

Optimization Learning

Pascal Van Hentenryck

NSF AI Institute for Advances in Optimization (AI4OPT)
 Georgia Institute of Technology
 Coda Building 12th/14th floor
 756 W Peachtree St NW, Atlanta, GA 30308
 Email: pvh@gatech.edu

January 8, 2025

Abstract

This article introduces the concept of *optimization learning*, a methodology to design optimization proxies that learn the input/output mapping of parametric optimization problems. These optimization proxies are trustworthy by design: they compute feasible solutions to the underlying optimization problems, provide quality guarantees on the returned solutions, and scale to large instances. Optimization proxies are differentiable programs that combine traditional deep learning technology with repair or completion layers to produce feasible solutions. The article shows that optimization proxies can be trained end-to-end in a self-supervised way. It presents methodologies to provide performance guarantees and to scale optimization proxies to large-scale optimization problems. The potential of optimization proxies is highlighted through applications in power systems and, in particular, real-time risk assessment and security-constrained optimal power flow.

1 Introduction

Optimization technologies have been widely successful in industry: they dispatch power grids, route transportation and logistic systems, plan and operate supply chains, schedule manufacturing systems, orchestrate evacuations and relief operations, clear markets for organ exchanges to name only a few. Yet there remain applications where optimization technologies still face computational challenges, including when solutions must be produced in real time or when there are planners and operators in the loop who expect fast interactions with an underlying decision-support system.

Many engineering applications, however, operate on physical infrastructures that change relatively slowly. As a result, optimization technologies are often used to solve the same core applications repeatedly on instances that are highly similar in nature. In addition, it is reasonable to assume that these instances followed a distribution learned from historical data, possibly augmented with forecasts for future conditions. The overarching scientific question is whether machine learning can help optimization technologies expand the realm of tractable optimization problems.

This paper, which summarizes a significant part of my keynote at the EURO-2024 conference in Copenhagen, reviews the concept of *optimization learning*, to address this challenge. The fundamental idea behind optimization learning is to learn the input/output mapping of optimization problems, i.e., the mapping from inputs to optimal solutions. The paper presents three fundamental methodologies in optimization learning: primal optimization proxies, dual optimization proxies, and primal-dual learning. Primal optimization proxies return (near-optimal) feasible solutions to parametric optimization problem through a combination of deep learning and repair layers. Dual optimization proxies return (near-optimal) dual solutions to parametric optimization problems through a combination of deep learning and completion layers. Primal-dual learning mimics traditional augmented Lagrangian methods to provide primal proxies that learn both a primal and a dual network.

These methodologies are illustrated on two significant applications in power systems operations: the real-time risk assessment of a transmission systems and the security-constrained optimal power

under N-1 Generator and Line Contingencies. In each case, optimization learning brings orders of magnitude improvements in efficiency, making it possible to solve the applications in real time with high accuracy, an outcome that could not have been achieved by state-of-the-art optimization technology.

The rest of this paper is organized as follows. Section 2 presents the problem formulation, i.e., the learning task considered in the paper. Section 3 presents a running example for illustrating the methodologies: the economic dispatch problem run every five minutes by independent system operators in the US. Section 4 discusses empirical risk minimization in machine learning and baseline approaches for approaching optimization learning. The section provides the background for understanding the subsequent section and, in particular, a high-level view of how machine learning models are trained. Section 5 presents primal optimization and their application to real-time risk assessment. Section 6 presents dual optimization proxies and their application to DC optimal power flow. Section 7 presents primal-dual learning and its application to security-constrained optimal power flows. Section 8 concludes the paper and identifies future research directions.

The paper unifies and reformulates, in a general framework, prior research presented in several papers [9, 23, 36, 37], where extensive literature reviews are available. “Recent” reviews of constrained optimization learning and machine learning for optimal power flows are available in [26, 43]. For space reasons, this paper does not review the applications of optimization learning for combinatorial optimization. Some applications in supply chains, manufacturing, and transportation, where the optimization problems contain discrete variables, can be found in [8, 25, 34, 48]. For a general introduction to machine learning for combinatorial optimization, see [7].

2 Problem Formulation: The Learning Task

This paper considers applications that require the solving of *parametric optimization problems*

$$\Phi(\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}} f_{\mathbf{x}}(\mathbf{y}) \text{ subject to } \mathbf{h}_{\mathbf{x}}(\mathbf{y}) = 0 \text{ & } \mathbf{g}_{\mathbf{x}}(\mathbf{y}) \geq 0, \quad (1)$$

where \mathbf{x} represents instance parameters that determine the objective function $f_{\mathbf{x}}$ and the constraints $\mathbf{h}_{\mathbf{x}}$ and $\mathbf{g}_{\mathbf{x}}$. Such parametric optimization can be viewed as mapping from an input \mathbf{x} to an output $\mathbf{y} \Phi(\mathbf{x})$ that represents its optimal solution (or a selected optimal solution). In addition, for many applications, it is reasonable to assume that the instance parameters \mathbf{x} are characterized by a distribution learned from historical data, possibly augmented to cover future conditions with forecasting algorithms. The goal of this paper is to study how to use the fusion of machine learning and optimization for “solving” (unseen) optimization instances $\Phi(\mathbf{x}_j)$ orders of magnitude faster than a state-of-the-art optimization solver can. This task is called *optimization learning* in the rest of this paper.

More formally, given a probability distribution \mathcal{P} of instance parameters, the goal is to learn a *primal optimization proxy* $\Phi^{\uparrow} : \mathbb{R}^m \mapsto \mathbb{R}^a$ that produces feasible solutions, i.e.,

$$\forall \mathbf{x} \sim \mathcal{P} : \mathbf{h}_{\mathbf{x}}(\Phi^{\uparrow}(\mathbf{x})) = 0 \text{ & } \mathbf{g}_{\mathbf{x}}(\Phi^{\uparrow}(\mathbf{x})) \geq 0$$

and a *dual optimization proxy* $\Phi^{\downarrow} : \mathbb{R}^m \mapsto \mathbb{R}^b$ that returns valid lower bounds, i.e.,

$$\forall \mathbf{x} \sim \mathcal{P} : f_{\mathbf{x}}(\Phi^{\downarrow}(\mathbf{x})) \leq f_{\mathbf{x}}(\Phi(\mathbf{x})).$$

3 The Economic Dispatch Optimization

This paper illustrates optimization learning on a running example and its variants: the Economic Dispatch (ED) optimization with reserve requirements. The ED optimization is run every five minutes in the US and is the backbone of power system operations. The formulation described in Model 1 captures the essence of how real-time markets are cleared by Independent Systems Operators (ISOs) in the United States, although the reality is obviously more complex.

Constraints (2b) and (2c) are the power balance and minimum reserve requirement constraints. Constraints (2d) ensure that the active power and reserves of each generator does not violate their maximum capacities. Constraints (2e) and (2f) enforce the limits on each generator dispatch and

Model 1 The Economic Dispatch Formulation.

$$\min_{\mathbf{p}, \mathbf{r}, \xi_{\text{th}}} c(\mathbf{p}) + M_{\text{th}} \|\xi_{\text{th}}\|_1 \quad (2a)$$

$$\text{s.t. } \mathbf{e}^\top \mathbf{p} = \mathbf{e}^\top \mathbf{d}, \quad (2b)$$

$$\mathbf{e}^\top \mathbf{r} \geq R, \quad (2c)$$

$$\mathbf{p} + \mathbf{r} \leq \bar{\mathbf{p}}, \quad (2d)$$

$$\mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}}, \quad (2e)$$

$$\mathbf{0} \leq \mathbf{r} \leq \bar{\mathbf{r}}, \quad (2f)$$

$$\underline{\mathbf{f}} - \xi_{\text{th}} \leq \mathbf{PTDF}(\mathbf{p} - \mathbf{d}) \leq \bar{\mathbf{f}} + \xi_{\text{th}}, \quad (2g)$$

$$\xi_{\text{th}} \geq \mathbf{0}. \quad (2h)$$

reserves. Constraints (2g) express the thermal limits on each limits using a Power Transfer Distribution Factor (PTDF) representation, which is the state of the art in industry [19, 29]. The thermal limits are soft constraints, which is how they are modeled by US ISOs (e.g., [29, 31]). Their violations incur a high cost, modeled using slack variables ξ_{th} which are penalized in the objective.

4 Empirical Risk Minimization

Machine learning seems to be an ideal approach to “solve” these parametric optimization problems ” and replace optimization all together. Indeed, an optimization problem can be viewed as mapping from its input \mathbf{x} to an optimal solution \mathbf{y} , and deep learning networks are universal approximators [20]. This section presents baseline machine learning approaches for optimization learning.

4.1 Supervised Learning

A key feature of optimization learning is its ability to generate correct labels for each input: it suffices to solve the optimization problem (1). Hence, to train a machine learning model, a data set $\mathcal{I} = \{\mathbf{x}_i\}_{i \in [n]}$ of inputs sampled from the distribution \mathcal{P} can be augmented with their optimal solutions to obtain a training data set $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i = \Phi(\mathbf{x}_i))\}_{i \in [n]}$. Testing and validation data sets can be obtained similarly.

Supervised learning can then used to fit the *learnable parameters* θ of a parametric machine learning model \mathcal{M}_θ by minimizing a loss function of the form

$$\operatorname{argmin}_\theta \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\mathbf{y}_i, \mathcal{M}_\theta(\mathbf{x}_i)) \quad (3)$$

where

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|.$$

The “optimal” value θ^* for the learnable parameters turns the machine learning model into a function \mathcal{M}_{θ^*} that approximates the parametric optimization Φ . This minimization problem is known as *an empirical risk minimization* and is typically solved using stochastic gradient descent as illustrated in Figure 1. The machine learning training consists of series of forward and backward passes. At iteration t , with θ^t as the values of the learnable parameters, the forward pass evaluates $\mathcal{M}_{\theta^t}(\mathbf{x})$ to obtain $\hat{\mathbf{y}}^t$, and the backward pass updates the learnable parameters θ^t using a gradient step

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}^t)}{\partial \theta}.$$

Unfortunately, in engineering tasks, such an approximation is not a primal optimization proxy: by virtue of being a regression, the machine learning predictions are unlikely to satisfy the problem

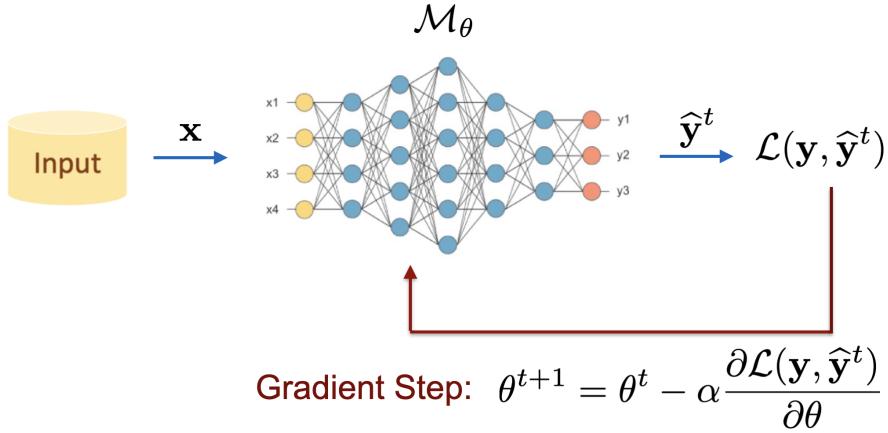


Figure 1: The Training of Supervised Learning Models.

constraints, which may have some significant consequences when optimization models are used to plan, operate, or control physical infrastructures. In fact, problem (3) should really be viewed as an *empirical minimization under constraints*, which raises significant challenges for machine learning technologies.

4.2 Lagrangian Duality in Supervised Learning

Early research on this topic [14, 15] proposed to combine machine learning and Lagrangian duality, using a parametric loss function of the form

$$\mathcal{L}_{\lambda, \nu}^d(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\| + \lambda |\mathbf{h}_x(\hat{\mathbf{y}})| + \nu \max(0, -\mathbf{g}_x(\hat{\mathbf{y}})).$$

where $|\mathbf{h}_x(\hat{\mathbf{y}})|$ and $\max(0, -\mathbf{g}_x(\hat{\mathbf{y}}))$ capture the violations of the equality and inequality constraints. The multipliers λ and ν define the penalties for these constraint violations; they can be trained by mimicking subgradient methods for computing Lagrangian duals, alternating between training the machine learning model and adjusting the multipliers with subgradient steps. At each step t , the Lagrangian dual optimization solves a training problem

$$\theta^{t+1} = \operatorname{argmin}_{\theta^t} \frac{1}{n} \sum_{i=1}^n \mathcal{L}_{\lambda^t, \nu^t}^d(\mathbf{y}_i, \mathcal{M}_{\theta^t}(\mathbf{x}_i))$$

with the multipliers fixed at λ^t and ν^t . The multipliers are then adjusted using the constraint violations, i.e.,

$$\begin{aligned} \lambda^{t+1} &= \lambda^t + \rho \frac{1}{n} \sum_{i=1}^n |\mathbf{h}_x(\mathcal{M}_{\theta^{t+1}}(\mathbf{x}_i))| \\ \nu^{t+1} &= \nu^t + \rho \frac{1}{n} \sum_{i=1}^n \max(0, -\mathbf{g}_x(\mathcal{M}_{\theta^{t+1}}(\mathbf{x}_i))). \end{aligned}$$

Experimentally, this Lagrangian dual approach has been shown to reduce violations, sometimes substantially. However, being a regression, it almost always violates constraints on unseen instances, and is not an primal optimization proxy.

5 Primal Optimization Proxies

Optimization proxies were introduced to remedy the limitations of the baseline machine learning models. Their high-level architecture is depicted in Figure 2. An optimization proxy combines a parametric machine learning model \mathcal{M}_θ (typically a deep neural network) that produces an approximation $\hat{\mathbf{y}}$ with a *repair layer* \mathcal{R} that transforms $\hat{\mathbf{y}}$ into a feasible solution $\check{\mathbf{y}} = \mathcal{R}(\hat{\mathbf{y}})$, i.e., $\check{\mathbf{y}}$ satisfies

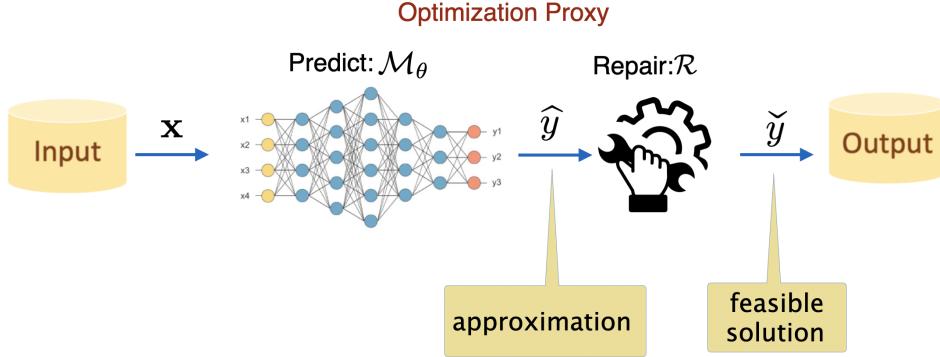


Figure 2: The Architecture of Optimization Proxies.

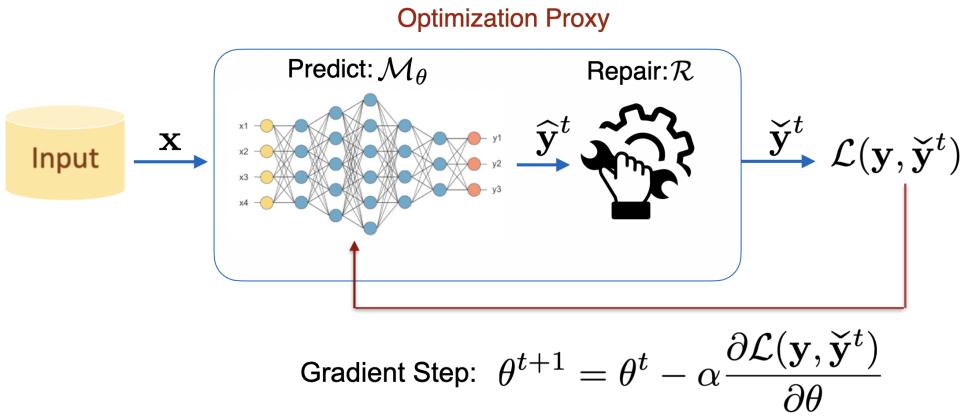


Figure 3: End to End Training of Optimization Proxies.

$\mathbf{h}_x(\check{y}) = 0$ & $\mathbf{g}_x(\check{y}) \geq 0$. The repair layer can be thought of as a projection of \hat{y} into the feasible space of the optimization problem.

The simplest way to implement an optimization proxy is to train the parametric machine learning as defined in Section 4 to obtain θ^* and to use the repair layer at inference time to obtain a feasible solution $\mathcal{R}(\mathcal{M}_{\theta^*}(\mathbf{x}))$ for any instance \mathbf{x} . The composition $\mathcal{R} \circ \mathcal{M}_{\theta^*}$ is a primal optimization proxy. Moreover, an elegant way to implement the repair layer is to solve the optimization model

$$\mathcal{R}(\hat{y}) = \operatorname{argmin}_{\mathbf{y}} \|\mathbf{y} - \hat{y}\| \text{ subject to } \mathbf{h}_x(\mathbf{y}) = 0 \text{ & } \mathbf{g}_x(\mathbf{y}) \geq 0 \quad (4)$$

that finds the feasible point closest to \hat{y} . This is the approach taken in [44].

An appealing alternative approach consists of training the optimization proxy end-to-end as shown in Figure 3. The challenge is to compute the gradient

$$\frac{\partial \mathcal{L}(\mathbf{y}, \check{y}^t)}{\partial \theta} = \frac{\partial \mathcal{L}(\mathbf{y}, \mathcal{R}(\mathcal{M}_{\theta^t}(\mathbf{x})))}{\partial \theta}.$$

By the chain rule, it can be computed as

$$\frac{\partial \mathcal{L}(\mathbf{y}, \check{y}^t)}{\partial \theta} = \frac{\partial \mathcal{L}(\mathbf{y}, \check{y}^t)}{\partial \check{y}^t} \frac{\partial \mathcal{R}(\check{y}^t)}{\partial \hat{y}^t} \frac{\partial \mathcal{M}_{\theta^t}(\mathbf{x})}{\partial \theta}$$

The difficulty, of course, is to differentiate the term $\frac{\partial \mathcal{R}(\check{y}^t)}{\partial \hat{y}^t}$, which is the topic of Sections 5.1 and 5.2.

5.1 Implicit Differentiation

A broad technique to differentiate through the repair layer is to use Cauchy's implicit function theorem [27]. Machine learning models can thus support two types of layers in their networks: *explicit* layers

	Explicit Layer	Implicit Layer
Forward Pass	$f(x)$	find y such that $f(x, y) = 0$
Backward Pass	$\frac{\partial f(x)}{\partial x}$	$\frac{\partial g(x)}{\partial x}$ where $f(x, g(x)) = 0$

Figure 4: Explicit versus Implicit Layers in Deep Learning.

that are specified by differentiable functions and *implicit* layers whose outputs are the solutions to systems of equations. Each layer should support the forward and backward passes as presented in Section 4. Figure 4 highlights the functionalities of explicit and implicit layers. The forward pass of an implicit layer receives an input \mathbf{x} and finds an output \mathbf{y} that satisfies a system of equations $f(\mathbf{x}, \mathbf{y}) = 0$, i.e., it solves a system of equations during training. The implicit function theorem is the key element for the backward pass. If $g(\mathbf{x})$ is a function that, given an input \mathbf{x} , returns an output \mathbf{y} that satisfies $f(\mathbf{x}, \mathbf{y}) = 0$. then the backward pass should return $\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}$. In general, there is no closed form for function g but the implicit function theorem gives a closed form for its gradient:

$$\left[\frac{\partial g_i(x)}{\partial x_j} \right] = \left[J_{f,y}(x, g(x)) \right]^{-1} \left[J_{f,x}(x, g(x)) \right]$$

where

$$J_{f,y}(a_1, a_2) = \left[\frac{\partial f_i}{\partial y_i}(a_1, a_2) \right]$$

and similarly for $J_{f,x}$.

The implicit function theorem can be applied to the repair layer in (4) by taking the KKT conditions of the projection optimization. These define a system of equations; the forward pass of this repair layer finds a solution to these KKT conditions; the backward pass uses the implicit function theorem and applies the closed form for the gradient of g . There are, of course, limitations to this approach. On the one hand, solving the KKT conditions during training for each instance and at inference time may be too costly. On the other hand, for discrete problems, the KKT conditions are not well-defined, but the primal proxy could use a continuous relaxation of the optimization problem.

Implicit layers have been a topic of interest in machine learning. They are the foundations of what are called *deep declarative networks* and differentiable optimization layers (e.g., [1, 2, 16]). Implicit layers have also been heavily used in decision-focused learning, where the goal is to train a machine learning model by taking into account the effect of its predictions on a downstream optimization. See, for instance, [47] and the survey in [26] to see relationships between the various approaches.

5.2 Dedicated Repair Layers

Because of the high computational cost of implicit layers, it is interesting to explore the design of dedicated repair layers and learning architectures that ensure fast training and inference times. This section illustrates this concept on the economic dispatch optimization, where the challenge is to restore the feasibility of the power balance and reserve constraints. Figure 5 depicts the architecture of a primal optimization proxy for the ED problem. It uses a sigmoid layer to enforce the bound constraints, followed by a repair layer for the power balance constraint and another repair layer for the reserve constraints.

Consider first a dedicated repair layer for the balance constraint. It takes as input an estimate \hat{p}_g for the power of each generator g . It uses ideas from control systems to scale all generators proportionally, up or down, to transform $\hat{\mathbf{p}} = \mathcal{M}_\theta(\mathbf{x})$ into a solution $\tilde{\mathbf{p}}$ to the balance constraint as follows:

$$\tilde{\mathbf{p}} = \begin{cases} (1 - \zeta^\uparrow)\hat{\mathbf{p}} + \zeta^\uparrow \bar{\mathbf{p}} & \text{if } \mathbf{1}^\top \hat{\mathbf{p}} < \mathbf{1}^\top \mathbf{d} \\ (1 - \zeta^\downarrow)\hat{\mathbf{p}} + \zeta^\downarrow \underline{\mathbf{p}} & \text{otherwise} \end{cases} \quad (5)$$

where ζ^\uparrow and ζ^\downarrow are defined as,

$$\zeta^\uparrow = \frac{\mathbf{1}^\top \mathbf{d} - \mathbf{1}^\top \hat{\mathbf{p}}}{\mathbf{1}^\top \bar{\mathbf{p}} - \mathbf{1}^\top \hat{\mathbf{p}}} \quad \zeta^\downarrow = \frac{\mathbf{1}^\top \hat{\mathbf{p}} - \mathbf{1}^\top \mathbf{d}}{\mathbf{1}^\top \bar{\mathbf{p}} - \mathbf{1}^\top \underline{\mathbf{p}}}$$

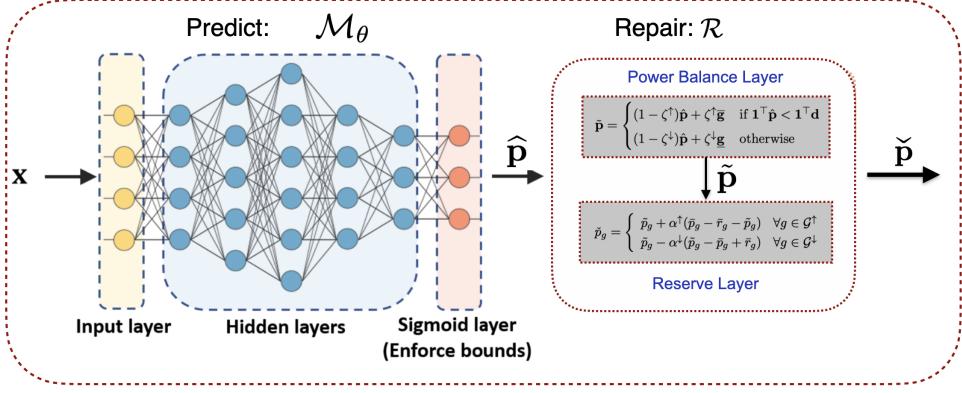


Figure 5: The Optimization Proxy for the Economic Dispatch Problem (adapted from [9]).

and \mathbf{p} and $\bar{\mathbf{p}}$ are the lower and upper bounds on the generators. This repair layer (5) is differentiable almost everywhere and can thus be naturally integrated in the training of the machine learning model. Note that the formula for $\tilde{\mathbf{p}}$, i.e.,

$$\tilde{\mathbf{p}} = (1 - \zeta^\uparrow)\hat{\mathbf{p}} + \zeta^\uparrow\bar{\mathbf{p}} \quad \text{or} \quad \tilde{\mathbf{p}} = (1 - \zeta^\downarrow)\hat{\mathbf{p}} + \zeta^\downarrow\bar{\mathbf{p}}$$

can be generated during the forward pass based on the result of the test $\mathbf{1}^\top \hat{\mathbf{p}} < \mathbf{1}^\top \mathbf{d}$. It also admits subgradients everywhere and can thus be backpropagated during the backward pass. This also highlights the power of Dynamic Computational Graphs (DCGs) and, more generally, *differentiable programming*: the computational graph does not have to be defined statically and is not necessarily the same for each learning iteration.

The reserve layer follows a similar approach but it is more involved. During the forward phase, the proxy first partitions the generators into (1) the set \mathcal{G}^\downarrow of units whose power output can be decreased to provide more reserves; and (2) the set \mathcal{G}^\uparrow of units whose power output can be increased without affecting their reserves

$$\mathcal{G}^\uparrow \leftarrow \{g \mid \tilde{p}_g \leq \bar{p}_g - \bar{r}_g\} \quad \mathcal{G}^\downarrow \leftarrow \{g \mid \tilde{p}_g > \bar{p}_g - \bar{r}_g\}.$$

The amount Δ^\downarrow of additional reserves and additional power Δ^\uparrow that can be provided are given by

$$\Delta^\downarrow \leftarrow \sum_{g \in \mathcal{G}^\downarrow} (\bar{p}_g - \bar{r}_g) \quad \Delta^\uparrow \leftarrow \sum_{g \in \mathcal{G}^\uparrow} (\bar{p}_g - \bar{r}_g) - \tilde{p}_g$$

It is then possible to compute the additional reserve amount Δ that is available without violating the balance constraint

$$\Delta \leftarrow \max(0, \min(\Delta_R, \Delta^\uparrow, \Delta^\downarrow))$$

where

$$\Delta_R \leftarrow R - \sum_g \min\{\bar{r}_g, \bar{p}_g - \tilde{p}_g\}$$

denotes the shortage in reserves. The generators can then be scaled proportionally by

$$\check{p}_g = \begin{cases} \tilde{p}_g + \alpha^\uparrow(\bar{p}_g - \bar{r}_g - \tilde{p}_g) & \forall g \in \mathcal{G}^\uparrow \\ \tilde{p}_g - \alpha^\downarrow(\tilde{p}_g - \bar{p}_g + \bar{r}_g) & \forall g \in \mathcal{G}^\downarrow \end{cases} \quad (6)$$

where $\alpha^\uparrow \leftarrow \Delta/\Delta^\uparrow$ and $\alpha^\downarrow \leftarrow \Delta/\Delta^\downarrow$. Again, the definition of \check{p}_g for each generator is differentiable. Moreover, these two layers are guaranteed to find a feasible solution if one exists.

The overall architecture (see Figure 5), called E2ELR, is a differentiable program: it can be trained end to end efficiently and runs in milliseconds during inference due to these dedicated repair layers.

System	Ref	$ \mathcal{N} $	$ \mathcal{E} $	$ \mathcal{G} $	D_{ref}^{\dagger}	α_r
ieee300	[42]	300	411	69	23.53	34.16%
pegase1k	[22]	1354	1991	260	73.06	19.82%
rte6470	[22]	6470	9005	761	96.59	14.25%
pegase9k	[22]	9241	16049	1445	312.35	4.70%
pegase13k	[22]	13659	20467	4092	381.43	1.32%
goc30k	[17]	30000	35393	3526	117.74	4.68%

Table 1: Selected test cases from PGLib [6]

5.3 Self-Supervised Learning

Primal optimization proxies also offer an intriguing alternative to supervised learning. Indeed, the approaches presented so far rely on the availability of the data set $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i = \Phi(\mathbf{x}_i))\}_{i \in [n]}$. However, since the parametric optimization problem is available in explicit form, it is possible to define a *self-supervised* version of the learning task that only uses the data set \mathcal{I} . Given a parametric primal optimization proxy Φ_θ^\uparrow , the self-supervised learning task amounts to solving the optimization problem

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n f_{\mathbf{x}_i}(\Phi_\theta^\uparrow(\mathbf{x}_i))$$

which is expressed in terms of the objective function $f_{\mathbf{x}}$ of the original parametric optimization problem. Note also that the learning task is different from the original optimization problem (1): the optimization is over the learnable parameters, not the decision variables of (1).

For instance, E2ELR can be trained end-to-end using self-supervised learning and the loss function

$$c(\check{\mathbf{p}}) + M_{\text{th}} \|\check{\xi}_{\text{th}}\|_1$$

which is then backpropagated through the repair layers to adjust the parameters θ of the neural network. The training of a self-supervised E2ELR does not require any solved instance.

This self-supervised approach has a significant benefit: it does not rely on the availability of the optimal solutions for a large number of instances, which may be costly to obtain in practice. In addition, the objective function in supervised learning approaches may not be perfectly aligned with the objective of the original optimization problem, reducing the accuracy of the learning step. It should be noted that supervised learning can also be used, say with the LD approach proposed earlier. However, this provides less guidance to the learning system than the labeled data in general, and the machine learning model typically trade increased violations for better objectives. Early papers on self-supervised learning for optimization can be found in (e.g., [9, 12, 21, 36, 46]).

5.4 Experimental Results

This section demonstrates the capabilities of primal optimization proxies, and E2ELR in particular, on power grids with up to 30,000 buses. All details can be found in [10].

5.4.1 Data Generation

To obtain reasonable training, validation, and testing data sets, benchmarks from the PGLib [6] library (v21.07) were modified by perturbing the loads. Denote by \mathbf{d}^{ref} the nodal load vector from the benchmarks. Load i becomes $\mathbf{d}^{(i)} = \gamma^{(i)} \times \eta^{(i)} \times \mathbf{d}^{\text{ref}}$, where $\gamma^{(i)} \in \mathbb{R}$ is a global scaling factor, $\eta \in \mathbb{R}^{|\mathcal{N}|}$ denotes load-level multiplicative white noise, and the multiplications are element-wise. γ is sampled from a uniform distribution $U[0.8, 1.2]$, and, for each load, η is sampled from a log-normal distribution with mean 1 and standard deviation 5%. The PGLib library does not include reserve information: for this reason, the instance generation mimics the contingency reserve requirements used in industry. It assumes $\bar{r}_g = \alpha_r \bar{p}_g$, where $\alpha_r = 5 \times \|\bar{\mathbf{p}}\|_\infty \times \|\bar{\mathbf{p}}\|_1^{-1}$, ensuring a total reserve capacity 5 times larger than the largest generator. The reserve requirements of each instance is sampled uniformly between

100% and 200% of the size of the largest generator. Table 1 presents the resulting systems: it reports the number of buses ($|\mathcal{N}|$), branches ($|\mathcal{E}|$), and generators ($|\mathcal{G}|$), the total active power demand in the reference PGLib case (D_{ref} , in GW), and the value of α_r . Large systems have a smaller value of α_r because they contain significantly more generators, but the size of the largest generator typically remains roughly the same. For every test case, 50,000 instances are generated and solved using Gurobi. This dataset is then split into training, validation, and test sets which comprise 40000, 5000, and 5000 instances.

5.4.2 Baseline Models and Performance Metrics

The experiments compare E2ELR with four baseline models. The simplest is a naïve, fully-connected DNN model without any feasibility layer (DNN): it simply includes a Sigmoid activation layer to enforce the generation bounds. The second baseline is a fully connected DNN model with the DeepOPF architecture [35] (DeepOPF). It uses an equality completion to ensure the satisfaction of equality constraints, but its output may violate inequality constraints. The third baseline is a fully connected DNN model with the DC3 architecture [12] (DC3). This architecture uses a partial repair layer, i.e., a fixed-step unrolled gradient descent to minimize constraint violations. This layer mimics the implicit layer described previously, but only performs a fixed number of gradient steps for efficiency reasons. DC3 is not guaranteed to reach zero violations however. The last model is a fully connected DNN model, combined with the LOOP-LC architecture from [28]. LOOP uses gauge functions to define a one-to-one mapping between the unit hypercube and the set of feasible solutions. The approach applies when all constraints are convex, the feasible set is bounded, and a strictly feasible point is available for each instance. LOOP was only evaluated on ED instances with no reserves, since its gauge mapping does not support the reserve constraint formulation in terms of generation variables only. The baselines are not intended to be exhaustive, but rather to demonstrate the potential of primal optimization proxies. Other approaches are possible (e.g., [24, 28, 41]).

5.4.3 Optimality Gaps

The first results measure the quality of the machine learning models using the optimality gap which is generalized to include penalties for constraint violations for a fair comparison. Given an instance \mathbf{x} with optimal solution \mathbf{p}^* and a prediction $\hat{\mathbf{p}}$, the optimality gap is defined as $\text{gap} = (\hat{Z} - Z^*) \times |Z^*|^{-1}$, where Z^* is the optimal objective value, and \hat{Z} is the objective value of the prediction defined as

$$c(\hat{\mathbf{p}}) + M_{\text{th}}\|\hat{\xi}_{\text{th}}\|_1 + M_{\text{pb}}|\mathbf{e}^\top(\hat{\mathbf{p}} - \mathbf{d})| + M_r\xi_r(\hat{\mathbf{p}}). \quad (7)$$

The paper uses realistic penalty prices, based on the values used in MISO operations [30, 32].

Table 2 reports the optimality gaps for the ED instances and for the same instances with no reserve constraints to evaluate LOOP (ED-NR). Both Supervised Learning (SL) and Self-Supervised Learning (SSL) are considered. Bold entries denote the best performing method. The results show *E2ELR has the smallest optimality gaps across all settings*, primarily because it always returns feasible solutions and avoids large penalties. The performance of LOOP on the ED-NR is not strong, although it produces feasible solutions: its non-convex gauge mapping guides the learning to a local optimum after a few epochs. Statistics on power balance violations for DNN, DeepOPF, and DC3 are reported in Table 3. E2ELR with self-supervised training is particularly effective. With the exception of `rte6470` (the real French transmission grid), its performance improves with system sizes.

5.4.4 Computing Times

The second set of results evaluates the computational efficiency of each ML model. The computational efficiency is measured by (i) the training time of ML models, including the data generation time when applicable, and (ii) the inference time. Note that ML models evaluate *batches* of instances, and inference times are reported per batch of 256 instances. Unless specified otherwise, average computing times are arithmetic means; other averages use shifted geometric means with a shift s of 1% for optimality gaps, and 1 p.u. for constraint violations.

Loss	System	ED-NR					ED			
		DNN	E2ELR	DeepOPF	DC3	LOOP	DNN	E2ELR	DeepOPF	DC3
SL	ieee300	69.55	1.42	2.81	3.03	38.93	75.06	1.52	2.80	2.94
	pegase1k	48.77	0.74	7.78	2.80	32.53	47.84	0.74	7.50	2.97
	rte6470	55.13	1.35	28.23	3.68	50.21	70.57	1.82	46.66	3.49
	pegase9k	76.06	0.38	33.20	1.25	33.78	81.19	0.38	30.84	1.29
	pegase13k	71.14	0.29	64.93	1.79	52.94	76.32	0.28	69.23	1.81
	goc30k	194.13	0.46	55.91	2.75	36.49	136.25	0.45	41.34	2.35
SSL	ieee300	35.66	0.74	2.23	2.51	37.78	45.56	0.78	2.82	2.80
	pegase1k	62.07	0.63	10.83	2.57	32.20	64.69	0.68	9.83	2.61
	rte6470	40.73	1.30	42.28	2.82	50.20	55.16	1.68	48.57	3.04
	pegase9k	43.68	0.32	34.33	0.82	33.76	44.74	0.29	42.06	0.93
	pegase13k	57.58	0.21	60.12	0.84	52.93	61.28	0.19	65.38	0.91
	goc30k	108.91	0.39	8.39	0.72	36.73	93.91	0.33	9.47	0.71

Table 2: Mean Optimality Gaps (%) (from [9]).

Loss	System	DNN		DeepOPF		DC3	
		%feas	viol	%feas	viol	%feas	viol
SL	ieee300	0%	0.70	100%	0.06	100%	0.25
	pegase1k	0%	1.90	70%	1.19	65%	0.08
	rte6470	0%	2.26	1%	2.81	7%	0.01
	pegase9k	0%	5.91	3%	5.99	29%	0.00
	pegase13k	0%	7.79	1%	7.46	22%	0.00
	goc30k	0%	2.31	54%	1.54	69%	0.00
SSL	ieee300	0%	0.73	100%	0.56	100%	0.29
	pegase1k	0%	2.30	63%	1.81	39%	0.05
	rte6470	0%	2.72	1%	2.51	5%	0.01
	pegase9k	0%	7.00	3%	7.87	25%	0.08
	pegase13k	0%	6.82	1%	7.83	20%	0.01
	goc30k	0%	2.61	49%	1.20	62%	0.02

Table 3: Violations of the Power Balance Constraint (from [9]).

Table 4 reports the sampling and training times. The sampling times represent the total time to obtain the optimal solutions to all instances (using the Gurobi solver on a single thread). The training times for SL and SSL are comparable but SSL has the significant benefit of not requiring optimal solutions to be produced offline. The training time of DC3 is significantly higher than other baselines because of its unrolled gradient steps: this is the issue mentioned earlier on the cost of using complex implicit layers. The main takeaway is that *the self-supervised E2ELR needs less than an hour of training to achieve optimality gaps under 0.5% for grids with thousands of buses.*

Table 5 report the average solving time using Gurobi (GRB) and the average inference times of the ML methods. The ML inference times are reported for a batch of 256 instances on a GPU. The number of gradient steps used by DC3 to recover feasibility is set to 200 for inference (compared to 50 for training). For systems with more than 6,000 buses, DC3 is typically 10–30 times slower than other baselines, due to its unrolled gradient steps. In contrast, DNN, DeepOPF, and E2ELR all require about 5–10 milliseconds to evaluate a batch of 256 instances. For the largest systems, this represents about 25,000 instances per second on a single GPU. Solving the same volume of instances would take Gurobi more than a day on a single CPU.

5.5 The Impact of Optimization Proxies

Optimization proxies have the potential to transform applications as they bring orders of magnitude improvements in efficiency. This section highlights these potential benefits on an important topic:

Loss	System	Sampling	DNN	E2ELR	DeepOPF	DC3
SL	<code>ieee300</code>	0.2 h	12 min	43 min	43 min	115 min
	<code>pegase1k</code>	0.8 h	14 min	19 min	19 min	53 min
	<code>rte6470</code>	4.6 h	14 min	19 min	19 min	71 min
	<code>pegase9k</code>	14.0 h	15 min	22 min	22 min	123 min
	<code>pegase13k</code>	22.7 h	16 min	27 min	27 min	126 min
	<code>goc30k</code>	65.9 h	32 min	39 min	38 min	129 min
SSL	<code>ieee300</code>	–	21 min	37 min	37 min	131 min
	<code>pegase1k</code>	–	6 min	19 min	19 min	67 min
	<code>rte6470</code>	–	12 min	21 min	21 min	71 min
	<code>pegase9k</code>	–	20 min	24 min	24 min	123 min
	<code>pegase13k</code>	–	13 min	22 min	22 min	125 min
	<code>goc30k</code>	–	52 min	53 min	53 min	128 min

Table 4: Sampling and Training Times (from [9]).

Loss	System	DNN	E2ELR	DeepOPF	DC3	GRB
SL	<code>ieee300</code>	3.9 ms	6.5 ms	4.6 ms	16.5 ms	12.6 ms
	<code>pegase1k</code>	4.5 ms	6.0 ms	4.8 ms	18.9 ms	56.5 ms
	<code>rte6470</code>	5.7 ms	10.4 ms	6.2 ms	36.1 ms	333.6 ms
	<code>pegase9k</code>	6.3 ms	7.7 ms	6.7 ms	91.6 ms	1008.0 ms
	<code>pegase13k</code>	8.3 ms	10.7 ms	8.8 ms	531.2 ms	1632.7 ms
	<code>goc30k</code>	9.3 ms	11.1 ms	10.6 ms	438.7 ms	4745.7 ms
SSL	<code>ieee300</code>	3.9 ms	7.6 ms	4.4 ms	17.6 ms	12.6 ms
	<code>pegase1k</code>	4.4 ms	5.9 ms	4.7 ms	19.1 ms	56.5 ms
	<code>rte6470</code>	6.4 ms	10.5 ms	6.7 ms	37.3 ms	333.6 ms
	<code>pegase9k</code>	7.1 ms	8.3 ms	7.2 ms	92.9 ms	1008.0 ms
	<code>pegase13k</code>	7.8 ms	8.9 ms	7.9 ms	522.4 ms	1632.7 ms
	<code>goc30k</code>	10.2 ms	12.4 ms	10.3 ms	435.8 ms	4745.7 ms

Table 5: Solving and Inference Times.

real-time risk assessment. Given the significant growth in volatility coming from renewable sources of energy, the electrification of the transportation infrastructure, and data centers, operators are increasingly worried about the operational and financial risks in their grids. They are interested in dashboards that give them the ability to quantify these risks in real time. Consider, for instance, the real-time risk-assessment framework described in Figure 6, which runs a collection of Monte-Carlo scenarios for the next 24 hours. These scenarios are obtained using forecasting methods (e.g., temporal fusion transformers). The evaluation of each scenario requires 288 optimizations, i.e., one ED every 5 minutes for the next 24 hours. Solving these 288 optimizations takes about 15 minutes, and a real-time risk assessment dashboard would need to evaluate thousands of them.

Optimization proxies are the key technology enabler to make such a real-time risk assessment platform a reality. When primal optimization proxies replace the optimizations, every scenario is evaluated in about 5 seconds [9], making it possible to quantify asset-level, system-level, and financial risks in real time using GPUs. For instance, Figure 7 reports the probability of an adverse event for the next 24 hours using E2ELR. It also compares E2ELR with other learning architectures (e.g., DeepOPF, DC3) and the ground truth, i.e., the risk assessment using a state-of-the-art optimization solver. The left figure depicts the probability of an adverse event on the balance constraint: both the ground truth and E2ELR show no such event. In contrast, the other methods systematically report potential adverse events, since they cannot guarantee feasibility of that constraint. The right figure shows the violations of the thermal limit of a congested line. Again, E2ELR isolates the potential violations with great accuracy. Other methods report many false positives throughout the day.

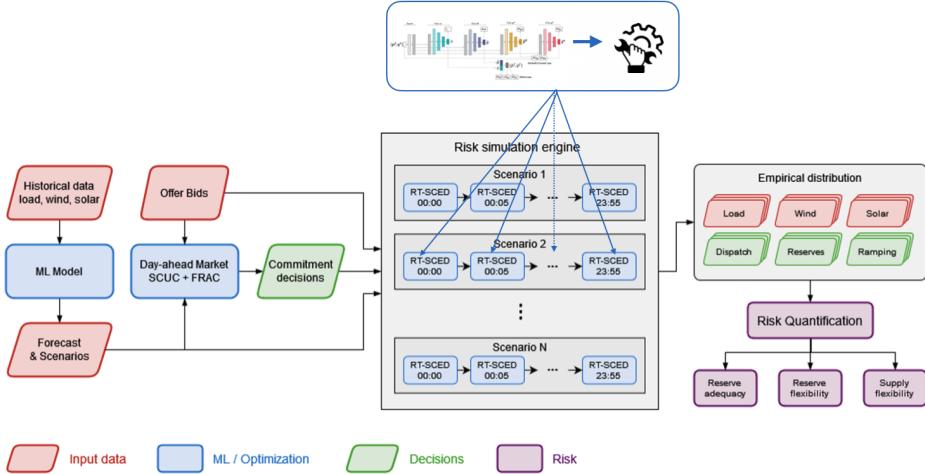


Figure 6: A Risk Assessment Framework with Optimization Proxies (from [11]).

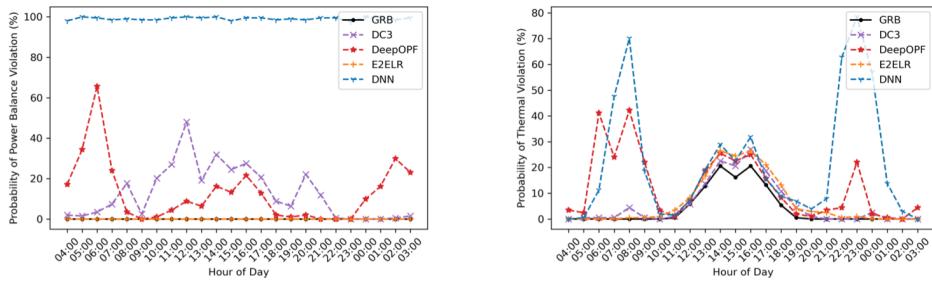


Figure 7: Probability of an Adverse Event with a Proxy-based Risk Assessment (from [11]).

6 Dual Optimization Proxies

Primal optimization proxies produce (high-quality) feasible solutions in milliseconds. However, an ideal outcome in optimization practice is to produce a pair of primal and dual solutions with a small duality gap. The counterpart for optimization learning would be to produce a pair of primal and dual proxies that would produce both a feasible solution and an assessment of its quality in milliseconds. This section describes a methodology to obtain dual optimization proxies for many practical applications. To convey the intuition underlying the methodology, consider the parametric optimization program and its dual

$$\begin{array}{ll} \min & \mathbf{c}_x^T \mathbf{y} \\ \text{s.t.} & \mathbf{A}_x \mathbf{y} = \mathbf{b}_x \\ & \mathbf{l}_x \leq \mathbf{y} \leq \mathbf{u}_x \end{array} \quad \begin{array}{ll} \max & \mathbf{b}_x^T \mathbf{z} + \mathbf{l}_x^T \mathbf{z}_l - \mathbf{u}_x^T \mathbf{z}_u \\ \text{s.t.} & \mathbf{z} \mathbf{A}_x + \mathbf{z}_l - \mathbf{z}_u = \mathbf{c}_x \\ & \mathbf{z}_l, \mathbf{z}_u \geq 0 \end{array}$$

The key insight comes from the fact that the bound constraints $\mathbf{l}_x \leq \mathbf{y} \leq \mathbf{u}_x$ in the primal model gives a simple way to find a dual feasible solution: obtain a prediction $\check{\mathbf{z}}$ for variables \mathbf{z} and use the free dual variables \mathbf{z}_l and \mathbf{z}_u to obtain a dual feasible solution $(\check{\mathbf{z}}, \check{\mathbf{z}}_l, \check{\mathbf{z}}_u)$. These primal bound constraints are natural in most engineering applications: for instance, generators in power systems have physical limits, and the number of trailers in a supply chain is finite.

Figure 8 provides the general structure of dual optimization proxies. Given decision variables $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2)$, a dual proxy consists of a parametric machine learning model \mathcal{M}_θ to obtain a prediction $\check{\mathbf{z}}_1$ for a subset of the decision variables, and a *completion* layer \mathcal{C} to assign the remaining variables $\mathbf{z}_2 = \check{\mathbf{z}}_2$ and deliver a feasible dual solution $\check{\mathbf{z}} = (\check{\mathbf{z}}_1, \check{\mathbf{z}}_2)$. The composition $\mathcal{C} \circ \mathcal{M}_\theta$ is a parametric dual optimization proxy.

Consider now a dual optimization proxy for parametric linear programming. Given a completion layer \mathcal{C}^{lp} , the composition $\mathcal{C}^{lp} \circ \mathcal{M}_\theta$ is a parametric dual optimization proxy that can be trained by

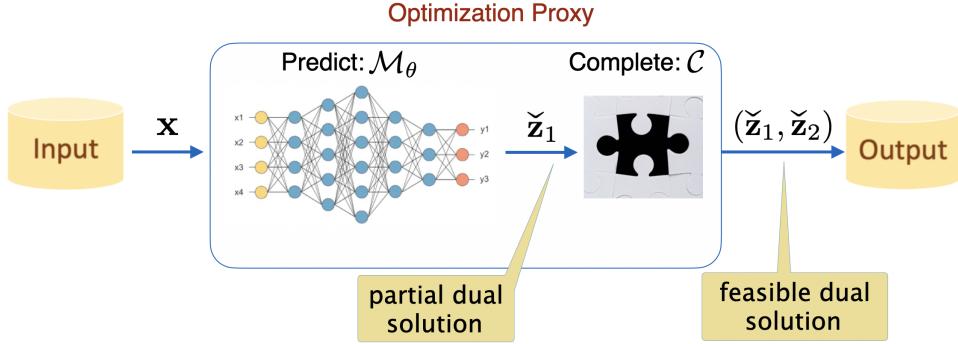


Figure 8: The Structure of Dual Optimization Proxies.

the following optimization:

$$\operatorname{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^n f_{\mathbf{x}_i}((\mathcal{C}^{lp} \circ \mathcal{M}_{\theta})(\mathbf{x}_i)).$$

where $f_{\mathbf{x}_i}((\mathbf{z}, \mathbf{z}_l, \mathbf{z}_u)) = \mathbf{b}_{\mathbf{x}_i}^T \mathbf{z} + \mathbf{l}_{\mathbf{x}_i}^T \mathbf{z}_l - \mathbf{u}_{\mathbf{x}_i}^T \mathbf{z}_u$.

It remains to derive the completion layer \mathcal{C}^{lp} . Given a prediction $\check{\mathbf{z}}^t = \mathcal{M}_{\theta^t}(\mathbf{x})$ at step t , the dual problem becomes

$$\begin{aligned} \max \quad & \mathbf{l}_{\mathbf{x}}^T \mathbf{z}_l^t - \mathbf{u}_{\mathbf{x}}^T \mathbf{z}_u^t \\ \text{s.t.} \quad & \mathbf{z}_l^t - \mathbf{z}_u^t = \mathbf{c}_{\mathbf{x}} - \check{\mathbf{z}}^t \mathbf{A}_{\mathbf{x}} \\ & \mathbf{z}_l^t, \mathbf{z}_u^t \geq 0 \end{aligned}$$

The optimal solution to this problem has a closed form [23]:

$$\begin{aligned} \check{\mathbf{z}}_l^t &= |\mathbf{c}_{\mathbf{x}} - \check{\mathbf{z}}^t \mathbf{A}_{\mathbf{x}}|^+ \\ \check{\mathbf{z}}_u^t &= |\mathbf{c}_{\mathbf{x}} - \check{\mathbf{z}}^t \mathbf{A}_{\mathbf{x}}|^- \end{aligned}$$

where $|v|^+ = \max(0, v)$, $|v|^- = \max(0, -v)$, and $(\check{\mathbf{z}}^t, \check{\mathbf{z}}_l^t, \check{\mathbf{z}}_u^t)$ is a feasible solution to the dual problem. Hence, the completion layer \mathcal{C}^{lp} for parametric linear programs is defined as

$$\mathcal{C}^{lp}(\check{\mathbf{z}}) = (\check{\mathbf{z}}, |\mathbf{c}_{\mathbf{x}} - \check{\mathbf{z}} \mathbf{A}_{\mathbf{x}}|^+, |\mathbf{c}_{\mathbf{x}} - \check{\mathbf{z}} \mathbf{A}_{\mathbf{x}}|^-).$$

The forward pass of the dual proxy $\mathcal{C}^{lp} \circ \mathcal{M}_{\theta}$ can be evaluated efficiently. Moreover, the dual proxy admits subgradients everywhere, enabling a fast backward pass as well.

Dual optimization proxies have been proposed first for the Second-Order Cone Programming (SOCP) relaxation of the AC power flow equations [38]. The completion layer is particularly interesting in that paper: it uses properties of the dual optimal solutions to determine how to complete the set of dual variables not predicted by the neural network. Again, the dual optimization proxy is a differentiable program that can be trained end-to-end to produce feasible solutions at training and inference times. Experimental results have shown that the resulting proxies can find near-optimal dual feasible solutions. Dual optimization proxies for conic optimization have been studied extensively in [39]. More generally, a methodology to find quality guarantees for an optimization problem Φ can be summarized as follows:

1. find a conic approximation Ψ to problem Φ ;
2. obtain the dual Ψ_d of Ψ ;
3. derive a dual optimization proxy Ψ_d^\downarrow for Ψ_d .

6.1 Experimental Results

This section presents some experimental result on the self-supervised Dual Optimal Proxy for Linear Programming (DOPLP) applied to the DC-OPF problem. The full details and results can be found

Model 2 The DC-OPF Model.

$$\operatorname{argmin}_{\mathbf{p}, \mathbf{f}} \mathbf{c}^\top \mathbf{p} \quad (9a)$$

$$s.t. \quad \mathbf{e}^\top \mathbf{p} = \mathbf{e}^\top \mathbf{d} \quad (9b)$$

$$\mathbf{f} = \mathbf{PTDF}(\mathbf{p} - \mathbf{d}) \quad (9c)$$

$$\underline{\mathbf{p}} \leq \mathbf{p} \leq \bar{\mathbf{p}} \quad (9d)$$

$$\underline{\mathbf{f}} \leq \mathbf{f} \leq \bar{\mathbf{f}} \quad (9e)$$

Benchmark	m	n	h	$ \theta $
1354_pegase	1992	2251	2048	16.6M
2869_pegase	4583	5092	4096	71.1M
6470_rte	9006	9766	8192	281M

Table 6: DCOPF Benchmark Summary

Benchmark	Minimum	Geometric Mean	99 th Percentile	Maximum
1354_pegase	0.14%($\pm .014\%$)	0.23%($\pm .016\%$)	2.15%($\pm .030\%$)	13.76%($\pm .015\%$)
2869_pegase	0.15%($\pm .018\%$)	0.25%($\pm .106\%$)	3.73%($\pm .194\%$)	10.26%($\pm .513\%$)
6470_rte	0.15%($\pm .013\%$)	0.50%($\pm .146\%$)	1.96%($\pm .615\%$)	6.95%($\pm .633\%$)

Table 7: Dual Gap Ratio (adapted from [23]).

in [23]. The DC-OPF formulation is shown in Model 2 It is close to the ED: it does not have reserve constraints but its line constraints are hard.

DOPLP uses a 3-layer fully connected neural network with ReLU activations. The training uses 10,000 feasible instances and 2,500 instances for validation. The testing set is the same set of 5,000 feasible instances. The experiments are repeated 10 times with different seeds to ensure results do not depend on a particular training/validation split or initial neural network weights. All training was run on NVIDIA Tesla V100 16GB GPUs.

DOPLP is evaluated on three large-scale DC Optimal Power Flows (DCOPF) instances from the PGLib library [6]: 1354_pegase, 2869_pegase, and 6470_rte. The instances are generated as in Section 5.4, except for 6470_rte where U is set to 1.05 instead, since the given instance is already congested. Table 6 reports the number of constraints m , the number of variables n , the hidden layer size h , and the total number of parameters $|\theta|$ in the neural network \mathcal{M}_θ for each benchmark. For the purposes of assessing the quality of the results, each instance is solved using Mosek 10.1.24 [33] to obtain the optimal dual solution \mathbf{y}^* .

Table 7 shows the minimum, geometric mean, 99th percentile, and maximum dual gap ratio over the testing set samples. Each entry in the table is the geometric mean \pm the standard deviation over the ten trials. *DOPLP produces mean optimality gaps in the range of [0.25, 0.5]%, showing that DOPLP is capable of producing dual feasible solutions with extremely tight dual bounds for large-scale DCOPF problems.*

7 Primal-Dual Learning

Another approach to find feasible solutions is to develop optimization proxies that adapt traditional optimization algorithms to the learning context. Consider the constrained optimization problem

$$\operatorname{argmin}_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) \text{ subject to } \mathbf{h}_{\mathbf{x}}(\mathbf{y}) = 0.$$

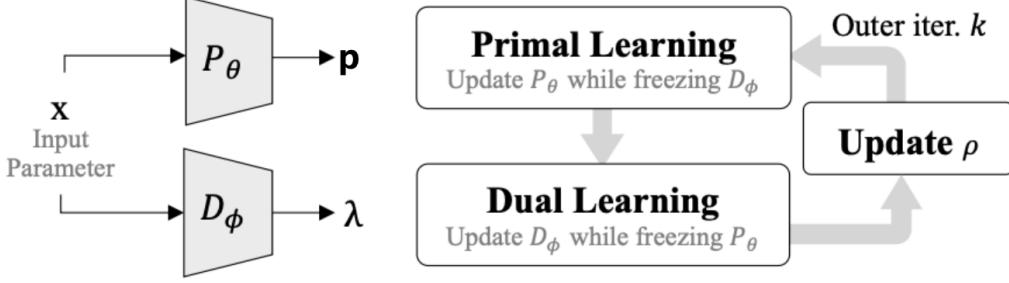


Figure 9: The Architecture of Primal-Dual Learning (from [36]).

The *Augmented Lagrangian Method* (ALM) solves unconstrained optimization problems of the form

$$\operatorname{argmin}_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \boldsymbol{\lambda}^T \mathbf{h}_{\mathbf{x}}(\mathbf{y}) + \frac{\rho}{2} \mathbf{1}^T \mathbf{h}_{\mathbf{x}}(\mathbf{y})^2 \quad (10)$$

where ρ is a penalty coefficient and $\boldsymbol{\lambda}$ are the Lagrangian multiplier approximations. These multipliers are updated using the rule

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \rho \mathbf{h}_{\mathbf{x}}(\mathbf{y}). \quad (11)$$

Primal-Dual Learning (PDL) [36] is a self-supervised method that jointly learns two neural networks: a primal neural network P_θ that learns the input/output mapping of the ALM unconstrained optimizations and a dual network D_ϕ that learns the dual updates. The overall architecture of PDL is shown in Figure 9. At each iteration, the *primal learning* step updates the learning parameters θ of the primal network while keeping the dual network D_ϕ fixed. After completion of the primal learning, PDL applies a *dual learning* step that updates the learnable parameters ϕ of the dual network D_ϕ .

More precisely, at step t , PDL has the values θ^t and ϕ^t for the primal and dual learning parameters and seeks to improve them into θ^{t+1} and ϕ^{t+1} . The primal learning problem of step t is the optimization

$$\theta^{t+1} = \operatorname{argmin}_{\theta} \frac{1}{n} \sum_{i \in [n]} f_{\mathbf{x}_i}(\mathbf{y}_i) + D_{\phi^t}(\mathbf{x}_i)^T \mathbf{h}_{\mathbf{x}_i}(\mathbf{y}_i) + \frac{\rho}{2} \mathbf{1}^T \mathbf{h}_{\mathbf{x}_i}(\mathbf{y}_i)^2, \quad (12)$$

where $\mathbf{y}_i = P_\theta(\mathbf{x}_i)$. The dual learning problem is the optimization

$$\phi^{t+1} = \operatorname{argmin}_{\phi} \frac{1}{n} \sum_{i \in [n]} \|D_\phi(\mathbf{x}_i) - D_{\phi^t}(\mathbf{x}_i) + \rho \mathbf{h}_{\mathbf{x}_i}(P_{\theta^{t+1}}(\mathbf{x}_i))\|. \quad (13)$$

These two steps, i.e., the training of the primal and dual networks that mimic the ALM, are iterated in sequence until convergence. To handle severe violations, each iteration may increase the penalty coefficient ρ . At iteration t , this update uses the maximum violation v^{t+1}

$$v^{t+1} = \max_{i \in [n]} \{\|\mathbf{h}_{\mathbf{x}}(P_{\theta^{t+1}}(\mathbf{x}_i))\|_\infty\}. \quad (14)$$

The penalty coefficient increases when the maximum violation v^{t+1} is greater than a tolerance value τ times the maximum violation from the previous iteration v^t , i.e.,

$$\rho \leftarrow \min\{\alpha\rho, \rho_{\max}\} \text{ if } v^{t+1} > \tau v^t, \quad (15)$$

where $\tau \in (0, 1)$ is the tolerance, $\alpha > 1$ is an update multiplier, and ρ_{\max} is an upper bound on the penalty coefficient. The PDL algorithm is summarized in Algorithm 1.

There is a fundamental difference between PDL and the Lagrangian dual method presented in Section 4. In these Lagrangian methods, the constraint multipliers used to balance feasibility and optimality are not instance-specific: they are aggregated for each constraint over all the instances. This aggregation limits their capability to ensure feasibility. In contrast, by jointly learning the primal and dual networks, PDL is capable of using *instance-specific multipliers* for penalizing constraints. As

Algorithm 1 The Primal-Dual Learning Algorithm (PDL).

Parameter: Initial penalty coefficient ρ , Maximum outer iteration T , Penalty coefficient updating multiplier α , Violation tolerance τ , Upper penalty coefficient safeguard ρ_{\max}

Input: Input Distribution \mathcal{P}

Output: learned primal and dual nets P_θ, D_ϕ

- 1: Initialize θ^1 and ϕ^1 randomly;
 - 2: **for** $t \in [T]$ **do**
 - 3: Train the primal network using Equation (12)
 - 4: Train the dual network using Equation (13)
 - 5: Update ρ using Equation (15)
 - 6: **end for**
 - 7: **return** $P_\theta = P_{\theta^{T+1}}$ and $D_\phi = D_{\phi^{T+1}}$
-

a result, during each iteration, the training process learns the primal and dual iteration points of the ALM algorithm. Eventually, these iteration points, and hence the primal and dual networks, are expected to converge to the primal and dual solutions of the optimization problem.

Primal-Dual Optimization proxies have the ability to deploy optimization models that would otherwise be too complex to meet real-time requirements. By shifting most of the computational burden offline during training, optimization proxies provide a way to produce high-quality solutions to optimization models that cannot be solved fast enough to meet real-time constraints. An interesting example is the Security-Constrained Optimal Power Flow problem (SCOPF) that captures the automatic primary response of generators in case of transmission line or generator contingencies. The next subsection briefly describes this application in some detail. The presentation follows [37], where the full results and technical description can be found. An interesting feature of this Primal-Dual Learning application is that it mimics the Column and Constraint Generation Algorithm (CCGA) in [44, 45], which is the state of the art for the SCOPF.

7.1 The SCOPF Problem

Model 3 presents the extensive formulation of the SCOPF with $N - 1$ generator and line contingencies. The formulation is similar in spirit to the ED, but it replaces the reserve constraints with actual generator and line contingencies. The primary objective of the SCOPF is to determine the generator setpoints for the base case while ensuring the feasibility of each generator and line contingencies. In other words, even if a generator or line fails, the base setpoints should provide enough flexibility to be transformed into a feasible solution for the contingency. The objective (16) sums the linear cost of the base case dispatch \mathbf{p} and the penalties for violating the thermal limits in the base case and in the contingencies.

The base case includes Constraints (17), (18), and (19). They include the hard power balance constraints in the base case, the soft thermal limits, and the generation limits. Each generator contingency imposes Constraints (20), (21), and (22) to enforce the power balance, the soft thermal limits, and the generation bounds under the generator contingency. The only difference from the base case is Constraint (23) that specifies that generator k should remain inactive under its contingency.

It is important to model accurately how the generators react after a contingency. In Model 3, their response follows an Automatic Primary Response (APR) control mechanism [3, 13], as shown in Constraints (24)–(26). The formulation here is based on the APR model used in [44, 45] where, for each generator contingency k , a system-wide signal $n_k \in [0, 1]$ represents the level of system response required to resolve the power imbalance. The APR control also ensures that the change in dispatch under a contingency is proportional to the droop slope, which is determined by the product of the generator capacity \ddot{p} and the predefined parameter γ as in [45]. The generator capacities are defined as $\ddot{p} = \bar{\mathbf{p}} - \mathbf{p}$. The APR constraints ensure that the generation dispatch under generator contingency remains within the generation limits, leading to the following generator response:

$$p_{k,i} = \min\{p_i + n_k \gamma_i \ddot{p}_i, \bar{p}_i\}, \quad \forall i \in \mathcal{G}, \forall k \in \mathcal{K}_g, i \neq k. \quad (31)$$

Model 3 The Extensive SCOPF Formulation.

$$\begin{aligned}
& \min_{\mathbf{p}, [\mathbf{p}_k, \rho_k, n_k]_{k \in \mathcal{K}_g}, [\xi_k]_{k \in \{0\} \cup \mathcal{K}_g \cup \mathcal{K}_e}} \mathbf{c}^\top \mathbf{p} + M_\xi \left(\sum_{k \in \{0\} \cup \mathcal{K}_g \cup \mathcal{K}_e} \|\xi_k\|_1 \right) & (16) \\
& \text{s. t.: } \mathbf{1}^\top \mathbf{p} = \mathbf{1}^\top \mathbf{d} & (17) \\
& \underline{\mathbf{f}} - \xi_0 \leq \mathbf{f} = \mathbf{PTDF}(\mathbf{d} - \mathbf{p}) \leq \bar{\mathbf{f}} + \xi_0 & (18) \\
& \underline{\mathbf{p}} \leq \mathbf{p} \leq \bar{\mathbf{p}} & (19) \\
& \mathbf{e}^\top \mathbf{p}_k = \mathbf{e}^\top \mathbf{d} & \forall k \in \mathcal{K}_g & (20) \\
& \underline{\mathbf{f}} - \xi_k \leq \mathbf{f}_k = \mathbf{PTDF}(\mathbf{d} - \mathbf{p}_k) \leq \bar{\mathbf{f}} + \xi_k & \forall k \in \mathcal{K}_g & (21) \\
& \underline{p}_i \leq p_{ki} \leq \bar{p}_i & \forall i \in \mathcal{G}, \forall k \in \mathcal{K}_g, i \neq k & (22) \\
& p_{k,k} = 0 & \forall k \in \mathcal{K}_g & (23) \\
& |p_{k,i} - p_i - n_k \gamma_i \ddot{p}_i| \leq \ddot{p}_i \rho_{k,i} & \forall i \in \mathcal{G}, \forall k \in \mathcal{K}_g, i \neq k & (24) \\
& p_i + n_k \gamma_i \ddot{p}_i \geq \ddot{p}_i \rho_{k,i} + \underline{p}_i & \forall i \in \mathcal{G}, \forall k \in \mathcal{K}_g, i \neq k & (25) \\
& p_{k,i} \geq \ddot{p}_i \rho_{k,i} + \underline{p}_i & \forall i \in \mathcal{G}, \forall k \in \mathcal{K}_g, i \neq k & (26) \\
& \underline{\mathbf{f}} - \xi_k \leq \mathbf{f} + f_k \mathbf{L}_k \leq \bar{\mathbf{f}} + \xi_k & \forall k \in \mathcal{K}_e & (27) \\
& \xi_k \geq 0 & \forall k \in \{0\} \cup \mathcal{K}_g \cup \mathcal{K}_e & (28) \\
& n_k \in [0, 1] & \forall k \in \mathcal{K}_g & (29) \\
& \rho_{k,i} \in \{0, 1\} & \forall i \in \mathcal{G}, \forall k \in \mathcal{K}_g, i \neq k & (30)
\end{aligned}$$

The binary variables $\rho_{k,i}$ implement the disjunctive constraint (31) for all $i \in \mathcal{G}$ and $k \in \mathcal{K}_g$ such that $k \neq i$. Line contingencies are handled by Constraints (27). These constraints ensure that, during a line contingency, there is an immediate redistribution of power flow specified by the Line Outage Distribution Factor (LODF) [18, 40]. Under line contingency k , the k -th column vector \mathbf{L}_k of the LODF matrix \mathbf{L} , delineates the redistribution of base case power flow f_k at line k to the other lines to ensure that there is no power flow at line k . Note that Model 3 is a large-scale MIP model.

7.2 Primal-Dual Learning for SCOPF

This section describes PDL-SCOPF, i.e., the Primal-Dual Learning of large-scale SCOPFs. The primal variables to estimate are $\mathbf{y} := \mathbf{p}, \{\mathbf{p}_k, n_k, \rho_k\}_{k \in \mathcal{K}_g}, \{\xi_k\}_{\{0\} \cup \mathcal{K}_g \cup \mathcal{K}_e}$, the objective function $f_{\mathbf{x}}(\mathbf{y})$ is the original objective function (16) of Model 3, and the constraints $\mathbf{h}_{\mathbf{x}}(\mathbf{y})$ capture the power balance equations (20) for the generator contingencies. Figure 10 provides a schematic representation of the primal and dual networks which, given the input configuration vector \mathbf{x} , estimate the primal and dual solutions for SCOPF. The design of the primal learning network of PDL-SCOPF has two notable features: (1) a repair layer for restoring the power balance of the base case like in E2ELR, and (2) a binary search layer to estimate the generator dispatches in the generator contingencies inspired by the CCGA.

7.3 The Primal and Dual Networks

The primal network estimates the nominal dispatch, the contingency dispatches, and the slacks of the thermal constraints. As shown in Figure 2, it uses three main components: (1) a fully connected layer with a sigmoid layer that produces a first approximation $\hat{\mathbf{p}}$ of the base dispatch; (2) a repair layer that produces a feasible dispatch $\check{\mathbf{p}}$ to the base dispatch; and (3) a binary search layer that computes an approximation $\tilde{\mathbf{p}}_k$ of dispatch for contingency k by mimicking the binary search of the CCGA. *These three components, and the computation of the constraint slacks, constitute a differentiable program for*

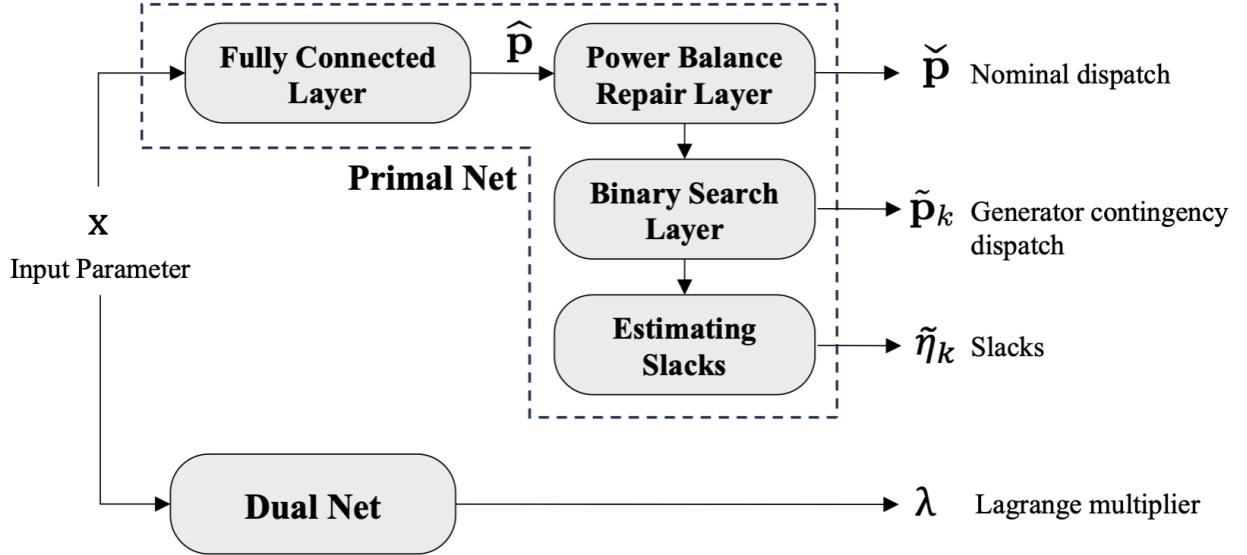


Figure 10: The Primal and Dual Networks of PDL-SCOPF (Adapted from [37]).

Algorithm 2 The Binary Search Layer BSLayer(\mathbf{p}, k) (Adapted from [37]).

Parameter: Maximum iteration t

Initialize: $n_k = 0.5$, $n_{\min} = 0$, $n_{\max} = 1$

```

1: for  $j = 0, 1, \dots, t$  do
2:    $p_{k,i}^{(j)} \leftarrow \min\{p_i + n_k \gamma_i \ddot{p}_i, \bar{p}_i\}$ ,  $\forall i \in \mathcal{G}$ 
3:    $p_{k,k}^{(j)} \leftarrow 0$ 
4:    $e_k \leftarrow \mathbf{1}^\top \mathbf{p}_k^{(j)} - \mathbf{1}^\top \mathbf{d}$ 
5:   if  $e_k > 0$  then:  $n_{\max} \leftarrow n_k$ 
6:   else:  $n_{\min} \leftarrow n_k$ 
7:    $n_k \leftarrow 0.5(n_{\max} + n_{\min})$ 
8: end for
9: for  $i \in \mathcal{G}$  do
10:   if  $p_i + n_k \gamma_i \ddot{p}_i > \bar{p}_i$  then:  $\rho_{k,i} \leftarrow 1$ 
11:   else:  $\rho_{k,i} \leftarrow 0$ 
12: end for
13: return  $\mathbf{p}_k^{(j)}, n_k, \boldsymbol{\rho}_k$ 

```

the primal learning step that is trained end-to-end.

Estimating the dispatches for all $N-1$ generator contingencies presents a computational challenge, as the number of binary variables grows quadratically with the number of generator contingencies. For this reason, the primal network only predicts the base dispatch and uses a binary search layer, inspired by the CCGA algorithm, to compute the generator dispatches under all contingencies. *In other words, the PDL proxy only predicts the base case dispatch and computes the contingency dispatches through an efficient implicit layer.* The binary search layer is an adaptation of its CCGA counterpart and is described in Algorithm 2. It estimates the dispatches under the generator contingencies (\mathbf{p}_k) and the APR-related variables ($n_k, \boldsymbol{\rho}_k$) from the base dispatches. The algorithm performs a binary search on the global signal n_k in order to try to find contingency dispatches that satisfy the power balance constraint. Contrary to its CCGA counterpart, Algorithm 2 may not always satisfy the power balance constraints in the contingencies for reasons described in [37]; They are however satisfied when training completes in the experiments.

The forward pass of the binary search layer is conducted using Algorithm 2. Its backward pass computes the gradient of the contingency dispatches, which are expressed in terms of the base case

Test Case	$ \mathcal{N} $	$ \mathcal{G} $	$ \mathcal{L} $	$ \mathcal{E} $	$ \mathcal{K}_g $	$ \mathcal{K}_e $	$\dim(\mathbf{x})$
1354_peg	1354	260	673	1991	193	1430	1193
1888_rte	1888	290	1000	2531	290	1567	1580
3022_goc	3022	327	1574	4135	327	3180	2228
4917_goc	4917	567	2619	6726	567	5066	3753
6515_rte	6515	684	3673	9037	657	6474	5041

Table 8: Specifications of the SCOPF Test Cases.

dispatches and the global signal n_k computed by the binary search in the forward pass, i.e.,

$$\tilde{p}_{k,i} = \min\{\check{p}_i + n_k \gamma_i \ddot{p}_i, \bar{p}_i\}, \quad \forall i \in \mathcal{G}, \forall k \in \mathcal{K}_g, i \neq k.$$

This expression obviously has subgradients everywhere.

Once the generation dispatches for the base case and generator contingencies are estimated, it is possible to calculate the slack variables for the base case, the generator contingencies (21), and the line contingencies (27) easily.

The PDL proxy has the nice property that almost all constraints are implicitly satisfied. Only the power balance constraints in the generator contingencies may be violated. It is precisely those constraints that are captured in the primal loss function of PDL-SCOPF, i.e.,

$$\mathbf{h}_x(\mathbf{y})_k = \mathbf{1}^\top \tilde{\mathbf{p}}_k - \mathbf{1}^\top \mathbf{d} \quad (k \in \mathcal{K}_g).$$

As a result, since all the other constraints are satisfied, the dual network produces optimal dual estimates $\boldsymbol{\lambda} = D_\phi(\mathbf{x})$ for the generator contingency power balance constraints.

7.4 Numerical Experiments

The full details of the experiments are given in [37]. This section summarizes some critical points. The effectiveness of PDL-SCOPF is assessed on five specific cases from the Power Grid Library (PGLIB) [5] given in Table 8.

Instance Generation The instance generation perturbs the load demands, the cost coefficients, and the upper bounds of the generation dispatch, i.e., $x := \{\mathbf{d}, \mathbf{c}, \bar{\mathbf{p}}\}$, in the PGLIB configuration. This generalizes earlier settings [12, 14, 36], where the only load demand \mathbf{d} is perturbed. The load demands were sampled from the input distribution \mathcal{I} , which is defined as the truncated multivariate Gaussian distribution. To perturb the cost coefficients and the dispatch upper bounds, the experiments use the base values already provided in PGLIB and multiply them by factors specific to each instance. Table 9 highlights the size of the extensive SCOPF formulation (3): it gives the numbers of variables and constraints before and after applying the Presolve (with the default setting) in Gurobi. Presolve cannot even complete due to out of memory for the larger test cases. For the 3022_goc, there are about 14.6 million continuous variables and 106,600 binary variables. In the largest case 6515_rte, there are about 64.9 million continuous variables, 449,400 binary variables, and 130.7 million constraints.

The evaluation uses 1,000 instances sampled from the predefined distribution. Optimal solutions were obtained using CCGA [45]. Table 10 presents the computing times and the number of iterations of the CCGA. Observe the challenging 6515_rte case, which requires at least 4648 seconds to solve.

PDL-SCOPF has an interesting feature: the instances used in training are generated on the fly from the distribution \mathcal{P} at each iteration of Algorithm 1. There is no need to generate a set of instances before training.

Baselines PDL-SCOPF is evaluated against three machine learning baselines to assess its performance. *It is important to stress that all baselines use the same primal neural architecture, including the repair layer for the power balance in the base case.* Only PDL-SCOPF also uses a dual network. The first baseline, denoted as *Penalty*, is a self-supervised framework that uses the function

Test Case	Before			After		
	#CV	#BV	#Cnst	#CV	#BV	#Cnst
1354_peg	3.3m	50.0k	6.7m	642.3k	50.0k	642.3k
1888_rte	4.8m	83.8k	9.7m	503.4k	83.8k	649.7k
3022_goc	14.6m	106.6k	29.4m	-	-	-
4917_goc	38.2m	321.5k	77.1m	-	-	-
6515_rte	64.9m	449.4k	130.7m	-	-	-

Table 9: The Number of Binary and Continuous Variables Denoted as #BV and #CV and Constraints (#Cnst) in Extensive SCOPF Problem Before and After Presolve. ‘–’ indicates that Presolve runs out of memory. k and m signify 10^3 and 10^6 .

Test Case	Solving Time (s)			#Iteration		
	min	mean	max	min	mean	max
1354_peg	42.66	97.10	1125.91	3	4.79	7
1888_rte	171.92	654.39	2565.6	3	4.93	6
3022_goc	628.67	6932.20	15844.52	8	10.71	17
4917_goc	3720.33	8035.86	15316.95	7	10.98	16
6515_rte	4648.31	9560.78	92430.47	4	6.65	10

Table 10: Elapsed Time and Iterations for Solving SCOPF Test Instances Using CCGA (from [37]).

$f_{\mathbf{x}}(\mathbf{y}) + \rho \mathbf{1}^\top \nu(\mathbf{h}_{\mathbf{x}}(\mathbf{y}))$ as a loss function. It is used to measure the impact of the dual network. The other two baselines are supervised learning (SL) frameworks. The first supervised learning framework, *Naïve*, uses a loss function that minimizes the distance between the base case generation dispatch estimates and the ground truth. The second supervised learning framework is the *Lagrangian Duality* (LD) framework. The supervised learning models are impractical in this context, since they require solving a very large number of instances optimally: they are used to show that self-supervised learning is indeed the technique of choice in this setting.

Architectural Details Both the primal and dual networks consist of four fully-connected layers, each followed by Rectified Linear Unit (ReLU) activations. Layer normalization [4] is applied before the fully connected layers for the primal network only. The number of hidden nodes in each fully connected layer is proportional to the dimension of the input parameter, and is set to be $1.5 \dim(\mathbf{x})$.

The training uses a mini-batch size of 8 and a maximum of 1,000 epochs for supervised baselines. PDL-SCOPF uses T=20 for the number of outer iterations. Each iteration uses 2,000 forward/backward passes with a minibatch size of 8 for both the primal and dual networks, resulting in 80,000 iterations. Note that, each iteration of PDL-SCOPF generates instances *on the fly* from the distribution \mathcal{P} for a total of $8 \times 80,000$ instances. The implementation uses PyTorch, and all models were trained on a computer with a NVIDIA Tesla V100 GPU and an Intel Xeon 2.7GHz CPU. Averaged performance results based on five independent training processes with different seeds are reported.

Numerical Results The numerical results compare PDL-SCOPF with the ground truth and the baselines by evaluating the optimality gap and constraint violations. Tables 11–12 show the violations of the power balance constraint for generator contingencies (20). PDL-SCOPF and the SSL penalty method have negligible violations of the power balance constraints on all instances (they are within a tolerance of 1e-4), thus producing a primal optimization proxy. Table 13 reports the mean optimality gap in percentage, providing a comparison between the CCGA algorithm and its learning counterparts. The results show that PDL-SCOPF is always the strongest method with optimality gaps often below 1% on these test cases. PDL-SCOPF significantly dominates the Penalty (SSL) method, showing the benefits of the dual network.

Table 14 reports the elapsed GPU times for the PDL-SCOPF training and the accumulated CPU

Test Case	Naïve (SL)	LD (SL)	Penalty (SSL)	PDL-SCOPF (SSL)
1354_peg	0.022	0.003	0.000	0.000
1888_rte	0.044	0.003	0.000	0.000
3022_goc	0.088	0.006	0.000	0.000
4917_goc	0.067	0.002	0.000	0.000
6515_rte	0.001	0.000	0.000	0.000

Table 11: Maximum Violations on Power Balance Constraints for Generator Contingency (in p.u.) (from [37]).

Test Case	Naïve (SL)	LD (SL)	Penalty (SSL)	PDL-SCOPF (SSL)
1354_peg	14(7.25%)	4(2.07%)	0(0.00%)	0(0.00%)
1888_rte	31(10.68%)	7(2.41%)	0(0.00%)	0(0.00%)
3022_goc	57(17.43%)	13(3.98%)	0(0.00%)	0(0.00%)
4917_goc	42(7.41%)	12(2.12%)	0(0.00%)	0(0.00%)
6515_rte	2(0.30%)	0(0.00%)	0(0.00%)	0(0.00%)

Table 12: The Number of Generator Contingencies with Violated Power Balance Constraints (Percentages in Parenthesis) (from [37]).

Test Case	Naïve (SL)	LD (SL)	Penalty (SSL)	PDL-SCOPF (SSL)
1354_peg	13.447	2.700	2.533	0.856
1888_rte	22.115	2.436	4.969	1.960
3022_goc	159.116	8.008	1.312	0.983
4917_goc	47.212	2.096	0.454	0.210
6515_rte	2.419	1.292	2.069	0.815

Table 13: Mean Optimality Gap (%) (best values in bold) (from [37]).

Test Case	Training		Sampling	
	8@GPU	1@GPU	8@CPU	1@CPU
1354_peg	36min		269hr 43min	
1888_rte	43min		1817hr 45min	
3022_goc	1hr 7min		19256hr 7min	
4917_goc	1hr 59min		22321hr 50min	
6515_rte	2hr 54min		26557hr 43min	

Table 14: Averaged Training Time (in GPU) for PDL-SCOPF and Accumulated CPU Time to Prepare the Supervised Training Dataset (from [37]).

Test Case	PDL-SCOPF Inference Time (ms)				Speedup
	8@GPU	1@GPU	8@CPU	1@CPU	
1354_peg	6.917	5.146	290.376	25.745	3771.60×
1888_rte	7.956	5.145	455.846	46.876	13960.02×
3022_goc	13.486	5.823	1484.019	183.879	37699.81×
4917_goc	31.775	8.043	3975.468	479.881	16745.53×
6515_rte	51.9370	10.576	6767.474	823.016	11616.76×

Table 15: Averaged Inference Time of PDL-SCOPF with 1 Instance or 8 Instances on CPU or GPU in Milliseconds and Averaged Speedup over Gurobi on 1 CPU (from [37]).

time for generating instances for the supervised baselines. The time to generate solutions offline

is extremely high, even with CCGA. The self-supervised learning including PDL-SCOPF does not need the ground truth, which brings substantial benefits in training time. Most interestingly, the training time of 2 hours and 54 minutes for the French transmission system (i.e., the largest test case `6515_rte`) is significantly smaller than the time required to solve the worst-case SCOPF instance. Table 15 reports the inference time in milliseconds of PDL-SCOPF on CPUs and GPUs for a single instance or a batch of 8 instances. Even for the largest test case, PDL-SCOPF provides a high-quality approximation to a single instance in about 10 milliseconds and to a batch of instances in about 50 milliseconds. These results show that PDL-SCOPF is 4 orders of magnitude faster than Gurobi on these instances.

PDL-SCOPF highlights the step change that optimization learning may bring. Primal-Dual Learning can produce near-optimal solutions to a complex power system applications in milliseconds, i.e., four orders of magnitude faster than the best commercial solver and the state of the art column and constraint generation algorithm. Optimization learning thus expands the realm of applications which can benefit from optimization technology.

8 Conclusion

This paper presented the concept of *optimization learning*, a methodology to design differentiable programs that can learn the input/output mapping of parametric optimization problems. These optimization proxies are trustworthy by design: they compute feasible solutions to the underlying optimization problems, provide quality guarantees on the returned solutions, and scale to large instances. Optimization proxies combine traditional deep learning technology with repair or completion layers to produce feasible solutions. The paper also showed that optimization proxies can be trained end-to-end in a self-supervised way.

The paper illustrated the impact of optimization learning on two applications of interest to the power industry: the real-time risk assessment of a transmission systems and the security-constrained optimal power under N-1 Generator and Line Contingencies. In each case, optimization proxies brings orders of magnitude improvements in efficiency, which makes it possible to solve the applications in real time with high accuracy, an outcome which that could not have been achieved by state-of-the-art optimization technology.

There are many open issues in optimization learning. They include (1) understanding how to derive effective repair layers for a wide range of applications; (2) studying how to apply optimization learning for combinatorial optimization problems, where the gradients are typically not meaningful, and (3) applying Primal-Dual learning to a variety of applications in nonlinear optimization.

Optimization learning is only one direction to fuse machine learning and optimization. Learning to optimize, where machine learning is used to speed up an existing algorithm is another avenue to leverage the strengths of both approaches.

Acknowledgements

This research was partly supported by NSF award 2112533 and ARPA-E PERFORM award AR0001136. Special thanks to the Optimization Proxy and the Power Systems teams at AI4OPT and, in particular, Wenbo Chen, Guancheng Qiu, Michael Klamkin, Terrence Mak, Seonho Park, and Matthieu Tanneau for their invaluable contributions to this research. Thanks to Ferdinando Fioretto and Andre Velloso for their initial collaborations that shaped this research too.

References

- [1] A. Agrawal, Amos B., S. Barratt, Boyd S., S. Diamond, and Z. Kolter. Differentiable Convex Optimization Layers. In *Proceedings of the Neural Information Processing Systems Conference (NeurIPS)*, 2019.

- [2] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, pages 136–145. JMLR. org, 2017.
- [3] Ignacio Aravena, Daniel K Molzahn, Shixuan Zhang, Cosmin G Petra, Frank E Curtis, Shenyinying Tu, Andreas Wächter, Ermin Wei, Elizabeth Wong, Amin Gholami, et al. Recent developments in security-constrained AC optimal power flow: Overview of challenge 1 in the ARPA-E Grid Optimization Competition. *arXiv preprint arXiv:2206.07843*, 2022.
- [4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [5] Sogol Babaeinejad sarookolae, Adam Birchfield, Richard D Christie, Carleton Coffrin, Christopher DeMarco, Ruisheng Diao, Michael Ferris, Stephane Fliscounakis, Scott Greene, Renke Huang, et al. The power grid library for benchmarking AC optimal power flow algorithms. *arXiv preprint arXiv:1908.02788*, 2019.
- [6] Sogol Babaeinejad sarookolae, Adam Birchfield, Richard D. Christie, Carleton Coffrin, Christopher DeMarco, Ruisheng Diao, Michael Ferris, Stephane Fliscounakis, Scott Greene, Renke Huang, Cedric Josz, Roman Korab, Bernard Lesieutre, Jean Maeght, Terrence W. K. Mak, Daniel K. Molzahn, Thomas J. Overbye, Patrick Pinciatici, Byungkwon Park, Jonathan Snodgrass, Ahmad Tbaileh, Pascal Van Hentenryck, and Ray Zimmerman. The Power Grid Library for Benchmarking AC Optimal Power Flow Algorithms, 2019.
- [7] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.
- [8] Wenbo Chen, Reem Khir, and Pascal Van Hentenryck. Two-stage learning for the flexible job shop scheduling problem. *CoRR*, abs/2301.09703, 2023.
- [9] Wenbo Chen, Mathieu Tanneau, and Pascal Van Hentenryck. End-to-end feasible optimization proxies for large-scale economic dispatch. *IEEE Transactions on Power Systems (to appear)*, 2023.
- [10] Wenbo Chen, Mathieu Tanneau, and Pascal Van Hentenryck. End-to-End Feasible Optimization Proxies for Large-Scale Economic Dispatch. *arXiv (preprint)*, 2023.
- [11] Wenbo Chen, Mathieu Tanneau, and Pascal Van Hentenryck. Real-time risk analysis with optimization proxies. In *Proceedings of the 23rd International Power Systems Computation Conference (PSCC’2024), Paris-Saclay, France*, June 2024.
- [12] 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.
- [13] Yury Dvorkin, Pierre Henneaux, Daniel S Kirschen, and Hrvoje Pandžić. Optimizing primary response in preventive security-constrained optimal power flow. *IEEE Systems Journal*, 12(1):414–423, 2016.
- [14] Fioretto Ferdinando, Terrence W. K. Mak, and Pascal Van Hentenryck. Predicting AC optimal power flows: Combining deep learning and lagrangian dual methods. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, pages 630–637. AAAI Press, February 2020.
- [15] Ferdinando Fioretto, Pascal Van Hentenryck, Terrence W. K. Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. Lagrangian duality for constrained deep learning. In Yuxiao Dong, Georgiana Ifrim, Dunja Mladenić, Craig Saunders, and Sofie Van Hoecke, editors, *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track*, pages 118–135, Cham, 2021. Springer International Publishing.

- [16] S. Gould, R. Hartley, and D. Campbell. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(08):3988–4004, aug 2022.
- [17] Grid Optimization Competition. “Grid optimization competition datasets, 2018.
- [18] Jiachun Guo, Yong Fu, Zuyi Li, and Mohammad Shahidehpour. Direct calculation of line outage distribution factors. *IEEE Transactions on Power Systems*, 24(3):1633–1634, 2009.
- [19] Jesse Holzer, Yonghong Chen, Zhongyu Wu, Feng Pan, and Arun Veeramany. Fast Simultaneous Feasibility Test for Security Constrained Unit Commitment, 2022.
- [20] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [21] Wanjun Huang and Minghua Chen. DeepOPF-NGT: Fast No Ground Truth Deep Learning-Based Approach for AC-OPF Problems. In *ICML 2021 Workshop Tackling Climate Change with Machine Learning*, 2021.
- [22] Cédric Josz, Stéphane Fliscounakis, Jean Maeght, and Patrick Panciatici. AC Power Flow Data in MATPOWER and QCQP Format: iTesla, RTE Snapshots, and PEGASE, 2016.
- [23] Michael Klamkin, Mathieu Tanneau, and Pascal Van Hentenryck. Dual interior-point optimization learning, 2024.
- [24] Andrei V Konstantinov and Lev V Utkin. A new computationally simple approach for implementing neural networks with output hard constraints. In *Doklady Mathematics*, pages 1–9. Springer, 2024.
- [25] James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. Fast approximations for job shop scheduling: A lagrangian dual deep learning method. In *In the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)*, February 2022.
- [26] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4475–4482. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track.
- [27] Steven Krantz and Harold Parks. *The Implicit Function Theorem*. Modern Birkhauser Classics. Birkhauser. ISBN 0-8176-4285-4, 2003.
- [28] Meiyi Li, Soheil Kolouri, and Javad Mohammadi. Learning to Solve Optimization Problems With Hard Linear Constraints. *IEEE Access*, 11:59995–60004, 2023.
- [29] Xingwang Ma, Haili Song, Mingguo Hong, Jie Wan, Yonghong Chen, and Eugene Zak. The security-constrained commitment and dispatch for midwest iso day-ahead co-optimized energy and ancillary service market. In *2009 IEEE Power & Energy Society General Meeting*, pages 1–8, 2009.
- [30] MISO. Schedule 28A – Demand Curves for Transmission Constraints, 2019.
- [31] MISO. Real-time energy and operating reserve market software formulations and business logic, 2022. Business Practices Manual Energy and Operating Reserve Markets Attachment D.
- [32] MISO. Schedule 28 – Demand Curves for TOOperating Reserves, 2023.
- [33] MOSEK ApS. *MOSEK Optimizer API for Julia 10.1.24*, 2022.
- [34] Ritesh Ojha, Wenbo Chen, Hanyu Zhang, Reem Khir, Alan Erera, and Pascal Van Hentenryck. Optimization-based learning for dynamic load planning in trucking service networks, 2024.

- [35] Xiang Pan, Tianyu Zhao, Minghua Chen, and Shengyu Zhang. DeepOPF: A Deep Neural Network Approach for Security-Constrained DC OPF. *IEEE Transactions on Power Systems*, 36(3):1725–1735, 2020.
- [36] Seonho Park and Pascal Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4052–4060, 2023.
- [37] Seonho Park and Pascal Van Hentenryck. Self-supervised learning for large-scale preventive security constrained dc optimal power flow. *IEEE Transactions on Power Systems*, pages 1–11, 2024.
- [38] Guancheng Qiu, Mathieu Tanneau, and Pascal Van Hentenryck. Dual Conic Proxies for AC Optimal Power Flow. In *Proceedings of the 23rd International Power Systems Computation Conference (PSCC'2024), Paris-Saclay, France*, June 2024.
- [39] Mathieu Tanneau and Pascal Van Hentenryck. Dual lagrangian learning for conic optimization, 2024.
- [40] Diego A Tejada-Arango, Pedro Sánchez-Martin, and Andres Ramos. Security constrained unit commitment using line outage distribution factors. *IEEE Transactions on power systems*, 33(1):329–337, 2017.
- [41] Jesus Tordesillas, Jonathan P How, and Marco Hutter. Rayen: Imposition of hard convex constraints on neural networks. 2023.
- [42] University of Washington, Dept. of Electrical Engineering. Power systems test case archive, 1999.
- [43] Pascal Van Hentenryck. Machine Learning for Optimal Power Flows. *Tutorials in Operations Research*, pages 62–82, December 2021.
- [44] Alexandre Velloso and Pascal Van Hentenryck. Combining deep learning and optimization for preventive security-constrained DC optimal power flow. *IEEE Transactions on Power Systems*, 36(4):3618–3628, 2021.
- [45] Alexandre Velloso, Pascal Van Hentenryck, and Emma S Johnson. An exact and scalable problem decomposition for security-constrained optimal power flow. *Electric Power Systems Research*, 195:106677, 2021.
- [46] Junfei Wang and Pirathayini Srikantha. Fast optimal power flow with guarantees via an unsupervised generative model. *IEEE Transactions on Power Systems*, 2022.
- [47] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *AAAI*, volume 33, pages 1658–1665, 2019.
- [48] Enpeng Yuan, Wenbo Chen, and Pascal Van Hentenryck. Reinforcement learning from optimization proxy for ride-hailing vehicle relocation. *J. Artif. Intell. Res.*, 75:985–1002, 2022.