ENSURING DNN SOLUTION FEASIBILITY FOR OPTIMIZATION PROBLEMS WITH LINEAR CONSTRAINTS

Tianyu Zhao[1,2], Xiang Pan[2], Minghua Chen[3,*], Steven H. Low[4]
[1]Lenovo Machine Intelligence Center, [2]The Chinese University of Hong Kong,
[3]City University of Hong Kong, [4]California Institute of Technology
tzhao3@lenovo.com, Xpan@link.cuhk.edu.hk,
minghua.chen@cityu.edu.hk (*Corresponding author), slow@caltech.edu

ABSTRACT

We propose *preventive learning* as the first framework to guarantee Deep Neural Network (DNN) solution feasibility for optimization problems with linear constraints without post-processing, upon satisfying a mild condition on constraint calibration. Without loss of generality, we focus on problems with only inequality constraints. We systematically calibrate the inequality constraints used in training, thereby anticipating DNN prediction errors and ensuring the obtained solutions remain feasible. We characterize the calibration rate and a critical DNN size, based on which we can directly construct a DNN with provable solution feasibility guarantee. We further propose an *Adversarial-Sample Aware* training algorithm to improve its optimality performance. We apply the framework to develop DeepOPF+ for solving essential DC optimal power flow problems in grid operation. Simulation results over IEEE test cases show that it outperforms existing strong DNN baselines in ensuring 100% feasibility and attaining consistent optimality loss ($<$0.19%) and speedup (up to $\times$228) in both light-load and heavy-load regimes, as compared to a state-of-the-art solver. We also apply our framework to a non-convex problem and show its performance advantage over existing schemes.

1 INTRODUCTION

Recently, there have been increasing interests in employing neural networks, including deep neural networks (DNN), to solve constrained optimization problems in various problem domains, especially those needed to be solved repeatedly in real-time. The idea behind these machine learning approaches is to leverage the universal approximation capability of DNNs (Hornik et al., 1989; Leshno et al., 1993) to learn the mapping between the input parameters to the solution of an optimization problem. Then one can directly pass the input parameters through the trained DNN to obtain a quality solution much faster than iterative solvers. For example, researchers have developed DNN schemes to solve essential optimal power flow problems in grid operation with sub-percentage optimality loss and several orders of magnitude speedup as compared to conventional solvers (Pan et al., 2020a;b; Donti et al., 2021; Chatzos et al., 2020; Lei et al., 2020). Similarly, DNN schemes also obtain desirable results for real-time power control and beam-forming design (Sun et al., 2018; Xia et al., 2019) problems in wireless communication in a fraction of time used by existing solvers.

Despite these promising results, however, a major criticism of DNN and machine learning schemes is that they usually cannot guarantee the solution feasibility with respect to all the inequality and equality constraints of the optimization problem (Zhao et al., 2020; Pan et al., 2020b). This is due to the inherent neural network prediction errors. Existing works address the feasibility concern mainly by incorporating the constraints violation (e.g., a Lagrangian relaxation to compute constraint violation with Lagrangian multipliers) into the loss function to guide THE DNN training. These endeavors, while generating great insights to the DNN design and working to some extent in case studies, can not guarantee the solution feasibility without resorting to expensive post-processing procedures, e.g., feeding the DNN solution as a warm start point into an iterative solver to obtain a feasible solution. See Sec. 2 for more discussions. To date, it remains a largely open issue of ensuring DNN solution (output of DNN) feasibility for constrained optimization problems.

In this paper, we address this challenge for general Optimization Problems with Linear (inequality) Constraints (OPLC) with varying problem inputs and fixed objective/constraints parameters. Since linear equality constraints can be exploited to reduce the number of decision variables without losing optimality (and removed), it suffices to focus on problems with inequality constraints. Our idea is to train DNN in a preventive manner to ensure the resulting solutions remain feasible even with prediction errors, thus avoiding the need of post-processing. We make the following contributions:

▷ After formulating OPLC in Sec. 3, we propose *preventive learning* as the first framework to ensure the DNN solution feasibility for OPLC without post-processing in Sec. 4. We systematically calibrate inequality constraints used in DNN training, thereby anticipating prediction errors and ensuring the resulting DNN solutions (outputs of the DNN) remain feasible.

▷ We characterize the calibration rate allowed in Sec. 4.1, i.e., the rate of adjusting (reducing) constraints limits that represents the room for (prediction) errors without violating constraints, and a sufficient DNN size for ensuring DNN solution feasibility in Sec. 4.2. We then directly construct a DNN with provably guaranteed solution feasibility.

▷ Observing the feasibility-guaranteed DNN may not achieve strong optimality result, in Sec. 4.3, we propose an adversarial training algorithm, called *Adversarial-Sample Aware* algorithm to further improve its optimality without sacrificing feasibility guarantee and derive its performance guarantee.

▷ We apply the framework to design a DNN scheme, DeepOPF+, to solve DC optimal power flow (DC-OPF) problems in grid operation. Simulation results over IEEE 30/118/300-bus test cases show that it outperforms existing strong DNN baselines in ensuring 100% feasibility and attaining consistent optimality loss ($<$0.19%) and speedup (up to $\times$228) in both light-load and heavy-load regimes, as compared to a state-of-the-art solver. We also apply our framework to a non-convex problem and show its performance advantage over existing schemes.

2 Related Work

There have been active studies in employing machine learning models, including DNNs, to solve constrained optimizations directly (Kotary et al., 2021b; Pan et al., 2019; 2020b; Zhou et al., 2022; Guha et al., 2019; Zamzam & Baker, 2020; Fioretto et al., 2020; Dobbe et al., 2019; Sanseverino et al., 2016; Elmachtoub & Grigas, 2022; Huang et al., 2021; Huang & Chen, 2021), obtaining close-to-optimal solution much faster than conventional iterative solvers. However, these schemes usually cannot guarantee solution feasibility w.r.t. constraints due to inherent prediction errors.

Some existing works tackle the feasibility concern by incorporating the constraints violation in DNN training (Pan et al., 2020a; Donti et al., 2021). In (Nellikkath & Chatzivasileiadis, 2021; 2022), physics-informed neural networks are applied to predict solutions while incorporating the KKT conditions of optimizations during training. These approaches, while attaining insightful performance in case studies, do not provide solution feasibility guarantee and may resort to expensive projection procedure (Pan et al., 2020b) or post-processing equivalent projection layers (Amos & Kolter, 2017; Agrawal et al., 2019) to recover feasibility. A gradient-based violation correction is proposed in (Donti et al., 2021). Though a feasible solution can be recovered for linear constraints, it can be computationally inefficient and may not converge for general optimizations. A DNN scheme applying gauge function that maps a point in an l_1-norm unit ball to the (sub)-optimal solution is proposed in (Li et al., 2022). However, its feasibility enforcement is achieved from a computationally expensive interior-point finder program. There is also a line of work (Ferrari, 2009; ul Abdeen et al., 2022; Qin et al., 2019; Limanond & Si, 1998) focusing on verifying whether the output of a given DNN satisfies a set of requirements/constraints. However, these approaches are only used for evaluation and not capable of obtaining a DNN with feasibility-guarantee and strong optimality. To our best knowledge, this work is the *first* to guarantee DNN solution feasibility without post-processing.

Some techniques used in our study (for constrained problems) are related to those for verifying DNN accuracy against input perturbations for unconstrained classification (Sheikholeslami et al., 2020). Our work also significantly differs from (Zhao et al., 2020) in we can provably guarantee DNN solution feasibility for OPLC and develop a new learning algorithm to improve solution optimality.

3 Optimization Problems with Linear Constraints (OPLC)

We focus on the standard OPLC formulated as (Faísca et al., 2007):

$$\min\ f(\boldsymbol{x},\boldsymbol{\theta}) \quad \text{s.t.} \quad g_j(\boldsymbol{x},\boldsymbol{\theta}) \triangleq \boldsymbol{a_j^T x} + \boldsymbol{b_j^T \theta} \le e_j,\ j \in \mathcal{E}, \tag{1}$$

$$\text{var.} \quad \underline{x}_k \le x_k \le \bar{x}_k,\ k = 1, \dots, N. \tag{2}$$

$\boldsymbol{x} \in \mathcal{R}^N$ are the decision variables, $\mathcal{E}$ is the set of inequality constraints, and $\boldsymbol{\theta} \in \mathcal{D}$ are the OPLC inputs. Convex polytope $\mathcal{D} = \{\boldsymbol{\theta} \in \mathcal{R}^M | \mathbf{A}_{\boldsymbol{\theta}}\boldsymbol{\theta} \le \boldsymbol{b}_{\boldsymbol{\theta}}, \exists \mathbf{x} : (1), (2) \text{ hold}\}$ is specified by matrix $\mathbf{A}_{\boldsymbol{\theta}}$ and vector $\boldsymbol{b}_{\boldsymbol{\theta}}$ such that $\forall \boldsymbol{\theta} \in \mathcal{D}$, OPLC in (1)-(2) admits a unique optimum.[1] The OPLC objective f is a general convex/non-convex function. For ease of presentation, we use $g_j(\boldsymbol{x}, \boldsymbol{\theta})$ to denote

[1]Our approach is also applicable to non-unique solution and unbounded $\boldsymbol{x}$. See Appendix A for a discussion.

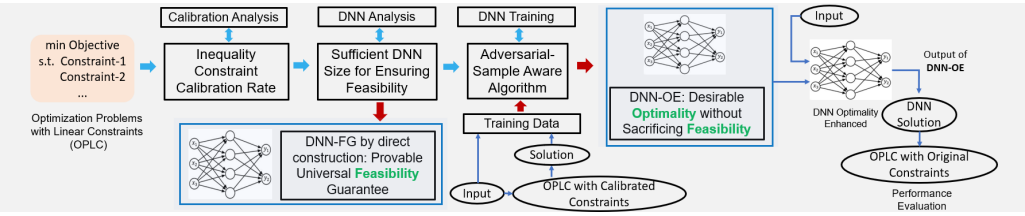


Figure 1: Overview of the preventive learning framework to solve OPLC. The calibration rate is first obtained. The sufficient DNN size in ensuring universal feasibility is then determined, and a DNN model can be constructed directly with universal feasibility guarantee in this step. With the determined calibration rate and sufficient DNN size, a DNN model with enhanced optimality without sacrificing feasibility is obtained using the *Adversarial-Sample Aware* algorithm.

$\boldsymbol{a}_j^T\boldsymbol{x} + \boldsymbol{b}_j^T\boldsymbol{\theta}$.[2] We assume that each $g_j(\boldsymbol{x}, \boldsymbol{\theta}) \leq e_j$ is active for at least one combination of $\boldsymbol{\theta} \in \mathcal{D}$ and $\boldsymbol{x}$ satisfying (2) without loss of generality (otherwise g_j is unnecessary and can be removed). We note that linear equality constraints can be exploited (and removed) to reduce the number of decision variables without losing optimality as discussed in Appendix B, it suffices to focus on OPLC with inequality constraints as formulated in (1)-(2).

The OPLC in (1)-(2) has wide applications in various engineering domains, e.g., DC-OPF problems in power systems (Pan et al., 2019) and model-predictive control problems in control systems (Bemporad et al., 2000). While many numerical solvers, e.g., those based on interior-point methods (Ye & Tse, 1989), can be applied to obtain its solution, the time complexity can be significant and limits their practical applications especially considering the problem input uncertainty under various scenarios. The observation that opens the door for DNN scheme development lies in that solving OPLC is equivalent to learning the mapping from input $\boldsymbol{\theta}$ to optimal solution $\boldsymbol{x}^*(\boldsymbol{\theta})$ (Pan et al., 2020a; Bemporad & Filippi, 2006). Thus, one can leverage the universal approximation capability of DNNs (Hornik et al., 1989; Leshno et al., 1993) to learn the input-solution mapping and apply the trained DNN to obtain the optimal solution for any $\boldsymbol{\theta} \in \mathcal{D}$ with significantly lower time complexity. See a concrete example in (Pan et al., 2020b). While DNN schemes achieve promising speedup and optimality performance, a fundamental challenge lies in ensuring solution feasibility, which is nontrivial due to inherent DNN prediction errors. In the following, we propose preventive learning as the first framework to tackle this issue and design DNN schemes for solving OPLC in (1)-(2).

4 Preventive Learning for Solving OPLC

We propose a preventive learning framework to develop DNN schemes for solving OPLC in (1)-(2). We calibrate inequality constraints used in DNN training, thereby anticipating prediction errors and ensuring the resulting DNN solutions remain feasible. See Fig. 2 for illustrations.

First, in Sec. 4.1, we determine the maximum calibration rate for inequality constraints, so that solutions from a preventively-trained DNN using the calibrated constraints respect the original constraints for all possible inputs. Here we refer the output of the DNN as the DNN solution.

Second, in Sec. 4.2, we determine a sufficient DNN size so that with preventive learning there exists a DNN whose worst-case violation on calibrated constraints is smaller than the maximum calibration rate, thus ensuring DNN solution feasibility, i.e., DNN's output always satisfies (1)-(2) for any input. We construct a provable feasibility-guaranteed DNN model, namely DNN-FG, as shown in Fig. 1.

Third, observing DNN-FG may not achieve strong optimality performance, in Sec. 4.3, we propose an adversarial *Adversarial-Sample Aware* training algorithm to further improve DNN's optimality without sacrificing feasibility guarantee, obtaining an optimality-enhanced DNN as shown in Fig. 1.

4.1 Inequality Constraint Calibration Rate

We calibrate each inequality limit $g_j(\boldsymbol{x}, \boldsymbol{\theta}) \leq e_j, j \in \mathcal{E}$ by a calibration rate $\eta_j \geq 0$:[3]

$$g_j(\boldsymbol{x}, \boldsymbol{\theta}) \leq \hat{e}_j = \begin{cases} e_j\,(1-\eta_j)\,, & \text{if } e_j \geq 0; \\ e_j\,(1+\eta_j)\,, & \text{otherwise.} \end{cases} \tag{3}$$

[2]Multiple scalars $\boldsymbol{b}_j^T\boldsymbol{\theta}, j \in \mathcal{E}$ are correlated via $\boldsymbol{\theta}$. Studying varying $\boldsymbol{a_j}, \boldsymbol{b_j}, e_j$ is also a promising future work.

[3]For g_j with $e_j = 0$, one can add an auxiliary constant $\tilde{e}_j \neq 0$ such that $g_j(\boldsymbol{x}, \boldsymbol{\theta}) + \tilde{e}_j \leq \tilde{e}_j$ for the design and formulation consistency. The choice of $\tilde{e}_j$ can be problem dependent. For example, in our simulation, $\tilde{e}_j$ is set as the maximum slack bus generation for its lower bound limit in OPF discussed in Appendix L.

However, an inappropriate calibration rate could lead to poor performance of DNN. If one adjusts the limits too much, some input $\boldsymbol{\theta} \in \mathcal{D}$ will become infeasible under the calibrated constraints and hence lead to poor generalization of the preventively-trained DNN. To this end, we solve the following bi-level optimization to obtain the maximum calibration rate, such that the calibrated feasibility set of $\boldsymbol{x}$ can still support the input region, i.e., the OPLC in (1)-(2) with a reduced feasible set has a solution for any $\boldsymbol{\theta} \in \mathcal{D}$.

$$\min_{\boldsymbol{\theta}\in\mathcal{D}} \max_{\boldsymbol{x},\nu^c} \quad \nu^c \tag{4}$$

$$\text{s.t. } (1), (2)$$

$$\nu^c \leq (e_j - g_j(\boldsymbol{\theta}, \boldsymbol{x}))/|e_j|, \ \forall j \in \mathcal{E}. \tag{5}$$

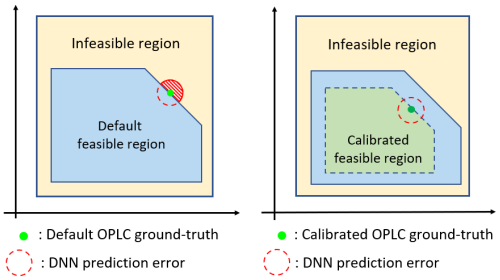


Figure 2: Left: solution of DNN trained with default OPLC ground-truth can be infeasible due to inevitable prediction errors. Right: solution of DNN trained with calibrated OPLC ground-truth ensures universal feasibility even with prediction errors.

(1)-(2) enforce the feasibility of $\boldsymbol{x}$ for input $\boldsymbol{\theta} \in \mathcal{D}$. (5) represents the maximum element-wise least redundancy among all constraints, i.e., the maximum constraint calibration rate. Therefore, solving (4)-(5) gives the maximum allowed calibration rate for all inequality constraints and $\boldsymbol{\theta} \in \mathcal{D}$.

It is challenging to solve the above bi-level problem exactly. We apply the following procedure to obtain a lower bound of its optimal objective in polynomial time. See Appendix C for details.

Step 1. Reformulate the bi-level problem (4)-(5) to an equivalent single-level one by replacing the inner problem with its KKT conditions (Boyd & Vandenberghe, 2004).

Step 2. Transform the single-level optimization problem into a MILP by replacing the bilinear equality constraints (comes from the complementary slackness in KKT conditions) with equivalent mixed-integer linear inequality constraints.

Step 3. Solve the MILP using the branch-and-bound algorithm (Lawler & Wood, 1966). Let the obtained objective value be $\Delta \geq 0$ from the primal constraint (1) and constraint (5).

Remark: (i) the branch-and-bound algorithm returns Δ (lower bound of the maximum calibration rate ν^{c*}) with a polynomial time complexity of $\mathcal{O}((M + 4|\mathcal{E}| + 5N)^{2.5})$ (Vaidya, 1989), where M and N are the dimensions of the input and decision variables, and $|\mathcal{E}|$ is the number of constraints. (ii) Δ is a lower bound to the maximum calibration rate as the algorithm may not solve the MILP exactly (by relaxing (some of) the integer variables). Such a lower bound still guarantees that the input region is supported. (iii) If $\Delta = 0$, reducing the feasibility set may lead to no feasible solution for some inputs. (iv) If $\Delta > 0$, we can obtain a DNN with provably solution feasibility guarantee as shown in Sec. 4.2. (v) After solving (4)-(5), we set each η_j in (3) to be Δ, such uniform constraints calibration forms the *outer bound* of the minimum supporting calibration region. See Appendix D for a discussion; (vi) we observe that the branch-and-bound algorithm can actually return the exact optimal ν^{c*} in less than 20 mins for numerical examples studied in Sec. 6.

4.2 Sufficient DNN Size for Ensuring Feasibility

In this subsection, we first model DNN with ReLU activations. Then, we introduce a method to determine the sufficient DNN size for guaranteeing solution feasibility.

4.2.1 DNN Model Representation.

We employ a DNN with N_{hid} hidden layers (depth) and N_{neu} neurons in each hidden layer (width), using multi-layer feed-forward neural network structure with ReLU activation function to approximate the input-solution mapping for OPLC, which is defined as:

$$\begin{aligned} &\boldsymbol{h_0} = \boldsymbol{\theta}, \, \boldsymbol{h_i} = \sigma\left(\boldsymbol{W_i h_{i-1}} + \boldsymbol{b_i}\right), \, i = 1, \ldots, N_{\text{hid}}, \\ &\tilde{\boldsymbol{h}} = \sigma\left(\boldsymbol{W_o h_{N_{\text{hid}}}} + \boldsymbol{b_o} - \underline{\boldsymbol{x}}\right) + \underline{\boldsymbol{x}}, \hat{\boldsymbol{x}} = -\sigma\left(\bar{\boldsymbol{x}} - \tilde{\boldsymbol{h}}\right) + \bar{\boldsymbol{x}}. \end{aligned} \tag{6}$$

$\boldsymbol{\theta}$ is the DNN input and $\boldsymbol{h_i}$ is the output of the i-th layer. $\boldsymbol{W_i}$ and $\boldsymbol{b_i}$ are the i-th layer's weight matrix and bias. $\sigma(x) = \max(x, 0)$ is the ReLU activation, taking element-wise max operation over the input vector. $\tilde{\boldsymbol{h}}$ enforces output feasibility w.r.t. the lower bounds in (2) while final output $\hat{\boldsymbol{x}}$ further satisfies upper bounds. Here we present the last two *clamp*-equivalent actions as (6) for further DNN

analysis. To better include the DNN equations in our designed optimization to analysis DNN's worst case feasibility guarantee performance, we adopt the technique in (Tjeng et al., 2018) to reformulate the ReLU activations expression in (6). For $i = 1, \ldots, N_{\text{hid}}$, let $\hat{\boldsymbol{h}}_i$ denotes $\boldsymbol{W_i h_{i-1}} + \boldsymbol{b_i}$. The output of neuron with ReLU activation is represented as: for $k = 1, \ldots, N_{\text{neu}}$ and $i = 1, \ldots, N_{\text{hid}}$,

$$\hat{h}_i^k \leq h_i^k \leq \hat{h}_i^k - h_i^{\min,k}(1 - z_i^k), \tag{7}$$

$$0 \leq h_i^k \leq h_i^{\max,k} z_i^k,\ z_i^k \in \{0, 1\}. \tag{8}$$

Here we use superscript k to denote the k-th element of a vector. z_i^k are (auxiliary) binary variables indicating the state of the corresponding neuron, i.e., 1 (resp. 0) indicates activated (resp. non-activated). We remark that given the value of DNN weights and bias, z_i^k can be determined (z_i^k can be either 0/1 if $\hat{h}_i^k = 0$) for each input $\boldsymbol{\theta}$. $h_i^{\max,k}/h_i^{\min,k}$ are the upper/lower bound on the neuron outputs. See Appendix E.1 for a discussion. With (7)-(8), the input-output relationship in DNN can be represented with a set of mixed-integer linear inequalities. We discuss how to employ (7)-(8) to determine the sufficient DNN size in guaranteeing universal feasibility in Sec. 4.2.2. For ease of representation, we use $(\mathbf{W}, \mathbf{b})$ to denote DNN weights and bias in the following.

4.2.2 Sufficient DNN size in Guaranteeing Universal Feasibility.

As a methodological contribution, we propose an iterative approach to determine the sufficient DNN size for guaranteeing universal solution feasibility in the input region. The idea is to iteratively verify whether the worst-case prediction error of the given DNN is within the room of error (maximum calibration rate), and doubles the DNN's width (with fixed depth) if not. We outline the design of the proposed approach below, under the setting where all hidden layers share the same width. Let the depth and (initial) width of the DNN model be N_{hid} and N_{neu}, respectively. Here we define *universal solution feasibility* as that for any input $\boldsymbol{\theta} \in \mathcal{D}$, the output of DNN always satisfies (1)-(2).

For each iteration, the proposed approach first evaluates the least maximum relative violations among all constraints for all $\boldsymbol{\theta} \in \mathcal{D}$ for the current DNN model via solving the following bi-level program:

$$\min_{\mathbf{W},\mathbf{b}} \max_{\boldsymbol{\theta} \in \mathcal{D}} \ \nu^f, \ \text{s.t.} \ (7) - (8), 1 \leq i \leq N_{\text{hid}}, 1 \leq k \leq N_{\text{neu}}, \tag{9}$$

$$\nu^f = \max_{j \in \mathcal{E}} \{(g_j(\boldsymbol{\theta}, \hat{\boldsymbol{x}}) - \hat{e}_j)/|e_j|\}, \tag{10}$$

where (9)-(10) express the outcome of the DNN as a function of input $\boldsymbol{\theta}$. ν^f is maximum constraints violation degree among all constraints. Thus, solving (9)-(10) gives the least maximum DNN constraint violation over the input region $\mathcal{D}$. We apply gradient descent to solve the above bi-level optimization problem, see Appendix E for details. Let ρ be the obtained objective value of (9)-(10) and $(\mathbf{W}^f, \mathbf{b}^f)$ be the corresponding DNN parameters, with which we can directly construct a DNN model. Recall that the determined calibration rate is Δ, the proposed approach then verifies whether the constructed DNN is sufficient for guaranteeing feasibility by the following proposition.

Proposition 1 *Consider the DNN with N_{hid} hidden layers each having N_{neu} neurons and parameters $(\mathbf{W}^f, \mathbf{b}^f)$. If $\rho \leq \Delta$, then $\forall \boldsymbol{\theta} \in \mathcal{D}$, the solution generated by this DNN is feasible w.r.t (1)-(2).*

The proof is shown in Appendix F. Proposition 1 states that if $\rho \leq \Delta$, the worst-case prediction error of current DNN model is within the maximum calibration rate and hence the current DNN size is sufficient for guaranteeing universal feasibility; otherwise, it doubles the width of DNN and moves to the next iteration. Recall that the input-solution mapping for OPLC is continuous. Hence, there exists a DNN such that universal feasibility of DNN solution is guaranteed given the DNN size (width) is sufficiently large according to the universal approximation theorem (Hornik, 1991). See Appendix G for the discussion.[4] Details of the procedures are shown in Algorithm 1. After the initialization of DNN model (line 3-4), the proposed approach repeatedly compare the obtained maximum constraints violation (ρ) with the calibration rate (Δ), doubles the DNN width (line 5-7), and return the width as N_{neu}^* until $\rho \leq \Delta$. Thus, we can construct a provable feasibility-guaranteed DNN model by the proposed approach, namely DNN-FG as shown in Fig. 1.

We remark that it is challenging to solve the bi-level problem (9)-(10) exactly, i.e., the obtained ρ is an upper bound of the optimal objective of (9)-(10) in each iteration. Nevertheless, as discussed in the following proposition, the upper bound is still useful for analyzing universal solution feasibility.

[4]One can also increase the DNN depth to achieve universal approximation for more degree of freedom in DNN parameters. In this work, we focus on increasing the DNN width for sufficient DNN learning ability.

Algorithm 1: Determining Sufficient DNN Size

1: **Input:** Δ; Initial width $N_{\text{neu}}^{\text{init}}$
2: **Output:** Determined DNN width: N_{neu}^{*}
3: Set $t = 0$; Set $N_{\text{neu}}^{t} = N_{\text{neu}}^{\text{init}}$; Obtain ρ via solving (9)-(10)
4: **while** $\rho \geq \Delta$ **do**
5: Set $N_{\text{neu}}^{t+1} = 2 \times N_{\text{neu}}^{t}$; Set $t = t + 1$; Solve (9)-(10) and update ρ
6: **end while**
7: Set $N_{\text{neu}}^{*} = N_{\text{neu}}^{t}$
8: **Return:** N_{neu}^{*}

Proposition 2 *Assume $\Delta > 0$, Algorithm 1 is guaranteed to terminate in a finite number of iterations. At each iteration t, consider the DNN with N_{hid} hidden layers each having N_{neu}^{t} neurons, we can obtain ρ as an upper bound to the optimal objective of (9)-(10) with a time complexity $\mathcal{O}((M + |\mathcal{E}| + 2N_{hid}N_{neu}^{t} + 4N)^{2.5})$. If $\rho \leq \Delta$, then the DNN with depth N_{hid} and width N_{neu}^{t} is sufficient in guaranteeing universal feasibility. Furthermore, one can construct a feasibility-guaranteed DNN with the obtained DNN parameters $(\mathbf{W}^{f}, \mathbf{b}^{f})$ such that for any $\theta \in \mathcal{D}$, the solution generated by this DNN is feasible w.r.t. (1)-(2).*

Proposition 2 indicates ρ can be obtained in polynomial time. If $\rho \leq \Delta$, it means the current DNN size is sufficient to preserve universal solution feasibility in the input region; otherwise, it means the current DNN size may not be sufficient for the purpose and it needs to double the DNN width.

We also remark that the obtained sufficient DNN size may not be the minimal sufficient one if the above bi-level optimization problem is not solved exactly. Please refer to Appendix H for detailed discussions. In our case study in Sec. 6, we observe that the evaluated initial DNN size can always guarantee universal feasibility without constraints violation, and we hence conduct further simulations with such determined sufficient DNN sizes.

4.3 Adversarial-Sample Aware Algorithm

While we can directly construct a feasibility-guaranteed DNN (without training) as shown in Proposition 2, it may not achieve strong optimality performance. To this end, we propose an *Adversarial-Sample Aware (ASA)* algorithm to further improve the optimality performance. The algorithm leverages the ideas of adversarial learning (Chakraborty et al., 2018) and active learning (Ren et al., 2021) techniques, which adaptively incorporates adversarial inputs, i.e., the inputs that cause infeasible DNN solutions, for pursuing strong optimality result while preserving universal feasibility guarantee. We outline the algorithm in the following. Denote the initial training set as $\mathcal{T}^{0}$, containing randomly-generated input and the corresponding ground-truth obtained by solving the calibrated OPLC (with calibration rate Δ). The proposed *ASA* algorithm first pre-trains a DNN model with the sufficient size determined by the approach discussed in Sec. 4.2.2, using the initial training set $\mathcal{T}^{0}$ and the following loss function $\mathcal{L}$ for each instance as the supervised learning approach:

$$\mathcal{L} = \frac{w_1}{N} \|\hat{\boldsymbol{x}} - \boldsymbol{x}^{*}\|_2^2 + \frac{w_2}{|\mathcal{E}|} \sum_{j \in \mathcal{E}} \max(g_j(\hat{\boldsymbol{x}}, \boldsymbol{\theta}) - \hat{e}_j, 0). \tag{11}$$

We leverage the penalty-based training idea in (11). The first term is the mean square error between DNN prediction $\hat{\boldsymbol{x}}$ and the ground-truth $\boldsymbol{x}^{*}$ provided by the solver for each input. The second term is the inequality constraints violation w.r.t calibrated limits $\hat{e}_j$. w_1 and w_2 are positive weighting factors to balance prediction error and penalty. Hence, training DNN by minimizing (11) can pursue a strong optimality performance as DNN prediction error is also minimized.

However, traditional penalty-based training by only minimizing (11) can not guarantee universal feasibility (Venzke et al., 2020; Pan et al., 2020b). To address this issue, the *ASA* algorithm repeatedly updates the DNN model with adversarial samples, anticipating the post-trained DNNs can eliminate violations around such inputs. Specifically, given current DNN parameters, it finds the worst-case input $\boldsymbol{\theta}^{i} \in \mathcal{D}$ by solving the inner maximization problem of (9)-(10). Let γ be the obtained objective value. Recall that the calibration rate is Δ. If $\gamma \leq \Delta$, the algorithm terminates; otherwise, it incorporates a subset of samples randomly sampled around $\boldsymbol{\theta}^{i}$ and solves the calibrated OPLC with Δ, and starts a new round of training. Details of the *ASA* algorithm are shown in Appendix I. We highlight the difference between the DNN obtained in Sec. 4.2.2 and that from *ASA* algorithm as follows. The former is directly constructed via solving (9)-(10), which guarantees universal feasibility whilst without considering optimality. In contrast, the latter is expected to enhance optimality while preserving universal feasibility as both optimality and feasibility are considered during training. We further provide theoretical guarantee of it in ensuring universal feasibility in the following.

Proposition 3 *Consider a DNN model with N_{hid} hidden layers each having N^*_{neu} neurons. For each iteration i, assume such a DNN trained with the ASA algorithm can maintain feasibility at the constructed neighborhood $\hat{\mathcal{D}}^j = \{\boldsymbol{\theta}|\boldsymbol{\theta}^j \cdot (1-a) \leq \boldsymbol{\theta} \leq \boldsymbol{\theta}^j \cdot (1+a), \boldsymbol{\theta} \in \mathcal{D}\}$ around $\boldsymbol{\theta}^j$ with some small constant $a > 0$ for $\forall j \leq i$. There exists a constant C such that the algorithm is guaranteed to ensure universal feasibility as the number of iterations is larger than C.*

The proof idea is shown in Appendix J. Proposition 3 indicates that, with the iterations is large enough, the *ASA* algorithm can ensure universal feasibility by progressively improving the DNN performance around each region around worst-case input. It provides a theoretical understanding of the justifiability of the *ASA* algorithm. In practice, we can terminate the *ASA* algorithm whenever the maximum solution violation is smaller than the inequality calibration rate, which implies universal feasibility guarantee. We note that the feasibility enforcement in the empirical/heuristic algorithm achieves strong theoretical grounding while its performance can be affected by the training method chosen. Nevertheless, as observed in the case study in Appendix M, the *ASA* algorithm terminates in at most 52 iterations with 7% calibration rate, showing its efficiency in practical application.

5 Performance Analysis of the Preventive Learning Framework

5.1 Universal Feasibility Guarantee

We provide the following proposition showing that the preventive learning framework generates two DNN models with universal feasibility guarantees.

Proposition 4 *Let Δ, ρ, and γ be the determined maximum calibration rate, the obtained objective value of (9)-(10) to determine the sufficient DNN size, and the obtained maximum relative violation of the trained DNN from Adversarial-Sample Aware algorithm following steps in preventive framework, respectively. Assume (i) $\Delta > 0$, (ii) $\rho \leq \Delta$, and (iii) $\gamma \leq \Delta$. The DNN-FG obtained from determining sufficient DNN size can provably guarantee universal feasibility and the DNN from ASA algorithm further improves optimality without sacrificing feasibility guarantee $\forall \boldsymbol{\theta} \in \mathcal{D}$.*

Proposition 4 indicates the DNN model obtained by preventive learning framework is expected to guarantee the universal solution feasibility, which is verified by simulations in Sec. 6.

5.2 Run-time Complexity

We present the complexity of the traditional method in solving the optimization problems with linear constraints. To the best of our knowledge, OPLC in its most general form is NP-hard cannot be solved in polynomial tie unless P=NP. To better deliver the results here, we consider the specific case of OPLC, namely the mp-QP problem, with linear constraints and quadratic objective function. We remark that the complexity of solving mp-QP provides a lower bower for the general OPLC problem. Under this setting, the DNN based framework has a complexity of $\mathcal{O}\left(N^2\right)$ whilst the best known iterative algorithm (Ye & Tse, 1989) requires $\mathcal{O}\left(N^4 + |\mathcal{E}|M\right)$. This means that the computational complexity of the proposed framework is lower than that of traditional algorithms. The comparison results demonstrate the efficiency of the preventive learning framework. See Appendix K for details.

5.3 Trade-off between Feasibility and Optimality

We remark that to guarantee universal feasibility, the preventive learning framework shrinks the feasible region used in preparing training data. Therefore, the DNN solution may incur a larger optimality loss due to the (sub)-optimal training data. It indicates a trade-off between optimality and feasibility, i.e., larger calibration rate leads to better feasibility but worse optimality. To further enhance DNN optimality performance, one can choose a smaller calibration rate than Δ while enlarging DNN size for better approximation ability and hence achieve satisfactory optimality and guarantee universal feasibility simultaneously.

6 Application in Solving DC-OPF and Non-Convex Optimization

6.1 DC-OPF Problem and DeepOPF+

DC-OPF is a fundamental problem for modern grid operation. It aims to determine the least-cost generator output to meet the load in a power network subject to physical and operational constraints. With the penetration of renewables and flexible load, the system operators need to handle significant uncertainty in load input during daily operation. They need to solve DC-OPF problem under many scenarios more frequently and quickly in a short interval, e.g., 1000 scenarios in 5 minutes, to obtain a stochastically optimized solution for stable and economical operations. However, iterative solvers may fail to solve a large number of DC-OPF problems for large-scale power networks fast enough for the purpose. Although recent DNN-based schemes obtain close-to-optimal solution much faster than conventional methods, they do not guarantee solution feasibility. We design DeepOPF+ by employing the preventive learning framework to tackle this issue. Consider the DC-OPF formulation:

$$\min_{\boldsymbol{P_G},\,\Phi} \sum_{i\in\mathcal{G}} c_i\left(P_{Gi}\right) \quad \text{s.t. } \boldsymbol{P_G^{\min}} \le \boldsymbol{P_G} \le \boldsymbol{P_G^{\max}}, \mathbf{M}\cdot\Phi = \boldsymbol{P_G} - \boldsymbol{P_D}, -\boldsymbol{P}_{\text{line}}^{\max} \le \mathbf{B}_{\text{line}}\cdot\Phi \le \boldsymbol{P}_{\text{line}}^{\max}, \quad (12)$$

$\boldsymbol{P_G^{\min}} \in \mathcal{R}^{|\mathcal{B}|}$ (resp. $\boldsymbol{P_G^{\max}}$) and $\boldsymbol{P}_{\text{line}}^{\max} \in \mathcal{R}^{|\mathcal{K}|}$ are the minimum (resp. maximum) generation limits of generators[5] and branch flow limits of the set of transmission lines denoted as $\mathcal{K}$. $\mathcal{G}, \mathcal{B}$, $\mathbf{M}$, $\mathbf{B}_{\text{line}}$, and $\Phi \in \mathcal{R}^{|\mathcal{B}|}$ denote the set of generators, buses, bus admittance matrix, line admittance matrix, and bus phase angles, respectively. The objective is the total generation cost and $c_i\left(\cdot\right)$ is the cost function of each generator, which is usually strictly quadratic (Park et al., 1993; tpc, 2018) from generator's heat rate curve. Constraints in (12) enforce nodal power balance equations and the limits on active power generation and branch flow. DC-OPF is hence a quadratic programming and admits a unique optimal solution w.r.t. load input $\boldsymbol{P_D}$. Analogy to OPLC (1)-(2), $\sum_{i\in\mathcal{G}} c_i\left(P_{Gi}\right)$ is the objective function f in (1). $\boldsymbol{P_D}$ is the problem input θ and $(\boldsymbol{P_G}, \Phi)$ are the decision variables $\boldsymbol{x}$.

We apply the proposed preventive-learning framework to design a DNN scheme, named DeepOPF+, for solving DC-OPF problems. We refer interested readers to Appendix L for details. Denote Δ, ρ, and γ as the obtained maximum calibration rate, the obtained objective value of (9)-(10) to determine sufficient DNN size, and the maximum relative violation of the trained DNN from *Adversarial-Sample Aware* algorithm in DeepOPF+ design, respectively. We highlight the feasibility guarantee and computational efficiency of DeepOPF+ in following proposition.

Corollary 1 *Consider the DC-OPF problem and DNN model defined in (6). Assume (i) $\Delta > 0$, (ii) $\rho \le \Delta$, and (iii) $\gamma \le \Delta$, then the DeepOPF+ generates a DNN guarantees universal feasibility for any $\boldsymbol{P_D} \in \mathcal{D}$. Suppose the DNN width is the same order of number of bus B, then DeepOPF+ has a smaller computational complexity of $\mathcal{O}\left(B^2\right)$ compared with that of state-of-the-art iterative methods $\mathcal{O}\left(B^4\right)$, where B is the number of buses.*

Corollary 1 says that DeepOPF+ can solve DC-OPF with universal feasibility guarantee at lower computational complexity compared to conventional iterative solvers,[6] as DNNs with width $\mathcal{O}(B)$ can achieve desirable feasibility/optimality. Such an assumption is validated in existing literature (Pan et al., 2019) and our simulation. To our best knowledge, DeepOPF+ is the first DNN scheme to solve DC-OPF with solution feasibility guarantee without post-processing. We remark that the DeepOPF+ design can be easily generalized to other linearized OPF models (Cain et al., 2012; Yang et al., 2018; Bolognani & Dörfler, 2015) .

6.2 Performance Evaluation over IEEE Test Cases

We evaluate its performance over IEEE 30-/118-/300- bus test cases (tpc, 2018) on the input load region of $[100\%, 130\%]$ of the default load covering both the light-load ($[100\%, 115\%]$) and heavy-load ($[115\%, 130\%]$) regimes, respectively. We conduct simulations in CentOS 7.6 with a quad-core (i7-3770@3.40G Hz) CPU and 16GB RAM. We compare DeepOPF+ with five baselines on the same training/test setting: (i) Pypower: the conventional iterative OPF solver; (ii) DNN-P: A DNN scheme adapted from (Pan et al., 2019). It learns the load-solution mapping using penalty approach without constraints calibration and incorporates a projection post processing if the DNN solution is infeasible; (iii) DNN-D: A penalty-based DNN scheme adapted from (Donti et al., 2021). It includes a correction step for infeasible solutions in training/testing; (iv) DNN-W: A hybrid method adapted from (Dong et al., 2020a). It trains a DNN to predict the primal and dual variables as the warm-start points to the conventional solver; (v) DNN-G: A gauge-function based DNN scheme adapted from (Li et al., 2022). It enforces solution feasibility by first solving a linear program to find a feasible interior point, and then constructing the mapping between DNN prediction in an l_∞ unit ball and the optimum. For better evaluation, we implement two DeepOPF+ schemes with different DNN sizes and calibration rate (3%, 7%) that are all within the maximum allowable one, namely DeepOPF+-3, and DeepOPF+-7. The detailed designs/results are presented in Appendix M.

We use the following performance metrics: (i) the percentage of the feasible solution obtained by DNN, (ii) the average relative optimality difference between the objective values obtained by DNN and Pypower, (iii) the average speedup, i.e., the average running-time ratios of Pypower to DNN-

[5] $P_{G_i} = P_{G_i}^{\min} = P_{G_i}^{\max} = 0, \forall i \notin \mathcal{G}$, and $P_{D_i} = 0, \forall i \notin \mathcal{A}$, where $\mathcal{A}$ denotes the set of load buses.

[6] We remark that the training of DNN is conducted offline; thus, its complexity is minor as amortized over many DC-OPF instances, e.g., 1000 scenarios per 5 mins. Meanwhile, the extra cost to solve the new-introduced programs in our design is also minor observing that existing solvers like Gurobi can solve the problems efficiently, e.g., <20 minutes to solve th MILPs to determine calibration rate and DNN size. Thus, we consider the run-time complexity of the DNN scheme, which is widely used in existing studies.

Table 1: Performance comparison with SOTA DNN schemes in light-load and heavy-load regimes.

Case	Scheme	Average speedups		Feasibility rate (%)		Optimality loss (%)		Worst-case violation (%)	
		light-load	heavy-load	light-load	heavy-load	light-load	heavy-load	light-load	heavy-load
Case30	DNN-P	×85	×86	100	88.12	0.02	0.03	0	5.43
	DNN-D	×85	×84	100	93.36	0.02	0.03	0	11.19
	DNN-W	×0.90	×0.86	100	100	0	0	0	0
	DNN-G	×24	×26	100	100	0.13	0.04	0	0
	DeepOPF+-3	×86	×92	100	100	0.03	0.04	0	0
	DeepOPF+-7	×86	×93	100	100	0.03	0.09	0	0
Case118	DNN-P	×137	×125	68.84	54.92	0.17	0.21	19.5	44.8
	DNN-D	×138	×124	73.42	55.37	0.20	0.24	16.69	43.1
	DNN-W	×2.08	×2.26	100	100	0	0	0	0
	DNN-G	×26	×16	100	100	1.29	0.39	0	0
	DeepOPF+-3	×201	×226	100	100	0.18	0.19	0	0
	DeepOPF+-7	×202	×228	100	100	0.37	0.41	0	0
Case300	DNN-P	×115	×98	91.29	78.42	0.06	0.08	261.1	443.0
	DNN-D	×115	×102	91.99	82.92	0.07	0.07	231.6	348.1
	DNN-W	×1.04	×1.08	100	100	0	0	0	0
	DNN-G	×2.44	×2.65	100	100	0.32	0.06	0	0
	DeepOPF+-3	×129	×136	100	100	0.03	0.03	0	0
	DeepOPF+-7	×130	×138	100	100	0.10	0.06	0	0

* Feasibility rate and Worst-case violation are the results *before* post-processing. Feasibility rates (resp Worst-case violation) after post-processing are 100% (resp 0) for all DNN schemes. We hence report the results before post-processing to better show the advantage of our design. Speedup and Optimality loss are the results *after* post-processing of the final obtained feasible solutions.
* The *correction* step in DNN-D (with 10^{-3} rate) takes longer time compared with l_1-projection in DNN-P, resulting in lower speedups.
* We empirically observe that DNN-G requires more training epochs for satisfactory performance. We report its best results at 500 epochs for Case118/300 in heavy-load and the results at 400 epochs for the other cases. The training epochs for the other DNN schemes are 200.

based approach for the test instances, respectively. (iv) the worst-case violation rate, i.e., the largest constraints violation rate of DNN solutions in the entire load domain.

6.2.1 Performance Comparisons between DeepOPF+ and Existing DNN Schemes.

The results are shown in Table 1 with the following observations. First, DeepOPF+ improves over DNN-P/DNN-D in that it achieves consistent speedups in both light-load and heavy-load regimes. DNN-P/DNN-D achieves a lower speedup in the heavy-load regime than in the light-load regime as a large percentage of its solutions are infeasible, and it needs to involve a post-processing procedure to recover the feasible solutions. Note that though DNN-P/DNN-D may perform well on the test set in light-load regime with a higher feasibility rate, its worst-case performance over the entire input domain can be significant, e.g., more than 443% constraints violation for Case300 in the heavy-load region. In contrast, DeepOPF+ guarantees solution feasibility in both light-load and heavy-load regimes, eliminating the need for post-processing and hence achieving consistent speedups. Second, though the warm-start/interior point based scheme DNN-W/DNN-G ensures the feasibility of obtained solutions, they suffer low speedups/large optimality loss. As compared, DeepOPF+ achieves noticeably better speedups as avoiding the iterations in conventional solvers. Third, the optimality loss of DeepOPF+ is minor and comparable with these of the existing state-of-the-art DNN schemes, indicating the effectiveness of the proposed *Adversarial-Sample Aware* training algorithm. Fourth, we observe that the optimality loss of DeepOPF+ increases with a larger calibration rate, which is consistent with the trade-off between optimality and calibration rate discussed in Sec. 5.3. We remark that DC-OPF is an approximation to the original non-convex non-linear AC-OPF in power grid operation under several simplifications. DC-OPF is widely used for its convexity and scalability. Expanding the work to AC-OPF is a promising future work as discussed in Appendix B.

Moreover, we apply our framework to a non-convex problem in (Donti et al., 2021) and show its performance advantage over existing schemes. Detailed design/results are shown in Appendix N.

7 Concluding Remarks

We propose preventive learning as the first framework to develop DNN schemes to solve OPLC with solution feasibility guarantee. Given a sufficiently large DNN, we calibrate inequality constraints used in training, thereby anticipating DNN prediction errors and ensuring the obtained solutions remain feasible. We propose an *Adversarial-Sample Aware* training algorithm to improve DNN's optimality. We apply the framework to develop DeepOPF+ to solve and DC-OPF problems in grid operation. Simulations show that it outperforms existing strong DNN baselines in ensuring feasibility and attaining consistent optimality loss and speedup in both light-load and heavy-load regimes. We also apply our framework to a non-convex problem and show its performance advantage over existing schemes. We remark that the proposed scheme can work for large-scale systems because of the desirable scalability of DNN. Future directions include extending the framework to general non-linear constrained optimization problems like ACOPF and evaluating its performance over systems with several thousand buses and realistic loads as discussed in Appendix B.

ACKNOWLEDGEMENTS

The work presented in this paper was supported in part by a General Research Fund from Research Grants Council, Hong Kong (Project No. 11203122), an InnoHK initiative, The Government of the HKSAR, and Laboratory for AI-Powered Financial Technologies. We thank Dr. Andreas Venzke for his contributions to some initial results and the discussions related to the fully-developed results presented in the paper. We would also like to thank the anonymous reviewers and program committee of ICLR 2023 for giving insightful comments on this article.

REFERENCES

Power Systems Test Case Archive, 2018. `http://labs.ece.uw.edu/pstca/`.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32:9562–9574, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145, 2017.

Alberto Bemporad and Carlo Filippi. An algorithm for approximate multiparametric convex programming. *Computational optimization and applications*, 35(1):87–108, 2006.

Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit solution of model predictive control via multiparametric quadratic programming. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*, volume 2, pp. 872–876. IEEE, 2000.

Saverio Bolognani and Florian Dörfler. Fast power system analysis via implicit linearization of the power flow manifold. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 402–409. IEEE, 2015.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Mary B Cain, Richard P O'neill, and Anya Castillo. History of optimal power flow and formulations. *Federal Energy Regulatory Commission*, 1:1–36, 2012.

Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

Spyros Chatzivasileiadis. Optimization in modern power systems. *Lecture Notes. Tech. Univ. of Denmark. Available online: https://arxiv. org/pdf/1811.00943. pdf*, 2018.

Minas Chatzos, Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. High-fidelity machine learning approximations of large-scale optimal power flow. *arXiv preprint arXiv:2006.16356*, 2020.

John M Danskin. *The theory of max-min and its application to weapons allocation problems*, volume 5. Springer Science & Business Media, 2012.

Roel Dobbe, Oscar Sondermeijer, David Fridovich-Keil, Daniel Arnold, Duncan Callaway, and Claire Tomlin. Toward distributed energy services: Decentralizing optimal power flow with machine learning. *IEEE Transactions on Smart Grid*, 11(2):1296–1306, 2019.

Wenqian Dong, Zhen Xie, Gokcen Kestor, and Dong Li. Smart-pgsim: using neural network to accelerate AC-OPF power grid simulation. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2020a.

Yinpeng Dong, Zhijie Deng, Tianyu Pang, Jun Zhu, and Hang Su. Adversarial distributional training for robust deep learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 8270–8283, 2020b.

Priya L. Donti, David Rolnick, and J Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1): 9–26, 2022.

Nuno P Faísca, Vivek Dua, and Efstratios N Pistikopoulos. Multiparametric linear and quadratic programming, 2007.

Silvia Ferrari. Multiobjective algebraic synthesis of neural control systems by implicit model following. *IEEE Transactions on Neural Networks*, 20(3):406–419, 2009.

Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. Predicting AC optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 630–637, 2020.

José Fortuny-Amat and Bruce McCarl. A representation and economic interpretation of a two-level programming problem. *Journal of the operational Research Society*, 32(9):783–792, 1981.

Neel Guha, Zhecheng Wang, Matt Wytock, and Arun Majumdar. Machine learning for AC optimal power flow. *arXiv preprint arXiv:1910.08842*, 2019.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2):251–257, 1991.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Wanjun Huang and Minghua Chen. DeepOPF-NGT: A fast unsupervised learning approach for solving AC-OPF problems without ground truth. In *ICML 2021 Workshop on Tackling Climate Change with Machine Learning*, 2021.

Wanjun Huang, Xiang Pan, Minghua Chen, and Steven H Low. Deepopf-V: Solving AC-OPF problems efficiently. *IEEE Transactions on Power Systems*, 37(1):800–803, 2021.

James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. Learning hard optimization problems: A data generation perspective. *Advances in Neural Information Processing Systems*, 34: 24981–24992, 2021a.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, 2021b.

Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

Xingyu Lei, Zhifang Yang, Juan Yu, Junbo Zhao, Qian Gao, and Hongxin Yu. Data-driven optimal power flow: A physics-informed machine learning approach. *IEEE Transactions on Power Systems*, 36(1):346–354, 2020.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Meiyi Li, Soheil Kolouri, and Javad Mohammadi. Learning to solve optimization problems with hard linear constraints. *arXiv preprint arXiv:2208.10611*, 2022.

Suttipan Limanond and Jennie Si. Neural network-based control design: An LMI approach. *IEEE Transactions on Neural Networks*, 9(6):1422–1429, 1998.

Rahul Nellikkath and Spyros Chatzivasileiadis. Physics-informed neural networks for minimising worst-case violations in DC optimal power flow. In *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 419–424. IEEE, 2021.

Rahul Nellikkath and Spyros Chatzivasileiadis. Physics-informed neural networks for AC optimal power flow. *Electric Power Systems Research*, 212:108412, 2022.

Xiang Pan, Tianyu Zhao, and Minghua 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)*, 2019.

Xiang Pan, Minghua Chen, Tianyu Zhao, and Steven H Low. DeepOPF: A Feasibility-Optimized Deep Neural Network Approach for AC Optimal Power Flow Problems. *arXiv preprint arXiv:2007.01002*, 2020a.

Xiang Pan, Tianyu Zhao, Minghua Chen, and Shengyu Zhang. DeepOPF: A deep neural network approach for security-constrained DC optimal power flow. *IEEE Transactions on Power Systems*, 36(3):1725–1735, 2020b.

Xiang Pan, Wanjun Huang, Minghua Chen, and Steven H Low. Deepopf-AL: Augmented learning for solving AC-OPF problems with multiple load-solution mappings. In *Proceedings of the Fourteenth ACM International Conference on Future Energy Systems (ACM e-Energy 2023)*, 2023.

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*, 8(3):1030–1038, Aug 1993. ISSN 0885-8950.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12 (1):145–151, 1999.

Chongli Qin, Krishnamurthy (Dj) Dvijotham, Brendan O'Donoghue, Rudy Bunel, Robert Stanforth, Sven Gowal, Jonathan Uesato, Grzegorz Swirszcz, and Pushmeet Kohli. Verification of non-linear specifications for neural networks. In *International Conference on Learning Representations*, 2019.

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *ACM computing surveys (CSUR)*, 54(9):1–40, 2021.

E Riva Sanseverino, ML Di Silvestre, L Mineo, S Favuzza, NQ Nguyen, and QTT Tran. A multi-agent system reinforcement learning based optimal power flow for islanded microgrids. In *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, 2016.

Fatemeh Sheikholeslami, Ali Lotfi, and J Zico Kolter. Provably robust classification of adversarial examples with detection. In *International Conference on Learning Representations*, 2020.

Haoran Sun, Xiangyi Chen, Qingjiang Shi, Mingyi Hong, Xiao Fu, and Nicholas D Sidiropoulos. Learning to optimize: Training deep neural networks for interference management. *IEEE Transactions on Signal Processing*, 66(20):5438–5453, 2018.

Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2018.

Zain ul Abdeen, He Yin, Vassilis Kekatos, and Ming Jin. Learning neural networks under input-output specifications. In *2022 American Control Conference (ACC)*, pp. 1515–1520. IEEE, 2022.

Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *IEEE FOCS*, pp. 332–337, 1989.

Andreas Venzke, Guannan Qu, Steven Low, and Spyros Chatzivasileiadis. Learning optimal power flow: Worst-case guarantees for neural networks. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020.

Wenchao Xia, Gan Zheng, Yongxu Zhu, Jun Zhang, Jiangzhou Wang, and Athina P Petropulu. A deep learning framework for optimization of miso downlink beamforming. *IEEE Transactions on Communications*, 68(3):1866–1880, 2019.

Zhifang Yang, Kaigui Xie, Juan Yu, Haiwang Zhong, Ning Zhang, and Qing Xia. A general formulation of linear power flow models: Basic theory and error analysis. *IEEE Transactions on Power Systems*, 34(2):1315–1324, 2018.

Yinyu Ye and Edison Tse. An extension of Karmarkar's projective algorithm for convex quadratic programming. *Mathematical Programming*, 44(1):157–179, May 1989.

Ahmed S Zamzam and Kyri Baker. Learning optimal solutions for extremely fast AC optimal power flow. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020.

Tianyu Zhao, Xiang Pan, Minghua Chen, Andreas Venzke, and Steven H Low. DeepOPF+: A deep neural network approach for DC optimal power flow for ensuring feasibility. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020.

Min Zhou, Minghua Chen, and Steven H Low. DeepOPF-FT: One deep neural network for multiple AC-OPF problems with flexible topology. *IEEE Transactions on Power Systems*, 38(1):964–967, 2022.

Supplementary Material

A Analytical formulation of $\mathcal{D}$, uniqueness of the OPLC solution, and unbounded variable

Set $\mathcal{D}$ is of problem dependent. For example, in DC-OPF problems, $\mathcal{D}$ represents the interested load input domain which is set by the system operator, e.g., feasible load within [100%, 130%] of the default load. For others applications, $\mathcal{D}$ represents region of *possible* feasible problem inputs. Calculating the analytical representation of the feasible region of θ is known as projection of a polyhedral set to lower dimension subspace. That is, $\mathcal{D}$ can be analytically obtained by projecting the following set

$$\mathcal{P} = \{(\theta, \mathbf{x}) | \mathbf{A}_{\boldsymbol{\theta}}\boldsymbol{\theta} \leq \boldsymbol{b}_{\boldsymbol{\theta}}, \text{and} (1), (2) \text{ hold}\}$$

onto the subspace of θ, which is still a convex polytope. The goal can be achieve using the Fourier–Motzkin elimination technique. Nevertheless, in our design, we do not need to access the full analytical formulation of $\mathcal{D}$. Instead, we introduce a set of auxiliary variable $\tilde{\mathbf{x}}$ associated with each θ. That is, the constraint $\theta \in \mathcal{D}$ is indeed represented as $\{\mathbf{A}_{\boldsymbol{\theta}}\boldsymbol{\theta} \leq \boldsymbol{b}_{\boldsymbol{\theta}}, g_j(\tilde{\mathbf{x}}, \theta) \leq e_j, \forall j \in \mathcal{E}\}$.

A.1 Uniqueness of the OPLC solution

We would like to further discuss the assumption of the uniqueness of the OPLC solution. First, many OPLC are unique given their objective functions are strictly convex. Such a condition holds for DC-OPF problems in power systems (Pan et al., 2019) and model-predictive control problems in control systems (Bemporad et al., 2000). As proved in (Pan et al., 2020a), if the optimal solution is unique, the input-solution mapping is continuous while the DNN function is also continuous, which forms the underlying reason why DNN can be applied to learning such a mapping from the Universal Approximation Theorem of DNN for continuous functions.

We would like to further discuss the situation if the optimal solution is not unique, which is an open problem and the challenge of the existing end-to-end DNN design.

Given a OPLC that admits multiple optimal solutions for the input, there indeed does not exist an injective mapping between input to solution, i.e., there exist multiple input-solution mappings. Consider the DNN training in this case, if the ground-truth training data are from different input-solution mappings, the DNN could present unsatisfactory performance as solutions to closely related instances may exhibit large differences and the learning task can become inherently more difficult (Kotary et al., 2021a; Huang & Chen, 2021; Pan et al., 2023). Nevertheless, our approach is still applicable to such a scenario as the first obtained DNN-FG after determining the sufficient DNN size can still guarantee universal feasibility. As introduced in Sec. 4.1 and Sec. 4.2, deriving the calibration rate and determining the sufficient DNN size is only related to the OPLC constraints. These steps only require obtaining one of the continuous feasible mappings but not optimality. Towards the Adversarial-Sample Aware algorithm, it is straightforward to adopt the approaches in (Kotary et al., 2021a; Huang & Chen, 2021; Pan et al., 2023) by improving the training data quality, applying the unsupervised learning idea, or learning the high-dimensional input+initial point to optimal solution mapping, which we leave for future work. Finally, the simulations on non-convex optimization (can have non-unique optimum) in Appendix N show that the ASA algorithm can still work well, showing the robustness of the design.

A.2 Unbounded decision variables

There are two approaches to handle the unbounded variables: 1) setting $\underline{x}_i$ or $\bar{x}_i$ to be some arbitrarily small/large numbers. 2) only includes the bounded constraints into (4)-(5) and (6), e.g., for the variables 1) without lower bound, the DNN output is $\hat{\boldsymbol{x}}_i = -\sigma\left(\bar{\boldsymbol{x}}_i - (\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_{\boldsymbol{N}_{\text{hid}}} + \boldsymbol{b}_{\boldsymbol{o}})_i\right) + \bar{\boldsymbol{x}}_i$; 2) without upper bound $\hat{\boldsymbol{x}}_i = \tilde{\boldsymbol{h}}_i$; 3) without both upper and lower bound, $\hat{\boldsymbol{x}}_i = (\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_{\boldsymbol{N}_{\text{hid}}} + \boldsymbol{b}_{\boldsymbol{o}})_i$.

B Handling Equality and Non-linear Constraints

We remark that for general OPLC and other constrained optimizations, we can always removing the equality constraints explicitly/implicitly. Given $N + p$ variables and p (linear) equality con-

straints, we can remove these equalities and representing p variables by the remaining N variables using the equality constraints, e.g., applying the coefficient matrix inversion as discussed in Appendix L without losing optimality. We thus focus on OPLC with inequality constraints only. The similar predict-and-reconstruct idea is proposed in (Pan et al., 2019; Donti et al., 2021). In addition, we note that the proposed *preventive leaning* framework is also applicable to non-linear inequality constraints, e.g., AC-OPF problems with several thousand buses, but with additional computational challenges in solving the related programs corresponding to the required steps. We leave the application to optimization with non-linear constraints and non-convex objective with large DNN size for future study.

In this work, we consider the variation of the RHS of the linear inequality constraints. It is also interesting to study the varying a_j, b_j, e_j. We believe our approach is still applicable to such a case while may have additional computational challenges as the problem turn to be non-linearly constrained. Nevertheless, it is also great interest to study problems whose parameters are not varying. For example, in DC-OPF, a_j, b_j, e_j are determined by power network topology, which will not change significantly over a long time scale, e.g., months to years. Hence, it is reasonable and practical to study OPLC with varying inputs only.

C Mixed-Integer Reformulation of Bi-Level Linear Programs

Consider the following the linear constrained bi-level min-max problem:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{x}} \quad \boldsymbol{c}^{\boldsymbol{T}}\boldsymbol{x} \tag{13}$$

$$\text{s.t.} \quad \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} + \mathbf{F}\boldsymbol{\theta}, \tag{14}$$

$$\boldsymbol{\theta} \in \mathcal{D}, \tag{15}$$

where $\mathbf{A} \in \mathcal{R}^{p\times N}$, $\boldsymbol{b} \in \mathcal{R}^{p}$, $\mathbf{F} \in \mathcal{R}^{p\times M}$.

The above linear constrained bi-level program can be reformulated by replacing the inner maximization problem by its sufficient and necessary KKT conditions (Boyd & Vandenberghe, 2004). We present the reformulated program in the following:

$$\min_{\boldsymbol{\theta},\boldsymbol{x},\boldsymbol{y}} \quad \boldsymbol{c}^{\boldsymbol{T}}\boldsymbol{x} \tag{16}$$

$$\text{s.t.} \quad \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} + \mathbf{F}\boldsymbol{\theta}, \tag{17}$$

$$\mathbf{A}^T\boldsymbol{y} = \boldsymbol{c}, \tag{18}$$

$$y_i \geq 0,\ i = 1, \ldots, p, \tag{19}$$

$$y_i(\boldsymbol{a_i}^{\boldsymbol{T}}\boldsymbol{x} - b_i - \boldsymbol{f_i}^{\boldsymbol{T}}\boldsymbol{\theta}) = 0,\ i = 1, \ldots, p, \tag{20}$$

$$\boldsymbol{\theta} \in \mathcal{D}. \tag{21}$$

Here $\boldsymbol{a_i}$ and $\boldsymbol{f_i}$ denote the i-th row of matrix $\mathbf{A}$ and $\mathbf{F}$ respectively. We remark that the non-linear *Complementary Slackness* condition in (20) can be reformulated to be mixed-integer linear using the Fortuny-Amat McCarl linearization (Fortuny-Amat & McCarl, 1981):

$$y_i \leq (1 - r_i)C, \quad \boldsymbol{a_i}^{\boldsymbol{T}}\boldsymbol{x} - b_i - \boldsymbol{f_i}^{\boldsymbol{T}}\boldsymbol{\theta} \geq -r_iC. \tag{22}$$

Here the non-linear complementary slackness conditions are reformulated with the binary variable r_i and the large non-binding constant C for each $i = 1, \ldots, p$. Therefore, problem (16)-(21) can be reformulated to be the mixed-integer linear program (MILP).

We remark that if $\nu^{f*} = 0$, implying that the system is too binding, e.g., for DC-OPF problem, some line/generator must always be at its capacity upper bound. Such a restrictive condition seldom happen in practice for the power system safety operation. Under such a scenario, one can consider a smaller input region $\mathcal{D}$ such that the input is not so extreme and there could always exist an interior for the input region.

D Minimum Supporting Calibration Region

We remark that the obtained uniform calibration rate on each constraints forms the *outer bound* of the minimum supporting calibration region defined as follows:

Definition 1 *The minimum supporting calibration region is defined as the set of calibration rate* $\{\eta_j\}_{j\in\mathcal{E}}$ *and for each* $\boldsymbol{\theta} \in \mathcal{D}$*, there exist an* $\boldsymbol{x}$ *such that (2) and (3) hold. Meanwhile, there exist a* $\boldsymbol{\theta} \in \mathcal{D}$ *and there does not exist an* $\boldsymbol{x}$ *such that (2) and (3) hold under* $\{\eta_j + \delta_j\}_{j\in\mathcal{E}}$ *for any* $\delta_j \geq 0$ *and at least one* $\delta_j > 0$*.*

The minimum supporting calibration region describes the set of maximum calibration rate such that 1) the input parameter region is maintained, and 2) any further calibration on the constraints will lead some input to be infeasible. We remark that such minimum supporting calibration region is not unique. See the following example and the approach to obtain (one of) such minimum supporting calibration region.

We first provide a toy example to demonstrate the non-uniqueness of the minimum supporting calibration region defined in Def. 1. Consider the following modified network flow problem:

$$\begin{aligned}
\min \quad & x_1^2 + x_2^2 + x_3^2 && (23)\\
\text{s.t.} \quad & 0 \leq x_1 \leq 90, && (24)\\
& 0 \leq x_2 \leq 90, && (25)\\
& x_3 \leq 70, && (26)\\
& x_1 + x_2 \leq 90, && (27)\\
& x_2 + x_3 \leq 90, && (28)\\
& x_1 + x_2 + x_3 = l. && (29)
\end{aligned}$$

Here l is the input load within $[0, 100]$ and x_1, x_2, and x_3 can be seen as the network flow on the edges. Similar to the analysis in Sec. 4.1, the constraints (26)-(28) can be calibrated by at most 37.5% uniformly. However, such a calibration region is not the minimum one while forms the outer bound of it. Denote the calibration rate on (26)-(28) as (x, y, z), it is easy to see that any combination such that $7x + 9y = 6$ and $z = 8/9 - y$ is the minimum supporting one.

We further provide the follow procedures to determine (one of) the minimum supporting region.

- Step 1. Solve (4)-(5) to obtain the uniform maximum calibration rate Δ. Let $k = 1$.
- Step 2. For constraint g_k, solve

$$\begin{aligned}
\min_{\boldsymbol{\theta}\in\mathcal{D}} \max_{\boldsymbol{x}} \quad & \frac{\hat{e}_k - g_k(\boldsymbol{\theta}, \boldsymbol{x})}{|e_j|} && (30)\\
\text{s.t.} \quad & (2)\\
& g_j(\boldsymbol{\theta}, \boldsymbol{x}) \leq \hat{e}_j, \ \forall j \in \mathcal{E}, && (31)
\end{aligned}$$

  where $\hat{e}_k = e_k \times (1_{e_k\geq 0}(1-\Delta) + 1_{e_k<0}(1+\Delta))$. Denote the optimal value of (30)-(31) as δ_k, which represent the maximum additional individual calibration rate of constraint g_k considering all other constraints' calibrations.
- Update $\hat{e}_k$ to be $e_k \times (1_{e_k\geq 0}(1-\Delta-\delta_k) + 1_{e_k<0}(1+\Delta+\delta_k))$ and proceed to the next iteration $k+1$. Go to Step 2.

We remark that after each update of $\hat{e}_k$, the next g_{k+1} is studied on a tighter region described by $\{\hat{e}_j, j = 1, \ldots, k\}$. After solving the programs for each g_k, one can easily see that the calibration region $\{\Delta + \delta_j\}_{j\in\mathcal{E}}$ is the minimum supporting calibration region.

In this work, we consider the uniform calibration rate Δ for further analysis. We remark that the uniform calibration method may introduce the asymmetry on the calibration rate as large limit would have large calibration rate. An alternative approach is to set the individual calibration rate η_j for each constraint while maintain the supported input region as discussed above. However, the choice of such individual calibration rates is not unique due to the non-uniqueness of the minimum supporting calibration region. We leave the analysis of such individual constraints calibration for future study.

E Details of applying *Danskin's Theorem* to the Bi-level Problem to Determine the Sufficient DNN Size

We provide the details of applying *Danskin's Theorem* to solve the bi-level mined-integer non-linear problem (9)-(10).

To solve such bi-level optimization problem, we optimize the upper-level variables $(\mathbf{W}, \mathbf{b})$ by gradient descent. This would simply involve repeatedly computing the gradient w.r.t. $(\mathbf{W}, \mathbf{b})$ for the object function, and taking a step in this negative direction. That is, we want to repeat the update

$$\mathbf{W} \coloneqq \mathbf{W} - \alpha \cdot \nabla_{\mathbf{W}}(\max_{\boldsymbol{\theta}} \nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta})), \tag{32}$$

$$\mathbf{b} \coloneqq \mathbf{b} - \alpha \cdot \nabla_{\mathbf{b}}(\max_{\boldsymbol{\theta}} \nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta})). \tag{33}$$

Here $\max_{\boldsymbol{\theta}} \nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta})$ denotes the maximum violation among the calibrated inequality constraints within the entire inputs domain $\mathcal{D}$, given the specific value of DNN parameters $(\mathbf{W}, \mathbf{b})$. Note that the inner function itself contains a maximization problem. We apply the *Danskin's Theorem* to compute the gradient of the inner term. It states that the gradient of the inner function involving the maximization term is simply given by the gradient of the function evaluated at this maximum. In other words, to compute the (sub)gradient of a function containing a $\max(\cdot)$ term, we need to simply: 1) find the maximum, and 2) compute the normal gradient evaluated at this point (Dong et al., 2020b; Danskin, 2012). Hence, the relevant gradient is given by

$$\nabla_{\mathbf{W}}(\max_{\boldsymbol{\theta}} \nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta})) = \nabla_{\mathbf{W}} \nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta}^*), \tag{34}$$

$$\nabla_{\mathbf{b}}(\max_{\boldsymbol{\theta}} \nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta})) = \nabla_{\mathbf{b}} \nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta}^*), \tag{35}$$

where

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta}). \tag{36}$$

Here the optimal $\boldsymbol{\theta}^*$ depends on the choice of DNN parameters $(\mathbf{W}, \mathbf{b})$. Therefore, at each iterative update of $(\mathbf{W}, \mathbf{b})$, we need to solve the inner maximization problem once. Note that the optimal $\boldsymbol{\theta}^*$ may not be unique. However, the gradient of $\nu^f(\mathbf{W}, \mathbf{b}, \boldsymbol{\theta}^*)$ w.r.t. $(\mathbf{W}, \mathbf{b})$ can still be obtained given a specific $\boldsymbol{\theta}^*$, which is (one of the) gradient that optimizes the deep neural network. We remark that such approach is indeed widely adopted in existing literature (Dong et al., 2020b; Danskin, 2012). In addition, though the involved program is a mixed-integer linear problem, we observe that the solver can indeed provide its optimum efficiently, e.g., <20 mins for Case300 in DC-OPF problem in simulation. Nevertheless, we remark that finding a (sub-optimal) feasible solution for the inner maximization problem can be easily obtained by a heuristic trial of some particular $\boldsymbol{\theta}$, e.g., the worst-case input at the previous round as the initial point and the associate integer values in the DNN constraints (7)-(8), which are fixed given the specification of DNN parameters. Such a solution can still be utilized for the further steps to calculate the sub-gradient of the DNN. One can see the analogy between it and DNN training with stochastic gradient decent method.

In addition, note that to obtain the upper bound ρ, we do not need to access any feasible point of the inner maximization problem. The upper bound is provided by the relaxation in the branch-and-bound algorithm, e.g., relax (some) integer variables to continuous. This can be efficiently obtained by the solvers, e.g., APOPT or Gurobi. Such an upper bound is applied to verify whether universal feasibility guarantee is obtained and whether the DNN size is sufficient.

E.1 Determining the values of $h_i^{\max,k}/h_i^{\min,k}$

$h_i^{max,k}/h_i^{min,k}$ are constants and fixed during solving the (inner) MILP in optimization (9)-(10) (Tjeng et al., 2018). These numbers represent the maximum/minimum bounds on the values of the neuron outputs, which should be large/small enough numbers to let the DNN constraints not be binding in the reformulation (7)-(8). In our design, we follow the technique in (Venzke et al., 2020) to obtain such (tighter) upper/lower bounds for each updated $(\mathbf{W}, \mathbf{b})$. In particular, we minimize and maximize the output of each neuron subject to the linear relaxation of the binary variables (to be continuous within 0 and 1) in the DNN constraints with parameters $(\mathbf{W}, \mathbf{b})$ in (7)-(8) and entire input region $\mathcal{D}$. Such upper/lower bounds can be efficiently obtained by solving the LPs after relaxation, which guarantees that the neuron output will not exceed the corresponding values. We note that for different DNN parameters $(\mathbf{W}, \mathbf{b})$, $h_i^{max,k}/h_i^{min,k}$ could take different values that can always be efficiently obtained from the LPs after linear relaxation.

F PROOF OF PROPOSITION 1

Proof: Consider the DNN with N_{hid} hidden layers each having N_{neu} neurons and parameters $(\mathbf{W}^f, \mathbf{b}^f)$ and $\rho \leq \Delta$. Since ρ is an upper bound on the optimal value of the bi-level problem (9)-(10), we have

$$(g_j(\boldsymbol{\theta}, \hat{\boldsymbol{x}}) - \hat{e}_j)/|e_j| \leq \rho, \forall \boldsymbol{\theta} \in \mathcal{D}, \forall j \in \mathcal{E}. \tag{37}$$

Therefore, we have for any $\boldsymbol{\theta} \in \mathcal{D}$ and $j \in \mathcal{E}$

$$\begin{cases} g_j(\boldsymbol{\theta}, \hat{\boldsymbol{x}}) - e_j(1-\Delta) \leq \rho \cdot e_j, & \text{if } e_j \geq 0; \\ g_j(\boldsymbol{\theta}, \hat{\boldsymbol{x}}) - e_j(1+\Delta) \leq -\rho \cdot e_j, & \text{otherwise,} \end{cases} \tag{38}$$

which is equivalent to

$$\begin{cases} g_j(\boldsymbol{\theta}, \hat{\boldsymbol{x}}) \leq e_j + (\rho - \Delta) \cdot e_j, & \text{if } e_j \geq 0; \\ g_j(\boldsymbol{\theta}, \hat{\boldsymbol{x}}) \leq e_j + (\Delta - \rho) \cdot e_j, & \text{otherwise.} \end{cases} \tag{39}$$

Since $\rho \leq \Delta$, we have

$$g_j(\boldsymbol{\theta}, \hat{\boldsymbol{x}}) \leq e_j, \forall \boldsymbol{\theta} \in \mathcal{D}, \forall j \in \mathcal{E}. \tag{40}$$

This completes the proof of Proposition 1.

G UNIVERSAL APPROXIMATION CAPABILITY OF DNN

We highlight the *Universal Approximation* of DNNs for approximate the input-solution for the OPLC in the following proposition.

Proposition 5 *(Hornik, 1991) Assume the target function to learn is continuous, there always exists a DNN whose output function can approach the target function arbitrarily well, i.e.,*

$$\max_{\boldsymbol{\theta} \in \mathcal{D}} \|h(\boldsymbol{\theta}) - \hat{h}(\boldsymbol{\theta})\| < \varepsilon,$$

hold for any ϵ arbitrarily small (distance from h to $\hat{h}$ can be infinitely small). Here $h(\boldsymbol{\theta})$ and $\hat{h}(\boldsymbol{\theta})$ represent the target mapping to be approximated and the DNN function respectively.

Furthermore, given the fixed depth N_{hid} of the DNN, the learning ability of the DNN is increasing monotonically w.r.t. the width of the DNN. That is, consider two DNN width N^1_{neu} and N^2_{neu} such that $N^1_{neu} > N^2_{neu}$, we have

$$\min_{h \in \mathcal{C}^{N^1_{neu}}} \max_{\boldsymbol{\theta} \in \mathcal{D}} \|h(\boldsymbol{\theta}) - \hat{h}(\boldsymbol{\theta})\| \leq \min_{h \in \mathcal{C}^{N^2_{neu}}} \max_{\boldsymbol{\theta} \in \mathcal{D}} \|h(\boldsymbol{\theta}) - \hat{h}(\boldsymbol{\theta})\|,$$

where $\mathcal{C}^{N^1_{neu}}$ and $\mathcal{C}^{N^2_{neu}}$ denote the class of all functions generated by a N_{hid} depth neural network with width N^1_{neu} and N^2_{neu} respectively.

Proposition 5 provides as the strong observation and theoretical basis for designing the iterative approach to determine the sufficient DNN size in guaranteeing universal feasibility.

H MINIMAL SUFFICIENT DNN SIZE

We remark that the obtained sufficient DNN size by doubling the DNN width may be substantial, introducing additional training time to train the DNN model and higher computational time when applied to solve OPLC. One can also determine the corresponding minimum sufficient DNN size by a simple and efficient binary search between

- the obtained sufficient DNN size N^*_{neu} and the pre-obtained DNN size $N^*_{\text{neu}}/2$ (before doubling the DNN width) which fails to achieve universal feasibility, if the initial tested DNN can not guarantee universal feasibility;

- the initial tested DNN size and some small DNN, e.g., zero width DNN, if the initial tested DNN size is sufficient in guaranteeing universal feasibility.

Such a minimum sufficient DNN size denotes the minimum width required for a given DNN structure with depth N_{hid} to achieve universal feasibility within the entire input domain. We use $\hat{N}_{\text{neu}}$ to denote the determined minimum sufficient DNN size and propose the following proposition.

Proposition 6 *Consider the DNN with N_{hid} hidden layers each having $\hat{N}_{neu}$ neurons, any DNN with depth N_{hid} and a smaller width than $\hat{N}_{neu}$ can not guarantee universal feasibility for all input $\boldsymbol{\theta} \in \mathcal{D}$. Meanwhile, any DNN with depth N_{hid} and at least $\hat{N}_{neu}$ width can always achieve universal feasibility.*

I Description of Adversarial-Sample Aware Algorithm

We outline the Adversarial-Sample Aware algorithm in Algorithm 2. As seen, the *Adversarial Sample-Aware* algorithm pre-trains the DNN model (line 3) and starts the iteration (line 5). Each iteration verifies whether the worst-case prediction error is within the room (maximum calibration rate) (lines 6 - 13). If the maximum constraints violation of the adversarial input exceeds the determined calibrated rate, the *Adversarial-Sample Aware* algorithm incorporates a set of adversarial samples into the existing training set and updates the DNN parameters (line 14 - 19), expecting the constraints violation of the adversaries are eliminated.

We expect that after a few training epochs, the post-trained DNN can restore feasibility at the identified adversarial sample $\boldsymbol{\theta}^t$ and the points around it in $\mathcal{S}^t$. This is inspired by the observation that after adding the previously identified training pairs $\mathcal{S}^t$ into the training set, the DNN training loss is dominated by the approximation errors and the penalties at the samples in $\mathcal{S}^t$. Though the training loss may not be optimized to 0, e.g., still has violations w.r.t. the calibrated constraints limits, the DNN solution is expected to satisfy the original inequality constraints after such preventive training procedure. Therefore, the post-trained DNN is capable of preserving feasibility and good accuracy at these input regions. Simulation results in Sec. 6.2.1 show the effectiveness of the propose algorithm. We provide the following proposition to state the guarantee of the algorithm in ensuring universal feasibility.

J Proof of Proposition 3

Proof idea: Here we consider the post-trained DNN with N_{hid} hidden layers each having N^*_{neu} neurons. Given current iteration i, for $\forall j \leq i$, suppose it can always maintain feasibility at the correspondingly constructed neighborhoods around the identified worst-case input, i.e., $\hat{\mathcal{D}}^j$, by training on $\mathcal{T}^{i+1}$ that combines $\mathcal{T}^0$ and all the auxiliary subset $\mathcal{S}^j$ around the identified adversarial input $\boldsymbol{\theta}^j, \forall j \leq i$. Therefore, when the number of iterations is large enough, the union of the feasible regions $\tilde{\mathcal{D}}^{i>C} = \hat{\mathcal{D}}^1 \cup \hat{\mathcal{D}}^2 \cup \ldots \hat{\mathcal{D}}^i$ can cover the entire input domain $\mathcal{D}$. That is, the post-trained DNN can ensure feasibility for each small region $\hat{\mathcal{D}}^i$ within the input domain $\mathcal{D}$, and hence universal feasibility is guaranteed. Such observation is similar to the topic of minimum covering ball problem of the compact set in real analysis.

Such a condition generally requires the DNN to preserve feasibility within some small regions by especially including the input-solution information during training, which may not be hard to satisfy. This can be understood from the observation that the worst-case violation in the smaller inner domain can be reduced significantly by training on the broader outer input domain (Venzke et al., 2020; Nellikkath & Chatzivasileiadis, 2022) as the adversarial inputs are always element-wise at the boundary of the entire input domain $\mathcal{D}$, which echoes our simulation findings in Sec. 6. Therefore, the post-trained DNN is expected to perform good feasibility guarantee in all small regions $\hat{\mathcal{D}}^j, \forall j \leq i$ after the preventive training procedure on $\mathcal{T}^{i+1}$, the training set on the entire domain $\mathcal{D}$. We remark that after gradually including these subsets $\mathcal{S}^i$ into the existing training set, the loss function is determined by the joint loss among all samples in these regions. After the training process, the post-obtained DNN is hence expected to maintain feasibility at the points in the training set and the regions around them.

Algorithm 2: Adversarial-Sample Aware Algorithm

Input: Δ, Initial training set $\mathcal{T}^0$, Training epochs T, Number of iterations I
Output: DNN model with parameters $(\mathbf{W}^*, \mathbf{b}^*)$
Pre-train the DNN model on $\mathcal{T}^0$ using loss function (11) for T epochs
Save DNN parameters as $(\mathbf{W}^0, \mathbf{b}^0)$
for $i = 0$ **to** I **do**
  Find the maximum violation of $(\mathbf{W}^i, \mathbf{b}^i)$ by solving:

$$\boldsymbol{\theta^i} = \arg\max_{\boldsymbol{\theta} \in \mathcal{D}} \ \nu^f \ \text{ s.t. } (7) - (8), 1 \leq i \leq N_{\text{hid}}, 1 \leq k \leq N_{\text{neu}}, \ (10).$$

  Set $\gamma = \nu^f(\boldsymbol{\theta}^i)$
  if $\gamma \leq \Delta$ **then**
    Set $\mathbf{W}^* = \mathbf{W}^i, \mathbf{b}^* = \mathbf{b}^i$; Break
  else
    Construct $\mathcal{S}^i$ by randomly sampling around θ^i
    Set $\mathcal{T}^{i+1} = \mathcal{T}^i \cup \mathcal{S}^i$ and $(\mathbf{W}_0^i, \mathbf{b}_0^i) = (\mathbf{W}^i, \mathbf{b}^i)$
  end if
  for $t = 0$ **to** T **do**
    Train the DNN on $\mathcal{T}^{i+1}$ using loss function (11) and update DNN parameters to $(\mathbf{W}_{t+1}^i, \mathbf{b}_{t+1}^i)$
    Feed each $\boldsymbol{\theta} \in \mathcal{S}^i$ in the DNN model to obtain predicted solution $\hat{\boldsymbol{x}}$
    if Each $\hat{\boldsymbol{x}}$ is feasible w.r.t. (1)- (2) **then**
      Break
    end if
  end for
  Set $(\mathbf{W}^{i+1}, \mathbf{b}^{i+1}) = (\mathbf{W}_{t+1}^i, \mathbf{b}_{t+1}^i)$
end for

K Run-time Complexity of the Framework

The computational complexity of the framework consists of the complexity of using DNN to predict the solutions, which is $\mathcal{O}\left(N_{\text{hid}} N_{\text{neu}}^2\right)$ (Pan et al., 2020b). Recall that N_{hid} denote the number of hidden layers in DNN (depth), and N_{neu} denotes the number of neurons at each layer (width). In practice, we set N_{hid} to be 3 and observe that the DNN with width N_{neu} of $\mathcal{O}(N)$ can achieve satisfactory optimality performance with universal feasibility guarantee. Therefore, the complexity of using DNN to predict the N variables is $\mathcal{O}\left(N^2\right)$.

Note that the number of decision variables to be optimized is N. After taking $\mathcal{O}(|\mathcal{E}|M)$ operations to calculate the value of $\boldsymbol{b_j^T \theta}$ for each $j \in \mathcal{E}$, the computational complexity of interior-point methods for solving such programs is $\mathcal{O}\left(N^4\right)$, measured by the number of elementary operations assuming that it takes a fixed time to execute each operation (Ye & Tse, 1989). Therefore, the traditional method for solve the OPLC has a computational complexity of $\mathcal{O}\left(N^4 + |\mathcal{E}|M\right)$.

We remark that the computational complexity of the proposed framework is lower than that of traditional algorithms.

L Implementations of DeepOPF+

Recall that the DC-OPF formulation is given as

$$\min_{\boldsymbol{P_G},\, \Phi} \ \sum_{i \in \mathcal{G}} c_i\left(P_{Gi}\right) \tag{41}$$

$$\text{s.t. } \boldsymbol{P_G^{\min}} \leq \boldsymbol{P_G} \leq \boldsymbol{P_G^{\max}}, \tag{42}$$

$$\mathbf{M} \cdot \Phi = \boldsymbol{P_G} - \boldsymbol{P_D}, \tag{43}$$

$$-\boldsymbol{P}_{\text{line}}^{\max} \leq \mathbf{B}_{\text{line}} \cdot \Phi \leq \boldsymbol{P}_{\text{line}}^{\max}. \tag{44}$$

We first reduce the number of decision variables (without losing optimality) by adopting the predict-and-reconstruct framework (Pan et al., 2019). Specifically, it leverages that the admittance matrix (after removing the entries corresponding to the slack bus) $\tilde{\mathbf{M}}$ is of full rank $B-1$, where $B=|\mathcal{B}|$ and is the size of the set of buses. Thus, given each $\boldsymbol{P_D}$, once the non-slack generations $\{P_{Gi}\}_{i\in\mathcal{G}\backslash n_0}$ (n_0 denotes the slack bus index) are determined, the slack generation and the bus phase angles of all buses can be uniquely reconstructed:

$$P_G^{\text{slack}} = \sum_{i\in\mathcal{B}} P_{Di} - \sum_{i\in\mathcal{G}\backslash n_0} P_{Gi}, \tag{45}$$

$$\tilde{\Phi} = \tilde{\mathbf{M}}^{-1}\left(\tilde{\boldsymbol{P}}_G - \tilde{\boldsymbol{P}}_D\right), \tag{46}$$

where n_0 and P_G^{slack} denote the slack bus index and slack bus generation respectively. $\tilde{\boldsymbol{P}}_G$ and $\tilde{\boldsymbol{P}}_D$ are the $(B-1)$-dimensional generation and load vectors for all buses except the slack bus. Consequently, the line flow capacity constraints in (44) can be reformulated as

$$-\boldsymbol{P}_{\text{line}}^{\mathbf{max}} \le \tilde{\mathbf{B}}_{\text{line}}\tilde{\mathbf{M}}^{-1}\left(\tilde{\boldsymbol{P}}_G - \tilde{\boldsymbol{P}}_D\right) \le \boldsymbol{P}_{\text{line}}^{\mathbf{max}}, \tag{47}$$

where $\tilde{\mathbf{B}}_{\text{line}}$ is the line admittance matrix after removing the column of slack bus.[7] Therefore, the reformulated DC-OPF problem takes the form of

$$\min_{\tilde{\boldsymbol{P}}_{\boldsymbol{G}}} \sum_{i\in\mathcal{G}\backslash n_0} c_i\left(P_{Gi}\right) + c_{n_0}\left(\sum_{i\in\mathcal{B}} P_{Di} - \sum_{i\in\mathcal{G}\backslash n_0} P_{Gi}\right) \tag{48}$$

$$\text{s.t. } (47),$$

$$P_{Gi}^{\min} \le P_{Gi} \le P_{Gi}^{\max}, \forall i \in \mathcal{G}\backslash n_0, \tag{49}$$

$$P_{\text{slack}}^{\min} \le \sum_{i\in\mathcal{B}} P_{Di} - \sum_{i\in\mathcal{G}\backslash n_0} P_{Gi} \le P_{\text{slack}}^{\max}. \tag{50}$$

Therefore, we can solve DC-OPF by employing DNNs to depict the mapping between $\boldsymbol{P_D}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{G}}$.

L.1 Removing Non-Critical Inequality Constraints

L.1.1 Removing Non-critical Branch Limits.

We propose the following program for each branch i to remove the non-critical branch limits given the entire load and generation space:

$$\max_{\tilde{\boldsymbol{P}}_{\boldsymbol{G}},\boldsymbol{P_D}} \quad \nu_i - 1 \tag{51}$$

$$\text{s.t.} \quad (49),$$

$$\boldsymbol{P_D} \in \mathcal{D}, \tag{52}$$

$$\boldsymbol{\nu} = |\tilde{\mathbf{X}}\left(\tilde{\boldsymbol{P}}_G - \tilde{\boldsymbol{P}}_D\right)|. \tag{53}$$

Here we assume the load domain $\mathcal{D} = \{\boldsymbol{P_D}|\mathbf{A}_d\boldsymbol{P_D} \le \boldsymbol{b_d}, \exists\tilde{\boldsymbol{P}}_{\boldsymbol{G}} : (47)-(50) \text{ hold}\})$ is restricted to a convex polytope described by matrix $\boldsymbol{A_d}$ and vector $\boldsymbol{b_d}$ and the corresponding constraints. (49) enforces the feasibility of non-slack generations. (53) represents the normalized power flow level at each branch, where $\tilde{\mathbf{X}}$ is obtained from (47) by dividing each row of matrix $\tilde{\mathbf{B}}_{\text{line}}\tilde{\mathbf{M}}^{-1}$ with the value of corresponding line capacity and $\boldsymbol{\nu} \in \mathcal{R}^{|\mathcal{E}|}$.

We remark that problem (51)-(53) can be reformulated as two linear programmings to perform the inference of the absolute sign of power flows in (53):

$$\max_{\tilde{\boldsymbol{P}}_{\boldsymbol{G}},\boldsymbol{P_D}} \; / \; \min_{\tilde{\boldsymbol{P}}_{\boldsymbol{G}},\boldsymbol{P_D}} \quad \tilde{\nu}_i \tag{54}$$

$$\text{s.t.} \quad (49), (52),$$

$$\tilde{\boldsymbol{\nu}} = \tilde{\mathbf{X}}\left(\tilde{\boldsymbol{P}}_G - \tilde{\boldsymbol{P}}_D\right). \tag{55}$$

[7]The matrix $\tilde{\mathbf{B}}_{\text{line}}\tilde{\mathbf{M}}^{-1}$ is well-known as "Power Transfer Distribution Factors" (PTDF) matrix (Chatzivasileiadis, 2018).

If the optimal value of the above maximization (respectively minimization) problem is smaller or equal (respectively greater or equal) than 1 (respectively -1), then the optimal value of (54)-(55) is non-positive for some branch i. Therefore, such non-critical inequality constraint does not affect the feasible solution space such that it is always respected given any input load $\boldsymbol{P_D}$ and can be removed from the DC-OPF problem. By solving (54)-(55), we can derive the set $\mathcal{E}$ of critical branch capacity constraints whose optimal objectives are positive.[8]

L.1.2 Removing Non-critical Slack Bus Generation Limits.

We provide the formulation to identify the critical slack generation limits given the entire load and generation space and the possible violation degree w.r.t. the upper and lower bounds here.

$$\max_{\tilde{\boldsymbol{P}}_{\boldsymbol{G}}, \boldsymbol{P_D}} \quad \nu^u_{\text{slack}} \tag{56}$$

$$\text{s.t.} \quad (45), (49), (52),$$

$$\nu^u_{\text{slack}} = \frac{P^{\text{slack}}_G - P^{\max}_{\text{slack}}}{P^{\max}_{\text{slack}} - P^{\min}_{\text{slack}}}, \tag{57}$$

and

$$\max_{\tilde{\boldsymbol{P}}_{\boldsymbol{G}}, \boldsymbol{P_D}} \quad \nu^l_{\text{slack}} \tag{58}$$

$$\text{s.t.} \quad (45), (49), (52),$$

$$\nu^l_{\text{slack}} = \frac{P^{\min}_{\text{slack}} - P^{\text{slack}}_G}{P^{\max}_{\text{slack}} - P^{\min}_{\text{slack}}}, \tag{59}$$

respectively. Here (57) and (59) denote the (normalized) exceeding of slack bus generation exceeding its upper bound and lower bound, respectively. Therefore, if the optimal values of these proposed optimization problem is non-positive, such slack generation limit is non-critical and does not affect the load-solution feasible region.

We remark that problems (56)-(57), and (58)-(59) are indeed linear programs and can be efficiently solved by the state-of-the-art solvers such as CPLEX or Gurobi. We find that all three test cases could have both critical upper bound and lower bound limits, i.e., both (56)-(57) and (58)-(59) have positive optimal values.

L.2 Maximum Constraints Calibration Rate

To determine the maximum constraints calibration rate while preserving the input region, we solve the following bi-level optimization program:

$$\min_{\boldsymbol{P_D}} \max_{\boldsymbol{P_G}} \ \nu^c \tag{60}$$

$$\text{s.t.} \quad (42),\ (45)-(47),\ (52),$$

$$|PF_{ij}| = |\frac{1}{r_{ij}}(\phi_i - \phi_j)|,\ \forall (i,j) \in \mathcal{E}, \tag{61}$$

$$P^u_{\text{slack}} = (P^{\max}_{\text{slack}} - P^{\text{slack}}_G)/(P^{\max}_{\text{slack}} - P^{\min}_{\text{slack}}), \tag{62}$$

$$P^l_{\text{slack}} = (P^{\text{slack}}_G - P^{\min}_{\text{slack}})/(P^{\max}_{\text{slack}} - P^{\min}_{\text{slack}}), \tag{63}$$

$$\nu^c \leq \frac{P^{\max}_{Tij} - |PF_{ij}|}{P^{\max}_{Tij}}, \forall (i,j) \in \mathcal{E}, \tag{64}$$

$$\nu^c \leq P^u_{\text{slack}}, \tag{65}$$

$$\nu^c \leq P^l_{\text{slack}}, \tag{66}$$

where PF_{ij} denotes the power flow on branch $(i,j) \in \mathcal{E}$. P^u_{slack} and P^l_{slack} represent the relative upper and lower bounds redundancy on slack bus, respectively. Constraint (52) describes the convex

[8]For the critical branch constraints not in $\mathcal{E}$, it is possible to encounter such load input and generation solution profiles using the DNN scheme with the upper/lower bounds truncate ReLU functions in (6) at output layer under the worst-case scenarios with which the power flow on branch i exceeds its transmission limit.

polytope of $\boldsymbol{P_D}$. Constraints (42) and (45)-(46) denote the feasibility of the corresponding $\boldsymbol{P_G}$. Constraints (64)-(66) enforce ν^c as the maximum calibration rate. We employ the KKT-based approach in Sec. 4.1 to solve the above bi-level problem and obtain the calibration rate for DeepOPF+.

L.3 DNN Loss Function in DC-OPF Problem

When adopting the *Adversarial-Sample Aware* algorithm, we design the loss function $\mathcal{L}$ consisting of two parts to guide the training process.

Similar to (Pan et al., 2019), we first represent the feasible active power generation P_{Gi} that satisfies (42) as:

$$P_{Gi} = P_{Gi}^{\min} + \alpha_i \cdot \left(P_{Gi}^{\max} - P_{Gi}^{\min}\right),\ \alpha_i \in [0,1], i \in \mathcal{G}. \tag{67}$$

Therefore, instead of predicting $\{P_{Gi}\}_{i\in\mathcal{G}\setminus n_0}$, we use DNNs to generate the corresponding scaling factors and reconstruct the $\{P_{Gi}\}_{i\in\mathcal{G}\setminus n_0}$ and remaining variables in implementation. The first term of the loss function is the sum of mean square error between the generated scaling factors $\hat{\alpha}_i$ and the reference ones α_i of the optimal solutions:

$$\mathcal{L}_{P_G} = \frac{1}{|\mathcal{G}-1|}\sum_{i\in\mathcal{G}\setminus n_0}(\hat{\alpha}_i - \alpha_i)^2. \tag{68}$$

The second part consists of penalty terms (denoted as $\mathcal{L}_{pen}$) as the summation of the violations for line flow limits and slack bus generation:

$$\begin{aligned}
\mathcal{L}_{pen}^{\text{line}} =& \frac{1}{|\mathcal{E}|}\sum_{k=1}^{|\mathcal{E}|}\max\left(\left(\mathbf{X}\left(\tilde{\boldsymbol{P}}_G - \tilde{\boldsymbol{P}}_D\right)\right)_k^2 - 1, 0\right)\\
\mathcal{L}_{pen}^{\text{slack}} =& \frac{1}{|\mathcal{E}|}\max\left(\frac{P_G^{\text{slack}} - P_{\text{slack}}^{\max}}{P_{\text{slack}}^{\max} - P_{\text{slack}}^{\min}}, 0\right) +\\
& \frac{1}{|\mathcal{E}|}\max\left(\frac{P_{\text{slack}}^{\min} - P_G^{\text{slack}}}{P_{\text{slack}}^{\max} - P_{\text{slack}}^{\min}}, 0\right)
\end{aligned} \tag{69}$$

Here matrix $\mathbf{X}$ is obtained from (47) by dividing each row of matrix $\tilde{\mathbf{B}}_{\text{line}}\tilde{\mathbf{M}}^{-1}$ with the value of corresponding line capacity. The first and second terms of $\mathcal{L}_{pen}^{\text{slack}}$ denote (normalized) the violations of upper bound and lower bound on slack generation, respectively. We remark that after the constraints calibration, the penalty loss is with respect to the adjusted limits. Note here the non-slack generations are always feasible as we predict the $(0,1)$ scaling factors in (67). The total loss is a weighted sum of the two:

$$\mathcal{L} = w_1 \cdot \mathcal{L}_{P_G} + w_2 \cdot \mathcal{L}_{pen}, \tag{70}$$

where w_1 and w_2 are positive weighting factors representing the balance between prediction error and penalty. We apply the widely-used stochastic gradient descent (SGD) with momentum (Qian, 1999) method to update DNN's parameters $(\mathbf{W}, \mathbf{b})$ at each iteration.

L.4 Run-time Complexity of DeepOPF+

According to Appendix K, the computational complexity of DeepOPF+ to predict the non-slack generations $\{P_{Gi}\}_{i\in\mathcal{G}\setminus n_0}$ is $\mathcal{O}\left(B^2\right)$. Reconstructing the phase angles Φ can be achieved by (46), which requires $\mathcal{O}\left(B^2\right)$ operations. Overall, the computational complexity of DeepOPF+ is $\mathcal{O}\left(B^2\right)$. For the traditional solver, the computational complexity of interior-point methods for solving DC-OPF is $\mathcal{O}\left(B^4\right)$, measured by the number of elementary operations. We remark that the computational complexity of DeepOPF+ is lower than that of traditional algorithms.

M Details of DeepOPF+ Design

We present the detailed result of each step in DeepOPF+ design in this appendix.

Table 2: Maximum calibration rates for IEEE Case30/118/300.

	Case30	Case118	Case300
Maximum calibration rate	7.0%	16.7%	21.6%

Table 3: Parameters for test cases.

Case	Number of buses	Number of generators	Number of load buses	Number of branches
Case30	30	6	20	41
Case118	118	19	99	186
Case300	300	69	199	411

* The number of load buses is calculated based on the default load on each bus. A bus is considered a load bus if its default active power consumption is non-zero.

Table 4: Parameters settings of DeepOPF+ for IEEE Case30/118/300

Test case	Variants	Calibration rate	Neurons per hidden layer
Case30	DeepOPF+-3	3.0%	60/30/15
	DeepOPF+-7	7.0%	32/16/8
Case118	DeepOPF+-3	3.0%	200/100/50
	DeepOPF+-7	7.0%	128/64/32
Case300	DeepOPF+-3	3.0%	360/180/90
	DeepOPF+-7	7.0%	256/128/64

Table 5: Preprocessing time to setup DeepOPF+ for IEEE Case30/118/300 in heavy-load regime

Test case	Variants	Determine Calibration rate	Determine DNN size	ASA algorithm	Total time
Case30	DeepOPF+-3	0.2 seconds	0.15 hours	0.83 hour	0.98 hour
	DeepOPF+-7	0.2 seconds	0.15 hours	0.73 hour	0.88 hour
Case118	DeepOPF+-3	20.9 seconds	5.47 hours	7.94 hour	13.42 hour
	DeepOPF+-7	20.9 seconds	5.47 hours	5.31 hour	10.79 hour
Case300	DeepOPF+-3	1185.7 seconds	178.46 hours	25.72 hour	204.51 hour
	DeepOPF+-7	1185.7 seconds	178.46 hours	10.52 hour	189.31 hour

Table 6: Preprocessing time to setup DeepOPF+ for IEEE Case30/118/300 in light-load regime

Test case	Variants	Determine Calibration rate	Determine DNN size	ASA algorithm	Total time
Case30	DeepOPF+-3	0.2 seconds	0.15 hours	0.81 hour	0.96 hour
	DeepOPF+-7	0.2 seconds	0.15 hours	0.72 hour	0.87 hour
Case118	DeepOPF+-3	20.9 seconds	5.47 hours	6.99 hours	12.47 hours
	DeepOPF+-7	20.9 seconds	5.47 hours	4.79 hours	10.27 hours
Case300	DeepOPF+-3	1185.7 seconds	178.46 hours	52.46 hours	231.25 hours
	DeepOPF+-7	1185.7 seconds	178.46 hours	15.82 hours	194.61 hours

First, for determining the maximum calibration rate, the obtained result in shown in Table 2, representing the room for DNN prediction error. We note that the off-the-shell solver returns exact solutions for the problem in (4)-(5).

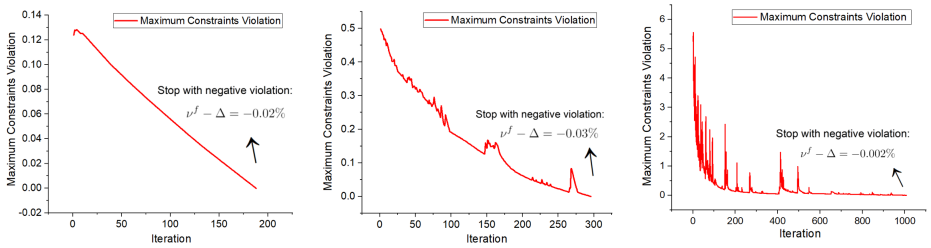
Table 7: Average cost and runtime of SOTA DNN schemes in heavy-load regime.

Case	Scheme	Average cost ($/hr)		Average running time (ms)	
		DNN scheme	Ref.	DNN scheme	Ref.
Case30	DNN-P	732.5	732.2	0.58	45.6
	DNN-D	732.4		0.63	
	DNN-W	732.2		53.02	
	DNN-G	732.5		1.78	
	DeepOPF+-3	732.4		0.50	
	DeepOPF+-7	732.9		0.49	
Case118	DNN-P	121074.7	120822.1	2.13	124.9
	DNN-D	121112.1		15.60	
	DNN-W	120822.1		55.33	
	DNN-G	121299.6		7.72	
	DeepOPF+-3	121051.3		0.56	
	DeepOPF+-7	121313.9		0.55	
Case300	DNN-P	926660.6	925955.0	3.33	83.5
	DNN-D	926590.1		57.92	
	DNN-W	925955.0		77.48	
	DNN-G	926512.3		31.55	
	DeepOPF+-3	926198.4		0.61	
	DeepOPF+-7	926500.4		0.60	

Table 8: Average cost and runtime of SOTA DNN schemes in light-load regime.

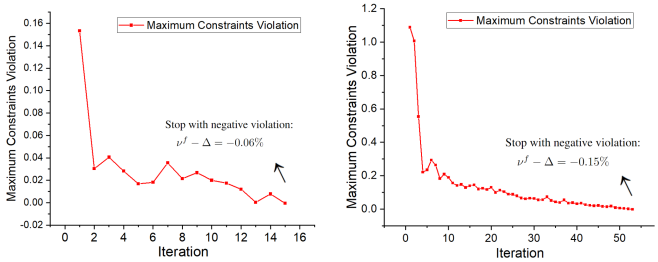
Case	Scheme	Average cost ($/hr)		Average running time (ms)	
		DNN scheme	Ref.	DNN scheme	Ref.
Case30	DNN-P	619.8	619.7	0.50	42.4
	DNN-D	619.8		0.50	
	DNN-W	619.7		46.93	
	DNN-G	620.4		1.75	
	DeepOPF+-3	619.9		0.50	
	DeepOPF+-7	619.8		0.49	
Case118	DNN-P	101843.2	101673.0	1.71	115.4
	DNN-D	101873.6		5.02	
	DNN-W	101673.0		55.55	
	DNN-G	102983.3		4.37	
	DeepOPF+-3	101852.3		0.58	
	DeepOPF+-7	102049.3		0.57	
Case300	DNN-P	778342.4	777878.4	1.71	78.7
	DNN-D	778404.3		25.93	
	DNN-W	777878.4		75.77	
	DNN-G	780368.9		32.30	
	DeepOPF+-3	778070.6		0.60	
	DeepOPF+-7	778675.2		0.60	

Second, for determining the sufficient DNN size, we show the change of the difference between maximum relative constraints violation and calibration rate during iterative solving process via the *Danskin's Theorem* in Fig. 3. From Fig. 3, we observe that for all three test cases, the proposed approach succeeds in reaching a relative constraints violation no larger than the corresponding calibration rate Δ, i.e., $\rho \leq \Delta$, indicating that the verified DNNs, i.e., 32/16/8 neurons, 128/64/32 neurons and 256/128/64 neurons, for IEEE 30-/118/300-bus test cases respective, have enough size to guarantee feasibility within the given load input domain of $[100\%, 130\%]$ of the default load. Note that we can directly construct DNNs to ensure universal feasibility for the three IEEE test cases. We further evaluate the performance of the DNN model obtained following the steps in Sec. 4.2.2 without using the *Adversarial-Sample Aware* algorithm. While ensuring universal feasibility, it suffers from an undesirable optimality loss, up to 2.31% and more than 130% prediction error.



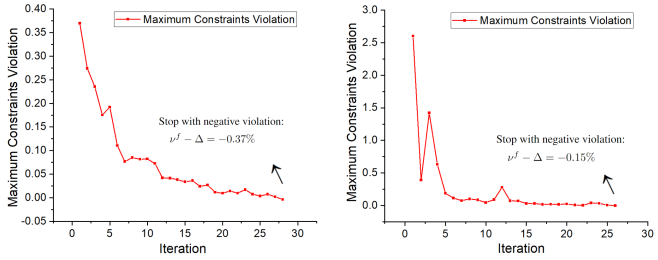
(a) Case30. (b) Case118. (c) Case300.

Figure 3: Maximum relative constraints violation compared with calibration rate ($\nu^f - \Delta$) at each iteration for IEEE Case30, Case118, and Case300 test case.



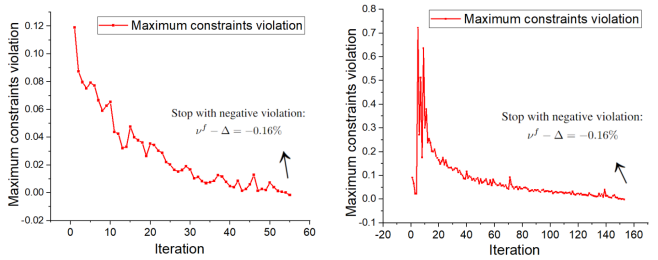
(a) Case118. (b) Case300.

Figure 4: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for IEEE Case118 and IEEE Case300 in light-load regime with 7% calibration rate.



(a) Case118. (b) Case300.

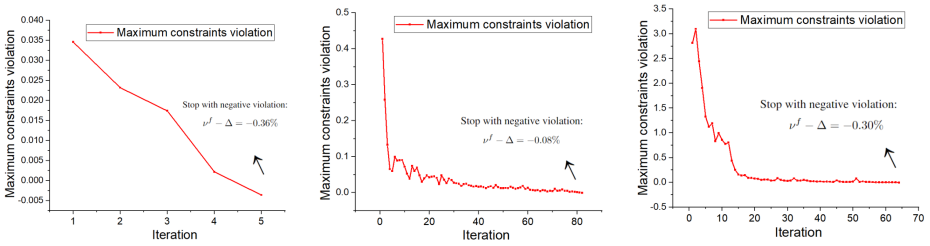
Figure 5: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for IEEE Case118 and IEEE Case300 in heavy-load regime with 7% calibration rate.



(a) Case118. (b) Case300.

Figure 6: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for IEEE Case30/118/300 in light-load regime with 3% calibration rate.

Third, the DNN models trained with the *Adversarial-Sample Aware* algorithm achieve lower optimality loss (up to 0.19%) while preserving universal feasibility. The observation justifies the effectiveness of *Adversarial-Sample Aware* algorithm. We further present the relative violation ($\nu^f - \Delta$) on IEEE 30-/118/300-bus test cases at each iteration in both light-load and heavy-load regimes for



(a) Case30. (b) Case118. (c) Case300.

Figure 7: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for IEEE Case30/118/300 in heavy-load regime with 3% calibration rate.

illustration in Fig. 4 and Fig. 5 with a 7% calibration rate. The above observations show that the *Adversarial-Sample Aware* can efficiently achieve universal feasibility guarantee within both light-load and heavy-load regimes for IEEE 118-/300-bus test cases with at most 52 iterations. We remark that for Case30, the initial worst-case violation of the trained DNN with 7% calibration rate is less than zero (-9.28% and -2.93% in light-load and heavy-load regimes respectively) and hence without the need for adversarial training. The results under the 3% calibration rate are presented in Fig. 6 and Fig. 7, for which we observe that the ASA would take a longer number of iterations to achieve the universal feasibility guarantee due to the smaller room for prediction errors, e.g., at most 152 iterations. For Case30 under light-load regime with 3% calibration rate, its initial worst-case violation is less than zero (-7.53%) and hence without the need of ASA iterations.

Furthermore, we present the parameters of three IEEE test cases and the settings of two DeepOPF+ schemes in Table 3 and Table 4 respectively. The detailed runtime and cost and the time to configure the framework are listed in Table 6-Table 8 for each test case. Note that though a single DC-OPF may be efficient solved by the existing solver, due to increasing uncertainty from renewable generation and flexible load, grid operators now need to solve DC-OPF problems under many scenarios in a short interval, e.g., 1000 scenarios in 1 minutes, to obtain a stochastically optimized solution, e.g., $\sim$2 minutes for the iterative solvers to solve a large number of DC-OPF problems for Case118, resulting the fail of real-time operation. In contrast, the developed DNN scheme can return the solution with $\times$228 speedups, i.e., less than 0.6 seconds in total. In addition, though our method takes additional training efforts, 1) it is conducted offline, once the DNN is configured, it can be continuously applied to many test instances such that the complexity is amortized, e.g., < 0.5 ms for DC-OPF problems if the system operator needs to solve DC-OPF per 5 minutes over 1000 scenarios over a year; 2) as illustrated, the obtained DNN outperforms the existing approaching in avoiding any post-processing and resulting in a lower real-time runtime complexity, showing its advantage; 3) our theoretical analysis shows that the design can always provide the corresponding useful upper/lower bounds in each step of the framework in polynomial time, which can still be utilized for constraints calibration and DNN performance analysis; 4) the process can be further accelerated by applying advanced computation parallel techniques. Finally, we remark that if an impractically large DNN size is required, it would introduce an additional computational challenge, which can require more configuration efforts of the approach and it can be a potential limitation. It is also an interesting direction for solving the constrained program w.r.t. the DNN parameters and determining the sufficient DNN size more efficiently. We would like to leave how to set up the DNNs more efficiently and accelerate the corresponding steps as future work, which is non-trivial and still an open problem in DNN scheme design.

N NON-CONVEX OPTIMIZATION EXAMPLE

We further consider solving a non-convex linearly constrained program with a non-convex objective function and linear constraints adapted from (Donti et al., 2021). We examine this task for illustration:

$$\min_{y \in \mathcal{R}^n} \quad \frac{1}{2} y^T Q y + p^T \sin(y), \text{s.t.} \quad Ay = x, -h \leq Gy \leq h, \tag{71}$$

for constants problem parameter $Q \in \mathcal{R}^{n\times n}, p \in \mathcal{R}^n, A \in \mathcal{R}^{n_{\text{eq}}\times n}, G \in \mathcal{R}^{n_{\text{ineq}}\times n}, h \in \mathcal{R}^{n_{\text{ineq}}}$. Here $x \in \mathcal{R}^{n_{\text{eq}}}$ is the problem input and $y \in \mathcal{R}^n$ denotes the decision variable. n_{ineq} and n_{eq} are the number of inequality and equality constraints. Here we focus on the non-degenerate case such that $n_{\text{eq}} \leq n$. Therefore, the DNN task aims to learn the mapping between x to the optimal y. Similar to (Donti et al., 2021), Q is set to be a diagonal matrix whose diagonal entries are drawn i.i.d. from the uniform distribution on $[0, 1]$. The entries of A, G are drawn i.i.d. from the unit normal distribution. The problem input region of x is set to be $[-1, 1]$ for each dimension. To ensure the problem feasibility, we set $h_i = \sum_j |(GA^+)_{ij}|$, where A^+ is the Moore-Penrose pseudoinverse of A. The feasibility of the problem can be seen that the point $y = A^+x$ is feasible. However, such a point can be generally non-optimal with large optimality loss. In our simulation, we set $n = 50$, $n_{\text{eq}} = 25$, and $n_{\text{ineq}} = 25$. Therefore, the considered optimization has 50 variables, 25 equality constraints, and 100 inequality constraints.

We follow the procedures in the *preventive learning* framework to generate the DNN with universal feasibility guarantee and achieve strong optimality performance.

N.1 Reformatting the problem with only inequality constraints

We reformulate the non-convex optimization with only $n - n_{\text{eq}}$ independent variables of y_2. Note that the equality constraints can be reformulated as

$$[A_1, A_2] \left[\begin{array}{c} y_1 \\ y_2 \end{array} \right] = x \tag{72}$$

Here $A_1 \in \mathcal{R}^{n_{\text{eq}}\times n_{\text{eq}}}, y_1 \in \mathcal{R}^{n_{\text{eq}}}$ and $A_2 \in \mathcal{R}^{n_{\text{eq}}\times(n-n_{\text{eq}})}, y_2 \in \mathcal{R}^{n-n_{\text{eq}}}$. Therefore, given x and y_2, the corresponding y_1 can be uniquely recovered, i.e., $y_1 = A_1^{-1}(x - A_2y_2)$. Based on the above reformulation, the inequality constraints are given as

$$[G_1, G_2] \left[\begin{array}{c} y_1 \\ y_2 \end{array} \right] \leq h, \quad -[G_1, G_2] \left[\begin{array}{c} y_1 \\ y_2 \end{array} \right] \leq h \tag{73}$$

and hence

$$G_1A_1^{-1}x + (G_2 - G_1A_1^{-1}A_2)y_2 \leq h, \quad G_1A_1^{-1}x + (G_2 - G_1A_1^{-1}A_2)y_2 \geq -h \tag{74}$$

The objective can be equivalent modified by replacing the terms w.r.t. y_1 to be y_2 from $y_1 = A_1^{-1}(x - A_2y_2)$. This completes the pre-reformulation of the above non-convex optimization.

N.2 Determine the maximum allowable calibration rate

We first examine that all inequality constraints are critical, i.e., exist a y such that the constraint is binding. We then further determine the maximum calibration rate. From the description in Sec. 4.1, the program to determine the maximum calibration rate is given as

$$\min_{x\in[-1,1]} \max_{y,\nu^c} \quad \nu^c \tag{75}$$

$$\text{s.t. } (74)$$

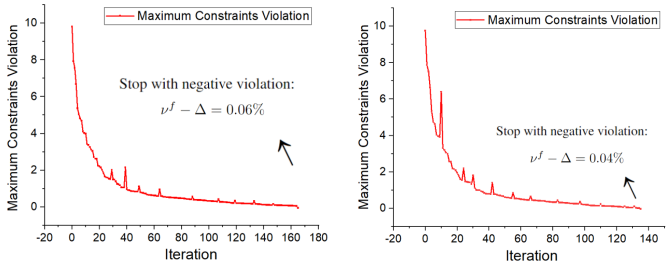
$$\nu^c \leq (h_i - (G_1A_1^{-1}x + (G_2 - G_1A_1^{-1}A_2)y_2)_i)/h_i, \ i = 1, ..., n_{\text{ineq}}, \tag{76}$$

$$\nu^c \leq (h_i + (G_1A_1^{-1}x + (G_2 - G_1A_1^{-1}A_2)y_2)_i)/h_i, \ i = 1, ..., n_{\text{ineq}}. \tag{77}$$

Note that given x, the inner problem is an LP and can be equivalently expressed by its sufficient and necessary KKT conditions. Following the MILP steps in Sec. 4.1, we solve the above program to determine the maximum allowable calibration rate, we observe that the Gurobi solver with the branch-and-bound provides its optimal solution with zero optimality gap within 42ms. The corresponding optimal $\nu^{c*} = 100\%$, implying we can set $h = 0$ such that problem is still feasible for each problem input $x \in [-1, 1]$.

N.3 Determine the sufficient DNN size to guarantee universal feasibility

In our simulation, we consider a DNN with 3 hidden layers and each layer has 50 neurons. Following the steps in Sec. 4.2, we observe that such a DNN size is sufficient to guarantee universal feasibility. The corresponding program is given as



(a) 5% calibration rate. (b) 10% calibration rate.

Figure 8: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for the non-convex optimization example with 5% and 10% calibration rate.

$$\min_{\mathbf{W},\mathbf{b}} \max_{x\in[-1,1]} \quad \nu^f \tag{78}$$

$$\text{s.t.} \quad (7)-(8), 1 \le i \le N_{\text{hid}}, 1 \le k \le N_{\text{neu}},$$

$$\nu^f = \max_{i=1,\ldots,n_{\text{ineq}}} \left\{ \begin{array}{c} (G_1 A_1^{-1} x + (G_2 - G_1 A_1^{-1} A_2)\hat{y}_2)_i / h_i \\ -(G_1 A_1^{-1} x + (G_2 - G_1 A_1^{-1} A_2)\hat{y}_2)_i / h_i \end{array} \right\}. \tag{79}$$

Here $\hat{y}_2$ is the prediction of the DNN. We observe that the tested DNN size is sufficient to guarantee universal feasibility by achieve an upper bound of the relative violation of $\rho - \nu^f$ as -9.3% within $\sim$ 6 minutes. It implies that the tested DNN size is sufficient to guarantee universal feasibility. Recall that the obtained DNN-FG achieves unsatisfactory optimality performance (71.38% optimality loss) as it only focuses on feasibility.

N.4 Application of Adversarial-Sample Aware training algorithm

We hence implement the proposed ASA training algorithm to further improve the optimality performance of the DNN with 5% and 10% calibration rates respectively. The time to obtain the corresponding (Pre-DNN-5, Pre-DNN-10) with 5% and 10% calibration rate are < 52 minutes and < 44 minutes respectively.

We compare our approach against the classical non-convex optimization solver IPOPT and the other DNN schemes DNN-P, DNN-D,DNN-W, and DNN-G. The number of training data is 15,000, and the number of test data is 3,000. The DNN size is set as 3 hidden layers and each layer has 50 neurons. The results are listed in Table 9, and the worst-case violation at each iteration in the ASA training algorithm are given in Fig. 8. Here the optimality *Loss* metric is calculated as the average of (DNN objective$-$Optimal objective)$/|$Optimal objective$|$. The negativity of *Scheme* and *Ref* simply means that the obtained DNN objective and Optimal objective of optimization (71) is negative.

Table 9: Simulation results of different DNN schemes for the non-convex optimization example.

Scheme	Average objective			Average running time (ms)			Feasibility rate (%)	Worst-case violation (%)
	Scheme	Ref.	Loss (%)	Scheme	Ref.	Speedup		
DNN-P	-5.44	-5.47	0.40	1.36	86.6	85.7	39.8	68.3
DNN-D	-5.44		0.42	0.79		117.0	39.8	41.5
DNN-W	-5.47		0	86.6		1.02	100	0
DNN-G	53.69		1076.0	1.00		87.0	100	0
Pre-DNN-5	-5.45		0.34	0.60		144.9	100	0
Pre-DNN-10	-5.43		0.67	0.60		145.3	100	0

* Feasibility rate and Worst-case violation are the results *before* post-processing. Feasibility rates (resp Worst-case violation) after post-processing are 100% (resp 0) for all DNN schemes. We hence report the results before post-processing to better show the advantage of our design. Speedup and Optimality loss are the results *after* post-processing of the final obtained feasible solutions.

* The *correction* step in DNN-D (with 10^{-4} rate) is faster compared with l_1-projection in DNN-P, resulting in higher speedups.

We remark that our obtained DNN schemes (Pre-DNN-5, Pre-DNN-10) with 5% and 10% calibration rates outperform the existing DNN scheme in ensuring universal feasibility and maintaining minor optimality loss. The speedups of our scheme are also significantly larger than the other methods as post-processing steps to recover solution feasibility are avoided.