Descent-Net: Learning Descent Directions for Linearly Constrained Optimization[†]

Zi-sheng Zhou[1], Deng-yu Zheng[1], Zi-rui Chen[1] and Shi-xiang Chen[1*]

[1] School of Mathematical Sciences, Key Laboratory of the Ministry of Education for Mathematical Foundations and Applications of Digital Technology, University of Science and Technology of China, Hefei, Anhui, China .

*Corresponding author(s). E-mail(s): shxchen@ustc.edu.cn;
Contributing authors: zzs935@mail.ustc.edu.cn;
zq_renius919@mail.ustc.edu.cn; zirchen@mail.ustc.edu.cn;

Abstract

Deep learning approaches, known for their ability to model complex relationships and fast execution, are increasingly being applied to solve large-scale optimization problems. However, existing methods often face challenges in simultaneously ensuring feasibility and achieving an optimal objective value. To address this issue, we propose Descent-Net, a neural network designed to learn effective descent directions from feasible solutions for linearly constrained optimization problems, improving the objective value while preserving feasibility with theoretical guarantees on convergence. Our method demonstrates strong performance on synthetic linearly constrained optimization tasks. Furthermore, it exhibits strong scalability on large-scale portfolio optimization experiments with thousands of assets, and we extend the approach to the real-world AC optimal power flow problem with nonlinear constraints, where promising empirical performance is observed despite the lack of theoretical guarantees.

Keywords: Learning to Optimize, Linearly Constrained Optimization, Feasible direction, Unrolling

MSC Classification: 90C20 , 90C25 , 68T07

[†]This research was supported by the Young Scientists Fund of the National Natural Science Foundation of China (Grant No. 12 401 417).

1 Introduction

Constrained optimization problems are ubiquitous in practical applications. While traditional optimization algorithms [29, 30] offer strong theoretical guarantees, their computational efficiency often falls short when applied to modern large-scale problems. As a result, there is increasing interest in leveraging neural network-based methods to tackle constrained optimization tasks. In recent years, many emerging works have proposed end-to-end frameworks for solving constrained optimization problems [1, 2, 17, 22, 42].

This research direction falls under the broader framework of Learning to Optimize (L2O)[6, 13], which aims to leverage deep learning to improve the efficiency and scalability of optimization algorithms. Unlike traditional methods that rely on handcrafted update rules, L2O methods attempt to automatically learn optimization behaviors through data-driven approaches. However, most existing works consider unconstrained optimization problems. This motivates the development of more flexible frameworks that can incorporate feasibility into the learning dynamics while remaining scalable to large or structured problems.

In this work, we propose Descent-Net, a neural network architecture that takes as input the gradients of both the objective and constraint functions at a given feasible point. The network is trained to predict an effective descent direction and an appropriate step size, enabling objective improvement while maintaining feasibility. Initialized from feasible solutions obtained by methods such as DC3 [18], H-proj [28], etc., our method typically produces near-optimal solutions in just a few update steps.

Our main contributions are summarized as follows:

- We design a new exact penalty subproblem that generates feasible descent directions for linearly constrained optimization problems, forming the foundation of our approach with theoretical convergence guarantees.
- We propose a neural network architecture, **Descent-Net**, which unrolls a projected subgradient method to solve the proposed subproblem. The network iteratively refines feasible solutions by learning effective descent directions at each step.
- We validate the effectiveness and scalability of Descent-Net through extensive experiments on quadratic programs and a simple nonconvex variant of QP with linear constraints. We also include the nonlinear AC optimal power flow problem as an empirical extension to examine the practical applicability of the framework beyond the linearly constrained setting. The method scales to QP instances with up to 5 000 variables and to large-scale portfolio optimization tasks with up to 4 000 assets, consistently producing high-quality feasible solutions with low relative errors and strong computational efficiency.

2 Related work

Classical methods for constrained optimization. Classical approaches to constrained optimization, including projected gradient descent, feasible direction methods [35, 45], and primal-dual algorithms [9, 29, 30], have been extensively studied and widely applied. These methods typically offer convergence guarantees under suitable assumptions, but often suffer from high iteration complexity and significant computational cost. Recently, GPU-accelerated algorithms have also been developed, such as `HPR-QP` [11] and `CuClarabel` [15].

Learning to optimize (L2O). L2O seeks to replace hand-crafted optimization routines with learnable architectures that generalize across problem instances. Broadly speaking, L2O methods can be classified into model-free and model-based approaches [13]. Model-free methods, such as those based on recurrent neural networks (e.g., LSTM) [3, 23], aim to learn update rules directly from data. Model-based methods, on the other hand, incorporate algorithmic structure into the design of the network. Notable examples include LISTA [14], unrolled manifold optimization algorithms [21]. However, most existing L2O methods focus on unconstrained or simple constrained problems and fail to guarantee feasibility when applied to general constrained settings.

To address this, recent works incorporate constrained optimization structures into neural networks via projection layers [28, 41] or differentiable optimization modules [1, 2, 8]. However, these methods typically suffer from scalability and the need to solve nested optimization problems during training. Some approaches target special cases, such as linear constraints [37], but their applicability to more general problems remains limited. An alternative line of work draws inspiration from primal-dual methods, leading to neural architectures based on ADMM [40] and PDHG [27]. Such methods are usually evaluated by the KKT error, where feasibility and objective optimality are of the same order of magnitude, which makes them less suitable for scenarios requiring strict constraint satisfaction. Recent efforts have attempted to address this by designing networks that explicitly return feasible points [18, 38]; however, such methods still fall short of reaching near-optimal solutions in practice.

Implicit layers. A growing body of work explores the use of implicit neural architectures, including optimization layers [2], neural ordinary differential equations (ODEs) [12], and deep equilibrium models (DEQs) [4]. These models define network outputs via the solution of fixed-point or optimization problems, allowing compact yet highly expressive representations. Despite their potential, these approaches often incur high computational costs during both forward and backward passes. In the context of constrained optimization, additional challenges arise when estimating gradients of projection operators, particularly in the presence of complex or nonconvex constraints. Approximate techniques such as gradient perturbation or stochastic sampling [7, 32] have been proposed, but typically come at the expense of increased variance and computational overhead.

3 Problem setup

For any given data $x \in \mathbb{R}^d$, we solve the following constrained optimization problem

$$\min_{y\in\mathbb{R}^n} f_x(y), \quad \text{s.t.} \quad y \in \mathcal{C} := \{y \mid g_x(y) \leqslant 0,\, h_x(y) = 0\}, \tag{1}$$

where f_x, g_x, and h_x are smooth (but not necessarily convex) functions that may depend on the input data x. We assume that there are m equality constraints and l inequality constraints:

$$h_x(y) = [h_{x,1}(y), h_{x,2}(y), \cdots, h_{x,m}(y)]^\top = 0,$$
$$g_x(y) = [g_{x,1}(y), g_{x,2}(y), \cdots, g_{x,l}(y)]^\top \leqslant 0,$$

where $h_{x,i} : \mathbb{R}^n \to \mathbb{R}$ and $g_{x,j} : \mathbb{R}^n \to \mathbb{R}$ for all $i = 1, \cdots, m$ and $j = 1, \cdots, l$. We have the following common assumptions for this problem.

Assumption 1 *The feasible set $\mathcal{C}$ is non-empty and closed; the sub-level set $\{y \in \mathcal{C} \mid f_x(y) \leqslant f_x(y_0)\}$ is bounded.*

Assumption 2 *We assume that at any feasible point y, the gradients of the equality constraints, $\nabla h_{x,i}(y)$, for $i = 1, 2, \cdots, m$, are linearly independent.*

We will use the following LICQ condition to convert Fritz–John stationarity into KKT stationarity at accumulation points.

Assumption 3 (LICQ) *Whenever an algorithmic sequence considered in the convergence analysis has an accumulation point $\bar{y}$, LICQ holds at $\bar{y}$. That is, we assume that the set of active constraint gradients at $\bar{y}$,*
$\left\{\nabla h_{x,i}(\bar{y})\right\}_{i=1}^m \cup \left\{\nabla g_{x,j}(\bar{y})\right\}_{j\in\mathcal{A}(\bar{y})}$, *where* $\mathcal{A}(\bar{y}) := \{j \in \{1, \cdots, l\} \mid g_{x,j}(\bar{y}) = 0\}$, *is linearly independent.*

The notation $\mathcal{A}$ denotes the *active set*[1], i.e., the set of inequality constraints that are satisfied with equality.

Assumption 4 *Let $\mathcal{X} \subseteq \mathbb{R}^p$ be a compact set and assume that all training and test parameters satisfy $x \in \mathcal{X}$. For each $x \in \mathcal{X}$, consider the feasible set $\mathcal{C}$. We assume that:*

1. *(**Uniform boundedness of feasible sets**) There exists a compact set $Y \subseteq \mathbb{R}^n$ such that*
$$\mathcal{C} \subseteq Y \quad \text{for all } x \in \mathcal{X}.$$

[1]This is distinct from the standard definition of the active set, which typically includes the indices corresponding to the equality constraints.

2. ***(Smoothness and uniform gradient bound)*** *The functions f_x, h_x, g_x are continuously differentiable in y, and the maps*
$$(x, y) \mapsto \nabla_y f_x(y) \quad \text{and} \quad (x, y) \mapsto \nabla_y g_x(y)$$
are continuous on $\mathcal{X} \times Y$. Then, by compactness, there exist constants $L_f > 0$ and $L_g > 0$ such that
$$\|\nabla_y f_x(y)\|_2 \leqslant L_f \quad \text{and} \quad \|\nabla_y g_x(y)\|_2 \leqslant L_g \quad \text{for all } x \in \mathcal{X},\ y \in \mathcal{C}.$$

This assumption is reasonable, since practical training always involves a finite dataset, guaranteeing the existence of an appropriate upper bound.

Assumption 5 ***(Uniform Hoffman-type error bound).*** *For any fixed constant $M > 0$, for each $x \in \mathcal{X}$ and $y \in \mathcal{C}$, define*
$$E_x(y) := \nabla h_x(y)^\top, \qquad A_x(y) := \begin{bmatrix} \nabla g_{x,1}(y)^\top \\ \vdots \\ \nabla g_{x,l}(y)^\top \end{bmatrix}, \qquad b_x(y) := -M g_x(y).$$
Let
$$\widetilde{F}(x, y) := \{d \in \mathbb{R}^n \mid E_x(y)d = 0,\ A_x(y)d \leqslant b_x(y)\}.$$
We assume that there exists a constant $\kappa_H > 0$, independent of x and y, such that for all $x \in \mathcal{X}$, all $y \in \mathcal{C}$, and all $d \in \mathbb{R}^n$,
$$\operatorname{dist}\big(d, \widetilde{F}(x, y)\big) \leqslant \kappa_H \left(\|E_x(y)d\|_1 + \|(A_x(y)d - b_x(y))_+\|_1\right).$$

Assumption 5 is a uniform Hoffman-type error-bound condition for the family of linearized constraint systems. It will be used for later analysis. For each fixed linear system, such an error bound follows from the classical Hoffman lemma [24], with a constant depending only on the constraint matrices. In the present setting, however, the matrices $\nabla h_x(y)$ and $\nabla g_x(y)$ may vary with (x, y); hence the assumption requires the corresponding Hoffman constants to be uniformly bounded over the relevant set of feasible pairs (x, y).

3.1 Feasible directions method

The method of feasible directions (MFD) was originally developed by Zoutendijk in the 1960s [45]. However, a well-known drawback of MFD is that it may fail to converge due to the so-called jamming phenomenon. To address this issue, various fundamental modifications and extensions of MFD have since been proposed and studied [29, 31, 35, 45]. In this section, we briefly review the framework of MFD under the assumption that the constraints $h_x(y)$ and $g_x(y)$ are linear.

Based on the first-order approximation of the constraint functions, it can be inferred that, to maintain the feasibility of the solution, a suitable descent direction d at the current iterate y should satisfy the following conditions:

$$\begin{aligned} &\langle d, \nabla h_{x,i}(y)\rangle = 0, \quad \text{for } i = 1, \cdots, m, \\ &\langle d, \nabla g_{x,j}(y)\rangle \leqslant 0, \quad \text{for } j \in \mathcal{A} = \{1 \leqslant j \leqslant l :\ g_{x,j}(y) = 0\}, \end{aligned} \tag{2}$$

where $\nabla h_{x,i}$ denotes the gradient of the equality constraints and $\nabla g_{x,j}$ corresponds to the inequality constraints.

Zoutendijk Direction-Finding Subproblem. [45]

The Zoutendijk method computes a search direction $d \in \mathbb{R}^n$ at a feasible point y by solving the following linear program:

$$\begin{aligned} \min_{d\in\mathbb{R}^n} \quad & \nabla f_x(y)^\top d \\ \text{s.t.} \quad & \nabla h_x(y)^\top d = 0, \quad \nabla g_{x,j}(y)^\top d \leqslant 0, \quad j \in \mathcal{A}, \quad \|d\|_\infty \leqslant 1. \end{aligned} \tag{MFD}$$

Here, $\nabla h_x(y)^\top \in \mathbb{R}^{m\times n}$ denotes the Jacobian matrix of equality constraints.

The first constraint ensures that the direction is tangent to the equality constraint, while the second maintains feasibility with respect to the active inequalities. The infinity norm constraint serves to normalize the direction and keep the subproblem bounded.

The step size is then chosen as the largest feasible value such that $y + \alpha d$ remains in the feasible set:

$$\bar{\alpha} = \max\{\alpha \in (0, 1] \mid y + \alpha d \in \mathcal{C}\}.$$

However, when the iterate approaches the boundary of the feasible region, the step size $\bar{\alpha}$ may become arbitrarily small, potentially causing convergence issues [35].

Topkis–Veinott Uniformly Feasible Direction Subproblem [35]

To resolve this issue, Topkis and Veinott proposed a uniformly feasible direction (UFD) formulation:

$$\begin{aligned} \min_{d\in\mathbb{R}^n} \quad & \nabla f_x(y)^\top d \\ \text{s.t.} \quad & \nabla h_x(y)^\top d = 0, \quad \nabla g_{x,j}(y)^\top d \leqslant -M \cdot g_{x,j}(y), \quad j = 1, \cdots, l, \\ & \sum_{i=1}^{n} |d_i| = 1. \end{aligned} \tag{UFD}$$

The main difference lies in the fact that all inequality constraints are considered, and a constant $M > 0$ is introduced. Notably, setting $M = \infty$ recovers the original formulation in (MFD). This modification ensures that the computed direction d satisfies

$$g_{x,j}(y + \alpha d) \leqslant 0, \quad \text{for all } j, \quad \text{as long as } \alpha \leqslant \frac{1}{M},$$

thus providing a uniform lower bound on feasible step sizes and overcoming the stalling issues of the original method. It can be shown that the direction

obtained from (UFD) is a feasible descent direction. Moreover, under Assumption 3, any accumulation point of the iterates generated by this method [20, 45] satisfies the KKT conditions.

4 Algorithm

4.1 Reformulation of UFD subproblem

Our goal is to design a learning-to-optimize (L2O) algorithm for solving the structured problem described above. However, both the Zoutendijk and Topkis–Veinott methods require solving constrained subproblems at each iteration, which are not suitable for direct embedding into neural networks. To address this, we reformulate the subproblem by exact penalty method. This enables us to implement the solver as an unrolled optimization process of projected subgradient method, forming the basis of our L2O algorithm. In the following, we describe the reformulated subproblem and the corresponding L2O architecture.

Motivated by (MFD), we first formulate the following penalized subproblem:

$$\min_{d} \nabla f_x(y)^\top d + \sum_{j=1}^{l} c_j \max\left(\langle d, \nabla g_{x,j}(y)\rangle, -M g_{x,j}(y)\right), \quad \text{s.t.} \quad d \in \mathcal{D}, \tag{Pen}$$

where $\mathcal{D} = \{d : \|d\|_2 \leqslant 1 \text{ and } \langle d, \nabla h_{x,i}(y)\rangle = 0,\ \forall i = 1, \cdots, m\}$, $c_j > 0$ and $M > 0$ are the regularization parameters. We use the ℓ_2-ball normalization in (Pen) for algorithmic convenience. The norm constraint in MFD/UFD mainly serves to normalize the search direction and make the direction-finding subproblem bounded. In our unrolled architecture, however, the subproblem will be solved by projected subgradient iterations, and hence the efficiency of the projection step is important. The ℓ_2 ball, together with the linearized equality constraints, admits the simple closed-form projection in Proposition 1, which makes it suitable for neural network unrolling. Other normalizations, such as ℓ_∞ or ℓ_1, could also be used in principle, but they would lead to more complicated projection steps.

The following lemma gives a sufficient condition under which the hinge penalty is exact.

Lemma 1 (Exact hinge penalty) *Given any feasible point $y \in \mathcal{C}$, denote $c_{\min} := \min_{1 \leqslant j \leqslant l} c_j$. Under Assumptions 4 and 5, if*

$$c_{\min} > L_f \kappa_H, \tag{3}$$

where $c_{\min}$ is selected independently of x, then problem (Pen) *and* (UFD-L2) *have the same set of global minimizers:*

$$\begin{aligned} \min_{\|d\|_2 \leqslant 1} \quad & \nabla f_x(y)^\top d \\ \text{s.t.} \quad & \nabla h_x(y)^\top d = 0, \\ & \nabla g_{x,j}(y)^\top d \leqslant -M g_{x,j}(y), \quad j = 1, \cdots, l. \end{aligned} \tag{UFD-L2}$$

Proof For simplicity, define

$$p := \nabla f_x(y), \qquad E := \nabla h_x(y)^\top,$$

and

$$a_j := \nabla g_{x,j}(y), \qquad b_j := -M g_{x,j}(y), \quad j = 1, \cdots, l.$$

Let

$$A := \begin{bmatrix} a_1^\top \\ \vdots \\ a_l^\top \end{bmatrix}, \qquad b := (b_1, \cdots, b_l)^\top.$$

Also define

$$D := \left\{ d \in \mathbb{R}^n \;\middle|\; Ed = 0,\ \|d\|_2 \leqslant 1 \right\},$$
$$F := \{ d \in D \;\mid\; Ad \leqslant b \} = \{ d \in \mathbb{R}^n \mid Ed = 0, Ad \leqslant b, \|d\|_2 \leqslant 1 \},$$

and

$$\widetilde{F} := \left\{ d \in \mathbb{R}^n \;\middle|\; Ed = 0,\ Ad \leqslant b \right\}.$$

Since $y \in \mathcal{C}$, we have $g_x(y) \leqslant 0$, and hence

$$b = -M g_x(y) \geqslant 0.$$

Therefore $0 \in \widetilde{F}$ and $0 \in F$. In particular, both $\widetilde{F}$ and F are nonempty.

We first show that for every $d \in D$,

$$\operatorname{dist}(d, F) \leqslant \kappa_H \|(Ad - b)_+\|_1. \tag{4}$$

Since $d \in D$, we have $Ed = 0$. By Assumption 5, we have

$$\operatorname{dist}(d, \widetilde{F}) \leqslant \kappa_H \left(\|Ed\|_1 + \|(Ad - b)_+\|_1 \right) = \kappa_H \|(Ad - b)_+\|_1. \tag{5}$$

Let $\bar{d} := \Pi_{\widetilde{F}}(d)$ be the Euclidean projection of d onto $\widetilde{F}$. Since $0 \in \widetilde{F}$, we have $\Pi_{\widetilde{F}}(0) = 0$. By the nonexpansiveness of the projection onto a closed convex set,

$$\|\bar{d}\|_2 = \|\Pi_{\widetilde{F}}(d) - \Pi_{\widetilde{F}}(0)\|_2 \leqslant \|d\|_2 \leqslant 1.$$

Thus $\bar{d} \in F$. Combining this with (5) proves (4).

Now define the violation of feasibility vector

$$r(d) := (Ad - b)_+ = \left((a_1^\top d - b_1)_+, \cdots, (a_l^\top d - b_l)_+ \right)^\top.$$

For any $d \in D$, by (4), there exists $\bar{d} \in F$ such that

$$\|d - \bar{d}\|_2 \leqslant \kappa_H \|r(d)\|_1. \tag{6}$$

The penalized objective can be rewritten as

$$\Phi(d) = p^\top d + \sum_{j=1}^{l} c_j \max\{a_j^\top d, b_j\} = p^\top d + \sum_{j=1}^{l} c_j b_j + \sum_{j=1}^{l} c_j r_j(d). \tag{7}$$

Since $\bar{d} \in F$, we have $r(\bar{d}) = 0$, and hence

$$\Phi(\bar{d}) = p^\top \bar{d} + \sum_{j=1}^{l} c_j b_j. \tag{8}$$

Subtracting (8) from (7), we obtain

$$\Phi(d) - \Phi(\bar{d}) = p^\top (d - \bar{d}) + \sum_{j=1}^{l} c_j r_j(d). \tag{9}$$

By Assumption 4, we have the following uniform bound for all x, y:

$$\|p\|_2 = \|\nabla f_x(y)\|_2 \leqslant L_f.$$

Thus, using (6), we get

$$p^\top (d - \bar{d}) \geqslant -\|p\|_2 \|d - \bar{d}\|_2 \geqslant -L_f \kappa_H \|r(d)\|_1. \tag{10}$$

Moreover, we have

$$\sum_{j=1}^{l} c_j r_j(d) \geqslant c_{\min} \|r(d)\|_1. \tag{11}$$

Combining (9), (10), and (11), we get

$$\Phi(d) - \Phi(\bar{d}) \geqslant (c_{\min} - L_f \kappa_H) \|r(d)\|_1.$$

Since $c_{\min} > L_f \kappa_H$, it follows that

$$\Phi(d) > \Phi(\bar{d})$$

whenever $r(d) \neq 0$, i.e., whenever $d \notin F$.

Now let $d^\star$ be a global minimizer of (Pen). If $d^\star \notin F$, then $r(d^\star) \neq 0$. By the argument above, there exists $\bar{d}^\star \in F \subseteq D$ such that

$$\Phi(d^\star) > \Phi(\bar{d}^\star),$$

which contradicts the global optimality of $d^\star$ for (Pen). Therefore, every global minimizer of (Pen) belongs to F. Furthermore, suppose $d^\star$ is not a global minimizer of (UFD-L2), there would exist $\hat{d} \in F$ such that

$$p^\top \hat{d} < p^\top d^\star.$$

Since both $\hat{d}$ and $d^\star$ belong to F, we have

$$\Phi(\hat{d}) = p^\top \hat{d} + \sum_{j=1}^{l} c_j b_j < p^\top d^\star + \sum_{j=1}^{l} c_j b_j = \Phi(d^\star),$$

contradicting the global optimality of $d^\star$ for (Pen). Therefore every global minimizer of (Pen) is a global minimizer of (UFD-L2).

Conversely, let $d^\star$ be a global minimizer of (UFD-L2). For any $d \in D$, there are two cases.

If $d \in F$, then we have

$$\Phi(d) = p^\top d + \sum_{j=1}^{l} c_j b_j \geqslant p^\top d^\star + \sum_{j=1}^{l} c_j b_j = \Phi(d^\star).$$

If $d \notin F$, then by the preceding argument there exists $\bar{d} \in F$ such that

$$\Phi(d) > \Phi(\bar{d}).$$

Since $d^\star$ minimizes $p^\top d$ over F, we have

$$\Phi(\bar{d}) = p^\top \bar{d} + \sum_{j=1}^{l} c_j b_j \geqslant p^\top d^\star + \sum_{j=1}^{l} c_j b_j = \Phi(d^\star).$$

Therefore $\Phi(d) \geqslant \Phi(d^\star)$ for all $d \in D$, and hence $d^\star$ is also a global minimizer of (Pen).

Consequently, (Pen) and (UFD-L2) have the same set of global minimizers. This completes the proof. □

In the practical implementation, we use the following safeguarded adaptive weights:

$$c_j = c_0 + c_1 \frac{\|\nabla f_x(y)\|_2}{\varepsilon - \frac{1}{2} M g_{x,j}(y)}, \qquad j = 1, \cdots, l, \tag{12}$$

where $c_0 > 0$, $c_1 \geq 0$, and $\varepsilon > 0$. The constant term c_0 prevents the penalty parameter from vanishing, while the adaptive term assigns larger weights to nearly active constraints and thereby encourages the learned direction to satisfy the corresponding linearized inequalities more accurately. When a certified upper bound for $L_f \kappa_H$ is available, one may choose $c_0 > L_f \kappa_H$ to satisfy the condition in Lemma 1. Otherwise, the above formula should be viewed as a practical safeguarded weighting strategy.

Proposition 1 *Let $H \in \mathbb{R}^{n \times m}$ denote the matrix formed by the gradients of the equality constraints*

$$H = \left[\nabla h_{x,1}(y) \ \cdots \ \nabla h_{x,m}(y)\right].$$

Then, under Assumption 2, the expression for the projection onto $\mathcal{D}$ is given by

$$\mathcal{P}(d) = \begin{cases} \hat{d}, & \text{if } \|\hat{d}\|_2 \leqslant 1, \\ \hat{d}/\|\hat{d}\|_2, & \text{otherwise,} \end{cases} \quad \text{where} \quad \hat{d} = d - H(H^\top H)^{-1} H^\top d. \tag{13}$$

Proof Given any vector $d \in \mathbb{R}^n$, we aim to compute its projection onto $\mathcal{D}$, i.e., solve the following problem:

$$\min_{d' \in \mathbb{R}^n} \frac{1}{2}\|d' - d\|_2^2 \quad \text{s.t.} \quad \|d'\|_2 \leqslant 1, \quad H^\top d' = 0.$$

Under Assumption 2, the columns of H are linearly independent, which implies that $H^\top H$ is invertible. Now we derive the KKT conditions for this problem from the Lagrangian

$$\mathcal{L}(d', \lambda, \mu) = \frac{1}{2}\|d' - d\|_2^2 + \lambda^\top H^\top d' + \mu(\|d'\|_2^2 - 1),$$

where $\lambda \in \mathbb{R}^m$ and $\mu \geqslant 0$ are the Lagrange multipliers.

Taking the gradient with respect to d' and setting it to zero gives:

$$d' - d + H\lambda + 2\mu d' = 0 \quad \Rightarrow \quad (1 + 2\mu)d' + H\lambda = d.$$

Since $H^\top d' = 0$, we have

$$H^\top H\lambda = (1 + 2\mu)H^\top d' + H^\top H\lambda = H^\top d \quad \Rightarrow \quad \lambda = (H^\top H)^{-1} H^\top d.$$

We consider two cases: **Case 1:** If $\mu = 0$, then the projection is

$$d' = d - H\lambda = d - H(H^\top H)^{-1}H^\top d = \hat{d}.$$

Case 2: If $\mu > 0$, then we have $\|d'\| = 1$ and

$$(1+2\mu)^2 = (1+2\mu)^2 (d')^\top d' = (d - H\lambda)^\top (d - H\lambda) = \hat{d}^\top \hat{d} = \|\hat{d}\|^2.$$

Hence, the projection is:

$$d' = \frac{1}{1+2\mu}\left(d - H(H^\top H)^{-1}H^\top d\right) = \frac{1}{\|\hat{d}\|}\hat{d}.$$

□

Consequently, the procedure of the projected subgradient method for solving this problem is as follows:

$$d_{k+1} = \mathcal{P}\big(d_k - \gamma_k \mathbf{u}_k\big), \tag{14}$$

where $\gamma_k > 0$ is the step size and $\mathcal{P}$ is the projection operator defined in (13). Let $\mathbf{1}_{\{.\}}$ denote the indicator function, the subgradient term $\mathbf{u}_k$ is given by

$$\mathbf{u}_k = \nabla f_x(y) + \sum_{j=1}^{l} c_j \mathbf{1}_{\{\langle d^k, \nabla g_{x,j}(y)\rangle \geqslant -M g_{x,j}(y)\}} \nabla g_{x,j}(y). \tag{15}$$

Note that in many practical problems, the matrix H is fixed across instances or does not change frequently. In such cases, the projection in (13) can be precomputed, making the computational cost manageable. Examples include decision-focused learning setups, where the equality constraints remain constant across instances [34]. In other problems, the equality constraints are simple, allowing the projection to be computed efficiently; for instance, in classical portfolio optimization[19], the budget constraint enables a straightforward projection.

4.2 Design of Descent Module

Directly solving problem (Pen) using the projected subgradient method usually results in slow convergence, due to the use of diminishing step size. To address this, we propose Descent-Module, which is designed by unrolling the projected subgradient algorithm. In our proposed network architecture, each layer takes the form of one iteration of the projected (sub)gradient method,

$$d_{k+1} = \mathcal{P}\big(d_k - \gamma_k T^k(\mathbf{u}_k)\big). \tag{16}$$

The key difference is that we apply learnable modules T^k to the subgradient term $\mathbf{u}_k$, and the definition of T^k is given as follows:

$$T^k(\mathbf{u}_k) = \mathbf{V}^k \mathrm{ReLU}\left(\mathbf{W}^k \mathbf{u}_k + \mathbf{b}_1^k\right) + \mathbf{b}_2^k, \tag{17}$$

where $\mathbf{W}^k \in \mathbb{R}^{q\times n}, \mathbf{V}^k \in \mathbb{R}^{n\times q}$ and $\mathbf{b}_1^k \in \mathbb{R}^q, \mathbf{b}_2^k \in \mathbb{R}^n$ are the weight matrix and bias that we need to learn and $\mathrm{ReLU}(x) = \max(x, 0)$. The design of the operator T follows the work in [39], which demonstrates that such an architecture has strong universal approximation capabilities.

The step size γ_k is also set as a learnable parameter, which avoids the need for manual tuning, and we provide in Appendix E the learned values of γ_k across different layers in our experiments.

The architectures of the Descent Module are illustrated in Figure 1. Each Descent Module consists of K Descent Layers sharing the same architecture and the input to the first layer is chosen as $d_0 = -\nabla f_x(y)$.

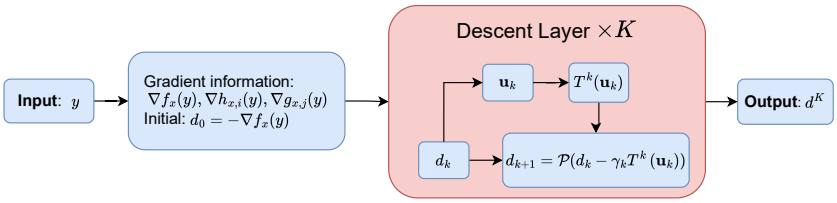

Fig. 1 Overall structure of the Descent Module

Theorem 1 (Representability of projected subgradient iterations) *Let d^* be an optimal solution of Problem* (Pen)*, and let g denote its objective function. For any $\varepsilon > 0$, there exists a K-layer Descent-Module with a parameter assignment independent of the particular instance x within the considered problem class, where $K \leqslant C/\varepsilon^2$ for some constant $C > 0$, such that the module reproduces the first K projected subgradient iterates for Problem* (Pen)*. Consequently, at least one intermediate layer output d_k, $1 \leqslant k \leqslant K$, satisfies*

$$g(d_k) - g(d^*) \leqslant \varepsilon.$$

The proof of Theorem 1 relies on two technical lemmas, which we present below before giving the full argument.

Lemma 2 *For any $\varepsilon > 0$, there exists a constant $C > 0$ such that if we set $K = \frac{C}{\varepsilon^2}$ and choose the step size in (14) as $\gamma_k = \frac{1}{\sqrt{K}}$, then*

$$\min_{1\leqslant k\leqslant K} g(d_k) - g(d^*) \leqslant \varepsilon,$$

*where d^* denotes the optimal solution of Problem (*Pen*) and g denote its objective function.*

This lemma is a direct consequence of the classical convergence theory of the projected subgradient method, and we therefore omit its proof.

Lemma 3 *Assume that the hidden width satisfies $q \geqslant n$. Given the sequence of iterates $\{d_1^{proj}, \cdots, d_K^{proj}\}$ generated by the projected subgradient method (14) with initial input d_0, there exists a K-layer Descent Module with a specific parameter assignment that, starting from the same initial input d_0, it produces the same iterative sequence, i.e., $d_k = d_k^{proj}$ for all $1 \leqslant k \leqslant K$.*

Proof It suffices to show that there exists a set of parameters such that $T^k(\mathbf{u}_k) = \mathbf{u}_k$ for all $1 \leqslant k \leqslant K$, where $\mathbf{u}_k$ is defined in (15).

Let $\mathbf{W}^k \in \mathbb{R}^{q\times n}$ be a full column rank matrix , so its left pseudo-inverse $(\mathbf{W}^k)^\dagger \in \mathbb{R}^{n\times q}$ exists and satisfies $(\mathbf{W}^k)^\dagger \mathbf{W}^k = I_n$.

By Assumption 4 and the definition of $\mathbf{u_k}$, we have

$$\begin{aligned}\|\mathbf{u}_k\|_2 &\leqslant \|\nabla f_x(y)\|_2 + \sum_{j=1}^{l} |c_j| \cdot \sqrt{n} \cdot |\nabla g_{x,j}(y)| \\ &\leqslant \|\nabla f_x(y)\|_2 + \max_j(c_j) \cdot \sqrt{n} \cdot \|\nabla g_x(y)\|_1 \\ &\leqslant \|\nabla f_x(y)\|_2 + \max_j(c_j) \cdot n \cdot \|\nabla g_x(y)\|_2 \\ &\leqslant L_f + \max_j(c_j) \cdot n \cdot L_g\end{aligned}$$

Then, we have

$$\|W^k u_k\|_\infty \leqslant \|W^k u_k\|_2 \leqslant \|W^k\|_2 U, \quad \textit{with} \quad U = L_f + \max_j(c_j) \cdot n \cdot L_g.$$

Let $L := \|W^k\|_2 U$ and set $b_1^k = L\mathbf{1}_q$, where $\mathbf{1}_q$ denotes the q-dimensional vector with all entries equal to one. Then every coordinate of $W^k u_k + b_1^k$ is nonnegative, and hence

$$\mathrm{ReLU}(W^k u_k + b_1^k) = W^k u_k + b_1^k.$$

Now, let the second layer weight be $\mathbf{V}^k = (\mathbf{W}^k)^\dagger$, and the second bias be $\mathbf{b}_2^k = -(\mathbf{W}^k)^\dagger \mathbf{b}_1^k$. Then we have:

$$T^k(\mathbf{u}_k) = \mathbf{V}^k \cdot \mathrm{ReLU}(\mathbf{W}^k \mathbf{u}_k + \mathbf{b}_1^k) + \mathbf{b}_2^k = (\mathbf{W}^k)^\dagger (\mathbf{W}^k \mathbf{u}_k + \mathbf{b}_1^k) - (\mathbf{W}^k)^\dagger \mathbf{b}_1^k = \mathbf{u}_k.$$

This completes the proof. □

We now proceed to prove Theorem 1.

Proof By Lemma 2, choose $K \leqslant C/\varepsilon^2$ such that

$$\min_{1\leqslant k\leqslant K} g(d_k^{\mathrm{proj}}) - g(d^*) \leqslant \varepsilon.$$

Let

$$k_\varepsilon \in \arg\min_{1\leqslant k\leqslant K} g(d_k^{\mathrm{proj}}).$$

By Lemma 3, there exists a K-layer Descent-Module whose layer outputs satisfy $d_k = d_k^{\mathrm{proj}}$ for all $1 \leqslant k \leqslant K$. Therefore,

$$g(d_{k_\varepsilon}) - g(d^*) = g(d_{k_\varepsilon}^{\mathrm{proj}}) - g(d^*) \leqslant \varepsilon.$$

Thus at least one intermediate layer output satisfies the desired bound. □

Theorem 1 establishes a theoretical connection between the Descent-Module and the classical projected subgradient method. Specifically, it shows that the Descent-Module is sufficiently expressive to reproduce the projected subgradient iterations for solving the penalized subproblem. This result should be interpreted as a representational guarantee rather than an explanation of the empirical performance gains of the trained model. In practice, the trainable parameters are learned from data and may produce more efficient update patterns than explicitly mimicking the classical projected subgradient method.

4.3 Step size

After obtaining the descent direction d from the Descent module, we still need to determine a suitable step size. We assume that all constraints are linear. Since the Descent module contains the projection operator $\mathcal{P}$, the final descent direction d produced by Descent module is orthogonal to the gradients of the equality constraints. Therefore, updating along d will not violate the equality constraints.

We only need to ensure that the step size is not too large to violate the inequality constraints. For each linear inequality constraint $g_{x,j}$, we have:

$$g_{x,j}(y + \alpha d) = g_{x,j}(y) + \alpha \cdot \langle d, \nabla g_{x,j}(y) \rangle.$$

If $\langle d, \nabla g_{x,j}(y) \rangle > 0$, updating the solution along d will increase $g_{x,j}$. To preserve the feasibility of the inequality constraint, i.e., $g_{x,j}(y + \alpha d) \leqslant 0$, the step size α must satisfy

$$\alpha \leqslant \frac{-g_{x,j}(y)}{\langle d, \nabla g_{x,j}(y) \rangle}.$$

Therefore, the maximum allowable step size is given by

$$\alpha_{\max} = \min_{j \in \mathcal{I}} \frac{-g_{x,j}(y)}{\langle d, \nabla g_{x,j}(y) \rangle}, \quad \text{where } \mathcal{I} = \{j \mid \langle d, \nabla g_{x,j}(y) \rangle > 0\}. \tag{18}$$

Note that if $I = \emptyset$, then no inequality constraint increases along d, and we set $\alpha_{\max} = 1/M$. To encourage objective decrease while preserving feasibility, the step size α should also satisfy $f_x(y + \alpha d) < f_x(y)$. We introduce a learnable parameter $\beta \in \mathbb{R}$ and use the sigmoid function $\sigma(\cdot)$ to map it into $(0, 1)$. We then use this factor to scale $\alpha_{\max}$, and the final update rule for y is

$$y^{\text{new}} = y^{\text{old}} + \sigma(\beta)\alpha_{\max} \cdot d. \tag{19}$$

In addition, if the descent direction d obtained from the Descent-Net is the optimal solution of Problem (Pen), Lemma 1 ensures that a fixed step size of $\alpha = 1/M$ is feasible. However, we found that such a fixed step size does not perform well in practice, and in the appendix C we provide a comparison of different step-size selection strategies.

4.4 Descent-Net

Our Descent-Net consists of S Descent Modules. The input of the network is an initial feasible solution y_0 of Problem (1). At each stage, the s-th module takes the gradient information at the current iterate y_s and outputs a descent direction d_s, which is then used to update the solution to y_{s+1} according to the update rule (19). By repeatedly updating the solution in this manner, the network finally produces a high-accuracy feasible solution y_S. The overall procedure of the proposed method is summarized in Algorithm 1.

Algorithm 1 Descent-Net

1: **Input:** initial feasible point $y_0 \in \mathcal{C}$, S modules and K layers in each module.
2: **Learnable parameters:** $\Theta_s := \{\mathbf{V}_s^k, \mathbf{W}_s^k, \mathbf{b}_{1,s}^k, \mathbf{b}_{2,s}^k, \gamma_{k,s}\}_{k=0,1,\cdots,K-1}$ for $s = 0, \cdots, S-1$, together with $\{\beta_s\}_{s=0,\cdots,S-1}$.
3: **for** $s = 0, 1, \cdots, S-1$ **do**
4: $\quad d_0 = -\nabla f_x(y_s)$
5: $\quad$ **for** $k = 0, \cdots, K-1$ **do**
6: $\quad\quad \mathbf{u}_k = \nabla f_x(y_s) + \sum_{j=1}^{l} c_j \mathbf{1}_{\{\langle d_k, \nabla g_{x,j}(y_s)\rangle \geqslant -M g_{x,j}(y_s)\}} \nabla g_{x,j}(y_s)$
7: $\quad\quad d_{k+1} = \mathcal{P}\left(d_k - \gamma_{k,s} T_s^k(\mathbf{u}_k)\right)$, where $\mathcal{P}$ is defined in (13) and T_s^k is defined in (17) with parameters $\{\mathbf{V}_s^k, \mathbf{W}_s^k, \mathbf{b}_{1,s}^k, \mathbf{b}_{2,s}^k\}$
8: $\quad$ **end for**
9: $\quad y_{s+1} = y_s + \sigma(\beta_s) \cdot \alpha_s d_K$ as defined by (19), where α_s is obtained by (18).
10: **end for**
11: Train the parameters with loss: $\ell_p(y) = f_x(y) + \lambda_g \|\mathrm{ReLU}(g_x(y))\|_1 + \lambda_h \|h_x(y)\|_1$
12: **Output:** y_S.

We briefly analyze the model complexity in terms of parameter size and computational cost. The total number of learnable parameters is determined by (n, q, S, K), since each Descent-Module contains matrix parameters of size $n \times q$ and $q \times n$, repeated over K layers and S modules. For computational cost, both training and inference are dominated by matrix multiplications. In each Descent-Module, the main operations consist of multiplying an n-dimensional vector by an $n \times q$ matrix and then by a $q \times n$ matrix, leading to a complexity of $O(nq)$ per module. Over S modules and K layers, the overall computational cost is $O(SKnq)$ for both training and inference. In practice, q is typically chosen as a constant multiple of n, while S and K are small constants. Therefore, the overall complexity scales quadratically with n, i.e., $O(n^2)$ under typical configurations.

We emphasize that the following result is an existence statement for an idealized parameterization of the proposed architecture. The convergence guarantee is inherited from the idealized UFD-penalty method with exact subproblem solutions, whereas the role of Descent-Net is to show that the proposed

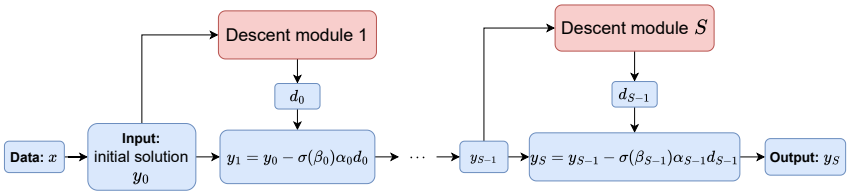

Fig. 2 Architecture of the entire network.

architecture has sufficient representational capacity to reproduce this idealized procedure.

Theorem 2 (Existence of an Idealized Descent-Net Parameterization) *Suppose Assumptions* 1-5 *hold. In addition, assume that h_x and g_x are linear and that ∇f_x is Lipschitz continuous on the relevant sublevel set. Let the penalty parameters satisfy the condition in Lemma* 1*, namely $c_{\min} > L_f \kappa_H$. Given an initial feasible solution y_0 of problem* (1)*, the idealized UFD-penalty method with exact solutions of the direction-finding subproblems generates a sequence whose every accumulation point satisfies the KKT conditions of problem* (1)*. Moreover, on any finite horizon, the Descent-Net architecture can reproduce the corresponding finite projected-subgradient implementation of the UFD-penalty method under a suitable parameter assignment.*

To prove Theorem 2, we require several technical definitions and lemmas, which we present below before proceeding to the proof.

Definition 1 (Fritz–John point) Let $y \in \mathbb{R}^n$ be a feasible point for the problem

$$\min f(y) \quad \text{s.t.} \quad h_i(y) = 0,\ g_j(y) \leqslant 0.$$

Then y is called a *Fritz–John point* if there exist multipliers $\lambda_0 \geqslant 0$, $\lambda_j \geqslant 0$ for $j = 1, \cdots, l$, and $\mu_i \in \mathbb{R}$ for $i = 1, \cdots, m$, not all zero, such that

$$\lambda_0 \nabla f(y) + \sum_{j=1}^{l} \lambda_j \nabla g_j(y) + \sum_{i=1}^{m} \mu_i \nabla h_i(y) = 0,$$
$$\lambda_j \cdot g_j(y) = 0, \quad j = 1, \cdots, l.$$

Definition 2 (KKT point) A feasible point $y \in \mathbb{R}^n$ is called a *Karush–Kuhn–Tucker (KKT) point* if there exist multipliers $\lambda_j \geqslant 0$ and $\mu_i \in \mathbb{R}$ such that

$$\nabla f(y) + \sum_{j=1}^{l} \lambda_j \nabla g_j(y) + \sum_{i=1}^{m} \mu_i \nabla h_i(y) = 0,$$
$$\lambda_j \cdot g_j(y) = 0, \quad j = 1, \cdots, l.$$

If the LICQ condition holds at $\bar{y}$, then the Fritz-John point is also a KKT point.

Lemma 4 (Farkas Lemma with equality constraints) *Let $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times n}$, and $b \in \mathbb{R}^m$.*

Then exactly one of the following two systems has a solution:

(a) There exists $x \in \mathbb{R}^n$ such that

$$Ax < b, \quad Bx = 0.$$

(b) There exists $(\lambda, \mu) \in \mathbb{R}^m \times \mathbb{R}^p$, not both zero, such that

$$\lambda \geqslant 0, \quad A^\top \lambda + B^\top \mu = 0, \quad \lambda^\top b \leqslant 0.$$

Moreover, both systems cannot be simultaneously feasible.

We now show that the problem (UFD-L2) has negative value if y is not a Fritz-John point.

Lemma 5 (Descent direction under failure of Fritz–John) *Let $\bar{y} \in \mathcal{C}$ be a feasible point. Suppose that the set of vectors*

$$\left\{ v = \lambda_0 \nabla f_x(\bar{y}) + \sum_{j=1}^{l} \lambda_j \nabla g_j(\bar{y}) + \sum_{i=1}^{m} \mu_i \nabla h_i(\bar{y}) \;\middle|\; \lambda_0 \geqslant 0,\ \lambda_j \geqslant 0,\ (\lambda, \mu) \neq 0,\ \lambda_j g_j(\bar{y}) = 0 \right\}$$

does not contain the zero vector. That is, $\bar{y}$ is not a Fritz–John point.

Then there exists a vector $d \in \mathbb{R}^n$ such that:

- $\nabla h_x(\bar{y})^\top d = 0$,
- $\nabla f_x(\bar{y})^\top d < 0$,
- $\nabla g_j(\bar{y})^\top d < -M g_j(\bar{y})$ *for all* $j = 1, \cdots, l$,
- $\|d\|_2 \leqslant 1$.

Proof Let $H := [\nabla h_1(\bar{y}), \cdots, \nabla h_m(\bar{y})] \in \mathbb{R}^{n\times m}$, and define the subspace of directions satisfying the linearised equality constraints:

$$\mathcal{T} := \left\{ d \in \mathbb{R}^n \mid H^\top d = 0 \right\}.$$

Let $A \in \mathbb{R}^{(l+1)\times n}$ be the matrix whose rows are:

$$A := \begin{bmatrix} \nabla f_x(\bar{y})^\top \\ \nabla g_1(\bar{y})^\top \\ \vdots \\ \nabla g_l(\bar{y})^\top \end{bmatrix}, \qquad b := \begin{bmatrix} 0 \\ -M g_1(\bar{y}) \\ \vdots \\ -M g_l(\bar{y}) \end{bmatrix}.$$

Then we consider the system:

$$Ad < b, \quad \text{subject to } H^\top d = 0.$$

Since $\bar{y}$ is not a Fritz–John point, the system of equalities

$$\lambda_0 \nabla f_x(\bar{y}) + \sum_j \lambda_j \nabla g_j(\bar{y}) + \sum_i \mu_i \nabla h_i(\bar{y}) = 0 \quad \text{with } \lambda_0 \geqslant 0,\ \lambda_j \geqslant 0, \mu_i \text{ not all zero},$$

has no solution satisfying the complementarity condition $\lambda_j g_j(\bar{y}) = 0$.

Therefore, by the Farkas Lemma 4, the dual system:

$$\text{find } d \in \mathcal{T} \text{ such that } Ad < b$$

is feasible.

Because A, b are fixed and $b_j = -Mg_j(\bar{y}) \geqslant 0$, which is finite for all $j = 1, \cdots, l$, and $\mathcal{T}$ is a linear subspace, the feasible set is convex and open in $\mathcal{T}$. We can scale d such that $\|d\|_2 \leqslant 1$.

Hence, such a direction d exists satisfying the claimed conditions. □

By Lemma 1 and Theorem 1, to establish Theorem 2 it suffices to show that if the following algorithm—constructed from the subproblem (Pen)—converges to a KKT point. We therefore begin by analyzing the convergence of the algorithm stated below.

Algorithm (UFD–penalty method).

Given a feasible starting point $y_0 \in \mathcal{C}$, repeat for $k = 0, 1, \cdots$

1. With the condition (3) holds, solve the sub–problem (Pen) at the current iterate y_k and obtain a minimizer d_k.
2. Update $y_{k+1} = y_k + \alpha_k d_k$, where $\alpha_k := \arg\min_\alpha \{f_x(y_k + \alpha d_k) \mid \alpha \in [0, 1/M]\}$, (*Lemma 1 implies* $y_{k+1} \in \mathcal{C}$)

We have the following result, which is similar to the Topkis–Veinott method [20, 45].

Theorem 3 (global convergence of the UFD–L_2 method) *Suppose Assumptions* 1,2, 4,5 *hold. Furthermore, we assume that the gradient* $\nabla f_x(y)$ *is* $L-$*Lipschitz continuous,* h_x, g_x *are linear, and the penalty parameters satisfy* $c_{\min} > L_f \kappa_H$. *Then every accumulation point of* $\{y_k\}$ *is a Fritz–John point of* (1). *Assume in addition that Assumption* 3 *holds, then every accumulation point is a KKT point.*

Proof **(i) Feasibility and boundedness.** The proof of Lemma 1 shows that every d_k satisfies $\nabla g_j(y_k)^\top d_k \leqslant -Mg_j(y_k)$, hence $g_j(y_{k+1}) = g_j(y_k) + \alpha \nabla g_j(y_k)^\top d_k \leqslant 0$, where $\alpha \in [0, 1/M]$. Equality constraints are preserved by $\nabla h_x(y_k)^\top d_k = 0$, so $y_{k+1} \in \mathcal{C}$. Because $\{y \in \mathcal{C} \mid f_x(y) \leqslant f_x(y_0)\}$ is bounded, $\{y_k\}$ is bounded and admits convergent subsequences.

(ii) every accumulation point is a Fritz–John point.

Let $\bar{y}$ be an accumulation point of $\{y_k\}$, extracted from a subsequence $\{y_{k_s}\}$. Suppose, for contradiction, that $\bar{y}$ is not a Fritz–John point.

Then, by Lemma 5, there exist $z < 0$ and a direction $d \in \mathbb{R}^n$ satisfying:

$$\|d\|_2 \leqslant 1, \quad \nabla h_x(\bar{y})^\top d = 0, \quad \nabla f_x(\bar{y})^\top d < z < 0, \quad \nabla g_j(\bar{y})^\top d < -Mg_j(\bar{y}) + z.$$

Since all the above inequalities are strict and the number of constraints is finite, there exists a constant $\rho > 0$ such that

$$\nabla f_x(\bar{y})^\top d \leqslant -\rho,$$

and

$$\nabla g_{x,j}(\bar{y})^\top d + M g_{x,j}(\bar{y}) \leqslant -\rho, \qquad j = 1, \cdots, l.$$

By the continuity of ∇f_x, $\nabla g_{x,j}$, and $g_{x,j}$, for all sufficiently large s,

$$\nabla f_x(y_{k_s})^\top d \leqslant -\frac{\rho}{2},$$

and

$$\nabla g_{x,j}(y_{k_s})^\top d + M g_{x,j}(y_{k_s}) \leqslant -\frac{\rho}{2}, \qquad j = 1, \cdots, l.$$

Equivalently,

$$\nabla g_{x,j}(y_{k_s})^\top d \leqslant -M g_{x,j}(y_{k_s}) - \frac{\rho}{2} < -M g_{x,j}(y_{k_s}), \qquad j = 1, \cdots, l.$$

Moreover, since h_x is linear and

$$\nabla h_x(\bar{y})^\top d = 0,$$

we have

$$\nabla h_x(y_{k_s})^\top d = 0.$$

Therefore, for all sufficiently large s, d is feasible for the UFD-L2 subproblem at y_{k_s}.

Let d_{k_s} be an optimal solution of the UFD-L2 subproblem at y_{k_s}, and set

$$z_s := \nabla f_x(y_{k_s})^\top d_{k_s}.$$

By the optimality of d_{k_s}, we obtain

$$z_s = \nabla f_x(y_{k_s})^\top d_{k_s} \leqslant \nabla f_x(y_{k_s})^\top d \leqslant -\frac{\rho}{2} < 0.$$

Since ∇f_x is L-Lipschitz continuous and $\|d_{k_s}\|_2 \leqslant 1$, for any $\alpha \in [0, 1/M]$, we have

$$f_x(y_{k_s} + \alpha d_{k_s}) \leqslant f_x(y_{k_s}) + \alpha \nabla f_x(y_{k_s})^\top d_{k_s} + \frac{L}{2}\alpha^2.$$

Using $z_s = \nabla f_x(y_{k_s})^\top d_{k_s} \leqslant -\rho/2$, we obtain

$$f_x(y_{k_s} + \alpha d_{k_s}) \leqslant f_x(y_{k_s}) - \frac{\rho}{2}\alpha + \frac{L}{2}\alpha^2.$$

Choose

$$\bar{\alpha} = \min\left\{\frac{1}{M}, \frac{\rho}{2L}\right\}$$

when $L > 0$. If $L = 0$, choose any fixed $\bar{\alpha} \in (0, 1/M]$. Then there exists a constant $\eta > 0$, independent of s, such that

$$f_x(y_{k_s} + \bar{\alpha} d_{k_s}) \leqslant f_x(y_{k_s}) - \eta.$$

Since α_{k_s} is chosen by exact line search over $[0, 1/M]$, it follows that

$$f_x(y_{k_s+1}) \leqslant f_x(y_{k_s} + \bar{\alpha} d_{k_s}) \leqslant f_x(y_{k_s}) - \eta.$$

On the other hand, $\{f_x(y_k)\}$ is nonincreasing and bounded below on the bounded sublevel set in Assumption 1, and hence it converges. Therefore both $\{f_x(y_{k_s})\}$ and $\{f_x(y_{k_s+1})\}$ converge to the same limit, which contradicts the inequality above. Hence every accumulation point $\bar{y}$ must be a Fritz–John point.

(iii) LICQ ⇒ KKT. Under LICQ the Fritz–John multipliers have $\lambda_0 > 0$, so the KKT system holds at $\bar{y}$. □

5 Experiment

We evaluate our Descent-Net on three types of problems: convex quadratic programs, a simple class of non-convex optimization problems, and the AC optimal power flow (AC-OPF) problem. To further evaluate the scalability of our method and to demonstrate its performance on a practical real-world instance of quadratic programming, we additionally include a large-scale portfolio optimization task. Detailed experimental settings are provided in Appendix A.

5.1 Baselines and Evaluation Criteria

We compare our method against several benchmarks, including:

- **Optimizer**: Traditional solvers include `OSQP` [33] and `qpth` [2] for convex QPs; the GPU-accelerated `HPR-QP` [11] and `CuClarabel` [15], both of which are executed on an H100 GPU in our experiments; `IPOPT` [36] for general non-convex problems; and the `PYPOWER` solver, a Python port of `MATPOWER` [44], for AC-OPF.
- **DC3**[18]: The full DC3 framework that combines both completion and correction operators.
- **Projection method**:Trains an MLP and projects its output onto the feasible set. For problems with linear constraints (convex QP and simple non-convex cases), the projection is solved using OptNet [2]. For the AC-OPF problem, the projection follows the differentiable solver of [10].
- **Warm start**: The infeasible NN prediction is directly used as the warm-start for the optimizer of [10], following the warm-starting schemes of [16] and [5].
- **CBWF**[38]: Inspired by the classical active set method, this approach explores the boundaries around inequality constraints and updates the initial solution to obtain a better objective value.
- **DeepOPF-V[26]**: a supervised learning approach for solving AC-OPF problems. It trains a fully connected neural network using optimal solutions as labels to predict bus voltage magnitudes, and then reconstructs generator dispatch through power flow equations and a post-processing step to enforce operational constraints.
- **DeepOPF-NGT[25]**: an unsupervised learning method for AC-OPF. It minimizes a loss function consisting of the objective value and constraint violations, and predicts bus voltages from which all remaining variables are recovered using power flow equations.

Other approaches reported in the literature, such as simple neural network models trained with penalty functions, are not included due to poor constraint satisfaction.

The performance of all methods is assessed according to the following criteria:

- **Feasibility:** measured by the average constraint violation of both equality and inequality constraints, i.e., $\frac{1}{m}\sum_{i=1}^{m}|h_{x,i}(y)|$ and $\frac{1}{l}\sum_{j=1}^{l}\mathrm{ReLU}\left(g_{x,j}(y)\right)$.

- **Optimality:** measured by the average relative and absolute errors (in the ℓ_1 norm) for both the solution and the objective value, where the optimal solution is approximated by optimizer.
- **Efficiency:** the computational time. The `Init` term denotes the initialization time required to construct a feasible starting point. The `Refine` term corresponds to the additional time spent on iterative refinement steps after initialization. For neural network-based methods, we report the batched inference time `Batched inf`. For classical solvers, we report the single-instance runtime `Per-inst`. We do not convert batched time to per-instance time for neural networks, since it does not correspond to the runtime of processing a single instance independently due to different execution behavior. For classical solvers, scaling by batch size is also not meaningful since they can be executed in parallel across instances using multiple CPUs.

For neural network-based methods, the model is trained once per problem setting and reused across many instances from the same distribution, making the training cost a one-time overhead that can be amortized over repeated use. Although this cost is not negligible, it does not affect the per-instance inference efficiency reported in the experimental results, and the proposed method remains computationally advantageous in terms of overall runtime due to its significantly faster inference.

5.2 Convex quadratic programs

We first consider convex QPs with quadratic objectives and linear constraints:

$$\min_{y\in\mathbb{R}^n} \frac{1}{2}y^\top Qy + p^\top y, \quad \text{s.t. } Ay = x,\ Gy \leqslant h, \tag{20}$$

where $Q \in \mathbb{R}^{n\times n} \succeq 0$, $p \in \mathbb{R}^n$, $A \in \mathbb{R}^{n_{eq}\times n}$, $G \in \mathbb{R}^{n_{ineq}\times n}$, and $h \in \mathbb{R}^{n_{ineq}}$ are fixed. The input $x \in \mathbb{R}^{n_{eq}}$ varies across problem instances, and the goal is to approximate the optimal y given x.

We generated 10 000 instances of x for three problem sizes, $n = 100$, $n = 1\,000$, and $n = 5\,000$. For all settings, Descent-Net uses the solutions produced by the DC3 model as its initial points. For $n = 100$, the experimental results are summarized in Table 1, and the final solutions obtained by Descent-Net attain a relative objective error of 3.1×10^{-4}.

For the larger scales $n = 1\,000$ and $n = 5\,000$, the corresponding results are presented in Table 2 and Table 3, respectively. Descent-Net achieves relative objective errors of 5.8×10^{-3} for $n = 1\,000$ and 3.2×10^{-3} for $n = 5\,000$. The lower accuracy compared to the $n = 100$ case can be attributed to the fact that the matrices have condition numbers on the order of n^2, causing the underlying solution map to become increasingly ill-conditioned as n grows and thus making the approximation task more challenging for the neural network.

Table 1 Results on the convex QP task evaluated on the test set with 833 samples, with problem dimension $n = 100$, $n_{eq} = 50$, and $n_{ineq} = 50$. The reported runtime corresponds to the average time to process one batch, where the batch size is 833.

Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Init. (s)	Refine. (s)	Batched inf. (s)	Per-inst. (s)
OSQP	0.000 0	0.000 0	0.0	0.0	–	–	–	0.002 8
HPR-QP	0.000 0	0.000 0	2.3×10^{-8}	6.3×10^{-9}	–	–	–	0.108 0
CuClarabel	0.000 0	0.000 0	1.6×10^{-3}	1.8×10^{-5}	–	–	–	0.053 9
qpth	0.000 0	0.000 0	2.4×10^{-6}	4.7×10^{-10}	–	–	–	1.759 3
Projection method	0.000 0	0.000 0	3.2×10^{-2}	8.4×10^{-4}	–	–	0.212 4	–
CBWF	0.000 0	0.000 0	2.1×10^{-1}	6.6×10^{-2}	0.003 6	0.033 0	0.036 6	–
DC3	0.000 0	0.000 0	5.2×10^{-1}	4.2×10^{-1}	–	–	0.003 6	–
Descent-Net	0.000 0	0.000 0	1.2×10^{-2}	3.1×10^{-4}	0.003 6	0.009 6	0.013 2	–

Nonetheless, the speed advantage of Descent-Net becomes more pronounced as the problem scale increases. For a fair comparison, we also include `HPR-QP fast` and `CuClarabel fast`, where each solver is terminated once it reaches the same accuracy level as Descent-Net (a relative objective error of about 10^{-3}). Both methods require more time than Descent-Net to attain this accuracy. For the largest-scale setting, Descent-Net processes a batch of 8 instances in 0.015 3 seconds, while OSQP requires 13.460 9 seconds per instance. This indicates a substantial throughput advantage, although the two timings reflect different execution modes.

Table 2 Results on the convex QP task evaluated on the test set with 833 samples, with problem dimension $n = 1\,000$, $n_{eq} = 500$, and $n_{ineq} = 500$. The reported runtime corresponds to the average time to process one batch, where the batch size is 32.

Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Init. (s)	Refine. (s)	Batched inf. (s)	Per-inst. (s)
OSQP	0.000 0	0.000 0	0.0	0.0	–	–	–	2.217 4
HPR-QP	0.000 0	0.000 0	2.3×10^{-6}	1.8×10^{-9}	–	–	–	0.271 0
HPR-QP fast	0.000 8	0.009 2	2.4×10^{-3}	1.1×10^{-3}	–	–	–	0.219 1
CuClarabel	0.000 0	0.000 0	5.6×10^{-4}	3.0×10^{-7}	–	–	–	0.382 2
CuClarabel fast	0.000 0	0.000 0	3.8×10^{-2}	4.7×10^{-3}	–	–	–	0.205 8
qpth	0.000 0	0.000 0	2.7×10^{-8}	9.7×10^{-11}	–	–	–	3.668 1
DC3	0.000 0	0.000 0	9.5×10^{-1}	1.1	–	–	0.003 6	–
Descent-Net	0.000 0	0.000 0	2.9×10^{-2}	5.8×10^{-3}	0.003 6	0.004 2	0.007 8	–

Table 3 Results on the convex QP task evaluated on the test set with 833 samples, with problem dimension $n = 5\,000$, $n_{eq} = 500$, and $n_{ineq} = 500$. The reported runtime corresponds to the average time to process one batch, where the batch size is 8.

Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Init. (s)	Refine. (s)	Batched inf. (s)	Per-inst. (s)
OSQP	0.000 0	0.000 0	0.0	0.0	–	–	–	13.460 9
HPR-QP	0.000 0	0.000 0	5.8×10^{-8}	1.1×10^{-11}	–	–	–	0.916 0
HPR-QP fast	0.154 5	0.381 1	8.7×10^{-3}	1.1×10^{-3}	–	–	–	0.549 3
CuClarabel	0.000 0	0.000 0	6.1×10^{-6}	2.1×10^{-9}	–	–	–	0.468 1
CuClarabel fast	0.000 0	0.000 0	1.3×10^{-2}	1.4×10^{-3}	–	–	–	0.307 2
DC3	0.000 0	0.000 0	9.8×10^{-1}	9.2×10^{-1}	–	–	0.003 8	–
Descent-Net	0.000 0	0.000 0	1.7×10^{-2}	3.2×10^{-3}	0.003 8	0.011 5	0.015 3	–

In addition, to further illustrate the effectiveness of Descent-Module, we examine the error between the descent direction d and the optimal solution of its corresponding subproblem (Pen). The experimental results are provided

in appendix D. We also compare Descent-Module with the original projected subgradient method, and the results are reported in appendix F.

5.3 Simple non-convex optimization

We now examine a simple non-convex adaptation of the quadratic program

$$\min_{y\in\mathbb{R}^n} \frac{1}{2}y^\top Qy + p^\top \sin(y), \quad \text{s.t. } Ay = x,\ Gy \leqslant h, \tag{21}$$

where $\sin(y)$ represents the component-wise application of the sine function to the vector y. Compared to problem (20), the only difference is that y in the objective function is replaced with $\sin(y)$, which makes the problem non-convex.

The experimental results are presented in Table 4. The initial solutions use those from DC3, and the final solutions produced by Descent-Net achieve a relative objective error of 2.3×10^{-4}. Moreover, Descent-Net processes one batch in 0.014 5 seconds, while IPOPT requires 0.336 4 seconds to solve a single instance, indicating a substantial runtime advantage under the reported evaluation protocol.

Table 4 Results on the simple non-convex task evaluated on the test set with 833 samples, with problem dimension $n = 100$, $n_{eq} = 50$, and $n_{ineq} = 50$. The reported runtime corresponds to the average time to process one batch, where the batch size is 833.

Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Init. (s)	Refine. (s)	Batched inf. (s)	Per-inst. (s)
IPOPT	0.000 0	0.000 0	0.0	0.0	–	–	–	0.336 4
Projection method	0.000 0	0.000 0	5.4×10^{-2}	1.8×10^{-3}	–	–	0.247 2	–
CBWF	0.000 0	0.000 0	2.6×10^{-1}	5.5×10^{-2}	0.002 5	0.033 9	0.036 4	–
DC3	0.000 0	0.000 0	4.4×10^{-1}	3.1×10^{-1}	–	–	0.002 5	–
Descent-Net	0.000 0	0.000 0	1.5×10^{-2}	2.3×10^{-4}	0.002 5	0.012 0	0.014 5	–

5.4 Portfolio optimization

A widely applicable instance of quadratic programming in real-world settings is the mean-variance portfolio optimization problem. The objective is to minimize portfolio risk while satisfying practical portfolio allocation constraints:

$$\min_{\mathbf{w}\in\mathbb{R}^n} \mathbf{w}^\top \Sigma \mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^\top \mathbf{1} = 1,\ \mathbf{w}^\top \mu \geqslant r_{\min},\ \mathbf{w} \geqslant 0, \tag{22}$$

where $\mathbf{w}$ denotes asset weights, $\Sigma \in \mathbb{R}^{n\times n}$ is the covariance matrix, $\mu \in \mathbb{R}^n$ is the expected return vector, and $r_{\min}$ is the minimum return requirement.

We conduct portfolio optimization experiments with $n = 100$, $n = 800$, and $n = 4\,000$ assets to evaluate the practical effectiveness and scalability of our method. For each problem size, we generate 10 000 synthetic benchmark instances and use the equal-weighted portfolio $w_i = 1/n$ as the initial solution. It is worth noting that this choice of initialization does not guarantee feasibility

in general; however, under our experimental setup, due to the specific data generation procedure described in Appendix A, the feasibility rate is 100%. Descent-Net is compared against the solver `OSQP` and DC3, with the combined numerical results summarized in Table 5.

Table 5 Results on the portfolio optimization task evaluated on the test set with 1 000 samples. The reported runtime corresponds to the average runtime for a batch of instances, with batch sizes of 512, 100, and 10 for problem sizes $n = 100$, $n = 800$, and $n = 4\,000$, respectively.

Number of assets $n = 100$						
Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Batched inf. (s)	Per-inst. (s)
OSQP	0.000 0	0.000 0	0.0	0.0	–	0.001 5
HPR-QP	0.000 0	0.000 0	6.9×10^{-6}	1.1×10^{-7}	–	0.064 5
CuClarabel	0.000 0	0.000 0	1.1×10^{-5}	4.0×10^{-9}	–	0.040 7
DC3	0.000 0	0.000 0	2.8	5.4×10^{1}	0.012 5	–
Descent-Net	0.000 0	0.000 0	1.4×10^{-4}	4.9×10^{-6}	0.001 9	–
Number of assets $n = 800$						
Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Batched inf. (s)	Per-inst. (s)
OSQP	0.000 0	0.000 0	0.0	0.0	–	0.020 7
HPR-QP	0.000 0	0.000 0	5.0×10^{-6}	6.2×10^{-8}	–	0.168 3
CuClarabel	0.000 0	0.000 0	5.4×10^{-5}	4.0×10^{-8}	–	0.116 8
Descent-Net	0.000 0	0.000 0	9.3×10^{-4}	7.3×10^{-6}	0.001 9	–
Number of assets $n = 4\,000$						
Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Batched inf. (s)	Per-inst. (s)
OSQP	0.000 0	0.000 0	0.0	0.0	–	0.602 4
HPR-QP	0.000 0	0.000 0	1.6×10^{-5}	2.2×10^{-6}	–	0.367 6
CuClarabel	0.000 0	0.000 0	4.2×10^{-5}	2.2×10^{-6}	–	0.523 2
Descent-Net	0.000 0	0.000 0	1.6×10^{-4}	1.2×10^{-6}	0.004 4	–

We find that DC3 fails to produce feasible solutions for $n = 800$ and $n = 4\,000$ because its training diverges. This is likely due to DC3's reliance on gradient steps, which are used to enforce inequality constraints, but whose step sizes and momentum decay parameters are difficult to tune for large-scale settings. We performed a grid search over approximately 100 configurations of step sizes and momentum decay parameters. However, even for the best-performing configuration, DC3 still yields large constraint violations, with the average inequality violation remaining at approximately 1.16×10^2. In contrast, Descent-Net remains accurate and highly efficient across all problem sizes.

As shown, Descent-Net achieves lower runtimes while maintaining objective errors on the order of 10^{-6}. The runtime comparison indicates a clear advantage of Descent-Net under the reported batched-inference setting, especially for large-scale instances..

5.5 AC-OPF

The objective of the AC optimal power flow (AC-OPF) problem is to determine the optimal power generation that balances supply and demand while satisfying both physical laws and operational constraints of the network. Unlike the linearly constrained problems considered in our theoretical analysis, AC-OPF involves nonlinear constraints, and therefore the convergence results in Section 4 do not directly apply to this setting. Nevertheless, as shown in the following experiments, Descent-Net still achieves strong empirical performance, demonstrating its potential to generalize beyond the current theoretical scope. A compact formulation of the AC-OPF problem is as follows:

$$\begin{aligned} \min_{p_g\in\mathbb{R}^n,\, q_g\in\mathbb{R}^n,\, v\in\mathbb{C}^n} \quad & p_g^\top Q p_g + b^\top p_g \\ \text{s.t.} \quad & p_g^{\min} \leqslant p_g \leqslant p_g^{\max}, \quad q_g^{\min} \leqslant q_g \leqslant q_g^{\max}, \quad v_m^{\min} \leqslant |v| \leqslant v_m^{\max}, \\ & v_a^{\min} \leqslant \angle v_i - \angle v_j \leqslant v_a^{\max}, \quad |v_i(\overline{v_i} - \overline{v_j})\overline{w}_{ij}| \leqslant S_{ij}^{\max}, \\ & (p_g - p_d) + (q_g - q_d)i = \operatorname{diag}(v)\overline{W}\,\overline{v}. \end{aligned} \tag{23}$$

Here, $p_d, q_d \in \mathbb{R}^n$ denote the active and reactive power demands, and $p_g, q_g \in \mathbb{R}^n$ are the corresponding power generations. The complex bus voltage is represented by $v \in \mathbb{C}^n$. The nodal admittance matrix $W \in \mathbb{C}^{n\times n}$ encodes the network topology.

Since the equality constraints in this problem are nonlinear, a first-order approximation is not very accurate. As a result, even if the descent direction d is orthogonal to the gradients of all equality constraints, the updated point may still fail to satisfy them. To address this issue, we adopt an equation completion approach, and the details are provided in the appendix G.

We conduct experiments on two AC-OPF problem instances of different scales. Besides D-Proj (i.e., DC3), we use H-Proj [28] as another initialization strategy, and denote the corresponding solutions by y^D and y^H. D-Proj originally reduces violations of inequality constraints by performing a gradient descent step on the ℓ_2 norm of constraint violations. In our experiments, we found that this gradient step is time-consuming and, in practice, often unnecessary. Therefore, we introduce an improved variant of D-Proj by removing the gradient-descent step. This modification significantly reduces the computational time while maintaining comparable satisfaction of the inequality constraints. The optimized initialization obtained using this approach is denoted by y^{D^*}.

The results in Table 6 indicate that Descent-Net produces solutions with relative objective errors on the order of 10^{-4} across all cases, outperforming all neural network–based solvers in accuracy. Moreover, compared with the standard solver `PYPOWER`, it achieves substantially faster inference on both test systems. On the 30-bus system, Descent-Net processes one batch in 0.043 4 seconds, whereas PYPOWER requires 0.289 0 seconds for a single instance. Under

this batch-versus-single-instance comparison, this corresponds to approximately a 6.7-fold runtime advantage. On the 118-bus system, Descent-Net requires 0.162 2 seconds to process a batch of instances, while PYPOWER requires 0.642 3 seconds for a single instance, corresponding to approximately 4 times speedup. The relative error of the solution obtained by Descent-Net decreases only marginally compared to the initial point, which may be due to the non-convex nature of the AC-OPF problem.

Table 6 Results on the AC-OPF task evaluated on the test set with 1 024 samples. The runtime corresponds to the time required to process one batch with a batch size of 1 024.

30-bus system: $n_{\text{eq}} = 60, n_{\text{ineq}} = 84$								
Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Init. (s)	Refine. (s)	Batched inf. (s)	Per-inst. (s)
PYPOWER	0.000 0	0.000 0	0.0	0.0	–	–	–	0.289 0
Projection method	0.000 0	0.000 0	5.6×10^{-3}	1.7×10^{-2}	–	–	0.039 3	–
Warm start	0.000 0	0.000 0	5.5×10^{-3}	1.7×10^{-2}	–	–	0.039 7	–
DeepOPF-V	0.000 3	0.003 8	3.0×10^{-3}	1.4×10^{-2}	–	–	0.009 1	–
DeepOPF-NGT	0.000 0	0.010 7	3.1×10^{-2}	1.9×10^{-1}	–	–	4.290 6	–
D-Proj	0.000 0	0.000 0	5.9×10^{-3}	1.9×10^{-2}	–	–	0.244 2	–
H-Proj	0.000 0	0.000 0	5.8×10^{-3}	1.7×10^{-2}	–	–	0.286 5	–
Descent-Net ($y_0 = y^D$)	0.000 0	0.000 0	4.2×10^{-3}	3.6×10^{-4}	0.244 2	0.017 7	0.261 9	–
Descent-Net ($y_0 = y^H$)	0.000 0	0.000 0	3.5×10^{-3}	3.3×10^{-4}	0.286 5	0.017 4	0.303 9	–
Descent-Net ($y_0 = y^{D^*}$)	0.000 0	0.000 0	3.6×10^{-3}	2.8×10^{-4}	0.014 0	0.029 4	0.043 4	–
118-bus system: $n_{\text{eq}} = 236, n_{\text{ineq}} = 452$								
Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Init. (s)	Refine. (s)	Batched inf. (s)	Per-inst. (s)
PYPOWER	0.000 0	0.000 0	0.0	0.0	–	–	–	0.642 3
Projection method	0.000 0	0.000 0	1.5×10^{-2}	2.4×10^{-3}	–	–	0.304 0	–
Warm start	0.000 0	0.000 0	9.3×10^{-3}	1.8×10^{-3}	–	–	0.313 7	–
DeepOPF-V	0.000 6	0.008 7	6.1×10^{-3}	4.4×10^{-3}	–	–	4.275 0	–
DeepOPF-NGT	0.000 0	0.009 2	2.3×10^{-1}	1.7×10^{-1}	–	–	4.701 3	–
D-Proj	0.000 0	0.000 0	1.3×10^{-2}	2.4×10^{-3}	–	–	0.754 2	–
H-Proj	0.000 0	0.000 0	1.4×10^{-2}	3.1×10^{-3}	–	–	0.668 2	–
Descent-Net ($y_0 = y^D$)	0.000 0	0.000 0	1.2×10^{-2}	2.5×10^{-4}	0.754 2	0.193 8	0.948 0	–
Descent-Net ($y_0 = y^H$)	0.000 0	0.000 0	1.4×10^{-2}	7.2×10^{-4}	0.668 2	0.195 5	0.863 7	–
Descent-Net ($y_0 = y^{D^*}$)	0.000 0	0.000 0	2.2×10^{-3}	3.0×10^{-4}	0.015 5	0.146 7	0.162 2	–

6 Conclusion

In this work, we presented Descent-Net, a neural architecture that incorporates first-order optimality structure for solving linearly constrained optimization problems, with the goal of improving the effectiveness of existing L2O methods. The model generalizes well across different types of instances, including quadratic programs and simple non-convex variants of QPs. We also empirically extend the framework to problems with nonlinear constraints such as AC-OPF, where feasibility is handled through a heuristic equation completion strategy. In our experiments, Descent-Net produces more accurate solutions than prior L2O approaches while also offering faster inference compared to classical solvers. Moreover, we demonstrate its scalability on large-scale QP problems and portfolio optimization tasks with thousands of assets, highlighting its potential for real-world applications. Future work includes extending the framework to nonlinear constraints and using Descent-Net solutions as warm starts for standard solvers to get high-accurate solutions.

Appendix A Experiment setting

In all experiments, we set the penalty weights in the descent subproblem by the heuristic rule (12), where $c_0 = c_1 = 1$ in all experiments and the other parameters are given in Table A1. This adaptive choice assigns larger weights to nearly active constraints, since $g_{x,j}(y)$ close to zero leads to a smaller denominator, thereby encouraging the learned direction to satisfy the corresponding linearized inequality more accurately. The constant $\varepsilon > 0$ is used for numerical stability.

For the convex QPs and the simple non-convex problems, the parameters are generated as follows. The matrix Q is constructed as $Q = R^\top R$, where $R \in \mathbb{R}^{n\times n}$ has i.i.d. standard normal entries, ensuring that Q is symmetric and positive semidefinite. The constraint matrices A and G are also sampled with i.i.d. standard normal entries. For each instance, the components of x are drawn i.i.d. from the uniform distribution on $[-1, 1]$. To ensure that the generated problem has a feasible solution, we set $h_i = \sum_j |(GA^\dagger)_{ij}|$, where $A^\dagger$ denotes the Moore–Penrose pseudoinverse of A. For the AC-OPF experiments, we use the datasets provided in [28].

For the portfolio optimization experiments, to model slowly evolving asset co-movements, the covariance matrix is fixed across instances and constructed as $\Sigma = R^\top R$, where $R \in \mathbb{R}^{n\times n}$ has i.i.d. standard normal entries. For each instance, the expected return vector μ is sampled independently from a uniform distribution over $[0, 1]$, and the return threshold $r_{\min}$ is drawn independently: for training, $r_{\min} \sim \text{Uniform}[0.05, 0.4]$, while for testing, $r_{\min}$ is taken as a linearly spaced sequence over the same interval.

We summarize the hyperparameters used in our experiments in Table A1. Below we briefly describe several important parameters:

- S: the number of update steps performed in our Descent Net.
- K: the number of layers within each Descent module, controlling the expressive power of the network.
- λ_h: the penalty factor for equality constraint violations.
- λ_g: the penalty factor for inequality constraint violations.
- q: the hidden dimension of operator T, which specifies the capacity of feature transformation inside each descent step.
- M, ε: parameters in c_j, which is defined in (12).

For the experiments other than portfolio optimization, we train the Descent module using the Adam optimizer with an initial learning rate of 0.01. The learning rate is reduced by a factor of 0.1 at epochs 50, 100, and 150. The step-size parameter β is updated separately using the SGD optimizer with a fixed learning rate of 0.01. In the AC-OPF experiments, we additionally clip the gradient norm at a threshold of 1 to stabilize training, following [43].

In contrast, the portfolio optimization experiments adopt a different training configuration. Both the Descent module and the step size β are optimized using Adam. The initial learning rates are set to 1×10^{-3} for the Descent module, and to 0.1, 0.1, and 0.01 for β in the $n = 100$, $n = 800$, and $n = 4\,000$

Table A1 Hyperparameters used in different experiments

Problem	**Epochs**	S	K	λ_h	λ_g	q	M	ε
QP (n=100)	150	8	3	5	5	300	1	0.000 5
QP (n=1 000)	300	5	1	5	5	3 000	1	0.000 5
QP (n=5 000)	300	8	1	5	5	15 000	1	0.000 5
Non-convex	150	10	3	5	5	300	1	0.000 5
AC-OPF (node=30)	300	3	3	5	5	120	1	0.000 1
AC-OPF (node=118)	300	3	3	5	5	1 080	1	0.000 1
Portfolio (n=100)	300	3	1	5	5	800	1	0.000 1
Portfolio (n=800)	300	2	1	5	5	1 200	1	0.000 1
Portfolio (n=4 000)	300	2	1	5	5	6 000	1	0.000 1

settings, respectively. All learning rates decay by a factor of 0.1 at epochs 100, 150, and 200 over a total of 300 training epochs.

All experiments were conducted on a server equipped with two AMD EPYC 9754 CPUs (128 cores each, 3.1 GHz) and an NVIDIA H100 GPU.

Appendix B Effect of Layer number K and Descent steps S

We conducted experiments on the convex quadratic program (20) to evaluate the performance of Descent-Modules with different numbers of layers K. The results are shown in Table B2. It can be seen that increasing K leads to a slight improvement in performance, but the gains are not significant. Considering computational efficiency, we ultimately choose $K = 3$ as the number of layers.

Table B2 Performance of Descent-Module with varying K

Layer	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.
$K = 1$	0.000 0	0.000 0	9.8×10^{-2}	2.2×10^{-2}
$K = 2$	0.000 0	0.000 0	9.3×10^{-2}	1.8×10^{-2}
$K = 3$	0.000 0	0.000 0	9.2×10^{-2}	1.7×10^{-2}
$K = 4$	0.000 0	0.000 0	8.9×10^{-2}	1.7×10^{-2}

With K fixed at 3, we further examined the effect of different Descent steps S, as summarized in Table B3. When the number of update steps is 1, Descent-Net already achieves a solution with a relative error on the order of 10^{-2}. Increasing the steps to 4 reduces the error to the 10^{-3} level, and further increasing to 8 reduces it to the 10^{-4} level.

Appendix C Step Size Selection Strategies

We compare the effectiveness of three different step size selection strategies:

- A fixed step size $\alpha = 1/M$;

Table B3 Performance of Descent-Net with varying S

Descent Step	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.
$S = 1$	0.000 0	0.000 0	8.9×10^{-2}	1.7×10^{-2}
$S = 2$	0.000 0	0.000 0	6.0×10^{-2}	1.1×10^{-2}
$S = 4$	0.000 0	0.000 0	3.5×10^{-2}	3.4×10^{-3}
$S = 6$	0.000 0	0.000 0	2.5×10^{-2}	1.8×10^{-3}
$S = 8$	0.000 0	0.000 0	1.2×10^{-2}	3.1×10^{-4}

- The maximum feasible step size $\alpha_{\max}$ that ensures feasibility;
- A learnable scale factor $\sigma(\beta)$ applied to $\alpha_{\max}$.

We perform comparative experiments on the convex quadratic program (20), evaluating three methods based on the feasibility and optimality of their solutions after a fixed number of update steps $S = 8$. The corresponding results are presented in Table C4. As shown, both the fixed step size $1/M$ and the maximum feasible step size $\alpha_{\max}$ perform worse than our final choice $\alpha = \sigma(\beta)\alpha_{\max}$. The limitation of $1/M$ lies in its lack of flexibility, as a fixed step size cannot adapt to the varying landscape of the problem. And $\sigma(\beta)\alpha_{\max}$ outperforms $\alpha_{\max}$ because the learnable parameter β captures useful information that enables a more appropriate scaling of the maximum step size.

Table C4 Comparison of different step size selection strategies

Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.
$1/M$	0.000 0	0.000 0	9.0×10^{-2}	1.2×10^{-2}
$\alpha_{\max}$	0.000 0	0.000 0	1.1×10^{-1}	3.3×10^{-2}
$\sigma(\beta)\alpha_{\max}$	0.000 0	0.000 0	1.2×10^{-2}	3.1×10^{-4}

Appendix D Subproblem

In our method, each descent direction d_s is obtained by solving a subproblem (Pen). To assess the ability of the Descent-Net to solve this subproblem, we measure the relative error of the subproblem's objective value between each layer's output d_k and the corresponding optimal solution.

We conduct experiments on the convex QP task. For simplicity, we set $S = 1$, performing only a single update, and fix the number of Descent-Net layers to $K = 3$. We then evaluate the trained network, with the results reported in Table D5. As shown, the objective value of the subproblem (Descent value) decreases progressively across layers, and by the final layer (layer 3), the relative error in the objective value has already been reduced to 0.001, demonstrating the efficiency of the Descent-Net in solving the subproblem.

Table D5 Effectiveness of Descent-Net in solving subproblem

Layer	Descent Value	Relative Error
0	1 740.481 7	2.646 3
1	505.259 1	0.058 5
2	478.272 1	0.002 0
3	477.789 3	0.001 0

Appendix E Learnable γ in Descent-Net

We recorded the values of the learnable parameter γ in each layer of the S Descent Modules of the trained Descent-Net. For both QP and Nonconvex problems, γ is initialized to 0.1, while for the AC-OPF problem it is initialized to 1. The results are presented in Table E6, Table E7 and Table E8. These results indicate that the network is able to adjust γ dynamically across layers. In many cases, the values of γ tend to decrease with the layer depth, which is consistent with the requirement of diminishing step sizes for convergence in subgradient methods.

Table E6 Values of γ in Descent-Net across some steps for Simple QP problems

	Step 1	Step 3	Step 5	Step 7
γ_1	1.15×10^{-2}	6.37×10^{-3}	1.03×10^{-2}	1.75×10^{-2}
γ_2	1.41×10^{-2}	5.39×10^{-2}	5.71×10^{-3}	4.49×10^{-3}
γ_3	9.85×10^{-2}	1.00×10^{-1}	9.93×10^{-4}	2.23×10^{-3}

Table E7 Values of γ in Descent-Net across some steps for Nonconvex problems

	Step 1	Step 3	Step 5	Step 7	Step 9
γ_1	4.79×10^{-2}	1.02×10^{-2}	3.40×10^{-3}	8.22×10^{-3}	1.95×10^{-2}
γ_2	4.92×10^{-2}	9.53×10^{-2}	4.39×10^{-4}	3.08×10^{-3}	4.78×10^{-3}
γ_3	1.19×10^{-1}	5.53×10^{-4}	3.53×10^{-2}	1.94×10^{-3}	3.25×10^{-3}

Appendix F Comparison with PGM (Projected Subgradient Method)

We compare Descent-Net with the original PGM. Specifically, we remove the operator T^k in Descent-Net so that each layer reduces to (14). We still treat the step size γ_k as a learnable parameter and train this degenerated network in the same manner as Descent-Net.

We evaluate the performance under different numbers of iterations K, with the results reported in Table F9. We observe that PGM is inefficient, as the

Table E8 Values of γ in Descent-Net across steps for AC-OPF problems

node = 30, H-Proj				**node = 30, D-Proj**			
	Step 1	Step 2	Step 3		Step 1	Step 2	Step 3
γ_1	1.00	1.00	1.00	γ_1	1.00	1.00	1.00
γ_2	0.99	1.01	0.99	γ_2	0.99	0.99	1.00
γ_3	1.10	0.01	0.84	γ_3	1.42	0.20	1.01
node = 118, H-Proj				**node = 118, D-Proj**			
	Step 1	Step 2	Step 3		Step 1	Step 2	Step 3
γ_1	1.00	1.00	1.00	γ_1	1.00	1.00	1.00
γ_2	1.00	1.00	1.00	γ_2	1.00	1.00	1.00
γ_3	1.07	1.06	1.10	γ_3	1.29	1.03	0.99

relative error in the objective value compared to the initial solution decreases very little with increasing iterations. This is likely due to the difficulty of selecting an appropriate step size for PGM. In contrast, the Descent-Net achieves strong solution quality with only three layers, which also leads to a significant advantage in computational efficiency.

Table F9 Comparison of Descent-Net and PGM on the convex QP task.

Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.	Time (s)
PGM ($K = 10$)	0.000 0	0.000 0	1.9×10^{-1}	1.1×10^{-1}	0.027 0
PGM ($K = 20$)	0.000 0	0.000 0	1.9×10^{-1}	1.1×10^{-1}	0.050 1
PGM ($K = 50$)	0.000 0	0.000 0	1.9×10^{-1}	1.1×10^{-1}	0.111 9
Descent-Net ($K = 3$)	0.000 0	0.000 0	1.2×10^{-2}	3.1×10^{-4}	0.013 2

Appendix G Descent Updates in the AC-OPF Problem

In the AC-OPF problem, given $(n-m)$ entries of a feasible point $y \in \mathbb{R}^n$, the remaining m entries are, in general, determined by the m equality constraints $h_x(y) = 0$.

Following the method in [18, 28, 38], we assume the existence of a function $\varphi_x : \mathbb{R}^{n-m} \to \mathbb{R}^m$ such that $h_x([z, \varphi_x(z)]) = 0$. This allows us to eliminate the equality constraints and reformulate the problem in terms of the partial variable z. We can then perform descent direction updates on z, where the optimization problem involves only the inequality constraints:

$$\min_{z \in \mathbb{R}^{n-m}} \tilde{f}_x(z), \quad \text{s.t.} \quad \tilde{g}_x(z) \leqslant 0, \tag{G1}$$

where $\tilde{f}_x(z) = f_x\left([z^\top, \varphi_x(z)^\top]^\top\right)$ and $\tilde{g}_x(z) = g_x\left([z^\top, \varphi_x(z)^\top]^\top\right)$.

Using the chain rule, we can compute the derivative of φ_x with respect to z, even without an explicit expression of φ_x:

$$\begin{aligned} 0 = \frac{\mathrm{d}}{\mathrm{d}z} h_x\big(\varphi_x(z)\big) &= \frac{\partial h_x}{\partial z} + \frac{\partial h_x}{\partial \varphi_x(z)} \frac{\partial \varphi_x(z)}{\partial z} \\ &= J^h_{:,0:m} + J^h_{:,m:n} \frac{\partial \varphi_x(z)}{\partial z}, \\ \Rightarrow \quad \frac{\partial \varphi_x(z)}{\partial z} &= -\big(J^h_{:,m:n}\big)^{-1} J^h_{:,0:m}. \end{aligned}$$

Here, $J^h \in \mathbb{R}^{m \times n}$ denotes the Jacobian matrix of the equality constraints $h_x(y)$ with respect to y. The notation $J^h_{:,0:m}$ and $J^h_{:,m:n}$ represent the submatrices corresponding to the partial derivatives with respect to z and $\varphi_x(z)$, respectively.

From this result, we can further obtain the gradients of the objective and inequality constraints with respect to z. These gradient information is then passed to the Descent-Net, which outputs the descent direction d_z for the partial variable z.

In order to obtain the complete descent direction $d = [d_z, d_\varphi]$ for y, we also need the expression of d_φ. To ensure that the equality constraints remain satisfied, we require the following first-order condition:

$$\begin{aligned} h(z + \alpha d_z, \varphi(z) + \alpha d_\varphi) \approx & h\big(z, \varphi(z)\big) + \alpha\, J^h \begin{bmatrix} d_z \\ d_\varphi \end{bmatrix} \\ = & h\big(z, \varphi(z)\big) + \alpha\big(J^h_{:,0:m} d_z + J^h_{:,m:n} d_\varphi\big) = 0, \end{aligned}$$

where $\alpha > 0$ is the step size. Hence, we obtain

$$d_\varphi = -\big(J^h_{:,m:n}\big)^{-1} J^h_{:,0:m}\, d_z \;-\; \big(J^h_{:,m:n}\big)^{-1} \frac{h\big(z, \varphi(z)\big)}{\alpha}.$$

Appendix H Effect of Initialization on Performance

To further investigate the impact of initialization quality, we conduct an additional test on convex QP, nonconvex QP, and AC-OPF problems. For the convex and nonconvex QP cases, we use a mildly infeasible initial solution generated by a DC3 model at an intermediate stage of training, while for the AC-OPF case, the initialization is obtained from a DeepOPF model. Such an initial point does not strictly satisfy all constraints and is therefore outside the setting covered by our theoretical analysis, which assumes a feasible starting point. The results are summarized in Table H10.

From Table H10, we observe different behaviors across problem classes. For the convex QP and the nonconvex QP with linear constraints, Descent-Net substantially reduces the inequality violation and improves both the solution

Table H10 Performance of Descent-Net under mildly infeasible initializations.

Convex QP: $n = 100, n_{\mathbf{eq}} = n_{\mathbf{ineq}} = 50$				
Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.
Init	0.013 4	0.000 0	7.9×10^{-1}	9.1×10^{-1}
Descent-Net	0.000 1	0.000 0	9.6×10^{-2}	5.3×10^{-2}
Nonconvex: $n = 100, n_{\mathrm{eq}} = n_{\mathrm{ineq}} = 50$				
Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.
Init	0.010 4	0.000 0	9.6×10^{-1}	1.1
Descent-Net	0.000 0	0.000 0	6.9×10^{-2}	2.3×10^{-2}
AC-OPF: $n = 300, n_{\mathrm{eq}} = 600, n_{\mathrm{ineq}} = 876$				
Method	ineq. vio.	eq. vio.	sol. rel. err.	obj. rel. err.
Init	0.000 2	0.005 2	2.7×10^{-3}	3.4×10^{-3}
Descent-Net	0.000 2	0.005 3	9.9×10^{-3}	1.7×10^{-3}

error and the objective error compared with the mildly infeasible initial point. This suggests that, in the linearly constrained setting, the learned refinement step has some robustness to small infeasibility in the initialization, although the final accuracy is worse than that obtained from strictly feasible initializations.

For the AC-OPF problem, the behavior is more delicate. Descent-Net reduces the objective error, but it does not improve the equality violation and the solution error becomes larger. This indicates that the nonlinear equality constraints make the refinement process more sensitive to initialization. This observation is consistent with the scope of our theory, which covers linearly constrained problems, while the AC-OPF experiment is presented only as an empirical extension beyond the current theoretical guarantees.

Author Contributions

Zi-sheng Zhou was responsible for the algorithm design, theoretical analysis, numerical experiments. Deng-yu Zheng contributed to part of the theoretical analysis and conducted supporting numerical experiments. Zi-rui Chen assisted with the implementation and execution of numerical experiments. Shi-xiang Chen served as the corresponding author and was responsible for the overall research conception, methodological guidance, critical revision and the primary drafting of the manuscript.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., Kolter, J.Z.: Differentiable convex optimization layers. Advances in neural information processing systems **32** (2019)

[2] Amos, B., Kolter, J.Z.: Optnet: Differentiable optimization as a layer in neural networks. In: International Conference on Machine Learning, pp. 136–145 (2017). PMLR

[3] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. Advances in neural information processing systems **29** (2016)

[4] Bai, S., Kolter, J.Z., Koltun, V.: Deep equilibrium models. Advances in Neural Information Processing Systems **32** (2019)

[5] Baker, K.: Learning warm-start points for ac optimal power flow. In: 2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP), pp. 1–6 (2019). IEEE

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

[7] Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., Bach, F.: Learning with differentiable pertubed optimizers. Advances in neural information processing systems **33**, 9508–9519 (2020)

[8] Bolte, J., Le, T., Pauwels, E., Silveti-Falls, T.: Nonsmooth implicit differentiation for machine-learning and optimization. Advances in neural information processing systems **34**, 13537–13549 (2021)

[9] Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., *et al.*: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine learning **3**(1), 1–122 (2011)

[10] Chen, B., Donti, P.L., Baker, K., Kolter, J.Z., Bergés, M.: Enforcing policy feasibility constraints through differentiable projection for energy optimization. In: Proceedings of the Twelfth ACM International Conference on Future Energy Systems, pp. 199–210 (2021)

[11] Chen, K., Sun, D., Yuan, Y., Zhang, G., Zhao, X.: Hpr-qp: A dual halpern peaceman-rachford method for solving large-scale convex composite quadratic programming. arXiv preprint arXiv:2507.02470 (2025)

[12] Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. Advances in neural information processing systems **31** (2018)

[13] Chen, T., Chen, X., Chen, W., Heaton, H., Liu, J., Wang, Z., Yin, W.: Learning to optimize: A primer and a benchmark. Journal of Machine Learning Research **23**(189), 1–59 (2022)

[14] Chen, X., Liu, J., Wang, Z., Yin, W.: Theoretical linear convergence of unfolded ista and its practical weights and thresholds. Advances in Neural Information Processing Systems **31** (2018)

[15] Chen, Y., Tse, D., Nobel, P., Goulart, P., Boyd, S.: CuClarabel: GPU Acceleration for a Conic Optimization Solver (2024). https://arxiv.org/abs/2412.19027

[16] Diehl, F.: Warm-starting ac optimal power flow with graph neural networks. In: 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), pp. 1–6 (2019)

[17] Donti, P., Amos, B., Kolter, J.Z.: Task-based end-to-end model learning in stochastic optimization. Advances in neural information processing systems **30** (2017)

[18] Donti, P.L., Rolnick, D., Kolter, J.Z.: Dc3: A learning method for optimization with hard constraints. In: International Conference on Learning Representations (2021)

[19] Fabozzi, F.J., Markowitz, H.M., Gupta, F.: Portfolio selection. Handbook of finance **2**, 3–13 (2008)

[20] Faigle, U., Kern, W., Still, G.: Algorithmic Principles of Mathematical Programming vol. 24. Springer, Dordrecht (2013)

[21] Gao, Z., Wu, Y., Fan, X., Harandi, M., Jia, Y.: Learning to optimize on riemannian manifolds. IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(5), 5935–5952 (2022)

[22] Geng, Z., Johnson, D., Fedkiw, R.: Coercing machine learning to output physically accurate results. Journal of Computational Physics **406**, 109099 (2020)

[23] Graves, A.: Generating Sequences With Recurrent Neural Networks (2014)

[24] Hoffman, A.J.: On approximate solutions of systems of linear inequalities. Journal of Research of the National Bureau of Standards **49**(4), 263–265

(1952)

[25] Huang, W., Chen, M., Low, S.H.: Unsupervised learning for solving ac optimal power flows: Design, analysis, and experiment. IEEE Transactions on Power Systems **39**(6), 7102–7114 (2024)

[26] Huang, W., Pan, X., Chen, M., Low, S.H.: Deepopf-v: Solving ac-opf problems efficiently. IEEE Transactions on Power Systems **37**(1), 800–803 (2021)

[27] Li, B., Yang, L., Chen, Y., Wang, S., Mao, H., Chen, Q., Ma, Y., Wang, A., Ding, T., Tang, J., *et al.*: Pdhg-unrolled learning-to-optimize method for large-scale linear programming. In: Proceedings of the 41st International Conference on Machine Learning, pp. 29164–29180 (2024)

[28] Liang, E., Chen, M., Low, S.H.: Homeomorphic projection to ensure neural-network solution feasibility for constrained optimization. Journal of Machine Learning Research **25**(329), 1–55 (2024)

[29] Luenberger, D.G., Ye, Y., *et al.*: Linear and Nonlinear Programming vol. 2. Springer, New York, NY (1984)

[30] Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, New York, NY (1999)

[31] Pironneau, O., Polak, E.: Rate of convergence of a class of methods of feasible directions. SIAM Journal on Numerical Analysis **10**(1), 161–174 (1973)

[32] Pogančić, M.V., Paulus, A., Musil, V., Martius, G., Rolinek, M.: Differentiation of blackbox combinatorial solvers. In: International Conference on Learning Representations (2019)

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

[34] Tan, Y., Terekhov, D., Delong, A.: Learning linear programs from optimal decisions. Advances in Neural Information Processing Systems **33**, 19738–19749 (2020)

[35] Topkis, D.M., Veinott, A.F. Jr: On the convergence of some feasible direction algorithms for nonlinear programming. SIAM Journal on Control **5**(2), 268–279 (1967)

[36] Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming.

Mathematical programming **106**, 25–57 (2006)

[37] Wang, R., Zhang, Y., Guo, Z., Chen, T., Yang, X., Yan, J.: Linsatnet: the positive linear satisfiability neural networks. In: International Conference on Machine Learning, pp. 36605–36625 (2023). PMLR

[38] Wu, S., Chen, S., Shen, L., Zhang, L., Tao, D.: Constraint boundary wandering framework: Enhancing constrained optimization with deep neural networks. IEEE Transactions on Pattern Analysis & Machine Intelligence **47**(08), 6369–6381 (2025)

[39] Wu, Z., Xiao, M., Fang, C., Lin, Z.: Designing universally-approximating deep neural networks: A first-order optimization approach. IEEE Transactions on Pattern Analysis and Machine Intelligence (2024)

[40] Xie, X., Wu, J., Liu, G., Zhong, Z., Lin, Z.: Differentiable linearized admm. In: International Conference on Machine Learning, pp. 6902–6911 (2019). PMLR

[41] Yang, T.-Y., Rosca, J., Narasimhan, K., Ramadge, P.J.: Projection-based constrained policy optimization. In: International Conference on Learning Representations (2020)

[42] Zhang, J., Ghanem, B.: Ista-net: Interpretable optimization-inspired deep network for image compressive sensing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1828–1837 (2018)

[43] Zhang, J., He, T., Sra, S., Jadbabaie, A.: Why gradient clipping accelerates training: A theoretical justification for adaptivity. arXiv preprint arXiv:1905.11881 (2019)

[44] Zimmerman, R.D., Murillo-Sánchez, C.E., Gan, D.: A MATLAB power system simulation package (2005)

[45] Zoutendijk, G.: Methods of Feasible Directions vol. 960. Elsevier, Amsterdam (1960)