Electric Power Systems Research 213 (2022) 108595

Contents lists available at ScienceDirect

Electric Power Systems Research

journal homepage: www.elsevier.com/locate/epsr

Learning to solve DCOPF: A duality approach

Yize Chen [a,*], Ling Zhang [b], Baosen Zhang [b]

[a] Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[b] University of Washington, Seattle, WA, USA

ARTICLE INFO

Keywords:
Machine learning
Optimal power flow

ABSTRACT

The optimal power flow (OPF) problem is a fundamental tool in power system operation and control. Because of the increase in uncertain renewable resources, solving OPF problems fast and accurately provides significant values because of a large number of load and generation scenarios need to be accounted for. Recent works have focused on using neural networks to replace iterative solvers to speed up the computation of OPF problems. A critical challenge is to ensure solutions satisfy the hard constraints in the OPF problem, which is difficult to do in end-to-end machine learning. In this work, by leveraging the rich theory of duality and physical interpretations of OPF, we design a learning-based approach that has theoretical characterizations of constraint satisfaction. This approach is an order of magnitude faster than standard solvers, and performs much better than other learning methods in terms of feasibility and optimality.

1. Introduction

The optimal power flow (OPF) problem is a fundamental tool used in power systems planning and operations [1–3]. The OPF problem finds the generator outputs that minimize the cost of generation while satisfying the power flow equations and other operational constraints. In this paper we consider the DCOPF formulation by linearizing the power flow equations [4].

The DCOPF problem has been studied extensively in the last sixty years and is a workhorse of the power industry [3]. If the generator cost is linear, the DCOPF is a linear program. If the costs are quadratic, then it is a quadratic program with linear constraints. Both types of optimization problems can be solved efficiently by a variety of algorithms, which have been implemented in a number of software packages [5,6]. Today, a DCOPF problem can be solved quickly for fairly large networks [4,7].

Because of the uncertainties brought by the renewables on many of the nodes, the number of generation and load scenarios that need to be considered are starting to grow exponentially [8–10]. Even if each scenario under consideration takes less than a second to solve using modern solvers, not all of them can be completed within the required time period. For example, if one instance of DCOPF can be solved in 0.5 s, then solving it for 2,000 scenarios would take more than 15 min, while outside the 5 min time resolution that real-time DCOPF are performed in practice. Therefore, using neural networks (NNs) to learn the mapping between input load profiles and the corresponding optimal generation outputs has gained significant attention, since making inference via a trained architecture can be potentially orders of magnitude faster than an iterative solver [11,12]. Many of application problems in power systems have been to use end-to-end supervised learning to find the mapping from input data to the optimal solution [13,14]. Such learning-based approaches can also help tasks such as hosting capacity analysis in distribution grids and strategic investments in energy markets [15,16].

Using end-to-end neural networks to directly make decisions about optimization problems has also been investigated in the computer science literature. Surprisingly, the answer to this question has been largely negative [17,18]. A key challenge is that the learned solution must satisfy hard constraints, which is difficult to enforce using fixed neural neural networks. In DCOPF, these constraints are the generator limits, the line flow limits and the power balance constraints. The generator limits can be built into a neural network by clipping the outputs (e.g., using the tanh function), yet the other constraints cannot be directly enforced [12]. One approach is to add costs to constraint violations during training [19], but the generalization performance is not certain [20], and it does not guarantee constraints are satisfied for new test samples and degrades optimality and training speeds. Some recent efforts have focused on the feasibility issues of learning-based OPF solvers [21], yet the solution is either highly dependent on the training performance [22], or requires specific design on correction steps of violated constraints [23].

Another drawback of end-to-end solvers is that they do not fully utilize additional information about the underlying physical systems.

* Corresponding author.
E-mail addresses: yizechen@lbl.gov (Y. Chen), lzhang18@uw.edu (L. Zhang), zhangbao@uw.edu (B. Zhang).

https://doi.org/10.1016/j.epsr.2022.108595
Received 3 October 2021; Received in revised form 17 April 2022; Accepted 2 July 2022

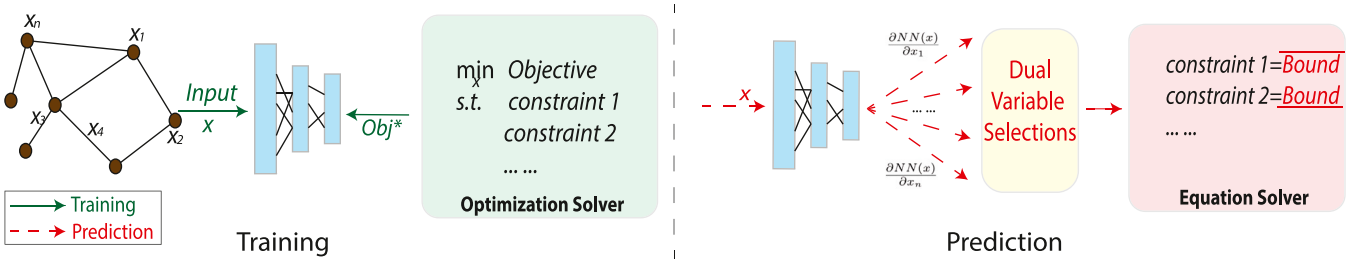

Fig. 1. The schematic of our proposed Neural Decoder for solving OPF problems. A neural network is trained to predict the optimal objective value; during implementation for solving OPF, the network's gradient is interpreted as a noisy codeword for active constraints, and a linear equation is solved to obtain optimal solutions.

DCOPF is often solved to find the locational marginal prices (LMPs) at the buses [4]. During probabilistic load forecasting, the system operator may be interested in finding the LMPs corresponding to each load scenario [11]. Most iterative solvers provide the LMPs as a byproduct of the optimal solution, but end-to-end methods cannot yield them easily. In addition, LMPs are not continuous in the load and thus are hard to learn directly using neural networks [14].

To overcome these challenges, we do not view learning DCOPF as an end-to-end task. Instead, we leverage the rich algorithmic understanding of DCOPF and convex optimization, as well as the economic interpretation of the primal and dual variables to offer a novel solution architecture. Our workflow is shown in Fig. 1. Concretely, we construct a neural network that takes the net load at each node as the input and outputs the optimal system cost (a scalar). Of course, the optimal value of the cost is not the solution of the LP. Rather, using the neural network, we compute the *gradient* of the cost with respect to the net loads. Identifying these as the *dual variables or the LMPs*, we use them to predict the binding nodal and line constraints. Once these constraints are identified, the optimal solution is given by solving a simple linear system of equations. The overall procedure can be seen as an efficient, robust surrogate learning model for optimization solvers.

We make the following contributions in this work:

(1) We provide a novel architecture to solve the DCOPF problem with both linear and quadratic costs. Resulting data-driven duality-based approach provide extremely efficient ways to identify the active constraints.
(2) Our method can be thought as decoding a system where the codebook is given by the KKT conditions of the DCOPF. It is *error correcting*, since even if output of the neural network is noisy, the correct set of active constraints at optimal solutions can still be identified.
(3) We show that our method can provide more than an order of magnitude speedup compared to current state-of-the-art iterative solvers and much better performance in terms of feasibility and optimality compared to other learning approaches.
(4) LMPs can be learned accurately since they serve as the intermediate step of our approach. Given a set of forecasted loads, their corresponding LMPs at the buses can be computed extremely efficiently.

2. Model and problem formulation

2.1. DCOPF formulation

We use the standard single period DC power flow model in this paper [24]. We assume the power system is connected and it has n buses and m lines. Let $\mathbf{x} \in \mathbb{R}^n$ denote the generation at the buses and $\boldsymbol{\ell} \in \mathbb{R}^n$ denote the load at the buses. The generation at bus i is bounded by 0 and $\bar{x}_i$,[1] and the upper bound for a load bus is set to be 0.

[1] The lower bound is set to be 0 for notional simplicity and without loss of generality, since the generation can always be shifted to make it 0.

We use the idea of fundamental flows to model the line flows between buses to make the subsequent derivations more succinct. Because of Kirchhoff's voltage law, a weighted linear combination of the flows in a cycle is always 0. Therefore, all of the flows lie in a $n-1$ subspace and the basis of this subspace is called the fundamental flows. Let $\mathbf{f} \in \mathbb{R}^{n-1}$ denote the set of fundamental flow. Let $\mathbf{K} \in \mathbb{R}^{m\times n-1}$ be the mapping from the fundamental flows to all flows. We use the fundamental flows to represent the physical constraints in the power grids, as we will show such formulation enjoys more structured identification processes of binding constraints. See Appendix A for an example and more details.

With the above notations, the power balance equations can be written as $\mathbf{x}+\tilde{\mathbf{A}}\mathbf{f}=\boldsymbol{\ell}$, where $\tilde{\mathbf{A}}$ is the modified admittance matrix from the fundamental flows to the buses. The DCOPF problem is:

$$J(\boldsymbol{\ell}) = \min_{\mathbf{x},\mathbf{f}} \quad \sum_{i=1}^{n} \frac{q_i}{2} x_i^2 + c_i x_i \tag{1a}$$

$$\text{s.t.} \quad \mathbf{0} \le \mathbf{x} \le \bar{\mathbf{x}} \tag{1b}$$

$$-\bar{\mathbf{f}} \le \mathbf{K}\mathbf{f} \le \bar{\mathbf{f}} \tag{1c}$$

$$\mathbf{x}+\tilde{\mathbf{A}}\mathbf{f}=\boldsymbol{\ell}, \tag{1d}$$

where q_i and c_i are the cost coefficients. The value of the optimization problem is denoted by J. And we sometimes write it as $J(\boldsymbol{\ell})$ to emphasis it is a function of the load.

In practice, both the quadratic costs and purely linear costs (with $q_i = 0$ for all i) are used [25]. These are also qualitatively different from an optimization perspective, since the solution for a linear program will be at an extreme point while the solution under a quadratic cost can be in the interior of the feasible space. It turns out that the solution methods are not exactly the same for these two cases, and we treat them separately in the later sections with detailed descriptions.

2.2. Price forecasting

In addition to the optimal solutions, the LMPs are often of interest in power system operations [25]. In this paper, we consider the setting of scenario-based price forecasting. We assume that a set of load scenarios is given (e.g., coming from a probabilistic load forecasting algorithm [26,27]). A straightforward method to compute the LMPs is to repeatedly solve the DCOPF problem, but that maybe too computationally inefficient.

2.3. Using machine learning

The central problem addressed in this paper is to replace the iterative solver block in Fig. 2(a) with faster blocks in Fig. 2(b). The input of the problem is a load vector and the output is the set of generation levels for the DCOPF problem and the LMPs for the price forecasting problem. The performance metrics are the computation speed, the feasibility of the output and their optimally.

As stated in the introduction, it is tempting to directly learn the optimal solution or the active constraints. However, both are nontrivial to learn, even for moderately sized systems experiencing high levels

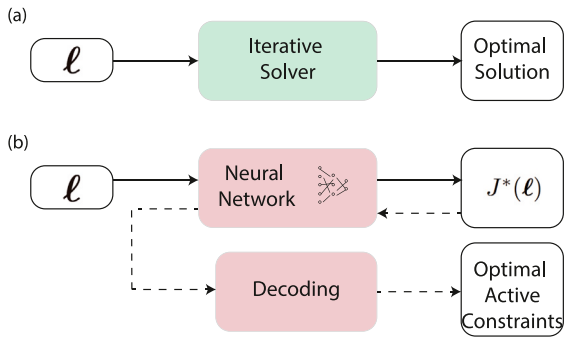

Fig. 2. Compared to standard optimization workflow (a), we propose to use trained neural network to give codewords for decoding active constraints at optimal solutions (b).

of uncertainty. The next section describes a learning methodology that circumvents the challenges of directly learning high dimensional outputs.

3. Solution architecture

Our solution architecture proceeds in two steps as shown in the right side of Fig. 1. The first step is to learn the dual variables associated with the load balance constraint in (1d). The second step is using the dual variables to find the active constraints. In this section we discuss how to learn the dual variables, and Sections 4 and 5 show how they are used to identify the active constraints.

3.1. Interpretation of dual variables

We use the following fact from power system engineering and linear programming:

Theorem 1. *Suppose the optimization problem in (1) is feasible for a given load $\boldsymbol{\ell}$. Let $\boldsymbol{\mu}^* \in \mathbb{R}^n$ denote the optimal dual variables associated with equality constraints (1d). Then $\nabla_{\boldsymbol{\ell}} J = \boldsymbol{\mu}^*$, where J is the optimal value of (1) and ∇ is the gradient operator.*

This theorem states that the multipliers associated with the power balance constraint in the DCOPF problem are the LMPs: the incremental cost of providing one more unit of power at the buses.[2] The proof of Theorem 1 is standard and can be found in [2,25].

The usefulness of Theorem 1 for our problem is that if the multipliers are known, then the set of optimal active constraints can be easily discovered. For example, consider DCOPF with linear costs ($q_i = 0$ for all i in (1a)). We have the following corollary about the active generation constraints:

Corollary 1. *Let $\boldsymbol{\mu}^*$ be the optimal multipliers of (1d) for a given load $\boldsymbol{\ell}$. Then the optimal generations are associated with the following active/inactive constraints:*

$$x_i = \begin{cases} \bar{x}_i, & \text{if } \mu_i^* - c_i > 0 \\ 0 & \text{if } \mu_i^* - c_i < 0 \\ (0, \bar{x}_i), & \text{otherwise} \end{cases} \tag{2}$$

The proof of the corollary is immediate from economic interpretations of the dual variable. If the LMP at a bus is higher than the cost at the bus, then the upper bound must be binding. Conversely, if the LMP is lower than the cost at a bus, the lower bound must be binding. Otherwise, the generator is on the margin and neither bounds are binding.

Determining the binding line flow constraints and binding generator constraints for quadratic costs are more complicated than Corollary 1, but not by much. We will cover how to find those constraints in Sections 4 and 5. We describe the learning methodology to find the optimal multiplier in the rest of this section.

3.2. Learning the LMPs

We do not directly use a neural network to learn $\boldsymbol{\mu}^*$, although it may seem to be the natural thing to do. Since $\boldsymbol{\mu}^*$ is not continuous in $\boldsymbol{\ell}$ but most neural networks are continuous in their inputs, a direct regression approach is hard to use while practical issues exist like data class imbalances. We can also think of learning $\boldsymbol{\mu}^*$ directly as a classification problem, but the problem is difficult because the large number of possible values as the system scales up.

Rather, we fit a neural network (parameterized by θ) $g_\theta(\boldsymbol{\ell})$ that maps optimization model input $\boldsymbol{\ell}$ to the value of the optimization problem $J(\boldsymbol{\ell})$. The value is a scalar function of the load. It is continuous and piecewise linear, with distinct "breakpoints", where the derivative changes value. Therefore, it is naturally parameterized by using ReLU activation units. To train the neural network, we use regression loss on two terms:

(1) Regression loss defined between $g_\theta(\boldsymbol{\ell})$ and $J^*(\boldsymbol{\ell})$ over the neural network's output;
(2) Regression loss for optimal dual variable defined between $\nabla_{\boldsymbol{\ell}} g_\theta(\boldsymbol{\ell})$ and $\boldsymbol{\mu}^*$.

The training loss is the sum of these terms

$$\mathscr{L}(\theta) = \|g_\theta(\boldsymbol{\ell}) - J^*(\boldsymbol{\ell})\|_2^2 + \gamma_1 \|\nabla_{\boldsymbol{\ell}} g_\theta(\boldsymbol{\ell}) - \boldsymbol{\mu}^*\|_2^2. \tag{3}$$

3.3. Fast adaptation to operating conditions

In Fig. 2(c), we show a practical use case for the proposed algorithm. Once g_θ is trained, its gradient $\nabla_{\boldsymbol{\ell}} g_\theta$ is our estimate of the LMPs $\boldsymbol{\mu}$. Taking the derivative of a neural network is equivalent to back propagation, which is highly optimized in modern machine learning platforms. For instance, such operations can be implemented via `tf.gradients()` in Tensorflow or `autograd.grad()` in PyTorch, while the computational cost is less than a millisecond thanks to automatic differentiation.

4. Linear costs

In this section, we assume that the LMPs have been learned, and we will describe how it can be used to determine both the active generator and line constraints when the cost is linear.

4.1. Lagrangian dual

The dual of (1) under linear cost ($q_i = 0$) is

$$\max_{\boldsymbol{\mu}, \bar{\boldsymbol{\lambda}}, \underline{\boldsymbol{\lambda}}, \bar{\boldsymbol{\nu}}, \underline{\boldsymbol{\nu}}} \quad \boldsymbol{\mu}^T \boldsymbol{\ell} - \underline{\boldsymbol{\lambda}}^T \underline{\bar{\mathbf{f}}} - \bar{\boldsymbol{\lambda}}^T \bar{\mathbf{f}} - \bar{\boldsymbol{\nu}}^T \bar{\mathbf{x}} \tag{4a}$$

$$\text{s.t.} \quad \mathbf{c} - \boldsymbol{\mu} - \underline{\boldsymbol{\nu}} + \bar{\boldsymbol{\nu}} = \mathbf{0} \tag{4b}$$

$$-\tilde{\mathbf{A}}^T \boldsymbol{\mu} - \mathbf{K}^T \underline{\boldsymbol{\lambda}} + \mathbf{K}^T \bar{\boldsymbol{\lambda}} = \mathbf{0} \tag{4c}$$

$$\bar{\boldsymbol{\nu}} \geq \mathbf{0}, \underline{\boldsymbol{\nu}} \geq \mathbf{0}, \bar{\boldsymbol{\lambda}} \geq \mathbf{0}, \underline{\boldsymbol{\lambda}} \geq \mathbf{0}. \tag{4d}$$

If $\boldsymbol{\mu}$ is given, the dual problem decouples into two separate parts: one for generators and the one for line flows.

The part of the dual associated with generators is

$$\max_{\bar{\boldsymbol{\nu}}, \underline{\boldsymbol{\nu}}} \quad -\bar{\boldsymbol{\nu}}^T \bar{\mathbf{x}} \tag{5a}$$

[2] Technically Theorem 1 holds for all most all feasible load (except for a set of measure zero). For loads that are at the exact boundary when binding constraint changes, the gradient need to be replaced by a subgradient.

$$\text{s.t.} \quad \mathbf{c} - \boldsymbol{\mu} - \underline{\boldsymbol{\nu}} + \bar{\boldsymbol{\nu}} = \mathbf{0} \tag{5b}$$

$$\bar{\boldsymbol{\nu}} \geq \mathbf{0}, \underline{\boldsymbol{\nu}} \geq \mathbf{0}. \tag{5c}$$

By inspection, the optimal solution of (5) follows the structure given in Corollary 1.

Identifying the binding line flows is a more interesting problem. The part of the dual associated with the flows is

$$\max_{\bar{\lambda}, \underline{\lambda}} \quad -\underline{\lambda}^T \bar{\mathbf{f}} - \bar{\lambda}^T \bar{\mathbf{f}} \tag{6a}$$

$$\text{s.t.} \quad \mathbf{K}^T(\bar{\lambda} - \underline{\lambda}) = \tilde{\mathbf{A}}^T \boldsymbol{\mu}^* \tag{6b}$$

$$\bar{\lambda} \geq \mathbf{0}, \ \underline{\lambda} \geq \mathbf{0}. \tag{6c}$$

The solution structure of (6) is not as apparent as the one for generators. It turns out to be useful to transform (6) into an equivalent LP using the following lemma.

Lemma 1. *The problem (6) is equivalent to*

$$\min_{\mathbf{y}} \quad \|\mathbf{y}\|_1 \tag{7a}$$

$$s.t. \quad \hat{\mathbf{K}}^T \mathbf{y} = \tilde{\mathbf{A}}^T \boldsymbol{\mu}^* \tag{7b}$$

where $y_i = \bar{f}_i \cdot \bar{\lambda}_i - \bar{f}_i \cdot \underline{\lambda}_i$, $|y_i| = |\bar{f}_i \cdot \bar{\lambda}_i - \bar{f}_i \cdot \underline{\lambda}_i|$, *and* $\hat{\mathbf{K}} = diag(1/\bar{f}_1, \ldots, 1/\bar{f}_m)\mathbf{K}$.

The proof of Lemma 1 is given in Appendix B. The importance of the transformed $\mathcal{L}_1$ minimization problem in (7) is that it is extremely well studied. This problem is actually the canonical form of the sparse signal recovery in compressed sensing, where signal $\mathbf{y}$ needs to be recovered via observation $\tilde{\mathbf{A}}^T \boldsymbol{\mu}^*$. For more information, please see [28–30] and the references within. Note that in sparse recovery, $\mathcal{L}_1$ minimization is a surrogate problem where the ultimate goal is to find the sparsest solution to the problem. For us, (7) is the exact problem we want to solve to finish the active constraints identification.

Because there are limited number of active constraints which is equivalent to limited number of nonzero entries in $\mathbf{y}$, while $\hat{\mathbf{K}}$ is very sparse, there are many algorithms that can solve (7) extremely efficiently. We use a family of greedy algorithms such as iterative hard thresholding can be used to find $\mathbf{y}$ [31] in a fixed number of iterations.

4.2. Solving for the optimal solutions

Assuming that the DCOPF problem with linear cost is not degenerate, there are exactly the same number of binding constraints as there are variables [32]. That is, we would have $2n-1$ equations from the binding generators and the line flow constraints, corresponding to the n generation variables and $n-1$ fundamental flow variables. This linear equation can be solved to find the optimal $\mathbf{x}^*$ and $\mathbf{f}^*$.

This linear system of equations is much easier to solve than the full DCOPF. Firstly, it has $2n-1$ equations, much smaller than the $3n+2m$ constraints in the DCOPF problem. Secondly, the system is quite sparse, allowing us to use a variety of techniques. In Section 6 we show that the speedup in computation time is significant.

4.3. Robustness to errors

Because we are dealing with continuous values, the prediction $J(\boldsymbol{\ell})$ of the neural network will invariably have errors. Therefore, it is important that the errors made do not add up and cause incorrect identification of the active constraints. Using an analogy from communication theory, we think of the derivatives of the learned neural network $g_\theta(\boldsymbol{\ell})$ as *noisy versions of a codeword*. We are essentially providing an error-correcting approach to decode active constraints that is robust to errors made by the neural network.

Observe that for a given $\boldsymbol{\mu}^*$ and some noise $\boldsymbol{\delta}$, though the optimal solution for the optimization problem (6) may be different for $\boldsymbol{\mu}^*$ and $\boldsymbol{\mu}^* + \boldsymbol{\delta}$, the set of active constraints at optimal solutions can remain the same. Therefore, there exists a region around a ground truth optimal dual solution where as long as the noise does not push the solution outside of this region, the set of active constraints remains the same. It turns out these regions are polytopes and easily characterized by solving another linear program. This falls under the well studied area of linear programming sensitivity analysis [32,33], and is formalized by the next Lemma

Lemma 2. *Consider a given $\boldsymbol{\ell}$ and its associated optimal dual variables $\boldsymbol{\mu}^*$. There exists a polytope $\mathcal{P}_{\boldsymbol{\mu}^*}$ around $\boldsymbol{\mu}^*$ such that if $\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}^* \in \mathcal{P}_{\boldsymbol{\mu}^*}$, then the active constraints determined using $\tilde{\boldsymbol{\mu}}$ is correct. The set $\mathcal{P}_{\boldsymbol{\mu}^*}$ is computed by a linear program.*

The proof of this lemma is given in Appendix C. It shows that solutions are robust, since we do not require that the value of predictions $\nabla_{\boldsymbol{\ell}} g_\theta(\boldsymbol{\ell})$ to be exact, and there is no compounding of errors as compared to other end-to-end learning approaches. It also provides a way to check whether training is good enough: we can compare the error between the learned $\boldsymbol{\mu}$ and the actual $\boldsymbol{\mu}^*$, and if it falls within the polytope $\mathcal{P}_{\boldsymbol{\mu}^*}$, then we are confident that the training results would be accurate.

4.4. Augmenting neural network training

We can augment the training loss in (3) to include the accuracy of detecting the active constraints. Let $h(g_\theta(\boldsymbol{\ell}))$ denote the binary vector indicating the active constraints determined by learner's solution based on the procedure described in the last section. We can add the hamming distance between $h(g_\theta(\boldsymbol{\ell}))$ and $s^*(\boldsymbol{\ell})$, where $s^*(\cdot)$ is the binary vector indicating the true active constraints. The new training loss is

$$\begin{aligned}\mathscr{L}(\theta) =& \|g_\theta(\boldsymbol{\ell}) - \boldsymbol{J}^*(\boldsymbol{\ell})\|_2^2 + \gamma_1 \|\nabla_{\boldsymbol{\ell}} g_\theta(\boldsymbol{\ell}) - \boldsymbol{\mu}^*\|_2^2 \\ &+ \gamma_2 \|h(g_\theta(\boldsymbol{\ell})) - s^*(\boldsymbol{\ell})\|_H\end{aligned} \tag{8}$$

where γ_1, γ_2 are tuning parameters. For detailed implementation, please refer to https://github.com/chennnnnyize/Neural-Decoding-for-OPF.

5. Quadratic costs

As in Section 4, we assume that the LMPs have been learned and describe how it can be used to determine the active constraints when cost is quadratic. If q_i's are not zero, the dual of (1) is

$$\max_{\boldsymbol{\mu}, \bar{\lambda}, \underline{\lambda}, \bar{\nu}, \underline{\nu}} \quad \boldsymbol{\mu}^T \ \boldsymbol{\ell} - \underline{\lambda}^T \bar{\mathbf{f}} - \bar{\lambda}^T \bar{\mathbf{f}} - \bar{\nu}^T \bar{\mathbf{x}} \tag{9a}$$

$$\text{s.t.} \quad \mathbf{Q}\mathbf{x} + \mathbf{c} - \boldsymbol{\mu} - \underline{\boldsymbol{\nu}} + \bar{\boldsymbol{\nu}} = \mathbf{0} \tag{9b}$$

$$-\tilde{\mathbf{A}}^T \boldsymbol{\mu} - \mathbf{K}^T \underline{\lambda} + \mathbf{K}^T \bar{\lambda} = \mathbf{0} \tag{9c}$$

$$\bar{\boldsymbol{\nu}} \geq \mathbf{0}, \underline{\boldsymbol{\nu}} \geq \mathbf{0}, \bar{\lambda} \geq \mathbf{0}, \underline{\lambda} \geq \mathbf{0}, \tag{9d}$$

where $\mathbf{Q}$ is a diagonal matrix with the value of q_i on the i'th diagonal. There are two differences between (4) and (9). The first is that the constraint associated with the generations, (9b), also include the primal variables $\mathbf{x}$. The second is that unlike a linear program, a quadratic program may not have the same number of binding constraints as the variables.

5.1. Binding constraints

Again we assume $\boldsymbol{\mu}$ is known (coming from the gradient of the learned J). We use the following lemma to determine whether a generator constraint is binding:

Lemma 3. *Given the optimal LMP $\boldsymbol{\mu}^*$, x_i is associated with the following active/inactive constraints:*

$$x_i = \begin{cases} \bar{x}_i, & \text{if } \mu_i^* - c_i - q_i \bar{x}_i > 0 \\ 0 & \text{if } \mu_i^* - c_i < 0 \\ (0, \bar{x}_i), & \text{otherwise} \end{cases} \tag{10}$$

This lemma shows that the binding generation constraints can again be found through simple comparisons.

Proof. The rule in (10) follows from simple economic principles. If a generator is marginal, then its LMP is $\mu_i^* = c_i + 2q_i x_i$. Otherwise, a generator is binding at its upper bound if $\mu_i^* > c_i + 2q_i \bar{x}_i$ and at its lower bound if $\mu_i^* < c_i$. □

The binding line constraints can be recovered through the same process as the linear cost case. Once $\boldsymbol{\mu}$ is known, the dual problem associated with $\underline{\lambda}$ and $\overline{\lambda}$ is the same as (6).

5.2. Finding the optimal solutions

Once we identify all of the binding constraints, we can encode it into a matrix of the form $\mathbf{My} = \mathbf{a}$, where $\mathbf{y}$ is the concatenation of $\mathbf{x}$ and $\mathbf{f}$. For a quadratic program, the number of constraints (rows of $\mathbf{M}$) can be less then the number of variables. Therefore, we still need to solve the following optimization problem:

$$\min \frac{1}{2}\mathbf{y}^T \hat{\mathbf{Q}}\mathbf{y} + \hat{\mathbf{c}}^T\mathbf{y} \tag{11a}$$

$$\text{s.t. } \mathbf{My} = \mathbf{a}, \tag{11b}$$

where $\hat{\mathbf{Q}}$ is a diagonal matrix with $(q_1, \dots, q_n, 0, \dots 0)$ on its diagonal and $\hat{\mathbf{c}} = (c_1, \dots, c_n, 0, \dots, 0)$. They come from the fact that costs are not assigned to the flows. Fortunately (11) can be solved as a linear system:

Lemma 4. *Let $\boldsymbol{\tau}^*$ be the optimal Lagrangian multiplier of (11b), then the optimal solution of (11) is given by the following linear system*

$$\begin{bmatrix} \hat{\mathbf{Q}} & \mathbf{M}^T \\ \mathbf{M} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y}^* \\ \boldsymbol{\tau}^* \end{bmatrix} = \begin{bmatrix} -\hat{\mathbf{c}} \\ \mathbf{a} \end{bmatrix}. \tag{12}$$

The significance of the lemma is that solving the problem with quadratic costs is no more difficult than solving it with linear costs. Once the active constraints are identified, a linear system of equations can be solved to find the optimal solutions.

Proof. The Lagrangian of (12) is

$$\frac{1}{2}\mathbf{y}^T \hat{\mathbf{Q}}\mathbf{y} + \hat{\mathbf{c}}^T\mathbf{y} + \boldsymbol{\tau}^T(\mathbf{My} - \mathbf{a}),$$

and differentiating with respect to $\mathbf{y}$ gives the stationarity condition

$$\hat{\mathbf{Q}}\ + \hat{\mathbf{c}} - \mathbf{M}^T\boldsymbol{\tau} = 0. \tag{13}$$

Combining (13) with the equality constraint (11b) gives the system of equations in (12). □

6. Case studies

6.1. Experiment setup

We evaluate the proposed learning approach, the *Neural Decoder*, on the IEEE 14-Bus and 39-Bus system [34] for both linear and quadratic costs. Specifically, over a wide range of problem input settings, we examine (1) solution quality in terms of constraint satisfaction and optimality and (2) computational efficiency over existing convex optimization solvers. Simulations are run on an unloaded Macbook Pro with Intel Core i5 8259U CPU @2.30 GHz. The code and simulation data can be found at https://github.com/chennnnnyize/Neural-Decoding-for-OPF.

To generate the training set, we use CVXPY [35] powered by a CVXOPT solver [36] to solve (1). For each setting, we generate $\boldsymbol{\ell}$ by sampling from uniform distribution, with variations of 20%, 50% and 80%. We solve 60,000 data samples using CVXPY for each network model under each variance setting, and split 20% of the data as test samples.

We train and compare three other learning models in terms of finding optimal solutions while satisfying all constraints:

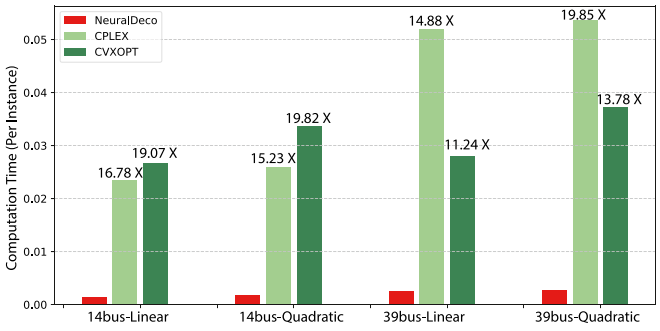

Fig. 3. Average solving time per instance along with speedup statistics comparison for Neural Decoder, iterative solvers CVXOPT and CPLEX.

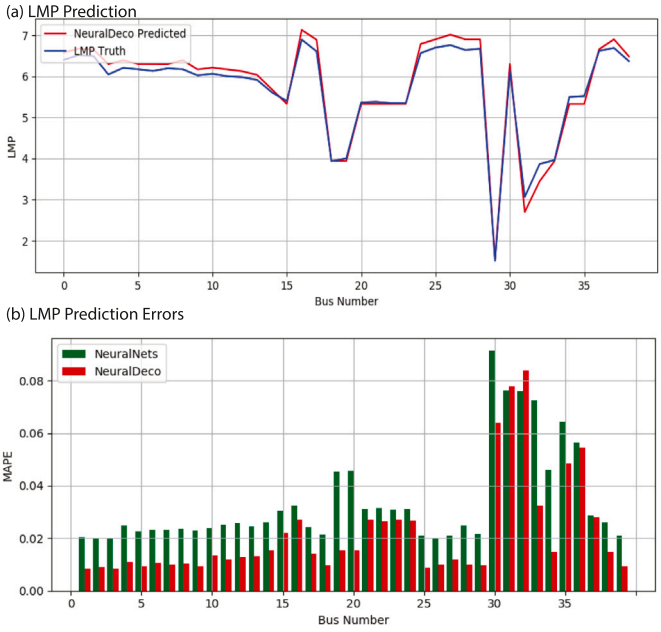

Fig. 4. Simulation results on LMP forecasting on 39-bus system with 80% load variations. (a) LMPs of a single sample and (b). Average MAPE across all testing samples compared with neural network-based forecasts.

(1) *Nearest neighbor for active constraints:* This benchmark is a sanity check on whether a deep neural network is needed or a simpler method would suffice. We fit and find a 3-nearest neighbor algorithm achieves the highest classification accuracy in predicting the active constraints.
(2) *End-to-end regression:* Following [12], we construct a 4-layer neural network to fit the regression task of predicting optimal solution based on input $\boldsymbol{\ell}$. Mean squared error is used as training loss.
(3) *Classification for active constraint sets:* Following [37], we construct a 4-layer neural network to predict the set of active constraints at optimal solution. We use one-hot encoding for different set of active constraints, and use cross-entropy as the loss function.

We term our framework the **Neural Decoder** and use a 3-layer neural network with 200 neurons on the first layer. For all machine learning models, we keep a validation dataset and avoid training overfitting during the training stage. We feed load vector $\boldsymbol{\ell}$ as the input for all of the methods. Once the active constraints are predicted, a linear equation solver is used to find final solutions.

Table 1
Performance comparison on linear cost case with different load input variations.

		80%–120%				50%–150%				20%–180%			
		NeuralDeco	E-to-E	Class	Neighbor	NeuralDeco	E-to-E	Class	Neighbor	NeuralDeco	E-to-E	Class	Neighbor
14-Bus	Generator	98.49	96.36	94.64	89.03	96.15	96.04	78.07	88.92	93.19	90.39	71.37	73.65
	Line	99.91	96.06	98.52	92.44	98.16	95.78	98.06	89.68	93.47	93.77	94.07	84.33
	Infeasibility	0.77	3.36	6.63	11.28	1.86	3.86	21.92	18.17	4.03	13.69	24.62	29.19
39-Bus	Generator	99.91	93.30	98.95	93.40	96.39	70.95	95.28	77.98	93.95	44.38	90.19	63.90
	Line	99.91	94.85	99.24	59.68	96.33	88.03	95.96	78.64	93.48	72.22	86.28	71.34
	Infeasibility	0.02	11.07	0.01	29.38	0.56	46.53	2.62	21.68	4.25	61.25	13.49	38.71

Table 2
Performance comparison on quadratic cost case with different load input variations.

		80%–120%				50%–150%				20%–180%			
		NeuralDeco	E-to-E	Class	Neighbor	NeuralDeco	E-to-E	Class	Neighbor	NeuralDeco	E-to-E	Class	Neighbor
14-Bus	Generator	93.43	95.39	89.73	92.44	92.06	68.26	69.63	73.52	94.19	78.68	68.71	65.18
	Line	98.61	93.81	98.85	92.90	92.02	79.69	92.00	86.78	81.01	72.55	67.51	74.51
	Infeasibility	2.12	6.01	4.07	5.82	2.48	50.08	9.37	28.56	8.18	31.87	17.81	29.64
39-Bus	Generator	97.95	96.87	98.12	97.99	86.56	46.16	31.32	40.34	85.08	36.71	29.06	14.28
	Line	99.08	99.18	99.45	98.97	85.85	38.51	32.18	33.67	84.32	42.82	56.18	21.07
	Infeasibility	0.38	2.84	0.42	1.14	7.82	63.24	46.59	53.72	11.63	73.52	58.62	78.92

6.2. Simulation results

Results on learning for the 14-bus and 39-bus OPF problem with linear costs and quadratic costs are listed in Tables 1 and 2 respectively, where we report the accuracy of active generators' constraints and lines' constraints separately. Note that these simulation load samples are not sufficient to cover the input space and some test samples that do not reside in the same region as the training samples. Our method generalizes and has the lowest infeasibility across all test settings. For feasible test instances, the mean solution costs compared to optimal solutions provided by CVXPY is within 0.5%.

Feasibility and Optimality. Interestingly, our proposed method (NeuralDeco) can provide solutions with lower percentage of infeasibility than the sum of error on active generator and line constraints predictions. This is because identification steps of active generators and active lines are performed in sequence and we are making use of the information on total number of active constraints in the linear case, so the error on each step does not compound and lead to infeasible solutions.

The other methods perform much poorer. The nearest neighbor approach is not able to achieve good performance when the input variations are greater, showing that more sophisticated learning approaches are needed. The end-to-end prediction is hard to use for higher load variances, mainly because it does not explicitly include line flow constraints, so it tends to produce infeasible flow solutions. The classification approach is also not a proper learning strategy for larger-scale optimization problems, since the growing number of possible combination of active constraints leads to huge one-hot encodings at the classifier's output.

Computation Time. Compared to the solving process of optimization solvers, our proposed solver provides significant speed-up in all testing benchmarks as shown in Fig. 3. Compared to state-of-the-art commercial solvers, it can provide an order of magnitude in efficiency. We are using standard Python packages for neural network derivations and equation solvers, while further acceleration can be achieved by taking a batch of evaluating samples to calculate $\nabla_{\ell} g_{\theta}(\ell)$, or adopting special linear equation solvers to solve the resulting sparse linear equations once all active constraints are identified. Please see [38] for more details and statistics, as well as just comparing the "core" computation times. That is, we discount the overhead in problem conversion, constraint translation and so on, and only record the time used by lower level linear algebra packages. The relative speeds remain virtually unchanged in this comparison from Fig. 3.

Price Forecasting. The results are shown in Fig. 4 with forecasted prices and forecasting errors. For the case of 39-bus with 80% input variations, the overall LMP forecasting mean absolute percentage error (MAPE) on test samples is 2.15%. We also compare proposed method to a straightforward forecasting method, where we trained a neural network with nodal electricity demand as input and LMP as output, whose error is about 50% larger.

7. Discussion and conclusion

In this paper, we propose a novel machine learning paradigm to tackle both the computation and feasibility challenges in OPF. Built upon the foundations of convex optimization and algorithmic understanding of DCOPF, proposed method is able to efficiently find the OPF solutions given load vectors with large fluctuations. We demonstrate its speed advantage over conventional iterative solvers and better feasibility performances compared to other machine learning methods. We show that by incorporating the underlying grid topology and parameters, it is possible to design machine learning algorithms with guarantee. In the future work, we will explore the potentials of using machine learning for ACOPF problems by again leveraging the interpretations of active constraints and the dual variables (LMPs), and design fast learning-based decision-maker for safe power system operations.

CRediT authorship contribution statement

Yize Chen: Conceptualization, Methodology, Software. **Ling Zhang:** Data curation, Review & editing. **Baosen Zhang:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Fundamental flows

In the DC power flow model the power flow on the lines are determined by the angle differences. Let θ_i be the angle of bus i and $f_{ij} = b_{ij}(\theta_i - \theta_j)$ be the flow along the line connecting i and j. If a network has cycles, let buses $1, \ldots, n_c$ be the buses in a cycle, counted

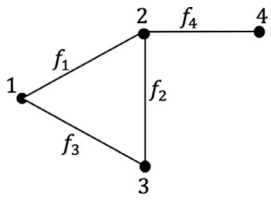

Fig. 5. Example of DC power flow model.

in either the clockwise or counterclockwise direction. Consider the weighted sum

$$\frac{f_{12}}{b_{12}} + \frac{f_{23}}{b_{23}} + \cdots + \frac{f_{n_c 1}}{b_{n_c 1}}$$
$$=(\theta_1 - \theta_2) + (\theta_2 - \theta_3) + \cdots + (\theta_{n_c} - \theta_1)$$
$$=0.$$

Therefore, the flows lie in a subspace and are not independent from each other. Repeating the above calculation for every cycle gives that the flows lie in a subspace of dimension $n-1$ for a connected network with n buses. A basis of this subspace is called a set of fundamental flows. A basis can be constructed by choosing a spanning tree and consider the flows on the branches as fundamental, and everything else can be derived from these flows.

Fig. 5 shows an example. Assuming that all the line susceptances are 1 p.u., i.e., $b_{ij} = 1$, and taking the line flows f_1, f_2, f_4 to the fundamental flows. Then the matrix $\mathbf{K}$ mapping the fundamental flows to all line flows and the matrix $\tilde{\mathbf{A}}$ mapping fundamental flows to bus injections are:

$$\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \tilde{\mathbf{A}} = \begin{bmatrix} 2 & 1 & 0 \\ -1 & 1 & 1 \\ -1 & -2 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

Appendix B. Proof of Lemma 1

Proof. The original optimization problem is reproduced below:

$$\max_{\bar{\lambda},\underline{\lambda}} \quad -\underline{\lambda}^T \underline{\mathbf{f}} - \bar{\lambda}^T \bar{\mathbf{f}} \tag{14a}$$
$$\text{s.t.} \quad \mathbf{K}^T(\bar{\lambda} - \underline{\lambda}) = \tilde{\mathbf{A}}^T \boldsymbol{\mu}^* \tag{14b}$$
$$\bar{\lambda} \geq \mathbf{0},\ \underline{\lambda} \geq \mathbf{0}. \tag{14c}$$

Let $\boldsymbol{y} = \bar{\mathbf{f}} \odot \bar{\lambda} + \bar{\mathbf{f}} \odot \underline{\lambda}$ and $\mathbf{v} = \bar{\lambda} - \underline{\lambda}$, where $\odot$ is component-wise multiplication. Then the optimization problem in (6) becomes

$$\min_{\boldsymbol{y},\mathbf{v}} \quad \sum_{i=1}^{n} y_i \tag{15a}$$
$$\text{s.t.} \quad \mathbf{K}^T \mathbf{v} = \tilde{\mathbf{A}}^T \boldsymbol{\mu}^* \tag{15b}$$
$$\boldsymbol{y} \geq \bar{\mathbf{f}} \odot \mathbf{v} \tag{15c}$$
$$\boldsymbol{y} \geq -\bar{\mathbf{f}} \odot \mathbf{v}, \tag{15d}$$

where the last two inequalities come from the nonnegativity constraint of $\underline{\lambda}$. Suppose $\boldsymbol{y}, \mathbf{v}$ are optimal solutions. Because we are minimizing the sum of the components of $\boldsymbol{y}$, $y_i = \bar{f}_i \max(v_i, -v_i)$. This is equivalent to $y_i = \bar{f}_i |v_i|$. □

Appendix C. Proof of Lemma 2

We use the following lemma:

Lemma 5. *For an LP problem* $\{\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ *with* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^m, m < n$, *let* $(1), (2), \ldots, (m)$ *be the indices of selected columns of* $\mathbf{A}$, *such that* $\boldsymbol{B}\boldsymbol{x}^* = \mathbf{b}$ *with* $\mathbf{B} = [\mathbf{a}_{(1)} \ \ldots \ \mathbf{a}_{(m)}] \in \mathbb{R}^{(m\times m)}$ *as an invertible basis. For any* $\tilde{\mathbf{b}} = \mathbf{b} + \boldsymbol{\delta}$, *the optimal solution is still given by* $\mathbf{B}$ *if and only if* $\boldsymbol{B}^{-1}\mathbf{b} + \boldsymbol{B}^{-1}\boldsymbol{\delta} \geq \mathbf{0}$.

The proof of Lemma 2 follows from Lemma 5 by converting our LP of interest to the standard form. Despite the latter being a known result in linear programming, we have not been able to find the proof of the vector form in existing literature (the component by component result can be found in [32,33]). Therefore we provide the following proof for completeness.

Proof. To find the region where $\tilde{\mathbf{b}} = \mathbf{b} + \boldsymbol{\delta}$ has the same set of active constraints at the optimal solution, denote the new optimal solution as $\tilde{\mathbf{x}}^*$ that satisfies $\mathbf{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$. So the new optimization problem involving $\tilde{\mathbf{b}}$ becomes

$$\tilde{\mathbf{x}} = \arg\min \quad \mathbf{c}^T\mathbf{x} \tag{16a}$$
$$s.t. \quad \mathbf{A}\mathbf{x} = \mathbf{b} + \boldsymbol{\delta} \tag{16b}$$
$$\mathbf{x} \geq \mathbf{0} \tag{16c}$$

Since we have $\mathbf{x}^* = \mathbf{B}^{-1}\mathbf{b} \geq 0$ with input $\mathbf{b}$, and since $\mathbf{x}^*$ and $\tilde{\mathbf{x}}$ can be represented by the same basis $\mathbf{B} \in \mathbb{R}^{m\times m}$, we have

$$\mathbf{B}^{-1}(\mathbf{b} + \boldsymbol{\delta}) \geq \mathbf{0} \tag{17}$$

to ensure the optimal solution's feasibility. So the resulting $\boldsymbol{\delta}$ must satisfy (17).

On the other hand, if $\mathbf{B}^{-1}(\mathbf{b} + \boldsymbol{\delta}) \geq 0$, while $\tilde{\mathbf{x}} = \mathbf{B}^{-1}(\mathbf{b} + \boldsymbol{\delta})$ satisfies both equality and inequality constraints. By checking the KKT conditions, it is also the optimal solution, which completes the proof. □

References

[1] H.W. Dommel, W.F. Tinney, Optimal power flow solutions, IEEE Trans. Power Appar. Syst. (10) (1968) 1866–1876.
[2] R. Baldick, Applied Optimization: Formulation and Algorithms for Engineering Systems, Cambridge University Press, 2006.
[3] J.D. Glover, T.J. Overbye, M.S. Sarma, Power System Analysis and Design, CENGAGE Learning, 2017.
[4] B. Stott, J. Jardim, O. Alsaç, DC power flow revisited, IEEE Trans. Power Syst. 24 (3) (2009) 1290–1300.
[5] R.D. Zimmerman, C.E. Murillo-Sánchez, R.J. Thomas, MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education, IEEE Trans. Power Syst. 26 (1) (2010) 12–19.
[6] F. Milano, An open source power system analysis toolbox, IEEE Trans. Power Syst. 20 (3) (2005) 1199–1206.
[7] F. Li, R. Bo, DCOPF-based LMP simulation: algorithm, comparison with ACOPF, and sensitivity, IEEE Trans. Power Syst. 22 (4) (2007) 1475–1485.
[8] A. Hauswirth, S. Bolognani, G. Hug, F. Dörfler, Projected gradient descent on Riemannian manifolds with applications to online power system optimization, in: 2016 54th Annual Allerton Conference on Communication, Control, and Computing, Allerton, IEEE, 2016, pp. 225–232.
[9] Y. Zhang, E. Dall'Anese, M. Hong, Dynamic ADMM for real-time optimal power flow, in: 2017 IEEE Global Conference on Signal and Information Processing, GlobalSIP, IEEE, 2017, pp. 1085–1089.
[10] S. Huang, V. Dinavahi, Fast batched solution for real-time optimal power flow with penetration of renewable energy, IEEE Access 6 (2018) 13898–13910.
[11] W. Deng, Y. Ji, L. Tong, Probabilistic forecasting and simulation of electricity markets via online dictionary learning, 2016, arXiv preprint arXiv:1606.07855.
[12] X. Pan, T. Zhao, M. Chen, Deepopf: Deep neural network for dc optimal power flow, in: 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids, SmartGridComm, IEEE, 2019, pp. 1–6.
[13] R. Canyasse, G. Dalal, S. Mannor, Supervised learning for optimal power flow as a real-time proxy, in: 2017 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference, ISGT, IEEE, 2017, pp. 1–5.
[14] Y. Ji, Operation Under Uncertainty in Electric Grid: A Multiparametric Programming Approach (Ph.D. thesis), Cornell University, 2017.
[15] S. Taheri, V. Kekatos, H. Veeramachaneni, Strategic investment in energy markets: A multiparametric programming approach, 2020, arXiv preprint arXiv:2004.06483.
[16] S. Taheri, M. Jalali, V. Kekatos, L. Tong, Fast probabilistic hosting capacity analysis for active distribution systems, IEEE Trans. Smart Grid 12 (3) (2020) 2000–2012.

[17] B. Amos, J.Z. Kolter, Optnet: Differentiable optimization as a layer in neural networks, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 136–145.
[18] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, J.Z. Kolter, Differentiable convex optimization layers, in: Advances in Neural Information Processing Systems, 2019, pp. 9558–9570.
[19] X. Pan, T. Zhao, M. Chen, Deepopf: A deep neural network approach for security-constrained dc optimal power flow, 2019, arXiv preprint arXiv:1910.14448.
[20] L. Zhang, Y. Chen, B. Zhang, A convex neural network solver for dcopf with generalization guarantees, IEEE Trans. Control Netw. Syst. (2021).
[21] M.K. Singh, S. Gupta, V. Kekatos, G. Cavraro, A. Bernstein, Learning to optimize power distribution grids using sensitivity-informed deep neural networks, in: 2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids, SmartGridComm, IEEE, 2020, pp. 1–6.
[22] T. Zhao, X. Pan, M. Chen, A. Venzke, S.H. Low, DeepOPF+: A deep neural network approach for DC optimal power flow for ensuring feasibility, in: 2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids, SmartGridComm, IEEE, 2020, pp. 1–6.
[23] P.L. Donti, D. Rolnick, J.Z. Kolter, DC3: A learning method for optimization with hard constraints, in: International Conference on Learning Representations, 2020.
[24] J.D. Glover, M.S. Sarma, T. Overbye, Power System Analysis & Design, SI Version, Cengage Learning, 2012.
[25] D.S. Kirschen, G. Strbac, Fundamentals of Power System Economics, John Wiley & Sons, 2018.
[26] J. Xie, T. Hong, Temperature scenario generation for probabilistic load forecasting, IEEE Trans. Smart Grid 9 (3) (2016) 1680–1687.
[27] Y. Chen, Y. Wang, D. Kirschen, B. Zhang, Model-free renewable scenario generation using generative adversarial networks, IEEE Trans. Power Syst. 33 (3) (2018) 3265–3275.
[28] D.L. Donoho, Compressed sensing, IEEE Trans. Inform. Theory 52 (4) (2006) 1289–1306.
[29] J.A. Tropp, A.C. Gilbert, Signal recovery from random measurements via orthogonal matching pursuit, IEEE Trans. Inform. Theory 53 (12) (2007) 4655–4666.
[30] Y.C. Eldar, G. Kutyniok, Compressed Sensing: Theory and Applications, Cambridge University Press, 2012.
[31] T. Blumensath, M.E. Davies, Iterative thresholding for sparse approximations, J. Fourier Anal. Appl. 14 (5–6) (2008) 629–654.
[32] D. Bertsimas, J.N. Tsitsiklis, Introduction to Linear Optimization, Vol. 6, Athena Scientific Belmont, MA, 1997.
[33] B. Jansen, J. De Jong, C. Roos, T. Terlaky, Sensitivity analysis in linear programming: just be careful!, European J. Oper. Res. 101 (1) (1997) 15–28.
[34] T. Athay, R. Podmore, S. Virmani, A practical method for the direct analysis of transient stability, IEEE Trans. Power Appar. Syst. (2) (1979) 573–584.
[35] S. Diamond, S. Boyd, CVXPY: A python-embedded modeling language for convex optimization, J. Mach. Learn. Res. 17 (1) (2016) 2909–2913.
[36] M.S. Andersen, J. Dahl, L. Vandenberghe, CVXOPT: A python package for convex optimization, version 1.1. 6, 2013, 54 Available At Cvxopt. Org.
[37] Y. Ng, S. Misra, L.A. Roald, S. Backhaus, Statistical learning for DC optimal power flow, in: 2018 Power Systems Computation Conference, PSCC, IEEE, 2018, pp. 1–7.
[38] Y. Chen, B. Zhang, Learning to solve network flow problems via neural decoding, 2020, arXiv:2002.04091.