misra, "Learning for DC-OPF: Classifying active sets using neural nets," in *IEEE PowerTech*, Milan, Italy, Jun. 2019, pp. 1–6.

[14] A. S. Zamzam and K. Baker, "Learning optimal solutions for extremely fast ac optimal power flow," in *Proc. IEEE Intl. Conf. on Smart Grid Commun.* IEEE, 2020, pp. 1–6.

[15] Y. Chen and B. Zhang, "Learning to solve network flow problems via neural decoding," *arXiv preprint arXiv:2002.04091*, 2020.

[16] 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.*, Tempe, AZ, Nov. 2020, pp. 1–6.

[17] 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.*, 2021, (early access).

[18] S. Gupta, V. Kekatos, and M. Jin, "Deep learning for reactive power control of smart inverters under communication constraints," in *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, Tempe, AZ, 2020, pp. 1–6.

[19] ——, "Controlling smart inverters using proxies: A chance-constrained DNN-based approach," *IEEE Trans. Smart Grid*, vol. 13, no. 2, pp. 1310–1321, Mar. 2022.

[20] W. Huang and M. Chen, "DeepOPF-NGT: A fast unsupervised learning approach for solving AC-OPF problems without ground truth," in *ICML 2021 Workshop on Tackling Climate Change with Machine Learning*, Jul. 2021.

[21] P. L. Donti, D. Rolnick, and J. Z. Kolter, "DC3: A learning method for optimization with hard constraints," in *International Conference on Learning Representations*, May 2021.

[22] G. Wang, V. Kekatos, A.-J. Conejo, and G. B. Giannakis, "Ergodic energy management leveraging resource variability in distribution grids," *IEEE Trans. Power Syst.*, vol. 31, no. 6, Nov. 2016.

[23] V. Kekatos, G. Wang, and G. B. Giannakis, "Stochastic loss minimization for power distribution networks," in *Proc. North American Power Symposium*, Pullman, WA, Sep. 2014, pp. 1–6.

[24] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.

[25] 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*, vol. 67, no. 10, pp. 2775–2790, May 2019.

[26] A. Nemirovski and A. Shapiro, "Convex approximations of chance constrained programs," *SIAM Journal on Optimization*, vol. 17, no. 4, pp. 969–996, 2007.

[27] C. Chen and O. L. Mangasarian, "Smoothing methods for convex inequalities and linear complementarity problems," *Mathematical Programming*, vol. 71, no. 1, pp. 51–69, 1995.

[28] 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.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[29] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring, "GNU Octave version 6.1.0 manual: a high-level interactive language for numerical computations," 2020. [Online]. Available: https://www.gnu.org/software/octave/doc/v6.1.0/

[30] "Oct2py: Python to gnu octave bridge." [Online]. Available: https://oct2py.readthedocs.io/en/latest/index.html

[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*, vol. 9, no. 5, pp. 4282–4292, Sep. 2018.