Towards Constituting Mathematical Structures for Learning to Optimize

Jialin Liu [*1] **Xiaohan Chen** [*1] **Zhangyang Wang** [2] **Wotao Yin** [1] **HanQin Cai** [3]

Abstract

Learning to Optimize (L2O), a technique that utilizes machine learning to learn an optimization algorithm automatically from data, has gained arising attention in recent years. A generic L2O approach parameterizes the iterative update rule and learns the update direction as a black-box network. While the generic approach is widely applicable, the learned model can overfit and may not generalize well to out-of-distribution test sets. In this paper, we derive the basic mathematical conditions that successful update rules commonly satisfy. Consequently, we propose a novel L2O model with a mathematics-inspired structure that is broadly applicable and generalized well to out-of-distribution problems. Numerical simulations validate our theoretical findings and demonstrate the superior empirical performance of the proposed L2O model.

1. Introduction

Solving mathematical problems with the help of artificial intelligence, particularly machine learning techniques, has gained increasing interest recently (Davies et al., 2021; Charton, 2021; Polu et al., 2022; Drori et al., 2021). Optimization problems, a type of math problem that finds a point with minimal objective function value in a given space, can also be solved with machine learning models (Gregor & LeCun, 2010; Andrychowicz et al., 2016; Chen et al., 2021a; Bengio et al., 2021). Such technique is coined as *Learning to Optimize (L2O)*.

As an example, we consider an unconstrained optimization problem $\min_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x})$ where F is differentiable. A classic algorithm to solve this problem is *gradient descent*:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla F(\boldsymbol{x}_k), \quad k = 0, 1, 2, \cdots,$$

where the estimate of $\boldsymbol{x}$ is updated in an iterative manner, $\alpha_k > 0$ is a positive scalar named as step size, and the update direction $\alpha_k \nabla F(\boldsymbol{x}_k)$ is aligned with the gradient of F at $\boldsymbol{x}_k$. Instead of the vanilla gradient descent, (Andrychowicz et al., 2016) proposes to parameterize the update rule into a learnable model that suggests the update directions by taking the current estimate and the gradient of F as inputs

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_k, \nabla F(\boldsymbol{x}_k); \phi), \ k = 0, 1, \cdots, K-1, \quad (1)$$

where ϕ is the learnable parameter that can be trained by minimizing a loss function:

$$\min_{\phi} \mathcal{L}(\phi) := \mathbb{E}_{F\in\mathcal{F}}\Big[\sum_{k=1}^{K} w_k F(\boldsymbol{x}_k)\Big], \quad (2)$$

where $\mathcal{F}$ is the problem set we concern and $\{w_k\}_{k=1}^K$ is a set of hand-tuned weighting coefficients. Such loss function aims at finding an update rule of $\boldsymbol{x}_k$ such that the objective values $\{F(\boldsymbol{x}_k)\}$ are as small as possible for all $F \in \mathcal{F}$. This work and its following works (Lv et al., 2017; Wichrowska et al., 2017; Wu et al., 2018; Metz et al., 2019; Chen et al., 2020; Shen et al., 2021; Harrison et al., 2022) show that modeling $\boldsymbol{d}_k$ with a deep neural network and learning a good update rule from data is doable. To train such models, they randomly pick some training samples from $\mathcal{F}$ and build estimates of the loss function defined in (2). Such learned rules are able to generalize to unseen instances from $\mathcal{F}$, i.e., the problems similar to the training samples. This method is quite generic and we can use it as long as we can access the gradient or subgradient of F. For simplicity, we name the method in (1) as *generic L2O*.

Generic L2O is flexible and applicable to a broad class of problems. However, generalizing the learned update rules to out-of-distribution testing problems is quite challenging and a totally free $\boldsymbol{d}_k$ usually leads to overfitting (Metz et al., 2020; 2022). In this paper, we propose an approach to explicitly regularize the update rule $\boldsymbol{d}_k$. Our motivation comes from some common properties that basic optimization algorithms should satisfy. For example, if an iterate $\boldsymbol{x}_k$ reaches one of the minimizers of the objective $F(\boldsymbol{x})$, the next iterate $\boldsymbol{x}_{k+1}$ should be fixed. Such condition is satisfied by many

[*]Equal contribution [1]Alibaba Group (U.S.) Inc, Bellevue, WA, USA [2]Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA [3]Department of Statistics and Data Science and Department of Computer Science, University of Central Florida, Orlando, FL, USA. Correspondence to: HanQin Cai <hqcai@ucf.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

basic algorithms like gradient descent, but not necessarily satisfied if $\boldsymbol{d}_k$ is free to choose. In this paper, we refer to an update rule as *a good rule* if it fulfills these conditions. Our main contributions are three-fold:

1. We strictly describe some basic mathematical conditions that a good update rule should satisfy on convex optimization problems.
2. Based on these conditions, we derive a math-inspired structure of the explicit update rule on $\boldsymbol{x}_k$.
3. We numerically validate that our proposed scheme has superior generalization performance. An update rule trained with randomly generated data can even perform surprisingly well on real datasets.

Organization. The rest of this paper is organized as follows. In Section 2, we derive mathematical structures for L2O models. In Section 3, we propose a novel L2O model and discuss its relationship with other L2O models. In Section 4, we verify the empirical performance of the proposed model via numerical experiments. In Section 5, we conclude the paper with some discussions on the future directions.

2. Deriving Mathematical Structures for L2O Update Rule

In this study, we consider optimization problems in the form of $\min_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x}) = f(\boldsymbol{x}) + r(\boldsymbol{x})$, where $f(\boldsymbol{x})$ is a smooth convex function with Lipschitz continuous gradient, and $r(\boldsymbol{x})$ is a convex function that may be non-smooth. More rigorously, we write that $f \in \mathcal{F}_L(\mathbb{R}^n)$ and $r \in \mathcal{F}(\mathbb{R}^n)$ where the function spaces $\mathcal{F}(\mathbb{R}^n)$ and $\mathcal{F}_L(\mathbb{R}^n)$ are defined below.

Definition 1 (Spaces of Objective Functions)**.** *We define function spaces $\mathcal{F}(\mathbb{R}^n)$ and $\mathcal{F}_L(\mathbb{R}^n)$ as*

$$\mathcal{F}(\mathbb{R}^n) = \left\{ r : \mathbb{R}^n \to \mathbb{R} \;\middle|\; r \text{ is proper, closed and convex} \right\},$$

$$\mathcal{F}_L(\mathbb{R}^n) = \left\{ f : \mathbb{R}^n \to \mathbb{R} \;\middle|\; f \text{ is convex, differentiable, and } \|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \le L\|\boldsymbol{x} - \boldsymbol{y}\|, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \right\}.$$

The first derivative of F plays an important role in the update rule (1). For differentiable function $f \in \mathcal{F}_L(\mathbb{R}^n)$, we can access to its gradient $\nabla f(\boldsymbol{x})$. For non-differentiable function $r \in \mathcal{F}(\mathbb{R}^n)$, we have to use the concepts of *subgradient* and *subdifferential* that are described below.

Definition 2 (Subdifferential and Subgradient)**.** *For $r \in \mathcal{F}(\mathbb{R}^n)$, its subdifferential at $\boldsymbol{x}$ is defined as*

$$\partial r(\boldsymbol{x}) = \left\{ \boldsymbol{g} \in \mathbb{R}^n \;\middle|\; r(\boldsymbol{y}) - r(\boldsymbol{x}) \ge \boldsymbol{g}^\top(\boldsymbol{y} - \boldsymbol{x}),\ \forall \boldsymbol{y} \in \mathbb{R}^n \right\}.$$

Each element in the subdifferential, i.e., each $\boldsymbol{g} \in \partial r(\boldsymbol{x})$, is a subgradient of function r at point $\boldsymbol{x}$.

Settings of $\boldsymbol{d}_k$. We clarify some definitions about the update direction $\boldsymbol{d}_k$. A general parameterized update rule can be written as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{z}_k; \phi), \tag{3}$$

where $\boldsymbol{z}_k \in \mathcal{Z}$ is the *input vector* and $\mathcal{Z}$ is the *input space*. The input vector may involve dynamic information such as $\{\boldsymbol{x}_k, F(\boldsymbol{x}_k), \nabla F(\boldsymbol{x}_k)\}$. Take (Andrychowicz et al., 2016) as an example, as described in (1), the input vector is $\boldsymbol{z}_k = [\boldsymbol{x}_k^\top, \nabla F(\boldsymbol{x}_k)^\top]^\top$ and the input space is $\mathcal{Z} = \mathbb{R}^{2n}$. In our theoretical analysis, we relax the structure of (3) and use a general update rule $\boldsymbol{d}_k : \mathcal{Z} \to \mathbb{R}^n$ instead of the parameterized rule $\boldsymbol{d}_k(\boldsymbol{z}_k; \phi)$ and write (3) as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{z}_k). \tag{4}$$

To facilitate the theoretical analysis, we assume the update direction $\boldsymbol{d}_k$ is differentiable with respect to the input vector, and its derivatives are bounded. Specifically, $\boldsymbol{d}_k$ is taken from the space $\mathcal{D}_C(\mathcal{Z})$ which is defined below.

Definition 3 (Space of Update Rules)**.** *Let $\mathrm{J}\boldsymbol{d}(\boldsymbol{z})$ denote the Jacobian matrix of operator $\boldsymbol{d} : \mathcal{Z} \to \mathbb{R}^n$ and $\|\cdot\|_{\mathrm{F}}$ denote the Frobenius norm, we define the space:*

$$\mathcal{D}_C(\mathcal{Z}) = \left\{ \boldsymbol{d} : \mathcal{Z} \to \mathbb{R}^n \;\middle|\; \boldsymbol{d} \text{ is differentiable}, \|\mathrm{J}\boldsymbol{d}(\boldsymbol{z})\|_{\mathrm{F}} \le C,\ \forall \boldsymbol{z} \in \mathcal{Z} \right\}.$$

In practice, training the deep network that is parameterized from $\boldsymbol{d}_k$ will usually need the derivatives of $\boldsymbol{d}_k$. Thus, the differentiability and bounded Jacobian of $\boldsymbol{d}_k$ are important for this study. Note that $\boldsymbol{d}_k \in \mathcal{D}_C(\mathcal{Z})$ has been commonly used and satisfied in many existing parameterization approaches, e.g., Long Short-Term Memory (LSTM), which is one of the most popular models adopted in L2O (Andrychowicz et al., 2016; Lv et al., 2017).

2.1. Smooth Case

In the smooth case, $\nabla F(\boldsymbol{x})$ equals to $\nabla f(\boldsymbol{x})$ as the non-smooth part $r(\boldsymbol{x}) = 0$. Thus, (4) can be written as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k)). \tag{5}$$

We refer (5) to be a good update rule if it satisfies the following two assumptions:

Asymptotic Fixed Point Condition. We assume that $\boldsymbol{x}_{k+1} = \boldsymbol{x}_*$ as long as $\boldsymbol{x}_k = \boldsymbol{x}_*$, where $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$. In other words, if $\boldsymbol{x}_k$ is exactly a solution, the next iterate should be fixed. Substituting both $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ with $\boldsymbol{x}_*$, we obtain:

$$\boldsymbol{x}_* = \boldsymbol{x}_* - \boldsymbol{d}_k(\boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*)), \quad \text{for all } k = 0, 1, 2, \cdots.$$

Convex analysis theory tells us $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$, and we obtain $\boldsymbol{d}_k(\boldsymbol{x}_*, \mathbf{0}) = \mathbf{0}$. Instead of using this strong assumption, we relax it and assume $\boldsymbol{d}_k(\boldsymbol{x}_*, \mathbf{0}) \to \mathbf{0}$ as $k \to \infty$. Formally, it is written as below and coined as (FP1):

$$\text{For any } \boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}),\ \lim_{k\to\infty} \boldsymbol{d}_k(\boldsymbol{x}_*, \mathbf{0}) = \mathbf{0}. \quad \text{(FP1)}$$

Such a condition can be viewed as an extension to the Fixed Point Encoding (Ryu & Yin, 2022) in optimization, which is useful guidance for designing efficient convex optimization algorithms.

Global Convergence. We assume that, the sequence $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$ converges to one of the minimizers of the objective function $f(\boldsymbol{x})$, as long as it yields the update rule (5). Formally, it is written as (GC1):

$$\text{For any sequences } \{\boldsymbol{x}_k\}_{k=0}^{\infty} \text{ generated by (5), there exists } \boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ such that } \lim_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{x}_*. \quad \text{(GC1)}$$

Actually, assumptions (FP1) and (GC1) are fundamental in the field of optimization and can be satisfied by many basic update schemes. For example, as long as $f \in \mathcal{F}_L(\mathbb{R}^n)$, gradient descent satisfies (FP1) unconditionally and satisfies (GC1) with a properly chosen step size. To outperform the vanilla update rules like gradient descent, a learned update rule $\boldsymbol{d}_k$ should also satisfy (FP1) and (GC1).

The following theorem provides an analysis on $\boldsymbol{d}_k$ under (FP1) and (GC1). Note that proofs of all theorems in this section are deferred to the Appendix.

Theorem 1. *Given $f \in \mathcal{F}_L(\mathbb{R}^n)$, we pick a sequence of operators $\{\boldsymbol{d}_k\}_{k=0}^{\infty}$ with $\boldsymbol{d}_k \in \mathcal{D}_C(\mathbb{R}^{2n})$ and generate $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$ by (5). If both (FP1) and (GC1) hold, then for all $k = 0, 1, 2, \cdots$, there exist $\mathbf{P}_k \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b}_k \in \mathbb{R}^n$ satisfying*

$$\boldsymbol{d}_k(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k)) = \mathbf{P}_k \nabla f(\boldsymbol{x}_k) + \boldsymbol{b}_k,$$

with $\mathbf{P}_k$ is bounded and $\boldsymbol{b}_k \to \mathbf{0}$ as $k \to \infty$.

Theorem 1 illustrates that an update rule $\boldsymbol{d}_k$ is not completely free under assumptions (FP1) and (GC1). It suggests the following structured update rule instead of the free update rule (5):

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{P}_k \nabla f(\boldsymbol{x}_k) - \boldsymbol{b}_k, \quad (6)$$

where $\mathbf{P}_k$ is named as a *preconditioner* and $\boldsymbol{b}_k$ is named as a *bias*. The scheme (6) covers several classical algorithms. For example, with $\mathbf{P}_k = \alpha\mathbf{I}$ and $\boldsymbol{b}_k = \beta(\boldsymbol{x}_k - \alpha\nabla f(\boldsymbol{x}_k) - \boldsymbol{x}_{k-1} - \alpha\nabla f(\boldsymbol{x}_{k-1}))$, it reduces to Nesterov accelerated gradient descent (Nesterov, 1983); with $\boldsymbol{b}_k = \mathbf{0}$ and properly chosen $\mathbf{P}_k$, it covers Newton's method and quasi-Newton method like L-BFGS (Liu & Nocedal, 1989).

Furthermore, Theorem 1 implies that, as long as (FP1) and (GC1) are both satisfied, finding the optimal update direction $\boldsymbol{d}_k$ equals to finding the optimal preconditioner and bias. To train an update rule for smooth convex objective functions $f \in \mathcal{F}_L$, one may parameterize $\mathbf{P}_k$ and $\boldsymbol{b}_k$ instead of parameterizing the entire $\boldsymbol{d}_k$, that is,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{P}_k\big(\boldsymbol{z}_k; \phi\big)\nabla f(\boldsymbol{x}_k) - \boldsymbol{b}_k\big(\boldsymbol{z}_k; \psi\big),$$

where the input vector $\boldsymbol{z}_k = [\boldsymbol{x}_k^\top, \nabla f(\boldsymbol{x}_k)^\top]^\top$. Detailed parameterization and training methods are later described in Section 3.

Note that even if the update rule satisfies (6) instead of (5), one cannot guarantee (FP1) and (GC1) hold. Under our assumptions, if one further makes assumptions on the trajectory of $\{x_k\}_{k=0}^{K}$, then they would obtain a convergence guarantee. This idea falls into a subfield of optimization called Performance Estimation Problem (PEP) (Taylor et al., 2017; Ryu et al., 2020). However, algorithms obtained by PEP are much slower than L2O on specific types of problems that a single user is concerned with. This is because PEP imposes many restrictions on the iteration path, while L2O can find shortcuts for particular problem types.

Although we also impose restrictions (Asymptotic Fixed Point Condition and Global Convergence) on L2O, these restrictions target the asymptotic performance and the final fixed point, rather than the path. As a result, the math-structured L2O proposed in this paper avoids the limitations of convergence guarantees while allowing shortcuts. By analyzing the fixed point, we discover a more effective learnable optimizer structure.

2.2. Non-Smooth Case

In the non-smooth case (i.e., $f(\boldsymbol{x}) = 0$), we use subgradient instead of gradient. A simple extension to (5) is to pick a subgradient $\boldsymbol{g}_k$ from subdifferential $\partial r(\boldsymbol{x}_k)$ in each iteration and use it in the update rule:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_k, \boldsymbol{g}_k), \quad \boldsymbol{g}_k \in \partial r(\boldsymbol{x}_k). \quad (7)$$

Such an update rule is an extension to subgradient descent method: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k$. Compared with gradient descent in the smooth case, the convergence of subgradient descent method is usually unstable, and it may not converge to the solution if a constant step size is used. To guarantee convergence, one has to use certain diminishing step sizes, which may lead to slow convergence. (Bertsekas, 2015). For non-smooth problems, Proximal Point Algorithm (PPA) (Rockafellar, 1976) usually converges faster and more stably than the subgradient descent method. While subgradient descent method uses the *explicit update*, PPA takes *implicit update rule*: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_{k+1}$, where $\boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1})$. Inspired by PPA, we propose to use the

following implicit rule instead:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_{k+1}, \boldsymbol{g}_{k+1}), \quad \boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1}). \tag{8}$$

Generally speaking, it is hard to calculate $\boldsymbol{x}_{k+1}$ from the implicit formula (8) given $\boldsymbol{x}_k$. However, the following discussion provides a mathematical structure of $\boldsymbol{d}_k$ in (8), and, under some mild assumptions, (8) can be written in a much more practical way.

With the same argument in the smooth case, we obtain the math description of the asymptotic fixed point condition: For any $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}} r(\boldsymbol{x})$, there exists a $\boldsymbol{g}_* \in \partial r(\boldsymbol{x}_*)$ such that $\boldsymbol{d}_k(\boldsymbol{x}_*, \boldsymbol{g}_*) \to \boldsymbol{0}$ as $k \to \infty$. With convex analysis theory, it holds that $\boldsymbol{0} \in \partial r(\boldsymbol{x}_*)$ if and only if $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}} r(\boldsymbol{x})$. Thus, it is natural to take $\boldsymbol{g}_* = \boldsymbol{0}$. Formally, it is written as

$$\text{For any } \boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} r(\boldsymbol{x}),\ \lim_{k\to\infty} \boldsymbol{d}_k(\boldsymbol{x}_*, \boldsymbol{0}) = \boldsymbol{0}. \tag{FP2}$$

Similar to (GC1), in the non-smooth case, we require the sequence $\{\boldsymbol{x}_k\}$ converges to one of the minimizers of the function $r(\boldsymbol{x})$. Formally, it is written as

$$\text{For any sequences } \{\boldsymbol{x}_k\}_{k=0}^{\infty} \text{ generated by (8), there exists } \boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} r(\boldsymbol{x}) \text{ such that } \lim_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{x}_*. \tag{GC2}$$

Theorem 2. *Given $r \in \mathcal{F}(\mathbb{R}^n)$, we pick a sequence of operators $\{\boldsymbol{d}_k\}_{k=0}^{\infty}$ with $\boldsymbol{d}_k \in \mathcal{D}_C(\mathbb{R}^{2n})$ and generate $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$ by (8). If both (FP2) and (GC2) hold, then for all $k = 0, 1, 2, \cdots$, there exist $\mathbf{P}_k \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b}_k \in \mathbb{R}^n$ satisfying*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{P}_k \boldsymbol{g}_{k+1} - \boldsymbol{b}_k, \quad \boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1}),$$

with $\mathbf{P}_k$ is bounded and $\boldsymbol{b}_k \to \boldsymbol{0}$ as $k \to \infty$. If we further assume $\mathbf{P}_k$ is symmetric positive definite, then $\boldsymbol{x}_{k+1}$ can be uniquely determined through

$$\boldsymbol{x}_{k+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} r(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}_k + \boldsymbol{b}_k\|^2_{\mathbf{P}_k^{-1}}, \tag{9}$$

where the norm $\|\cdot\|_{\mathbf{P}_k^{-1}}$ is defined as $\|\boldsymbol{x}\|_{\mathbf{P}_k^{-1}} := \sqrt{\boldsymbol{x}^\top \mathbf{P}_k^{-1} \boldsymbol{x}}$.

Define the preconditioned *proximal operator* of $r(\boldsymbol{x})$ as

$$\operatorname{prox}_{r,\mathbf{P}}(\bar{\boldsymbol{x}}) := \arg\min_{\boldsymbol{x}} r(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|^2_{\mathbf{P}^{-1}}, \tag{10}$$

where $\mathbf{P}$ is a symmetric positive definite preconditioner. Then (9) can be written as $\boldsymbol{x}_{k+1} = \operatorname{prox}_{r,\mathbf{P}_k}(\boldsymbol{x}_k - \boldsymbol{b}_k)$. If we set $\mathbf{P}_k = \mathbf{I}$ and $\boldsymbol{b}_k = \boldsymbol{0}$, (9) reduces to the standard PPA. Instead of learning a free update rule as shown in (8), Theorem 2 suggests learning the preconditioner $\mathbf{P}_k$ and the bias $\boldsymbol{b}_k$ in a structured rule, as illustrated in equation (9).

2.3. Composite Case

As special cases, the smooth case and non-smooth case provide important preliminaries to the composite case: $F(\boldsymbol{x}) = f(\boldsymbol{x}) + r(\boldsymbol{x})$. Inspired by Theorems 1 and 2, we use explicit formula for f and implicit formula for r in the composite case:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1}, \boldsymbol{g}_{k+1}), \tag{11}$$

where $\boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1})$ and the input vector $\boldsymbol{z}_k = [\boldsymbol{x}_k^\top, \nabla f(\boldsymbol{x}_k)^\top, \boldsymbol{x}_{k+1}^\top, \boldsymbol{g}_{k+1}^\top]^\top$ and input space is $\mathcal{Z} = \mathbb{R}^{4n}$.

To derive the asymptotic fixed point condition in this case, we use the same arguments in Sections 2.1 and 2.2, and we obtain the following statement for all $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}} F(\boldsymbol{x})$:

$$\lim_{k\to\infty} \boldsymbol{d}_k(\boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, \boldsymbol{g}_*) = \boldsymbol{0}, \text{ for some } \boldsymbol{g}_* \in \partial r(\boldsymbol{x}_*).$$

The convexity of f and r implies that $\boldsymbol{0} \in \nabla f(\boldsymbol{x}_*) + \partial r(\boldsymbol{x}_*)$ if and only if $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}} F(\boldsymbol{x})$. Thus, it holds that $-\nabla f(\boldsymbol{x}_*) \in \partial r(\boldsymbol{x}_*)$. With $\boldsymbol{g}_* = -\nabla f(\boldsymbol{x}_*)$, one could obtain the formal statement of the fixed point condition. For any $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x})$, it holds that

$$\lim_{k\to\infty} \boldsymbol{d}_k(\boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, -\nabla f(\boldsymbol{x}_*)) = \boldsymbol{0}. \tag{FP3}$$

The global convergence is stated as:

$$\text{For any sequences } \{\boldsymbol{x}_k\}_{k=0}^{\infty} \text{ generated by (11), there exists } \boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x}) \text{ such that } \lim_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{x}_*. \tag{GC3}$$

Theorem 3. *Given $f \in \mathcal{F}_L(\mathbb{R}^n)$ and $r \in \mathcal{F}(\mathbb{R}^n)$, we pick a sequence of operators $\{\boldsymbol{d}_k\}_{k=0}^{\infty}$ with $\boldsymbol{d}_k \in \mathcal{D}_C(\mathbb{R}^{4n})$ and generate $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$ by (11). If both (FP3) and (GC3) hold, then for all $k = 0, 1, 2, \cdots$, there exist $\mathbf{P}_k \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b}_k \in \mathbb{R}^n$ satisfying*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{P}_k(\nabla f(\boldsymbol{x}_k) - \boldsymbol{g}_{k+1}) - \boldsymbol{b}_k,\ \boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1}),$$

with $\mathbf{P}_k$ is bounded and $\boldsymbol{b}_k \to \boldsymbol{0}$ as $k \to \infty$. If we further assume $\mathbf{P}_k$ is symmetric positive definite, then $\boldsymbol{x}_{k+1}$ can be uniquely determined given $\boldsymbol{x}_k$ through

$$\boldsymbol{x}_{k+1} = \operatorname{prox}_{r,\mathbf{P}_k}(\boldsymbol{x}_k - \mathbf{P}_k \nabla f(\boldsymbol{x}_k) - \boldsymbol{b}_k). \tag{12}$$

With $\boldsymbol{b}_k = \boldsymbol{0}$ and $\mathbf{P}_k = \alpha\mathbf{I}$, (12) reduces to a standard Proximal Gradient Descent (PGD). Therefore, scheme (12) is actually an extension of PGD with a preconditioner $\mathbf{P}_k$ and a bias $\boldsymbol{b}_k$. Theorem 3 implies that it's enough to learn an extended PGD instead of a free scheme (11).

2.4. Longer Horizon

Those update schemes (5), (8) and (11) introduced in previous sections explicitly depend on only the current status

$\boldsymbol{x}_k$. Now we introduce an auxiliary variable $\boldsymbol{y}_k$ that encodes historical information through operator $\boldsymbol{m}$:

$$\boldsymbol{y}_k = \boldsymbol{m}(\boldsymbol{x}_k, \boldsymbol{x}_{k-1}, \cdots, \boldsymbol{x}_{k-T}). \tag{13}$$

To facilitate parameterization and training, we assume $\boldsymbol{m}$ is differentiable: $\boldsymbol{m} \in \mathcal{D}_C(\mathbb{R}^{(T+1)\times n})$ (see Definition 3). With $\boldsymbol{y}_k$, we could encode more information into the update rule and extend (11) to the following:

$$\begin{aligned} &\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1}, \boldsymbol{g}_{k+1}, \boldsymbol{y}_k, \nabla f(\boldsymbol{y}_k)), \\ &\text{where} \quad \boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1}). \end{aligned} \tag{14}$$

Now let's derive the asymptotic fixed point condition and the global convergence that (13) and (14) should follow. Since the global convergence requires $\boldsymbol{x}_k \to \boldsymbol{x}_*$, the continuity of operator $\boldsymbol{m}$ implies the convergence of sequence $\{\boldsymbol{y}_k\}$. If the limit point of $\{\boldsymbol{y}_k\}$ is denoted by $\boldsymbol{y}_*$, we can assume $\boldsymbol{y}_* = \boldsymbol{x}_*$ without loss of generality because, for any operator $\boldsymbol{m}$, we can always construct another operator by shifting the output: $\hat{\boldsymbol{m}} = \boldsymbol{m} - \boldsymbol{y}_* + \boldsymbol{x}_*$ such that the sequence generated by $\hat{\boldsymbol{m}}$ converges to $\boldsymbol{x}_*$. Roughly speaking, we assume that the sequence $\{\boldsymbol{x}_k, \boldsymbol{y}_k\}$ generated by (13) and (14) satisfies $\boldsymbol{x}_k \to \boldsymbol{x}_*$ and $\boldsymbol{y}_k \to \boldsymbol{x}_*$. By extending (FP3) and (GC3), we obtain the formal statement of our assumptions that (13) and (14) should follow.

For any $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x})$, it holds that

$$\begin{aligned} &\lim_{k\to\infty} \boldsymbol{d}_k(\boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, -\nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*)) = \boldsymbol{0}, \\ &\boldsymbol{m}(\boldsymbol{x}_*, \boldsymbol{x}_*, \cdots, \boldsymbol{x}_*) = \boldsymbol{x}_*. \end{aligned} \tag{FP4}$$

For any sequences $\{\boldsymbol{x}_k, \boldsymbol{y}_k\}_{k=0}^{\infty}$ generated by (13) and (14), there exists one $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x})$ such that

$$\lim_{k\to\infty} \boldsymbol{x}_k = \lim_{k\to\infty} \boldsymbol{y}_k = \boldsymbol{x}_*. \tag{GC4}$$

Theorem 4. *Suppose $T = 1$. Given $f \in \mathcal{F}_L(\mathbb{R}^n)$ and $r \in \mathcal{F}(\mathbb{R}^n)$, we pick an operator $\boldsymbol{m} \in \mathcal{D}_C(\mathbb{R}^{2n})$ and a sequence of operators $\{\boldsymbol{d}_k\}_{k=0}^{\infty}$ with $\boldsymbol{d}_k \in \mathcal{D}_C(\mathbb{R}^{6n})$. If both (FP4) and (GC4) hold, for any bounded matrix sequence $\{\mathbf{B}_k\}_{k=0}^{\infty}$, there exist $\mathbf{P}_{1,k}, \mathbf{P}_{2,k}, \mathbf{A}_k \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b}_{1,k}, \boldsymbol{b}_{2,k} \in \mathbb{R}^n$ satisfying*

$$\begin{aligned} \boldsymbol{x}_{k+1} = \boldsymbol{x}_k &- (\mathbf{P}_{1,k} - \mathbf{P}_{2,k})\nabla f(\boldsymbol{x}_k) - \mathbf{P}_{2,k}\nabla f(\boldsymbol{y}_k) - \boldsymbol{b}_{1,k} \\ &- \mathbf{P}_{1,k}\boldsymbol{g}_{k+1} - \mathbf{B}_k(\boldsymbol{y}_k - \boldsymbol{x}_k), \ \boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1}), \end{aligned} \tag{15}$$

$$\boldsymbol{y}_{k+1} = (\mathbf{I} - \mathbf{A}_k)\boldsymbol{x}_{k+1} + \mathbf{A}_k\boldsymbol{x}_k + \boldsymbol{b}_{2,k} \tag{16}$$

for all $k = 0, 1, 2, \cdots$, with $\{\mathbf{P}_{1,k}, \mathbf{P}_{2,k}, \mathbf{A}_k\}$ are bounded and $\boldsymbol{b}_{1,k} \to \boldsymbol{0}, \boldsymbol{b}_{2,k} \to \boldsymbol{0}$ as $k \to \infty$. If we further assume $\mathbf{P}_{1,k}$ is uniformly symmetric positive definite[1], then we can substitute $\mathbf{P}_{2,k}\mathbf{P}_{1,k}^{-1}$ with $\mathbf{B}_k$ and obtain

$$\begin{aligned} \hat{\boldsymbol{x}}_k &= \boldsymbol{x}_k - \mathbf{P}_{1,k}\nabla f(\boldsymbol{x}_k), \\ \hat{\boldsymbol{y}}_k &= \boldsymbol{y}_k - \mathbf{P}_{1,k}\nabla f(\boldsymbol{y}_k), \\ \boldsymbol{x}_{k+1} &= \operatorname{prox}_{r,\mathbf{P}_{1,k}}\Big((\mathbf{I} - \mathbf{B}_k)\hat{\boldsymbol{x}}_k + \mathbf{B}_k\hat{\boldsymbol{y}}_k - \boldsymbol{b}_{1,k}\Big), \\ \boldsymbol{y}_{k+1} &= \boldsymbol{x}_{k+1} + \mathbf{A}_k(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \boldsymbol{b}_{2,k}. \end{aligned} \tag{17}$$

In the update scheme (17), $\boldsymbol{b}_{1,k}$ and $\boldsymbol{b}_{2,k}$ are biases that play the same role with $\boldsymbol{b}_k$ in (12); $\mathbf{A}_k$ can be viewed as an extension of Nesterov momentum and we name it as an *accelerator*; $\mathbf{P}_{1,k}$ is the preconditioner that plays a similar role as $\mathbf{P}_k$ in (12); $\mathbf{B}_k$ is a balancing term between $\hat{\boldsymbol{x}}_k$ and $\hat{\boldsymbol{y}}_k$. If $\mathbf{B}_k = \mathbf{0}$, then $\boldsymbol{x}_{k+1}$ merely depends on $\boldsymbol{x}_k$ and (17) reduces to (12); and if $\mathbf{B}_k = \mathbf{I}$, then $\boldsymbol{x}_{k+1}$ merely depends on $\boldsymbol{y}_k$ explicitly.

3. An Efficient Math-Inspired L2O Model

As long as the basic assumptions (FP4) and (GC4) hold, one could derive a math-structured update rule (17) from generic update rule (13) and (14). Moreover, we suggest using diagonal matrices for $\mathbf{P}_{1,k}, \mathbf{B}_k, \mathbf{A}_k$ in practice:

$$\mathbf{P}_{1,k} = \operatorname{diag}(\boldsymbol{p}_k), \quad \mathbf{B}_k = \operatorname{diag}(\boldsymbol{b}_k), \quad \mathbf{A}_k = \operatorname{diag}(\boldsymbol{a}_k),$$

where $\boldsymbol{p}_k, \boldsymbol{b}_k, \boldsymbol{a}_k \in \mathbb{R}^n$ are vectors. Let $\mathbf{1}$ be the vector whose all elements are ones and $\odot$ be the element-wise multiplication. The suggested update rule then becomes:

$$\begin{aligned} \hat{\boldsymbol{x}}_k &= \boldsymbol{x}_k - \boldsymbol{p}_k \odot \nabla f(\boldsymbol{x}_k), \\ \hat{\boldsymbol{y}}_k &= \boldsymbol{y}_k - \boldsymbol{p}_k \odot \nabla f(\boldsymbol{y}_k), \\ \boldsymbol{x}_{k+1} &= \operatorname{prox}_{r,\boldsymbol{p}_k}\Big((\mathbf{1} - \boldsymbol{b}_k) \odot \hat{\boldsymbol{x}}_k + \boldsymbol{b}_k \odot \hat{\boldsymbol{y}}_k - \boldsymbol{b}_{1,k}\Big), \\ \boldsymbol{y}_{k+1} &= \boldsymbol{x}_{k+1} + \boldsymbol{a}_k \odot (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \boldsymbol{b}_{2,k}. \end{aligned} \tag{18}$$

We choose diagonal $\mathbf{P}_{1,k}, \mathbf{B}_k, \mathbf{A}_k$ over full matrices for efficiency. On one hand, the diagonal formulation reduces the degree of freedom of the update rule. Therefore, when $\mathbf{P}_{1,k}, \mathbf{B}_k, \mathbf{A}_k$ are parameterized as the output of, or a part of, a learnable model and trained with data, the difficulty of training is decreased and thus the efficiency improved. On the other hand, for a broad class of $r(\boldsymbol{x})$, the proximal operator $\operatorname{prox}_{r,\boldsymbol{p}_k}$ has efficient explicit formula with the diagonal preconditioner[2] $\boldsymbol{p}_k$. Although the non-diagonal formulation may lead to a (theoretically) better convergence rate, it could increase the computational difficulty of the proximal operator $\operatorname{prox}_{r,\mathbf{P}_k}$. An interesting future topic is how to calculate (18) with non-diagonal $\mathbf{P}_k$ and how to train deep models to generate such $\mathbf{P}_k$.

LSTM Parameterization. Similar to (Andrychowicz et al., 2016; Lv et al., 2017), we model $\boldsymbol{p}_k$, $\boldsymbol{a}_k$, $\boldsymbol{b}_k$, $\boldsymbol{b}_{1,k}$,

[1] A sequence of uniformly symmetric positive definite matrices means that the smallest eigenvalues of all symmetric positive definite matrices are uniformly bounded away from zero.

[2] Examples can be found in Appendix D.

$\boldsymbol{b}_{2,k}$ as the output of a coordinate-wise LSTM, which is parameterized by learnable parameters ϕ_{LSTM} and takes the current estimate $\boldsymbol{x}_k$ and the gradient $\nabla f(\boldsymbol{x}_k)$ as the input:

$$\begin{aligned} \boldsymbol{o}_k, \boldsymbol{h}_k &= \text{LSTM}\big(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k), \boldsymbol{h}_{k-1}; \phi_{\text{LSTM}}\big), \\ \boldsymbol{p}_k, \boldsymbol{a}_k, \boldsymbol{b}_k, \boldsymbol{b}_{1,k}, \boldsymbol{b}_{2,k} &= \text{MLP}(\boldsymbol{o}_k; \phi_{\text{MLP}}). \end{aligned} \tag{19}$$

Here, $\boldsymbol{h}_k$ is the internal state maintained by the LSTM with $\boldsymbol{h}_0$ randomly sampled from Gaussian distribution. It is common in classic optimization algorithms to take positive $\boldsymbol{p}_k$ and $\boldsymbol{a}_k$. Hence we post-process $\boldsymbol{p}_k$ and $\boldsymbol{a}_k$ with an additional activation function such as sigmoid and softplus. A "coordinate-wise" LSTM means that the same network is shared across all coordinates of $\boldsymbol{x}_k$, so that this single model can be applied to optimization problems of any scale. (18) and (19) together define an optimization scheme. We call it an *L2O optimizer*.

Training. We train the proposed L2O optimizer, that is to find the optimal ϕ_{LSTM} and ϕ_{MLP} in (19), on a dataset $\mathcal{F}$ of optimization problems. Each sample in $\mathcal{F}$ is an instance of the optimization problem, which we call an *optimizee*, and is characterized by its objective function F. During training, we apply optimizer to each optimizee F for K iterations to generate a sequence of iterates $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K)$, and optimize ϕ_{LSTM} and ϕ_{MLP} by minimizing the following loss function:

$$\min_{\phi_{\text{LSTM}}, \phi_{\text{MLP}}} \mathcal{L}(\phi_{\text{LSTM}}, \phi_{\text{MLP}}) := \frac{1}{|\mathcal{F}|} \sum_{F \in \mathcal{F}} \Big[\frac{1}{K} \sum_{k=1}^{K} F(\boldsymbol{y}_k) \Big].$$

Compared with Algorithm Unrolling. Algorithm Unrolling (Monga et al., 2019) is another line of works parallel to generic L2O. It was first proposed to fast approximate the solution of sparse coding (Gregor & LeCun, 2010) which is named Learned ISTA (LISTA). Since then, many efforts have been made to further improve or better understand LISTA (Xin et al., 2016; Metzler et al., 2017; Moreau & Bruna, 2017; Chen et al., 2018; Liu et al., 2019; Ito et al., 2019; Chen et al., 2021b), as well as applying this idea to different optimization problems (Yang et al., 2016; Zhang & Ghanem, 2018; Adler & Öktem, 2018; Mardani et al., 2018; Gupta et al., 2018; Solomon et al., 2019; Xie et al., 2019; Cai et al., 2021).

The main difference between our method and algorithm-unrolling methods lies at how parameterization is done. Different from the LSTM parameterization (19), algorithm-unrolling methods turn $\boldsymbol{p}_k$, $\boldsymbol{a}_k$, $\boldsymbol{b}_k$, $\boldsymbol{b}_{1,k}$, $\boldsymbol{b}_{2,k}$ themselves as learnable parameters and directly optimize them from data.

However, such direct parameterization causes limitations on the flexibility of the model in many ways. It loses the ability to capture dynamics between iterations and tends to memorize more about the datasets. Moreover, direct parameterization means that we need to match the dimensions of the learnable parameters with the problem scale, which implies that the trained model can not be applied to optimization problems of a different scale at all during inference time. Although this can be worked around by reducing the parameters to scalars, it will significantly decrease the capacity of the model.

Table 1. Comparison between different types of methods of their theoretical convergence analysis (**Theory**), convergence speed (**Fast**), and **Flexibility**.

Methods	Theory	Fast	Flexibility
Classic Algorithms	✓	–	✓
Algorithm Unrolling	✓	✓	–
Generic L2O	–	✓	✓
Math-Inspired (Ours)	✓	✓	✓

In fact, despite the difference in parameterization, our proposed scheme (18) covers many existing algorithm-unrolling methods in the literature. For example, if we use the standard LASSO objective as the objective function $F(\boldsymbol{x})$ and set $\boldsymbol{a}_k = \boldsymbol{b}_k = \boldsymbol{b}_{2,k} = \boldsymbol{0}$, we will recover Step-LISTA (Ablin et al., 2019) with properly chosen $\boldsymbol{p}_k, \boldsymbol{b}_{1,k}$. More details and proofs are provided in the Appendix.

Compared with Generic L2O. Our proposed method uses a similar coordinate-wise LSTM parameterization as generic L2O methods. Therefore, both of these two share the flexibility of being applied to optimization problems of any scale. However, we constrain the update rule to have a specific form, i.e., the formulation in (18). The reduced degree of freedom enables the convergence analysis in Section 2 from the theoretical perspective and empirically works as a regularization so that the trained L2O optimizer is more stable and can generalize better, which is validated by our numerical observations in the next section.

We summarize in Table 1 the comparison between classic algorithms, algorithm-unrolling methods, generic L2O methods, and our math-inspired method in terms of theoretical convergence analysis, convergence speed, and flexibility.

Relationship with Operator Learning. In this paper, we consider the proximal operator as an accessible basic routine and focus on learning the overall update rule that results in fast convergence. However, if the proximal operator is difficult to compute, one might explore another aspect of L2O: learning fast approximations of proximal operators (Zhang et al., 2017; Meinhardt et al., 2017; Li et al., 2022). For instance, if the matrix $\mathbf{P}_{1,k}$ in (17) is not diagonal, calculating the proximal operator would be challenging. Additionally, our assumptions (FP4) and (GC4) allow the optimization problem $F(\boldsymbol{x})$ to have multiple optima. As a result, investigating diverse optima following the idea of (Li et al., 2022) using our proposed scheme could be an

intriguing topic for future research.

Relationship with Meta-Learning. L2O and Meta-Learning are closely related topics as they both deal with learning from experience in previous tasks to improve performance on new tasks. L2O treats tasks as optimization problems and aims to discover superior optimization algorithms, while Meta-Learning is a more comprehensive concept that focuses on training a model on a set of related tasks or problems to swiftly adapt to new, unseen tasks using knowledge acquired from prior tasks. For example, in our paper's equation (1), L2O seeks to learn $\boldsymbol{d}_k$ while keeping the initialization $\boldsymbol{x}_0$ fixed or randomized. Meanwhile, a typical Meta-Learning method like (Khodak et al., 2019a) learns a suitable initialization from a series of observed tasks, enabling quick adaptation to unseen tasks. In addition to the initialization, (Khodak et al., 2019b) also learns a meta-learning rate shared among different tasks. Furthermore, one can learn a regularization term based on the distance to a bias vector (Denevi et al., 2019) or even a conditional regularization term (Denevi et al., 2020) from a set of tasks.

4. Experimental Results

We strictly follow the setting in (Lv et al., 2017) for experiment setup. More specifically, in all our experiments on the LSTM-based L2O models (including our method and other baseline competitors), we use two-layer LSTM cells with 20 hidden units with sigmoid activation functions. During training, each minibatch contains 64 instances of optimization problems, to which the L2O optimizers will be applied for 100 iterations. The 100 iterations are evenly segmented into 5 periods of 20 iterations. Within each of these, the L2O optimizers are trained with truncated Backpropagation Through Time (BPTT) with an Adam optimizer. All models are trained with 500 minibatches (32,000 optimization problems in total) generated synthetically, but are evaluated on both synthesized testing sets and real-world testing sets. We elaborate more on the data generation in the Appendix. The code is available online at `https://github.com/xhchrn/MS4L2O`.

4.1. Ablation Study

We conduct an ablation study on LASSO to figure out the roles of $\boldsymbol{p}_k, \boldsymbol{a}_k, \boldsymbol{b}_k, \boldsymbol{b}_{1,k}, \boldsymbol{b}_{2,k}$ in our proposed scheme (18). Both the training and testing samples are independently sampled from the same random distribution. The form of LASSO is given below

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x}) = \frac{1}{2}\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|^2 + \lambda\|\boldsymbol{x}\|_1, \tag{20}$$

where each tuple of $(\mathbf{A}, \boldsymbol{b}, \lambda)$ determines an objective function and thus a LASSO problem instance. The size of each instance is $\mathbf{A} \in \mathbb{R}^{250\times 500}$ and $\boldsymbol{b} \in \mathbb{R}^{500}$ and other details are provided in the Appendix. We do not fix $\mathbf{A}$ and, instead, let each LASSO instance take an independently generated $\mathbf{A}$. This setting is fundamentally more challenging than those in most of algorithm-unrolling works (Gregor & LeCun, 2010; Liu et al., 2019; Ablin et al., 2019; Behrens et al., 2021).

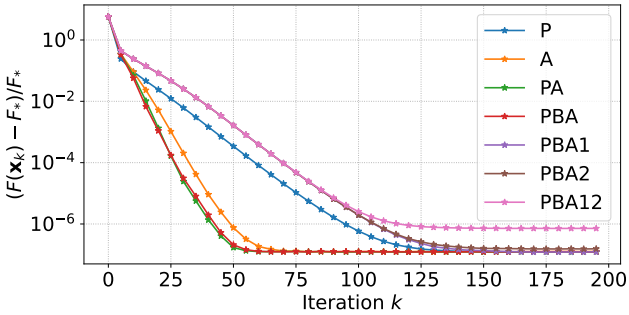

Figure 1. Ablation study on LASSO.

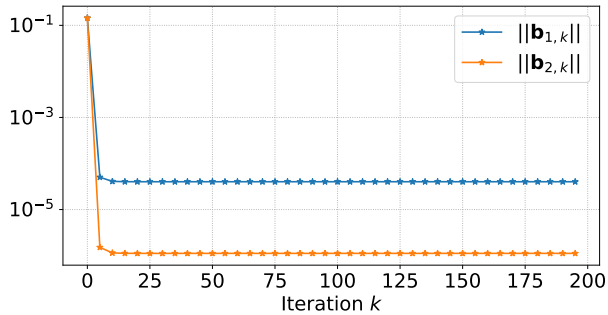

Figure 2. Visualization of the learned $\boldsymbol{b}_{1,k}, \boldsymbol{b}_{2,k}$.

On the benchmark, we compare the following settings: **PBA12**: $\boldsymbol{p}_k$,$\boldsymbol{a}_k$,$\boldsymbol{b}_k$,$\boldsymbol{b}_{1,k}$,$\boldsymbol{b}_{2,k}$ are all learnable. **PBA1**: $\boldsymbol{p}_k$,$\boldsymbol{a}_k$,$\boldsymbol{b}_k$,$\boldsymbol{b}_{1,k}$ are learnable; $\boldsymbol{b}_{2,k}$ is fixed as $\mathbf{0}$. **PBA2**: $\boldsymbol{p}_k$,$\boldsymbol{a}_k$,$\boldsymbol{b}_k$,$\boldsymbol{b}_{2,k}$ are learnable; $\boldsymbol{b}_{1,k}$ is fixed as $\mathbf{0}$. **PBA**: $\boldsymbol{p}_k$,$\boldsymbol{a}_k$,$\boldsymbol{b}_k$ are learnable; $\boldsymbol{b}_{2,k}$ and $\boldsymbol{b}_{1,k}$ are both fixed as $\mathbf{0}$. **PA**: $\boldsymbol{p}_k$,$\boldsymbol{a}_k$ are learnable; $\boldsymbol{b}_{2,k}$ and $\boldsymbol{b}_{1,k}$ are both fixed as $\mathbf{0}$; $\boldsymbol{b}_k$ is fixed as $\mathbf{1}$. **P**: only $\boldsymbol{p}_k$ is learnable; $\boldsymbol{a}_k$, $\boldsymbol{b}_{2,k}$, $\boldsymbol{b}_{1,k}$ are fixed as $\mathbf{0}$; $\boldsymbol{b}_k$ is fixed as $\mathbf{1}$. **A**: only $\boldsymbol{a}_k$ is learnable; $\boldsymbol{b}_{2,k}$ and $\boldsymbol{b}_{1,k}$ are both fixed as $\mathbf{0}$; $\boldsymbol{b}_k$ is fixed as $\mathbf{1}$; $\boldsymbol{p}_k$ is fixed as $(1/L)\mathbf{1}$. The results are reported in Figure 1.

In Figure 1, F_* denotes the best objective value of each instance and $(F(\boldsymbol{x}_k) - F_*)/F_*$ measures the average optimality gap on the test set. Each curve describes the convergence performance of a learned optimizer. From Figure 1, one can conclude that, with all of the learned optimizers, convergence can be reached within the machine precision.

An interesting observation is that the proposed scheme (18) may not benefit from parameterizing and learning more components. Comparing PBA, PBA1, PBA2 and PBA12, we find that fixing $\boldsymbol{b}_{1,k} = \boldsymbol{b}_{2,k} = \mathbf{0}$ is a better choice than

learning them. This phenomenon can be explained by our theory. In Theorem 4, both $\boldsymbol{b}_{2,k}$ and $\boldsymbol{b}_{1,k}$ are expected to converge to zero, otherwise, the convergence would be violated. However, in Figure 2 where we report the average values of $\|\boldsymbol{b}_{1,k}\|$ and $\|\boldsymbol{b}_{2,k}\|$ in PBA12, both $\|\boldsymbol{b}_{1,k}\|$ and $\|\boldsymbol{b}_{2,k}\|$ converge to a relatively small value after a few (≤ 10) iterations, but there is no guarantee that they converge exactly to zero given parameterization (19). Such observation actually validates our theories and suggests to fix $\boldsymbol{b}_{2,k}$ and $\boldsymbol{b}_{1,k}$ as $\mathbf{0}$ instead of learning them.

A Simplified Scheme. Furthermore, comparing PA and PBA in Figure 1, we find that parameterizing $\boldsymbol{b}_k$ does not show significant benefits. To reduce computational overhead, we adopt PA and fix $\boldsymbol{b}_{1,k} = \boldsymbol{b}_{2,k} = \mathbf{0}$ and $\boldsymbol{b}_k = \mathbf{1}$, which reduces (18) and (19) to the following simplified scheme:

$$\begin{aligned} \boldsymbol{o}_k, \boldsymbol{h}_k &= \mathrm{LSTM}\big(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k), \boldsymbol{h}_{k-1}; \phi_{\mathrm{LSTM}}\big), \\ \boldsymbol{p}_k, \boldsymbol{a}_k &= \mathrm{MLP}(\boldsymbol{o}_k; \phi_{\mathrm{MLP}}), \\ \boldsymbol{x}_{k+1} &= \mathrm{prox}_{r,\boldsymbol{p}_k}\big(\boldsymbol{y}_k - \boldsymbol{p}_k \nabla f(\boldsymbol{y}_k)\big), \\ \boldsymbol{y}_{k+1} &= \boldsymbol{x}_{k+1} + \boldsymbol{a}_k \odot (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k). \end{aligned} \tag{L2O-PA}$$

4.2. Comparison with Competitors

We compare (L2O-PA) with some competitors in two settings. In the first setting (also coined as In-Distribution), the training and testing samples are generated independently with the same distribution. The second setting is much more challenging, where the models are trained on randomly synthetic data but tested on real data, which is coined as Out-of-Distribution or OOD.

Solving LASSO Regression. In-Distribution experiments follow the settings in Section 4.1, where the training and testing sets are generated randomly and independently, but share the same distribution. In contrast, Out-of-Distribution (OOD) experiments also generate training samples similarly to In-Distribution experiments, but the test samples are generated based on a natural image benchmark BSDS500 (Martin et al., 2001). Detailed information about the synthetic and real data used can be found in the Appendix.

We first choose ISTA, FISTA (Beck & Teboulle, 2009), and Adam (Kingma & Ba, 2014) as baselines since they are classical manually-designed update rules. We choose a state-of-the-art Algorithm-Unrolling method Ada-LISTA (Aberdam et al., 2021) as another baseline since it is applicable in the settings of various $\mathbf{A}$. Additionally, we choose two generic L2O methods that following (3) as baselines: L2O-DM (Andrychowicz et al., 2016) and L2O-RNNprop (Lv et al., 2017). Finally, we choose AdamHD (Baydin et al., 2017), a hyperparameter optimization (HPO) method that adaptively tunes the learning rate in Adam with online learning, as a baseline. Since Ada-LISTA is not applicable to problems

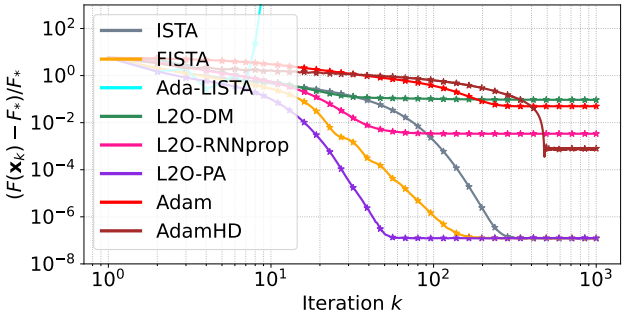

Figure 3. LASSO: Train and test on synthetic data.

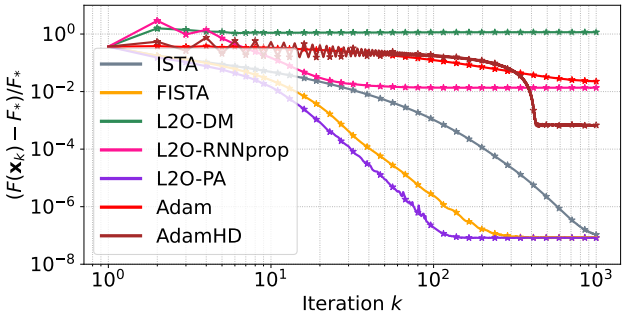

Figure 4. LASSO: Train on synthetic data and test on real data.

that have different sizes with training samples, we only compare our method with Ada-LISTA in In-Distribution experiments.

In-Distribution results are reported in Figure 3 and Out-of-Distribution results are reported in Figure 4. In both of the settings, the proposed (L2O-PA) performs competitively. In the OOD setting, the other two learning-based methods, L2O-DM and L2O-RNNprop, both struggle to converge with optimality tolerance 10^{-2}, but (L2O-PA) is still able to converge until touching the machine precision.

Solving ℓ_1-regularized Logistic Regression. An ℓ_1-regularized logistic regression for binary classification is characterized by set of training examples $\{(\boldsymbol{a}_i, b_i) \in \mathbb{R}^n \times \{0,1\}\}_{i=1}^m$ where $\boldsymbol{a}_i$ is an n-dimensional feature vector and b_i is a binary label. To solve the ℓ_1-regularized logistic regression problem is to find an optimal $\boldsymbol{x}_*$ so that $h(\boldsymbol{a}_i^\top \boldsymbol{x}_*)$ predicts $p(b_i|\boldsymbol{a}_i)$ well, where $h(c) = 1/(1+e^{-c})$ is the logistic function. The exact formula of the objective function can be found in the Appendix.

We train L2O models on randomly generated logistic regression datasets for binary classification. Each dataset contains 1,000 samples that have 50 features. We evaluate the trained models in both the In-Distribution and OOD settings. Results are shown in Figures 5 and 6 respectively. Details about synthetic data generation and real-world datasets are provided in the Appendix.

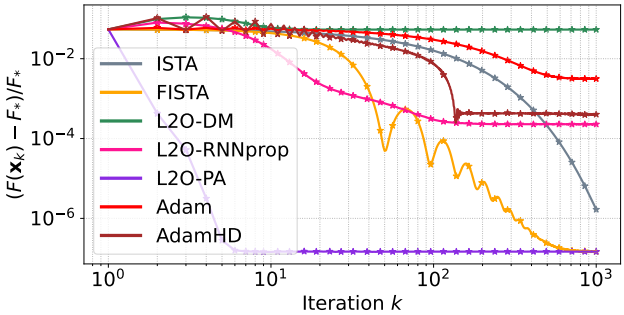

Figure 5. Logistic: Train and test on synthetic data.

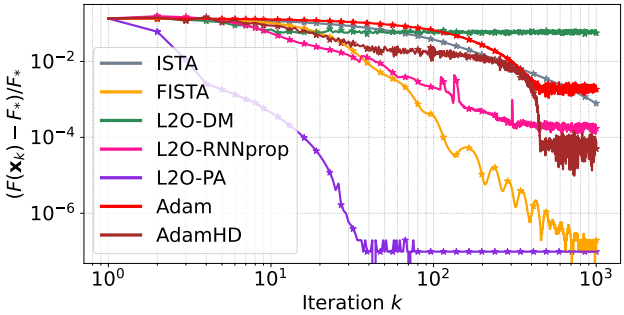

Figure 6. Logistic: Train on synthetic data and test on real data.

Under the In-Distribution setting (Figure 5), our method, i.e., L2O-PA, can converge to optimal solutions within 10 iterations, almost 100× faster than FISTA, while the other two generic L2O baselines fail to converge. When tested on OOD optimization problems (Figure 6), L2O-PA can still converge within 100 iterations, faster than FISTA by more than 10 times.

5. Conclusions and Future Work

By establishing the basic conditions of the update rule, we incorporate mathematical structures into machine-learning-based optimization algorithms (learned optimizers). In our settings, we do not learn the entire update rule as a black box. Instead, we propose to learn a preconditioner and an accelerator, and then construct the update rule in the style of proximal gradient descent. Our approach is applicable to a broad class of optimization problems while providing superior empirical performance. This study actually provides some insights toward answering an important question in L2O: Which part of the model should be mathematically grounded and which part could be learned?

There are several lines of future work. Firstly, relaxing the assumption of convexity and studying the nonconvex settings will be a significant future direction. Another interesting topic is to extend (13) and consider update rules that adopt more input information and longer horizons.

Acknowledgements

The work of HanQin Cai was partially supported by NSF DMS 2304489.

References

Aberdam, A., Golts, A., and Elad, M. Ada-lista: Learned solvers adaptive to varying models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Ablin, P., Moreau, T., Massias, M., and Gramfort, A. Learning step sizes for unfolded sparse coding. *Advances in Neural Information Processing Systems*, 32, 2019.

Adler, J. and Öktem, O. Learned primal-dual reconstruction. *IEEE Transactions on Medical Imaging*, 37(6): 1322–1332, 2018.

Aharon, M., Elad, M., and Bruckstein, A. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006.

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

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*, 2017.

Beck, A. and Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Behrens, F., Sauder, J., and Jung, P. Neurally augmented ALISTA. In *International Conference on Learning Representations*, 2021.

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

Bertsekas, D. *Convex optimization algorithms*. Athena Scientific, 2015.

Cai, H., Liu, J., and Yin, W. Learned robust PCA: A scalable deep unfolding approach for high-dimensional outlier detection. *Advances in Neural Information Processing Systems*, 34:16977–16989, 2021.

Charton, F. Linear algebra with transformers. *arXiv preprint arXiv:2112.01898*, 2021.

Chen, T., Zhang, W., Jingyang, Z., Chang, S., Liu, S., Amini, L., and Wang, Z. Training stronger baselines for learning to optimize. *Advances in Neural Information Processing Systems*, 33:7332–7343, 2020.

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

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

Chen, X., Liu, J., Wang, Z., and Yin, W. Hyperparameter tuning is all you need for lista. *Advances in Neural Information Processing Systems*, 34:11678–11689, 2021b.

Cowen, B., Saridena, A. N., and Choromanska, A. Lsalsa: accelerated source separation via learned sparse coding. *Machine Learning*, 108:1307–1327, 2019.

Davies, A., Veličković, P., Buesing, L., Blackwell, S., Zheng, D., Tomašev, N., Tanburn, R., Battaglia, P., Blundell, C., Juhász, A., et al. Advancing mathematics by guiding human intuition with ai. *Nature*, 600(7887):70–74, 2021.

Denevi, G., Ciliberto, C., Grazzi, R., and Pontil, M. Learning-to-learn stochastic gradient descent with biased regularization. In *International Conference on Machine Learning*, pp. 1566–1575. PMLR, 2019.

Denevi, G., Pontil, M., and Ciliberto, C. The advantage of conditional meta-learning for biased regularization and fine tuning. *Advances in Neural Information Processing Systems*, 33:964–974, 2020.

Drori, I., Tran, S., Wang, R., Cheng, N., Liu, K., Tang, L., Ke, E., Singh, N., Patti, T. L., Lynch, J., et al. A neural network solves and generates mathematics problems by program synthesis: Calculus, differential equations, linear algebra, and more. *arXiv preprint arXiv:2112.15594*, 2021.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Gregor, K. and LeCun, Y. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pp. 399–406, 2010.

Gupta, H., Jin, K. H., Nguyen, H. Q., McCann, M. T., and Unser, M. Cnn-based projected gradient descent for consistent ct image reconstruction. *IEEE transactions on medical imaging*, 37(6):1440–1453, 2018.

Harrison, J., Metz, L., and Sohl-Dickstein, J. A closer look at learned optimization: Stability, robustness, and inductive biases. *arXiv preprint arXiv:2209.11208*, 2022.

Ito, D., Takabe, S., and Wadayama, T. Trainable ISTA for sparse signal recovery. *IEEE Transactions on Signal Processing*, 67(12):3113–3125, 2019.

Khodak, M., Balcan, M., and Talwalkar, A. Provable guarantees for gradient-based meta-learning. In *International Conference on Machine Learning*, 2019a.

Khodak, M., Balcan, M.-F. F., and Talwalkar, A. S. Adaptive gradient-based meta-learning methods. *Advances in Neural Information Processing Systems*, 32, 2019b.

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

Li, L., Aigerman, N., Kim, V. G., Li, J., Greenewald, K., Yurochkin, M., and Solomon, J. Learning proximal operators to discover multiple optima. *arXiv preprint arXiv:2201.11945*, 2022.

Liu, D. C. and Nocedal, J. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

Liu, J., Chen, X., Wang, Z., and Yin, W. ALISTA: Analytic weights are as good as learned weights in LISTA. In *International Conference on Learning Representations (ICLR)*, 2019.

Lv, K., Jiang, S., and Li, J. Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning*, pp. 2247–2255. PMLR, 2017.

Mardani, M., Sun, Q., Donoho, D., Papyan, V., Monajemi, H., Vasanawala, S., and Pauly, J. Neural proximal gradient descent for compressive imaging. *Advances in Neural Information Processing Systems*, 31, 2018.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pp. 416–423, July 2001.

Meinhardt, T., Moller, M., Hazirbas, C., and Cremers, D. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1781–1790, 2017.

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., and Sohl-Dickstein, J. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pp. 4556–4565. PMLR, 2019.

Metz, L., Maheswaranathan, N., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020.

Metz, L., Freeman, C. D., Harrison, J., Maheswaranathan, N., and Sohl-Dickstein, J. Practical tradeoffs between memory, compute, and performance in learned optimizers. In *Conference on Lifelong Learning Agents*, pp. 142–164. PMLR, 2022.

Metzler, C. A., Mousavi, A., and Baraniuk, R. G. Learned D-AMP: Principled neural network based compressive image recovery. *Advances in Neural Information Processing Systems*, pp. 1773–1784, 2017.

Monga, V., Li, Y., and Eldar, Y. C. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *arXiv preprint arXiv:1912.10557*, 2019.

Moreau, T. and Bruna, J. Understanding neural sparse coding with matrix factorization. In *International Conference on Learning Representation (ICLR)*, 2017.

Nesterov, Y. E. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pp. 543–547, 1983.

Parikh, N. and Boyd, S. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

Park, Y., Dhar, S., Boyd, S., and Shah, M. Variable metric proximal gradient method with diagonal barzilai-borwein stepsize. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3597–3601. IEEE, 2020.

Polu, S., Han, J. M., Zheng, K., Baksys, M., Babuschkin, I., and Sutskever, I. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*, 2022.

Rockafellar, R. T. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

Ryu, E. K. and Yin, W. *Large-Scale Convex Optimization: Algorithms & Analyses via Monotone Operators*. Cambridge University Press, 2022.

Ryu, E. K., Taylor, A. B., Bergeling, C., and Giselsson, P. Operator splitting performance estimation: Tight contraction factors and optimal parameter selection. *SIAM Journal on Optimization*, 30(3):2251–2271, 2020.

Shen, J., Chen, X., Heaton, H., Chen, T., Liu, J., Yin, W., and Wang, Z. Learning a minimax optimizer: A pilot study. In *International Conference on Learning Representations*, 2021.

Solomon, O., Cohen, R., Zhang, Y., Yang, Y., He, Q., Luo, J., van Sloun, R. J., and Eldar, Y. C. Deep unfolded robust PCA with application to clutter suppression in ultrasound. *IEEE transactions on medical imaging*, 39(4):1051–1063, 2019.

Taylor, A. B., Hendrickx, J. M., and Glineur, F. Exact worst-case performance of first-order methods for composite convex optimization. *SIAM Journal on Optimization*, 27 (3):1283–1313, 2017.

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Freitas, N., and Sohl-Dickstein, J. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, pp. 3751–3760. PMLR, 2017.

Wu, Y., Ren, M., Liao, R., and Grosse, R. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

Xie, X., Wu, J., Liu, G., Zhong, Z., and Lin, Z. Differentiable linearized admm. In *International Conference on Machine Learning*, pp. 6902–6911. PMLR, 2019.

Xin, B., Wang, Y., Gao, W., Wipf, D., and Wang, B. Maximal sparsity with deep networks? *Advances in Neural Information Processing Systems*, 29:4340–4348, 2016.

Yang, Y., Sun, J., Li, H., and Xu, Z. Deep ADMM-Net for compressive sensing MRI. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 10–18, 2016.

Zhang, J. and 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.

Zhang, K., Zuo, W., Gu, S., and Zhang, L. Learning deep cnn denoiser prior for image restoration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3929–3938, 2017.

A. Proof of Theorems

A.1. Preliminaries

In our proofs, $\|\mathcal{A}\|$ denotes the spectral norm of matrix $\mathcal{A}$, $\|\mathcal{A}\|_{\mathrm{F}}$ denotes the Frobenius norm of matrix $\mathcal{A}$, $\|\boldsymbol{x}\|$ denotes the ℓ_2-norm of vector $\boldsymbol{x}$, and $\|\boldsymbol{x}\|_1$ denotes the ℓ_1-norm of vector $\boldsymbol{x}$.

Before the proofs of those theorems in the main text, we describe a lemma here to facilitate our proofs.

Lemma 1. *For any operator $\boldsymbol{o} \in \mathcal{D}_C(\mathbb{R}^{m\times n})$ and any $\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2, \cdots, \boldsymbol{x}_m, \boldsymbol{y}_m \in \mathbb{R}^n$, there exist matrices $\mathbf{J}_1, \mathbf{J}_2, \cdots, \mathbf{J}_m \in \mathbb{R}^{n\times n}$ such that*

$$\boldsymbol{o}(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m) - \boldsymbol{o}(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m) = \sum_{j=1}^{m} \mathbf{J}_j(\boldsymbol{x}_j - \boldsymbol{y}_j), \tag{21}$$

and

$$\|\mathbf{J}_1\| \le \sqrt{n}C, \quad \|\mathbf{J}_2\| \le \sqrt{n}C, \quad \cdots, \quad \|\mathbf{J}_m\| \le \sqrt{n}C. \tag{22}$$

Proof. Since $\boldsymbol{o} \in \mathcal{D}_C(\mathbb{R}^{m\times n})$, the outcome of operator $\boldsymbol{o}$ is an n-dimensional vector. Now we denote the i-th element as $o_i(1 \le i \le n)$ and write operator $\boldsymbol{o}$ in a matrix form:

$$\boldsymbol{o}(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m) = \Big[o_1(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m),\ \cdots,\ o_n(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m)\Big]^{\top},$$
$$\boldsymbol{o}(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m) = \Big[o_1(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m),\ \cdots,\ o_n(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m)\Big]^{\top}.$$

Applying the Mean Value Theorem on $o_i(1 \le i \le n)$, one could obtain

$$\begin{aligned} & o_i(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m) - o_i(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m) \\ = & \sum_{j=1}^{m} \left\langle \frac{\partial o_i}{\partial \boldsymbol{x}_j}\Big(\xi_i \boldsymbol{x}_1 + (1-\xi_i)\boldsymbol{y}_1, \cdots, \xi_i \boldsymbol{x}_m + (1-\xi_i)\boldsymbol{y}_m\Big), \boldsymbol{x}_j - \boldsymbol{y}_j \right\rangle, \end{aligned} \tag{23}$$

for some $\xi_i \in (0,1)$. For simplicity, we denote

$$\boldsymbol{z}_i := \Big[\xi_i \boldsymbol{x}_1 + (1-\xi_i)\boldsymbol{y}_1, \xi_i \boldsymbol{x}_2 + (1-\xi_i)\boldsymbol{y}_2, \cdots, \xi_i \boldsymbol{x}_m + (1-\xi_i)\boldsymbol{y}_m\Big]^{\top}, \quad \forall 1 \le i \le n,$$

and stack all partial derivatives into one matrix as

$$\mathbf{J}_j = \left[\frac{\partial o_1}{\partial \boldsymbol{x}_j}(\boldsymbol{z}_1),\ \frac{\partial o_2}{\partial \boldsymbol{x}_j}(\boldsymbol{z}_2),\ \cdots,\ \frac{\partial o_n}{\partial \boldsymbol{x}_j}(\boldsymbol{z}_n)\right]^{\top} \in \mathbb{R}^{n\times n}.$$

Then one can immediately get (21) from (23). The upper bound of $\|\mathbf{J}_j\|(1 \le j \le m)$ can be estimated by

$$\|\mathbf{J}_j\|^2 \le \|\mathbf{J}_j\|_{\mathrm{F}}^2 = \sum_{i=1}^{n} \left\| \frac{\partial o_i}{\partial \boldsymbol{x}_j}(\boldsymbol{z}_i) \right\|^2 \le nC^2.$$

Therefore, it concludes that $\|\mathbf{J}_j\| \le \sqrt{n}C$ for all $1 \le j \le m$, which finishes the proof.

Note that such upper bound of $\|\mathbf{J}_j\|$ is tight. We could NOT directly conclude $\|\mathbf{J}_j\| \le C$ because $\partial o_1/\partial \boldsymbol{x}_j, \cdots, \partial o_n/\partial \boldsymbol{x}_j$ are evaluated respectively at points $\boldsymbol{z}_1, \cdots, \boldsymbol{z}_n$, and, consequently, the whole matrix $\mathbf{J}_j$ is not a Jacobian matrix of operator $\boldsymbol{o}$. □

A.2. Proof of Theorem 1

Proof. Denote

$$\hat{\boldsymbol{d}}_k := \boldsymbol{d}_k(\boldsymbol{x}_*, \mathbf{0}).$$

Plugging the above equation into (5), we obtain

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k)) + \boldsymbol{d}_k(\boldsymbol{x}_*, \mathbf{0}) - \hat{\boldsymbol{d}}_k.$$

Applying Lemma 1, we have

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) - \mathbf{J}_{2,k}\nabla f(\boldsymbol{x}_k) - \hat{\boldsymbol{d}}_k,$$

for some $\mathbf{J}_{1,k}, \mathbf{J}_{2,k} \in \mathbb{R}^{n\times n}$ that satisfy

$$\|\mathbf{J}_{1,k}\| \le \sqrt{n}C, \quad \|\mathbf{J}_{2,k}\| \le \sqrt{n}C.$$

Define

$$\mathbf{P}_k := \mathbf{J}_{2,k}, \quad \boldsymbol{b}_k := \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) + \hat{\boldsymbol{d}}_k.$$

Then we obtain $\boldsymbol{d}_k(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k)) = \mathbf{P}_k\nabla f(\boldsymbol{x}_k) + \boldsymbol{b}_k$ and it holds that

$$\begin{aligned}\|\mathbf{P}_k\| &\le \sqrt{n}C,\\ \|\boldsymbol{b}_k\| &\le \sqrt{n}C\|\boldsymbol{x}_k - \boldsymbol{x}_*\| + \|\hat{\boldsymbol{d}}_k\|.\end{aligned}$$

Assumption (FP1) leads to $\|\hat{\boldsymbol{d}}_k\| \to 0$ and Assumption (GC1) leads to $\|\boldsymbol{x}_k - \boldsymbol{x}_*\| \to 0$. Consequently, $\|\boldsymbol{b}_k\| \to 0$, which finishes the proof. □

A.3. Proof of Theorem 2

Proof. Following the same proof line with that of Theorem 1, we first denote $\hat{\boldsymbol{d}}_k := \boldsymbol{d}_k(\boldsymbol{x}_*, \mathbf{0})$ and then obtain

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_{k+1}, \boldsymbol{g}_{k+1}) + \boldsymbol{d}_k(\boldsymbol{x}_*, \mathbf{0}) - \hat{\boldsymbol{d}}_k\\ &= \boldsymbol{x}_k - \mathbf{J}_{1,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) - \mathbf{J}_{2,k}\boldsymbol{g}_{k+1} - \hat{\boldsymbol{d}}_k,\\ &= \boldsymbol{x}_k - \Big(\underbrace{\mathbf{J}_{2,k}}_{\mathbf{P}_k}\boldsymbol{g}_{k+1} + \underbrace{\mathbf{J}_{1,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) + \hat{\boldsymbol{d}}_k}_{\boldsymbol{b}_k}\Big),\end{aligned}$$

where $\boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1})$, $\mathbf{P}_k$ is bounded, and $\boldsymbol{b}_k \to \mathbf{0}$ as $k \to \infty$. In another word, $\boldsymbol{x}_{k+1}$ satisfies

$$\boldsymbol{x}_k - \boldsymbol{b}_k \in \boldsymbol{x}_{k+1} + \mathbf{P}_k\partial r(\boldsymbol{x}_{k+1}). \tag{24}$$

Note that $\mathbf{P}_k$ and $\boldsymbol{b}_k$ may depend on $\boldsymbol{x}_{k+1}$, but it does not hurt our conclusion: For any operator sequence $\{\boldsymbol{d}_k\}$ that satisfies (FP2) and any sequence $\{\boldsymbol{x}_k\}$ that is generated by (8) and satisfies the Global Convergence, there must exist $\{\mathbf{P}_k, \boldsymbol{b}_k\}$ such that (24) holds for all k.

Meanwhile, since $\mathbf{P}_k$ is assumed to be symmetric positive definite, function $\hat{F}_k(\boldsymbol{x}) = f(\boldsymbol{x}) + (1/2)\|\boldsymbol{x} - \boldsymbol{x}_k + \boldsymbol{b}_k\|^2_{\mathbf{P}_k^{-1}}$ is strongly convex. Therefore, $\boldsymbol{x}$ is the unique minimizer of $\hat{F}_k$ if and only if:

$$\mathbf{0} \in \partial r(\boldsymbol{x}) + \mathbf{P}_k^{-1}(\boldsymbol{x} - \boldsymbol{x}_k + \boldsymbol{b}_k).$$

With reorganization, the above condition is equivalent with

$$\boldsymbol{x}_k - \boldsymbol{b}_k \in \boldsymbol{x} + \mathbf{P}_k\partial r(\boldsymbol{x}),$$

which is exactly the condition (24) that $\boldsymbol{x}_{k+1}$ satisfies. Thus, $\boldsymbol{x}_{k+1}$ is the unique minimizer of $\hat{F}_k(\boldsymbol{x})$ and is the unique point satisfying (24), which finishes the proof. □

A.4. Proof of Theorem 3

Proof. Denote

$$\hat{\boldsymbol{d}}_k := \boldsymbol{d}_k(\boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, -\nabla f(\boldsymbol{x}_*)).$$

Plugging the above equation into (11), we obtain

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1}, \boldsymbol{g}_{k+1}) + \boldsymbol{d}_k(\boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, -\nabla f(\boldsymbol{x}_*)) - \hat{\boldsymbol{d}}_k.$$

Applying Lemma 1, we obtain

$$\begin{aligned}\boldsymbol{x}_{k+1} = \ & \boldsymbol{x}_k - \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) - \mathbf{J}_{2,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*)\\ & - \mathbf{J}_{3,k}(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)) - \mathbf{J}_{4,k}(\boldsymbol{g}_{k+1} + \nabla f(\boldsymbol{x}_*)) - \hat{\boldsymbol{d}}_k\end{aligned}$$

for some $\mathbf{J}_{1,k}, \mathbf{J}_{2,k}, \mathbf{J}_{3,k}, \mathbf{J}_{4,k} \in \mathbb{R}^{n\times n}$ that satisfy

$$\|\mathbf{J}_{j,k}\| \le \sqrt{n}C, \quad \forall j = 1,2,3,4.$$

Reorganizing the above equation, we have

$$\begin{aligned}\boldsymbol{x}_{k+1} = \ & \boldsymbol{x}_k - \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) - \mathbf{J}_{2,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) \\ & - (\mathbf{J}_{3,k} - \mathbf{J}_{4,k})(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)) - \mathbf{J}_{4,k}(\boldsymbol{g}_{k+1} + \nabla f(\boldsymbol{x}_k)) - \hat{\boldsymbol{d}}_k.\end{aligned}$$

With

$$\begin{aligned}&\mathbf{P}_k := \mathbf{J}_{4,k}, \\ &\boldsymbol{b}_k := \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) + \mathbf{J}_{2,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) + (\mathbf{J}_{3,k} - \mathbf{J}_{4,k})(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)) + \hat{\boldsymbol{d}}_k,\end{aligned}$$

we have

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{P}_k(\nabla f(\boldsymbol{x}_k) + \boldsymbol{g}_{k+1}) - \boldsymbol{b}_k, \quad \boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1}),$$

and

$$\begin{aligned}&\|\mathbf{P}_k\| \le \sqrt{n}C, \\ &\|\boldsymbol{b}_k\| \le \sqrt{n}C\|\boldsymbol{x}_k - \boldsymbol{x}_*\| + \sqrt{n}C\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\| + 2\sqrt{n}C\|\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)\| + \|\hat{\boldsymbol{d}}_k\|.\end{aligned}$$

The smoothness of f implies $\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*) \to \mathbf{0}$. Consequently, we conclude with $\boldsymbol{b}_k \to \mathbf{0}$ and this finishes the proof of the first part in Theorem 3.

Now it is enough to prove that the following inclusion equation of $\boldsymbol{x}$ has a unique solution and is equivalent with (12).

$$\boldsymbol{x} = \boldsymbol{x}_k - \mathbf{P}_k(\nabla f(\boldsymbol{x}_k) + \boldsymbol{g}) - \boldsymbol{b}_k, \quad \boldsymbol{g} \in \partial r(\boldsymbol{x}). \tag{25}$$

The above equation is equivalent with

$$\boldsymbol{x} \in \boldsymbol{x}_k - \mathbf{P}_k(\nabla f(\boldsymbol{x}_k) + \partial r(\boldsymbol{x})) - \boldsymbol{b}_k.$$

Since $\mathbf{P}_k$ is assumed to be symmetric positive definite, one could obtain another equivalent form with reorganization:

$$\mathbf{0} \in \partial r(\boldsymbol{x}) + \mathbf{P}_k^{-1}(\boldsymbol{x} - \boldsymbol{x}_k + \mathbf{P}_k\nabla f(\boldsymbol{x}_k) + \boldsymbol{b}_k).$$

Thanks to $f \in \mathcal{F}_L(\mathbb{R}^n)$ and $r \in \mathcal{F}(\mathbb{R}^n)$, the above equation has an unique solution $\boldsymbol{x}^+$ that yields

$$\boldsymbol{x}^+ = \arg\min_{\boldsymbol{x}} r(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}_k + \mathbf{P}_k\nabla f(\boldsymbol{x}_k) + \boldsymbol{b}_k\|^2_{\mathbf{P}_k^{-1}} = \operatorname{prox}_{r,\mathbf{P}_k}(\boldsymbol{x}_k - \mathbf{P}_k\nabla f(\boldsymbol{x}_k) - \boldsymbol{b}_k).$$

Since $\boldsymbol{x}_{k+1}$ satisfies (25), we conclude that $\boldsymbol{x}_{k+1} = \boldsymbol{x}^+$ and finish the whole proof. □

A.5. Proof of Theorem 4

Step 1: Analyzing (14). Denote

$$\hat{\boldsymbol{d}}_k := \boldsymbol{d}_k(\boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, -\nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*)).$$

Then (14) can be written as

$$\begin{aligned}\boldsymbol{x}_{k+1} = \boldsymbol{x}_k & - \boldsymbol{d}_k(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1}, \boldsymbol{g}_{k+1}, \boldsymbol{y}_k, \nabla f(\boldsymbol{y}_k)) \\ & + \boldsymbol{d}_k(\boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, -\nabla f(\boldsymbol{x}_*), \boldsymbol{x}_*, \nabla f(\boldsymbol{x}_*)) - \hat{\boldsymbol{d}}_k,\end{aligned}$$

where $\boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1})$. Applying Lemma 1, we have

$$\begin{aligned}\boldsymbol{x}_{k+1} = \boldsymbol{x}_k & - \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) - \mathbf{J}_{2,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) - \mathbf{J}_{3,k}(\boldsymbol{y}_k - \boldsymbol{x}_*) - \hat{\boldsymbol{d}}_k \\ & - \mathbf{J}_{4,k}(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)) - \mathbf{J}_{5,k}(\boldsymbol{g}_{k+1} + \nabla f(\boldsymbol{x}_*)) - \mathbf{J}_{6,k}(\nabla f(\boldsymbol{y}_k) - \nabla f(\boldsymbol{x}_*)),\end{aligned}$$

where matrices $\mathbf{J}_{j,k}(1 \le j \le 6)$ satisfy

$$\|\mathbf{J}_{j,k}\| \le \sqrt{n}C, \quad \forall j = 1,2,3,4,5,6.$$

Then we do some calculations and get

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) - \mathbf{J}_{2,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) - \mathbf{J}_{3,k}(\boldsymbol{y}_k - \boldsymbol{x}_*) - \hat{\boldsymbol{d}}_k \\ &\quad - (\mathbf{J}_{4,k} - \mathbf{J}_{5,k} + \mathbf{J}_{6,k})(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)) - (\mathbf{J}_{5,k} - \mathbf{J}_{6,k})(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)) \\ &\quad - \mathbf{J}_{5,k}(\boldsymbol{g}_{k+1} + \nabla f(\boldsymbol{x}_*)) - \mathbf{J}_{6,k}(\nabla f(\boldsymbol{y}_k) - \nabla f(\boldsymbol{x}_*)) \\ &= \boldsymbol{x}_k - \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) - \mathbf{J}_{2,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) - \mathbf{J}_{3,k}(\boldsymbol{y}_k - \boldsymbol{x}_*) - \hat{\boldsymbol{d}}_k \\ &\quad - (\mathbf{J}_{4,k} - \mathbf{J}_{5,k} + \mathbf{J}_{6,k})(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)) \\ &\quad - (\mathbf{J}_{5,k} - \mathbf{J}_{6,k})\nabla f(\boldsymbol{x}_k) - \mathbf{J}_{5,k}\, \boldsymbol{g}_{k+1} - \mathbf{J}_{6,k}\nabla f(\boldsymbol{y}_k).\end{aligned}$$

Given any $\mathbf{B}_k \in \mathbb{R}^{n\times n}$, we define:

$$\begin{aligned}&\mathbf{P}_{1,k} := \mathbf{J}_{5,k}, \\ &\mathbf{P}_{2,k} := \mathbf{J}_{6,k}, \\ &\boldsymbol{b}_{1,k} := \mathbf{J}_{1,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) + \mathbf{J}_{2,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) + \mathbf{J}_{3,k}(\boldsymbol{y}_k - \boldsymbol{x}_*) + \hat{\boldsymbol{d}}_k \\ &\qquad + (\mathbf{J}_{4,k} - \mathbf{J}_{5,k} + \mathbf{J}_{6,k})(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)) + \mathbf{B}_k(\boldsymbol{y}_k - \boldsymbol{x}_k).\end{aligned}$$

Then we have

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\mathbf{P}_{1,k} - \mathbf{P}_{2,k})\nabla f(\boldsymbol{x}_k) - \mathbf{P}_{2,k}\nabla f(\boldsymbol{y}_k) - \mathbf{P}_{1,k}\, \boldsymbol{g}_{k+1} + \mathbf{B}_k(\boldsymbol{y}_k - \boldsymbol{x}_k) - \boldsymbol{b}_{1,k},$$

which immediately leads to (15). The upper bounds of $\mathbf{J}_{j,k}(1 \le j \le 6)$ imply that $\mathbf{P}_{1,k}, \mathbf{P}_{2,k}$ are bounded:

$$\|\mathbf{P}_{1,k}\| \le \sqrt{n}C, \quad \|\mathbf{P}_{2,k}\| \le \sqrt{n}C,$$

and $\boldsymbol{b}_{1,k}$ is controlled by

$$\begin{aligned}\|\boldsymbol{b}_{1,k}\| &\le \sqrt{n}C\Big(\|\boldsymbol{x}_k - \boldsymbol{x}_*\| + \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\| + \|\boldsymbol{y}_k - \boldsymbol{x}_*\|\Big) + \|\hat{\boldsymbol{d}}_k\| \\ &\quad + 3\sqrt{n}C\|\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)\| + \|\mathbf{B}_k\|\|\boldsymbol{y}_k - \boldsymbol{x}_k\|.\end{aligned}$$

Assumption (GC4) implies that

$$\|\boldsymbol{x}_k - \boldsymbol{x}_*\| \to 0, \quad \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\| \to 0, \quad \|\boldsymbol{y}_k - \boldsymbol{x}_*\| \to 0, \quad \|\boldsymbol{y}_k - \boldsymbol{x}_k\| \to 0,$$

and Assumption (FP4) implies that $\|\hat{\boldsymbol{d}}_k\| \to 0$. The smoothness of f implies $\|\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)\| \to 0$. In the theorem statement, we assume that $\{\mathbf{B}_k\}$ could be any bounded matrix sequence. Therefore, it concludes that $\|\boldsymbol{b}_{1,k}\| \to 0$ as $k \to \infty$.

Step 2: Analyzing (13). Since $T = 1$, equation (13) reduces to

$$\boldsymbol{y}_{k+1} = \boldsymbol{m}(\boldsymbol{x}_{k+1}, \boldsymbol{x}_k).$$

Due to Assumption (FP4), $\boldsymbol{x}_* = \boldsymbol{m}(\boldsymbol{x}_*, \boldsymbol{x}_*)$, equation (13) is equivalent to

$$\boldsymbol{y}_{k+1} = \boldsymbol{m}(\boldsymbol{x}_{k+1}, \boldsymbol{x}_k) - \boldsymbol{m}(\boldsymbol{x}_*, \boldsymbol{x}_*) + \boldsymbol{x}_*.$$

Then one could apply Lemma 1 and obtain

$$\boldsymbol{y}_{k+1} = \mathbf{J}_{7,k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) + \mathbf{J}_{8,k}(\boldsymbol{x}_k - \boldsymbol{x}_*) + \boldsymbol{x}_*,$$

where matrices $\mathbf{J}_{7,k}$ and $\mathbf{J}_{8,k}$ satisfy

$$\|\mathbf{J}_{7,k}\| \le \sqrt{n}C, \quad \|\mathbf{J}_{8,k}\| \le \sqrt{n}C.$$

With calculation, we get

$$\begin{aligned}\boldsymbol{y}_{k+1} &= \mathbf{J}_{7,k}\boldsymbol{x}_{k+1} + \mathbf{J}_{8,k}\boldsymbol{x}_k + (\mathbf{I} - \mathbf{J}_{7,k} - \mathbf{J}_{8,k})\boldsymbol{x}_* \\ &= (\mathbf{I} - \mathbf{J}_{8,k})\boldsymbol{x}_{k+1} + \mathbf{J}_{8,k}\boldsymbol{x}_k + (\mathbf{I} - \mathbf{J}_{7,k} - \mathbf{J}_{8,k})(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*)\end{aligned}$$

With

$$\mathbf{A}_k := \mathbf{J}_{8,k}, \quad \boldsymbol{b}_{2,k} := (\mathbf{I} - \mathbf{J}_{7,k} - \mathbf{J}_{8,k})(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*),$$

one can immediately obtain (16) the following bounds

$$\begin{aligned}&\|\mathbf{A}_k\| \le \sqrt{n}C \\ &\|\boldsymbol{b}_{2,k}\| \le \|\mathbf{I} - \mathbf{J}_{7,k} - \mathbf{J}_{8,k}\|\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\| \le \big(1 + 2\sqrt{n}C\big)\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\| \to 0.\end{aligned}$$

Step 3: Proof of (17). To prove (17), we assume sequence $\{\mathbf{P}_{1,k}\}$ is uniformly symmetric positive definite, i.e., the smallest eigenvalues of symmetric positive definite $\{\mathbf{P}_{1,k}\}$ are uniformly bounded away from zero. Thus, the matrix sequence $\mathbf{P}_{1,k}^{-1}\mathbf{P}_{2,k}$ is bounded. Since equation (15) holds for all bounded matrix sequence $\mathbf{B}_k$, we let

$$\mathbf{B}_k := \mathbf{P}_{2,k}\mathbf{P}_{1,k}^{-1},$$

and obtain

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\mathbf{P}_{1,k} - \mathbf{P}_{2,k})\nabla f(\boldsymbol{x}_k) - \mathbf{P}_{2,k}\nabla f(\boldsymbol{y}_k) - \mathbf{P}_{1,k}\boldsymbol{g}_{k+1} + \mathbf{P}_{2,k}\mathbf{P}_{1,k}^{-1}(\boldsymbol{y}_k - \boldsymbol{x}_k) - \boldsymbol{b}_{1,k}.$$

Therefore, it holds that

$$\begin{aligned}
\boldsymbol{x}_{k+1} + \mathbf{P}_{1,k}\boldsymbol{g}_{k+1} &= \boldsymbol{x}_k - (\mathbf{P}_{1,k} - \mathbf{P}_{2,k})\nabla f(\boldsymbol{x}_k) - \mathbf{P}_{2,k}\nabla f(\boldsymbol{y}_k) + \mathbf{P}_{1,k}^{-1}\mathbf{P}_{2,k}(\boldsymbol{y}_k - \boldsymbol{x}_k) - \boldsymbol{b}_{1,k} && (26)\\
&= \Big(\mathbf{I} - \mathbf{P}_{2,k}\mathbf{P}_{1,k}^{-1}\Big)\Big(\boldsymbol{x}_k - \mathbf{P}_{1,k}\nabla f(\boldsymbol{x}_k)\Big) + \mathbf{P}_{2,k}\mathbf{P}_{1,k}^{-1}\Big(\boldsymbol{y}_k - \mathbf{P}_{1,k}\nabla f(\boldsymbol{y}_k)\Big) - \boldsymbol{b}_{1,k} && (27)\\
&= \Big(\mathbf{I} - \mathbf{P}_{2,k}\mathbf{P}_{1,k}^{-1}\Big)\hat{\boldsymbol{x}}_k + \mathbf{P}_{2,k}\mathbf{P}_{1,k}^{-1}\hat{\boldsymbol{y}}_k - \boldsymbol{b}_{1,k} && (28)\\
&= \Big(\mathbf{I} - \mathbf{B}_k\Big)\hat{\boldsymbol{x}}_k + \mathbf{B}_k\hat{\boldsymbol{y}}_k - \boldsymbol{b}_{1,k}. && (29)
\end{aligned}$$

Since $\boldsymbol{g}_{k+1} \in \partial r(\boldsymbol{x}_{k+1})$, we have $\boldsymbol{x}_{k+1}$ yields the following inclusion equation

$$\mathbf{0} \in \partial r(\boldsymbol{x}) + \mathbf{P}_{1,k}^{-1}\left(\boldsymbol{x} - \Big(\mathbf{I} - \mathbf{B}_k\Big)\hat{\boldsymbol{x}}_k - \mathbf{B}_k\hat{\boldsymbol{y}}_k + \boldsymbol{b}_{1,k}\right).$$

Consequently, $\boldsymbol{x}_{k+1}$ is the unique minimizer of the following convex optimization problem

$$\min_{\boldsymbol{x}} r(\boldsymbol{x}) + \frac{1}{2}\left\|\boldsymbol{x} - \Big(\mathbf{I} - \mathbf{B}_k\Big)\hat{\boldsymbol{x}}_k - \mathbf{B}_k\hat{\boldsymbol{y}}_k + \boldsymbol{b}_{1,k}\right\|_{\mathbf{P}_{1,k}^{-1}}^2. \tag{30}$$

Applying the definition of preconditioned proximal operator (10), one could immediately get (17). The strong convexity of (30) implies that (30) admits a unique minimizer, which concludes the uniqueness of $\boldsymbol{x}_{k+1}$ and finishes the whole proof.

B. Other Theoretical Results

In this section, we study the explicit update rule (7) in the non-smooth case. To facilitate reading, we rewrite (7) here:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}_k(\boldsymbol{x}_k, \boldsymbol{g}_k), \quad \boldsymbol{g}_k \in \partial r(\boldsymbol{x}_k). \tag{31}$$

We show that, even for some simple functions, one may not expect to obtain an efficient update rule if $\boldsymbol{d}_k \in \mathcal{D}_C(\mathbb{R}^n \times \mathbb{R}^n)$. The one-dimensional case is presented in Proposition 1 and the n-dimensional case is presented in Proposition 2.

In the one-dimensional case, we consider function $r(x) = |x|$. It has unique minimizer $x_* = 0$. Its subdifferential is:

$$\partial r(x) = \begin{cases} \operatorname{sign}(x), & x \neq 0; \\ [-1, 1], & x = 0. \end{cases} \tag{32}$$

Since $x_* = 0$, the asymptotic fixed point condition is $d_k(0, 0) \to 0$. Furthermore, we assume all sequences generated by (31) with initial points $x_0 \in [-1, 1]$ converges to 0 *uniformly*. In another word, there is a uniform convergence rate for all possible sequences. Due to the uniqueness of minimizer, one may expect a good update rule satisfy such uniform convergence.

Proposition 1. *Consider 1-D function $r(x) = |x|$. Suppose we pick d_k from $\mathcal{D}_C(\mathbb{R})$ and form a operator sequence $\{d_k\}_{k=0}^{\infty}$. If we assume:*

- *It holds that $d_k(0, 0) \to 0$ as $k \to \infty$.*
- *Any sequences $\{x_k\}$ generated by (31) converges to 0* uniformly *for all initial points $x_0 \in [-1, 1]$.*

then there exist $\{p_k, b_k\}_{k=0}^{\infty}$ satisfying

$$d_k(x_k, g_k) = p_k g_k + b_k, \quad g_k \in \partial r(x_k), \quad \textit{for all } k = 0, 1, 2, \cdots,$$

$p_k \to 0$, and $b_k \to 0$ as $k \to \infty$.

This proposition demonstrates that if $r(x) = |x|$, any update rule in the form of (31) actually equals to subgradient descent method with adaptive step size p_k and bias b_k. The step size p_k must be diminishing, otherwise, the uniform convergence would be broken. Diminishing step size usually leads to a slower convergence rate than constant step size. Thus, one may not expect to obtain an efficient update rule in this case.

In the n-dimensional case, we consider a family of n-dim function

$$\mathcal{F}_{\ell_1}(\mathbb{R}^n) = \{\|\mathbf{A}\boldsymbol{x}\|_1 : \mathbf{A} \in \mathbb{R}^{n\times n}, \|\mathbf{A}\| \leq 1, \text{and } \mathbf{A} \text{ is non-singular}\}.$$

All functions in $\mathcal{F}_{\ell_1}(\mathbb{R}^n)$ have a unique minimizer $\boldsymbol{x}_* = \mathbf{0}$. Its subdifferential is defined as

$$\partial r(\boldsymbol{x}) = \mathbf{A}^\top \partial \|\mathbf{A}\boldsymbol{x}\|_1.$$

If every element of $\mathbf{A}\boldsymbol{x}$ is non-zero, it holds that

$$\partial r(\boldsymbol{x}) = \mathbf{A}^\top \text{sign}(\mathbf{A}\boldsymbol{x}). \tag{33}$$

As an extension to Proposition 1, the asymptotic fixed point condition becomes $d_k(\mathbf{0}, \mathbf{0}) \to \mathbf{0}$. We also assume that all sequences generated by (31) converge to $\mathbf{0}$ *uniformly* for all possible functions $r(\boldsymbol{x}) \in \mathcal{F}_{\ell_1}(\mathbb{R}^n)$ due to these function share the same minimizer.

Proposition 2. *Suppose we pick $\boldsymbol{d}_k$ from $\mathcal{D}_C(\mathbb{R}^n \times \mathbb{R}^n)$ and form an operator sequence $\{\boldsymbol{d}_k\}_{k=0}^\infty$. If we assume:*

- *It holds that $d_k(\mathbf{0}, \mathbf{0}) \to \mathbf{0}$.*
- *Any sequences $\{\boldsymbol{x}_k\}$ generated by (31) converges to $\mathbf{0}$* uniformly *for all $r(\boldsymbol{x}) \in \mathcal{F}_{\ell_1}(\mathbb{R}^n)$ and all initial points $\boldsymbol{x}_0 \in [-1, 1]^n$.*

then there exist $\{\mathbf{P}_k, \boldsymbol{b}_k\}_{k=0}^\infty$ satisfying

$$\boldsymbol{d}_k(\boldsymbol{x}_k, \boldsymbol{g}_k) = \mathbf{P}_k \boldsymbol{g}_k + \boldsymbol{b}_k, \quad \boldsymbol{g}_k \in \partial r(\boldsymbol{x}_k), \quad \textit{for all } k = 0, 1, 2, \cdots,$$

where $\mathbf{P}_k \to \mathbf{0}$ and $\boldsymbol{b}_k \to \mathbf{0}$ as $k \to \infty$.

The conclusion is similar to Proposition 1. The preconditioner $\mathbf{P}_k$ goes smaller and smaller as $k \to \infty$, which means the update step size should be diminishing. The convergence rate gets slower and slower as k increases. Thus, the explicit update rule (31) is not efficient.

B.1. Proof of Proposition 1

Proof. Following the same proof line with that of Theorem 1, we can get the conclusion: for any sequence $\{d_k\}_{k=0}^\infty$ satisfying the conditions described in Proposition 1, there exists a sequence $\{p_k, b_k\}_{k=0}^\infty$ such that

$$x_{k+1} = x_k - p_k g_k - b_k,$$

where $g_k \in \partial r(x_k)$ and $|p_k| \leq C$ and $b_k \to 0$.

Then we want to show that, as long as all sequences $\{x_k\}$ generated by (7) *uniformly* converges to x_*, it must hold that $p_k \to 0$. We show this by contradiction and assume p_k does not converge to zero. In another word, there exist a fixed real number $\varepsilon > 0$ and a sub-sequence of $\{p_k\}$ such that

$$|p_{k_l}| > \varepsilon, \quad l = 1, 2, \cdots.$$

Now we claim that: given $\{p_k, b_k\}_{k=0}^\infty$, for any $\hat{k} > 0$, there exits an initial point x_0 such that $x_k \neq 0$ for all $k \leq \hat{k}$. The proof is as follows:

- Given $x_0 \neq 0$, we have $g_0 = 1$ or $g_0 = -1$ due to (32).
- To guarantee $x_1 \neq 0$, it's enough that $x_0 + p_0 - b_0 \neq 0, x_0 + p_0 - b_0 \neq 0$.

- Define
$$\mathcal{X}_1 := \{b_0 + p_0, b_0 - p_0\}.$$
As long as $x_0 \notin \mathcal{X}_0 \cup \mathcal{X}_1$, we can guarantee $x_0 \neq 0$ and $x_1 \neq 0$.
- Define
$$\mathcal{X}_2 := \{b_0 + p_0 + b_1 + p_1, b_0 + p_0 + b_1 - p_1, b_0 - p_0 + b_1 + p_1, b_0 - p_0 + b_1 - p_1\}.$$
As long as $x_0 \notin \mathcal{X}_0 \cup \mathcal{X}_1 \cup \mathcal{X}_2$, we can guarantee $x_0 \neq 0$ and $x_1 \neq 0$ and $x_2 \neq 0$.
- ...

Repeat the above statement for $\hat{k}$ times, we obtain: $x_0 \notin \bigcup_{k \leq \hat{k}} \mathcal{X}_k$ implies $x_k \neq 0$ for all $k \leq \hat{k}$, where the set $\mathcal{X}_k$ contains 2^k elements. Thus, x_0 can be chosen almost freely within $[-1, 1]$ excluding a set with a finite number of elements. The claim is proven.

With $\hat{k} = k_l$, we conclude that, for all $l = 1, 2, \cdots$, there exists an initial point x_0 such that $x_k \neq 0$ for all $k \leq k_l$. Consequently, it holds that $g_{k_l} = 1$ or $g_{k_l} = -1$. Then,
$$|x_{k_l+1} - x_{k_l}| \geq |p_{k_l} g_{k_l}| - |b_{k_l}| = \varepsilon - |b_{k_l}|,$$
which contradicts with the fact that $b_k \to 0$ and $x_k \to 0$ uniformly for all initial points. This completes the proof for $p_k \to 0$. □

B.2. Proof of Proposition 2

Proof. The proof of Proposition 2 extends the proof of Proposition 1 and follows a similar proof sketch. But the n-dim case is much more complicated than the 1-dim case. Consequently, we need stronger assumptions: $\boldsymbol{x}_k$ converges uniformly not only for all initial points, but also for a family of objective function $f \in \mathcal{F}_{\ell_1}(\mathbb{R}^n)$.

In our proof, we denote $(\mathbf{A}\boldsymbol{x})^i$ as the i-th element of vector $\mathbf{A}\boldsymbol{x}$ and the index of a matrix is denoted by $(:,:)$. For example, $\mathbf{A}(i,:)$ means the i-th row of $\mathbf{A}$; $\mathbf{A}(:,i)$ means the i-th column of $\mathbf{A}$.

Following the same proof line with that of Theorem 1, we can get the conclusion (similar with Proposition 1): for any sequence $\{\boldsymbol{d}_k\}_{k=0}^{\infty}$ satisfying the conditions described in Proposition 2, there exists a sequence $\{\mathbf{P}_k, \boldsymbol{b}_k\}_{k=0}^{\infty}$ such that
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{P}_k \boldsymbol{g}_k - \boldsymbol{b}_k,$$
where $\boldsymbol{g}_k \in \partial r(\boldsymbol{x}_k)$ and $\|\mathbf{P}_k\| \leq \sqrt{n}C$ and $\boldsymbol{b}_k \to \mathbf{0}$. It's enough to show that $\mathbf{P}_k \to \mathbf{0}$.

Before proving $\mathbf{P}_k \to \mathbf{0}$, we first claim and prove an statement: given $\{\mathbf{P}_k, \boldsymbol{b}_k\}_{k=0}^{\infty}$ and any $r(\boldsymbol{x}) = \|\mathbf{A}\boldsymbol{x}\|_1 \in \mathcal{F}_{\ell_1}(\mathbb{R}^n)$ and any $\hat{k} > 0$, there exits an initial point $\boldsymbol{x}_0$ such that $(\mathbf{A}\boldsymbol{x}_k)^i \neq 0$ for all $k \leq \hat{k}$ and $i = 1, 2, \cdots, n$. The proof is as follows:

- To guarantee $(\mathbf{A}\boldsymbol{x}_0)^i \neq 0$, $\boldsymbol{x}_0$ must satisfy:
$$\boldsymbol{x}_0 \notin \mathcal{X}_0^i = \{\boldsymbol{x} : \mathbf{A}(i,:)\boldsymbol{x} = 0\}.$$
- Given $(\mathbf{A}\boldsymbol{x}_0)^i \neq 0, 1 \leq i \leq n$, we have $\boldsymbol{g}_0 = \mathbf{A}^\top \mathrm{sign}(\mathbf{A}\boldsymbol{x}_0)$, where $\mathrm{sign}(\mathbf{A}\boldsymbol{x}_0) \in \{1, -1\}^n$, due to (33).
- To guarantee $(\mathbf{A}\boldsymbol{x}_1)^i \neq 0$, it's enough that $\mathbf{A}(i,:)(\boldsymbol{x}_0 - \mathbf{P}_0\mathbf{A}^\top \boldsymbol{s} - \boldsymbol{b}_0) \neq 0$ for all $\boldsymbol{s} \in \{1, -1\}^n$.
- Define
$$\mathcal{X}_1^i = \{\boldsymbol{x} : \mathbf{A}(i,:)(\boldsymbol{x} - \mathbf{P}_0\mathbf{A}^\top \boldsymbol{s}_0 - \boldsymbol{b}_0) = 0 \text{ for some } \boldsymbol{s}_0 \in \{1, -1\}^n\}.$$
As long as $\boldsymbol{x}_0 \notin \bigcup_{1 \leq i \leq n, 0 \leq k \leq 1} \mathcal{X}_k^i$, we guarantee that $(\mathbf{A}\boldsymbol{x}_k)^i \neq 0$ for all $k \leq 1$ and $i = 1, 2, \cdots, n$.
- Define
$$\mathcal{X}_2^i = \{\boldsymbol{x} : \mathbf{A}(i,:)(\boldsymbol{x} - \mathbf{P}_0\mathbf{A}^\top \boldsymbol{s}_0 - \boldsymbol{b}_0 - \mathbf{P}_1\mathbf{A}^\top \boldsymbol{s}_1 - \boldsymbol{b}_1) = 0 \text{ for some } \boldsymbol{s}_0, \boldsymbol{s}_1 \in \{1, -1\}^n\}.$$
As long as $\boldsymbol{x}_0 \notin \bigcup_{1 \leq i \leq n, 0 \leq k \leq 2} \mathcal{X}_k^i$, we guarantee that $(\mathbf{A}\boldsymbol{x}_k)^i \neq 0$ for all $k \leq 2$ and $i = 1, 2, \cdots, n$.

- ...

Repeat the above statement for $\hat{k}$ times, we obtain: $\boldsymbol{x}_0 \notin \bigcup_{1\le i\le n, 0\le k\le \hat{k}} \mathcal{X}_k^i$ implies the conclusion we want. Moreover, the set $\mathcal{X}_k^i$ has measurement zero in the space $\mathbb{R}^n$ due to the fact that each row of matrix $\mathbf{A}$: $\mathbf{A}(i,:)$ is not zero ($\mathbf{A}$ is non-singular). Finite union of $\mathcal{X}_k^i$ also has measurement zero. Thus, $\boldsymbol{x}_0$ can be chosen almost freely in $[-1,1]^n$ excluding a set with zero measurements. The claim is proven.

Now we show $\mathbf{P}_k \to \mathbf{0}$ by contradiction. Assume there exist a fixed real number $\varepsilon > 0$ and a sub-sequence of $\{\mathbf{P}_k\}$ such that

$$\|\mathbf{P}_{k_l}\| > \varepsilon, \quad l = 1, 2, \cdots. \tag{34}$$

Conduct SVD on $\mathbf{P}_{k_l}$:

$$\mathbf{P}_{k_l} = \mathbf{U}_{k_l}\Sigma_{k_l}\mathbf{V}_{k_l}^\top = \mathbf{U}_{k_l}\begin{bmatrix}\sigma_{k_l}^1 & & & \\ & \sigma_{k_l}^2 & & \\ & & \ddots & \\ & & & \sigma_{k_l}^n\end{bmatrix}\mathbf{V}_{k_l}^\top.$$

Inequality (34) implies the largest singular value of $\mathbf{P}_{k_l}$ should be greater than ε. WLOG, we assume $\sigma_{k_l}^1$ is the largest one and, consequently, $\sigma_{k_l}^1 > \varepsilon$. Given such $\mathbf{V}_{k_l}$, we define

$$f_{k_l}(\boldsymbol{x}) = \|\mathbf{A}_{k_l}\boldsymbol{x}\|_1, \quad \mathbf{A}_{k_l} = \mathbf{V}_{k_l}^\top.$$

It's easy to check that $f_{k_l} \in \mathcal{F}_{\ell_1}(\mathbb{R}^n)$.

Given $\{\mathbf{P}_{k_l}, \boldsymbol{b}_{k_l}\}_{l=0}^\infty$ and function $f_{k_l}(\boldsymbol{x})$, we take an initial point $\boldsymbol{x}_0 \notin \bigcup_{1\le i\le n, 0\le k\le k_l} \mathcal{X}_k^i$, then we have $(\mathbf{A}_{k_l}\boldsymbol{x}_{k_l})^i \neq 0$. Thus, it holds that

$$\boldsymbol{x}_{k_l+1} = \boldsymbol{x}_{k_l} - \mathbf{P}_{k_l}\mathbf{A}_{k_l}^\top \operatorname{sign}(\mathbf{A}_{k_l}\boldsymbol{x}_{k_l}) - \boldsymbol{b}_{k_l} = \boldsymbol{x}_{k_l} - \mathbf{P}_{k_l}\mathbf{A}_{k_l}^\top \boldsymbol{s}_{k_l} - \boldsymbol{b}_{k_l}$$

for some $\boldsymbol{s}_{k_l} \in \{1,-1\}^n$. Rewrite the second term on the right-hand side

$$\mathbf{P}_{k_l}\mathbf{A}_{k_l}^\top \boldsymbol{s}_{k_l} = \mathbf{U}_{k_l}\begin{bmatrix}\sigma_{k_l}^1 & & & \\ & \sigma_{k_l}^2 & & \\ & & \ddots & \\ & & & \sigma_{k_l}^n\end{bmatrix}\boldsymbol{s}_{k_l} = \mathbf{U}_{k_l}\begin{bmatrix}\sigma_{k_l}^1 s_{k_l}^1 \\ \sigma_{k_l}^2 s_{k_l}^2 \\ \vdots \\ \sigma_{k_l}^n s_{k_l}^n\end{bmatrix}.$$

Its norm is lower bounded by

$$\left\|\mathbf{P}_{k_l}\mathbf{A}_{k_l}^\top \boldsymbol{s}_{k_l}\right\| \ge |\sigma_{k_l}^1 s_{k_l}^1| > \varepsilon.$$

Then we get

$$\|\boldsymbol{x}_{k_l+1} - \boldsymbol{x}_{k_l}\| \ge \left\|\mathbf{P}_{k_l}\mathbf{A}_{k_l}^\top \boldsymbol{s}_{k_l}\right\| - \|\boldsymbol{b}_{k_l}\|,$$

which contradicts with the fact that $\boldsymbol{b}_k \to \mathbf{0}$ and $\boldsymbol{x}_k \to \mathbf{0}$ uniformly. $\mathbf{P}_k \to \mathbf{0}$ is proved and this finishes the proof. □

C. Scheme (18) Covers Many Schemes in the Literature

In this section, we show that our proposed scheme (18) covers FISTA (Beck & Teboulle, 2009), PGD with variable metric (Park et al., 2020), Step-LISTA (Ablin et al., 2019), and Ada-LISTA (Aberdam et al., 2021). We will show them one by one. To facilitate reading, we rewrite (18) here:

$$\begin{aligned}
\hat{\boldsymbol{x}}_k &= \boldsymbol{x}_k - \boldsymbol{p}_k \odot \nabla f(\boldsymbol{x}_k), \\
\hat{\boldsymbol{y}}_k &= \boldsymbol{y}_k - \boldsymbol{p}_k \odot \nabla f(\boldsymbol{y}_k), \\
\boldsymbol{x}_{k+1} &= \operatorname{prox}_{r,\boldsymbol{p}_k}\Big((\mathbf{1} - \boldsymbol{b}_k) \odot \hat{\boldsymbol{x}}_k + \boldsymbol{b}_k \odot \hat{\boldsymbol{y}}_k - \boldsymbol{b}_{1,k}\Big), \\
\boldsymbol{y}_{k+1} &= \boldsymbol{x}_{k+1} + \boldsymbol{a}_k \odot (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \boldsymbol{b}_{2,k}.
\end{aligned}$$

FISTA. The update rule of FISTA (with constant step size) writes

$$\begin{aligned} \boldsymbol{y}_{k+1} &= \operatorname{prox}_{r,(1/L)\mathbf{1}}\Big(\boldsymbol{x}_k - \frac{1}{L}\nabla f(\boldsymbol{x}_k)\Big), \\ t_{k+1} &= \frac{1+\sqrt{1+4t_k^2}}{2}, \\ \boldsymbol{x}_{k+1} &= \boldsymbol{y}_{k+1} + \frac{t_k - 1}{t_{k+1}}(\boldsymbol{y}_{k+1} - \boldsymbol{y}_k), \end{aligned} \tag{35}$$

where L is the Lypuschitz constant of ∇f. Thus, as long as $\boldsymbol{b}_k = \mathbf{1}$, $\boldsymbol{b}_{1,k} = \boldsymbol{b}_{2,k} = \mathbf{0}$, $\boldsymbol{p}_k = (1/L)\mathbf{1}$, and $\boldsymbol{a}_k = \frac{t_k - 1}{t_{k+1}}\mathbf{1}$, (18) is equal to (35).

PGD. PGD with variable metric writes

$$\boldsymbol{x}_{k+1} = \operatorname{prox}_{r,\boldsymbol{p}_k}\Big(\boldsymbol{x}_k - \boldsymbol{p}_k \odot \nabla f(\boldsymbol{x}_k)\Big). \tag{36}$$

If $\boldsymbol{b}_k = \boldsymbol{a}_k = \boldsymbol{b}_{1,k} = \boldsymbol{b}_{2,k} = \mathbf{0}$, (18) reduces to (36).

Step-LISTA. The update rule of Step-LISTA writes

$$\boldsymbol{x}_{k+1} = \sigma\Big(\boldsymbol{x}_k - p_k \mathbf{A}^\top\big(\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\big), \theta_k\Big). \tag{37}$$

If the objective function is taken as standard LASSO and we take $\boldsymbol{p}_k = p_k\mathbf{1}$

$$F(\boldsymbol{x}) = \underbrace{\frac{1}{2}\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|^2}_{f(\boldsymbol{x})} + \underbrace{\lambda\|\boldsymbol{x}\|_1}_{r(\boldsymbol{x})},$$

then we have

$$\nabla f(\boldsymbol{x}) = \mathbf{A}^\top(\mathbf{A}\boldsymbol{x} - \boldsymbol{b}), \quad \operatorname{prox}_{r,\boldsymbol{p}_k}(\boldsymbol{x}) = \sigma(\boldsymbol{x}, \lambda p_k).$$

We want to show that, for any sequence $\{\theta_k\}_{k=0}^\infty$, there exists a sequence $\{\boldsymbol{a}_k, \boldsymbol{b}_k, \boldsymbol{b}_{1,k}, \boldsymbol{b}_{2,k}\}_{k=0}^\infty$ such that (18) equals to (37).

Proof. Take $\boldsymbol{p}_k = p_k\mathbf{1}$ and $\boldsymbol{a}_k = \boldsymbol{b}_k = \boldsymbol{b}_{2,k} = \mathbf{0}$, (18) reduces to

$$\boldsymbol{x}_{k+1} = \sigma\Big(\boldsymbol{x}_k - p_k\mathbf{A}^\top(\mathbf{A}\boldsymbol{x}_k - \boldsymbol{b}) - \boldsymbol{b}_{1,k}, \lambda p_k\Big).$$

Define

$$\hat{\boldsymbol{x}}_k = \boldsymbol{x}_k - p_k\mathbf{A}^\top(\mathbf{A}\boldsymbol{x}_k - \boldsymbol{b}).$$

If $\theta_k > \lambda p_k$, define $\boldsymbol{b}_{1,k}$ component-wisely as (Here $(\boldsymbol{b}_{1,k})_i$ means the i-th component of vector $\boldsymbol{b}_{1,k}$):

$$(\boldsymbol{b}_{1,k})_i = \operatorname{sign}\big((\hat{\boldsymbol{x}}_k)_i\big)\min(\theta_k - \lambda p_k, \big|(\hat{\boldsymbol{x}}_k)_i\big|).$$

Then one can check that

$$\sigma\big(\hat{\boldsymbol{x}} - \boldsymbol{b}_{1,k}, \lambda p_k\big) = \sigma\big(\hat{\boldsymbol{x}}, \theta_k\big), \tag{38}$$

where the role of $\boldsymbol{b}_{1,k}$ is enhancing the soft thereshold from λp_k to θ_k.

If $\theta_k < \lambda p_k$, define $\boldsymbol{b}_{1,k}$ with:

$$(\boldsymbol{b}_{1,k})_i = \begin{cases} \operatorname{sign}((\hat{\boldsymbol{x}}_k)_i)(\theta_k - \lambda p_k), & (\hat{\boldsymbol{x}}_k)_i > \theta_k \\ 0, & (\hat{\boldsymbol{x}}_k)_i \le \theta_k \end{cases},$$

then (38) also holds in this case.

With the $\{p_k, \boldsymbol{a}_k, \boldsymbol{b}_k, \boldsymbol{b}_{1,k}, \boldsymbol{b}_{2,k}\}$ defined above, it holds that (18) equals to (37), which finishes the proof. □

Ada-LISTA. The update rule of Ada-LISTA with single weight matrix writes

$$\boldsymbol{x}_{k+1} = \sigma\Big(\boldsymbol{x}_k - p_k \mathbf{A}^\top \mathbf{M}^\top \mathbf{M}\big(\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\big), \theta_k\Big). \tag{39}$$

If the objective function is taken as LASSO with a learned dictionary $\mathbf{M}$:

$$F(\boldsymbol{x}) = \underbrace{\frac{1}{2}\Big\|\mathbf{M}\big(\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\big)\Big\|^2}_{f(\boldsymbol{x})} + \underbrace{\lambda\|\boldsymbol{x}\|_1}_{r(\boldsymbol{x})},$$

and we follow the same proof line as that of Step-LISTA, then we obtain that (18) covers (39).

D. Examples of Explicit Proximal Operator

As long as one can evaluate ∇f and the proximal operator $\mathrm{prox}_{r,\boldsymbol{p}_k}$, the update rule (18) is applicable. The gradient ∇f is accessible since $f \in \mathcal{F}_L(\mathbb{R}^n)$. For a broad class of $r(\boldsymbol{x})$, the operator $\mathrm{prox}_{r,\boldsymbol{p}_k}$ has efficient explicit formula. We list some examples here and more examples can be found in (Parikh & Boyd, 2014; Park et al., 2020).

- (ℓ_1-norm) Suppose $r(\boldsymbol{x}) = \lambda\|\boldsymbol{x}\|_1$, then the proximal operator is a scaled soft-thresholding operator that is component-wisely defined as $\big(\mathrm{prox}_{r,\boldsymbol{p}_k}(\boldsymbol{x})\big)_i := \mathrm{sign}(x_i)\max(0, |x_i| - \lambda(\boldsymbol{p}_k)_i)$, for $1 \le i \le n$.
- (Non-negative constraint) Suppose $r(\boldsymbol{x}) = \iota_{\mathcal{X}}(\boldsymbol{x})$ where $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^n : x_i \ge 0, 1 \le i \le n\}$ and $\iota_{\mathcal{X}}$ is the indicator function (i.e., $\iota_{\mathcal{X}}(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \mathcal{X}$; $\iota_{\mathcal{X}}(\boldsymbol{x}) = +\infty$ otherwise), then $\mathrm{prox}_{r,\boldsymbol{p}_k}$ is component-wisely defined as $\big(\mathrm{prox}_{r,\boldsymbol{p}_k}(\boldsymbol{x})\big)_i := \max(0, x_i),\ \ 1 \le i \le n$.
- (Simplex constraint) Suppose $r(\boldsymbol{x}) = \iota_{\mathcal{X}}(\boldsymbol{x})$ where $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^n : x_i \ge 0, 1 \le i \le n; \mathbf{1}^\top\boldsymbol{x} = 1.\}$, then $\mathrm{prox}_{r,\mathbf{P}_k}$ is component-wisely defined as $\big(\mathrm{prox}_{r,\boldsymbol{p}_k}(\boldsymbol{x})\big)_i := \max(0, x_i - \xi(\boldsymbol{p}_k)_i),\ 1 \le i \le n$, where $\xi \in \mathbb{R}^+$ can be determined efficiently via bisection.

E. Details in Our Experiments

LASSO Regression. In this paragraph, we provide the details of the LASSO benchmarks used in this paper.

- (Synthetic data). Each element in $\mathbf{A} \in \mathbb{R}^{250\times 500}$ is sampled *i.i.d.* from the normal distribution, and each column of $\mathbf{A}$ is normalized to have a unit ℓ_2-norm. Then we randomly generate sparse vector $\boldsymbol{x}_* \in \mathbb{R}^{500}$. In each sparse vector, we first uniformly sample 50 out of 500 entries to be nonzero, and the value of each nonzero is sampled independently from the normal distribution. With $\mathbf{A}$ and $\boldsymbol{x}_*$, we generate $\boldsymbol{b}$ with $\boldsymbol{b} = \mathbf{A}\boldsymbol{x}_*$. Such tuple $(\mathbf{A}, \boldsymbol{b}, \lambda)$ forms an instance of LASSO, and we repeatedly generate $(\mathbf{A}, \boldsymbol{b}, \lambda)$ in the above approach. The training set includes $32,000$ independent optimization problems and the testing set includes $1,024$ independent optimization problems. We take $\lambda = 0.1$ for all synthetic LASSO instances.
- (Real data). We extract 1000 patches with size 8×8 at random positions from testing images that are randomly chosen from BSDS500 (Martin et al., 2001). Each patch is flattened to a vector in space $\mathbb{R}^{64}$ and normalized and mean-removed. Then we conduct K-SVD (Aharon et al., 2006) to obtain a dictionary $\mathbf{A} \in \mathbb{R}^{64\times 128}$. Each vector in $\mathbb{R}^{64}$ can be viewed as an instance of $\boldsymbol{b}$ in LASSO (20). With a shared matrix $\mathbf{A}$, we construct 1000 instances of LASSO and test our methods on them. We take $\lambda = 0.5$ for all real-data LASSO instances.

Logistic Regression. Given a set of training examples $\{(\boldsymbol{a}_i, b_i) \in \mathbb{R}^n \times \{0,1\}\}_{i=1}^m$, the objective function of the ℓ_1-regularized logistic regression problem is defined as

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x}) = -\frac{1}{m}\sum_{i=1}^m [b_i \log(h(\boldsymbol{a}_i^\top\boldsymbol{x})) + (1 - b_i)\log(1 - h(\boldsymbol{a}_i^\top\boldsymbol{x}))] + \lambda\|\boldsymbol{x}\|_1, \tag{40}$$

where $h(c) = 1/(1 + e^{-c})$ is the logistic function. For each logistic regression problem, we generate 1,000 feature vectors, each of which $\boldsymbol{a} \in \mathbb{R}^{50}$ is sampled *i.i.d.* from the normal distribution. Then we randomly generate a sparse vector $\boldsymbol{x}_* \in \mathbb{R}^{50}$

and uniformly sample 20 out of its 50 entries to be nonzero, and the value of each nonzero is sampled independently from the normal distribution. With $\boldsymbol{a}_i$ and $\boldsymbol{x}_*$, we generate the binary classification label b_i with $b_i = \mathbb{1}(\boldsymbol{a}_i^\top \boldsymbol{x}_* >= 0)$, where $\mathbb{1}(\cdot)$ is the indicator function. Finally, we fix $\lambda = 0.1$. Such a pair $(\{(\boldsymbol{a}_i, b_i)\}_{i=1}^m, \lambda)$ forms an instance of logistic regression with ℓ_1 regularization, and we repeatedly generate such pairs in the above approach. The training set includes $32,000$ independent optimization problems and the testing set includes $1,024$ independent optimization problems.

We evaluate L2O optimizers (trained on synthesized datasets) on two real-world datasets from the UCI Machine Learning Repository (Dua & Graff, 2017): (i) *Ionosphere* containing 351 samples of 34 features, and (ii) *Spambase* containing 4,061 samples of 57 features. The results on the Ionosphere dataset is shown in the main text Figure 6. And here we present the results on the Spambase dataset in Figure 7. Our observation is consistent: L2O-PA is superior in stability and fast convergence compared to all other baselines and is almost 20× faster than FISTA.

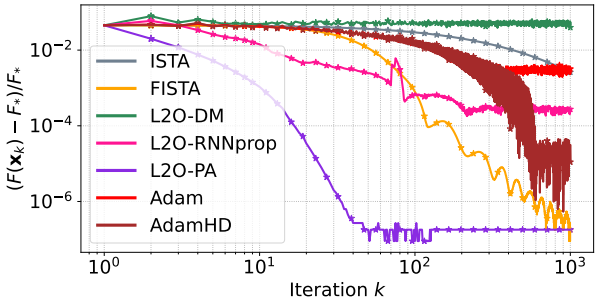

Figure 7. Logistic: Train on synthetic data and test on real data (Spambase).

F. Extra Experiments

Running Time Comparison. Considering that HPO methods such as AdamHD do not require LSTM and consume less time per iteration compared to L2O-PA, we compared the running time of our proposed method L2O-PA and AdamHD in Table 2. The experiment settings follow those in Section 4.2. In these tables, "Time/Iters" represents the average time consumed for each iteration across the 1024 testing examples. The "Iters" column indicates the number of iterations needed to achieve the specified precision, while the "Time" column denotes the time required to reach that precision. "N/A" is used when AdamHD cannot attain a precision of 10^{-6} and "Gap" means the optimality gap $(F(\boldsymbol{x}_k) - F_*)/F_*$. Table 2 clearly shows that L2O-PA requires much less time than AdamHD, even though its per-iteration complexity is higher than that of AdamHD.

Table 2. Runtime Comparison between L2O-PA and AdamHD.

		Stopping condition: Gap $<10^{-3}$		Stopping condition: Gap $<10^{-6}$	
	Time/Iters	Iters	Total Time	Iters	Total Time
LASSO (Synthetic)					
L2O-PA	2.31×10^{-2} ms	21	0.485 ms	42	0.971 ms
AdamHD	8.09×10^{-3} ms	477	3.858 ms	N/A	N/A
Logistic (Spambase)					
L2O-PA	7.845×10^{-1} ms	10	7.845 ms	33	25.89 ms
AdamHD	2.605×10^{-1} ms	390	101.6 ms	N/A	N/A

Large-Scale LASSO. To evaluate our model's performance on large problems, we generate 256 independent LASSO instances of size 2500×5000, following the same distribution described in Section E. All the learning models are trained with instances of size 250×500 and tested on these 256 large testing problems. The results, reported in Figure 8, clearly demonstrate that our proposed (L2O-PA) exhibits a superior ability to generalize to large problems.

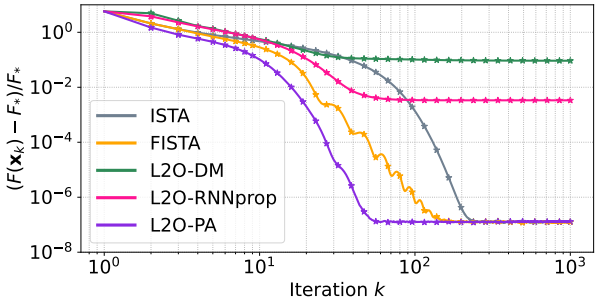

Figure 8. LASSO: Train with small instances and test on large instances.

Longer-Horizen Experiments. To test the performance of our method with longer horizons, say 10^4 iterations, we applied our proposed approach (L2O-PA) to a logistic regression problem with ℓ_1 regularization on CIFAR-10 for classification, and compared with other baselines. We randomly sampled 5000 images in the training set of CIFAR-10 (500 images from each class out of 10) for the logistic regression, which followed a similar manner as in (Cowen et al., 2019). Each image was normalized and flattened into a 3072-dim vector (i.e., $3 \times 32 \times 32$). Since the feature dimension is significantly higher than what we considered in Section 4.2, we used a much smaller regularization coefficient $\lambda = 10^{-4}$ to avoid all zero solutions.

We trained learning-based models (L2O-PA, L2O-DM and L2O-RNNprop) on a set of synthesized ℓ_1-regularized logistic regression tasks for binary classification with $\lambda = 10^{-4}$ in the same way as we did in the second part of Section 4.2 and described in Section E. Each logistic regression task contains a dataset of 1000 samples with 50 features. After training, all models are directly applied to optimizing the 10-class logistic regression on CIFAR-10 for 10^4 steps. The results, with comparisons to ISTA and FISTA, are shown in Figure 9. From the results we can see that:

- Our method, L2O-PA, converged quite stably in both near and further horizons compared to L2O-DM and L2O-RNNprop, which fluctuated wildly in later iterations. This shows the impressive generalization ability of L2O-PA considering the fact that it was trained in short-horizon settings (100 optimization steps).
- Compared to FISTA, L2O-PA can still achieve impressive acceleration in earlier iterations (25 iterations of L2O-PA comparable to FISTA at 1000+ iterations, and 300 steps vs 2500+ steps for FISTA to reach 10^{-2} relative error).

Therefore, the conclusion is that L2O-PA can still generalize well, to some extent, to longer-horizon tasks even if it is trained in short-horizon settings but it does struggle to converge fast in later iterations. We are happy to include this discussion in the main text as limitations and improve in this direction in the future.

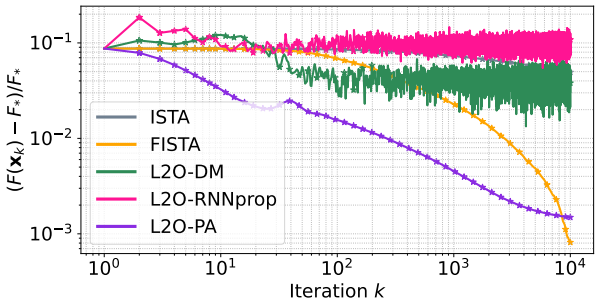

Figure 9. Logistic: Train on synthetic data and test on real data (CIFAR-10).

More-Challenging OOD Experiment. To further test the generalization performance of our method, we conduct an even more challenging OOD experiment. We directly tested learned optimizers, which were trained on synthetic LASSO problems, on synthetic ℓ_1-regularized Logistic Regression. This setting renders changes in the objective function and thus the structure of the loss function. The results are shown in Figure 10. We can see that **L2O-PA-LASSO**, the model that was trained with LASSO problems, is able to converge stably at a faster speed than that of FISTA and other L2O competitors

(except for L2O-PA) on Logistic Regression. It is worth noting that all other L2O competitors are trained directly on Logistic Regression.

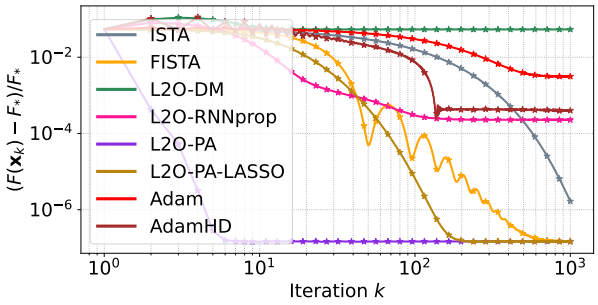

Figure 10. Logistic: Test on synthetic data.

Platform. All the experiments are conducted on a workstation equipped with four NVIDIA RTX A6000 GPUs. We used PyTorch 1.12 and CUDA 11.3.