DeepOPF: A Deep Neural Network Approach for Security-Constrained DC Optimal Power Flow

Xiang Pan, *Student Member, IEEE*, Tianyu Zhao, Minghua Chen, *Senior Member, IEEE*, and Shengyu Zhang

***Abstract*—We develop DeepOPF as a Deep Neural Network (DNN) approach for solving security-constrained direct current optimal power flow (SC-DCOPF) problems, which are critical for reliable and cost-effective power system operation. DeepOPF is inspired by the observation that solving SC-DCOPF problems for a given power network is equivalent to depicting a high-dimensional mapping from the load inputs to the generation and phase angle outputs. We first train a DNN to learn the mapping and predict the generations from the load inputs. We then directly reconstruct the phase angles from the generations and loads by using the power flow equations. Such a predict-and-reconstruct approach reduces the dimension of the mapping to learn, subsequently cutting down the size of the DNN and the amount of training data needed. We further derive a condition for tuning the size of the DNN according to the desired approximation accuracy of the load-generation mapping. We develop a post-processing procedure based on ℓ_1-projection to ensure the feasibility of the obtained solution, which can be of independent interest. Simulation results for IEEE test cases show that DeepOPF generates feasible solutions with less than 0.2% optimality loss, while speeding up the computation time by up to two orders of magnitude as compared to a state-of-the-art solver.**

***Index Terms*—Deep learning, Deep neural network, Optimal power flow.**

Nomenclature

Variable	Definition
$\mathcal{N}$	Set of buses, $N \triangleq \lvert\mathcal{N}\rvert$
$\mathcal{E}$	Set of branch
$\mathcal{G}$	Set of generators
$\mathcal{D}$	Set of load
$\mathcal{C}$	Set of contingency cases
P_G	Power generation injection vector, $[P_{G_i}, i \in \mathcal{N}]$
$P_G^{\min}$	Minimum generator output vector, $[P_{G_i}^{\min}, i \in \mathcal{N}]$
$P_G^{\max}$	Maximum generator output vector, $[P_{G_i}^{\max}, i \in \mathcal{N}]$
P_D	Power load vector, $[P_{D_i}, i \in \mathcal{N}]$
Θ_c	Voltage angle vector under the c-th contingency
$\theta_{c,i}$	Voltage angle under the c-th contingency for bus i
$\mathbf{B}_c$	Admittance matrix under the c-th contingency
$x_{ij,c}$	Line reactance from bus i to j under the c-th contingency
$P_{Tij,c}^{\max}$	Line transmission limit from bus i to j under the c-th contingency
N_{hid}	The number of hidden layers in the neural network

We use $|\cdot|$ to denote the size of a set. $P_{G_i} = P_{G_i}^{\min} = P_{G_i}^{\max} = 0, \forall i \notin \mathcal{G}$, and $P_{D_i} = 0, \forall i \notin \mathcal{D}$.

The work presented in this paper was supported in part by a Start-up Grant (Project No. 9380118) from City University of Hong Kong.

Xiang Pan and Tianyu Zhao are with the Department of Information Engineering, The Chinese University of Hong Kong (email: px018@ie.cuhk.edu.hk; zt017@ie.cuhk.edu.hk). Minghua Chen is with the School of Data Science, City University of Hong Kong (minghua.chen@cityu.edu.hk). Shengyu Zhang is with the Tencent Quantum Laboratory (email: shengyzhang@tencent.com). Corresponding author: Minghua Chen.

I. Introduction

The "deep learning revolution" largely enlightened by the October 2012 ImageNet victory [1] has transformed various industries in human society, including artificial intelligence, health care, online advertising, transportation, and robotics. As the most widely-used and mature model in deep learning, Deep Neural Network (DNN) [2] demonstrates superb performance in complex engineering tasks such as recommendation [3], bio-informatics [4], mastering difficult game like Go [5], and human pose estimation [6]. The capability of approximating continuous mappings and the desirable scalability make DNN a favorable choice in the arsenal of solving large-scale optimization and decision problems in engineering systems. In this paper, we apply DNN to power systems for solving the essential security-constrained direct current optimal power flow (SC-DCOPF) problem in power system operation.

The OPF problem, first posed by Carpentier in 1962 in [7], is to minimize an objective function, such as the cost of power generation, subject to all physical, operational, and technical constraints, by optimizing the dispatch and transmission decisions. These constraints include Kirchhoff's laws, operating limits of generators, voltage levels, and loading limits of transmission lines [8]. The OPF problem is central to power system operations as it underpins various applications including economic dispatch, unit commitment, stability and reliability assessment, and demand response. While OPF with a full AC power flow formulation (AC-OPF) is most accurate, it is a non-convex problem and its complexity obscures practicability. Meanwhile, based on linearized power flows, DC-OPF is a convex problem admitting a wide variety of applications, including electricity market clearing and power transmission management. See e.g., [9], [10] for a survey.

The SC-DCOPF problem, a variant of DC-OPF, is critical for reliable power system operation against contingencies caused by equipment failure [11]. It considers not only constraints under normal operation, but also additional steady-state security constraints for each possible contingency[1] [13]. Meanwhile, solving SC-DCOPF incurs excessive computa-

[1]There are two types of SC-DCOPF problems, namely the preventive SC-DCOPF problem and the corrective SC-DCOPF problem. Both of them are critical in practice. We focus on the preventive SC-DCOPF problem in this paper, in which the system operating decisions stay unchanged once determined and they need to satisfy both the pre- and post- contingency constraints. Usually, only line contingencies are considered in the preventive SC-DCOPF problem [12]. Our DeepOPF approach is also useful for the corrective SC-DCOPF problem, where the system operator only has a short time to adjust the operating points after the occurrence of a contingency. By DeepOPF, the system operator can obtain new operating points in a fraction of the time used by conventional solvers.

tional complexity, limiting its applicability in large-scale power networks [14].

To this end, we propose a machine learning approach for directly solving the SC-DCOPF problem. Our approach is based on the following observations.

- Given a power network, solving the SC-DCOPF problem is equivalent to depicting a high-dimensional mapping between load inputs and generations and voltages outputs.
- In practice, the SC-DCOPF problem is usually solved repeatedly for the same network, e.g., every 5 minutes [11], with different load inputs at different time epochs.

As such, it is conceivable to leverage the universal approximation capability of deep feed-forward neural networks [15]–[19], to learn the input-to-output mapping for a given power network, and then apply the mapping to obtain operating decisions upon giving load inputs (e.g., once every 5 minutes).[2]

Specifically, we develop DeepOPF as a DNN based solution for the SC-DCOPF problem. As compared to conventional approaches based on interior-point methods [20], DeepOPF excels in (i) reducing computing time and (ii) scaling well with the problem size. These salient features are particularly appealing for solving large-scale SC-DCOPF problems. Note that the complexity of constructing and training a DNN model is minor if amortized over many problem instances (e.g., one per every 5 minutes) that can be solved using the same model. We summarize our contributions as follows.

First, after reviewing the SC-DCOPF problem in Sec. III, we prospose DeepOPF as a DNN framework for solving the SC-DCOPF problem in Sec. IV. In DeepOPF, we first train a DNN to learn the load-generation mapping and predict the generations from the load inputs. We then directly reconstruct the phase angles from the generations and loads by using the (linearized) power flow equations. Such a predict-and-reconstruct two-step procedure reduces the dimension of the mapping to learn, subsequently cutting down the size of our DNN and the amount of training data/time needed. We also design a post-processing procedure based on ℓ_1-projection to ensure the feasibility of the final solution, which can be of independent interest.

Then in Sec. V, we derive a condition suggesting that the approximation accuracy of the neural network in DeepOPF decreases exponentially in the number of layers and polynomially in the number of neurons per layer. This allows us to tune the size of the neural network in DeepOPF according to the desired performance. We also analyze the computational complexity of DeepOPF.

Finally, we carry out simulations and summarize the results in Sec. VI. Simulation results of IEEE test cases show that DeepOPF generates feasible solutions with less than 0.2% optimality loss. As compared to a state-of-the-art solver, DeepOPF speeds up the computation time by up to two orders of magnitude under the typical load condition and by up to one order of magnitude under the congested load condition. The results also suggest a trade-off between the prediction accuracy and running time of DeepOPF.

Due to the space limitation, all proofs are in the supplementary material.

[2]Given a power network, as discussed in Sec. V, the mapping between the load input and the optimal solution of the SC-DCOPF problem is continuous and piece-wise linear. Existing works [15]–[19] show that the feed-forward neural networks can approximate real-valued continuous functions arbitrary well as the neural network size goes to infinity. Thus one can expect that a well-trained DNN would generate a close-to-optimal solution for the SC-DCOPF problem.

II. Related Work

Existing studies on solving SC-OPF focus on four lines of approaches. The first is on iteration-based algorithms. The SC-OPF problem is first approximated as an optimization problem, e.g., quadratic programming [21] or linear programming [22]. Then iteration-based algorithms, e.g., the interior-point method [20], [23], are applied to solve the approximated problems. The time complexity of iteration-based algorithms, however, can be substantial for large-scale power systems, limiting its applicability in practice. This is due to the significant number of constraints introduced by the consideration of a large number of contingencies. See, e.g., [13] for a survey on the iteration-based algorithms for solving SC-OPF problems.

The second approach is based on computational intelligence, e.g., evolutionary programming [24]–[26]. For instance, the authors of [24] propose a particle swarm optimization method for solving SC-OPF problems, in which they apply the particle swarm optimization (PSO) algorithm with reconstruction operators (PSO-RO) to find the solutions and designed an external penalty to ensure the feasibility of the obtained solution. Two limitations of this approach are the lack of performance guarantee and high computational complexity [27].

The third is on learning-based methods. There have been researches applying machine learning to various tasks in the power system, e.g., power system state estimation (PSSE) [28]; see [29] for a comprehensive survey. On solving OPF problems, existing studies mainly focus on integrating the learning techniques into conventional algorithms to facilitate the solving process [30], [31]. For instance, [30] applies a neural network to learn the system security boundaries as an explicit function to be used in the OPF formulation.

Recently, there is a line of research on determining the active/inactive constraints set to reduce the size of power system optimization problems, e.g., unit commitment and OPF problems, to accelerate the solving process [32]–[34]. The idea of this category of approaches is to reduce the scale of the problem without losing optimality. The speedup comes from the problem size reduction as any solvers can solve the reduced problem faster than solving the original problem directly. There is no optimality loss for this category of approach if the inactive/active constraints are identified correctly. Meanwhile, our DeepOPF approach develops a DNN-based solver for the OPF problem. It relies on having a large number of training data to train a DNN to predict generations from the input loads. The approach designs one solver for every interested OPF formulation. The advantage lies in that the DNN solvers can be much faster than conventional solvers. The optimality loss can be adjusted by constructing and training a larger DNN. The speedup comes from the new framework for designing OPF solvers. The disadvantage is two-fold. First, the approach

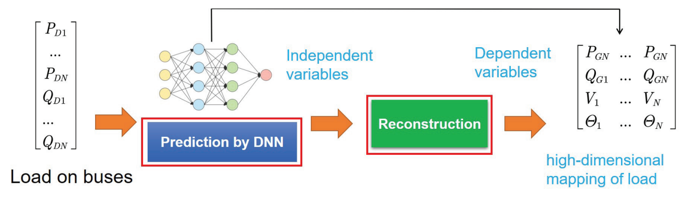

P_{D1} ... P_{DN} Q_{D1} ... Q_{DN}
Load on buses
Prediction by DNN
Independent variables
Reconstruction
Dependent variables
P_{G1} ... P_{GN} Q_{G1} ... Q_{GN} V_1 ... V_N Θ_1 ... Θ_N
high-dimensional mapping of load

Fig. 1: Overview of the predict-and-reconstruct framework.

would always incur optimality loss, due to the approximation errors of DNN. Second, one will need to design and train different solvers for different power networks and prepare a large amount of training data. In practice, these can be done offline.

We note that two approaches are orthogonal to each other and can be combined to achieve a speedup better than those by individual approaches. Specifically, given an OPF formulation, one can first identify and remove the inactive constraints to obtain a reduced problem (assuming no optimality loss). Then one can apply the pre-trained DNN solver for the reduced problem to obtain the solution. This way, the overall speedup performance is the product of that achieved by reducing problem size and that achieved by the DNN solver, better than those achievable by individual approaches. Both approaches may generate infeasible solutions. In such cases, one may apply the ℓ_1-projection based post-processing procedure, in Sec. IV-F, to obtain a feasible solution with less computational time than re-solving the original (quadratic) SC-DCOPF problem.

It is also conceivable to apply the K nearest neighbor (KNN) scheme to generate an approximate solution of the OPF problems given the load input. It is well understood that, as compared to the neural-network based approach, the KNN scheme incurs less training-time complexity but higher running-time complexity. This is due to that one has to identify the K nearest neighbors of the input, which is expensive for large-dimension problems.

To our best knowledge, DeepOPF is the first to develop a DNN-based solver for directly solving OPF problems. It learns the mapping from the load inputs to the generation and voltage outputs and directly obtains solutions for the SC-DCOPF problem with feasibility guarantees. As compared to our previous study in [35], this paper studies the more challenging SC-DCOPF problem and characterizes a condition for tuning the size of the DNN according to the desired approximation accuracy of the load-generation mapping. The predict-and-reconstruct DNN framework for solving OPF problems outlined in [35] (and this paper) applies to the AC-OPF setting as well. It has received growing interests with initial results reported in [36], [37], which demonstrate the speedup potential and highlight the challenges of ensuring solution feasibility.

III. Security-Constrained DCOPF Problem

We focus on the widely-studied $(N-1)$ SC-DCOPF problem considering contingencies due to the outage of any single transmission line. The objective is to minimize the total generation cost subject to the generator operation limits, the power balance equations, and the transmission line capacity constraints under all contingencies [38]. Assumed that the power network remains connected upon contingency, the SC-DCOPF problem is formulated as follows[3]:

$$\min_{\Theta_c, P_G} \quad \sum_{i\in\mathcal{G}} g_i\left(P_{Gi}\right) \tag{1}$$

$$\text{s.t.} \quad P_{Gi}^{\min} \leq P_{Gi} \leq P_{Gi}^{\max},\ i\in\mathcal{G}, \tag{2}$$

$$\mathbf{B}_c \cdot \Theta_c = P_G - P_D, c\in\mathcal{C}, \tag{3}$$

$$\frac{1}{x_{ij,c}}\left(\theta_{i,c}-\theta_{j,c}\right) \leq P_{Tij,c}^{\max},\ (i,j)\in\mathcal{E}, c\in\mathcal{C}. \tag{4}$$

Here $c=0$ denotes the case without any contingencies. $P_{Tij,c}^{\max}$ is the transmission limit for the branch connecting buses i and j. $\mathbf{B}_c$ is the admittance matrix for the c-th contingency, which is an $N\times N$ matrix with entries

$$B_{ij,c} = \begin{cases} 0, & \text{if } \ (i,j)\notin\mathcal{E}, i\neq j; \\ -\frac{1}{x_{ij,c}}, & \text{if } \ (i,j)\in\mathcal{E}; \\ \sum\limits_{k=1,k\neq i}^{N} \dfrac{1}{x_{ij,c}}, & \text{if } \ i=j. \end{cases}$$

The first set of constraints in the formulation describe the generation limits. The second set of constraints are the power flow equations with contingencies taken into account. The third set of constraints capture the line transmission capacity for both pre-contingency and post-contingency cases. In the objective, $g_i\left(P_{Gi}\right)$ is the cost function for the generator at the i-th bus, commonly modeled as a quadratic function [40]:

$$g_i\left(P_{Gi}\right) = \lambda_{1i}P_{Gi}^2 + \lambda_{2i}P_{Gi} + \lambda_{3i}, \tag{5}$$

where λ_{1i}, λ_{2i}, and λ_{3i} are the model parameters and can be obtained from measured data of the heat rate curve [41]. We note that the SC-DCOPF problem is a strictly convex (quadratic) problem and thus has a unique optimal solution. While the SC-DCOPF problem is important for reliable power system operation, solving it for large-scale power networks incurs excessive running time, limiting its practicability [14]. In the next section, we propose a neural network approach to solve the SC-DCOPF problem in a fraction of the time used by existing solvers.

IV. DeepOPF for Solving SC-DCOPF

A. A Neural-Network Approach for Solving OPF Problems

We outline a general predict-and-reconstruct framework for solving OPF in Fig. 1. Specifically, we exploit the dependency induced by the equality constraints among the decision variables in the OPF formulation. Given the load inputs, the learning model (e.g., DNN) is applied to predict only a set of *independent* variables. The remaining variables are then determined by leveraging the (power balance) equality constraints. This way, we not only reduce the number of variables to predict but also guarantee that the obtained solution always satisfies the equality constraints, which is usually difficult for

[3]We note that there is another formulation involving only generations as the phase angels can be uniquely determined by the generations and loads; see e.g., [38]. We focus on the standard formulation and both formulations incur the same order of running time complexity [39].

generic learning based approaches. In this paper, we follow this general approach to develop DeepOPF for solving the SC-DCOPF problem.

B. Overview of DeepOPF

The framework of DeepOPF is shown in Fig. 2, which is divided into a training stage and an inference stage. We first train a DNN to learn the load-generation mapping and predict the generations from the load inputs. We then directly compute the voltages from the generations and loads by using the (linearized) power flow equations.

We discuss the process of constructing and training the DNN model in the following subsections. In particular, we discuss the preparation of the training in Sec. IV-C, the variable prediction and reconstruction in Sec. IV-D, and the design and training of DNN in Sec. IV-E.

In the inference stage, we directly apply DeepOPF to solve the SC-DCOPF problem with given load inputs. DeepOPF may generate infeasible solutions due to the error in approximating the mapping. We describe a post-processing procedure based on ℓ_1-projection to ensure the feasibility of the obtained solutions in Sec. IV-F.

C. Load Sampling and Pre-processing

We sample the loads within $[(1-x)\cdot P_{Di}, (1+x)\cdot P_{Di}]$ uniformly at random, where P_{Di} is the default power load at the i-th bus and x is the percentage of sampling range, e.g., 10%. It is then fed into the traditional quadratic programming solver [42] to generate the optimal solutions. Uniform sampling is applied to avoid the over-fitting issue which is common in generic DNN approaches[4]. After that, the training data is normalized (using the statistical mean and standard variation) to improve training efficiency.

D. Generation Prediction and Phase Angle Reconstruction

We express P_{Gi} as follows, for $i \in \mathcal{G}$,

$$P_{Gi} = \alpha_i \cdot \left(P_{Gi}^{\max} - P_{Gi}^{\min}\right) + P_{Gi}^{\min}, \tag{6}$$

where $\alpha_i \in [0,1]$ is a scaling factor. It is clear that α_i and P_{G_i} have a one-to-one correspondence. Thus the scaling factors used in the training phase can be directly computed from the generated data. Meanwhile, instead of predicting the generations with diverse value ranges, we predict the scaling factor $\alpha_i \in [0,1]$ and recover P_{Gi} by using (6)). This simplifies the DNN output layer design to be discussed later. Note that the generation of the slack bus is obtained by subtracting generations of other buses from the total load.

Once we obtain P_G, we directly compute the phase angles by a useful property of the admittance matrices. We first obtain an $(N-1)\times(N-1)$ matrix, $\tilde{\mathbf{B}}_c$ by eliminating the row and column corresponding to the slack bus from the admittance matrix $\mathbf{B}_c$ for the c-th contingency. It is well-understood that $\tilde{\mathbf{B}}_c$ is a full-rank matrix [41]. Then we compute an $(N-1)$-dimensional phase angle vector $\tilde{\Theta}_c$ as

$$\tilde{\Theta}_c = \left(\tilde{\mathbf{B}}_c\right)^{-1}\left(\tilde{P}_G - \tilde{P}_D\right), \tag{7}$$

where $\tilde{P}_G$ and $\tilde{P}_D$ stand for the $(N-1)$-dimensional generation and load vectors for buses excluding the slack bus under each contingency, respectively. In the end, we output the N-dimensional phase angle vector $\Theta_{\underline{c}}$ by inserting a constant phase angle for the slack bus into $\tilde{\Theta}_c$.

There are two advantages to this design. On one hand, we use the property of the admittance matrix to reduce the number of variables to predict by our neural network, cutting down the size of our DNN model and the amount of training data/time needed. On the other hand, the equality constraints involving the generations and the phase angles can be satisfied automatically, which can be difficult to handle in alternative learning-based approaches.

E. The DNN Model

The core of DeepOPF is the DNN model, which is applied to approximate the load-generation mapping, given a power network. The DNN model is established based on the multi-layer feed-forward neural network structure, which consists of typical three-level network architecture: one input layer, several hidden layers, and one output layer. More specifically, the DNN model is defined as:

$$\begin{aligned} h_0 &= P_D, \\ h_i &= \sigma\left(W_i h_{i-1} + b_{i-1}\right), \forall\ i = 1, ..., N_{\text{hid}} \\ \hat{\alpha} &= \sigma'\left(w_o h_{\text{hid}} + b_o\right), \end{aligned}$$

where h_0 denotes the input vector of the network, h_i is the output vector of the i-th hidden layer and $\hat{\alpha}$ is the generated scaling factor vector for the generators.

1) The architecture: The i-th hidden layer models the interactions between features by introducing a connection weight matrix W_i and a bias vector b_i. The activation function $\sigma(\cdot)$ further introduces non-linearity into the hidden layers. We adopt the Rectified Linear Unit (ReLU) as the activation function of the hidden layers, which helps to accelerate the convergence and alleviate the vanishing gradient problem [1]. In addition, the Sigmoid function [2], $\sigma'(x) = \frac{1}{1+e^{-x}}$, is applied on the output layer to constrain the outputs within $(0,1)$.

2) The loss function: After constructing the DNN model, we need to design the corresponding loss function to guide the training. Since there exists a one-to-one correspondence between P_G and Θ_c, it suffices to focus on the loss of P_G, which is defined as the sum of mean square error between the obtained $\hat{\alpha}_i$ and the optimal scaling factors α_i as follows:

$$\mathcal{L}_{P_G} = \frac{1}{|\mathcal{G}|}\sum_{i\in\mathcal{G}}\left(\hat{\alpha}_i - \alpha_i\right)^2. \tag{8}$$

Meanwhile, we introduce a penalty term related to the inequality constraint into the loss function. We first introduce an $N_a \times N$ matrix $\mathbf{A}_c$ for each contingency c, where N_a is the

[4]For load inputs of large dimensions, the uniform mechanism may not be sufficient to guarantee enough good samples, especially near the boundary. In those cases, Markov chain Monte Carlo (MCMC) methods can be applied to sample according to a pre-specified probability distribution, to collect sufficient samples near the boundary of the sampling space.

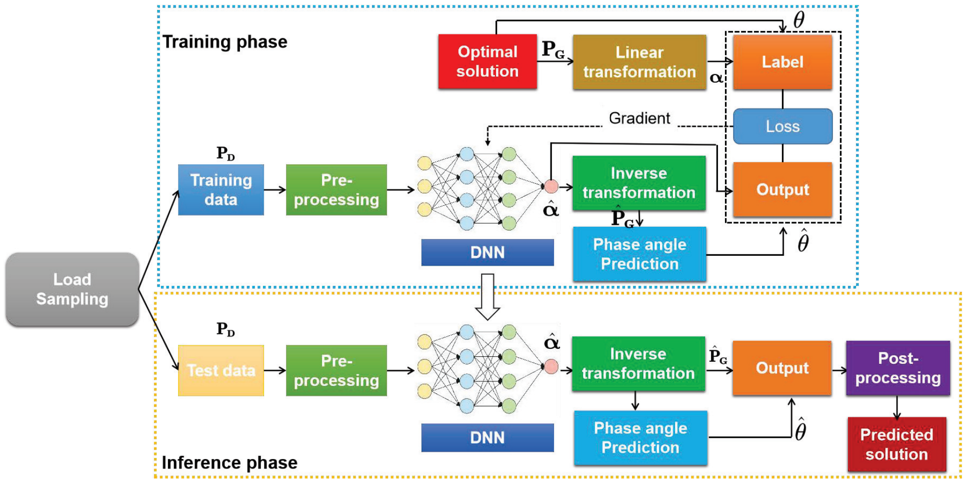


Fig. 2: The flow chart of DeepOPF.

number of adjacent buses. Each row in $\mathbf{A}_c$ corresponds to an adjacent bus pair. Given the k-th adjacent bus pair $(i_k, j_k) \in \mathcal{E}$, $k = 1, ..., N_a$, under the c-th contingency, let the power flow from the i_k-th bus to the j_k-th bus. Thus, the elements, $a_{ki_k,c}$ and $a_{kj_k,c}$, the corresponding entries of the matrix $\mathbf{A}_c$, are given as:

$$a_{ki_k,c} = \frac{1}{P^{\max}_{Ti_kj_k,c} \cdot x_{i_kj_k,c}} \text{ and } a_{kj_k,c} = \frac{-1}{P^{\max}_{Ti_kj_k,c} \cdot x_{i_kj_k,c}}. \tag{9}$$

Based on (7) and (9), the capacity constraints for the transmission line in (4) can be expressed as:

$$-1 \leq \left(\mathbf{A}_c \hat{\Theta}_c\right)_k \leq 1, k = 1, ..., N_a, c \in \mathcal{C}, \tag{10}$$

where $(\mathbf{A}_c \hat{\Theta}_c)_k$ represents the k-th element of $\mathbf{A}_c \hat{\Theta}_c$. Note that $\hat{\Theta}_c$ is the phase angle vector generated based on (7) and the discussion below it, and it is computed from P_G and P_D. We can then calculate $(\mathbf{A}_c \hat{\Theta}_c)_k$. The penalty term capturing the feasibility of the generated solutions is defined as:

$$\mathcal{L}_{pen} = \frac{1}{N_a} \sum_{k=1}^{N_a} \max\left(\left(\mathbf{A}_c \hat{\Theta}_c\right)_k^2 - 1, 0\right). \tag{11}$$

In summary, the loss function consists of two parts: the difference between the generated solution and the reference solution and the penalty upon solutions violating the inequality constraints. The total loss is a weighted sum of the two:

$$\mathcal{L}_{total} = w_1 \cdot \mathcal{L}_{P_G} + w_2 \cdot \mathcal{L}_{pen}, \tag{12}$$

where w_1 and w_2 are positive weighting factors for balancing the influence of each term in the training phase.

3) The training process: The training processing can be regarded as minimizing the average loss for the given training data by tuning the parameters of the DNN model as follows:

$$\min_{W_i, b_i} \frac{1}{N_T} \sum_{k=1}^{N_T} \mathcal{L}_{total,k} \tag{13}$$

where we recall that W_i and b_i, $i = 1, ..., N_{hid}$ represent the connection weight matrix and vector for layer i. N_T is the amount of training data and $\mathcal{L}_{total,k}$ is the loss of the k-th item in the training.

We apply the stochastic gradient descent (SGD) method with momentum [43] to solve the problem in (13), which is effective for the large-scale dataset and can economize on the computational cost at every iteration by choosing a subset of summation functions at every step.

F. Post-Processing

After obtaining a solution including the generations and phase angles, we check its feasibility by examining if it violates the generation limits and the line transmission limits. We output the solution if it passes the feasibility test. Otherwise, we solve the following ℓ_1-projection problem with linear constraints to obtain a feasible solution, [5]

$$\min \|\hat{P}_G - U\|_1 \quad \text{s.t.} \quad U \text{ satisfies (2)-(4)}, \tag{14}$$

where $\hat{P}_G$ is the solution predicted by DNN. We remark that such an ℓ_1-projection problem is indeed an LP and can be solved by off-the-shell solvers.

V. Performance Analysis of DeepOPF

A. Approximation Error of the Load-to-Generation Mapping

Given a power network, the SC-DCOPF problem is a quadratic programming problem with linear constraints. We denote the mapping between the load input P_D and the optimal generation P_G as $f^*(\cdot)$. Following the common practice in the deep-learning analysis (e.g., [44]–[46]) and without loss of generality, we focus on the case of one-dimensional output in the following analysis, i.e., $f^*(\cdot)$ is a scalar.[6] Assumed the

[5]It is common for machine learning approaches to generate infeasible solutions. The proposed post-processing procedure can then be applied to recover a feasible solution. The simulation results in Sec. VI show that DeepOPF with post-processing achieves decent speedup performance.

[6]To extend the results for mappings with one-dimensional output to mappings with multi-dimensional outputs, one can view the latter as multiple mappings each with one-dimensional output, apply the results for one-dimensional output multiple times, and combine them to get the one for multi-dimension output.

load input domain is compact, which usually holds in practice, $f^*(\cdot)$ has certain properties.

Lemma 1. *The function $f^*(\cdot)$ is piece-wise linear and Lipschitz-continuous. That is, there exists a constant $\Lambda > 0$, such that for any x_1, x_2 in the domain of $f^*(\cdot)$,*

$$|f^*(x_2) - f^*(x_1)| \leq \Lambda \cdot \|x_1 - x_2\|_2.$$

Define $f(\cdot)$ as the mapping between P_D and the generation obtained by DeepOPF by using a neural network with depth N_{hid} and maximum number of neurons per layer M. We focus on the case of one-dimensional output. As $f(\cdot)$ is generated from a neural network with ReLU activation functions, it is also piece-wise linear [47].

By exploiting the piece-wise linearity and the Lipschitz continuity, we analyze the approximation error between $f^*(\cdot)$ and $f(\cdot)$.

Theorem 2. *Let $\mathcal{H}$ be the class of all possible $f^*(\cdot)$ with a Lipschitz constant $\Lambda > 0$. Let $\mathcal{K}$ be the class of all $f(\cdot)$ generated by a neural network with depth N_{hid} and at most M neurons per layer.*

$$\max_{f^* \in \mathcal{H}} \min_{f \in \mathcal{K}} \max_{x \in \mathcal{S}} |f^*(x) - f(x)| \geq \Lambda \cdot \frac{d}{4 \cdot (2M)^{N_{hid}}}, \quad (15)$$

where d is the diameter of the load input domain $\mathcal{S}$.

The theorem characterizes a lower bound on the worst-case error of using neural networks to approximate load-generation mappings in SC-DCOPF problems. The bound is linear in d, which captures the size of the load input domain, and Λ, which captures the "curveness" of the mapping to learn. Meanwhile, interestingly, the bound decreases exponentially in the number of layers while polynomially in the number of neurons per layer. This suggests the benefits of using "deep" neural networks in mapping approximation, similar to the observations in [44]–[46][7].

A useful corollary suggested by Theorem 2 is the following.

Corollary 3. *The following gives a condition on the neural network parameters, such that it is ever possible to approximate the most difficult load-to-generation mapping with a Lipschitz constant Λ, up to an error of $\epsilon > 0$.*

$$(2M)^{N_{hid}} \geq \Lambda \cdot \frac{d}{4 \cdot \epsilon}, \quad (16)$$

where d is the diameter of the input domain $\mathcal{S}$.

The condition in (16) gives a necessary "size" of the neural network to achieve preferred approximation accuracy. If (16) is not satisfied, then there may exist a difficult mapping, even the smallest possible approximation error exceeds ϵ.

[7]While our observations are similar to those in [44]–[46], there is distinct difference in the results and the proof techniques as we explore the piece-wise linearity of the function unique to our setting.

B. Computational Complexity

Recall that N is the number of buses. The number of optimization variables in SC-DCOPF, including the generations and the phase angles of all the lines under all possible contingencies, and the constraints is $\mathcal{O}\left(N^3\right)$.

The computational complexity of interior point methods for solving SC-DCOPF as a convex quadratic problem is $\mathcal{O}\left(\left(N^3\right)^4\right) = \mathcal{O}\left(N^{12}\right)$, measured as the number of elementary operations assuming that each elementary operation takes a fixed amount of time to perform [20].

The computational complexity of DeepOPF consists of three parts. The first is the complexity of predicting the generations using the DNN, which is $\mathcal{O}\left(N_{hid}M^2\right)$ where M is the maximum number of neurons in each layer and N_{hid} is the number of hidden layers in DNN. See Appendix E of the supplementary materials for details of the analysis. To achieve satisfactory performance in terms of optmality loss and speed-up, we set M to be $\mathcal{O}(N)$ and N_{hid} to be 3. As such, the complexity for predicting the generations by our DNN is $\mathcal{O}\left(N^2\right)$.

The second is the complexity of computing the phase angles from the generations by directly solving (linearized) power flow equations and checking the feasibility of the results. The process involves solving $\mathcal{O}\left(N^2\right)$ sets of linear equations, one set for each contingency, and checking the transmission line limit constraints. The total complexity is $\mathcal{O}\left(N^5\right)$.

The third is the complexity of ℓ_1-projection, if the post-processing procedure is involved to ensure feasibility of the obtained solutions. The ℓ_1-projection is a linear programming problem and can be solved in $\mathcal{O}\left(\left(N^3\right)^{2.5}\right) = \mathcal{O}\left(N^{7.5}\right)$ amount of time by using algorithms based on fast matrix multiplication.

Overall, the total computational complexity of DeepOPF is $\mathcal{O}\left(N^5\right)$ if the post-processing procedure is not involved, for example, when the power system is operated in the light-load regime. Otherwise, it is $\mathcal{O}\left(N^{7.5}\right)$. In both cases, the complexity is substantially lower than that of solving the original SC-DCOPF problem directly by the conventional interior point method, which is $\mathcal{O}\left(N^{12}\right)$.

Our simulation results in Sec. VI corroborate the above observations. For both typical and congested settings, DeepOPF obtains quality solutions for SC-DCOPF problems in a fraction of the time used by a state-of-the-art solver with less than 0.2% optimality loss. We also note that the ℓ_1-projection in the post-processing procedure is an LP and can be solved efficiently by off-the-shelf solvers.

C. Trade-off between Accuracy and Complexity

The results in Theorem 2 and Proposition 6 suggest a trade-off between accuracy and complexity. In particular, we can tune the number of hidden layers N_{hid} and the maximum number of neurons per layer M to trade between the approximation accuracy and computational complexity of the DNN approach. It appears desirable to design multi-layer neural networks in DeepOPF as increasing N_{hid} may reduce the approximation error exponentially, but only increase the complexity linearly.

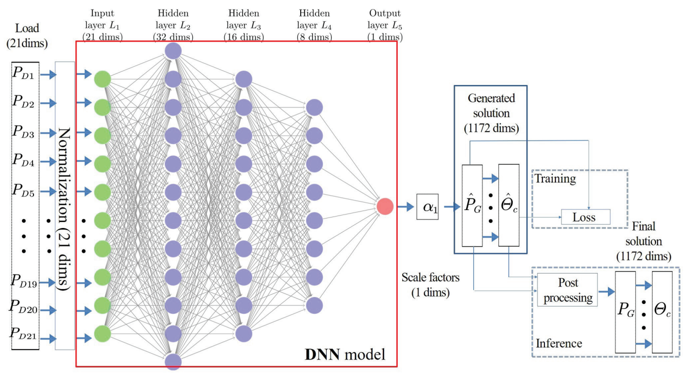


Fig. 3: The detail architecture of DNN model for IEEE Case30.

TABLE I: Parameters for test cases.

Case	N	$\lvert\mathcal{G}\rvert$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{E}\rvert$	N_{hid}	Neurons per hidden layer
IEEE Case30	30	2	21	41	3	32/16/8
IEEE Case57	57	4	42	80	3	32/16/8
IEEE Case118	118	19	99	186	3	128/64/32
IEEE Case300	300	57	199	411	3	256/128/64

* A bus is considered a load bus if its default active power consumption is positive.

VI. Numerical Experiments

A. Experiment Setup

1) Simulation environment: The experiments are conducted in CentOS 7.6 on the quad-core (i7-3770@3.40G Hz) CPU workstation and 16GB RAM.

2) Test case: We consider four IEEE standard cases in the Power Grid Lib [48] (version 19.05): the IEEE Case- /30/57/118/300 test systems, representing small-scale, medium-scale, and large-scale power networks, respectively. Their illustrations are in [49], [50] and their parameters are shown in Table I. For each case, we consider the typical operating conditions [48], where the active power loads are within the normal region and the branch limits are not binding during both the pre-/post- contingency cases. Note the power flow balance constraints are active so the SC-DCOPF under the typical operating conditions are still a constrained optimization problem. We illustrate the detailed architecture of our DNN model for the IEEE Case30 in Fig. 3.

3) Data preparation: In the training stage, the load data is sampled uniformly at random within $[90\%, 110\%]$ of the default value on each bus [48]. As the Power Grid Lib only has linear cost functions for generators, we use the cost functions from the test cases with same bus from MATPOWER [51] (version 7.0) while all other parameters are taken from the Power Grid Lib cases. Then we obtain the solution of the SC-DCOPF problems by Gurobi [42] (version 8.1.1). We sample 50,000 training data and 5,000 test data for each test case.

4) The implementation of the DNN model: We design the DNN model based on Pytorch platform and apply the stochastic gradient descent (SGD) method with momentum [43] to train the neural network. The epoch is set to 300 and the batch size is 64. We set the weighting factors in the loss function in (12) to be $w_1 = w_2 = 1$, based on empirical experience. The remaining parameters are shown in Table I, including the number of hidden layers and the number of neurons per layer.

5) Evaluation metrics: We compare the performance of DeepOPF and the state-of-the-art Gurobi solver[8] using the following metrics, averaged over 5,000 test instances. The first is the percentage of the feasible solution obtained by both approaches. The second is the objective cost obtained by both approaches. The third is the running time, i.e., the average computation time for obtaining solutions for the 5,000 instances. The fourth is the speedup, i.e., the average of the running-time ratios of the Gurobi solver to DeepOPF for all the test instances. It captures the average gain in computation time, of using DeepOPF over the Gurobi solver. We note that the speedup is the average of ratios, and it is different from the ratio of the average running times between the Gurobi solver and DeepOPF.

B. Performance under the Typical Operating Condition

The simulation results for the test cases under the typical operating conditions are shown in Table II and we have several observations. First, as compared to the Gurobi solver, DeepOPF speeds up the computing time by up to two orders of magnitude. The speedup increases as the test cases get larger, suggesting that DeepOPF is more efficient for large-scale power networks. Second, DeepOPF without involving the post-processing procedure always generates feasible solutions for IEEE Case30, IEEE Case57, and IEEE Case118, which justifies our design. We note that for IEEE Case300, DeepOPF achieves 81.7% feasibility rate before the post-processing procedure and overall 318 average speedup. Further analysis shows the average speedup for the test instances with feasible solutions generated by DNN (thus without involving the post-processing procedure) is 385 with an average running time of 15ms. For the remaining 18.3% test instances for which DNN generates infeasible solutions, it is due to the violation of 1 or 2 line capacity limit constraints. The ℓ_1-projection based post-processing procedure is involved to obtain feasible solutions, and the average running time of DeepOPF with ℓ_1-projection is 378ms. Overall, the average DeepOPF running time for all the IEEECase300 test instances is 81.4ms and the average speedup is 318. Third, the cost difference between the DeepOPF solution and the Gurobi solution is with less than 0.2% optimality loss (on average). We show detailed statistics of the optimality loss and the speedup for the IEEE Case118, in Appendix F of the supplementary materials, to further demonstrate the effectiveness of the DeepOPF. As compared to the optimal solution obtained by the Gurobi solver, DeepOPF achieves an average optimality loss less than 0.2% with the maximum around 1.2%. Meanwhile, DeepOPF achieves an average speedup of

[8]Gurobi implements the simplex algorithm for solving linear problems, which has a polynomial-time complexity with high probability [52] and a celebrated average-case running time performance. The Gurobi solver by default uses multi-threading technique, which affects the computing time due to the threads' communication overhead. For fair comparison, we use the single-threading setting in our simulations.

TABLE II: Performance comparison under typical operating conditions.

Test case	# Contingencies	# Variables	Feasibility before ℓ_1-projection (%)	Average cost ($/hr)		Optimality loss (%)	Running time (millisecond)		Speedup
				DeepOPF	Ref.		DeepOPF	Ref.	
IEEE Case30	38	1172	100	225.7	225.7	<0.1	0.72	17	×24
IEEE Case57	79	4564	100	9022.9	9021.6	<0.1	0.76	102	×133
IEEE Case118	177	21023	100	29197.9	29149.0	<0.2	2.48	698	×281
IEEE Case300	318	95757	81.7	156601.8	156542.5	<0.1	81.4	5766	×318

TABLE III: Performance under the typical, lightly-congested, and heavily-congested settings.

Scheme	Variants	Typical			Lightly-congested			Heavily-congested		
		Feasibility rate (%)	Optimality gap (%)	Speedup	Feasibility rate (%)	Optimality gap (%)	Speedup	Feasibility rate (%)	Optimality gap (%)	Speedup
DNN	with ℓ_1-projection	100	<0.1	338	100	<0.2	56	100	<0.2	×16.4
	without ℓ_1-projection	100	<0.1	338	15.7	<0.2	315	0	<0.2	–
KNN -50K	with ℓ_1-projection	100	<0.1	0.5	100	<0.6	0.7	100	<0.3	×1.5
	without ℓ_1-projection	100	<0.1	0.5	0	<0.9	–	0	<0.3	–

* '–' means the schemes fail to provide feasible solutions without post-processing thus do not associate with any speedup numbers.

×281 with the maximum around ×320. More details can be found in Appendix F of the supplementary materials.

C. Performance with High-Variation Load and under Congested Settings

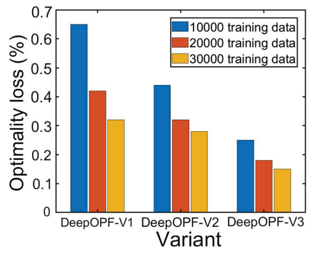
To stress-test DeepOPF, we enlarge the sampling range of the load on each bus and carry out simulations on IEEE Case118 under the typical, lightly-congested, and heavily-congested settings, by using and adjusting the typical and congested configurations of IEEE Case118 provided by the Power Grid Lib. For each setting, we sample 50,000 data in the load region for training and prepare another 5,000 for testing. For comparison, we evaluate the performance of the KNN scheme with $K = 50$, also using the same 50,000 sample data and 5,000 test data under each setting for fair comparison. We denote the scheme as KNN-50K. Its output is calculated as the average of the generation profiles (except the slack bus) of the K nearest neighbors of the input load in the training data set. Then the slack bus generation is computed to ensure the loads are satisfied. The phase angles on each bus can be uniquely determined by solving the power flow equations in (7). The feasibility of the power flow on each line is evaluated by (11).

The results are reported in Table III. For the typical setting, the load variation region is set as $[50\%, 150\%]$ of the default load and the line and generation limits are set according to the typical setting provided by the Power Grid Lib. Under such a setting, we observe none of the line constraints are binding in the test data set. As seen from the Table III, DeepOPF achieves a 100% feasibility rate, decent speedup, and 0.1% optimality loss. This implies DeepOPF works well on the high-variation load under the typical operating setting. Meanwhile, it achieves similar optimality gap performance as the KNN scheme, but a better speedup performance.

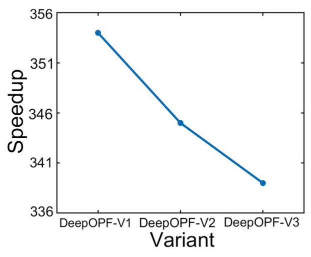
For the lightly-congested setting, the load variation region is set as $[50\%, 150\%]$ of the default load and the line and generation limits are set according to the congested setting in the Power Grid Lib. Under the setting, 85% of test cases have at least one line constraint binding. Under this setting, DeepOPF with the post-processing procedure generates feasible solutions with a 0.2% optimality loss and a ×56 speedup as compared to the Gurobi solver. For the 85% test cases with at least one line constraint binding, DeepOPF with the post-processing procedure obtains solutions with less than 0.2% optimality loss and a ×8 speedup. As compared to the KNN scheme, DeepOPF achieves better speedup and optimality gap performance. We note that the absolute load range under this lightly-congested setting is larger than the other two settings. Consequently, the samples are sparser, resulting in the worst optimality gap performance of KNN-50K, even worse than that under the heavily-congested setting.

For the heavily-congested setting, we set the load variation region to be within $[150\%, 160\%]$ and adjust the line flow limits under the largest load input. With the adjustment, all the 50,000 training and 5,000 test instances have about 20% line constraints binding. We note that the adjustment requires tuning the line limits to have as many lines constraints binding but without introducing post-contingency infeasibility. As we see from the simulation results, DeepOPF without post-processing fails to generate feasible solutions. In contrast, DeepOPF with post-processing generates 100% feasible solutions, with less than 0.2% optimality loss and a ×16 speedup as compared to the Gurobi solver. Under the heavily-congested setting, both KNN and DeepOPF scheme fail to generate feasible solutions for the test instances. After applying the l_1-projection based post-processing procedure, both schemes obtain feasible solutions and DeepOPF achieves better overall speedup performance and optimality loss performance.

Overall, under both lightly-congested and heavily-congested settings, our simulation results show that the post-processing step is more efficient than solving the original problem directly. The results also echo the general understanding that KNN incurs less training-time complexity but higher running-time complexity than neural-network based approaches.



(a)



(b)

Fig. 4: Performance under different neural network and training data sizes for IEEE Case118 under typical operating conditions.

D. *Performance with different DNN Scales and Training data Sizes*

When applying DNN approaches, it is of interest to evaluate the influence of the DNN's size and the amount of training data on the performance. In addition to the corresponding performance analysis w.r.t. the DNN's size in Sec. V, we carry out experiments to compare the optimality loss and speedup of DeepOPF with different neural network size and training data size for IEEE case118 under the typical operation condition. Three DNN models of different scales are used for comparison:

- DeepOPF-V1: A simple neural network with one hidden layer; the number of neurons is 16.
- DeepOPF-V2: A simple neural network with two hidden layers; the numbers of neurons per layer are 32 and 16, respectively.
- DeepOPF-V3: A simple neural network with three hidden layers; the numbers of neurons per layer are 64, 32, and 16, respectively.

The training data size varies from 10,000 to 30,000. The results are shown in Fig. 4(a) and Fig. 4(b). It is observed that larger training data size contributes to smaller optimality loss. Furthermore, we observe that when the depth and the size of the neural network increase, DeepOPF achieves better performance on optimality loss but less speedup. The above results correspond to our theoretical analysis on computational complexity and prediction accuracy in Sec. V-B and Sec. V-A, i.e., larger DNN size tends to have better prediction accuracy (smaller optimality loss) but also higher computational complexity. Having said so, the over-fitting issue may appear in practice if we keep increasing the depth and size. Thus, for different power networks (as IEEE test cases), the DNN model can be determined by educated guesses and iterative tuning, which is also by far the common practice in generic DNN approaches in various engineering domains.

E. *Performance with different Weighting Factors in Loss Function*

As shown Sec. IV-E, there are two weighting factors w_1 and w_2 in the loss function to balance between the training loss and the penalty of violating the inequality constraints. We carry out comparative experiments to evaluate the influence of the two hyper-parameters on the performance. More specifically, we use IEEE Case118 with 50% sampling range for testing, where the penalty is more likely to take effect as several transmission lines are binding. Three variants of the weighting factors in the loss function and the corresponding results are shown in Table IV. As seen, larger value of w_2 enhances the feasibility rate (before ℓ_1-projection) and the speedup as the post-processing step is involved in fewer test instances. In practice, the weight factors can be determined by educated guesses and iteratively adjusted to balance the influence of the two term in the loss function.

TABLE IV: Performance comparisons of different combinations of weights in the loss function.

Weight setting		Feasibility rate (%)	Optimality loss (%)	Speedup
$w_1 = 1$, $w_2 = 1$	with ℓ_1-projection	100	<0.2	×56
	without ℓ_1-projection	15.7	<0.2	×315
$w_1 = 1$, $w_2 = 10$	with ℓ_1-projection	100	<0.3	×83
	without ℓ_1-projection	23.8	<0.3	×324
$w_1 = 10$, $w_2 = 1$	with ℓ_1-projection	100	<0.1	×53
	without ℓ_1-projection	14.5	<0.1	×324

VII. Conclusion

We develop DeepOPF for solving SC-DCOPF problems. Given a power network, DeepOPF employs a DNN to learn a high-dimensional mapping between the load inputs and the dispatch decisions. With the learned mapping, it first obtains the generations from the load inputs and then directly computes the phase angels from the generations and loads. We also develop an ℓ_1-projection based post-processing procedure to ensure the feasibility of the obtained solution. We analyze the approximation capability and computational complexity of DeepOPF. Simulation results show that DeepOPF generates feasible solutions with less than 0.2% optimality loss. As compared to the Gurobi solver, DeepOPF speeds up the computation time by up to two orders of magnitude under the typical operating condition and by up to one order of magnitude under the congested condition. The approach may be computationally expensive in constructing and training the DNN model, which can be minor if amortized over many problem instances (e.g., one per every 5 minutes) that can be solved using the same model. Future directions include evaluating DeepOPF for national-scale power transmission networks and extending it to the AC-OPF setting [36].

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Proceedings of the International Conference on Neural Information Processing Systems, vol. 1, Lake Tahoe, Nevada, USA, 2012, pp. 1097–1105.
[2] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, Deep Learning. MIT Press Cambridge, 2016, vol. 1.

[3] P. Covington, J. Adams, and E. Sargin, “Deep Neural Networks for YouTube Recommendations,” in Proceedings of the ACM Conference on Recommender Systems, New York, NY, USA, Sep 2016, pp. 191–198.

[4] F. Wan, L. Hong, A. Xiao, T. Jiang, and J. Zeng, “NeoDTI: neural integration of neighbor information from a heterogeneous network for discovering new drug-target interactions,” Bioinformatics, vol. 35, no. 1, pp. 104–111, Jul 2018.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, no. 7587, pp. 484–489, Jan 2016.

[6] A. Toshev and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks,” in Proceeding of IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, June 2014, pp. 1653–1660.

[7] J. Carpentier, “Contribution to the economic dispatch problem,” Bulletin de la Societe Francoise des Electriciens, vol. 3, no. 8, pp. 431–447, 1962.

[8] D. E. Johnson, J. R. Johnson, J. L. Hilburn, and P. D. Scott, Electric Circuit Analysis. Prentice Hall Englewood Cliffs, 1989, vol. 3.

[9] S. Frank, I. Steponavice, and S. Rebennack, “Optimal power flow: a bibliographic survey i,” Energy Systems, vol. 3, no. 3, pp. 221–258, Sep 2012.

[10] ——, “Optimal power flow: a bibliographic survey ii,” Energy Systems, vol. 3, no. 3, pp. 259–289, Sep 2012.

[11] M. B. Cain, R. P. Oneill, and A. Castillo, “History of optimal power flow and formulations,” Federal Energy Regulatory Commission, vol. 1, pp. 1–36, 2012.

[12] A. J. Ardakani and F. Bouffard, “Identification of umbrella constraints in dc-based security-constrained optimal power flow,” IEEE Transactions on Power Systems, vol. 28, no. 4, pp. 3924–3934, 2013.

[13] F. Capitanescu, J. M. Ramos, P. Panciatici, D. Kirschen, A. M. Marcolini, L. Platbrood, and L. Wehenkel, “State-of-the-art, challenges, and future trends in security constrained optimal power flow,” Electric Power Systems Research, vol. 81, no. 8, pp. 1731–1741, 2011.

[14] N. Chiang and A. Grothey, “Solving security constrained optimal power flow problems by a structure exploiting interior point method,” Optimization and Engineering, vol. 16, no. 1, pp. 49–71, 2015.

[15] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,” Neural networks, vol. 6, no. 6, pp. 861–867, 1993.

[16] B. Hanin and M. Sellke, “Approximating continuous functions by relu nets of minimal width,” arXiv preprint arXiv:1710.11278, 2017.

[17] P. Kidger and T. Lyons, “Universal approximation with deep narrow networks,” arXiv preprint arXiv:1905.08539, 2019.

[18] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” Neural networks, vol. 4, no. 2, pp. 251–257, 1991.

[19] B. Karg and S. Lucia, “Efficient representation and approximation of model predictive control laws via deep learning,” arXiv preprint arXiv:1806.10644, 2018.

[20] Y. Ye and E. Tse, “An extension of karmarkar’s projective algorithm for convex quadratic programming,” Mathematical Programming, vol. 44, no. 1, pp. 157–179, May 1989.

[21] J. A. Momoh, “A generalized quadratic-based model for optimal power flow,” in Proceedings of IEEE International Conference on Systems, Man and Cybernetics, vol. 1, Cambridge, MA, USA, Nov 1989, pp. 261–271.

[22] S. H. Low, “Convex relaxation of optimal power flowpart i: Formulations and equivalence,” IEEE Transactions on Control of Network Systems, vol. 1, no. 1, pp. 15–27, March 2014.

[23] H. Wang, C. E. Murillo-Sanchez, R. D. Zimmerman, and R. J. Thomas, “On computational issues of market-based optimal power flow,” IEEE Transactions on Power Systems, vol. 22, no. 3, pp. 1185–1193, 2007.

[24] P. E. O. Yumbla, J. M. Ramirez, and C. A. C. Coello, “Optimal power flow subject to security constraints solved with a particle swarm optimizer,” IEEE Transactions on Power Systems, vol. 23, no. 1, pp. 33–40, 2008.

[25] N. Amjady, H. Fatemi, and H. Zareipour, “Solution of optimal power flow subject to security constraints by a new improved bacterial foraging method,” IEEE Transactions on Power Systems, vol. 27, no. 3, pp. 1311–1323, 2012.

[26] 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.

[27] P. A. Vikhar, “Evolutionary algorithms: A critical review and its future prospects,” in 2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC). IEEE, 2016, pp. 261–265.

[28] L. Zhang, G. Wang, and G. B. Giannakis, “Real-time power system state estimation and forecasting via deep unrolled neural networks,” IEEE Transactions on Signal Processing, vol. 67, no. 15, pp. 4069–4077, 2019.

[29] L. A. Wehenkel, Automatic learning techniques in power systems. Springer Science & Business Media, 2012.

[30] V. J. Gutierrez-Martinez, C. A. Cañizares, C. R. Fuerte-Esquivel, A. Pizano-Martinez, and X. Gu, “Neural-network security-boundary constrained optimal power flow,” IEEE Transactions on Power Systems, vol. 26, no. 1, pp. 63–72, 2010.

[31] F. Hasan, A. Kargarian, and A. Mohammadi, “A Survey on Applications of Machine Learning for Optimal Power Flow,” in 2020 IEEE Texas Power and Energy Conference (TPEC). IEEE, 2020, pp. 1–6.

[32] Q. Zhai, X. Guan, J. Cheng, and H. Wu, “Fast identification of inactive security constraints in scuc problems,” IEEE Transactions on Power Systems, vol. 25, no. 4, pp. 1946–1954, 2010.

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

[34] L. Duchesne, E. Karangelos, and L. Wehenkel, “Recent Developments in Machine Learning for Energy Systems Reliability Management,” Proceedings of the IEEE, vol. 108, no. 9, pp. 1656–1676, 2020.

[35] X. Pan, T. Zhao, and M. Chen, “DeepOPF: Deep Neural Network for DC Optimal Power Flow,” in IEEE SmartGridComm, Oct. 2019.

[36] X. Pan, M. Chen, T. Zhao, and S. H. Low, “DeepOPF: A Feasibility-Optimized Deep Neural Network Approach for AC Optimal Power Flow Problems,” arXiv preprint arXiv:2007.01002, 2020.

[37] A. Zamzam and K. Baker, “Learning optimal solutions for extremely fast AC optimal power flow,” arXiv preprint arXiv:1910.01213, 2019.

[38] B. Stott, J. Jardim, and O. Alsac, “DC Power Flow Revisited,” IEEE Transactions on Power Systems, vol. 24, no. 3, pp. 1290–1300, Aug 2009.

[39] V. H. Hinojosa and F. Gonzalez-Longatt, “Preventive Security-Constrained DCOPF Formulation Using Power Transmission Distribution Factors and Line Outage Distribution Factors,” Energies, vol. 11, no. 6, 2018.

[40] J. H. Park, Y. S. Kim, I. K. Eom, and K. Y. Lee, “Economic load dispatch for piecewise quadratic cost function using hopfield neural network,” IEEE Transactions on Power Systems, vol. 8, no. 3, pp. 1030–1038, Aug 1993.

[41] R. D. Christie, B. F. Wollenberg, and I. Wangensteen, “Transmission management in the deregulated environment,” Proceedings of the IEEE, vol. 88, no. 2, pp. 170–195, Feb 2000.

[42] L. Gurobi Optimization, “Gurobi optimizer reference manual,” 2019. [Online]. Available: http://www.gurobi.com

[43] N. Qian, “On the momentum term in gradient descent learning algorithms,” Neural networks, vol. 12, no. 1, pp. 145–151, 1999.

[44] D. Yarotsky, “Error bounds for approximations with deep ReLU networks,” Neural Networks, vol. 94, pp. 103–114, 2017.

[45] I. Safran and O. Shamir, “Depth-width tradeoffs in approximating natural functions with neural networks,” in International Conference on Machine Learning, 2017, pp. 2979–2987.

[46] S. Liang and R. Srikant, “Why deep neural networks for function approximation?” arXiv preprint arXiv:1610.04161, 2016.

[47] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, “On the number of linear regions of deep neural networks,” in Advances in Neural Information Processing Systems 27, 2014, pp. 2924–2932.

[48] S. Babaeinejadsarookolaee, A. Birchfield, R. D. Christie, C. Coffrin, C. DeMarco, R. Diao, M. Ferris, S. Fliscounakis, S. Greene, R. Huang et al., “The power grid library for benchmarking ac optimal power flow algorithms,” arXiv preprint arXiv:1908.02788, 2019.

[49] C. H. Liang, C. Y. Chung, K. P. Wong, and X. Z. Duan, “Parallel Optimal Reactive Power Flow Based on Cooperative Co-Evolutionary Differential Evolution and Power System Decomposition,” IEEE Transactions on Power Systems, vol. 22, no. 1, pp. 249–257, Feb 2007.

[50] “IEEE case300 topology,” 2018, https://www.al-roomi.org/power-flow/300-bus-system.

[51] R. D. Zimmerman, C. E. Murillo-Sánchez, R. J. Thomas et al., “MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education,” IEEE Transactions on Power Systems, vol. 26, no. 1, pp. 12–19, 2011.

[52] P. M. Vaidya, “Speeding-up linear programming using fast matrix multiplication,” in IEEE FOCS, 1989, pp. 332–337.

[53] M. Telgarsky, "Benefits of depth in neural networks," arXiv preprint arXiv:1602.04485, 2016.
[54] S.-G. Chen and P. Hsieh, "Fast computation of the nth root," Computers & Mathematics with Applications, vol. 17, no. 10, pp. 1423–1427, 1989.
[55] D. M. Gordon, "A survey of fast exponentiation methods," J. Algorithms, vol. 27, no. 1, pp. 129–146, 1998.

Xiang Pan (S'18) received the M.S. degree in computer science from Wuhan University, Wuhan, China, in 2018. He is currently pursuing the Ph.D. degree with the Department of Information Engineering, The Chinese University of Hong Kong. His research interest includes machine learning and its application in power systems.

Tianyu Zhao received the B.Eng. degree from the School of Energy and Power Engineering and Minor's Diploma in Business Administration from the School of Management, Xi'an Jiaotong University, Xi'an, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Information Engineering, The Chinese University of Hong Kong. His research interests include electricity market, microgrid operation, and machine learning applications in power systems.

Minghua Chen (S'04-M'06-SM'13) received his B.Eng. and M.S. degrees from the Dept. of Electronic Engineering at Tsinghua University. He received his Ph.D. degree from the Dept. of Electrical Engineering and Computer Sciences, University of California Berkeley. He is currently a Professor in the School of Data Science, City University of Hong Kong. He received the Eli Jury award from UC Berkeley in 2007 (presented to a graduate student or recent alumnus for outstanding achievement in the area of Systems, Communications, Control, or Signal Processing) and The Chinese University of Hong Kong Young Researcher Award in 2013. He also received several best paper awards, including the IEEE ICME Best Paper Award in 2009, the IEEE Transactions on Multimedia Prize Paper Award in 2009, and the ACM Multimedia Best Paper Award in 2012. He also co-authors several papers that are Best Paper Award Runner-up/Finalist/Candidate for ACM MobiHoc in 2014 and ACM e-Energy in 2015, 2016, 2018, and 2019. Prof. Chen serves as TPC Co-Chair, General Chair, and Steering Committee Chair of ACM e-Energy in 2016, 2017, and 2018-present, respectively. He also serves as Associate Editor of IEEE/ACM Transactions on Networking in 2014-2018. He receives the ACM Recognition of Service Award in 2017 for the service contribution to the research community. He is currently serving as TPC Co-Chair for ACM MobiHoc 2020. His current research interests include online optimization and algorithms, energy systems (e.g., smart power grids and energy-efficient data centers), intelligent transportation systems, distributed optimization, delay-constrained network communication, and capitalizing the benefit of data-driven prediction in algorithm/system design.

Shengyu Zhang obtained his bachelor degree in mathematics, Fudan University in 1999, master in computer science, Tsinghua University in 2002, under the supervision of Mingsheng Ying, and Ph.D. in computer science, Princeton University in 2006, under the supervision of Prof. Andrew Yao. He then worked in NEC Laboratories America as a summer intern, and in California Institute of Technology for a two-year postdoc, hosted by Prof. John Preskill, Prof. Leonard Schulman and Prof. Alexei Kitaev. He joined The Chinese University of Hong Kong as an assistant professor in 2008, and became an associate professor in 2014. In January 2018, he joined Tencent as a Distinguished Scientist, in charge of Tencent Quantum Laboratory.

Shengyu Zhang's research interest lies in quantum computing, algorithm designing, and foundation of artificial intelligence. He is an editor of Theoretical Computer Science, and of International Journal of Quantum Information. He published numerous papers and served as PC member in leading conferences in theoretical computer science, quantum computing, and artificial intelligence.

Supplementary Materials

Appendix A
Proof of Lemma 1

Proof. We now show the considered piece-wise linear one-dimensional output function $f^*(\cdot)$ is Lipschitz-continuous in the input domain $\mathcal{S}$, which can be partitioned into r different convex polyhedral regions, $R_i, i = 1, ..., r$. The mapping $f^*(\cdot)$ is piece-wise linear and can be defined as follows:

$$f^*(x) = \begin{cases} a_1 x + b_1, & \text{if } x \in R_1; \\ a_2 x + b_2, & \text{if } x \in R_2; \\ \cdots \\ a_r x + b_r, & \text{if } x \in R_r; \end{cases}$$

where $x \in \mathbf{R}^{n\times 1}, a_i \in \mathbf{R}^{1\times n}, i = 1, ..., r$ and $b_i \in \mathbf{R}^1, i = 1, ..., r$. Then, we can have:

$$|f^*(x_1) - f^*(x_2)| \le \|a_i\| \cdot \|x_1 - x_2\|, \ \forall x_1, x_2 \in \mathcal{S}.$$

Thus, let $\Lambda = \max\{\|a_i\|, \ldots, \|a_r\|\}$. We have

$$|f^*(x_1) - f^*(x_2)| \le \Lambda \cdot \|x_1 - x_2\|, \ \forall x_1, x_2 \in \mathcal{S}.$$

Therefore, $f^*(\cdot)$ is Lipschitz-continuous. □

Appendix B
Proof of Lemma 4

Before we proceed, we present a result on the approximation error between two scalar function classes.

Lemma 4. *Let $\mathcal{H}$ be the class of two-segment piece-wise linear functions with a Lipschitz constant $\Lambda > 0$, over an interval $[-\mu, \mu]$ ($\mu > 0$). Let $\mathcal{K}$ be the class of all linear scalar functions over $[-\mu, \mu]$. Then, the following holds,*

$$\max_{h\in\mathcal{H}} \min_{g\in\mathcal{K}} \max_{x\in[-\mu,\mu]} |h(x) - g(x)| \ge \Lambda \cdot \frac{\mu}{2}. \tag{17}$$

Essentially, the lemma gives a lower bound to the worst-case error of using a linear function to approximate a two-segment piece-wise linear function.

Proof. We can derive the lower bound to the worst-case L_∞-based approximation error as follows. Suppose we want to find a function $g(\cdot)$ belongs to the linear scalar function class $\mathcal{K}$ to approximate the function h belongs to the two-segment piece-wise linear function class $\mathcal{H}$ with a Lipschitz constant $\Lambda > 0$, over an interval $[-\mu, \mu]$ ($\mu > 0$). An illustration is shown in Fig. 5. Let $g(x) = a \cdot x + b$, for $x \in [-\mu, \mu]$. Let $\hat{h} \in \mathcal{H}$ be the following:

$$\hat{h}(x) = \begin{cases} \Lambda(x + \mu), & \text{if } x \in [-\mu, 0]; \\ -\Lambda(x - \mu), & \text{if } x \in [0, \mu]; \end{cases} \tag{18}$$

Then, we can obtain the lower bound for the L_∞-based approximation error of $\hat{h}(\cdot)$ and $g(\cdot)$ by the classification discussion on the intercept b.

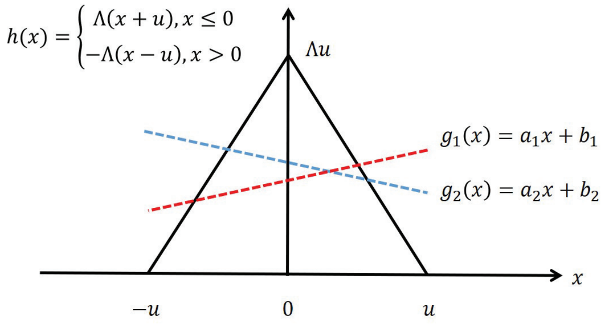

$h(x) = \begin{cases} \Lambda(x+u), x \le 0 \\ -\Lambda(x-u), x > 0 \end{cases}$
Λu
$g_1(x) = a_1x + b_1$
$g_2(x) = a_2x + b_2$
x
$-u$
0
u

Fig. 5: Illustration of approximating a two-segment piece-wise Lipschitz-continuous function $h(\cdot)$ by a linear function $g(\cdot)$.

- If $b \le \frac{\Lambda\mu}{2}$. Under this case, we can get:

$$\begin{aligned} \max_{x\in[-\mu,\mu]} \left|\hat{h}(x) - g(x)\right| &\ge \left|\hat{h}(0) - g(0)\right| \\ &\ge \Lambda \cdot \frac{\mu}{2n} \\ &= |\Lambda\mu - b| \\ &\ge \Lambda \cdot \frac{\mu}{2}. \end{aligned}$$

- Otherwise $\frac{\Lambda\mu}{2} < b$. If $a > 0$, under this case we can have:

$$\begin{aligned} \max_{x\in[-\mu,\mu]} \left|\hat{h}(x) - g(x)\right| &\ge \left|\hat{h}(\mu) - g(\mu)\right| \\ &\ge (\Lambda + a) \cdot \frac{\mu}{2} \\ &\ge \Lambda \cdot \frac{\mu}{2}. \end{aligned}$$

  Otherwise $a \le 0$, we can consider the point $x = -\mu$ and obtain the same result.

Thus overall, we observe

$$\min_{g\in\mathcal{K}} \max_{x\in[-\mu,\mu]} \left|\hat{h}(x) - g(x)\right| = \Lambda \cdot \frac{\mu}{2}.$$

For the worst-case L_∞-based approximation error, we have

$$\begin{aligned} &\max_{h\in\mathcal{H}} \min_{g\in\mathcal{K}} \max_{x\in[-\mu,\mu]} |h(x) - g(x)| \\ &\ge \min_{g\in\mathcal{K}} \max_{x\in[-\mu,\mu]} \left|\hat{h}(x) - g(x)\right| \\ &\ge \Lambda \cdot \frac{\mu}{2}. \end{aligned}$$

□

Appendix C
Proof of Theorem 2

Proof. Suppose $\mathcal{K}$ is the family of piece-wise linear functions generated by a neural network with depth N_{hid} and maximum number of neurons per layer M, on the load input domain $\mathcal{S}$ with the diameter d. The maximum number of segments any functions in $\mathcal{K}$ can have is defined as n. Let $\mathcal{H}$ be the class of all possible $f^*(\cdot)$ with a Lipschitz constant $\Lambda > 0$. Let $[a_i, a_{i+1}]$, $0 \le i \le 2n-1$, be $2n$ intervals with equal length portioning the diameter of input domain. Define $\hat{f} \in \mathcal{H}$ as follows:

$$\hat{f}(x) = \begin{cases} \Lambda(x - a_i), \text{ if } x \in [a_i, a_{i+1}], i = 0, 2, \ldots, 2n-2; \\ -\Lambda(x - a_{i+2}), \text{ if } x \in [a_{i+1}, a_{i+2}], i = 0, 2, \ldots, 2n-2. \end{cases}$$

Consider any $f \in \mathcal{K}$, since f is piece-wise linear with at most n segments over the input domain, it must be linear over one of the following n segments $[a_i, a_{i+2}], i = 0, 2, ..., 2n-2$. Over that particular segment, we apply Lemma 4 to bound the approximation error as in (17). Overall, we have

$$\min_{f \in \mathcal{K}} \max_{x \in \mathcal{S}} \left| \hat{f}(x) - f(x) \right| \geq \Lambda \cdot \frac{d}{4n}. \tag{19}$$

Since the above inequality holds for a particular choice of $\hat{f} \in \mathcal{H}$, we must have

$$\max_{f^* \in \mathcal{H}} \min_{f \in \mathcal{K}} \max_{x \in \mathcal{S}} |f^*(x) - f(x)| \geq \Lambda \cdot \frac{d}{4n}. \tag{20}$$

Meanwhile, we use the result in [53], of which the following is an immediate corollary.

Corollary 5. *The maximum number of linear segments generated from the family of ReLU neural networks with depth (the number of hidden layers) l and maximal width (neurons on the hidden layer) m is $(2m)^l$.*

By the above corollary, we have $n \leq (2M)^{N_{hid}}$. Plugging the relationship into (20), we have

$$\max_{f^* \in \mathcal{H}} \min_{f \in \mathcal{K}} \max_{x \in \mathcal{S}} |f^*(x) - f(x)| \geq \Lambda \cdot \frac{d}{4 \cdot (2M)^{N_{hid}}}. \tag{21}$$

□

Appendix D
Proof of Corollary 3

Proof. We next will show how to derive the Corollary 3. Suppose ϵ is defined as the upper bound for the worst-case approximation error, that is:

$$\max_{f^* \in \mathcal{H}} \min_{f \in \mathcal{G}} \max_{x \in \mathcal{D}} |f^*(x) - f(x)| \leq \epsilon \tag{22}$$

Then, we can derive the following inequality based on the above definition and Theorem 2:

$$\Lambda \cdot \frac{d}{4 \cdot (2M)^{N_{hid}}} \leq \epsilon, \tag{23}$$

After some transformations, we can obtain the following necessary condition related to the DNN's scale on the Corollary 3, which can guarantee that the designed DNN's ever possible to approximate the most difficult load-to-generation mapping with a Lipschitz constant Λ, up to an error of $\epsilon > 0$:

$$(2M)^{N_{hid}} \geq \Lambda \cdot \frac{d}{4 \cdot \epsilon}. \tag{24}$$

□

Appendix E
Computational complexity of DeepOPF for Predicting the Generations

Recall that the number of bus and the number of contingencies are N and C, respectively. The input and the output of the DNN model have K_{in} and K_{out} dimensions, and the DNN model has N_{hid} hidden layers and each hidden layer has at most M neurons. Specifically, in our setting, K_{in} equals to the number of buses with load and K_{out} equals to the number of generators. Therefore, the input and output dimensions are of the same order of N. From empirical experience, we set M to be on the same order of N and set N_{hid} to be a constant. Once we finish training the DNN model, the complexity of generating solutions by using DeepOPF is characterized in the following proposition.

Proposition 6. *The computational complexity (measured as the number of arithmetic operations) to generate the generations to the SC-DCOPF problem by using DeepOPF is*

$$T = K_{in}K_1 + \sum_{i=1}^{N_{hid}-1} K_i K_{i+1} + K_{out}N_{hid}, \tag{25}$$

which is $\mathcal{O}\left(N_{hid}M^2\right)$.

Note that N_{hid} is set to 3 and M is set to be $\mathcal{O}(N)$. The complexity of DeepOPF for predicting the generations is $\mathcal{O}\left(N^2\right)$, smaller than that of the interior point method.

Proof. We next will show how to derive the computational complexity of using the DNN model to obtain the generation output from the given input. Recall that the input and the output of the DNN model in DeepOPF are K_{in} and K_{out} dimensions, respectively, and the DNN model has N_{hid} hidden layers and each hidden layer has K_{i} neurons, for $i = 1, ..., N_{hid}$. The maximal neurons on the hidden layers is M neurons. For each neuron in the DNN model, we can regard the computation complexity on each neuron (measured by basic arithmetic operation) as $\mathcal{O}(1)$. As we apply the fully-connected architecture, the output of each neuron is calculated by taking a weighted sum of the output from the neurons on the previous hidden layer and passing through a activation function.

Thus, the computational complexity (measured as the number of arithmetic operations) to generate the output from the input by a DNN model consists of the following three parts:

- Complexity of computation from the input to the first hidden layer. As each neuron on the first hidden layer will take the input data, thus the corresponding complexity is $\mathcal{O}(K_{in}K_1)$.
- Complexity of computation between the consecutive hidden layers. Since each neuron on the current hidden layer will take the output from each neuron on the previous hidden layer as the input data. Thus, thus the corresponding complexity is $\mathcal{O}\left(\sum_{i=1}^{N_{hid}-1} K_i K_{i+1}\right)$.
- Complexity of computation from the last hidden layer to the output. As the output of each neuron on the last hidden layer is used to calculated the output, the corresponding complexity is $\mathcal{O}(N_{hid}K_{out})$.

The Sigmoid function is applied to each element of the output in order to guarantee that the elements of the final output is within $(0, 1)$. The Sigmoid function takes the form of

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

and computing a Sigmoid function involves one addition operation, one division operation, and one exponentiation operation. The exponentiation operation is essentially a combination of n-th power operation and m-th root operation, where n and m are some integers depending on the output element x. That is, $x = \frac{n}{m}$. Previous works show that the computational complexity of n-th multiplication operations and m-th root operations is $\mathcal{O}(\log n \cdot \log m)$ [54], [55]. Therefore, a Sigmoid function requires $\mathcal{O}(\log n \cdot \log m)$ operations. In practice, both n and m are bounded by some constant integer M in the actual computation process, and therefore the computational complexity for the Sigmoid function is $(\log M)^2$, which is a constant too. In our DeepOPF solution, the output layer of DNN has K_{out} neurons with Sigmoid function, the corresponding computational complexity (the number of arithmetic operations) for the output layer is $\mathcal{O}(K_{out})$.

Hence, the overall complexity of the calculation by a DNN model is:

$$\begin{aligned} T &= \mathcal{O}\left(N_{in}K_1 + N_{hid}M^2 + N_{hid}K_{out}\right) \\ &= \mathcal{O}\left(N_{hid}M^2\right). \end{aligned}$$

$\square$

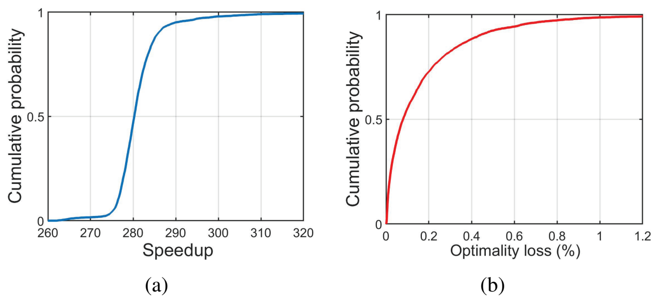


Fig. 6: Empirical cumulative distribution of speedup and optimality loss for the IEEE Case118 under typical operating conditions.

APPENDIX F
THE STATISTICAL RESULT OF THE SPEEDUP AND THE OPTIMALITY LOSS FOR THE IEEE CASE118.

We plot the empirical cumulative distribution of the speedup and the optimality loss for the IEEE Case118 in Fig. 6(a) and Fig. 6(b), respectively. As compared to the optimal solution obtained by the Gurobi solver, DeepOPF achieves an average optimality loss less than 0.2% with the maximum around 1.2%. Meanwhile, as compared to the solving time used by the Gurobi solver, DeepOPF achieves an average speedup of $\times 281$ with the maximum around $\times 320$.