This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

A Convex Neural Network Solver for DCOPF with Generalization Guarantees

Ling Zhang, Yize Chen and Baosen Zhang

***Abstract*—The DC optimal power flow (DCOPF) problem is a fundamental problem in power systems operations and planning. With high penetration of uncertain renewable resources in power systems, DCOPF needs to be solved repeatedly for a large amount of scenarios, which can be computationally challenging. As an alternative to iterative solvers, neural networks are often trained and used to solve DCOPF. These approaches can offer orders of magnitude reduction in computational time, but they cannot guarantee generalization, and small training error does not imply small testing errors. In this work, we propose a novel algorithm for solving DCOPF that guarantees the generalization performance. First, by utilizing the convexity of DCOPF problem, we train an input convex neural network. Second, we construct the training loss based on KKT optimality conditions. By combining these two techniques, the trained model has provable generalization properties, where small training error implies small testing errors. In experiments, our algorithm significantly outperforms other machine learning methods.**

I. Introduction

The optimal power flow (OPF) problem is a fundamental tool used in the planning and operation of power systems [1]–[3]. The OPF problem finds the least cost generator outputs that satisfy the power flow equations and other operational constraints. In this paper we specifically consider the DCOPF formulation of the OPF problem, which linearizes the power flow equations [4].

The DCOPF problem has been studied extensively for almost half a century and is a workhorse of the power industry. If the generator cost is linear, the DCOPF is a linear program (LP). These LPs 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].

Even though solving a single instance of DCOPF problems is easy, computational challenges are arising because of the increase in renewable resources, since they introduce significant uncertainties into generation and load [7], [8]. To account for these uncertainties, operators often need to repeatedly solve the DCOPF problem for a large number of scenarios [9]–[11]. If these scenarios are analyzed close to real-time, then using standard solvers can become too inefficient. For example, suppose a single instance of DCOPF can be solved in 1 second. Then solving a thousand instances would require more than 15 minutes, which would likely be too slow.

Recently, end-to-end neural network architectures have been proposed as surrogates to conventional LP solvers [12]–[18]. These neural networks treat load as the input and output the generation values. Since they only require simple function evaluations, they offer orders of magnitude speedup compared to iterative algorithms. Despite the increase in speed, these machine learning approaches lack provable guarantees on their performances. There are two broad classes of approaches using neural networks for DCOPF. In [12]–[15], the neural network is used to directly approximate the functional mapping from load to the optimal generations. In [16]–[18], neural networks are used to identify the binding constraints and the solutions are recovered by solving linear systems of equations. Both of these approaches rely on training with a large set of labeled data, then showing the performance of the algorithms through simulations.

A fundamental barrier in adopting these methods in practice is the need to show that small *training error* implies small *testing errors*. That is, we need to show that the method *generalizes*. Using machine learning for DCOPF is most useful when operators are faced with unfamiliar conditions, but there are many examples when machine learning precisely fails in these conditions [19]–[21]. The hesitancy in using machine learning (especially deep learning) methods also comes from the perception that they rely on "black-boxes" that are hard to understand [22], [23].

In this paper we propose a two-step approach for solving DCOPF. Firstly, a neural network is used to learn the *value* (i.e., the optimal cost) of the DCOPF, and its gradient with respect to the load is the locational marginal prices (LMPs). Then the binding constraints are identified based on the LMPs. This process is robust in the sense that if they are approximately correct, and the binding constraints are correctly identified. We provide formal guarantee to the generalization capability of this method by directly designing the fundamental features of the DCOPF problem into the machine learning algorithm. Specifically, we constrain the neural network architecture, and building KKT conditions into the training process.

The first technique we use to guarantee generalization is to constrain the neural network to have an input convex structure, since the cost of the DCOPF is a convex function of the loads. Therefore, we use input convex neural networks

L. Zhang and B. Zhang are with the Department of Electrical and Computer Engineering at the University of Washington. Emails: {lzhang18,zhangbao}@uw.edu. Y. Chen is with the Lawrence Berkeley National Labs, email yizechen@lbl.gov.

The authors are partially supported by NSF grants ECCS-1942326 and ECCS-2023531, and the Washington Clean Energy Institute.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

(ICNNs) to learn this relationship [24]–[26]. The ICNNs represent functions that are convex from input to the output by using ReLU as the activation functions and restricting the weights to be nonnegative. Then the convex structure follows from the composition of convex functions [25]. It turns out that the convex structure of the neural network is key to its generalization capability.

We show that a small training error for a finite number of samples implies that the error would be small for entire regions of inputs. Intuitively, this means that the gradient of the function (the LMPs) would be roughly correct as long as some points in the input region are sampled during training. This result is in contrast with standard generalization results in the literature, where most are of a statistical nature [27]. Instead, we show that the structure of the neural network is the key, since if convexity is not imposed, we can construct cases with zero training error but arbitrary large testing errors.

The second technique we use is to add KKT conditions to the training process. Perhaps the most direct way to improve generalization is to increase the number of labeled samples. However, for even moderately large power systems, covering the whole load space with labeled data is intractable due to the curse of dimensionality. We overcome this challenge by using that at optimality, the primal and dual variables satisfy the KKT conditions [28]. Interpreting the dual variables as the partial derivatives of the value function with respect to different parameters, the KKT conditions can be written as a set of partial differential equations. We train the ICNN by minimizing the training loss that is based on this set of partial differential equations. This enables us to include a much larger set of inputs without explicitly adding labeled training data.

In summary, our contributions are:

1) We constrain the neural network to have an input convex structure, which allows the model to generalize well. The guarantees on generalization performance are given in Theorem 6.4 and Theorem 6.5. These theorems prove our method can generalize to testing data points from spaces unseen in the training process. The effectiveness of our method in improving generalization performance is also demonstrated through simulations.
2) We add a term based on violations of KKT conditions to the regression loss, enabling us to use large amounts of unlabeled samples for training and further improving the generalization performance.

The challenge in solving DCOPF repeatedly for different loads is similar in spirit to solving linear systems of equations for changing right-hand-side vectors. In this paper we are trying to solve a parametric optimization problem rather than a parametric linear system [29]. However, unlike LU factorization, it is hard to guarantee that the machine learning methods would always recover the right answer. The approach in [30] can bound the worst case errors for a trained neural network with fixed parameters. But these types of ex post analysis is hard to generalize and do not shed light on why a method may perform better or worse. In this paper, we show how designing the structure of the neural network can lead to more robust guarantees. A similar observation was made in the context of voltage regulation problems in [31].

This paper is organized as follows. Section III provides the DCOPF model and Section IV describes the solution method. Section V describes the neural network design and the training loss based on KKT conditions, and Section VI states and proves the generalization guarantees. Simulations illustrating the results are shown in Section VII.

II. Motivation and Applications

As stated in the introduction, the challenge we address in this paper is the computational difficulties that come from having to repeat solving a large number of DCOPF problems either in real-time operations or in planning. The benefit of using a machine learning enabled method is that we can speed up the solution speed by an order of magnitude or more. We list three possible application domains.

Real-time operations. The increase in renewables introduces significant uncertainties into generation and load [7], [8]. Hence stochastic optimization formulation of OPF has received significant attention in recent years. The works in in [32]–[35] consider generator scheduling for systems with significant renewable penetration. The schedules are determined by solving multiple scenarios of *DCOPF problems*. The main limitations of these approaches are actually the number of scenarios that can be solved in real-time. In [32], for a network of about 100 buses, less than 100 scenarios can be considered because of computational limitations. The work in [33] considered stochastic dispatch in the timescale of hours, where the main bottleneck was number of scenarios that can be solved in an hour. The works in [34] and [35] treated more general stochastic linear programs used in energy systems, both pointing out that the large number of possible scenarios (or parametric LPs) is the main bottleneck in real-time decision making under uncertainty. In all of the above, different types of scenario reduction and approximation methods were used to overcome the computational bottleneck. Our approach can be seen as directly providing a faster solver for DCOPF, which eliminates the need to reduce the number of scenarios.

Planning. Planning problems are not solved in real-time, but they are typically very computationally intensive. The majority of planning problems use DCOPF (or even simpler versions) because of its computational tractability [36]. As uncertainties arise from renewables increase, planning problems are starting to consider a larger number of scenarios as well. When substantial uncertainties are present, both earlier works (e.g. [37]) and more recent ones (e.g., [38]–[40]) are all mainly dealing with the large number of parametric LPs that need to be solved in a reasonable amount of time (say within a day). Another example is the work in [41] considered planning tie-line scheduling subject to security constraints. The main algorithmic difficulty is the repeated computation of a large of number DCOPF problems and a significant amount of effort is on selecting which ones to solve. Therefore, our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

approach can be used as a subroutine to substantially speed up these problems.

Markets. DCOPFs are used [4] to find the locational marginal prices (LMPs) and the multipliers associate with line flows used in financial transmission rights [4]. Here ACOPF are typically not used because of convexity issues (regardless if it can be solved to optimality). During probabilistic load forecasting, the system operator are often interested in finding the LMPs corresponding to each load scenario [29], [42]. The number of scenarios that can be considered is again bounded by the speed of DCOPF solvers. By speeding up the solver, we can potentially avoid situations where the LMPs are prohibitively high. This also points out an advantage of our approach compared to end-to-end solvers. Since end-to-end methods do not yield the dual variables, they are not useful for these types of market operations.

An important future direction is to extend this work to ACOPF problems. For an ACOPF problem, if all the active constraints can be determined, the resulting AC power flow problem is still nonlinear, but in general simpler than the original optimization problem [43]. It would be important to compare against regression-based methods that learns a warm start to the ACOPF problem [14].

III. DCOPF

A. Model

Consider a power system where the n buses are connected by m edges. For each of the bus, we let x_i denote the output of the generator located at the bus, and let l_i denote the load consumed at the bus. Let $\mathbf{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{\ell} = (l_1, \ldots, l_n)$ be the generation and load vectors, respectively. The generation cost vector is denoted as $\mathbf{c} \in \mathbb{R}^n$. We assume that $\mathbf{c}$ is non-negative and has at least one positive component. We assume the system is connected. For notational simplicity, we assume that all buses have generation and load. Without loss of generality, we assume x_i is bounded by 0 and $\bar{x}_i$.[1] If bus i does not have any generation capability, we set $\bar{x}_i = 0$.

The line flows are related to the bus power injections through a linear relationship. Because of Kirchhoff's laws, not all flows in the m lines are independent. In particular, there are only $n-1$ fundamental flows and the rest of $m-n+1$ flows are linear combinations of the fundamental ones [44], [45]. Let $\mathbf{f} \in \mathbb{R}^{n-1}$ denote these $n-1$ fundamental flows. More details on line flow modeling is given in Appendix A.

The DCOPF problem asks for the least cost generations while satisfying all the loads and flow constraints:

$$J^\star(\boldsymbol{\ell}) = \min_{\mathbf{x},\mathbf{f}} \ \mathbf{c}^T\mathbf{x} \tag{1a}$$
$$\text{s.t. } \mathbf{0} \le \mathbf{x} \le \bar{\mathbf{x}} \tag{1b}$$
$$-\bar{\mathbf{f}} \le \mathbf{K}\mathbf{f} \le \bar{\mathbf{f}} \tag{1c}$$
$$\mathbf{x} + \tilde{\mathbf{A}}\mathbf{f} = \boldsymbol{\ell}, \tag{1d}$$

[1]Since the power flow equations are linear, nonzero lower bounds can be shifted to be zero by subtracting a constant from the generation values.

where the matrix $\mathbf{K} \in \mathbb{R}^{m\times(n-1)}$ maps $\mathbf{f}$ to the flows on all edges, and the matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{n\times(n-1)}$ is the modified incidence matrix that maps the fundamental flows to the nodal power injections. The value of the optimization problem is denoted as $J^\star(\boldsymbol{\ell})$ and the optimal solution is denoted as $\mathbf{x}^\star(\boldsymbol{\ell})$.

The DCOPF problem in (1) is an LP and is readily solved by most optimization packages such as CPLEX or Gurobi [5]. These solvers have been optimized to the point that a single LP can be solved in the order of seconds even for large systems. The challenge comes from the fact that repeatedly solving (1) for changing loads can be time-consuming, even if the time it takes for a single instance is small. The formulation in (1) can be easily extended to quadratic costs and is given in Appendix D.

B. Example

we present a small example that is used to illustrate many of the points in this paper. Consider a single-bus system with three generators (with cost \$1/MW, \$2/MW and \$3/MW, respectively) serving a load. The DCOPF problem becomes

$$J^\star(l) = \min\ x_1 + 2x_2 + 3x_3 \tag{2a}$$
$$\text{s.t. } 0 \le x_i \le \bar{x}_i, i = 1,2,3 \tag{2b}$$
$$x_1 + x_2 + x_3 = l. \tag{2c}$$

Figure 1 plots the cost against the load. In an end-to-end approach, the goal is to learn the generations directly. In the next section we will introduce a method that learns the curve $J^\star(l)$ and its derivatives.

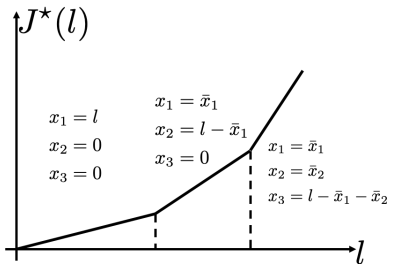


Fig. 1: The cost curve of a single bus load with three generators. The curve is piecewise linear, convex and increasing, with each piece corresponding to a different generation profile.

IV. Learning Active Constraints

The goal of using machine learning is to avoid resolving (1) every time the load changes. A number of algorithms have been proposed to directly learn the functional mapping from $\boldsymbol{\ell}$ to $\mathbf{x}^\star(\boldsymbol{\ell})$ [16]–[18]. However, it is difficult to ensure the learned solutions satisfy the constraints in (1b) to (1d). For the example in Fig. 1, each of the generations have upper and lower bounds as well as the load balance constraint (sum of generation is equal to the load). Instead of using end-to-end neural networks, the mapping $J^\star(\boldsymbol{\ell})$ is learned. Then the associated dual variables are obtained from the global

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

dependence on the right-hand-side vector in the LP. With the value of the dual variables, we are able to find out a set of active constraints for (1). The exact value of $\mathbf{x}^\star(\boldsymbol{\ell})$ is then found by solving a system of linear equations.

A. Global dependence on the right-hand side vector

The global dependence of the optimal cost function $J^\star(\boldsymbol{\ell})$ on the load vector $\boldsymbol{\ell}$ can be found through standard duality theory [28]. The Lagrangian associated with (1) is

$$L(\mathbf{x},\mathbf{f},\underline{\boldsymbol{\tau}},\bar{\boldsymbol{\tau}},\underline{\boldsymbol{\lambda}},\bar{\boldsymbol{\lambda}},\boldsymbol{\mu}) = \mathbf{c}^T\mathbf{x} - \underline{\boldsymbol{\tau}}^T\mathbf{x} + \bar{\boldsymbol{\tau}}^T(\mathbf{x}-\bar{\mathbf{x}}) - \underline{\boldsymbol{\lambda}}^T(\bar{\mathbf{f}}+\mathbf{K}\mathbf{f}) + \bar{\boldsymbol{\lambda}}^T(\mathbf{K}\mathbf{f}-\bar{\mathbf{f}}) + \boldsymbol{\mu}^T(\boldsymbol{\ell}-\mathbf{x}-\tilde{\mathbf{A}}\mathbf{f}), \quad (3)$$

where $\underline{\boldsymbol{\tau}},\bar{\boldsymbol{\tau}} \in \mathbb{R}^n$ are the dual variables associated with generator capacity constraints (1b), $\underline{\boldsymbol{\lambda}},\bar{\boldsymbol{\lambda}} \in \mathbb{R}^m$ are the dual variables associated with flow capacity constraints (1c), and $\boldsymbol{\mu} \in \mathbb{R}^n$ are the dual variables associated with equality constraints (1d). The dual variables $\boldsymbol{\mu}$ are called the locational marginal prices (LMPs) since they represent the marginal cost of supplying one more unit of power at a bus [46].

The dual problem of (1) is

$$\max_{\underline{\boldsymbol{\tau}},\bar{\boldsymbol{\tau}},\underline{\boldsymbol{\lambda}},\bar{\boldsymbol{\lambda}},\boldsymbol{\mu}} \quad \boldsymbol{\mu}^T\boldsymbol{\ell} - \underline{\boldsymbol{\lambda}}^T\bar{\mathbf{f}} - \bar{\boldsymbol{\lambda}}^T\bar{\mathbf{f}} - \bar{\boldsymbol{\tau}}^T\bar{\mathbf{x}} \quad (4)$$

$$\text{s.t.} \quad \mathbf{c} - \underline{\boldsymbol{\tau}} + \bar{\boldsymbol{\tau}} - \boldsymbol{\mu} = \mathbf{0} \quad (4a)$$

$$-\mathbf{K}^T\underline{\boldsymbol{\lambda}} + \mathbf{K}^T\bar{\boldsymbol{\lambda}} - \tilde{\mathbf{A}}^T\boldsymbol{\mu} = \mathbf{0} \quad (4b)$$

$$\underline{\boldsymbol{\tau}},\bar{\boldsymbol{\tau}},\underline{\boldsymbol{\lambda}},\bar{\boldsymbol{\lambda}} \geq \mathbf{0}. \quad (4c)$$

We assume that the load $\boldsymbol{\ell}$ is feasible. From the value of $\boldsymbol{\mu}$, we can learn the active constraints set for (1). To be specific, the optimal solutions for (1) are associated with the following active/inactive constraints through the value of $\boldsymbol{\mu}^\star$:

$$x_i^\star = \begin{cases} 0, & \text{if } \mu_i^\star - c_i < 0 \\ \bar{x}_i, & \text{if } \mu_i^\star - c_i > 0 \\ (0,\bar{x}_i), & \text{otherwise,} \end{cases} \quad (5)$$

and

$$f_i^\star = \begin{cases} \bar{f}_i, & \text{if } \bar{\lambda}_i^\star - \underline{\lambda}_i^\star > 0 \\ -\bar{f}_i, & \text{if } \bar{\lambda}_i - \underline{\lambda}_i^\star < 0 \\ (-\bar{f}_i,\bar{f}_i), & \text{otherwise,} \end{cases} \quad (6)$$

where, given the value of $\boldsymbol{\mu}^\star$, the value of $\bar{\lambda}_i^\star - \underline{\lambda}_i^\star$ can be determined by solving the following optimization problem:

$$\min_{\boldsymbol{\nu}} \ \|\boldsymbol{\nu}\|_1 \quad (7)$$
$$\text{s.t. } \mathbf{K}^T \mathit{diag}(\bar{f}_1,\cdots,\bar{f}_m)\boldsymbol{\nu} = \tilde{\mathbf{A}}^T\boldsymbol{\mu},$$

where $\mathit{diag}(\cdot)$ is a diagonal matrix. The value of $\bar{\lambda}_i^\star - \underline{\lambda}_i^\star$ is related to $\boldsymbol{\nu}$ by $\bar{\lambda}_i^\star - \underline{\lambda}_i^\star = \nu_i/\bar{f}_i$. Due to space constraints, we skip the detailed derivations. They are straightforward and can be found in the online companion in [18].

It turns out the learning problem becomes much simpler from the dual problem. Instead of directly learning the optimal solutions or the active constraints (neither are continuous in the load), it suffices to learn $J^\star$, which is a scalar function that is continuous in $\boldsymbol{\ell}$. The multipliers μ can be then recovered from the following theorem about the global dependence of the optimal cost $J^\star(\boldsymbol{\ell})$ on load $\boldsymbol{\ell}$:

Theorem 4.1. *A vector $\boldsymbol{\mu}^\star$ is an optimal solution to the dual problem (4) if and only if it is a (sub)gradient of the optimal cost $J^\star(\boldsymbol{\ell})$ at the point $\boldsymbol{\ell}$, that is,*

$$\nabla_{\boldsymbol{\ell}} J^\star = \boldsymbol{\mu}^\star. \quad (8)$$

The proof of this theorem is standard and can be found, for example, in [47]. The subgradient part of the statement comes from the fact that $J^\star$ is differentiable for almost all $\boldsymbol{\ell}$, but not everywhere. If this is the case, $\boldsymbol{\mu}^\star$ is customarily taken as the (componentwise) largest vector in the set of subgradients.

B. Solving DCOPF with known marginal prices

Suppose we can learn the cost function $J^\star$. Then $\boldsymbol{\mu}$ (the gradient with respect to the input) can be easily obtained through back propagation. For a nondegenerative LP problem, there would be exactly the same number of constraints as there are variables. Therefore, after using $\boldsymbol{\mu}^\star$ to find the active constraint sets, a linear system of equations is solved to find the optimal solution.

We summarize the algorithm to solving problem (1) after the value of $\boldsymbol{\mu}$ is known in *Algorithm 1*. This algorithm can offer an order of magnitude speedup compared to iterative solvers, since it only requires solving a sparse linear system of equations. This speed up is comparable to end-to-end regression methods which uses a feed forward neural network to obtain the generation solutions [12] (the detailed computation times are given in [18]). Here we concentrate on the feasibility and generalization properties of our proposed approach.

Algorithm 1: Solving DCOPF with given LMPs
Inputs: $\boldsymbol{\mu}$, $\boldsymbol{\ell}$
Parameters: $\tilde{\mathbf{A}}$, $\mathbf{K}$, $\mathbf{c}$, $\bar{\mathbf{f}}$, $\bar{\mathbf{x}}$
1: Given $\boldsymbol{\mu}$, identify active nodal constraints using (5)
2: Given $\boldsymbol{\mu}$, identify active flow constraints using (6)
3: EquationSolver($\mathbf{x}+\tilde{\mathbf{A}}\mathbf{f}=\boldsymbol{\ell}$, active constraints)
Outputs: Optimal solutions $\mathbf{x}^\star$ to (1)

TABLE I: Solving DCOPF for given LMPs.

V. Network architecture and training loss design

The previous algorithm essentially states that if we can learn $\boldsymbol{\mu}^\star$ well, then we can obtain the optimal solution to (1). Note that $\boldsymbol{\mu}^\star$ need not to be learned perfectly. Take the middle segment in Fig. 1 as an example. As long as the learned $J(l)$ has a derivative between 2 and 3 in this segment, we would detect the correct binding constraints. Therefore the key to success is to ensure that the learned $\boldsymbol{\mu}^\star$ *always* have small error.

However, small training error does not guarantee small generalization error. The effectiveness of most machine learning algorithms are demonstrated through simulations, but engineering applications usually require some a priori guarantees, since well-trained models can fail to make reasonable inference on unseen input samples. In this section, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

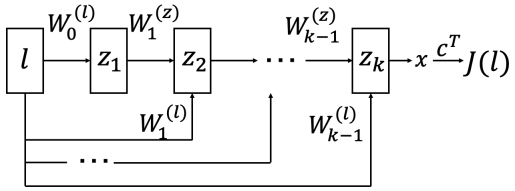


Fig. 2: The architecture of the trained ICNN. The weights $W_1^{(z)}, \ldots, W_{k-1}^{(z)}$ are restricted to be nonegative. The pass through links $W_1^{(\boldsymbol{\ell})}, \ldots, W_{k-1}^{(\boldsymbol{\ell})}$ are not sign restricted.

introduce two design features to guarantee the generalization ability of the trained model. By generalization, we mean a neural network that has a small training error in learning $\boldsymbol{\mu}^\star$ should have small testing errors on new samples. To guarantee generalization, we first utilize the convexity of the cost function to train an Input Convex Neural Network (ICNN). Then we leverage the Karush-Kuhn-Tucker (KKT) optimality conditions to design the training loss.

A. ICNN architecture

A useful result from linear programming that constrain the structure of $J^\star$ is that it is a convex function:

Theorem 5.2. *The optimal cost $J^\star(\boldsymbol{\ell})$ is a convex function of $\boldsymbol{\ell}$ with its domain as the set of all feasible loads. .*

The proof of this theorem is again standard and can be found in [47]. We adopt a special category of deep neural networks (DNNs), called Input Convex Neural Network (ICNN) to leverage this result [24], [25]. The network architecture that we use in this paper is shown in Fig. 2. The basic construction of ICNNs comes from composition of convex functions. Given two functions f and h, if f is convex and h is convex and nondecreasing, $g = h \circ f$ is convex. ICNNs satisfy this property by using ReLU as the activation functions and restricting the weights of the network to be nonnegative. This construction is shown to approximate all Lipschitz convex functions arbitrarily closely [25].

Suppose the fully-connected ICNN has $k+1$ layers, i.e., k hidden layers with "passthrough" and one extra linear output layer. The architecture can be written as follows:

$$\mathbf{z}_{i+1} = \sigma(\mathbf{W}_i^{(z)}\mathbf{z}_i + \mathbf{W}_i^{(\ell)}\boldsymbol{\ell} + \mathbf{b}_i), \text{ for } i = 0, \cdots, k-1 \quad (9)$$

$$\hat{J} = \mathbf{c}^T(\mathbf{z}_k), \quad (10)$$

where $\mathbf{z}_i$ represents the output of the i-th hidden layer, $\mathbf{W}_i^{(z)}$ represents the weight that connects the $i-1$-th hidden layer to the i-th hidden layer, and they are restricted to be nonnegative. The weights $\mathbf{W}_{i-1}^{(\ell)}$ represent the matrices that directly connects the input $\boldsymbol{\ell}$ to the i-th hidden layer. The symbol $\sigma(\cdot)$ represents the ReLU activation function.

We let $\boldsymbol{\theta}$ denote all trainable parameters in (9) and (10), i.e., $\boldsymbol{\theta} = \{\mathbf{W}_{1:k-1}^{(z)}, \mathbf{W}_{0:k-1}^{(\ell)}, \mathbf{b}_{0:k-1}\}$, and the parameterized function $g_{\boldsymbol{\theta}}(\cdot)$ denote the mapping from $\boldsymbol{\ell}$ to $\hat{J}$. Then (9) and (10) can be written in a more compact form as $\hat{J} = g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$. The estimated value of $\boldsymbol{\mu}$ can be obtained by taking the derivative of $g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$ with respect to $\boldsymbol{\ell}$, and is denoted by $\hat{\boldsymbol{\mu}} = \nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$. We show in Section VI that convexity is fundamental to the generalization property of the neural network.

The goal of training the network is to learn the value of $\boldsymbol{\theta}$ which minimizes a specified loss function $\mathcal{L}$, i.e.,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}\big(g_{\boldsymbol{\theta}}(\boldsymbol{\ell}), \nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell})\big). \quad (11)$$

Next, we illustrate how to construct this loss function.

B. Capturing KKT conditions

When the load vector $\boldsymbol{\ell}$ changes by certain amounts, the active constraint set and therefore the value of $\boldsymbol{\mu}^\star$ remain unchanged. In fact, we can divide the feasible input space of $\boldsymbol{\ell}$ into a finite number of regions [2]. Each of these region is a convex polytope and corresponds to a different combination of active constraints and a value of $\boldsymbol{\mu}^\star$.

If we have a number of ground-truth values of $(J^\star, \boldsymbol{\mu}^\star)$ for every possible region, then we can train the neural network by minimizing the regression loss. However, the number of possible regions grows quickly with respect to size of the system. For moderately sized system, there maybe a large number of regions and it is unlikely that historical observations would include data in every region. Even if data is generated offline, exhaustively covering all of the regions with labeled data become cumbersome.

If the data set only samples a small number of regions, the model trained by minimizing the regression loss performs poorly for input samples that come from unseen regions. This is not unexpected since there is no data to make prediction for these unseen regions. But in practice the value of using machine learning is to quickly determine what might happen for a large number of loads where some would come from unseen regions. Interestingly, because we are solving a well-defined optimization problem, we can mitigate this data challenge again by looking at duality theory.

We develop an augmented training approach for the neural network $g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$ based on violations of KKT conditions. Recall KKT conditions for the LP in (1) is

$$\mathbf{c} - \underline{\boldsymbol{\tau}} + \bar{\boldsymbol{\tau}} - \boldsymbol{\mu} = \mathbf{0} \quad (12a)$$

$$-\mathbf{K}^T\underline{\boldsymbol{\lambda}} + \mathbf{K}^T\bar{\boldsymbol{\lambda}} - \tilde{\mathbf{A}}^T\boldsymbol{\mu} = \mathbf{0} \quad (12b)$$

$$\mathbf{0} \le \mathbf{x} \le \bar{\mathbf{x}} \quad (12c)$$

$$-\bar{\mathbf{f}} \le \mathbf{K}\mathbf{f} \le \bar{\mathbf{f}} \quad (12d)$$

$$\mathbf{x} + \tilde{\mathbf{A}}\mathbf{f} = \boldsymbol{\ell} \quad (12e)$$

$$\underline{\boldsymbol{\tau}}, \bar{\boldsymbol{\tau}}, \underline{\boldsymbol{\lambda}}, \bar{\boldsymbol{\lambda}} \ge \mathbf{0} \quad (12f)$$

$$\underline{\tau}_i x_i = 0,\ \bar{\tau}_i(x_i - \bar{x}_i) = 0,\ \forall i \in \{1, \cdots, n\} \quad (12g)$$

$$\underline{\lambda}_j(\bar{f}_j + K_j\mathbf{f}) = 0 \quad (12h)$$

$$\bar{\lambda}_j(K_j^T\mathbf{f} - \bar{f}_j) = 0,\ \forall j \in \{1, \cdots, m\}, \quad (12i)$$

[2]Note that the division of the feasible input space is just for the analysis and illustrative purpose in the paper. We do not do the division in the practice and the implementation of our algorithm does not require this division either.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

where K_j is the $j-$th column of matrix $\mathbf{K}^T$.

Before introducing the loss term related to violations of KKT conditions, we have the following lemma:

Theorem 5.3. *The dual variables $\underline{\boldsymbol{\tau}}, \bar{\boldsymbol{\tau}}, \underline{\boldsymbol{\lambda}}, \bar{\boldsymbol{\lambda}}, \boldsymbol{\mu}$ satisfy the KKT conditions in* (12) *if and only if they satisfy equations* (13)*:*

$$[\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ - \bar{\boldsymbol{\lambda}} = \mathbf{0} \tag{13a}$$
$$[\underline{\boldsymbol{\lambda}} - (\mathbf{K}\mathbf{f} + \bar{\mathbf{f}})]^+ - \underline{\boldsymbol{\lambda}} = \mathbf{0} \tag{13b}$$
$$[\bar{\boldsymbol{\tau}} + (\mathbf{x} - \bar{\mathbf{x}})]^+ - \bar{\boldsymbol{\tau}} = \mathbf{0} \tag{13c}$$
$$[\underline{\boldsymbol{\tau}} - \mathbf{x}]^+ - \underline{\boldsymbol{\tau}} = \mathbf{0}. \tag{13d}$$

Proof. Here we consider (13a) and the rest follow in similar fashions. In particular, we prove the following two conditions are equivalent:

1) $\mathbf{K}\mathbf{f} \le \bar{\mathbf{f}}, \bar{\boldsymbol{\lambda}} \ge 0, (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}}) \odot \bar{\boldsymbol{\lambda}} = 0$
2) $[\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ - \bar{\boldsymbol{\lambda}} = \mathbf{0}$

where $\odot$ represents element-wise multiplication.

First we show that 1) implies 2). If $\mathbf{K}\mathbf{f} < \bar{\mathbf{f}}$, then we must have $\bar{\boldsymbol{\lambda}} = 0$ from the complementary slackness condition $(\mathbf{K}\mathbf{f} - \bar{\mathbf{f}}) \odot \bar{\boldsymbol{\lambda}} = 0$. So

$$[\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ - \bar{\boldsymbol{\lambda}} = [\mathbf{K}\mathbf{f} - \bar{\mathbf{f}}]^+ = 0.$$

If $\mathbf{K}\mathbf{f} = \bar{\mathbf{f}}$ and $\bar{\boldsymbol{\lambda}} \ge 0$, then we have

$$[\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ - \bar{\boldsymbol{\lambda}} = [\bar{\boldsymbol{\lambda}}]^+ - \bar{\boldsymbol{\lambda}} = 0.$$

Next we show 2) implies 1). The right-hand-side implies the following dual feasibility since $[\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ - \bar{\boldsymbol{\lambda}} = \mathbf{0}$ gives

$$\bar{\boldsymbol{\lambda}} = [\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ \ge \mathbf{0}.$$

Suppose the primal feasibility does not hold, i.e., $\mathbf{K}\mathbf{f} - \bar{\mathbf{f}} \ge 0$, then

$$[\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ - \bar{\boldsymbol{\lambda}} \tag{14}$$
$$= \bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}}) - \bar{\boldsymbol{\lambda}} \tag{15}$$
$$= \mathbf{K}\mathbf{f} - \bar{\mathbf{f}} \tag{16}$$
$$= 0. \tag{17}$$

Therefore, we at most have $\mathbf{K}\mathbf{f} = \bar{\mathbf{f}}$ and $\mathbf{K}\mathbf{f}$ cannot exceeds $\bar{\mathbf{f}}$.

For complementary slackness, suppose $\mathbf{K}\mathbf{f} < \bar{\mathbf{f}}$ but $\bar{\boldsymbol{\lambda}} \ne 0$, then we have

$$[\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ - \bar{\boldsymbol{\lambda}} \tag{18}$$
$$= \begin{cases} \text{either } \bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}}) - \bar{\boldsymbol{\lambda}} = \mathbf{K}\mathbf{f} - \bar{\mathbf{f}} = 0 \\ \text{or } 0 - \bar{\boldsymbol{\lambda}} = 0, \end{cases} \tag{19}$$

which contradicts our assumption. So, if $\mathbf{K}\mathbf{f} < \bar{\mathbf{f}}$, we must have $\bar{\boldsymbol{\lambda}} = 0$. Suppose $\bar{\boldsymbol{\lambda}} > 0$ but $\mathbf{K}\mathbf{f} < \bar{\mathbf{f}}$, we arrive a similar contradiction. ■

Note that dual variables $\underline{\boldsymbol{\tau}}, \bar{\boldsymbol{\tau}}, \underline{\boldsymbol{\lambda}}$, and $\bar{\boldsymbol{\lambda}}$ can all be represented in terms of $\boldsymbol{\mu}$. To be specific, based on (12a) and (12f), we can express the dual solutions $\underline{\boldsymbol{\tau}}$ and $\bar{\boldsymbol{\tau}}$ as follows

$$\underline{\boldsymbol{\tau}}(\boldsymbol{\mu}) = [\boldsymbol{\mu} - \mathbf{c}]^+ - (\boldsymbol{\mu} - \mathbf{c}) \tag{20a}$$
$$\bar{\boldsymbol{\tau}}(\boldsymbol{\mu}) = [\boldsymbol{\mu} - \mathbf{c}]^+, \tag{20b}$$

where the notation $[a]^+$ represents $[a]^+ = \max\{a, 0\}$. The dual solutions $\underline{\boldsymbol{\lambda}}$ and $\bar{\boldsymbol{\lambda}}$ can be expressed in terms of $\boldsymbol{\mu}$ by solving the $\ell 1$-minimization problem in (7), and we denote them by $\underline{\boldsymbol{\lambda}}(\boldsymbol{\mu})$ and $\bar{\boldsymbol{\lambda}}(\boldsymbol{\mu})$, respectively.

Plugging the expressions $\underline{\boldsymbol{\tau}}(\boldsymbol{\mu})$, $\bar{\boldsymbol{\tau}}(\boldsymbol{\mu})$, $\underline{\boldsymbol{\lambda}}(\boldsymbol{\mu})$ and $\bar{\boldsymbol{\lambda}}(\boldsymbol{\mu})$ into (13a)-(13d), we obtain a system of equations only related to $\boldsymbol{\mu}$. To capture the KKT optimality conditions in (12), we define *violation degrees* associated with equations (13) as follows

$$\nu_{\bar{\lambda}} = [\bar{\boldsymbol{\lambda}} + (\mathbf{K}\mathbf{f} - \bar{\mathbf{f}})]^+ - \bar{\boldsymbol{\lambda}} \tag{21a}$$
$$\nu_{\underline{\lambda}} = [\underline{\boldsymbol{\lambda}} - (\mathbf{K}\mathbf{f} + \bar{\mathbf{f}})]^+ - \underline{\boldsymbol{\lambda}} \tag{21b}$$
$$\nu_{\bar{\tau}} = [\bar{\boldsymbol{\tau}} + (\mathbf{x} - \bar{\mathbf{x}})]^+ - \bar{\boldsymbol{\tau}} \tag{21c}$$
$$\nu_{\underline{\tau}} = [\underline{\boldsymbol{\tau}} - \mathbf{x}]^+ - \underline{\boldsymbol{\tau}} \tag{21d}$$
$$\nu_p = \boldsymbol{\ell} - \mathbf{x} - \tilde{\mathbf{A}}\mathbf{f}. \tag{21e}$$

Based on *violation degrees* given in (13a)-(13d), we define the loss term to minimize violations of KKT conditions as follows

$$\mathcal{L}_{\mathcal{K}}(\theta) = \|\nu_{\bar{\lambda}}(\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}))\|_2^2 + \|\nu_{\bar{\lambda}}(\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}))\|_2^2 + \|\nu_{\bar{\tau}}(\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}))\|_2^2 + \|\nu_{\underline{\tau}}(\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}))\|_2^2 + \|\nu_p\|_2^2. \tag{22}$$

We also define the regression loss as follows

$$\mathcal{L}_{\mathcal{R}} = \|g_{\boldsymbol{\theta}}(\boldsymbol{\ell}) - J^\star\| + \|\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}) - \boldsymbol{\mu}^\star\|, \tag{23}$$

where the first term is the regression loss defined between $g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$ and $J^\star$ over the neural network's outputs, and the second term is the regression loss between $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$ and $\boldsymbol{\mu}^\star$ over calculated derivatives.

By combining the the regression loss defined in (23) and the KKT-related loss term (22), we can express the training loss for our proposed algorithm as follows

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathcal{R}}(\theta) + \mathcal{L}_{\mathcal{K}}(\theta). \tag{24}$$

By minimizing the loss function in (24), we can get the trained model $g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$ and use it to make predictions in an on-line way.

Note that the KKT-related loss term in (22) can be calculated for both labelled and unlabelled datasets. For the unlabelled dataset, denoted as $\mathcal{D}_{w/t}$, the data points in it only contain the input load, i.e., $\{\boldsymbol{\ell}^{(1)}, \boldsymbol{\ell}^{(2)}, \cdots, \boldsymbol{\ell}^{(N)}\}$. To calculate the KKT-related loss on $\mathcal{D}_{w/t}$ does not require any ground-truth labels. Since dual variables $\underline{\boldsymbol{\tau}}, \bar{\boldsymbol{\tau}}, \underline{\boldsymbol{\lambda}}$, and $\bar{\boldsymbol{\lambda}}$ can all be represented in terms of $\boldsymbol{\mu}$ by solving the $\ell 1$-minimization problem in (7) and using the expressions in (20), they can be obtained by learning $\boldsymbol{\mu}$ from the input load. Besides, as shown in the architecture of the ICNN in Fig. 2, we can interpret the layer before the output layer as the prediction of the power generation $\mathbf{x}$, from which we can get the value of the flow $\mathbf{f}$ using the equation (12e). Then we do not have to consider the violation degree in (21e). For calculating the rest of violation degrees in (21), all the information required can be obtained

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

by learning from the load. Therefore, to construct the KKT-related loss on $\mathcal{D}_{w/t}$, we do not ask for ground-truth labels.

We summarize our proposed algorithm for training the ICNN in Table II, and call it *Algorithm 2*. Although adding KKT-related loss term makes the loss function complex, we do not observe any numerical issues during the training process. In the simulations, the loss function always decreases and converges to a low level.

Algorithm 2: Algorithm for Training the ICNN
Inputs: Samples with labels $\mathcal{D}_w$
Inputs: Samples without labels $\mathcal{D}_{w/t}$
Inputs: Model to be trained $g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$
Parameters: $\tilde{\mathbf{A}}$, $\mathbf{K}$, $\bar{\mathbf{f}}$, $\bar{\mathbf{x}}$, $\underline{\mathbf{x}}$
1: Calculate regression loss term $\mathcal{L}_R$ on $\mathcal{D}_w$ using (23)
2: Calculate KKT conditions-related loss term $\mathcal{L}_K$ on $\mathcal{D}_w$ and $\mathcal{D}_{w/t}$ using (22)
3: Minimize $\mathcal{L}(\theta)$ in (24) to get the optimal parameter $\hat{\boldsymbol{\theta}}$
Outputs: Trained model $g_{\hat{\boldsymbol{\theta}}}(\boldsymbol{\ell})$

TABLE II: Algorithm for training the ICNN.

VI. Generalization

In this section we consider the generalization performance of our proposed method. By generalization, we mean the algorithm should perform well on test samples that were not seen during the training process. We adopt the standard method of analysis here: we assume that the neural network can be trained to zero error on the training samples, then we study the errors for testing samples [48]–[50]. We use the following definition as a shorthand:

Definition 1. *We say a neural network is well-trained if it achieves zero loss on the training data.*

Understanding the generalization properties is important because zero training error does not imply small test error. Consider the example given in Fig. 3. Suppose we are fitting a piece-wise linear function but only given two points that lie on a line and the training loss is the regression loss. There are infinitely many functions that pass through these points, implying that they have zero training error. Obviously, many of them can have large testing errors for other points on the line. This example also shows that it is not sufficient to just impose convexity or provide gradient information on their own. There are also infinitely many convex functions passing through the labeled points, and there are infinitely many functions with the right gradients at the given points. To constrain the class of functions to be learned, both convexity and the gradient information are needed.

Since $J^\star$ is piece-wise linear, we study generalization for two settings. The first is we assume that there are multiple training data within a region where $J^\star$ is linear, and we are interested in the testing performance of a new input from the same region. This setting is about whether the model is constrained enough to not overfit during training. We provide a positive (and simple) answer in Theorem 6.4.

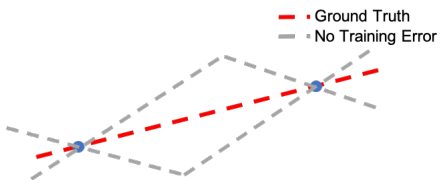


Fig. 3: Given two points on a line, there are infinitely many piecewise linear functions passing through them. Therefore, small training error (passing through the points) does not necessary imply small generalization error (recovering the line).

The second setting is to assume that the test data lies in a region that was unseen during the training process. This question is normally not asked since there is no expectation that learning would be useful for these type of unseen data. However, since there are large number of possible LP regions for DCOPF, it is likely that not all regions would be included in the training data. Therefore, learning algorithms must provide some guarantees on unseen regions. This is especially important as operating conditions change and the historical data are no longer reflective of future scenarios. In Theorem 6.5, we provide a positive answer showing that the gradients of the unseen region are still bounded.

A. Generalization for A Linear Region

In this part, we study the case where all training samples have the same value for $\boldsymbol{\mu}$. That is, they come from the same LP region. Let us denote the set of training samples as $\mathcal{D}_{trn}$, and $\operatorname{convhull}\mathcal{D}_{trn}$ is the convex hull of set $\mathcal{D}_{trn}$. The following theorem states the performance of the neural network for a new input $\boldsymbol{\ell}^{\text{new}}$ that is not in $\mathcal{D}_{trn}$.

Theorem 6.4. *Given N input loads $\mathcal{D}_{trn} = \{\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^N\}$ and assume $\nabla_{\boldsymbol{\ell}} g_{\hat{\boldsymbol{\theta}}}(\boldsymbol{\ell}^i) = \boldsymbol{\mu}$ for all $i = 1, \ldots, N$. Assume the ICNN model $g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$ is well-trained on $\mathcal{D}_{trn}$. Then for all points $\boldsymbol{\ell}^{new} \in \operatorname{convhull}\mathcal{D}_{trn}$, $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new}) = \boldsymbol{\mu}$.*

This theorem is useful for LP problems because the gradient of J is piecewise constant over convex polytopic regions. If we are given some points from a region, then a well-trained neural network guarantees that the function is learned correctly for all points within their convex hull. This result implies the correctness of the overall algorithms since they only rely on the gradient (dual variable) information.

Technically, the above theorem says that if the gradients of a convex function are equal at a set of points, then the function is linear on the convex hull of these points. This is not a surprising result, but it does show that by constraining the structure of the neural network and using gradient information, we can generalize to uncountable number of points (compact regions) by learning from a finite number of training points. The proof of the theorem is given in Appendix B.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

B. Generalization with KKT Loss

If training samples may not represent all possible regions, we can construct the KKT conditions augmented loss in (22) on a set of unlabeled data points, for which we only have access to the input load values but not the optimal cost and the optimal LMPs. Since we do not ask for labels, we can sample as many data points as we want. This allows us to train with a very large set. We call this set of unlabeled data points the helper set. To construct the KKT loss on the helper set, we calculate the violations of KKT conditions through equations (21), where the Lagrange multipliers are determined from the learned $\boldsymbol{\mu}$ by following steps (5) and (6) in the training process.

By training with the helper set, unseen regions become "seen" in the sense that the outputs from the trained model must satisfy KKT conditions. As long as the model is well-trained, the analysis of generalization is the same as Section VI-A. Therefore, the generalization performance of training with helper set can also be guaranteed by Theorem 6.4.

C. Generalization for Unseen Regions

Here we consider the case where a region is not represented at all by the training data, including both labeled and helper sets. Then if a test sample comes from this region, would our method output anything useful? Methods like classification and dictionary learning cannot make useful predictions since there is no basis to make inferences about unseen regions. The next theorem shows that our approach is still partially successful because the gradient of a test data point is bounded by the gradient of the training points:

Theorem 6.5. *Given N input loads $\mathcal{D}_{trn} = \{\boldsymbol{\ell}^1, \dots, \boldsymbol{\ell}^N\}$, assume the ICNN model $g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$ is well-trained on $\mathcal{D}_{trn}$. Assume that $N \geq n+1$ and $\mathcal{D}_{trn}$ does not lie in a lower dimensional subspace in $\mathbb{R}^n$. Then for all points $\boldsymbol{\ell}^{new} \in \operatorname{convhull} \mathcal{D}_{trn}$, $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})$ is contained in a bounded convex polytope.*

The exact characterization of the polytope is given in the proof in Appendix C and depends on the values of $\{\boldsymbol{\ell}^1, \dots, \boldsymbol{\ell}^N\}$ and $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})$. The significance of the theorem lies in that training data are able to constrain the gradient for all points that lies in its convex hull. Intuitively speaking, as long as some surrounding points are included in training, the gradient cannot be "very wrong" even for points coming from LP regions that were not seen during training.

Consider the curve in Fig 1. Suppose we learned the two end pieces correctly but there was no training data for the middle piece, as shown in Fig. 4. Then by Theorem 6.5, the slope of the middle piece is constrained to be between the slope of the two end pieces. Furthermore, even if the neural network is trained in such a way that there are more than one piece of the middle region, the slopes of all of the pieces are still bounded between the two end pieces. Since Algorithm 1 only relies on getting $\boldsymbol{\mu}$ to be in the correct range, the active constraints would be identified correctly for all of these cases.

Theorem 6.5 formalizes the picture in Fig. 4 to higher dimensions, but the geometric intuition remains the same. This theorem also formalizes the empirical observation in [30], where the error of neural network-based OPF is reduced if training points are on the boundary of the feasible region.

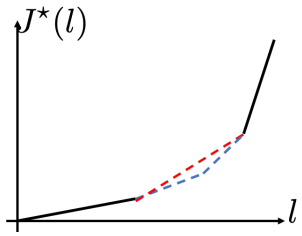


Fig. 4: Example when the middle region has no data. But as long as the other two regions are well trained (black lines), the slopes in the middle region are bounded (blue and red dashed lines) by Theorem 6.5. Then the active constraint detection (Algorithm 1) would still be correct.

VII. Experimental results

In this section, we demonstrate experimental results of using Algorithm 2 for training the ICNN and Algorithm 1 for solving DCOPF. We use the IEEE 14-bus system as the benchmark. We first examine the quality of solutions in terms of feasibility and optimality, then we examine the generalization performances. In Section VII-B, we show the simulation results of our model in the IEEE 118-bus system.

A. 14-bus System

To generate the training set, we first sample $\boldsymbol{\ell}$ from the uniform distribution. We use two different variations from nominal load values: 30% and 50%. The total size of data samples is 50000 for each setting. Then, for each value of $\boldsymbol{\ell}$, we solve the primal and dual problems using CVXPY [51] powered by CVXOPT [52]. The optimal cost and the dual solutions are recorded. We hold 20% of all data samples as the test set and use the remaining for training. The ICNN we train has 4 hidden layers.

In order to better evaluate the performance of the model trained using our algorithm, we also provide experimental results of the end-to-end learning model, which is by far the most popular in the field of using deep learning for solving DCOPF [12]–[15]. We constructed the end-to-end model with an architecture similar to [13], although other architectures do not change the conclusions. To be specific, for the end-to-end model, we train a 4-layers fully-connected ReLU network by minimizing the regression loss, and use the trained model to directly predict the optimal solution to (1).

Both the end-to-end model and our model using Algorithm 2 are implemented on the Tensorflow platform and are trained until the training loss converges. The end-to-end model can be implemented using existing modules in Tensorflow, however, calculating KKT-related loss is specific for solving DCOPF and cannot be implemented using existing modules in Tensorflow. So we need to do some customization. In our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

Input variations	30%			50%		
Solutions	Optimality	Feasibility	Infeasibility	Optimality	Feasibility	Infeasibility
End-to-End	18.34	21.98	78.02	17.76	46.30	53.70
Our model	**94.93**	**94.93**	**5.07**	**93.08**	**93.08**	**6.92**

TABLE III: Quality of solutions. We compare the solutions quality of our model to the end-to-end model. The ratios of optimal, feasible and infeasible solutions are listed and the numbers represent percentages. Results under two different input variations are given, i.e., 30% and 50% deviations from the nomial load value. More than 90% of the solutions obtained from our algorithm are optimal, while less than half of the solutions from the end-to-end model are feasible.

experiments, the customized modules take longer time to train than existing functions or modules in Tensorflow. Therefore, the training time of our model for each epoch is longer than the end-to-end model.

1) Overall performance: To evaluate the quality of solutions obtained by different learning models, we divide solutions into three categories: optimal, feasible, and infeasible solutions. In our model, we feed $\boldsymbol{\ell}$ into the neural network and find the active constraints set. Then we solve a system of linear equations using a standard solver to obtain final solutions. The end-to-end model can directly outputs the solution to $\mathbf{x}$. The ratios of optimal, feasible and infeasible solutions obained by different learning models are listed in Table III. As shown in Table III, the optimality ratio of the solutions obtained from our model is higher than 90% under both input variations. In comparison, almost 50% of the solutions obtained from the end-to-end model are infeasible. In terms of computational time, both methods are much faster than iterative solvers in online DCOPF solving.

To examine the solution feasibility in the end-to-end model, we use the output $\mathbf{x}$ and the nodal power balance (1d) to obtain $\mathbf{f}$. The value of feasibility ratio depends on how large the error tolerance is. In this paper, 0.3% mismatch is allowed when we calculate feasibility ratios. In Table IV, we also list the ratios of solutions that do not satisfy the nodal power balance, the limits on generators' outputs, and the limits on line flows, respectively. From Table IV, we can see that more than 98% of the solutions obtained from our model satisfy both generators limits and line flows limits. By contrast, only 24% of the solutions from the end-to-end model satisfy the line flow limits.

2) Generalization on Unseen Regions: To evaluate the generalization ability of our model, we create an illustrative example in the 14-bus system. Particularly, we examine the generalization performance on new data points that comes from region without any training samples. To generate the training set for this case, we only change the load values at two buses, but keep the remaining load values fixed. In this way, the space of input loads can be regarded as a two-dimensional plane. When varying the load values at the two buses, we can have four different combinations of active constraints, which correspond to four different values of $\boldsymbol{\mu}^\star$. Therefore, we divide the input load space as four regions, denoted as R_0, R_1, R_2 and R_3. The division of the input space is shown in Fig. 5. We take training samples from R_0, R_2, and R_3, and take testing samples from R_1. Let us denote the training set as $\mathcal{D}_{trn}$, and the testing set as $\mathcal{D}_{tst}$.

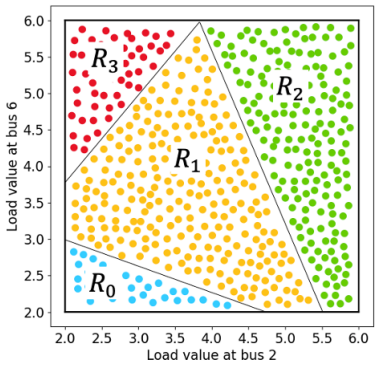


Fig. 5: Division of the input space. The axes are load values at the two buses. In this example, based on different combinations of active constraints, the input load space can be divided into four regions. We take samples from R_1 as the testing set and samples from surrounding regions R_0, R_2 and R_3 as the training set.

We use two different training approaches for our model. In the first training approach, we only use $\mathcal{D}_{trn}$ for training and minimize both the regression loss and the KKT-related loss on $\mathcal{D}_{trn}$. For the second training approach, we construct an additional training set, called helper set $\mathcal{D}_{help}$, which only contains the input load, i.e., $\{\boldsymbol{\ell}^{(1)}, \boldsymbol{\ell}^{(2)}, \cdots, \boldsymbol{\ell}^{(N)}\}$, and do not have any ground-truth values. To generate the helper set, we can sample $\boldsymbol{\ell}$ from uniform distributions $\ell_2 \sim \mathsf{Uniform}(2, 6)$ and $\ell_6 \sim \mathsf{Uniform}(2, 6)$. Therefore, $\mathcal{D}_{help}$ contains the testing region R_1. Aside from minimizing (24) on $\mathcal{D}_{trn}$, we also minimize the KKT-augmented loss (22) on $\mathcal{D}_{help}$. [3] End-to-end model only use labeled samples and is trained on $\mathcal{D}_{trn}$.

We list the ratios of optimal, feasible and infeasible solutions obtained from different learning models in Table V. As we can see, when we use $\mathcal{D}_{help}$ as an additional training set, we can obtain an optimality ratio as high as 96%. Even without $\mathcal{D}_{help}$, more than half of the solutions obtained from our model can achieve optimal values. As a comparison, the end-to-end model fails to make feasible predictions on test samples that come from never seen regions in the training process. The reason that our proposed algorithm outperforms the end-to-end model can be attributed to the KKT-related loss. By minimizing the KKT-related loss term, the trained model is able to learn the underlying KKT conditions in all four regions and make better predictions of $\boldsymbol{\mu}^\star$ on $\mathcal{D}_{tst}$.

[3] We know from Section V-B that to calculate the KKT-augmented loss on $\mathcal{D}_{help}$ does not require ground-truth labels.

Input variations	30%			50%		
Infeasibility	Nodal balance	Generators limits	Lines limits	Nodal balance	Generators limits	Lines limits
End-to-End	16.88	0	76.79	47.45	6.33	6.46
Our model	**5.07**	**0**	**1.18**	**6.92**	**0**	**0.79**

TABLE IV: Infeasibility of solutions. We compare our model to the end-to-end model. The ratios of solutions that do not satisfy the nodal power balance, generators' limits and the line limits are listed. 0.3% mismatch is allowed. We give results under two different input variations, i.e., 30% and 50%. In both settings, more than 98% of the solutions obtained from our model satisfy both generators limits and line flows limits, and more than 90% of our solutions satisfy the nodal power balance.

	Optimality	Feasibility
End-to-end	5.52	8.6
Our model, with $\mathcal{D}_{\mathbf{help}}$	**97.24**	**97.24**
Our model, without $\mathcal{D}_{\mathbf{help}}$	**62.25**	**72.31**

TABLE V: Generalization performance on test samples coming from never seen regions. With the helper set, our method is optimal 97% of the time (62% without). The end-to-end model fails to make reasonable predictions.

B. 118-bus System

In this part, we evaluate our model in the IEEE 118-bus system, where 54 generators and 183 edges are located, and compare with the classification approach. To generate the training set, we sample load values $\boldsymbol{\ell}$ from a Gaussian distribution with mean of 1.0 and standard deviation of 0.03. Then, for each value of $\boldsymbol{\ell}$, we solve the primal and dual problems using CVXPY powered by CVXOPT. We solve 60000 samples and split 20% of all data samples for testing and 80% for training. For our model, we train a 5-layer ICNN to learn the mapping from load values to cost function values. For the classification method, a 5-layer fully-connected neural network is constructed to predict the set of active constraints. We use one-hot encoding of different active sets as labels, and use cross-entropy as the loss function. The simulation results show that our model achieves an accuracy of 88.96% in terms of finding optimal solutions. In comparison, the accuracy for the classification task is 39.61%.

VIII. Conclusions

This paper proposes a new framework to use neural networks for solving DCOPF. By leveraging rich linear programming theories, we prove our framework guarantees generalization. First, using the convexity of optimal cost in DCOPF, we constrain the neural network to have an input convex structure. Second, using the KKT optimality conditions, we add violations of KKT conditions to the training loss. In this way, we are able to exploit large amounts of unlabeled data points for training and improve the generalization performance. Our method is evaluated on the IEEE 14-bus and 118-bus. The experimental results demonstrate that our method significantly outperforms existing end-to-end and classification approaches.

References

[1] H. W. Dommel and W. F. Tinney, "Optimal power flow solutions," *IEEE Transactions on power apparatus and systems*, no. 10, pp. 1866–1876, 1968.

[2] R. Baldick, *Applied optimization: formulation and algorithms for engineering systems*. Cambridge University Press, 2006.

[3] J. D. Glover, T. J. Overbye, and M. S. Sarma, *Power System Analysis and Design*. CENGAGE Learning, 2017.

[4] B. Stott, J. Jardim, and O. Alsaç, "Dc power flow revisited," *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1290–1300, 2009.

[5] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "Matpower: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on power systems*, vol. 26, no. 1, pp. 12–19, 2010.

[6] F. Milano, "An open source power system analysis toolbox," *IEEE Transactions on Power systems*, vol. 20, no. 3, pp. 1199–1206, 2005.

[7] Y. Tang, K. Dvijotham, and S. Low, "Real-time optimal power flow," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2963–2973, 2017.

[8] E. Mohagheghi, M. Alramlawi, A. Gabash, and P. Li, "A survey of real-time optimal power flow," *Energies*, vol. 11, no. 11, p. 3142, 2018.

[9] A. Hauswirth, S. Bolognani, G. Hug, and 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.

[10] Y. Zhang, E. Dall'Anese, and 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.

[11] S. Huang and V. Dinavahi, "Fast batched solution for real-time optimal power flow with penetration of renewable energy," *IEEE Access*, vol. 6, pp. 13 898–13 910, 2018.

[12] X. Pan, T. Zhao, and 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] ——, "Deepopf: A deep neural network approach for security-constrained dc optimal power flow," *arXiv preprint arXiv:1910.14448*, 2019.

[14] A. Zamzam and K. Baker, "Learning optimal solutions for extremely fast ac optimal power flow," *arXiv preprint arXiv:1910.01213*, 2019.

[15] R. Nellikkath and S. Chatzivasileiadis, "Physics-informed neural networks for minimising worst-case violations in dc optimal power flow," *arXiv preprint arXiv:2107.00465*, 2021.

[16] D. Deka and S. Misra, "Learning for dc-opf: Classifying active sets using neural nets," in *2019 IEEE Milan PowerTech*. IEEE, 2019, pp. 1–6.

[17] L. A. Roald and D. K. Molzahn, "Implied constraint satisfaction in power system optimization: The impacts of load variations," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019, pp. 308–315.

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

[19] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine learning*, vol. 15, no. 2, pp. 201–221, 1994.

[20] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 2016, pp. 372–387.

[21] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," 2018.

[22] D. E. Womble, "Machine learning: Issues and opportunities," Oak Ridge National Lab, Tech. Rep. 112595, 2018.

[23] J. L. Cremer, I. Konstantelos, and G. Strbac, "From optimization-based machine learning to interpretable security rules for operation," *IEEE Transactions on Power Systems*, vol. 34, no. 5, pp. 3826–3836, 2019.

[24] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," in *International Conference on Machine Learning*, 2017.

[25] Y. Chen, Y. Shi, and B. Zhang, "Optimal control via neural networks: A convex approach," in *International Conference on Learning Representations, ICLR*, 2019.

[26] ——, "Data-driven optimal voltage regulation using input convex neural networks," *Electric Power Systems Research*, vol. 189, 2020.

[27] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv:1609.04836*, 2016.

[28] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[29] Y. Ji, "Operation under uncertainty in electric grid: A multiparametric programming approach," Ph.D. dissertation, Cornell University, 2017.

[30] A. Venzke, G. Qu, S. Low, and S. Chatzivasileiadis, "Learning optimal power flow: Worst-case guarantees for neural networks," *arXiv preprint arXiv:2006.11029*, 2020.

[31] 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 *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2020, pp. 1–6.

[32] H. Gangammanavar, S. Sen, and V. M. Zavala, "Stochastic optimization of sub-hourly economic dispatch with wind energy," *IEEE Transactions on Power Systems*, vol. 31, no. 2, pp. 949–959, 2015.

[33] Y. Gu and L. Xie, "Stochastic look-ahead economic dispatch with variable generation resources," *IEEE Transactions on Power Systems*, vol. 32, no. 1, pp. 17–29, 2016.

[34] S. Sen and Y. Liu, "Mitigating uncertainty via compromise decisions in two-stage stochastic linear programming: Variance reduction," *Operations Research*, vol. 64, no. 6, pp. 1422–1437, 2016.

[35] A. Alqurashi, A. H. Etemadi, and A. Khodaei, "Treatment of uncertainty for next generation power systems: State-of-the-art in stochastic optimization," *Electric Power Systems Research*, vol. 141, pp. 233–245, 2016.

[36] H. Seifi and M. Sepasian, *Electric Power System Planning: Issues, Algorithms and Solutions*, ser. Power Systems. Springer Berlin Heidelberg, 2011. [Online]. Available: https://books.google.com/books?id=PW39l0GgPy8C

[37] B. G. Gorenstin, N. M. Campodonico, J. P. Costa, and M. V. F. Pereira, "Power system expansion planning under uncertainty," *IEEE Transactions on Power Systems*, vol. 8, no. 1, pp. 129–136, 1993.

[38] J. Li, F. Lan, and H. Wei, "A scenario optimal reduction method for wind power time series," *IEEE Transactions on Power Systems*, vol. 31, no. 2, pp. 1657–1658, 2016.

[39] T. Li, Z. Li, and W. Li, "Scenarios analysis on the cross-region integrating of renewable power based on a long-period cost-optimization power planning model," *Renewable Energy*, vol. 156, pp. 851–863, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960148120306261

[40] M. Padhee and A. Pal, "Fast dtw and fuzzy clustering for scenario generation in power system planning problems," *arXiv preprint arXiv:2007.00805*, 2020.

[41] W. Lin, Z. Yang, J. Yu, K. Xie, X. Wang, and W. Li, "Tie-line security region considering time coupling," *IEEE Transactions on Power Systems*, vol. 36, no. 2, pp. 1274–1284, 2021.

[42] W. Deng, Y. Ji, and L. Tong, "Probabilistic forecasting and simulation of electricity markets via online dictionary learning," *arXiv preprint arXiv:1606.07855*, 2016.

[43] F. Capitanescu, "Critical review of recent advances and further developments needed in ac optimal power flow," *Electric Power Systems Research*, vol. 136, pp. 57–68, 2016.

[44] B. Zhang, R. Rajagopal, and D. Tse, "Network risk limiting dispatch: Optimal control and price of uncertainty," *IEEE Transactions on Automatic Control*, vol. 59, no. 9, pp. 2442–2456, 2014.

[45] R. Diestel, "Graph theory, volume 173 of," *Graduate texts in mathematics*, p. 7, 2012.

[46] D. S. Kirschen and G. Strbac, *Fundamentals of power system economics*. John Wiley & Sons, 2018.

[47] D. Bertsimas, J. N. Tsitsiklis, and J. Tsitsiklis, *Introduction to Linear Optimization*. Athena Scientific, 1997, vol. 6.

[48] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical review letters*, vol. 59, no. 19, p. 2229, 1987.

[49] A. Andoni, R. Panigrahy, G. Valiant, and L. Zhang, "Learning polynomials with neural networks," in *International conference on machine learning*, 2014, pp. 1908–1916.

[50] Z. Zhu, Y. Li, and Y. Liang, "Learning and generalization in overparameterized neural networks, going beyond two layers," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 6158–6169.

[51] S. Diamond and S. Boyd, "Cvxpy: A python-embedded modeling language for convex optimization," *The Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[52] M. Andersen, J. Dahl, and L. Vandenberghe. Cvxopt: A python package for convex optimization, version 1.1.6. [Online]. Available: https://cvxopt.org/

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. Let $f_{ij} = b_{ij}(\theta_i - \theta_j)$ 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 in either clockwise or counterclockwise direction. The weighted sum $f_{12}/b_{12} + f_{23}/b_{23} + \cdots + f_{n_c1}/b_{n_c1} = 0$ and therefore, the flows lie in a subspace.

Repeating the above calculation for every cycle in a network 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. There are multiple bases to choose the fundamental flows from. A popular way is to choose a spanning tree and consider the flows on the branches as fundamental, and everything else can be derived from them.

B. Proof of Theorem 6.4

Proof. Since $\boldsymbol{\ell}^{\text{new}}$ is in the convex hull of $\mathcal{D}_{trn}$, there are positive coefficients $\alpha_1, \ldots, \alpha_N$ such that

$$\boldsymbol{\ell}^{\text{new}} = \alpha_1 \boldsymbol{\ell}^1 + \cdots + \alpha_N \boldsymbol{\ell}^N,$$

and $\alpha_1 + \ldots, \alpha_n = 1$. By convexity,

$$g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{\text{new}}) \leq \alpha_1 g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^1) + \cdots + \alpha_N g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^N). \tag{25}$$

By the assumption that $g_{\boldsymbol{\theta}}(\boldsymbol{\ell})$ is well-trained, we have

$$\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^i) = \boldsymbol{\mu}, \text{ for } i = 1, \cdots, N. \tag{26}$$

Using first-order conditions of convex functions, we have $g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{\text{new}}) \geq g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^i) + \boldsymbol{\mu}(\boldsymbol{\ell}^{\text{new}} - \boldsymbol{\ell}^i)$ for all i. Multiplying the i'th equation by α_i and summing gives

$$g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{\text{new}}) \geq \alpha_1 g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^1) + \cdots + \alpha_N g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^N). \tag{27}$$

Combining (25) and (27) gives

$$g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{\text{new}}) = \alpha_1 g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^1) + \cdots + \alpha_N g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^N). \tag{28}$$

This implies the function is linear in the convex hull of $\mathcal{D}_{trn}$ and all the points have the same gradient. ■

C. Proof of Theorem 6.5

Suppose $g_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}$ is a convex function. Given $\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^N$ in the domain of g and let $\boldsymbol{\mu}^i = \nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^i)$. Let $\boldsymbol{\ell}^{\text{new}}$ be a point in the convex hull of $\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^N$ and denote $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new}) = \boldsymbol{\mu}$. By the convexity of $g_{\boldsymbol{\theta}}$, we have

$$(\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^i) - \nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new}))^T (\boldsymbol{\ell}^{\text{new}} - \boldsymbol{\ell}^i) \leq 0, \forall i. \tag{29}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2021.3124283, IEEE Transactions on Control of Network Systems

The inequalities in (29) constrain the values that $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})$ can take. We show that these inequalities actually describe a bounded polytope in $\mathbb{R}^n$ through a proof by contradiction.

Suppose the region defined by the inequalities in (29) is not bounded. Then $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})$ can be scaled arbitrarily and all of the inequalities in (29) would still hold. Then we can take the norm of $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})$ to be large enough such that it would dominate the $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{i})$ terms. Then (29) becomes

$$\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})^T(\boldsymbol{\ell}^{\text{new}} - \boldsymbol{\ell}^i) \geq 0, \forall i. \tag{30}$$

Since $\boldsymbol{\ell}^{\text{new}}$ is in the convex hull of $\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^N$, we can write it as $\boldsymbol{\ell}^{\text{new}} = \alpha_1 \boldsymbol{\ell}^1 + \cdots + \alpha_N \boldsymbol{\ell}^N$ and $\alpha_i \geq 0$ and sums up to 1. Substituting this into (30) and rearranging the terms, we have

$$\boldsymbol{\ell}^N \nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})^T \boldsymbol{\ell}^i \geq \sum_{i=1}^{N} \alpha_i \nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})^T \boldsymbol{\ell}^i.$$

By the assumption that $N \geq n+1$ and $\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^N$ are not in a lower dimensional subspace of $\mathbb{R}^n$, $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})^T \boldsymbol{\ell}^i$ will be nonzero for at least two i's. But it is not possible to have a convex combination of scalars (the $\nabla_{\boldsymbol{\ell}} g_{\boldsymbol{\theta}}(\boldsymbol{\ell}^{new})^T \boldsymbol{\ell}^i$'s) larger than every scalar in the set when at least two are nonzero (this follows from Farkas' lemma). This contradicts the assumption that the polytope created by (29) is unbounded.

D. Quadratic Costs

The DCOPF problem with quadratic cost is:

$$J(\boldsymbol{\ell}) = \min_{\mathbf{x},\mathbf{f}} \quad \sum_{i=1}^{n} \frac{q_i}{2} x_i^2 + c_i x_i \tag{31a}$$

$$\text{s.t. (1b), (1c) and (1d),} \tag{31b}$$

where q_i and c_i are the cost coefficients. As with the linear cost case, we assume that the multipliers ($\boldsymbol{\mu}$) with respect to the power balance equations have been learned. If q_i's are not zero, the dual of (31) is

$$\max_{\boldsymbol{\mu},\bar{\boldsymbol{\lambda}},\underline{\boldsymbol{\lambda}},\bar{\boldsymbol{\nu}},\underline{\boldsymbol{\nu}}} \quad \boldsymbol{\mu}^T\ \boldsymbol{\ell} - \underline{\boldsymbol{\lambda}}^T \bar{\mathbf{f}} - \bar{\boldsymbol{\lambda}}^T \bar{\mathbf{f}} - \bar{\boldsymbol{\nu}}^T \bar{\mathbf{x}} \tag{32a}$$

$$\text{s.t.} \quad \mathbf{Q}\mathbf{x} + \mathbf{c} - \boldsymbol{\mu} - \underline{\boldsymbol{\nu}} + \bar{\boldsymbol{\nu}} = \mathbf{0} \tag{32b}$$

$$-\tilde{\mathbf{A}}^T \boldsymbol{\mu} - \mathbf{K}^T \underline{\boldsymbol{\lambda}} + \mathbf{K}^T \bar{\boldsymbol{\lambda}} = \mathbf{0} \tag{32c}$$

$$\bar{\boldsymbol{\nu}} \geq \mathbf{0}, \underline{\boldsymbol{\nu}} \geq \mathbf{0}, \bar{\boldsymbol{\lambda}} \geq \mathbf{0}, \underline{\boldsymbol{\lambda}} \geq \mathbf{0}, \tag{32d}$$

where $\mathbf{Q}$ is a diagonal matrix with the value of q_i on the i'th diagonal. There are two differences between the linear and quadratic costs. The first is that the constraint associated with the generations, (32b), also include the primal variables $\mathbf{x}$. The second is that a quadratic program may not have the same number of binding constraints as the variables.

We use the following simple lemma to determine whether a generator constraint is binding by following simple economic principles. Specifically, given the optimal LMP $\boldsymbol{\mu}^*$, x_i is associated with the following active/inactive constraints:

$$x_i = \begin{cases} \bar{x}_i, & \text{if } \mu_i^* - c_i - 2q_i\bar{x}_i > 0 \\ 0 & \text{if } \mu_i^* - c_i < 0 \\ (0,\ \bar{x}_i)\,, & \text{otherwise} \end{cases} \tag{33}$$

Once $\boldsymbol{\mu}$ is known, the dual problem associated with $\underline{\boldsymbol{\lambda}}$ and $\bar{\boldsymbol{\lambda}}$ is identical to the linear cost case and the binding line constraints can be recovered through the same process.

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

$$\min\ \frac{1}{2}\mathbf{y}^T \hat{\mathbf{Q}} \mathbf{y} + \hat{\mathbf{c}}^T \mathbf{y} \tag{34a}$$

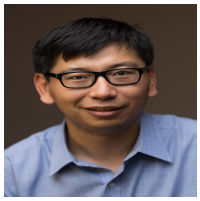
$$\text{s.t. } \mathbf{M}\mathbf{y} = \mathbf{a}, \tag{34b}$$

where $\hat{\mathbf{Q}}$ is a diagonal matrix with $(q_1, \ldots, q_n, 0, \ldots 0)$ on its diagonal and $\hat{\mathbf{c}} = (c_1, \ldots, c_n, 0, \ldots, 0)$. Fortunately (34) can be solved as a linear system. Following standard quadratic programming results, let $\boldsymbol{\tau}^*$ be the optimal Lagrangian multiplier of (34b), then the optimal solution of (34) is given by the following linear system

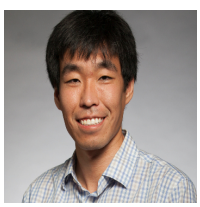
$$\begin{bmatrix} \hat{\mathbf{Q}} & \mathbf{M}^T \\ \mathbf{M} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y}^* \\ \boldsymbol{\tau}^* \end{bmatrix} = \begin{bmatrix} -\hat{\mathbf{c}} \\ \mathbf{a} \end{bmatrix}. \tag{35}$$

Once the active constraints are identified, a linear system of equations can again be solved to find the optimal solutions.

Ling Zhang received her B.S. degree in electronic engineering from the Ocean University of China in 2015, and the M.S. degree in the College of Information Science and Electronic Engineering from Zhejiang University, Hangzhou, China in 2018. She is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department from the University of Washington, Seattle, U.S. Her interests are in the area of power systems, learning and optimization.



Yize Chen is a Postdoctoral Scholar at the Berkeley Lab. He received his Bachelor of Science from Chu Kochen College at Zhejiang University in 2016 and his Ph.D. degree in ECE from the University of Washington in 2021. He will be joining the Hong Kong University of Science and Technology in 2022. His research interests are in control, optimization and machine learning for cyberphysical systems. He is a coauthor of papers that have received prize awards including ACM e-Energy, and Power Systems Computation Conference.



Baosen Zhang received his Bachelor in Engineering Science degree from the University of Toronto and his PhD degree in Electrical Engineering and Computer Sciences from University of California, Berkeley. He is currently the Keith and Nancy Endowed Career Development Professor in Electrical and Computer Engineering at the University of Washington. His research interests are in control, optimization and learning for power systems and other cyberphysical systems. He received the NSF CAREER award as well as several best paper awards.