Self-Supervised Primal-Dual Learning for Constrained Optimization

Seonho Park, Pascal Van Hentenryck

H. Milton Stewart School of Industrial & Systems Engineering, Georgia Institute of Technology
seonho.park@gatech.edu, pascal.vanhentenryck@isye.gatech.edu

Abstract

This paper studies how to train machine-learning models that directly approximate the optimal solutions of constrained optimization problems. This is an empirical risk minimization under constraints, which is challenging as training must balance optimality and feasibility conditions. Supervised learning methods often approach this challenge by training the model on a large collection of pre-solved instances. This paper takes a different route and proposes the idea of Primal-Dual Learning (PDL), a self-supervised training method that does not require a set of pre-solved instances or an optimization solver for training and inference. Instead, PDL mimics the trajectory of an Augmented Lagrangian Method (ALM) and jointly trains primal and dual neural networks. Being a primal-dual method, PDL uses instance-speci c penalties of the constraint terms in the loss function used to train the primal network. Experiments show that, on a set of nonlinear optimization benchmarks, PDL typically exhibits negligible constraint violations and minor optimality gaps, and is remarkably close to the ALM optimization. PDL also demonstrated improved or similar performance in terms of the optimality gaps, constraint violations, and training times compared to existing approaches. [1]

1 Introduction

This paper considers constrained optimization problems which are ubiquitous in many disciplines including power systems, supply chains, transportation and logistics, manufacturing, and design. Some of these problems need to be solved within time limits to meet business constraints. This is the case for example of market-clearing optimizations in power systems, sensitivity analyses in manufacturing, and real-time transportation systems to name only a few. In many cases, these optimization problems must be solved repeatedly for problem instances that are closely related to each other (the so-called structure hypothesis (Bengio, Lodi, and Prouvost 2021)). These observations have stimulated significant interest in recent years at the intersection of machine learning and optimization.

This paper is concerned with one speci c approach in this rich landscape: the idea of learning the input/output mapping of an optimization problem. This approach has raised significant interest in recent years, especially in power systems. Supervised machine learning has been the most prominent approach to tackling these problems (e.g., (Zamzam and Baker 2020; Fioretto, Mak, and Van Hentenryck 2020; Chatzos, Mak, and Van Hentenryck 2021; Kotary, Fioretto, and Van Hentenryck 2021; Chatzos et al. 2020; Kotary, Fioretto, and Van Hentenryck 2022; Chen et al. 2022)). Moreover, to balance feasibility and optimality, existing methods may perform a Lagrangian relaxation of the constraints. Supervised learning relies on a collection of pre-solved instances usually obtained through historical data and data generation.

These supervised methods have achieved promising results on a variety of problems. They also have some limitations including the need to generate the instance data which is often a time-consuming and resource-intensive process. Moreover, the data generation is often subtle as optimization problems may admit multiple solutions and exhibit many symmetries. As a result, the data generation must be orchestrated carefully to simplify the learning process (Kotary, Fioretto, and Van Hentenryck 2021). Recently, Donti, Rolnick, and Kolter (2021) have proposed a self-supervised learning method, called DC3, that addresses these limitations: it bypasses the data generation process by training the machine-learning model with a loss function that captures the objective function and constraints simultaneously.

Existing approaches are also primal methods: they learn a network to predict the values of the decision variables. As a result, the constraint multipliers they use to balance feasibility and optimality are not instance-speci c: they are aggregated for each constraint over all the instances. This aggregation limits the capability of the learning method to ensure feasibility. For these reasons, dedicated procedures have been proposed to restore feasibility in a post-processing step (e.g., (Zamzam and Baker 2020; Velloso and Van Hentenryck 2021; Chen et al. 2021a; Fioretto, Mak, and Van Hentenryck 2020)). This repair process however may lead to sub-optimal solutions.

This paper is an attempt to address these limitations for some classes of applications. It proposes Primal-Dual Learning (PDL), a self-supervised primal-dual learning method to approximate the input/output mapping of a constrained optimization problem. Being a self-supervised method, PDL does not rely on a set of pre-solved instances. Moreover, be-

[1] The extended version of this paper is available at https://arxiv.org/pdf/2208.09046

ing a primal-dual method, PDL is capable of using instance-speci c multipliers for penalizing constraints. The key idea underlying PDL is to mimic the trajectories of an Augmented Lagrangian Method (ALM) and to jointly train primal and dual neural networks. As 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.

The effectiveness of PDL is demonstrated on a number of benchmarks including convex quadratic programming (QP) problems and their non-convex QP variants, quadratic constrained quadratic programming (QCQP), and optimal power flows (OPF) for energy systems. The results highlight that PDL nds solutions with small optimality gaps and negligible constraint violations. Its approximations are remarkably close to those of optimization with ALM and, on the considered test cases, PDL exhibits improved accuracy compared to supervised methods and primal self-supervised approaches.

In summary, the key contributions of PDL are summarized as follows:

- PDL proposes a self-supervised learning method for learning the input/output mapping of constrained optimization problems. It does so by mimicking the trajectories of the ALM without requiring pre-solved instances or the use of an optimization solver.
- PDL is a primal-dual method: it jointly trains two networks to approximate the primal and dual solutions of the underlying optimization. PDL can leverage the dual network to impose instance-speci c multipliers/penalties for the constraints in the loss function.
- Experimental results show that, at inference time, PDL obtains solutions with negligible constraint violations and minor optimality gaps on a collection of benchmarks that include QP, QCQP, and OPF test cases.

The rest of this paper is organized as follows. Section 2 presents the related work at a high level. Section 3 presents the paper notation and learning goal, and it reviews existing approaches in more detail. Section 4 presents PDL and Section 5 reports the experimental results. Section 6 concludes the paper.

2 Related Work

This section gives a broad overview of related work. Detailed presentations of the key related work are introduced in Section 3. The use of machine learning for optimization problems can be categorized into two main threads (See the survey by Bengio, Lodi, and Prouvost (2021) for a broad overview of this area): i) *Learning to optimize* where machine learning is used to improve an optimization solver by providing better heuristics or branching rules for instance (Chen et al. 2021b; Liu, Fischetti, and Lodi 2022); ii) *optimization proxies/surrogates* where the machine-learning model directly approximates the input/output mapping of the optimization problem (Vesselinova et al. 2020; Kotary et al. 2021).

Reinforcement learning has been mostly used for the rst category including learning how to branch and how to apply cutting planes in mixed integer programming (e.g., (Khalil et al. 2016; Tang, Agrawal, and Faenza 2020; Liu, Fischetti, and Lodi 2022), and how to derive heuristics for combinatorial optimization problems on graphs (e.g., (Khalil et al. 2017)).

Supervised learning has been mostly used for the second category. For example, Fioretto, Mak, and Van Hentenryck (2020), Chatzos et al. (2020), and Zamzam and Baker (2020) propose deep-learning models for optimal power flow that approximate generator setpoints. These supervised approaches require a set of pre-solved instances obtained from historical data and complemented by a data augmentation process that uses an optimization solver. This data augmentation process can be very time-consuming and resource-intensive, and may not be practical for certain type of applications (e.g., (Chen et al. 2022)).

Donti, Rolnick, and Kolter (2021) propose a self-supervised learning approach that avoids the need for pre-solved instances, using the objective function and constraint violations to train the neural network directly.

3 Preliminaries

3.1 Notations

Bold lowercase notations are used for vectors or sets of functions, and bold uppercase notations are used for matrices. A capital calligraphic notation such as $\mathcal{G}$ represents a set, a distribution, or a loss function. $\|\cdot\|$ represents an arbitrary norm.

3.2 Assumptions and Goal

This paper considers a constrained optimization problem $P_{\mathbf{x}}\ \mathbf{y})$ of the form:

$$
\begin{aligned}
P_{\mathbf{x}}\ \mathbf{y}): \quad & \underset{\mathbf{y}\in}{\textbf{minimize}} && f_{\mathbf{x}}\ \mathbf{y}), \\
& \textbf{subject to} && g_{\mathbf{x},j}\ \mathbf{y}) \le 0, \ \ \forall j \in \mathcal{G}, \\
& && h_{\mathbf{x},j}\ \mathbf{y}) = 0, \ \ \forall j \in \mathcal{H}.
\end{aligned} \tag{1}
$$

$\mathbf{x} \in \ ^{p}$ represents instance parameters that determine the functions $f_{\mathbf{x}}$, $\mathbf{g}_{\mathbf{x}}$, and $\mathbf{h}_{\mathbf{x}}$. Each $\mathbf{x}$ thus de nes an instance of the optimization problem (1). $\mathcal{G}$ and $\mathcal{H}$ are the indices sets of the inequality and equality functions, respectively. $\mathbf{Y}$ is the variable domain, i.e., a closed subset of ^{n} de ned by some bound constraints. The functions $f_{\mathbf{x}}$, $\mathbf{g}_{\mathbf{x}}$, and $\mathbf{h}_{\mathbf{x}}$ are smooth over $\mathbf{Y}$, and possibly nonlinear and non-convex.

The goal is to approximate the mapping from $\mathbf{x}$ to an optimal solution $\mathbf{y}^*$ of the problem $P_{\mathbf{x}}\ \mathbf{y})$ (Eq. (1)). This work restricts the potential approximate mappings to a family of neural nets N parameterized by trainable parameters θ. The model N is trained offline and, at inference time, it is expected to yield, in real-time, a high-quality approximation to an optimal solution. This paper measures the quality of an approximation using optimality gaps and maximum constraint violations over unseen test instances.

3.3 Supervised Learning

This subsection revisits some supervised-learning schemes to meet the above requirements. These methods need the

ground truth, i.e., an actual optimal solution for each instance parameters $\mathbf{x}$ that can be obtained by solving problem $P_{\mathbf{x}}\ \mathbf{y})$ with an optimization solver. Given the dataset consisting of pairs $\{\mathbf{x}, \mathbf{y}^*\}$, the neural net N strives to yield an output $\mathbf{y}$ approximating the ground truth $\mathbf{y}^*$ as accurately as possible.[2]

The Naïve Approach The naïve supervised-learning approach (e.g., (Zamzam and Baker 2020)) minimizes a simple and intuitive loss function such as

$$\mathcal{L}_{\text{naïve}} = \|\mathbf{y} \quad \mathbf{y}^*\| . \tag{2}$$

MAE and MSE have been widely used as loss functions. The main limitation of the naïve approach is that it may provide approximations that violate constraints signi cantly.

The Supervised Penalty Approach The supervised penalty approach uses a loss function that includes terms penalizing constraint violations (e.g., (Nellikkath and Chatzivasileiadis 2022))

$$\mathcal{L}_{\text{naïve}} + \sum_{j\in\mathcal{G}} \rho_{g,j}\nu\ \ g_{\mathbf{x},j}\ \ \mathbf{y})) + \sum_{j\in\mathcal{H}} \rho_{h,j}\nu\ \ h_{\mathbf{x},j}\ \ \mathbf{y})) \tag{3}$$

where $g_{\mathbf{x},j}$ and $h_{\mathbf{x},j}$ are the j^{th} inequality and equality constraints, respectively, and $\nu\ \ \cdot)$ is any violation penalty function. Each constraint also has a penalty coef cient $\rho > 0$.

The Lagrangian Duality Method It is not an obvious task to choose penalty coef cients (or Lagrangian multipliers) to minimize constraint violations and balance them with the objective function. Fioretto, Mak, and Van Hentenryck (2020) propose a Lagrangian Duality (LD) approach to determine suitable Lagrangian multipliers. They employ a subgradient method that updates the multipliers every few epochs using the rules:

$$\rho_{g,j} \leftarrow \rho_{g,j} + \gamma\nu\ \ g_{\mathbf{x},j}\ \ \mathbf{y})) , \ \forall j \in \mathcal{G}, \text{ and}$$
$$\rho_{h,j} \leftarrow \rho_{h,j} + \gamma\nu\ \ h_{\mathbf{x},j}\ \ \mathbf{y})) , \ \forall j \in \mathcal{H},$$

where the hyperparameter $\gamma > 0$ represents the step size for updating ρ.

3.4 Self-Supervised Learning

Obtaining $\mathbf{y}^*$ for a large collection of instances is a time-consuming and resource-intensive task. In addition, the data augmentation process should proceed carefully since modern solvers are often randomized and may yield rather different optimal solutions to similar optimization problems (Kotary, Fioretto, and Van Hentenryck 2021). Self-supervised learning approaches remedy this limitation by learning the neural network directly without using any pre-solved instances. Instead, they use a loss function that includes the objective function of the optimization problem.

The Self-Supervised Penalty Approach The self-supervised penalty approach uses the loss function

$$f_{\mathbf{x}}\ \mathbf{y}) + \sum_{j\in\mathcal{G}} \rho_{g,j}\nu\ \ g_{\mathbf{x},j}\ \ \mathbf{y})) + \sum_{j\in\mathcal{H}} \rho_{h,j}\nu\ \ h_{\mathbf{x},j}\ \ \mathbf{y})). \tag{4}$$

[2]The loss functions in what follows generalize naturally to mini-batches of instances.

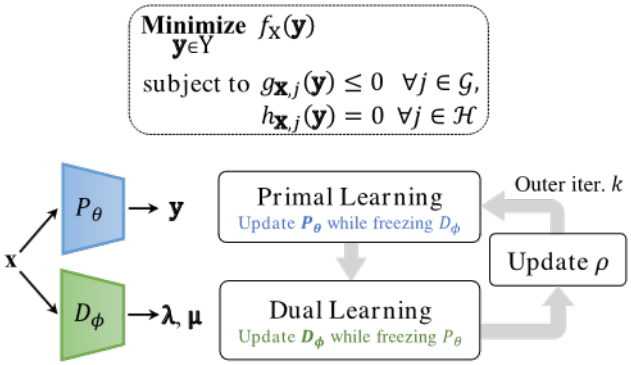

Figure 1: Schematic overview of PDL for training both primal and dual neural networks that output the optimal solutions of the constrained optimization problems.

This loss function resembles the supervised penalty approach, except that the objective function term replaces the norm term that measures the proximity to the ground truth.

DC3 DC3 (Deep Constraint Completion and Correction) is a self-supervised learning method proposed by Donti, Rolnick, and Kolter (2021). It uses the same loss as the self-supervised penalty approach (Eq. (4)), but also employs completion and correction steps to ensure feasibility. In particular, the core neural network only approximates a subset of the variables. It is then followed by a completion step that uses implicit differentiation to compute a variable assignment that satis es the equality constraints. The correction step then uses gradient descent to correct the output and satisfy the inequality constraints. Those steps ensure that a feasible solution is found. However, the obtained solution may be sub-optimal, as shown in Section 5.

4 Self-Supervised Primal-Dual Learning

This section is the core of the paper. It presents a self-supervised Primal-Dual Learning (PDL) that aims at combining the bene ts of self-supervised and Lagrangian dual methods. The key characteristics of PDL can be summarized as follows.

- **Self-Supervised Training Process Inspired by ALM** The training process of PDL mimics an Augmented Lagrangian Method (ALM) (Hestenes 1969; Powell 1969; Rockafellar 1974; Bertsekas 2014).
- **Primal and Dual Approximations** PDL jointly trains two independent networks: one for approximating primal variables and another for approximating dual variables that can then serve as Lagrangian multipliers in the primal training.
- **Instance-Speci c Lagrangian Multipliers** Contrary to the supervised methods, PDL uses instance-speci c Lagrangian multipliers, yielding a ne-grained balance between constraint satisfaction and optimality conditions.

A high-level overview of PDL is presented in Figure 1. It highlights that PDL alternates between primal learning, dual

learning, and updating a penalty coef cient at each outer iteration. Given instances $\mathbf{x}$, the *primal network* P outputs the primal solution estimates $\mathbf{y}$, i.e., $P\ \mathbf{x}) = \mathbf{y}$, whereas the *dual network* D_ϕ provides the dual solution estimates $\boldsymbol{\mu}$, for equalities and inequalities, i.e., $D_\phi\ \mathbf{x}) = \{\boldsymbol{\mu},\ \}$. The trainable parameters θ and ϕ, associated with the primal and dual networks, respectively, are learned during training. Speci cally, the *primal learning* updates P while D_ϕ is frozen. Conversely, the *dual learning* updates D_ϕ while P is frozen. Observe that PDL obtains both primal and dual estimates at inference time and is able to leverage any neural network architecture. The rest of this section provides all the details of PDL.

4.1 Primal Learning

The loss function for training the primal network is the objective function of the primal subproblem of the ALM: it combines a Lagrangian relaxation with a penalty on constraint violations. For each outer iteration k, the parameters θ of the primal network P are trained while freezing D_ϕ. The loss function of the primal training is given by

$$\mathcal{L}_{\text{p}}\ \mathbf{y}|\boldsymbol{\mu},\ ) = f_{\mathbf{x}}\ \mathbf{y}) + \boldsymbol{\mu}^T \mathbf{g}_{\mathbf{x}}\ \mathbf{y}) + \ ^T \mathbf{h}_{\mathbf{x}}\ \mathbf{y}) + \frac{\rho}{2}\left(\sum_{j\in\mathcal{G}} \nu\ g_{\mathbf{x},j}\ \mathbf{y})) + \sum_{j\in\mathcal{H}} \nu\ h_{\mathbf{x},j}\ \mathbf{y}))\right), \tag{5}$$

where $\boldsymbol{\mu}$ and are the output of the frozen dual network D_ϕ, i.e., the dual solution estimates for the inequality and equality constraints, respectively. *Note that these dual estimates are instance-speci c: for instance* $\mathbf{x}$*, the dual estimates are given by* $D_\phi\ \mathbf{x})$. The dual estimates are initialized to zero at the rst outer iteration (which is readily achievable by imposing the initial weights and bias parameters of the nal layer of D_ϕ to be zero). In Eq. (5), following the ALM, the violation functions for the inequality and equality constraints are de ned as $\nu\ g\ \mathbf{y})) = \max\{g\ \mathbf{y}), 0\}^2$ and $\nu\ h\ \mathbf{y})) = h\ \mathbf{y})^2$.

4.2 Dual Learning

The dual learning is particularly interesting. PDL does not store the dual estimates for all instances for ensuring ef- ciency and scalability. Instead, it trains the dual network D_ϕ to estimate them on a need basis. This training is again inspired by the ALM algorithm. At outer iteration k, the ALM updates the dual estimates using a proximal subgradient step:

$$\begin{aligned} \boldsymbol{\mu}_{k+1} &\leftarrow \max\{\boldsymbol{\mu}_k + \rho \mathbf{g}\ \mathbf{y}), 0\}, \\ \ _{k+1} &\leftarrow \ _k + \rho \mathbf{h}\ \mathbf{y}). \end{aligned} \tag{6}$$

This update step suggests the targets for updating D_ϕ. However, the dual training requires some care since the update rules refer both to the current estimates of the dual values, and to their new estimates. For this reason, at outer iteration k, the dual learning rst copies the dual network into a "frozen" network D_ϕ that will provide the current estimates $\boldsymbol{\mu}_k$ and $_k$. The dual learning can then update the dual network to yield a new estimate that is close to the RHS of the ALM dual update rule (Eq. (6)). Speci cally, the dual network D_ϕ is trained by minimizing the following dual loss function:

$$\mathcal{L}_{\text{d}}\ \boldsymbol{\mu},\ |\mathbf{y}, \boldsymbol{\mu}_k,\ _k) = \|\boldsymbol{\mu}\ \max\{\boldsymbol{\mu}_k + \rho \mathbf{g}_{\mathbf{x}}\ \mathbf{y}), 0\})\| + \|\ _k + \rho \mathbf{h}_{\mathbf{x}}\ \mathbf{y}))\|. \tag{7}$$

where $\boldsymbol{\mu}_k$ and $_k$ are the outputs of the frozen network D_ϕ and $\mathbf{y}$ is the output of the primal network P. Note that the primal network P is also frozen during the dual learning. Moreover, by minimizing the dual loss function (Eq. (7)), it is expected that, for each instance, the dual network yields the dual solution estimates that are close to those obtained in the ALM algorithm after applying the rst-order proximal rule (Eq. (6)).

4.3 Updating the Penalty Coef cient ρ

When constraint violations are severe, it is desirable to increase the penalty coef cient ρ, which is also used as a step size for updating the duals, in order to penalize violations more. PDL adopts the penalty coef cient updating rule from (Andreani et al. 2008) but slightly modi es it to consider all instances in the training dataset. At each outer iteration k, the maximum violation v_k is calculated as

$$v_k = \max_{\mathbf{x}\sim\mathcal{D}} \left\{\max\{\|\mathbf{h}_{\mathbf{x}}\ \mathbf{y})\|_\infty, \|\boldsymbol{\sigma}_{\mathbf{x}}\ \mathbf{y})\|_\infty\}\right\}, \tag{8}$$

$$\text{where } \sigma_{\mathbf{x},j}\ \mathbf{y}) = \max\{g_{\mathbf{x},j}\ \mathbf{y}),\ \frac{\lambda_{k,j}}{\rho}\},\ \forall j \in \mathcal{G}$$

and $\mathcal{D} = \{\mathbf{x}^{\ i)}\}_{i=1}^N$ is the training dataset. Informally speaking, the vector $\boldsymbol{\sigma}$ represents the infeasibility and complementarity for the inequality constraints (Andreani et al. 2008). At every outer iteration $k > 1$, the penalty coef - cient is increased when the maximum violation v_k is greater than a tolerance value $\tau v_{k\ 1}$ as follows:

$$\rho \leftarrow \min\{\ \rho, \rho_{\max}\} \text{ if } v_k > \tau v_{k\ 1}, \tag{9}$$

where $\tau \in\ 0, 1)$ is a tolerance to determine the update of the penalty coef cient ρ, > 1 is an update multiplier, and $\rho_{\max}$ is the upper safeguard of ρ.

Algorithm 1 Primal-Dual Learning (PDL)

Parameter: Initial penalty coef cient ρ, Maximum outer iteration K, Maximum inner iteration L, Penalty coef cient updating multiplier , Violation tolerance τ, Upper penalty coef cient safeguard $\rho_{\max}$
Input: Training dataset $\mathcal{D}$
Output: learned primal and dual networks P , D_ϕ

for $k \in \{1, \ldots, K\}$ **do**
  for $l \in \{1, \ldots, L\}$ **do** ▷ Primal Learning
    Update P using $\nabla\ \mathcal{L}_{\text{p}}$ (See Eq. (5))
  end for
  Calculate v_k as Eq. (8)
  De ne D_ϕ by copying D_ϕ
  for $l \in \{1, \ldots, L\}$ **do** ▷ Dual Learning
    Update D_ϕ using $\nabla_\phi \mathcal{L}_{\text{d}}$ (See Eq. (7))
  end for
  Update ρ with Eq. (9)
end for
return θ and ϕ

4.4 Training

The overall training procedure of PDL is detailed in Algorithm 1. The training alternates between the updates of the primal and dual networks. Unlike conventional training processes, the PDL training procedure contains outer and inner iterations. In an outer iteration, the two inner loops tackle the primal and dual learning subproblems. Each inner iteration samples a mini-batch and updates the trainable parameters (θ or ϕ). *Note that each outer iteration aims at estimating the primal and dual iteration points of the ALM algorithm. Eventually, these iteration points are expected to converge to the primal and dual solutions on the training instances and also provide the optimal solution estimates on unseen testing instances.*

5 Experiments

This section demonstrates the effectiveness of PDL on a number of constrained optimization problems. The performance of PDL is measured by its optimality gaps and maximum constraint violations, which quantify both optimality and feasibility, respectively. PDL is compared to the supervised (SL) and self-supervised (SSL) baselines introduced in Section 3, i.e.,

- **(Supervised) naïve MAE, naïve MSE:** they respectively use the l_1-norm and the l_2-norm between the optimal solution estimates and the actual optimal solutions as loss functions (See Eq. (2)).
- **(Supervised) MAE+Penalty, MSE+Penalty, LD:** on top of naïve-MAE (or naïve-MSE), these methods add terms to penalize constraint violations (See Eq. (3)). For LD, l_1-norm is associated as used in (Chatzos et al. 2020).
- **(Self-Supervised) Penalty** this method uses a loss function that represents the average of the penalty functions given in Eq. (4) over the training instances.
- **(Self-Supervised) DC3** The self-supervised method proposed by Donti, Rolnick, and Kolter (2021).

Note that DC3 is only tested on QP problems because the publicly available code[3] is only applicable to those problems. Also, the penalty function in MAE(MSE)+Penalty(SL) and Penalty(SSL) uses absolute violations i.e., $\nu\ a) = |a|$ for equalities and $\nu\ a) = \max\{a, 0\}$ for inequalities.

Note that PDL is generic and agnostic about the underlying neural network architectures. For comparison purposes with respect to the training methodology, the experimental evaluations are based on the simple fully-connected neural networks followed by ReLU activations. The implementation is based on PyTorch and the training was conducted using a Tesla RTX6000 GPU on a machine with Intel Xeon 2.7GHz. For training the models, the Adam optimizer (Kingma and Ba 2014) with the learning rate of 1e-4 was used. Other hyperparameters of PDL and the baselines were tuned using a grid search.

[3]https://github.com/locuslab/DC3

5.1 Performance Results

Convex QP & Non-convex QP Variant This benchmark uses convex QP problems and their non-convex variants proposed for evaluating DC3 (Donti, Rolnick, and Kolter 2021). The experiments followed the same experimental protocol as in that paper. Given various $\mathbf{x} \in \ ^{n_{\text{eq}}}$, the mathematical formulation for both cases reads as

$$\min_{\mathbf{y}\in\ ^{n}} \frac{1}{2}\mathbf{y}^T\mathbf{Q}\mathbf{y}+\mathbf{r}^T\mathbf{y},\ \text{s.t. } \mathbf{A}\mathbf{y}=\mathbf{x},\ \mathbf{G}\mathbf{y}\le\mathbf{h}, \quad (10)$$

$$\min_{\mathbf{y}\in\ ^{n}} \frac{1}{2}\mathbf{y}^T\mathbf{Q}\mathbf{y}+\mathbf{r}^T\sin\ \mathbf{y}),\ \text{s.t. } \mathbf{A}\mathbf{y}=\mathbf{x},\ \mathbf{G}\mathbf{y}\le\mathbf{h}, \quad (11)$$

where $\mathbf{Q} \in \ ^{n\times n} \succeq 0$, $\mathbf{r} \in \ ^{n}$, $\mathbf{A} \in \ ^{n_{\text{eq}}\times n}$, $\mathbf{G} \in \ ^{n_{\text{ineq}}\times n}$, and $\mathbf{h} \in \ ^{n_{\text{ineq}}}$. n, n_{eq}, and n_{ineq} denote the number of variables, equations, and inequality constraints, respectively. Eq. (11) is a non-convex variant of the convex QP (Eq. (10)) obtained by adding the element-wise sine operation. Overall, $10,000$ instances were generated and split into training/testing/validation datasets with the ratio of 10:1:1.

Table 1 reports the performance results of the convex QP problem and its non-convex variant on the test instances. The rst row for both tables reports the results of the ground truth, which is obtained by the optimization systems OSQP (Stellato et al. 2020) or IPOPT (Wächter and Biegler 2006). In addition, the results of the ALM are also reported: they show that the ALM converges well for these cases. The optimality gap in percentage is the average value of the optimality gaps over the test instances, i.e., $\frac{|f\ \mathbf{y}\ )\ \ f\ \mathbf{y})|}{|f\ \mathbf{y}\ )|}$. In the tables, columns 'Max eq.', 'Max ineq.', 'Mean eq.', 'Mean ineq.' represent the average violation of the maximum or average values for the equality or inequality constraints across the test instances. The performance of PDL, which is only slightly worse than ALM, is compelling when considering both optimality and feasibility. The baselines each present their own weakness. For example, supervised approaches tend to violate some constraints substantially, which can be marginally mitigated by adding penalty terms or employing the LD method. DC3, as it exploits the completion and correction steps, satis es the constraints but presents signi cant optimality gaps. The superiority of PDL for these cases is even more compelling when testing with various con gurations. Fig. 2 shows the optimality gaps and the maximum violations in various settings. Note that the PDL results are located close to the origin, meaning that it has negligible optimality gaps and constraint violations.

Quadratically Constrained Quadratic Program (QCQP) QCQP is basically a non-convex problem in which both objective function and constraints are quadratic functions. This experiment considers a binary least-square minimization problem, which can be de ned as

$$\min_{\mathbf{y}\in\ ^{n}} \ \mathbf{A}\mathbf{y}\ \ \mathbf{x})^T\ \mathbf{A}\mathbf{y}\ \ \mathbf{x}),\ \text{s.t. } y_i^2 = 1,\ \forall i\in\{1,\dots,n\}, \quad (12)$$

where $\mathbf{A} \in \ ^{n_{\text{aff}}\times n}$ and $\mathbf{x} \in \ ^{n_{\text{aff}}}$. Here, n_{aff} is the dimension of the af ne space. This problem can be formulated as a 0-1 integer programming problem as

$$\min_{\mathbf{y}\in\{\ \ 1,1\}^n} \ \mathbf{A}\mathbf{y}\ \ \mathbf{x})^T\ \mathbf{A}\mathbf{y}\ \ \mathbf{x}).$$

Method	Type	Obj.	Opt. Gap(%)	Max eq.	Max ineq.	Mean eq.	Mean ineq.
		convex QP (Eq. (10))					
Optimizer(OSQP)		-15.047	-	0.000	0.001	0.000	0.000
Optimizer(ALM)		-15.046	0.003	0.000	0.000	0.000	0.000
PDL(ours)		-15.017(0.009)	0.176(0.054)	0.005(0.001)	0.001(0.000)	0.002(0.000)	0.000(0.000)
Penalty	SSL	-15.149(0.003)	0.680(0.017)	0.048(0.003)	0.030(0.003)	0.013(0.000)	0.004(0.001)
DC3		-14.112(0.015)	6.219(0.098)	0.000(0.000)	0.000(0.000)	0.000(0.000)	0.000(0.000)
Naïve MAE		-15.051(0.003)	0.046(0.012)	0.025(0.002)	0.018(0.003)	0.008(0.001)	0.002(0.000)
Naïve MSE		-15.047(0.001)	0.122(0.025)	0.018(0.001)	0.023(0.004)	0.006(0.000)	0.003(0.001)
MAE+Penalty	SL	-15.043(0.004)	0.093(0.013)	0.031(0.001)	0.016(0.001)	0.010(0.000)	0.002(0.000)
MSE+Penalty		-15.047(0.001)	0.059(0.004)	0.026(0.003)	0.016(0.001)	0.008(0.001)	0.002(0.000)
LD		-15.043(0.006)	0.093(0.015)	0.033(0.001)	0.016(0.001)	0.011(0.001)	0.002(0.000)
		non-convex QP variant (Eq. (11))					
Optimizer(IPOPT)		-11.592	-	0.000	0.000	0.000	0.000
Optimizer(ALM)		-11.592	0.002	0.000	0.000	0.000	0.000
PDL (ours)		-11.552(0.006)	0.324(0.051)	0.004(0.001)	0.001(0.000)	0.001(0.000)	0.000(0.000)
Penalty	SSL	-11.654(0.002)	0.532(0.015)	0.042(0.002)	0.027(0.001)	0.011(0.000)	0.003(0.000)
DC3		-11.118(0.017)	4.103(0.151)	0.000(0.000)	0.000(0.000)	0.000(0.000)	0.000(0.000)
Naïve MAE		-11.593(0.004)	0.044(0.021)	0.020(0.002)	0.022(0.006)	0.006(0.000)	0.001(0.001)
Naïve MSE		-11.593(0.002)	0.031(0.008)	0.017(0.000)	0.029(0.010)	0.005(0.000)	0.002(0.001)
MAE+Penalty	SL	-11.591(0.004)	0.073(0.016)	0.033(0.001)	0.016(0.002)	0.010(0.000)	0.001(0.000)
MSE+Penalty		-11.593(0.001)	0.050(0.002)	0.023(0.002)	0.017(0.001)	0.007(0.001)	0.001(0.000)
LD		-11.593(0.005)	0.072(0.005)	0.032(0.001)	0.017(0.002)	0.010(0.000)	0.001(0.000)

Table 1: Performance results of the self-supervised (SSL) and supervised learning (SL) schemes for the convex QP problem (Top) and non-convex QP problem (Bottom) with $n = 100$, $n_{\text{eq}} = n_{\text{ineq}} = 50$ on 833 test instances. Std. dev. in parenthesis is evaluated across 5 independent runs. Underlined numbers denote poor results (opt. gap over 1% or max violation over 0.01).

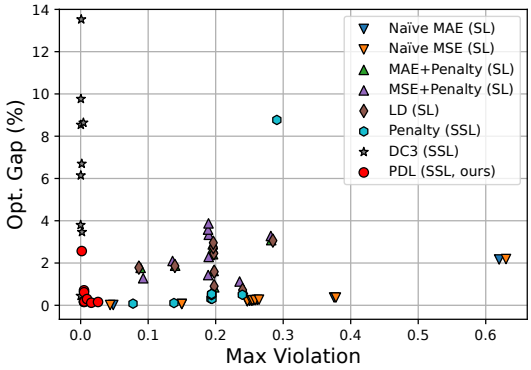

Figure 2: The optimality gaps (%) and maximum violations for various convex QP problems. Tested on 9 problem configurations using $n_{\text{eq}} = \{10, 30, 50, 70, 90\}$ and $n_{\text{ineq}} = 50$, or $n_{\text{eq}} = 50$ and $n_{\text{ineq}} = \{10, 30, 70, 90\}$.

Method	Obj.	Opt. Gap (%) mean	Opt. Gap (%) max	Time (s)
Opt.(Gurobi)	710.071	-	-	257.361
Opt.(CVXPY-QCQP)	1082.793	52.597	408.631	4.646
Opt.(ALM)	710.923	0.119	7.659	128.992
PDL(SSL, ours)	**711.064**	**0.141**	**8.307**	0.004
Penalty (SSL)	1295.388	82.555	109.406	0.004
Naïve MAE (SL)	716.132	0.834	67.051	0.004
Naïve MSE (SL)	716.695	0.911	85.073	0.004
MAE+Penalty (SL)	714.891	0.664	61.093	0.004
MSE+Penalty (SL)	717.869	1.072	80.577	0.004
LD (SL)	715.522	0.751	73.290	0.004

Table 2: Performance results for the QCQP problem (Eq. (12)) with $n = 50$ and $n_{\text{aff}} = 75$ on 1000 test instances. The best results are shown in bold.

Gurobi 9.5.2 was used as the optimization solver to produce the instance data for the supervised baselines, and also served as the reference for computing optimality gaps. In addition, CVXPY-QCQP (Park and Boyd 2017) was used as an alternative heuristic optimizer for additional comparisons. Since it is a heuristic method, CVXPY-QCQP tends to solve problems faster, but often converges to a suboptimal solution. The ALM is also included in the table for comparison. As it is a local search method and the local solution depends on the initial point, the table reports the best objective values obtained from 50 independent ALM runs with randomized initial points.

The overall results for $n = 50$ and $n_{\text{aff}} = 75$ are illustrated in Table 2. The supervised baselines are trained on 10,000 instances solved by Gurobi and the training instances for the self-supervised learning methods are generated on the fly during training. In this experiment, nding a feasible point is readily achievable as one can exploit a sigmoid

Method	case57		case118	
	Gap(%)	Max viol.	Gap(%)	Max viol.
PDL (SSL, ours)	0.21(0.02)	**0.01**(0.00)	0.73(0.12)	**0.04**(0.01)
Penalty (SSL)	9.73(0.03)	0.03(0.00)	5.51(0.11)	0.05(0.00)
Naïve MAE (SL)	0.39(0.54)	0.09(0.06)	0.73(0.25)	0.35(0.13)
Naïve MSE (SL)	0.06(0.01)	0.06(0.03)	**0.34**(0.38)	0.21(0.12)
MAE+Penalty (SL)	0.37(0.52)	0.03(0.01)	0.91(0.86)	0.08(0.02)
MSE+Penalty (SL)	**0.05**(0.01)	0.02(0.00)	0.37(0.33)	0.05(0.01)
LD (SL)	0.33(0.45)	0.02(0.00)	0.70(0.60)	0.06(0.00)

Table 3: Performance results for the case56 and case118 AC-OPF problems. Std. dev. in parenthesis is evaluated across 5 independent runs. The best values are in bold.

function in the last layer of the neural network. Hence, the constraint violation results are not reported. Similar to the ALM and also inspired by the stochastic multi-start local search (Schoen 1991), 10 independent models with different seeds were built for each training method, and the reported results in the table are based on the best values among 10 inferences. The times reported in the table denote the inference times for training the learning schemes or the solving times for the optimization runs.

The results in Table 2 show that PDL is particularly effective. PDL is almost as accurate as the ALM in terms of optimality gaps, with a mean optimality gap of 0.14% and a maximum optimality gap of only 8.31% among 1,000 test instances. Note that the ALM method has a mean optimality gap of 0.12% and a maximum optimality gap of only 7.66% when solving (not learning) the 1,000 test instances. As a result, *PDL provides about an order of magnitude improvement over the baselines.* Moreover, PDL presents a speedup of around $64,340\times$ over Gurobi at the cost of a reasonable loss in optimality.

AC-Optimal Power Flow Problems AC-optimal power flow (AC-OPF) is a fundamental building block for operating electrical power systems. AC-OPF determines the most economical setpoints for each generator, while satisfying the demand and physical/engineering constraints simultaneously. Different instances of an AC-OPF problem are given by the active and reactive power demands. The variables, which correspond to the output of the (primal) neural network, are the active and reactive power generations, the nodal voltage magnitudes, and the phase angles for all buses.

Table 3 reports the performance results of the case56 and case118 in PGLib (Babaeinejadsarookolaee et al. 2019). The results show that PDL produces competitive optimality gaps and maximum constraint violations, while supervised penalty approaches also show ne performances. However, for bulk power systems, where gathering the pre-solved instances is cumbersome, PDL may become a strong alternative to the supervised approaches for yielding real-time approximations of the solutions.

Test Case	Training Time (s)		Pre-solved Training Set Generation Time (s)
	PDL (SSL)	LD (SL)	
Convex QP	1558.5	1493.3	32.5
Nonconvex QP	1624.5	1550.4	630.1
AC-OPF(case57)	5932.5	8473.2	3690.2
AC-OPF(case118)	7605.1	9149.6	13120.3
QCQP	5553.2	3572.3	2573607.3 715 hours

Table 4: The averaged elapsed time (s) for training PDL and the representative supervised learning baseline, LD (mid-column). The CPU time for gathering the pre-solved training instances (right-column).

5.2 Training Time vs. Data Generation Time

Table 4 reports the training times for PDL, and shows that they are comparable to those of one of the supervised learning baselines, LD. Importantly, it also highlights a key bene t of PDL and self-supervised methods in general. Supervised learning schemes rely on pre-solved instances and may need to run an optimization solver on each input data. The QCQP problem required around 715 hours of CPU time to collect the training dataset. In contrast, PDL, being self-supervised, does not need this step. This potential bene t may be important for training input/output mappings for large constrained optimization problems in a timely manner.

6 Conclusion

This paper presented PDL, a self-supervised Primal-Dual Learning scheme, to learn the input/output mappings of constrained optimization problems. Contrary to supervised learning methods, PDL does not require a collection of pre-solved instances or an optimization solver. Instead, PDL tracks the iteration points of the ALM and jointly learns primal and dual networks that approximate primal and dual solution estimates. Moreover, being a primal-dual method, PDL features a loss function for primal training with instance-speci c multipliers for its constraint terms. Experimental results on a number of constrained optimization problems highlight that PDL nds solutions with small optimality gaps and negligible constraint violations. Its approximations are remarkably close to the optimal solutions from the ALM, and PDL typically exhibits improved accuracy compared to supervised methods and primal self-supervised approaches.

PDL has also a number of additional bene ts that remain to be explored. Being a primal-dual method, PDL provides dual approximations which can be useful for downstream tasks (e.g., approximating Locational Marginal Prices in power systems or prices in general). Dual solutions estimates can also be leveraged when learning is used for warm-starting an optimization or, more generally, speeding up an optimization solver by identifying the set of binding constraints. Finally, observe that PDL is generic with respect to the underlying primal and dual neural networks. Exploring more complex architectures for a variety of graph problems is also an interesting direction.

Acknowledgments

This research is partly supported by NSF under Award Number 2007095 and 2112533, and ARPA-E, U.S. Department of Energy under Award Number DE-AR0001280.

References

Andreani, R.; Birgin, E. G.; Martínez, J. M.; and Schuverdt, M. L. 2008. On augmented Lagrangian methods with general lower-level constraints. *SIAM Journal on Optimization*, 18(4): 1286–1309.

Babaeinejadsarookolaee, S.; Birch eld, A.; Christie, R. D.; Coffrin, C.; DeMarco, C.; Diao, R.; Ferris, M.; Fliscounakis, S.; Greene, S.; Huang, R.; et al. 2019. The power grid library for benchmarking ac optimal power flow algorithms. *arXiv preprint arXiv:1908.02788*.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421.

Bertsekas, D. P. 2014. *Constrained optimization and Lagrange multiplier methods*. Academic press.

Chatzos, M.; Fioretto, F.; Mak, T. W.; and Van Hentenryck, P. 2020. High- delity machine learning approximations of large-scale optimal power flow. *arXiv preprint arXiv:2006.16356*.

Chatzos, M.; Mak, T. W.; and Van Hentenryck, P. 2021. Spatial network decomposition for fast and scalable ac-opf learning. *IEEE Transactions on Power Systems*, 37(4): 2601–2612.

Chen, B.; Donti, P. L.; Baker, K.; Kolter, J. Z.; and Bergés, M. 2021a. Enforcing policy feasibility constraints through differentiable projection for energy optimization. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, 199–210.

Chen, T.; Chen, X.; Chen, W.; Heaton, H.; Liu, J.; Wang, Z.; and Yin, W. 2021b. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*.

Chen, W.; Park, S.; Tanneau, M.; and Van Hentenryck, P. 2022. Learning optimization proxies for large-scale security-constrained economic dispatch. *Electric Power Systems Research*, 213: 108566.

Donti, P. L.; Rolnick, D.; and Kolter, J. Z. 2021. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*.

Fioretto, F.; Mak, T. W.; and Van Hentenryck, P. 2020. Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 34, 630–637.

Hestenes, M. R. 1969. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5): 303–320.

Khalil, E.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30.

Khalil, E.; Le Bodic, P.; Song, L.; Nemhauser, G.; and Dilkina, B. 2016. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 30.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kotary, J.; Fioretto, F.; and Van Hentenryck, P. 2021. Learning hard optimization problems: A data generation perspective. *Advances in Neural Information Processing Systems*, 34: 24981–24992.

Kotary, J.; Fioretto, F.; and Van Hentenryck, P. 2022. Fast approximations for job shop scheduling: A lagrangian dual deep learning method. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 36, 7239–7246.

Kotary, J.; Fioretto, F.; Van Hentenryck, P.; and Wilder, B. 2021. End-to-end constrained optimization learning: A survey. *arXiv preprint arXiv:2103.16378*.

Liu, D.; Fischetti, M.; and Lodi, A. 2022. Learning to search in local branching. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 36, 3796–3803.

Nellikkath, R.; and Chatzivasileiadis, S. 2022. Physics-informed neural networks for ac optimal power flow. *Electric Power Systems Research*, 212: 108412.

Park, J.; and Boyd, S. 2017. General heuristics for nonconvex quadratically constrained quadratic programming. *arXiv preprint arXiv:1703.07870*.

Powell, M. J. 1969. A method for nonlinear constraints in minimization problems. *Optimization*, 283–298.

Rockafellar, R. T. 1974. Augmented Lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12(2): 268–285.

Schoen, F. 1991. Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, 1(3): 207–228.

Stellato, B.; Banjac, G.; Goulart, P.; Bemporad, A.; and Boyd, S. 2020. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4): 637–672.

Tang, Y.; Agrawal, S.; and Faenza, Y. 2020. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, 9367–9376. PMLR.

Velloso, A.; and Van Hentenryck, P. 2021. Combining deep learning and optimization for preventive security-constrained DC optimal power flow. *IEEE Transactions on Power Systems*, 36(4): 3618–3628.

Vesselinova, N.; Steinert, R.; Perez-Ramirez, D. F.; and Boman, M. 2020. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8: 120388–120416.

Wächter, A.; and Biegler, L. T. 2006. On the implementation of an interior-point lter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1): 25–57.

Zamzam, A. S.; and Baker, K. 2020. 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)*, 1–6. IEEE.