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-specific 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 specific 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-specific: 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-
where the machine-learning optimization proxies/surrogates in machine learning is used to improve an optimization solver by the optimization problem (Vesselinova et al. 2020; Kotary model directly approximates the input/output mapping of overview of this area): i) problems can be categorized into two main threads (See the in Section 3. The use of machine learning for optimization tailored presentations of the key related work are introduced This section gives a broad overview of related work. De-

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, Lodí, 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 first 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 Hen-
tenryck (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:

\[
P_\mathbf{x}(\mathbf{y}) : \quad \text{minimize} \quad f_\mathbf{x}(\mathbf{y}),
\]

\[
\text{subject to} \quad g_{\mathbf{x},j}(\mathbf{y}) \leq 0, \quad \forall j \in \mathcal{G}, \quad (1)
\]

\[
h_{\mathbf{x},j}(\mathbf{y}) = 0, \quad \forall j \in \mathcal{H}.
\]

\( \mathbf{x} \in \mathbb{R}^p \) represents instance parameters that determine the functions \( f_{\mathbf{x}}, g_{\mathbf{x}}, \) and \( h_{\mathbf{x}} \). Each \( \mathbf{x} \) thus defines 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 \( \mathbb{R}^n \) defined by some bound constraints. The functions \( f_{\mathbf{x}}, g_{\mathbf{x}}, \) and \( h_{\mathbf{x}} \) 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 \( \mathcal{N}_\theta \) parameterized by trainable parameters \( \theta \). The model \( \mathcal{N}_\theta \) 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
The Naïve Approach  The naïve supervised-learning approach (e.g., (Zamzam and Baker 2020)) minimizes a simple and intuitive loss function such as

\[ \mathcal{L}_{\text{naïve}} = ||y - y^*||. \]  

(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 significantly.

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

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

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

The Lagrangian Duality Method  It is not an obvious task to choose penalty coefficients (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:

\[
\rho_{g,j} \leftarrow \rho_{g,j} + \gamma \nu(g_{x,j}(y)), \quad \forall j \in G, \text{ and} \\
\rho_{h,j} \leftarrow \rho_{h,j} + \gamma \nu(h_{x,j}(y)), \quad \forall j \in H,
\]

where the hyperparameter \( \gamma > 0 \) represents the step size for updating \( \rho \).

3.4 Self-Supervised Learning

Obtaining \( y^* \) 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

\[
f_{\phi}(y) + \sum_{j \in G} \rho_{g,j} \nu(g_{x,j}(y)) + \sum_{j \in H} \rho_{h,j} \nu(h_{x,j}(y)). \tag{4}
\]

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 satisfies 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 benefits 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-Specific Lagrangian Multipliers**  Contrary to the supervised methods, PDL uses instance-specific Lagrangian multipliers, yielding a fine-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 coefficient at each outer iteration. Given instances \( \mathbf{x} \), the primal network \( P_\theta \) outputs the primal solution estimates \( \mathbf{y} \), i.e., \( P_\theta(\mathbf{x}) = \mathbf{y} \), whereas the dual network \( D_\phi \) provides the dual solution estimates \( \mu, \lambda \) for equalities and inequalities, i.e., \( D_\phi(\mathbf{x}) = \{ \mu, \lambda \} \).

The trainable parameters \( \theta \) and \( \phi \), associated with the primal and dual networks, respectively, are learned during training. Specifically, the primal learning updates \( P_\theta \) while \( D_\phi \) is frozen. Conversely, the dual learning updates \( D_\phi \) while \( P_\theta \) 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 \( \theta \) of the primal network \( P_\theta \) are trained while freezing \( D_\phi \).

The loss function of the primal training is given by

\[
L_p(\mathbf{y}| \mu, \lambda) = f_p(\mathbf{y}) + \mu^T g_p(\mathbf{y}) + \lambda^T h_p(\mathbf{y}) + \frac{\rho}{2} \left( \sum_{j \in \mathcal{G}} \nu(g_{p,j}(\mathbf{y})) + \sum_{j \in \mathcal{H}} \nu(h_{p,j}(\mathbf{y})) \right),
\]

where \( \mu \) and \( \lambda \) are the output of the frozen dual network \( D_\phi \), i.e., the dual solution estimates for the inequality and equality constraints, respectively. Note that these dual estimates are instance-specific: for instance \( \mathbf{x} \), the dual estimates are given by \( D_\phi(\mathbf{x}) \). The dual estimates are initialized to zero at the first outer iteration (which is readily achievable by imposing the initial weights and bias parameters of the final layer of \( D_\phi \) to be zero). In Eq. (5), following the ALM, the violation functions for the inequality and equality constraints are defined 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 efficiency and scalability. Instead, it trains the dual network \( D_\phi \) 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:

\[
\begin{align*}
\mu_{k+1} &\leftarrow \max\{\mu_k + \rho g(\mathbf{y}), 0\}, \\
\lambda_{k+1} &\leftarrow \lambda_k + \rho h(\mathbf{y}).
\end{align*}
\]

(6)

This update step suggests the targets for updating \( D_\phi \). 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 first copies the dual network into a “frozen” network \( D_{\phi_k} \) that will provide the current estimates \( \mu_k \) and \( \lambda_k \). 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)). Specifically, the dual network \( D_{\phi_k} \) is trained by minimizing the following dual loss function:

\[
L_\phi(\mu, \lambda, \mu_k, \lambda_k) = \|\mu - (\max\{\mu_k + \rho g_p(\mathbf{y}), 0\} \| + \|\lambda - (\lambda_k + \rho h_p(\mathbf{y}))\|.
\]

(7)

where \( \mu_k \) and \( \lambda_k \) are the outputs of the frozen network \( D_{\phi_k} \) and \( \mathbf{y} \) is the output of the primal network \( P_\theta \). Note that the primal network \( P_\theta \) 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 first-order proximal rule (Eq. (6)).

### 4.3 Updating the Penalty Coefficient \( \rho \)

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

\[
v_k = \max_{\mathbf{x} \sim \mathcal{D}} \max\{\|h_{\mathbf{x}}(\mathbf{y})\|_\infty, \|\sigma_{\mathbf{x}}(\mathbf{y})\|_\infty\},
\]

(8)

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 \( \sigma \) represents the infeasibility and complementarity for the inequality constraints (Andreani et al. 2008). At every outer iteration \( k > 1 \), the penalty coefficient is increased when the maximum violation \( v_k \) is greater than a tolerance value \( \tau v_{k-1} \) as follows:

\[
\rho \leftarrow \min\{\alpha \rho, \rho_{\max}\} \text{ if } v_k > \tau v_{k-1},
\]

(9)

where \( \tau \in (0, 1) \) is a tolerance to determine the update of the penalty coefficient \( \rho \), \( \alpha > 1 \) is an update multiplier, and \( \rho_{\max} \) is the upper safeguard of \( \rho \).
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 ($\theta$ or $\phi$). 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) naive MAE, naive 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 naive-MAE (or naive-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.

Note that DC3 is only tested on QP problems because the publicly available code\footnote{https://github.com/locuslab/DC3} 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.

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 $x \in \mathbb{R}^{n_x}$, the mathematical formulation for both cases reads as

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

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

where $Q \in \mathbb{R}^{n_y \times n_y} \geq 0$, $r \in \mathbb{R}^{n_y}$, $A \in \mathbb{R}^{n_{eq} \times n_y}$, $G \in \mathbb{R}^{n_{ineq} \times n_y}$, and $h \in \mathbb{R}^{n_{ineq}}$. $n_{eq}$ 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 first 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., $\frac{|f(y) - f(y^*)|}{|f(y^*)|}$. 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, satisfies the constraints but presents significant optimality gaps. The superiority of PDL for these cases is even more compelling when testing with various configurations. 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 defined as

$$
\min_{y \in \mathbb{R}^{n_y}} (A y - x)^T(A y - x), \quad \text{s.t.} \quad y_i^2 = 1, \quad \forall i \in \{1, \ldots, n\}, \quad (12)
$$

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

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

4056
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizer(OSQP)</td>
<td></td>
<td>-15.046</td>
<td>-15.017(0.009)</td>
<td>0.176(0.054)</td>
<td>0.005(0.001)</td>
<td>0.001(0.000)</td>
<td>0.002(0.000)</td>
</tr>
<tr>
<td>Optimizer( ALM)</td>
<td></td>
<td>-15.046</td>
<td>0.003</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Penalty</td>
<td>SSL</td>
<td>-15.046</td>
<td>0.025</td>
<td>0.001</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>DC3</td>
<td></td>
<td>-15.046</td>
<td>6.219(0.098)</td>
<td>0.015(0.003)</td>
<td>0.000(0.000)</td>
<td>0.000(0.000)</td>
<td>0.000(0.000)</td>
</tr>
<tr>
<td>Naïve MAE</td>
<td></td>
<td>-15.046</td>
<td>-15.043(0.006)</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Naïve MSE</td>
<td></td>
<td>-15.046</td>
<td>0.001</td>
<td>-15.043(0.006)</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>MAE+Penalty</td>
<td>SL</td>
<td>-15.046</td>
<td>0.000</td>
<td>0.016(0.001)</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>MSE+Penalty</td>
<td></td>
<td>-15.046</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>LD</td>
<td></td>
<td>-15.046</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizer( IPOPT)</td>
<td></td>
<td>-11.592</td>
<td>-11.552(0.006)</td>
<td>0.324(0.051)</td>
<td>0.004(0.001)</td>
<td>0.001(0.000)</td>
<td>0.000(0.000)</td>
</tr>
<tr>
<td>Optimizer( ALM)</td>
<td></td>
<td>-11.592</td>
<td>0.002</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Penalty</td>
<td>SSL</td>
<td>-11.592</td>
<td>0.042</td>
<td>0.027(0.001)</td>
<td>0.011(0.000)</td>
<td>0.003(0.000)</td>
<td>0.000(0.000)</td>
</tr>
<tr>
<td>DC3</td>
<td></td>
<td>-11.592</td>
<td>4.103(0.151)</td>
<td>0.006(0.000)</td>
<td>0.000(0.000)</td>
<td>0.000(0.000)</td>
<td>0.000(0.000)</td>
</tr>
<tr>
<td>Naïve MAE</td>
<td></td>
<td>-11.592</td>
<td>-11.593(0.004)</td>
<td>0.044(0.021)</td>
<td>0.020(0.002)</td>
<td>0.022(0.006)</td>
<td>0.006(0.000)</td>
</tr>
<tr>
<td>Naïve MSE</td>
<td></td>
<td>-11.592</td>
<td>-11.593(0.004)</td>
<td>0.031(0.008)</td>
<td>0.017(0.000)</td>
<td>0.029(0.010)</td>
<td>0.005(0.000)</td>
</tr>
<tr>
<td>MAE+Penalty</td>
<td>SL</td>
<td>-11.592</td>
<td>-11.593(0.004)</td>
<td>0.073(0.016)</td>
<td>0.033(0.001)</td>
<td>0.016(0.002)</td>
<td>0.010(0.000)</td>
</tr>
<tr>
<td>MSE+Penalty</td>
<td></td>
<td>-11.592</td>
<td>-11.593(0.004)</td>
<td>0.050(0.002)</td>
<td>0.025(0.002)</td>
<td>0.017(0.001)</td>
<td>0.007(0.001)</td>
</tr>
<tr>
<td>LD</td>
<td></td>
<td>-11.592</td>
<td>-11.593(0.004)</td>
<td>0.072(0.005)</td>
<td>0.032(0.001)</td>
<td>0.017(0.001)</td>
<td>0.010(0.000)</td>
</tr>
</tbody>
</table>

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 $n = 100$, $n_{eq} = n_{ineq} = 50$ 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 0.1).

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.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Obj.</th>
<th>Opt. Gap (%)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opt.(Gurobi)</td>
<td>710.071</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Opt.(CVXPY-QCQP)</td>
<td>1082.793</td>
<td>52.597</td>
<td>408.631</td>
</tr>
<tr>
<td>Opt.(ALM)</td>
<td>710.923</td>
<td>0.119</td>
<td>7.659</td>
</tr>
<tr>
<td>PDL(SSL, ours)</td>
<td>711.064</td>
<td>0.141</td>
<td>8.307</td>
</tr>
<tr>
<td>Penalty (SSL)</td>
<td>1295.388</td>
<td>82.555</td>
<td>109.406</td>
</tr>
<tr>
<td>Naïve MAE (SL)</td>
<td>716.132</td>
<td>0.834</td>
<td>67.051</td>
</tr>
<tr>
<td>Naïve MSE (SL)</td>
<td>716.695</td>
<td>0.911</td>
<td>85.073</td>
</tr>
<tr>
<td>MAE+Penalty (SL)</td>
<td>714.891</td>
<td>0.664</td>
<td>61.093</td>
</tr>
<tr>
<td>MSE+Penalty (SL)</td>
<td>717.869</td>
<td>1.072</td>
<td>80.577</td>
</tr>
<tr>
<td>LD (SL)</td>
<td>715.522</td>
<td>0.751</td>
<td>73.290</td>
</tr>
</tbody>
</table>

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

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 $n_{aff} = 75$ 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, finding a feasible point is readily achievable as one can exploit a sigmoid
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 an order of magnitude improvement over the baselines. Moreover, PDL presents a speedup of around 64,340× 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 AC-OPF problems. Std. dev. in parenthesis is evaluated across 5 independent runs. The best values are in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>case57</th>
<th>case118</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gap(%)</td>
<td>Max viol.</td>
</tr>
<tr>
<td>PDL (SSL, ours)</td>
<td>0.21(0.02)</td>
<td>0.01(0.00)</td>
</tr>
<tr>
<td>Penalty (SSL)</td>
<td>0.39(0.15)</td>
<td>0.02(0.06)</td>
</tr>
<tr>
<td>Naïve MSE (SL)</td>
<td>0.06(0.01)</td>
<td>0.06(0.03)</td>
</tr>
<tr>
<td>MAE+Penalty (SL)</td>
<td>0.37(0.52)</td>
<td>0.03(0.01)</td>
</tr>
<tr>
<td>MSE+Penalty (SL)</td>
<td>0.05(0.11)</td>
<td>0.02(0.00)</td>
</tr>
<tr>
<td>LD (SL)</td>
<td>0.35(0.45)</td>
<td>0.02(0.00)</td>
</tr>
</tbody>
</table>

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.

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 benefit 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 benefit 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-specific multipliers for its constraint terms. Experimental results on a number of constrained optimization problems highlight that PDL finds 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 benefits 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.
References


