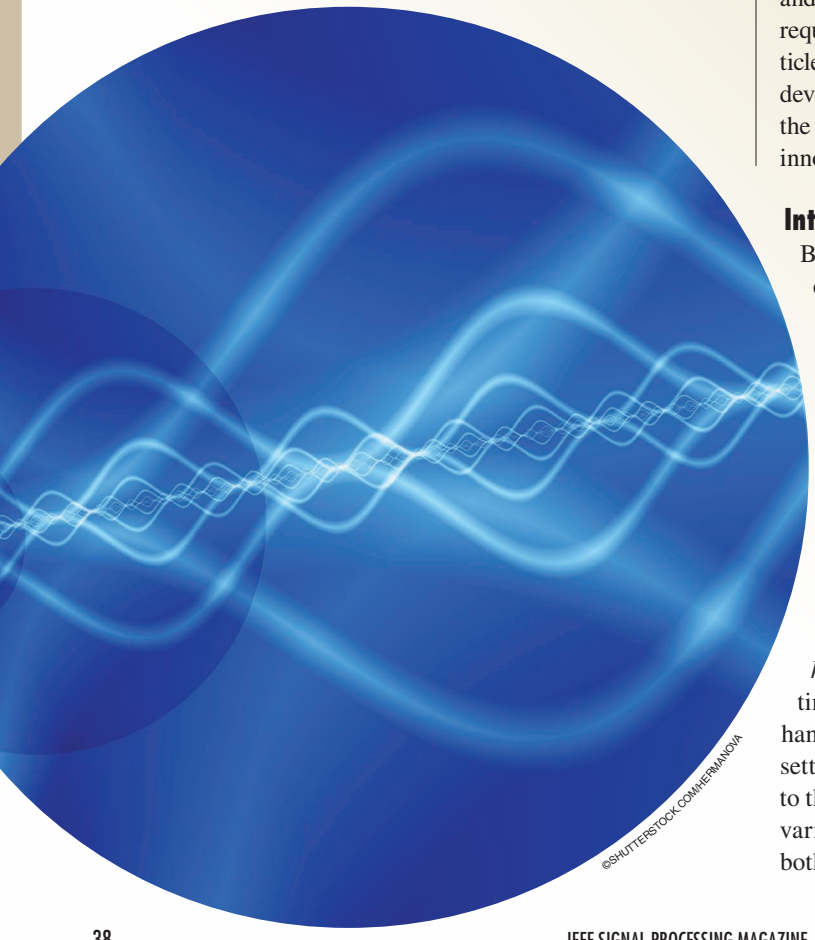
Yihua Zhang, Prashant Khanduri, Ioannis Tsaknakis, Yuguang Yao, Mingyi Hong, and Sijia Liu

An Introduction to Bilevel Optimization

Foundations and applications in signal processing and machine learning

Recently, bilevel optimization (BLO) has taken center stage in some very exciting developments in the area of signal processing (SP) and machine learning (ML). Roughly speaking, BLO is a classical optimization problem that involves two levels of hierarchy (i.e., upper and lower levels), wherein obtaining the solution to the upper-level problem requires solving the lower-level one. BLO has become popular largely because it is powerful in modeling problems in SP and ML, among others, that involve optimizing nested objective functions. Prominent applications of BLO range from resource allocation for wireless systems to adversarial ML. In this work, we focus on a class of tractable BLO problems that often appear in SP and ML applications. We provide an overview of some basic concepts of this class of BLO problems, such as their optimality conditions, standard algorithms (including their optimization principles and practical implementations) as well as how they can be leveraged to obtain state-of-the-art results for several key SP and ML applications. Further, we discuss some recent advances in BLO theory and its implications for applications, and we point out some limitations of the state of the art that require significant future research efforts. We hope that this article, together with the associated open source BLO toolbox we developed for algorithm benchmarking, can serve to accelerate the adoption of BLO as a generic tool to model, analyze, and innovate on a wide array of emerging SP and ML applications.

Digital Object Identifier 10.1109/MSP.2024.3358284
Date of current version: 15 April 2024



Introduction

BLO is a class of optimization problems involving two nested levels (*upper* and *lower levels*), where the objective and variables of the *upper-level* problem depend on the optimizer of the *lower-level* one. The canonical formulation of the BLO is given by

$$\overbrace{\underset{\boldsymbol{\theta}\in\mathcal{U}}{\text{minimize}}\ F(\boldsymbol{\theta}) \coloneqq f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))}^{\text{Upper-level optimization over } \boldsymbol{\theta}},$$
$$\text{subject to}\ \underbrace{\boldsymbol{\phi}^*(\boldsymbol{\theta}) \in \underset{h(\boldsymbol{\theta},\boldsymbol{\phi})\le 0}{\operatorname{argmin}}\, g(\boldsymbol{\theta}, \boldsymbol{\phi})}_{\text{Lower-level optimization over } \boldsymbol{\phi}}, \qquad \text{(BLO)}$$

where for analytical tractability we assume that f, g, and h are bivariate *smooth* functions; we note that in some settings f and/or g may be nonsmooth, and such settings can be handled by specialized algorithms based on their particular settings [1]; $\boldsymbol{\theta} \in \mathbb{R}^m$ denotes the upper-level variable subject to the upper-level constraint set $\mathcal{U}$; $\boldsymbol{\phi} \in \mathbb{R}^n$ is the lower-level variable subject to the constraint $h(\boldsymbol{\theta}, \boldsymbol{\phi}) \le 0$ that couples both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$; and $\boldsymbol{\phi}^*(\boldsymbol{\theta})$ is one lower-level optimal solution.

It is evident that the lower-level problem is an *auxiliary* problem since its solution supports the upper-level problem in finding a better solution.

The study of BLO can be traced to that of Stackelberg games [2], where the upper (respectively, lower) problem optimizes the action taken by a leader (respectively, the follower). Early works in optimization formulate BLO to solve resource allocation problems; see [3] for a comprehensive survey of BLO algorithms in the late 1990s and early 2000s and also some more recent surveys on discrete BLO [4], BLO under uncertainty [5], and nonlinear and nonconvex aspects of BLO [6]. In recent years, BLO has regained popularity because a subclass of BLO has been used to formulate and solve various challenging problems in SP, ML, and artificial intelligence. Notable applications in SP include resource management [7], signal demodulation [8], and image denoising and reconstruction [9]. In addition, BLO has been used to make ML models, especially deep neural networks (DNNs), robust [10], [11], [12], [13], generalizable [14], [15], [16], efficient [17], [18], [19], easier to train [20], [21], [22], [23], [24], and scalable [25].

BLO has become popular largely because it is powerful in modeling problems in SP and ML, among others, that involve optimizing nested objective functions.

As can be easily imagined, the popularity of BLO in the aforementioned applications is largely attributed to its ability to handle (often implicit) *hierarchical structures*. To better illustrate the challenges brought by the hierarchical architecture, see the example application of *coreset selection* for model training [7], [17] in "Motivating Application: Coreset Selection for Model Training."

Clearly, the coreset selection is a typical BLO problem, where the upper-level and lower-level tasks are intertwined: without knowing the training result, it is hard to gauge how representative the selected dataset is effectively, while without having the coreset, one cannot perform model training to evaluate the utility of the learned model further. More importantly, these two tasks form a *hierarchy*, with the model training problem being the main optimization problem, while the data selection problem is an *auxiliary* problem that supports the training.

Given the growing interest in BLO, in this work we present an overview of a class of *tractable* (BLO) problems that hold significant importance in SP and ML. Roughly speaking, the class of BLO problems we consider has some desirable properties (to be discussed shortly) that allow the development of efficient and practical algorithms. We will discuss the basic concepts for this class of BLO problems, along with their optimality conditions and standard algorithms (including their theoretical properties and practical implementations), and also how they can be used to obtain state-of-the-art results for a number of key SP and ML applications.

In the existing literature, several recent surveys have been conducted on general BLO problems [1], [5], [6]. However, these surveys primarily focus on the mathematical foundations of BLO through a classical optimization lens. Other works [19], [26] aim to provide comprehensive reviews of BLO algorithms, but they lack an in-depth discussion on "when" and "how" to apply them in practical applications. The most relevant work to ours is [26], which examines complex learning and vision problems from the BLO perspective. However, the theoretical component of BLO is missing, and it overlooks a significant portion of emerging SP and ML applications (e.g., those discussed in the sections "BLO for Wireless Resource Allocation," "BLO for Wireless Signal Demodulation," "BLO for Adversarially Robust Training," "BLO for Model Pruning," and "BLO for Invariant Representation Learning").

Unlike the existing surveys on BLO [5], [6], [19], [26], most of which provide a broad overview of BLO in its most generic form, we focus on the tractable and data-driven problems that

Motivating application: Coreset selection for model training

Many contemporary signal processing and machine learning applications are facing significant challenges in data storage, transportation, and computation because they have to deal with excessive amounts of data. Consequently, the task of identifying the most informative subset of data from a larger pool becomes crucial [7], [17]. This leads to the problem of *coreset selection*, which consists of two tasks: (T1), selecting the most representative data samples to form the coreset, and (T2), validating the performance of the selected coreset in model training. More specifically, the problem can be formulated as follows:

$$\underset{\mathbf{w}\in\mathcal{U}}{\text{minimize}}\ \ell_{\text{val}}(\boldsymbol{\theta}^*(\mathbf{w})) \tag{T1}$$

$$\text{subject to}\ \boldsymbol{\theta}^*(\mathbf{w}) = \underset{\boldsymbol{\theta}}{\text{argmin}} \frac{1}{N}\sum_{i=1}^{N} w_i \cdot \ell(\boldsymbol{\theta}; \mathbf{x}_i, y_i) \tag{T2}$$

where $\mathbf{w}$ represents the weight vector for data selection over N training data points, with $w_i = 0$ indicating that the ith data sample $(\mathbf{x}_i, y_i)$ is not selected, and ℓ denotes the loss function for individual training samples. These weights are subject to the sparsity constraint $\mathcal{U}$, such as $\|\mathbf{w}\|_1 \le k$, with k being the selection budget. The model parameters trained on the selected data points are denoted by $\boldsymbol{\theta}$. The training loss for the model $\boldsymbol{\theta}$ with the data selection scheme $\mathbf{w}$ is given by $\ell_{\text{tr}}(\boldsymbol{\theta}, \mathbf{w}) := 1/N \sum_{i=1}^{N} w_i \cdot \ell(\boldsymbol{\theta}; \mathbf{x}_i, y_i)$, while the validation loss measuring the performance of the learned model $\boldsymbol{\theta}^*(\mathbf{w})$ over the coreset is denoted by ℓ_{val}. The previous BLO formulation is also related to the data reweighting problem [20] and hyperparameter optimization [22].

are relevant to SP and ML applications. A few highlights of this article are listed next.

First, we distill the common structures and properties of BLO that emerge across applications related to developing robust, parsimonious, and generalizable data-driven models in SP and ML. Our goal is to provide insights about when, where, and how BLO formulations and algorithms can be best used to yield a significant performance boost, as compared with traditional, or heuristic algorithms. In this process, we present some recent theoretical results about BLO and the associated algorithms to give a flavor of the current advances in the research area, while discussing their practical and scalable implementations.

We focus on a class of tractable BLO problems that often appear in SP and ML applications.

Second, we dive deep to understand the performance of a selected subset of state-of-the-art gradient-based BLO algorithms on a number of representative applications. Instead of relying on results reported in existing works, which may not always be directly comparable because of implementation differences, we designed an experiment plan and implemented all benchmarking algorithms. The goal is not only to showcase the effectiveness of the BLO-based algorithms but also to analyze the pros (e.g., modeling flexibility and accuracy performance) and the cons (e.g., runtime efficiency) of different subclasses of BLO methods. This effort has led to the development of an open source project repository containing all of the codes for the experiments presented in the article.

Overall, we hope that our balanced treatment of the subject, together with the open source package developed to benchmark modern BLO algorithms, will serve as the cornerstone for the accelerated adoption of BLO in diverse application areas, including, but not limited to, SP and ML. Figure 1 provides an overview of the topics to be covered in this article.

Notation

We use lowercase letters (e.g., a), lowercase boldface letters (e.g., $\mathbf{a}$), and uppercase boldface letters (e.g., $\mathbf{A}$) to denote scalars, vectors, and matrices, respectively. For a vector $\mathbf{a}$, we use $\|\mathbf{a}\|_p$ to denote its ℓ_p-norm with the typical choice $p \in \{1, 2, \infty\}$. For a matrix $\mathbf{A}$, we use the superscript $\top$ (or $^{-1}$) to denote the transpose (or inverse) operation. We use $\mathbf{I}$ to represent the identity matrix. For a function $f(\mathbf{x}, \mathbf{y})$ (with $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^n$), we use $\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^m$ (or $(\partial f/\partial \mathbf{x})$) and $\nabla_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n$ (or $(\partial f/\partial \mathbf{y})$) to denote the *partial derivatives* of f with respect to the partial input argument $\mathbf{x}$ and $\mathbf{y}$, respectively. By contrast, we use $\nabla F(\mathbf{x})$ (or $(dF/d\mathbf{x}) \in \mathbb{R}^m$) to represent the *full derivative* of a possible implicit function (IF) $F(\cdot)$ with respect to $\mathbf{x}$, namely, $\nabla F(\mathbf{x}) = \nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}) + (d\mathbf{y}/d\mathbf{x})^\top \nabla_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$ following the chain rule, where $(d\mathbf{y}/d\mathbf{x})^\top \in \mathbb{R}^{m \times n}$ denotes the Jacobian matrix of $\mathbf{y}$ with respect to $\mathbf{x}$. For ease of notation, the transpose in $(d\mathbf{y}/d\mathbf{x})^\top \in \mathbb{R}^{m \times n}$ will be omitted if its definition is clear from the context. We use $\nabla_{\mathbf{x}} \nabla_{\mathbf{y}} f$ or $\nabla^2_{\mathbf{x},\mathbf{y}} f \in \mathbb{R}^{m \times n}$ to denote the second-order partial derivative of f.

Warm-up: Introducing the basic concepts of BLO

A class of tractable BLO problems

We start by discussing the challenges associated with the generic form of (BLO). Even under the assumption that all involved functions are well behaved, such as the convexity or concavity of $f(\cdot,\cdot)$ and $g(\cdot,\cdot)$, and the linearity of $h(\cdot,\cdot)$, solving the problem can still be highly challenging (i.e., *NP-hard*). To see this, let us consider the following simple example.

Example 1: Nonconvex BLO

Consider the following BLO, where $f(\theta, \phi) = -g(\theta, \phi) = \theta^2 - \theta \cdot \phi - \phi^2$:

$$\begin{aligned} &\underset{\theta \in [-1,1]}{\text{minimize}}\ \theta^2 - \theta \cdot \phi^*(\theta) - \phi^*(\theta)^2; \\ &\text{subject to } \phi^*(\theta) = \underset{\phi \in [-1,1], \theta - \phi = 0}{\operatorname{argmin}} -(\theta^2 - \theta \cdot \phi - \phi^2). \end{aligned}$$

Notice that the objective $f(\theta,\phi)$ (respectively, $g(\theta,\phi)$) is strongly convex (respectively, strongly concave) in θ and strongly concave in ϕ (respectively, strongly convex), and the previous BLO problem is subject to linear constraints in both the upper and lower levels. In other words, both the upper- and lower-level problems are "easy" problems with respect to their respective parameters. Nonetheless, it can be shown that solving the previous BLO requires tackling a *nonconvex problem*, which in general is NP-hard. Indeed, it is not hard to see that $\phi^*(\theta) = \theta$. As a result, the outer function can be expressed as $-\theta^2$, which is a nonconvex function.

We remark that the source of difficulty of the previous problem is the *coupling* constraint $\theta - \phi = 0$. If this constraint is removed, the problem will become a classical saddle-point problem, expressed next, whose global optimal solution can be easily obtained:

$$\underset{\theta \in [-1,1]}{\text{minimize}}\ \underset{\phi \in [-1,1]}{\text{maximize}}\ \theta^2 - \theta \cdot \phi - \phi^2. \tag{1}$$

The aforementioned examples, along with numerous others in existing survey papers like [1], [19], and [26], strongly motivate us to proceed with a *focused* discussion for the subset of tractable (BLO) problems. This subset often serves as a basis for developing practical BLO algorithms. By focusing on this subset, we can address the specific needs of modeling SP and ML problems, which frequently demand the development of efficient, and sometimes real-time, algorithms.

To this end, we consider some special classes of (BLO), with the following simplifications: 1) The lower-level constraint set, if present, is *linear* and is only related to $\boldsymbol{\phi}$; that is, $h(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbf{A}\boldsymbol{\phi} - \mathbf{b}$ for some matrix $\mathbf{A}$ and vector $\mathbf{b}$ of appropriate sizes; and 2) the solution of the lower-level problem is a

FIGURE 1. A taxonomy of the solvers and application tasks for BLO.

singleton, and in most cases we assume an even stronger condition that the objective function $g(\cdot,\cdot)$ is strongly convex in the second argument. With these simplifications, it is possible to show some nice properties; for example, the gradient of the upper-level objective function may exist, making algorithm design and analysis tractable. Note that there have been recent works that extend these conditions and are still able to develop efficient algorithms. We will discuss these works in the section "Theoretical Results of BLO."

In summary, depending on whether the lower-level problem has a constraint or not, we consider the following two classes of problems, referred to as the *lower unconstrained* (LU) and *lower constrained* (LC) BLO, respectively:

$$\begin{aligned} &\underset{\boldsymbol{\theta}\in\mathcal{U}}{\text{minimize}} \;\; F(\boldsymbol{\theta}) \coloneqq f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})), \\ &\text{subject to} \;\; \boldsymbol{\phi}^*(\boldsymbol{\theta}) = \underset{\boldsymbol{\phi}\in\mathbb{R}^n}{\operatorname{argmin}}\, g(\boldsymbol{\theta}, \boldsymbol{\phi}). \end{aligned} \tag{LU-BLO}$$

$$\begin{aligned} &\underset{\boldsymbol{\theta}\in\mathcal{U}}{\text{minimize}} \;\; F(\boldsymbol{\theta}) \coloneqq f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})), \\ &\text{subject to} \;\; \boldsymbol{\phi}^*(\boldsymbol{\theta}) = \underset{\boldsymbol{\phi}\in C}{\operatorname{argmin}}\, g(\boldsymbol{\theta}, \boldsymbol{\phi}). \end{aligned} \tag{LC-BLO}$$

where the set $C \coloneqq \{\boldsymbol{\phi} \mid \mathbf{A}\boldsymbol{\phi} - \mathbf{b} \leq \mathbf{0}\}$. As will be evident in the section "Algorithmic Foundations of BLO," the presence of lower-level constraints, even in the form of linear and uncoupled constraints, can make (LU-BLO) much harder to deal with than (LC-BLO).

Connections of BLO with game theory

It is important to note that BLO problems have strong ties with Stackelberg or leader–follower games [2], including Stackelberg congestion and security games. These are sequential games involving two players: the leader and the follower. The leader acts first, aiming to maximize its utility by leveraging its knowledge of the follower's anticipated response. The follower, acting second, maximizes its utility based on the leader's action. The connection between BLO and Stackelberg games can be summarized as follows. First, in certain Stackelberg games, the process of identifying a solution (i.e., a Stackelberg equilibrium) can be framed as a BLO. Second, BLO allows a (Stackelberg) game-theoretic interpretation, where the upper- and lower-level problems correspond to the tasks of identifying the optimal actions for the leader and the follower (i.e., the upper-/lower-level variables), respectively. A special case of Stackelberg game is min-max optimization (MMO) also referred to as the saddle point problem. MMO follows a bilevel structure, wherein the lower-level objective g in (BLO) is *exactly opposite* of the upper-level objective function f (i.e., $g = -f$), resulting in the following special case of BLO:

$$\underset{\boldsymbol{\theta}\in\mathcal{U}}{\text{minimize}} \; \underset{\boldsymbol{\phi}\in C}{\text{maximize}} \;\; f(\boldsymbol{\theta}, \boldsymbol{\phi}). \tag{MMO}$$

In fact, MMO is much simpler to deal with than BLO, and it has been heavily studied in the SP and ML communities. In addition to MMO, it is also worth noting that the algorithms reviewed in this article are generally applicable to Stackelberg games, provided that the game's BLO formulation adheres to the assumptions of the respective algorithms.

It is important to note that BLO problems have strong ties with Stackelberg or leader–follower games, including Stackelberg congestion and security games.

Implicit gradient

As alluded to previously, one important reason to consider problems (LU-BLO) and (LC-BLO) is that the objective functions of these problems are potentially *differentiable* with respect to $\boldsymbol{\theta}$. Indeed, by applying the chain rule and supposing for now that the Jacobian matrix $d\boldsymbol{\phi}^*(\boldsymbol{\theta})/d\boldsymbol{\theta}$ exists, we have

$$\nabla F(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) + \underbrace{\frac{d\boldsymbol{\phi}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}}}_{\text{IG}} \nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) \tag{2}$$

where we recall from our notational convention that $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \boldsymbol{\phi})$ and $\nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}, \boldsymbol{\phi})$ represent the *partial derivatives* of f with respect to the partial input arguments $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, respectively, and $\nabla F(\boldsymbol{\theta})$ denotes the *full derivative* of the IF, F, with respect to $\boldsymbol{\theta}$. For ease of notation, the transpose operation $^\top$ might be omitted in the rest of the article. We refer to the Jacobian matrix $d\boldsymbol{\phi}^*(\boldsymbol{\theta})/d\boldsymbol{\theta} \in \mathbb{R}^{n\times m}$ as the implicit gradient (IG). This term is introduced to characterize the gradient of the argmin-based lower-level objective function with respect to the upper-level variable $\boldsymbol{\theta}$. However, the IG does *not* always exist for generic BLO problems. Even for (LU-BLO) and (LC-BLO), relatively strong assumptions have to be imposed. For example, $g(\cdot,\cdot)$ needs to be strongly convex in its second argument. Further, even when the IG exists, computing it could be quite different for the two classes of problems (LU-BLO) and (LC-BLO). For the former, we will show in the section "Algorithmic Foundations of BLO" that the IG can be expressed in closed form using the implicit function theorem [27] based on the first-order stationary condition of the lower-level problem, i.e., $\nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) = \mathbf{0}$. For this reason, it is referred to as an *implicit* gradient. Yet, in (LC-BLO), the stationary condition cannot be used since a stationary point might violate the constraint $\boldsymbol{\phi} \in C$. Therefore, the IG generally does not admit any closed form. Additionally, for a more restricted subset of MMO problems, the influence of the IG-involved term $(\text{IG} \cdot \nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})))$ in (2) can be neglected. To see this, assume that the inner problem is unconstrained, i.e., $C \equiv \mathbb{R}^n$; then $\nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) = \mathbf{0}$ based on the fact that $\nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) = \mathbf{0}$ and $g = -f$ for MMO.

BLO with nonsingleton lower-level solutions

As we have mentioned in the section "A Class of Tractable BLO Problems," throughout this article, we will mostly focus on the case where the lower-level solution $\phi^*(\boldsymbol{\theta})$ is unique, i.e., a singleton. Yet, if the lower-level problem involves a

nonsingleton (NS) solution, the resulting BLO problem is typically cast as

$$\underset{\boldsymbol{\theta},\boldsymbol{\phi}' \in \mathcal{S}(\boldsymbol{\theta})}{\text{minimize}} f(\boldsymbol{\theta},\boldsymbol{\phi}') \ \text{ subject to } \ \mathcal{S}(\boldsymbol{\theta}) := \underset{\boldsymbol{\phi} \in C}{\text{argmin}}\, g(\boldsymbol{\theta},\boldsymbol{\phi}) \tag{NS-BLO}$$

where $\mathcal{S}(\boldsymbol{\theta})$ denotes a solution set. The previous formulation, also referred to as the *optimistic* BLO with NS lower-level solutions, has been discussed in the literature, for example, in [1] and [4]. Note that problem (NS-BLO) presents significantly greater challenges from both practical and theoretical perspectives compared to problem (BLO). The reason is that optimization over $\boldsymbol{\phi}$ is coupled across both upper- and lower-level objectives. While our work primarily focuses on BLO with a singleton lower-level solution, we will also explore in the section "Algorithmic Foundations of BLO" the applicability of BLO algorithms, derived from (BLO), to solve problem (NS-BLO).

Theory and algorithms for tractable BLO

In the next two sections, we will delve into the essential optimization principles employed in BLO, explore several popular classes of BLO algorithms, and examine their theoretical properties.

In comparison to IF- and GU-based methods, the VF-based approach has broader applicability in solving complex BLO problems.

Algorithmic foundations of BLO

This section presents an overview of three key optimization frameworks used to solve the tractable BLO problems (LU-BLO) and (LC-BLO). The first two classes both leverage (some approximated version of) the IG as defined in (2). The key difference is in how the approximation of the IG is conducted: one directly assumes that there is some given procedure that can provide a high-quality solution of the lower-level problem, while the other approximates the lower-level solution by unrolling a given algorithm for a fixed number of steps. The third class is referred to as the value function (VF)-based approach, which reformulates BLO as a single-level regularized optimization problem. It is worth mentioning that this approach offers flexibility in handling lower-level constraints and solving NS lower-level problems (NS-BLO).

The IF-based approach

The IF for lower-level unconstrained BLO

Let us examine the problem setup (LU-BLO) with a singleton lower-level solution. For ease of theoretical analysis in the section "Theoretical Results of BLO," we further assume that $g(\cdot)$ is strongly convex in $\boldsymbol{\phi}$. In certain applications, one can explicitly add a strongly convex regularization function, such as $\gamma \cdot \|\boldsymbol{\phi}\|_2^2$ (with large enough γ), to satisfy such an assumption.

The reason that we call this approach *IF based* is that we will explicitly utilize the implicit function theorem [27] to calculate the IG as expressed in (2). Recall from (LU-BLO) that $\boldsymbol{\phi}^*(\boldsymbol{\theta})$ is a lower-level solution; thus, it satisfies

$$\nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) = \mathbf{0}. \tag{3}$$

Following the implicit function theorem, we can take the first-order derivative of (3) with respect to $\boldsymbol{\theta}$, yielding $(d/d\boldsymbol{\theta})[\nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))] = \mathbf{0}$. Further assume that the lower-level objective $g(\cdot)$ is second-order differentiable; we can then obtain the IG in (2) with the following:

$$\frac{d\boldsymbol{\phi}^*(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = -\nabla^2_{\boldsymbol{\theta},\boldsymbol{\phi}} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) \nabla^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))^{-1}. \tag{4}$$

As observed previously, the computation of the IG involves the mixed (second-order) partial derivative $\nabla^2_{\boldsymbol{\theta},\boldsymbol{\phi}} g$ and the inverse of the Hessian $\nabla^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g$. Yet, computing these quantities can be challenging in practice. Therefore, the class using the IF approach utilizes different kinds of approximation techniques to approximately compute the IG as expressed in (4). We summarize the IF approach in Algorithm 1.

Practical considerations of IF

In Algorithm 1, the main computational overhead arises from the inverse Hessian gradient product $\mathbf{H}^{-1}\mathbf{g}$, where $\mathbf{H} := \tilde{\nabla}^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))$ and $\mathbf{v} := \nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))$. Yet, in many contemporary applications, directly computing and storing the Hessian is computationally prohibitive. To address the scalability challenge in the IF method, we introduce four approaches to approximate the inverse Hessian gradient product (or the inverse Hessian) $\mathbf{H}^{-1}\mathbf{g}$: the conjugate gradient (CG) method [28], the WoodFisher approximation [29], the Neumann-series method to directly estimate the inverse Hessian [30], [31], and a Hessian-free simplification [10], [18]. These methods offer different tradeoffs between computational costs, with the CG method being the most computationally expensive and the Hessian-free simplification being the least expensive.

First, the CG approach maps the product $\mathbf{H}^{-1}\mathbf{g}$ to the solution of a quadratic program defined as $\min_{\mathbf{x}} \mathbf{x}^\top \mathbf{H}\mathbf{x}/2 - \mathbf{g}^\top \mathbf{x}$. By utilizing the first-order GD algorithm, we can numerically

Algorithm 1: IF-based approach for solving (LU-BLO).

Given initialization $\boldsymbol{\theta}_0$, learning rate $\alpha > 0$, and iteration number T, iteration $t \geq 0$ yields the following:

- *Lower-level optimization:* Given $\boldsymbol{\theta}_t$, obtain an approximate solution of the lower-level problem, denoted as $\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)$.
- *Approximation:* Based on $\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)$, compute approximated versions of two second-order matrices in (4), denoted as $\tilde{\nabla}^2_{\boldsymbol{\theta},\boldsymbol{\phi}} g(\boldsymbol{\theta}, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))$ and $\tilde{\nabla}^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g(\boldsymbol{\theta}, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))^{-1}$; compute an approximated IG following (2):

$$\tilde{\nabla} F(\boldsymbol{\theta}_t) := \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)) - \tilde{\nabla}^2_{\boldsymbol{\theta},\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)) \tilde{\nabla}^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))^{-1} \nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)). \tag{5}$$

- *Upper-level optimization:* Utilize $\tilde{\nabla} F(\boldsymbol{\theta}_t)$ in (5) to update $\boldsymbol{\theta}_t$ through, e.g., gradient descent (GD): $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \tilde{\nabla} F(\boldsymbol{\theta}_t)$.

approximate $\mathbf{H}^{-1}\mathbf{g}$. However, the convergence speed of the CG method depends on the smallest eigenvalue of the positive definite matrix $\mathbf{H}$. Therefore, if the lower-level problem is not well conditioned, the CG method can be slow. This approach has also been employed in the context of model-agnostic metalearning (MAML) [16] and adversarially robust training [10].

Second, the WoodFisher approximation [29] expresses the Hessian as a recurrence of a rank-one modified Hessian estimate and calls the Woodbury matrix identity to compute the inverse of a rank-one modification to the given Hessian matrix. The *one-shot* WoodFisher approximation is equivalent to the quasi-Newton approximation, $\mathbf{H} \approx \mathbf{v}\mathbf{v}^T + \gamma\mathbf{I}$, where $\gamma > 0$ is the damping term to render the invertibility of $\mathbf{H}$. We can then readily obtain the inverse Hessian vector product by the Woodbury matrix identity $\mathbf{H}^{-1}\mathbf{v} = \gamma^{-1}\mathbf{v} + (\gamma^{-2}\mathbf{v}\mathbf{v}^\top/1 + \gamma^{-1}\mathbf{v}^\top\mathbf{v})\mathbf{v}$. Further, an *iterative* WoodFisher approximation for the Hessian inverse proposed in [29] enhances the estimation accuracy.

MAML, as an optimization-based meta-learning approach, has gained significant popularity in various fields, especially in scenarios with limited resources.

Third, one may use a Neumann-series approximation to estimate the inverse Hessian directly by the approximation $\mathbf{H}^{-1} \approx \Sigma_{i=0}^{K}[\mathbf{I} - \mathbf{H}]^i$. (assuming $\mathbf{H}$ is normalized to ensure $\|\mathbf{H}\| \leq 1$). Note that as $K \to \infty$ the approximation becomes increasingly more accurate. This technique is popular for approximating the inverse Hessian in a stochastic setting wherein the upper- and lower-level objectives are accessed via a stochastic oracle [30], [31]. Here, we briefly describe the procedure to approximate the inverse Hessian stochastically using the Neumann-series method. Let us choose k uniformly randomly from the set $\{0, 1, \ldots, K-1\}$, access batch samples of $g(\boldsymbol{\theta}, \boldsymbol{\phi})$ denoted by $\{g(\boldsymbol{\theta}, \boldsymbol{\phi}; \zeta_k)\}_{k=1}^{k}$, and compute

$$\mathbf{H}^{-1} \approx \frac{k}{L_g} \prod_{i=1}^{k} \left(\frac{I - \nabla^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g(\boldsymbol{\theta}, \boldsymbol{\phi}; \zeta_i)}{L_g} \right) \tag{6}$$

where L_g is the Lipschitz smoothness constant of $g(\boldsymbol{\theta}, \boldsymbol{\phi}; \zeta_k)$. This procedure requires the computation of k stochastic Hessians and their products. Importantly, this estimator is a biased estimator of the inverse Hessian with the bias decreasing exponentially with K [30, Lemma 3.2].

Finally, to ensure the local convexity, some quadratic regularization term is usually added to the lower-level problem in BLO [10], [18]. This modifies (LU-BLO) to

$$\begin{aligned} &\underset{\boldsymbol{\theta} \in \mathcal{U}}{\text{minimize}}\ F(\boldsymbol{\theta}) \coloneqq f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})), \\ &\text{subject to}\ \ \boldsymbol{\phi}^*(\boldsymbol{\theta}) = \underbrace{\underset{\boldsymbol{\phi} \in \mathbb{R}^n}{\operatorname{argmin}}\, g(\boldsymbol{\theta}, \boldsymbol{\phi}) + \frac{\lambda}{2}\|\boldsymbol{\phi}\|_2^2}_{g'(\boldsymbol{\theta}, \boldsymbol{\phi})} \end{aligned} \tag{7}$$

where we recall that $\lambda > 0$ is a regularization parameter. In this context, the Hessian-free simplification is usually adopted, which assumes $\tilde{\nabla}^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)) = \mathbf{0}$. This assumption can be reasonable when the lower-level objective function g involves deep model training. For instance, in the case of a neural network with rectified linear unit activation, the decision boundary is piecewise linear in a tropical hypersurface, leading to an approximate Hessian of zeros. This Hessian-free simplification has been used for pruning DNNs [18]. Thus, the Hessian matrix of $g'(\boldsymbol{\theta}, \boldsymbol{\phi})$ in (7) will be simplified to $\mathbf{H} \approx \lambda\mathbf{I}$.

Extension to lower-level constrained BLO

Unlike in the previous section, it turns out that when including constraints to the lower-level problem, the IG no longer has the closed-form expression because the stationary condition in (3) does not hold anymore. To see the impact of having constraints (even linear ones) on the lower-level problem, we present the following example, where the gradient $df/d\boldsymbol{\theta}$ is not rigorously defined.

Example 2: (LC-BLO) can still be nondifferentiable [32]

Consider the following special case of (LC-BLO), where the lower-level objective is strongly convex in both scalar variables θ and ϕ, the upper level is linear, and both levels are subject to linear constraints:

$$\underset{\theta \in [0,1]}{\text{minimize}}\ \theta + \phi^*(\theta) \qquad \text{subject to}\ \phi^*(\theta) \in \underset{1/2 \leq \phi \leq 1}{\operatorname{argmin}} (\theta - \phi)^2.$$

It follows that $\phi^*(\theta) = 1/2$, for $\theta \leq 1/2$, and $\phi^*(\theta) = \theta$, for $\theta > 1/2$. We notice that at the point $\theta = 1/2$ the mapping $\phi^*(\theta)$ is continuous, but not differentiable. As a result, the outer function $\theta + \phi^*(\theta)$ is *nondifferentiable*.

An immediate question is: Can we still leverage the IF-based approach for this subclass of problems? It turns out that if we make some additional assumptions on the matrix $\mathbf{C}$ in the constraint set of (LC-BLO), one can still apply the implicit function theorem to the Karush–Kuhn–Tucker (KKT) condition of the lower-level problem to calculate the IG [10], [32]. It is also important to note that IF-based approaches are typically not suitable for handling general nonlinear constraints in the lower-level problem. For problems with complex constraints, VF- or penalty-based approaches are often employed [33], [34].

The gradient unrolling-based approach

The gradient unrolling (GU)-based approach is another class of popular algorithms for solving BLO problems in practice [13], [22], [23], [24]. Unlike the IF-based framework, it employs an unrolled lower-level optimizer as an intermediary step to connect the lower-level solution with the upper-level optimization process. An automatic differentiation (AD) technique is then used to compute gradients with respect to the upper-level optimization variable $\boldsymbol{\theta}$. Consequently, the computation of the IG in GU is dependent on the choice of the lower-level optimizer, and it no longer uses the IF-based expressions (3) and (4).

GU-based approach for unconstrained BLO

In particular, the GU-based approach approximates $\boldsymbol{\phi}^*(\boldsymbol{\theta})$ by running a given algorithm for a fixed number of iterations and

then inserting the entire trajectory into the upper-level objective. See Algorithm 2 for an illustration of the idea.

To see the difference between the AD- and IF-based approaches, let us consider the simple case where $h(\cdot)$ is the gradient mapping, $q(\boldsymbol{\theta}_t, \boldsymbol{\phi}_{k-1}) = \boldsymbol{\phi}_{k-1} - \beta \times \nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \boldsymbol{\phi}_{k-1})$ (for some constant step-size $\beta > 0$), and $K = 1$ (i.e., a single-step GD step is performed for lower-level optimization). Further assume that $\boldsymbol{\phi}_0$ is independent of $\boldsymbol{\theta}$; then the closed-form expression of the IG can be written as

$$\frac{d\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)}{d\boldsymbol{\theta}} = \frac{d[\boldsymbol{\phi}_0 - \beta \times \nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \boldsymbol{\phi}_0)]}{d\boldsymbol{\theta}} = -\beta \nabla^2_{\boldsymbol{\theta},\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \boldsymbol{\phi}_0). \quad (9)$$

In some sense, the preceding computation is simpler than the computation of the IG (4) in the IF-based approach since the Hessian inverse is no longer needed. However, things can get much more complicated very quickly as the total number of inner iterations K increases. Suppose that $K = 2$; then we obtain

$$\begin{aligned}\frac{d\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)}{d\boldsymbol{\theta}} &= \frac{d[\boldsymbol{\phi}_1 - \beta \nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \boldsymbol{\phi}_1)]}{d\boldsymbol{\theta}} \\ &= -\beta[\mathbf{I} + \beta \times \nabla^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \boldsymbol{\phi}_1)] \nabla^2_{\boldsymbol{\theta},\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \boldsymbol{\phi}_0). \quad (10)\end{aligned}$$

Clearly, the Hessian inverse is still not needed, but as the number of unrolling steps increases, much higher computational and memory requirements will be involved.

Practical considerations

When the unrolling step K becomes too large or the problem scale itself is computationally expensive for GU, manual unrolling becomes necessary to save memory costs and reduce the computational overhead. Various GU approaches have been proposed to achieve this goal efficiently. Notable techniques include forward gradient unrolling (FGU) [24], backward gradient unrolling (BGU) [22], [23], [24], and truncated gradient unrolling (TGU) [13], [22]. FGU performs unrolling iteratively, and in the final step K, the Jacobian of ϕ_K with respect to $\boldsymbol{\theta}$ yields

$$\begin{aligned}\underbrace{\frac{d\boldsymbol{\phi}_K}{d\boldsymbol{\theta}}}_{\mathbf{Z}_K} &= \underbrace{\frac{\partial \boldsymbol{\phi}_K}{\partial \boldsymbol{\phi}_{K-1}}}_{\mathbf{A}_K} \underbrace{\frac{d\boldsymbol{\phi}_{K-1}}{d\boldsymbol{\theta}}}_{\mathbf{Z}_{K-1}} + \underbrace{\frac{\partial \boldsymbol{\phi}_K}{\partial \boldsymbol{\theta}}}_{\mathbf{B}_K} = \cdots = \sum_{i=1}^{K} \left(\prod_{t=i+1}^{K} \mathbf{A}_t \right) \mathbf{B}_i \\ &\quad + \left(\prod_{t=1}^{K} \cdot \mathbf{A}_t \right) \mathbf{Z}_0. \quad (11)\end{aligned}$$

We also assume that $\boldsymbol{\phi}_0$ is independent of $\boldsymbol{\theta}$, which implies $d\boldsymbol{\phi}_0/d\boldsymbol{\theta} = 0$. Consequently, expression (11) can be rewritten as the following iterative form:

$$\mathbf{Z}_k = \mathbf{A}_k \mathbf{Z}_{k-1} + \mathbf{B}_k, \; k = 1, 2, \ldots, K. \quad \text{(FGU)}$$

Both $\mathbf{A}_k$ and $\mathbf{B}_k$ will be calculated along with the kth lower-level step $\boldsymbol{\phi}_k = q(\boldsymbol{\theta}_t, \boldsymbol{\phi}_{k-1})$ and will be discarded immediately after $\mathbf{Z}_k$ is obtained. Such an iterative nature of (FGU) makes it particularly suitable for scenarios that involve a large number of unrolling steps K as the memory cost of calculating $\mathbf{A}_k$ and $\mathbf{B}_k$ only involves any gradient flow generated within the kth step. However, (FGU) requires keeping track of the matrices $\mathbf{A}_k, \mathbf{B}_k$, and $\mathbf{Z}_{k-1}$. Hence, it may not be suitable for problems with high-dimensional variables $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$.

To achieve more efficient computations when $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are of large scale, BGU is introduced, which eliminates the need for storing any intermediate matrices [22], [23], [24]. BGU explores the calculation of the IG following (2):

$$\begin{aligned}\frac{df(\boldsymbol{\theta}, \boldsymbol{\phi}_K)}{d\boldsymbol{\theta}} &= \underbrace{\frac{\partial f(\boldsymbol{\theta}, \boldsymbol{\phi}_K)}{\partial \boldsymbol{\theta}}}_{\mathbf{c}_K} + \underbrace{\frac{d\boldsymbol{\phi}_K^\top}{d\boldsymbol{\theta}}}_{\mathbf{Z}_K^\top} \underbrace{\frac{\partial f(\boldsymbol{\theta}, \boldsymbol{\phi}_K)}{\partial \boldsymbol{\phi}_K}}_{\mathbf{d}_K} \\ &\overset{(FGU)}{=} \mathbf{c}_K + (\mathbf{Z}_{K-1}^\top \mathbf{A}_K^\top + \mathbf{B}_K^\top) \mathbf{d}_K \\ &= \underbrace{(\mathbf{c}_K + \mathbf{B}_K^\top \mathbf{d}_K)}_{\mathbf{c}_{K-1}} + \mathbf{Z}_{K-1}^\top \cdot \underbrace{\mathbf{A}_K^\top \mathbf{d}_K}_{\mathbf{d}_{K-1}} = \mathbf{c}_{K-1} \\ &\quad + \mathbf{Z}_{K-1}^\top \mathbf{d}_{K-1} = \cdots = \mathbf{c}_0 + \mathbf{Z}_0^\top \mathbf{d}_0 = \mathbf{c}_{-1}. \quad (12)\end{aligned}$$

Instead of calculating IG explicitly like (FGU) does, (12) directly obtains the gradient of the upper-level variable, which can be further simplified with the following recursive formulas:

$$\begin{aligned}&\mathbf{c}_{k-1} = \mathbf{c}_k + \mathbf{B}_k^\top \mathbf{d}_k, \; k = 0, 1, \ldots, K, \text{ with } \mathbf{c}_K = \frac{\partial f(\boldsymbol{\theta}, \boldsymbol{\phi}_K)}{\partial \boldsymbol{\theta}} \in \mathbb{R}^m \\ &\mathbf{d}_{k-1} = \mathbf{A}_t^\top \mathbf{d}_k, \; k = 0, 1, \ldots, K, \text{ with } \mathbf{d}_K = \frac{\partial f(\boldsymbol{\theta}, \boldsymbol{\phi}_K)}{\partial \boldsymbol{\phi}_K} \in \mathbb{R}^n. \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(BGU)}\end{aligned}$$

It can be observed that (BGU) only requires storing vectors ($\mathbf{c}_k$ and $\mathbf{d}_k$) throughout the recursion by utilizing the Jacobian–vector product trick. As a result, BGU is particularly advantageous for problems with large-scale variables compared to FGU. Yet, because of its recursive nature, BGU can be conducted only after all of the K lower-level steps are finished. Thus, (BGU) needs to store all of the unrolling steps $\{\boldsymbol{\phi}_k \in \mathbb{R}^n\}_{k=1}^K$, compared with (FGU). Consequently, it may not be efficient for handling BLO as the number of unrolling steps K grows.

It should also be noted that GU differs from the IF as its computation relies on the choice of the lower-level optimizer. For instance, employing sign-based GD (signGD) as the lower-level optimizer leads to a computationally efficient GU variant

Algorithm 2: GU-based approach for solving (LU-BLO).

Given initializations $\boldsymbol{\phi}_0$ and $\boldsymbol{\theta}_0$ and iteration numbers K and T, let $q(\cdot): \mathcal{U} \times C \to C$ denote one step of a given algorithm, which takes both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ as input and outputs an updated $\boldsymbol{\phi}$. At iteration $t \geq 0$,

- *Lower-level optimization by K-step optimization:*

$$\boldsymbol{\phi}_\kappa = q(\boldsymbol{\theta}_t, \boldsymbol{\phi}_{\kappa-1}), \quad \kappa = 1, \ldots K. \quad (8)$$

Define $\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t) := \boldsymbol{\phi}_\kappa = q(\boldsymbol{\theta}_t, q(\boldsymbol{\theta}_t, \ldots, q(\boldsymbol{\theta}_t, \boldsymbol{\phi}_0)))$;

- *Upper-level optimization:* Leverage AD to compute the approximated gradient $\hat{\nabla} F(\boldsymbol{\theta}_t) := \frac{df(\boldsymbol{\theta}_t, q(\boldsymbol{\theta}_t, q(\boldsymbol{\theta}_t, \ldots, q(\boldsymbol{\theta}_t, \boldsymbol{\phi}_0))))}{d\boldsymbol{\theta}}$, and use this gradient to update $\boldsymbol{\theta}_t$.

referred to as signGD-based GU [15]. Specifically, the modified lower-level update rule (8) becomes

$$\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t) = \boldsymbol{\phi}_K;\ \ \boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} - \beta \operatorname{sign}(\nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \boldsymbol{\phi}_{k-1})),\quad k = 1, 2, \ldots, K, \tag{signGD}$$

where $\operatorname{sign}(\cdot)$ denotes the element-wise sign operation, and $\beta > 0$ is a certain learning rate. Given the approximation $d\operatorname{sign}(\mathbf{x})/d\mathbf{x} = \mathbf{0}$ (holding almost everywhere), the IG can be simplified to

$$\frac{d\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)}{d\boldsymbol{\theta}} = \frac{d\boldsymbol{\phi}_K}{d\boldsymbol{\theta}} = \frac{d\boldsymbol{\phi}_{K-1}}{d\boldsymbol{\theta}} = \cdots = \frac{d\boldsymbol{\phi}_0}{d\boldsymbol{\theta}}. \tag{13}$$

In the case that $\boldsymbol{\phi}_0$ is independent of $\boldsymbol{\theta}$, we can achieve the IG-free variant of the GU approach.

VF-based approach

VF-based methods [33], [34], [35], [36] can also avoid the computation of the inverse of the Hessian required in the IF method. The key technique is to reformulate a standard BLO problem into a constrained single-level optimization problem. This reformulation involves transforming the lower-level problem into an upper-level inequality constraint. The resulting VF-based variants can then be solved using algorithms for constrained optimization. Furthermore, in comparison to IF- and GU-based methods, the VF-based approach has broader applicability in solving complex BLO problems. Not only can it handle lower-level objectives with NS solutions (including both convex and nonconvex objectives), but more importantly, it can accommodate lower-level constraints as well. However, the VF-based approach has not yet been popular in practical SP and ML applications, partly because this approach has not been able to deal with large-scale *stochastic* problems. This point will be illustrated shortly in the section "Theoretical Results of BLO."

To understand the VF-based approach, consider the following *equivalent* reformulation of (LC-BLO):

$$\underset{\boldsymbol{\theta}, \boldsymbol{\phi} \in C}{\text{minimize}} f(\boldsymbol{\theta}, \boldsymbol{\phi}), \quad \text{subject to } g(\boldsymbol{\theta}, \boldsymbol{\phi}) \le g^*(\boldsymbol{\theta}) \tag{14}$$

where $g^*(\boldsymbol{\theta}) \coloneqq \min_{\boldsymbol{\phi} \in C} g(\boldsymbol{\theta}, \boldsymbol{\phi})$ is referred to as the VF of the lower-level problem. However, solving (14) is highly nontrivial, partly because $g^*(\boldsymbol{\theta})$ is not necessarily smooth, and it can be nonconvex. To address these challenges, a relaxed version of problem (14) is typically considered by replacing $g^*(\boldsymbol{\theta})$ with a smooth surrogate [33], [35]

$$g^*_{\mu}(\boldsymbol{\theta}) = \underset{\boldsymbol{\phi} \in C}{\text{minimize}}\, g(\boldsymbol{\theta}, \boldsymbol{\phi}) + \frac{\mu_1}{2} \| \boldsymbol{\phi} \|_2^2 + \mu_2 \tag{15}$$

where $\mu \coloneqq (\mu_1, \mu_2)$ is a pair of positive coefficients that are introduced to guarantee the smoothness of $g^*_{\mu}(\boldsymbol{\theta})$ and to ensure the feasibility of the inequality constraint $g(\boldsymbol{\theta}, \boldsymbol{\phi}) \le g^*_{\mu}(\boldsymbol{\theta})$.

Given the relaxed VF formulation, one can adopt standard nonlinear optimization algorithms, such as a penalty-based algorithm, to solve the constrained optimization problem (14). For example, a log-barrier interior-point method, called the bilevel value-function-based interior-point method (BVFIM), is leveraged in [35] to solve a sequence of smooth approximated single-level problems of (14). Other related methods to solve (14) include primal–dual bilevel optimization (PDBO) [33], bilevel optimization made easy (BOME!) [36], and V-penalty-based bilevel GD (PBGD) [34] methods.

Theoretical results of BLO

In this section, we examine the theoretical guarantees of various BLO methods. The section is divided into two parts. In the first part, we discuss the convergence results of popular algorithms for solving the (LU-BLO) problem, while the second part will focus on the more general formulations, such as (LC-BLO) and (NS-BLO). We list specific algorithms that can handle both stochastic and deterministic BLO problems. Given the differences in theoretical analysis between stochastic and deterministic optimization, we also consider a generalized stochastic version of BLO, whose upper- and lower-level objectives are

$$f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) \coloneqq \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}); \xi_i),$$
$$g(\boldsymbol{\theta}, \boldsymbol{\phi}) \coloneqq \frac{1}{N} \sum_{i=1}^{N} g(\boldsymbol{\theta}, \boldsymbol{\phi}; \zeta_i) \tag{16}$$

where $\xi_i \sim \mathcal{D}_f$ (respectively, $\zeta_i \sim \mathcal{D}_g$) represents the data sample of the upper-level (respectively, lower-level) objective from the distribution $\mathcal{D}_f$ (respectively, $\mathcal{D}_g$), and N is the total number of data samples.

Convergence measures of BLO

In what follows, we introduce the convergence measures utilized for evaluating the performance of BLO algorithms. These measures serve to assess the quality of solutions obtained by these algorithms. Note that in general the IF $F(\boldsymbol{\theta}) \coloneqq f(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))$ may be nonconvex; therefore, we define the concept of an ϵ-*stationary* point for (LU-BLO), which plays a crucial role in characterizing the convergence properties of BLO when the upper-level problem is unconstrained, i.e., $\mathcal{U} = \mathbb{R}^m$. In the deterministic setting of (LU-BLO), a point $\bar{\boldsymbol{\theta}} \in \mathbb{R}^m$ is considered an ϵ-stationary point if it satisfies $\| \nabla F(\bar{\boldsymbol{\theta}}) \|_2^2 \le \epsilon$. In the stochastic setting (16), where the algorithm incorporates randomness, the expectation is taken over the stochasticity of the algorithm. Thus, an ϵ-stationary point is defined as $\mathbb{E} \| \nabla F(\bar{\boldsymbol{\theta}}) \|_2^2 \le \epsilon$. It is important to note that when the upper-level problem in (LU-BLO) is constrained, i.e., $\mathcal{U} \subset \mathbb{R}^m$, then the upper-level objective $F(\bar{\boldsymbol{\theta}})$ may not be differentiable over $\bar{\boldsymbol{\theta}}$ in general. When solving (LC-BLO) using the IF-based approach, if the IF is differentiable, similar measures of stationarity as in standard optimization can be utilized. However, if the IF is nondifferentiable, alternative stationarity measures are commonly employed. These include subgradient optimality [32], proximal gradient methods [34], [37], and Moreau envelope techniques [31], [32], [38]. Furthermore, when employing VF-based approaches to solve (LC-BLO), a widely used measure of stationarity is to evaluate the convergence of the algorithms toward the KKT points of the constrained reformulation of the BLO problem [33].

In addition, the concept of *oracle complexity* is employed to quantify the number of gradient evaluations needed to obtain an ϵ-stationary solution, as defined earlier. We denote $\mathcal{G}(f,\epsilon)$ (respectively, $\mathcal{G}(g,\epsilon)$) as the total number of (stochastic) gradients of f (respectively, g) evaluated to achieve an ϵ-stationary solution. This measure provides insights into the computational requirements of BLO algorithms and their scalability with respect to the problem size and desired solution accuracy.

Convergence guarantees for LU-BLO

As discussed in the section "Algorithmic Foundations of BLO," solving (LU-BLO) requires computing the Hessian or its inverse. However, the stochastic formulation (16) leads to some challenges, especially for convergence analysis. For example, in the case of solving the stochastic BLO problem using Algorithm 1 (IF), the gradient estimates would be replaced with appropriate stochastic gradient estimates for both the upper- and lower-level updates. However, obtaining an *unbiased* estimator for the Hessian inverse term in the IG [as defined in (4)] is challenging. To overcome this challenge, a *biased* stochastic gradient estimator based on Neumann-series approximation, as discussed in the section "Practical Considerations of IF," has been used in [30] and [31]. We note that the bias of the estimator can be easily controlled by choosing a larger batch to compute the Hessian of the lower-level objective [30, Lemma 3.2]. Moreover, we point out that the stochastic CG (discussed in the section "Practical Considerations of IF") can also be utilized to obtain the inverse Hessian gradient vector product to approximate the IG [39], [40].

The problem of model pruning arises, aiming to reduce the sizes of an ML model by identifying and removing redundant model weights.

In addition, a key design choice for BLO algorithms is whether the inner problem is solved *accurately* or not. In a *single-loop* algorithm, one only performs a *fixed* number of steps for the lower-level updates before every upper-level update [31], [41], [42], while in a *double loop*, many lower-level updates are carried out to obtain a very accurate approximation of $\boldsymbol{\phi}^*(\boldsymbol{\theta})$ [30], [39], [40]. Typically, the former is simpler to implement in practice, while the latter is easier to analyze since the error caused by approximating $\boldsymbol{\phi}^*(\boldsymbol{\theta})$ can be well controlled. In addition, the stochastic descent direction to solve both (or either) upper- and lower-level problems can be constructed using either vanilla stochastic GD (SGD) [43] or variance-reduced (VR) algorithms [44]. Specifically, it is well known that VR-based algorithms can lead to improved theoretical convergence of stochastic algorithms compared to vanilla SGD-based algorithms to solve standard optimization problems. The VR-based algorithms accomplish this improved convergence by computing additional stochastic gradients on optimization variables computed in consecutive iterations [44]. Similar behavior is observed in solving BLO algorithms using VR-based gradient constructions, as discussed next.

In Algorithm 3, we provide a generic stochastic algorithm to solve BLO problems using the IF-based approach. As pointed out earlier, for double-loop algorithms, $\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)$ will approximate $\boldsymbol{\phi}^*(\boldsymbol{\theta}_t)$ closely, while for a single-loop algorithm, $\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)$ could be given by a crude approximation of $\boldsymbol{\phi}^*(\boldsymbol{\theta}_t)$. Table 1 provides a summary of the oracle complexities of existing BLO algorithms for achieving an ϵ-stationary point in solving problem (LU-BLO). The theoretical results are categorized based on three algorithmic families: stochastic BLO (16), VR-based BLO, and deterministic BLO. The convergence performance is evaluated in terms of the oracle complexities $\mathcal{G}(f,\epsilon)$ and $\mathcal{G}(g,\epsilon)$, as introduced in the section "Convergence Measures of BLO." In Table 1, we also illustrate the BLO solver employed by different optimization principles (i.e., IF, GU, or VF) and the algorithmic design choices, such as double versus single loop.

Stochastic BLO

This set of algorithms updates the lower- and upper-level variables using SGD, following the IF or GU optimization principle. The stochastic gradient estimates are evaluated as discussed in Algorithm 3, while for the deterministic setting, the gradients are approximated using the techniques discussed in the section "Algorithmic Foundations of BLO." Bilevel stochastic approximation (BSA) [30] was the first algorithm that offered finite-time convergence guarantees for solving unconstrained stochastic BLO problems. It employed a double-loop algorithm where the lower-level variable is iteratively estimated with multiple SGD updates, resulting in a larger oracle complexity of $\mathcal{O}(1/\epsilon^3)$ for the inner-level optimization compared to $\mathcal{O}(1/\epsilon^2)$ for the upper-level optimization. Two-timescale stochastic approximation (TTSA) [31], a fully single-loop algorithm (with projected SGD update for upper-level constrained optimization) improved the lower-level oracle complexity to $\mathcal{O}(1/\epsilon^{5/2})$; however, at the cost of worsening the upper-level oracle complexity to $\mathcal{O}(1/\epsilon^{5/2})$. More recently, the stochastic bilevel optimizer (stocBiO) [40], the stochastic bilevel algorithm (SOBA) [46], and the alternating stochastic GD (ALSET) [42] algorithm have been developed to achieve $\mathcal{O}(1/\epsilon^2)$ complexity for both the

Algorithm 3: (S)GD and VR for solving stochastic (LU-BLO) and (LC-BLO).

Given the initialization $\boldsymbol{\theta}_0$ and iteration number T, iteration $t \geq 0$ yields:

- *Lower-level optimization:* Given $\boldsymbol{\theta}_t$, call SGD (or VR based on both $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t-1}$) to obtain a lower-level solution $\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)$.
- *Approximation:* Given $\tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t)$, compute a stochastic gradient estimate of (5) as follows:
  - Estimate stochastic versions of $\nabla_\theta f(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))$, $\nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))$, $\tilde{\nabla}^2_{\theta,\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))$.
  - Approximate Hessian inverse, $\tilde{\nabla}^2_{\boldsymbol{\phi},\boldsymbol{\phi}} g(\boldsymbol{\theta}_t, \tilde{\boldsymbol{\phi}}(\boldsymbol{\theta}_t))^{-1}$, following (6) in the section "Practical Considerations of IF."
  - Obtain stochastic estimate of (5) and construct a descent direction $\tilde{\nabla} F(\boldsymbol{\theta}_t; \tilde{\xi})$ for $\boldsymbol{\theta}_t$.
- *Upper-level optimization:* Call SGD (or VR) to update $\boldsymbol{\theta}_t$ using $\tilde{\nabla} F(\boldsymbol{\theta}_t; \tilde{\xi})$ (or $\tilde{\nabla} F(\boldsymbol{\theta}_t; \tilde{\xi})$ and $\tilde{\nabla} F(\boldsymbol{\theta}_{t-1}; \tilde{\xi})$).

Table 1. Summary of convergence results of representative BLO algorithms for solving (LU-BLO).

	Stochastic BLO				VR BLO				Deterministic BLO			
Algorithm	**Principle**	**Design**	**Oracle Complexity**		**Algorithm**	**Principle**	**Design**	**Oracle Complexity**	**Algorithm**	**Principle**	**Design**	**Oracle Complexity**
			$\mathcal{G}(f,\epsilon)$	$\mathcal{G}(g,\epsilon)$				$\mathcal{G}(f,\epsilon)$ $\mathcal{G}(g,\epsilon)$				$\mathcal{G}(f,\epsilon)$ $\mathcal{G}(g,\epsilon)$
BSA [30]	IF	Double	$O(1/\epsilon^2)$	$O(1/\epsilon^3)$	STABLE [38]	IF	Single	$O(1/\epsilon^2)$ $O(1/\epsilon^2)$	BA [30]	IF	Double	$O(1/\epsilon)$ $O(1/\epsilon^{5/4})$
TTSA [31]	IF	Single	$O(1/\epsilon^{5/2})$	$O(1/\epsilon^{5/2})$	SUSTAIN [41]	IF	Single	$O(1/\epsilon^{3/2})$ $O(1/\epsilon^{3/2})$	AID-BiO [40]	IF	Single	$O(1/\epsilon)$ $O(1/\epsilon)$
stocBiO [40]	IF	Double	$O(1/\epsilon^2)$	$O(1/\epsilon^2)$	VRBO [45]	IF	Double	$O(1/\epsilon^{3/2})$ $O(1/\epsilon^{3/2})$	ITD-BiO [40]	GU	Double	$O(1/\epsilon)$ $O(1/\epsilon)$
ALSET [42]	IF	Single	$O(1/\epsilon^2)$	$O(1/\epsilon^2)$	SABA [46]	IF	Double	$O(N^{2/3}/\epsilon)$ $O(N^{2/3}/\epsilon)$	MSTSA [41]	IF	Single	$O(1/\epsilon)$ $O(1/\epsilon)$
F^{2}SA [48]	VF	Single	$O(1/\epsilon^{7/2})$	$O(1/\epsilon^{7/2})$	F^{3}SA [48]	VF	Single	$O(1/\epsilon^{5/2})$ $O(1/\epsilon^{5/2})$	K-RMD [22]	GU	Double	$O(1/\epsilon^2)$ $O(K/\epsilon^2)$
AmIGO [39]	IF	Double	$O(1/\epsilon^2)$	$O(1/\epsilon^2)$	SBFW [49]	IF	Single	$O(1/\epsilon^4)$ $O(1/\epsilon^4)$	FGU/BGU [23]	GU	Double	N/A N/A

The convergence performance is measured by the oracle complexity $\mathcal{G}(f,\epsilon)$ and $\mathcal{G}(g,\epsilon)$ for upper- and lower-level optimization, respectively. "Double" or "Single" refers to the choice of the algorithm design, either double loop or single loop. BSA: bilevel stochastic approximation; TTSA: two-timescale stochastic approximation; stocBiO: stochastic bilevel optimizer; ALSET: alternating stochastic GD; F^{2}SA: fully first-order stochastic approximation; STABLE: single-timescale stochastic bilevel optimization; SUSTAIN: single-timescale double-momentum stochastic approximation; VRBO: VR bilevel optimizer; SABA: stochastic average bilevel algorithm; F^{3}SA: faster fully first-order stochastic approximation; BA: bilevel approximation; AID-BiO: approximate implicit differentiation bilevel optimizer; MSTSA: momentum-assisted single-timescale stochastic approximation; SBFW: stochastic bi-level Frank-Wolfe; ITD-BiO: iterative differentiation-based bilevel optimization; K-RMD: K-step reverse mode differentiation.

upper- and the lower-level optimization. Note that stocBiO [40] requires a batch size of $O(1/\epsilon)$ to achieve this complexity, while ALSET [42] and SOBA [46] rely on only $O(1)$ batches to achieve the same performance. In [48], the authors developed fully first-order stochastic approximation (F^{2}SA), a VF-based algorithm to solve (LU-BLO). The algorithm achieved an oracle complexity of $O(1/\epsilon^{7/2})$ for both upper- and lower-level objectives while circumventing the need to compute Hessians (or Hessian vector products) during the execution of the algorithm.

VR stochastic BLO

Several VR algorithms have been proposed to improve the performance of vanilla stochastic algorithms by computing additional stochastic gradients in each iteration (see Algorithm 3). Examples including single-timescale stochastic bilevel optimization (STABLE) [38] and momentum-assisted single-timescale stochastic approximation (MSTSA) [41] utilized momentum-based variance reduction techniques [44] for upper-level optimization, improving the performance of TTSA [31] and BSA [30] and achieving an oracle complexity of $O(1/\epsilon^2)$ for both upper- and lower-level objectives. Single-timescale double-momentum stochastic approximation (SUSTAIN) [41], the momentum-based recursive bilevel optimizer [45], and the stochastic VR bilevel method [50] further improved the performance by applying variance reduction to both upper- and lower-level optimization, achieving an oracle complexity of $O(1/\epsilon^{3/2})$. The VR bilevel optimizer (VRBO) [45] employed the stochastic path-integrated differential estimator [51], a double-loop VR gradient estimator, to achieve the same complexity. More recently, a stochastic average bilevel algorithm (SABA) [46] was developed, applying SAGA (an incremental gradient estimator) [52] to achieve an oracle complexity of $O(N^{3/2}/\epsilon)$, where N is the number of empirical data points. In addition to F^{2}SA, the authors of [48] also proposed the faster fully first-order stochastic approximation (F^{3}SA), which improved on the oracle complexity of F^{2}SA. F^{3}SA utilized momentum-based variance reduction to achieve an oracle complexity of $O(1/\epsilon^{5/2})$ for both upper- and lower-level objectives.

Deterministic BLO

Popular approaches for solving deterministic BLO problems include iterative differentiation (ITD) and approximate implicit differentiation (AID) methods. ITD-based approaches, proposed in [23] and [24], established asymptotic convergence guarantees and were extended to TGU in [22], achieving oracle complexities of $O(1/\epsilon^2)$. Improved guarantees were later shown in [40], achieving oracle complexities of $O(1/\epsilon)$, comparable to solving a deterministic single-level optimization problem. AID-based approaches for solving deterministic BLO problems include bilevel approximation (BA) [30], the AID bilevel optimizer (AID-BiO) [40], and MSTSA [41]. BA, a double-loop algorithm, was the first to establish convergence guarantees for deterministic BLO using AID. BA achieved an oracle complexity

of $O(1/\epsilon)$ for the upper-level optimization and $O(1/\epsilon^{5/4})$ for the lower-level optimization. The performance of BA was improved in AID-BiO [40] and MSTSA [41], which achieved an oracle complexity of $O(1/\epsilon)$ for both upper- and lower-level objectives. Please refer to Table 1 for a summary of the discussed approaches.

Table 2. Convergence results of representative algorithms for solving (LC-BLO) and (NS-BLO) problems.

		Objective Functions			
Algorithm	**Principle**	**Upper Level**	**Lower Level**	**Constraints**	**Oracle Complexity**
BDA [21]	GU	Strongly convex	Convex	No	Asymptotic
BVFIM [35]	VF	Smooth	Nonconvex	No	Asymptotic
BOME! [36]	VF	Smooth	PL	No	$O(1/\epsilon^4)$
SIGD [32]	IF	Smooth	Strongly convex	Linear inequality	Asymptotic
AiPOD [37]	IF	Smooth	Strongly convex	Linear equality	$O(1/\epsilon^2)$
PDBO [33]	VF	Smooth	Convex	Nonlinear	$O(1/\epsilon^{3/2})$
PBGD [34]	VF	Smooth	PL	Nonlinear	$O(1/\epsilon^{3/2})$

Similar to Table 1, convergence is measured by oracle complexity. Other algorithmic details include optimization principles (IF, GU, and VF), problem setups [(LC-BLO) and (NS-BLO)], objective function types, and lower-level constraint types. BDA: bilevel descent aggregation; SIGD: smoothed implicit gradient; AiPOD: alternating implicit projected SGD.

Convergence guarantees for (LC-BLO) and (NS-BLO)

Obtaining convergence guarantees for BLO algorithms becomes more challenging when solving more complex problems, such as those involving lower-level constraints in (LC-BLO) or NS lower-level solutions in (NS-BLO). In the previous section, the majority of the algorithms discussed employed an IF-based approach to solve the BLO problem. However, in this section, only the algorithms designed to solve (LC-BLO) utilize IF-based approaches. It is also worth mentioning that IF-based approaches are not applicable for solving (NS-BLO) [or (LC-BLO) with general constraints] because of the inapplicability of the implicit function theorem in this context. Instead, standard approaches to solve these more complex problems include interior-point methods [35], primal–dual methods [33], dynamic barrier GD [36], and penalty-based GD [34]. As pointed out earlier, a major drawback of these algorithms is that they are exclusively developed for deterministic problems and lack efficient implementations for stochastic formulations (16). Consequently, they are not well suited for large-scale SP and ML applications, which often involve learning over large volumes of data.

In the following, we present a summary of recent theoretical advancements for solving highly complex BLO problems, such as (LC-BLO) and (NS-BLO) (see Table 2). We list the oracle complexities of representative methods, design principles, and the type of constraints present in the lower-level objective function. Note that in Table 2, we list the oracle complexity for only the upper-level objective with the notion of stationarity defined according to either the squared norm of the projected gradient [10], [32], [34] or KKT conditions [33], [36].

Theoretical results for (LC-BLO)

BLO problems of the form (LC-BLO) involving linear constraints of the form $C \coloneqq \{\boldsymbol{\phi} \mid h(\boldsymbol{\theta}, \boldsymbol{\phi}) \le \mathbf{0}\}$ with $h(\boldsymbol{\theta}, \boldsymbol{\phi}) \coloneqq \mathbf{A}\boldsymbol{\phi} - \mathbf{b}$ in the lower level have gained popularity in both theory and practice [10], [32], [37]. Under some regularity assumptions on upper- and lower-level objectives and the constraint set of the lower-level problem, IF-based methods can be developed for solving such problems. These algorithms (see, e.g., Algorithm 4) follow the same structure as the one presented in Algorithm 1; the key difference is that the construction of the (stochastic) gradient estimate depends on the lower-level constraints.

In [32], the SIGD approach was developed to handle (LC-BLO) with linear inequality constraints in the lower level. SIGD is an implicit GD algorithm that ensures the differentiability of the IF through perturbation-based smoothing. The SIGD algorithm and the expression for IG used in [32] are stated in (17). SIGD guarantees asymptotic convergence to a stationary point. A similar IF-based approach was also utilized in [10], termed Fast Bilevel Adversarial Training (Fast-BAT). Fast-BAT aims to enhance the robustness of deep learning models against adversarial attacks using BLO. It achieves an oracle complexity of $O(1/\epsilon)$ under certain smoothness assumptions. In [37], the stochastic BLO problem with linear equality constraints in both the upper- and lower-level problems is considered. The authors proposed an IF-based approach by constructing an approximate stochastic implicit gradient for linearly constrained BLO. They also proposed the alternating implicit projected SGD (AiPOD) algorithm, an alternating projection method that achieves an oracle complexity of $O(1/\epsilon^2)$ for both upper- and lower-level objectives. The work [33] considered BLO with general constraints in both upper- and lower-level objectives and NS lower-level solutions. It utilized the VF-based approach developed in BVFIM [35] for solving (NS-BLO) and proposed PDBO, a primal–dual algorithm for solving (14) when

Algorithm 4: Smoothed implicit gradient (SIGD), an IF-based approach for (LC-BLO).

Given the initialization $\boldsymbol{\theta}_0$ and iteration number T, iteration $t \ge 0$ yields the following:

- Call Algorithm 1 and use the following procedure to compute the IG.
- **Notation:** Let $\bar{\mathbf{A}}(\boldsymbol{\phi})$ be the matrix that contains the rows of $\mathbf{A}$ that correspond to the active constraints of inequality $\mathbf{A}\boldsymbol{\phi} - \mathbf{b} \le \mathbf{0}$, and $\hat{\boldsymbol{\lambda}}^*(\boldsymbol{\theta})$ is the Lagrange multipliers vector that corresponds to the active constraints at $\boldsymbol{\phi}^*(\boldsymbol{\theta})$; compute:

$$\begin{aligned}\text{IG}: \frac{d\boldsymbol{\phi}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}} &= [\nabla^2_{\phi\phi} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))]^{-1} \\ &\quad \times [-\nabla^2_{\theta\phi} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) - \bar{\mathbf{A}}^T \nabla \hat{\boldsymbol{\lambda}}^*(\boldsymbol{\theta})] \\ \nabla \hat{\boldsymbol{\lambda}}^*(\boldsymbol{\theta}) &= -[\bar{\mathbf{A}}[\nabla^2_{\phi\phi} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))]^{-1} \bar{\mathbf{A}}^\top]^{-1} \\ &\quad \times [\bar{\mathbf{A}}[\nabla^2_{\phi\phi} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))]^{-1} \nabla^2_{\theta\phi} g(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))]. \end{aligned} \tag{17}$$

the VF is approximated using (15). Under the assumptions of convex and compact constraint sets and convex lower-level objectives, PDBO achieves an oracle complexity of $\mathcal{O}(1/\epsilon^{3/2})$. Recently, the authors of [34] proposed PBGD for BLO with general constraints and NS lower-level solutions. The authors established the equivalence of BLO and its penalty-based reformulations based on VF and KKT conditions. PBGD achieved an oracle complexity of $\mathcal{O}(1/\epsilon^{3/2})$ for solving constrained BLO with lower-level objectives satisfying the Polyak-Łojasiewicz (PL) inequality. Note that the algorithms PDBO [33] and PBGD [34] can be utilized to solve both (LC-BLO) and (NS-BLO) problems. Next, we discuss specific algorithms for solving (NS-BLO).

Theoretical results for (NS-BLO)

An attempt to relax the lower-level singleton assumption for the lower-level problem was made in [21] with the introduction of bilevel descent aggregation (BDA), a bilevel descent framework for solving (NS-BLO). BDA assumes convexity of the lower-level objective and strong convexity of the upper-level objective with respect to $\boldsymbol{\phi}$. The framework updates the lower-level variable, $\boldsymbol{\phi}$, using a convex combination of the upper- and lower-level partial gradients and then updates the upper-level variable, $\boldsymbol{\theta}$, using standard FGU/BGU techniques. The authors established the asymptotic convergence of BDA in [21]. In [35], the authors relaxed the convexity assumptions on the lower- and upper-level objectives in (NS-BLO) and proposed BVFIM, a VF-based approach to solving the problem. BVFIM solves a sequence of penalty-based reformulations of the VF problem using the interior-point method with asymptotic convergence. In [36], the authors introduced BOME!, an alternative approach to directly solve the VF problem using a dynamic barrier GD algorithm. Under the assumption of PL inequality for the lower-level objective, BOME! achieves a finite-time sample complexity of $\mathcal{O}(1/\epsilon^4)$ in the worst case.

BLO-enabled SP and ML applications

In the following sections, we will showcase how BLO can be leveraged to obtain state-of-the-art results for a number of key SP and ML applications, such as wireless resource allocation (see the section "BLO for Wireless Resource Allocation"), signal demodulation (see the section "BLO for Wireless Signal Demodulation"), adversarial training for robustifying ML models (see the section "BLO for Adversarially Robust Training"), weight pruning for enhancing model efficiency (see the section "BLO for Model Pruning"), and invariant representation learning for improving domain generalization (see the section "BLO for Invariant Representation Learning"). Table 3 summarizes a number of emerging BLO application areas, together with some representative references.

BLO for wireless resource allocation

In this section, we explore the application of BLO in wireless communications, specifically in the context of wireless optimal resource allocation (power control) [7]. The goal is to allocate power efficiently among multiple transmitter–receiver pairs to maximize some system-level performance. We consider a dynamic environment where wireless channel statistics change *episodically*, where the environment statistics change in "episodes," and in each episode the environment is stationary. To solve this problem, we employ a neural network trained to predict the optimal power allocation for users based on channel information. However, neural networks often struggle when evaluated on data that deviate from the training distribution.

Table 3. An overview of emerging applications of BLO in SP and ML (👍 image indicates applications we studied).

Representative Applications	Application Areas	Problem Description	Selected References
Wireless resource allocation 👍	SP	To allocate wireless resources optimally and maximize their utilities	[7]
Signal demodulation 👍		To accurately estimate transmitted symbols from received baseband signals	[53]
Channel prediction		To predict the states of a communication channel by leveraging previous observations	[54]
Image reconstruction		To recover images from their sparse measurements	[9]
Adversarial training 👍	Robust ML	To train an ML model with adversarial robustness against adversarial attacks	[10], [11]
Poisoning attack generation		To generate malicious data into the training set, creating vulnerabilities/backdoors in ML models	[12], [13]
Model pruning 👍	Efficient ML	To find sparse subnetworks from a dense DNN without generalization loss	[18], [55]
Dataset condensation		To select a subset or distill a condensed version of the training set without generalization loss	[17], [19]
MAML 👍	Generalized ML	To train an ML model that can quickly adapt to new tasks using limited data	[14], [15], [16]
IRM 👍		To train an ML model with invariant features against distribution shift	[56], [57], [58]
Neural architecture search	Automated ML	To automatically optimize the architecture of DNNs for improved performance	[25]
Hyperparameter optimization		To optimize the hyperparameters and model selection schemes in an ML pipeline	[21], [22], [23], [24]

IRM: invariant risk minimization.

To address this challenge, we adopt a continual learning framework [59] and maintain a memory set, containing a representative subset of samples encountered so far, to facilitate adaptation to new episodes while preserving performance on previous ones. The model is trained not only on the current data batch but also on the memory set, and the dual requirement of continuously updating the memory set while optimizing the system performance leads to the bilevel formulation. In what follows, we provide a detailed problem formulation [7].

Formulation

Consider a dynamic wireless environment with T episodes, where the channel state information (CSI) statistics remain stationary within each episode. Consider a supervised learning setting, where the ith data pair $(\mathbf{h}^{(i)}, \mathbf{p}^{(i)})$ consists of the CSI vector $\mathbf{h}^{(i)}$ (the feature vector) capturing the channel characteristics and the corresponding *optimal* power allocation $\mathbf{p}^{(i)}$ across the users (the label). We train a neural network $\pi(\boldsymbol{\theta}; \mathbf{h}^{(i)})$ on these data pairs, where $\boldsymbol{\theta}$ represents the model parameters and $\mathbf{h}^{(i)}$ serves as the network input, with the output being the power allocation prediction. Assuming the data arrive sequentially in multiple batches, let D_t denote the batch we receive at time t. Assume that there is a fixed-size memory set M_t available, which stores representative historical data to be combined with D_t for training, and it is updated when a new batch arrives. The performance of a power allocation scheme $\mathbf{p}$ (for a given CSI $\mathbf{h}$) is measured by the weighted sum–rate loss function $R(\mathbf{p}; \mathbf{h})$ [7, equation (1)].

Note that all IRM variants outperform ERM, which justifies the importance of IRM training to improve model generalization across diverse environments.

At each time t, our problem involves two tasks. The first task is to train the neural network on a (weighted) set of training samples, aiming to find the optimal model parameter $\boldsymbol{\theta}$. The second task is to select the most representative subset from the available training data, which includes the samples in memory M_t and the current data batch D_t (denoted as $M_t \cup D_t$). This selected subset will be then used for training as well as to form the new memory to be used in the next time $t + 1$. Toward this end, we introduce the variable $\boldsymbol{\lambda}$, which represents the weights associated with each sample. Higher weights are assigned to samples that are more representative or challenging, as determined by the system performance metric $R(\mathbf{p}; \mathbf{h})$. These weighted samples (with nonzero weights) are then selected to form the updated memory set, and they will contribute to the next round of training. The idea is that by focusing on training the model on these challenging samples, we can expect better performance on the remaining easier samples. This problem can be naturally formulated as the following BLO problem:

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{minimize}} \quad & \sum_{i \in M_t \cup D_t} \lambda^{(i)}(\boldsymbol{\theta}) \ell(\boldsymbol{\theta}; \mathbf{h}^{(i)}, \mathbf{p}^{(i)}) \\ \text{subject to} \quad & \boldsymbol{\lambda}(\boldsymbol{\theta}) = \underset{1^T \boldsymbol{\lambda} = 1}{\operatorname{argmin}} \sum_{i \in M_t \cup D_t} \lambda^{(i)} R(\pi(\boldsymbol{\theta}; \mathbf{h}^{(i)}); \mathbf{h}^{(i)}) \end{aligned} \quad (18)$$

where $\ell(\boldsymbol{\theta}; \mathbf{h}^{(i)}, \mathbf{p}^{(i)}) = \left\| \mathbf{p}^{(i)} - \pi(\boldsymbol{\theta}; \mathbf{h}^{(i)}) \right\|_2^2$ is the mean-square-error loss over the ith sample, and $M_t \cup D_t$ are the available training samples at time t. At the upper level, supervised training is performed by using the weighted loss, while at the lower level, the weights are optimized based on their achieved rates, where higher weights are assigned to samples achieving lower rates. Based on solutions obtained by solving (18) [denoted as $(\boldsymbol{\theta}_t, \boldsymbol{\lambda}_t)$], in iteration $t + 1$ a new memory set is formed by $M_{t+1} := \{ i \,|\, \lambda_t^{(i)} > 0 \}$.

Methods

Problem (18) is a *constrained BLO* problem with linear equality constraints with respect to the lower-level variable $\boldsymbol{\lambda}$. Based on the problem structure and the optimization principles and algorithms introduced in the sections "Algorithmic Foundations of BLO" and "Convergence Guarantees for (LC-BLO) and (NS-BLO)," we utilize the IF-based SIGD method [32] to solve (18). However, it should be noted that the SIGD method assumes a strongly convex lower-level problem (see Table 2). To ensure this property, we introduce a regularization term $(\gamma/2)\| \boldsymbol{\lambda} \|_2^2$, where γ is the regularization parameter, as described in the section "Extension to Lower-Level Constrained BLO." As a classical baseline approach, we also consider transfer learning (TL). In TL, when a new data batch D_t arrives, the current model trained on data up to time $t - 1$ is fine-tuned using only D_t. This approach is motivated by the expectation that previous knowledge can be transferred to the new environment, enabling quick adaptation of the model. However, updating the model may result in a loss of prior knowledge, leading to performance degradation on the prior episodes.

Experiment results

We consider an experiment setting with $T = 4$ episodes and 10 users and with three different types of communication channels: Rayleigh fading, Rician fading, and Geometry channels; see [7, p. 13] for more details. The neural network trained for power allocation consists of three hidden layers with sizes 200, 80, 80, respectively. Figure 2 illustrates the performance of power allocation, measured by the sum rate; in the horizontal axis we have the total number of samples used for model training as these arrive sequentially in batches. Here the power allocation schemes are obtained using the BLO-based SIGD method and the baseline approach (TL), respectively. As we can see, the SIGD method exhibits smoother adaptation to each episode (note that the boundaries between episodes locate at x=2,4,6,8). It also experiences less deterioration in performance compared to the baseline approach. These results demonstrate the advantage of using BLO for power allocation.

BLO for wireless signal demodulation

In this section, we explore the application of BLO in wireless signal demodulation by associating it with another BLO

application, MAML [14]. Thus, we begin by introducing the fundamental concepts of MAML within the framework of BLO and then establish the connection between MAML and signal demodulation.

Fundamentals of BLO in MAML

MAML, as an optimization-based metalearning approach, has gained significant popularity in various fields, especially in scenarios with limited resources [14], [15], [16]. Specifically, MAML learns a *metainitialization* of optimized variables (e.g., model weights $\boldsymbol{\theta}$) to enable fast adaptation to new tasks when fine-tuning the model from the learned initialization with only a few new data points [14]. With N learning tasks $\{\mathcal{T}_i\}_{i=1}^N$, 1) a fine-tuning set $\mathcal{D}_i^{\text{tr}}$ is used in $\mathcal{T}_i$ for the *task-specific lower-level optimization* over the task-agnostic model initialization $\boldsymbol{\theta}$, and 2) a validation set $\mathcal{D}_i^{\text{val}}$ is used in the *upper-level optimization* for evaluating the fine-tuned model $\boldsymbol{\theta}_i^*$ from $\boldsymbol{\theta}$. Thus, MAML can be formulated as the following BLO problem:

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{minimize}} \quad & \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{(\mathcal{D}_i^{\text{tr}},\, \mathcal{D}_i^{\text{val}})\sim\mathcal{T}_i}\left[\ell_i\left(\boldsymbol{\theta}_i^*(\boldsymbol{\theta}); \mathcal{D}_i^{\text{val}}\right)\right] \\ \text{subject to} \quad & \boldsymbol{\theta}_i^*(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}_i \in \mathbb{R}^n}{\operatorname{argmin}}\, \ell_i(\boldsymbol{\theta}_i; \mathcal{D}_i^{\text{tr}}, \boldsymbol{\theta}), \end{aligned} \qquad \text{(MAML-BLO)}$$

where $\boldsymbol{\theta}_i^*(\boldsymbol{\theta})$ signifies the fine-tuned model weights using the initialization $\boldsymbol{\theta}$ under the task $\mathcal{T}_i$, and ℓ_i denotes the model training (or validation) loss over $\mathcal{D}_i^{\text{tr}}$ (or $\mathcal{D}_i^{\text{val}}$) with initialization $\boldsymbol{\theta}$ (or fine-tuned model $\boldsymbol{\theta}_i^*(\boldsymbol{\theta})$).

The MAML-BLO problem falls into the category of unconstrained BLO. Thereby, existing works, such as [14], [15], and [16], commonly employ the IF- or GU-based approaches to solve it. The vanilla MAML algorithm [14] utilizes a GU-based BLO solver, which carries out the upper- and lower-level updates using the following steps:

$$\begin{aligned} &\text{Lower: } \boldsymbol{\theta}_i^*(\boldsymbol{\theta}) = \boldsymbol{\theta}_i^{(M)},\ \boldsymbol{\theta}_i^{(m)} \leftarrow \boldsymbol{\theta}_i^{(m-1)} - \beta\nabla_{\boldsymbol{\theta}_i}\ell_i(\boldsymbol{\theta}_i^{(m-1)}), \\ &\qquad\qquad m = 1, \ldots, M, \text{ given } \boldsymbol{\theta}_i^{(0)} = \boldsymbol{\theta} \\ &\text{Upper: } \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{\theta}}\ell_i(\boldsymbol{\theta}_i^*(\boldsymbol{\theta})), \end{aligned} \qquad \text{(MAML)}$$

where $\alpha,\ \beta > 0$ represent the learning rates for the SGD updates in the upper and the lower level, respectively. As mentioned in the section "GU-Based Approach for Unconstrained BLO"), the choice of the lower-level optimizer will greatly influence the GU-based BLO solvers.

As described in the section "Practical Considerations," the usage of the sign-based SGD in the lower level can lead the vanilla MAML to a first-order BLO solver, known as Sign-MAML [15]. In contrast to the GU-based MAML methods discussed previously, the implicit MAML (iMAML) [16] utilizes an IF-based approach, where the CG method is used to compute the inverse Hessian gradient product. Compared to the vanilla MAML, iMAML shares the same lower-level updating rule, while adopting the IF-based upper-level iteration, similar to (5). Unlike IF and GU, the first-order MAML (FO-MAML) operates by alternating between SGD-based lower-level optimization and SGD-based upper-level optimization, without explicitly considering the implicit gradient. We refer to this optimization procedure as alternating optimization (AO).

We next demonstrate the effectiveness of BLO in MAML by applying it to benchmark few-shot learning tasks on the Omniglot and Mini-ImageNet datasets, where the generalization of the learned metainitialization is evaluated on the new

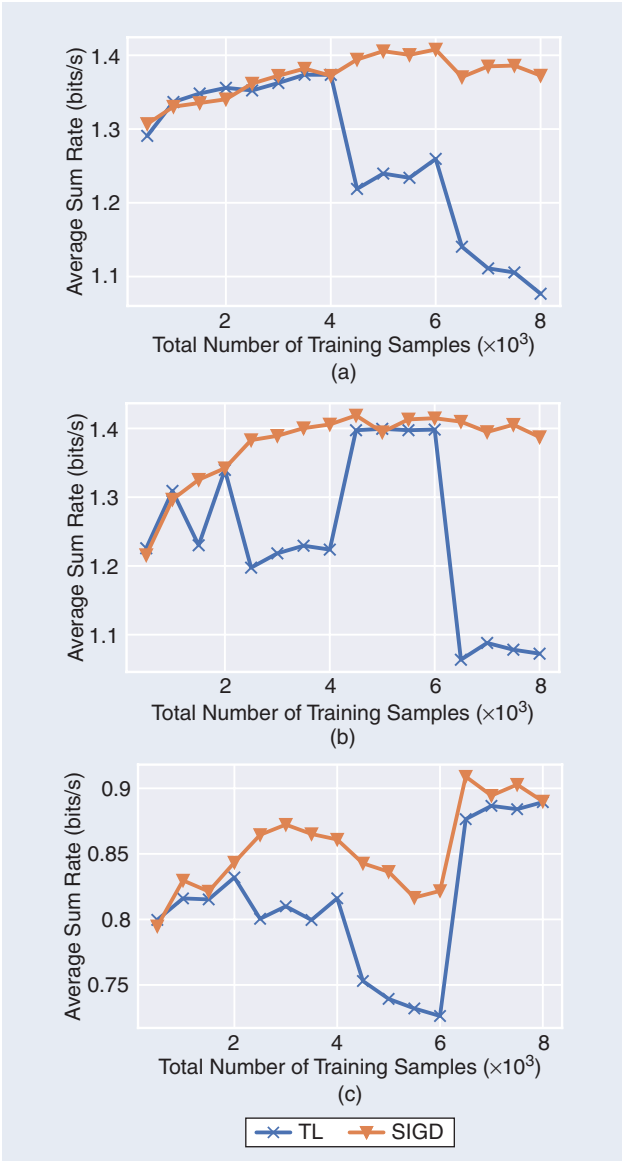


FIGURE 2. The average sum rate achieved on the combined test set (across all four episodes) is plotted as a function of the total number of samples used for model training, which arrive sequentially in batches. Three experiments are conducted with different types of channel statistics across the four episodes. (a) The channel sequence is Rayleigh-Rician-Geometry10-Geometry50. (b) The channel sequence is Rician-Geometry10-Rayleigh-Geometry50. (c) The channel sequence is Geometry10-Geometry20-Geometry50-Rayleigh. The numbers after the Geometry channel indicate the spatial arrangement of the nodes, such as a 20 m × 20 m area for Geometry20.

tasks, each with only a few examples. We follow the standard experimental setting [14], [15], [16], considering 20-way-1-shot and 20-way-5-shot learning on Omniglot and 5-way-1-shot and 5-way-5-shot learning on Mini-ImageNet. Here P-way-Q-shot refers to training a model using a small set of data points sampled from P classes, with each class containing Q examples. Table 4 provides an overview of the accuracy and runtime efficiency of different methods for solving MAML-BLO, including AO-based FO-MAML [14], GU-based Sign-MAML [15], GU-based vanilla MAML [14], and IF-based iMAML [16]. As we can see, the vanilla MAML and iMAML generally achieve higher testing accuracy than other MAML variants, but they require more computation time. This is expected for their fewer implementation assumptions, resulting in more precise metainitialization states. In the FO optimization category, we observe that Sign-MAML outperforms FO-MAML, demonstrating the advantage of using GU to solve BLO problems.

Developing efficient algorithms that can handle NS lower-level solutions and provide convergence guarantees is a key research direction.

BLO for wireless signal demodulation

Background and connection to MAML

We next examine the application of BLO in the context of wireless signal demodulation through the lens of MAML. Wireless signal demodulation aims to recover the *transmitted symbols* $\mathbf{s}$ from the *received signals* $\mathbf{y}$. Our investigation aligns with prior research [8] and focuses on a scenario where wireless devices frequently transmit short packets with a few pilot symbols through a varying channel. Notably, only a limited number of pilot symbols is available to optimize the demodulator when working with any new wireless transmitter device in the field. Meanwhile, the historical data pairs for previous devices and channel conditions can hardly transfer to new ones. Thus, demodulation modeling can be viewed as a few-shot metalearning problem, aiming to obtain a metainitialization that is able to quickly adapt to future devices. Similar to MAML, a metademodulator characterized by a learnable parameter $\boldsymbol{\theta}$ is trained to quickly adjust to new devices with only a few new pilot symbols.

Formulation

Specifically, N supervised datasets are collected to train the metademodulator, each associated with a specific device, which can be treated as N tasks. The ith dataset with K samples is given by $\mathcal{D}_i = \{(\mathbf{s}_i^{(k)}, \mathbf{y}_i^{(k)})\}_{k=1}^{K}$, where $\mathbf{y}_i^{(k)}$ represents the kth received signal, and $\mathbf{s}_i^{(k)}$ is its corresponding ground-truth transmitted symbol. The demodulation task for each single device can be formulated as a classification problem as each symbol $s_i^{(k)}$ can only be one of the several binary encodings, e.g., ranging from 0000 to 1111 following the 16-quadrature amplitude modulation (16-QAM) [8]. Thus, the cross-entropy loss between the predicted transmitted symbol $\hat{\mathbf{s}}(\mathbf{y}, \boldsymbol{\theta})$ and the true symbol $\mathbf{s}$ is used to train the demodulator model: $\ell_i(\boldsymbol{\theta}; D_i) = \mathbb{E}_{(\mathbf{s}_i^{(k)}, \mathbf{y}_i^{(k)}) \sim \mathcal{D}_i} \mathcal{L}_{\mathrm{CE}}(\hat{\mathbf{s}}(\mathbf{y}_i^{(k)}, \boldsymbol{\theta}), \mathbf{s}_i^{(k)})$, where we use a multilayer neural network as the demodulation model $\boldsymbol{\theta}$ following [8] to predict the transmitted symbol $\hat{\mathbf{s}}$ using the received signal $\mathbf{y}$. Given N devices (datasets) within the metatraining dataset, each contains K data samples, which are divided into K^{tr} for fine-tuning $\boldsymbol{\theta}$ and K^{val} for validating the performance of the fine-tuned model. In line with the notions used in (MAML-BLO), the demodulation of the ith device can be regarded as learning task $\mathcal{T}_i$, which consists of a fine-tuning dataset $\mathcal{D}_i^{\mathrm{tr}}$ and a validation dataset $\mathcal{D}_i^{\mathrm{val}}$. To this end, we can apply the previously introduced MAML methods [14], [15], [16] to address the problem of wireless signal demodulation.

Experiment results

During the metatraining phase, we consider N = 1,000 different devices, each of which has $K^{\mathrm{tr}} \in \{1, 5, 10, 20\}$ training samples designated as the fine-tuning set. For the metatesting phase, we use another set of 100 devices, each of which has K^{tr} pairs for the few-shot demodulator fine-tuning and an additional 10,000 pilot symbol pairs for symbol classification accuracy evaluation. Table 5 shows the average classification accuracy and running time for different MAML methods, providing insights into how the choice of K^{tr} affects the performance of these methods. As we can see, MAML and iMAML achieved the highest symbol classification accuracy, although they required more computational time, consistent with the findings in Table 4. Sign-MAML outperformed FO-MAML in the one-shot scenario, benefiting from the effectiveness of the GU solver. However, in scenarios with more shots, FO-MAML can achieve a performance on par with that of MAML.

Table 4. Performance comparison of various MAML methods on the commonly used datasets for few-shot learning tasks Omniglot and Mini-ImageNet.

		Test Accuracy (%)	Time (min)	Test Accuracy (%)	Time (min)	Test Accuracy (%)	Time (min)	Test Accuracy (%)	Time (min)
Method	**BLO (Solver)**	**Omniglot 20-way-1-shot**		**Omniglot 20-way-5-shot**		**Mini-ImageNet 5-way-1-shot**		**Mini-ImageNet 5-way-5-shot**	
FO-MAML [14]	AO	90.62 ± 0.29	2.71	96.44 ± 0.23	2.99	46.39 ± 0.44	4.32	54.45 ± 0.29	4.93
Sign-MAML [15]	GU	91.75 ± 0.26	2.85	97.79 ± 0.18	2.91	47.73 ± 0.55	4.51	55.12 ± 0.33	4.72
Vanilla MAML [14]	GU	95.65 ± 0.25	4.71	98.42 ± 0.23	4.94	48.77 ± 0.65	15.5	**55.72** ± 0.36	15.9
iMAML [16]	IF	**95.99** ± 0.19	3.64	**98.63** ± 0.14	3.85	**49.31** ± 0.41	11.6	54.71 ± 0.27	13.3

The best performance in each setting is marked in boldface. The standard deviations are reported based on five random trials. Rows marked in gray indicate BLO-enabled algorithms.

BLO for adversarially robust training

The lack of adversarial robustness in ML models has prompted extensive research on adversarial defense mechanisms [10], [11], [60]. While most of the existing defenses rely on MMO to minimize worst-case training loss by incorporating a synthesized adversary, this approach requires completely opposing objectives for the defender and attacker. This limits its applicability to scenarios where differing objectives are desired. Recent works [10], [11] have demonstrated the use of BLO with customizable attack objectives to improve the efficiency and robustness of robust model training across a wide range of adversarial attack strengths.

Formulation

We formulate the BLO-based robust training for defending against adversarial attacks. Using variables $\boldsymbol{\theta}$ for model parameters and $\boldsymbol{\delta}$ for input perturbation, and loss functions ℓ_{tr} for training and ℓ_{atk} for attacks, we define the task of robust training as a BLO problem (RT-BLO). The upper-level problem involves training the model $\boldsymbol{\theta}$, while the lower-level problem optimizes $\boldsymbol{\delta}$ for adversarial attack generation to produce worst-case input in model training. This yields the following:

$$\begin{aligned}&\underset{\boldsymbol{\theta}}{\text{minimize}}\ \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}}[\ell_{\text{tr}}(\boldsymbol{\theta},\mathbf{x}+\boldsymbol{\delta}^*(\boldsymbol{\theta}),y)],\\&\text{subject to}\ \boldsymbol{\delta}^*(\boldsymbol{\theta})=\underset{\boldsymbol{\delta}\in C}{\operatorname{argmin}}\,\ell_{\text{atk}}(\boldsymbol{\delta},\boldsymbol{\theta};\mathbf{x},y),\end{aligned}\qquad\text{(RT-BLO)}$$

where $(\mathbf{x}, y)$ is a data pair with feature $\mathbf{x}$ and label y drawn from the training dataset $\mathcal{D}$, $(\mathbf{x}+\boldsymbol{\delta})$ is an adversarial example with respect to $\mathbf{x}$, and $\boldsymbol{\delta}\in C$ denotes a perturbation constraint, e.g., $C=\{\boldsymbol{\delta}\,|\,\|\boldsymbol{\delta}\|_\infty\le\epsilon,\mathbf{x}+\boldsymbol{\delta}\in[\mathbf{0},\mathbf{1}]\}$ for an ϵ-tolerated ℓ_∞-norm constrained attack with normalized input in $[\mathbf{0},\mathbf{1}]$.

If we choose $\ell_{\text{atk}}=-\ell_{\text{tr}}$, then problem (RT-BLO) reduces to the MMO-based adversarial training [60]. However, the flexibility of independently choosing the lower-level attack objective allows for a broader range of robust training scenarios. In particular, it enables the development of a fast robust training variant called Fast-BAT [10]. The Fast-BAT formulation, given next, specifies the lower-level attack generation problem of (RT-BLO) as a constrained convex quadratic program:

$$\begin{aligned}&\underset{\boldsymbol{\theta}}{\text{minimize}}\quad\mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}}[\ell_{\text{CE}}(\boldsymbol{\theta},\mathbf{x}+\boldsymbol{\delta}^*(\boldsymbol{\theta};\mathbf{x},y),y)]\\&\text{subject to}\quad\boldsymbol{\delta}^*(\boldsymbol{\theta};\mathbf{x},y)\\&=\underset{\boldsymbol{\delta}\in C}{\operatorname{argmin}}\ \underbrace{-(\boldsymbol{\delta}-\boldsymbol{\delta}_0)^\top\nabla_{\boldsymbol{\delta}=\boldsymbol{\delta}_0}\ell_{\text{CE}}(\boldsymbol{\theta},\mathbf{x}+\boldsymbol{\delta},y)+(\gamma/2)\|\boldsymbol{\delta}-\boldsymbol{\delta}_0\|_2^2}_{:=\,\ell_{\text{atk}}(\boldsymbol{\theta},\boldsymbol{\delta};\mathbf{x},y)},\end{aligned}\qquad\text{(Fast-BAT)}$$

where $\ell_{\text{CE}}(\boldsymbol{\theta},\boldsymbol{x},y)$ is the cross-entropy loss for training model weights $\boldsymbol{\theta}$ evaluated at the data point $(\mathbf{x},y)$, and the lower-level attack objective is given by a first-order Taylor expansion of $-\ell_{\text{CE}}$ (at the linearization point $\boldsymbol{\delta}_0$) plus a quadratic residual with the regularization parameter $\gamma>0$. Since the lower-level problem becomes a convex quadratic program, it leads to the closed-form projected GD (PGD) solution $\boldsymbol{\delta}^*(\boldsymbol{\theta})=\mathcal{P}_C(\boldsymbol{\delta}_0-(1/\gamma)\nabla_{\boldsymbol{\delta}}\ell_{\text{CE}}(\boldsymbol{\theta},\boldsymbol{\delta}))\big|_{\boldsymbol{\delta}=\boldsymbol{\delta}_0}$.

Methods

As problem (Fast-BAT) falls into the category of (LC-BLO), we can solve it using optimization methods introduced in the section "Convergence Guarantees for (LC-BLO) and (NS-BLO)." Specifically, we consider the KKT-oriented IF approach (see the section "Extension to Lower-Level Constrained BLO") as our BLO solver (see Algorithm 4). In our experiments, we refer to this method as Fast-BAT-IF. Furthermore, we compare Fast-BAT-IF with non-BLO representative robust training baselines, such as Fast-AT [61] and Fast-AT-GA [62].

Experiment results

In Table 6, we empirically show the performance of different robust training methods to robustify PreActResNet-18 on the CIFAR-10 and Tiny-ImageNet datasets. The evaluation metrics include 1) the test-time robust accuracy (RA) of the learned model against 50-step PGD attacks [60] with 10 restarts (RA-PGD) using the perturbation budgets $\epsilon=8/255$ and 16/255; 2) RA against AutoAttack (RA-AA) [47] in a setup similar to RA-PGD; 3) the standard accuracy of the learned model on natural examples; and 4) the time consumption required for robust training. As observed, Fast-BAT-IF exhibits higher robustness compared to non-BLO baselines, highlighting the effectiveness of BLO in robust training.

BLO for model pruning

While overparameterized structures are key to the improved generalization of DNNs, they create new challenges—the huge number of parameters not only increases computational costs during inference, but it also poses serious deployment challenges on resource-limited devices. Thus, the problem of model pruning arises, aiming to reduce the sizes of an ML model by identifying and removing redundant model weights. In this section, we investigate the application of BLO in the context of model pruning [18], [55].

Table 5. Performance comparison of different MAML methods for few-shot demodulation of 16-QAM modulated wireless signals.

		Test Accuracy (%)	Time (min)	Test Accuracy (%)	Time (min)	Test Accuracy (%)	Time (min)	Test Accuracy (%)	Time (min)
Method	**BLO (Solver)**	**16-way-1-shot**		**16-way-5-shot**		**16-way-10-shot**		**16-way-20-shot**	
FO-MAML [14]	AO	94.46 ± 0.35	8.59	97.93 ± 0.11	8.73	99.13 ± 0.13	8.87	99.17 ± 0.04	9.01
Sign-MAML [15]	GU	96.91 ± 0.1	8.1	97.35 ± 0.28	8.14	97.42 ± 0.3	8.23	97.46 ± 0.24	8.32
Vanilla MAML [14]	GU	**98.84** ± 0.1	13.04	**99.49** ± 0.03	13.27	**99.63** ± 0.01	13.29	**99.66** ± 0.02	13.35
iMAML [16]	IF	97.66 ± 0.07	14.14	98.58 ± 0.04	14.52	99.35 ± 0.04	14.81	99.43 ± 0.02	14.88

The best performance in each setting is indicated in boldface. Standard deviations are reported based on five random trials. Rows marked in gray indicate BLO-enabled algorithms.

Formulation

The study of model pruning through the lens of BLO was first explored in [18]. Specifically, there exist two main tasks in model pruning: pruning and retraining. Pruning involves determining the sparse pattern of model weights, while retraining focuses on recovering model accuracy using the remaining nonzero weights [55]. To facilitate these tasks, one can introduce the binary pruning mask variable $\mathbf{m} \in \{0,1\}^m$ and the model weight variable $\boldsymbol{\phi} \in \mathbb{R}^m$, where m represents the total number of model parameters. Accordingly, the pruned model is given by $(\mathbf{m} \odot \boldsymbol{\phi})$, where $\odot$ denotes element-wise multiplication. To achieve a pruning ratio of $p\%$, we impose a sparsity constraint on $\mathbf{m}$, where $\mathbf{m} \in \Omega$ and $\Omega = \{\mathbf{m}, |, \mathbf{m} \in \{0,1\}^n, \mathbf{1}^T\mathbf{m} \le k\}$, with $k = (1-p\%)n$. Our goal is to prune the original dense model to the targeted pruning ratio of $p\%$ and obtain the optimal sparse model $(\mathbf{m} \odot \boldsymbol{\phi})$. To achieve this, we view the pruning task (①) and the model retraining task (②) as two optimization levels, leading to the formulation of bi-level pruning (BiP):

$$\begin{aligned} &\underset{\mathbf{m}\in\Omega}{\text{minimize}}\ \underbrace{\ell_{\text{tr}}(\mathbf{m} \odot \boldsymbol{\phi}^*(\mathbf{m}))}_{\text{①: Pruning task}}; \\ &\text{subject to}\ \underbrace{\boldsymbol{\phi}^*(\mathbf{m}) = \underset{\boldsymbol{\phi}\in\mathbb{R}^n}{\text{argmin}}\ \tilde{\ell}_{\text{tr}}(\mathbf{m} \odot \boldsymbol{\phi}) + \frac{\gamma}{2}\|\boldsymbol{\phi}\|_2^2}_{\text{②: Sparsity-fixed model retraining}}, \end{aligned} \quad \text{(BiP)}$$

where ℓ_{tr} and $\tilde{\ell}_{\text{tr}}$ denote the training losses under different data batches, $\mathbf{m}$ and $\boldsymbol{\phi}$ are the upper-level and lower-level optimization variables, respectively, and $\boldsymbol{\phi}^*(\mathbf{m})$ signifies the retrained model weights given the pruning mask $\mathbf{m}$. In (BiP), the lower-level training objective was regularized using a strongly convex regularizer $\gamma/2\|\boldsymbol{\phi}\|_2^2$ like (Fast-BAT).

Methods

Since BiP is an unconstrained BLO problem, it can be solved using BLO algorithms, e.g., IF and GU, introduced in the sections "IF for Lower-Level Unconstrained BLO" and "GU-Based Approach for Unconstrained BLO." Moreover, since the Hessian of the lower-level objective function with respect to model parameters is of high dimension, we impose the Hessian-free assumption $\nabla_{\boldsymbol{\phi},\boldsymbol{\phi}}\ell_{\text{tr}} = \mathbf{0}$ to make the BLO implementation computationally feasible. Following (4), one can then obtain the closed form of the IG [18]: $d\boldsymbol{\phi}^*(\mathbf{m})/d\mathbf{m} = -(1/\gamma)\nabla^2_{\mathbf{m},\boldsymbol{\phi}}\ell_{\text{tr}}(\mathbf{m} \odot \boldsymbol{\phi}^*)$, where $\boldsymbol{\phi}^*$ signifies a lower-level solution. Furthermore, the bilinearity of the pruning mask $\mathbf{m}$ and the model weights $\boldsymbol{\phi}$ allows us to further simplify the IG to

$$\frac{d\boldsymbol{\phi}^*(\mathbf{m})}{d\mathbf{m}} = -\frac{1}{\gamma}\text{diag}\left(\nabla_{\mathbf{z}}\ell_{\text{tr}}(\mathbf{z})\big|_{\mathbf{z}=\mathbf{m}\odot\boldsymbol{\phi}^*}\right) \tag{19}$$

where the Hessian-free assumption is adopted and $\text{diag}(\mathbf{a})$ denotes the diagonal matrix with $\mathbf{a}$ being the main diagonal vector. A detailed proof can be found in [18, Section 3].

Experiment results

To implement BiP, we adopt two BLO methods: the IF (see the section "IF for Lower-Level Unconstrained BLO") and GU (see the section "GU-Based Approach for Unconstrained BLO"). We term the resulting BLO-inspired model pruning approaches BiP-IF and BiP-GU. For comparison, we also consider two commonly used non-BLO-based pruning methods, the state-of-the-art iterative magnitude pruning (IMP) [55] and the most efficient one-shot magnitude pruning (OMP) [55]. We remark that the notable lottery ticket hypothesis [55] stated that IMP is able to identify a trainable sparse subnetwork (known as a "winning ticket") with a test accuracy surprisingly on par with or even better than that of the original model.

Figure 3 illustrates the pruning accuracy and the runtime efficiency of BLO-based pruning methods versus non-BLO approaches across diverse image classification datasets (including CIFAR-10, CIFAR-100, and Tiny-ImageNet) under ResNet-18. As we can see, BiP-IF yields the best performance in all of the dataset settings shown in Figure 3(a)–(c). This is also the pruning recipe used in [18]. Thanks to the closed-form expression of the IG in (19), the computation of BiP-IF is also more efficient than that of BiP-GU, as shown in Figure 3(d). In addition, BiP-GU can also provide competitive pruning accuracy to IMP and takes less computation time than IMP. Furthermore, we observe that OMP yields the least computation time but the worst pruning accuracy. This is not surprising

Table 6. Performance comparison of different robust training methods using PreActResNet-18 on CIFAR-10 and Tiny-ImageNet datasets.

Method	BLO (Solver)	Standard Accuracy (%) ($\epsilon = 8/255$)	RA-PGD (%) ($\epsilon = 8/255$)	RA-AA (%) ($\epsilon = 8/255$)	Standard Accuracy (%) ($\epsilon = 16/255$)	RA-PGD (%) ($\epsilon = 16/255$)	RA-AA (%) ($\epsilon = 16/255$)	Time (s/epoch)
CIFAR-10, PreActResNet-18								
Fast-AT [61]	N/A	**82.39** ± 0.14	45.49 ± 0.21	41.87 ± 0.15	44.15 ± 7.27	21.83 ± 1.32	12.49 ± 0.33	23.1
Fast-AT-GA [62]		79.71 ± 0.24	47.27 ± 0.22	43.24 ± 0.27	58.29 ± 1.32	26.01 ± 0.16	17.97 ± 0.33	75.3
Fast-BAT-IF [10]	IF	79.97 ± 0.12	**48.83** ± 0.17	**45.19** ± 0.12	**68.16** ± 0.25	**27.69** ± 0.16	**18.79** ± 0.24	61.4
Tiny-ImageNet, PreActResNet-18								
Fast-AT [61]	N/A	41.37 ± 3.08	17.05 ± 3.25	12.31 ± 2.73	31.38 ± 0.19	5.42 ± 2.17	3.13 ± 0.24	284.6
Fast-AT-GA [62]		45.52 ± 0.24	20.39 ± 0.19	16.25 ± 0.17	29.17 ± 0.32	6.79 ± 0.27	4.27 ± 0.15	592.7
Fast-BAT-IF [10]	IF	**45.80** ± 0.22	**21.97** ± 0.21	**17.64** ± 0.15	**33.78** ± 0.23	**8.83** ± 0.22	**5.52** ± 0.14	572.4

The training phase includes adversarial perturbations with two budgets: $\epsilon = 8/255$ and $16/255$ over 20 epochs. Results are presented as mean ± standard deviation over 10 random trials. Rows marked in gray indicate BLO-enabled algorithms. RA: robust accuracy; RA-AA: RA against AutoAttack.

since OMP adopts a noniterative pruning scheme to find the model's sparse pattern.

BLO for invariant representation learning

In this section, we explore the application of BLO in improving the generalization of ML models. Specifically, we investigate the use of BLO for acquiring training environment-agnostic data representations through invariant risk minimization (IRM) [56].

Formulation

IRM [56] is proposed to acquire invariant data representations and to enforce invariant predictions against distribution shifts. Unlike the conventional environment risk minimization (ERM)-based training, IRM yields a BLO-like formulation: the upper-level optimization task of IRM is to train a network backbone to capture environment-agnostic data representations, and the lower-level optimization task is to find an invariant prediction head (on top of the learned representation network) to produce a global optimum to all of the training environments. Formally, IRM can be cast as follows:

$$\begin{aligned} &\underset{\boldsymbol{\theta}}{\text{minimize}} \sum_{i=1}^{E} \ell_i(\boldsymbol{\phi}^*(\boldsymbol{\theta}) \circ \boldsymbol{\theta}) \\ &\text{subject to } \boldsymbol{\phi}^*(\boldsymbol{\theta}) \in \underset{\boldsymbol{\phi}}{\text{argmin}}\, \ell_i(\boldsymbol{\phi} \circ \boldsymbol{\theta}), \forall i \in [E] \end{aligned} \tag{IRM}$$

where $\boldsymbol{\phi} \circ \boldsymbol{\theta}$ denotes the representation–acquisition model $\boldsymbol{\theta}$, the predictor $\boldsymbol{\phi}$, ℓ_i is the training loss associated with the ith training environment, and E is the total number of training environments. The rationale behind (IRM) is that, given the invariant representation extractor $\boldsymbol{\theta}$, there exists an invariant predictor $\boldsymbol{\phi}^*(\boldsymbol{\theta})$ that is optimal across all of the training environments.

Methods

Solving problem (IRM) is highly nontrivial since the lower-level solution $\boldsymbol{\phi}(\boldsymbol{\theta})$ should be *universal* and applied to all E training environments. To circumvent this difficulty, IRM

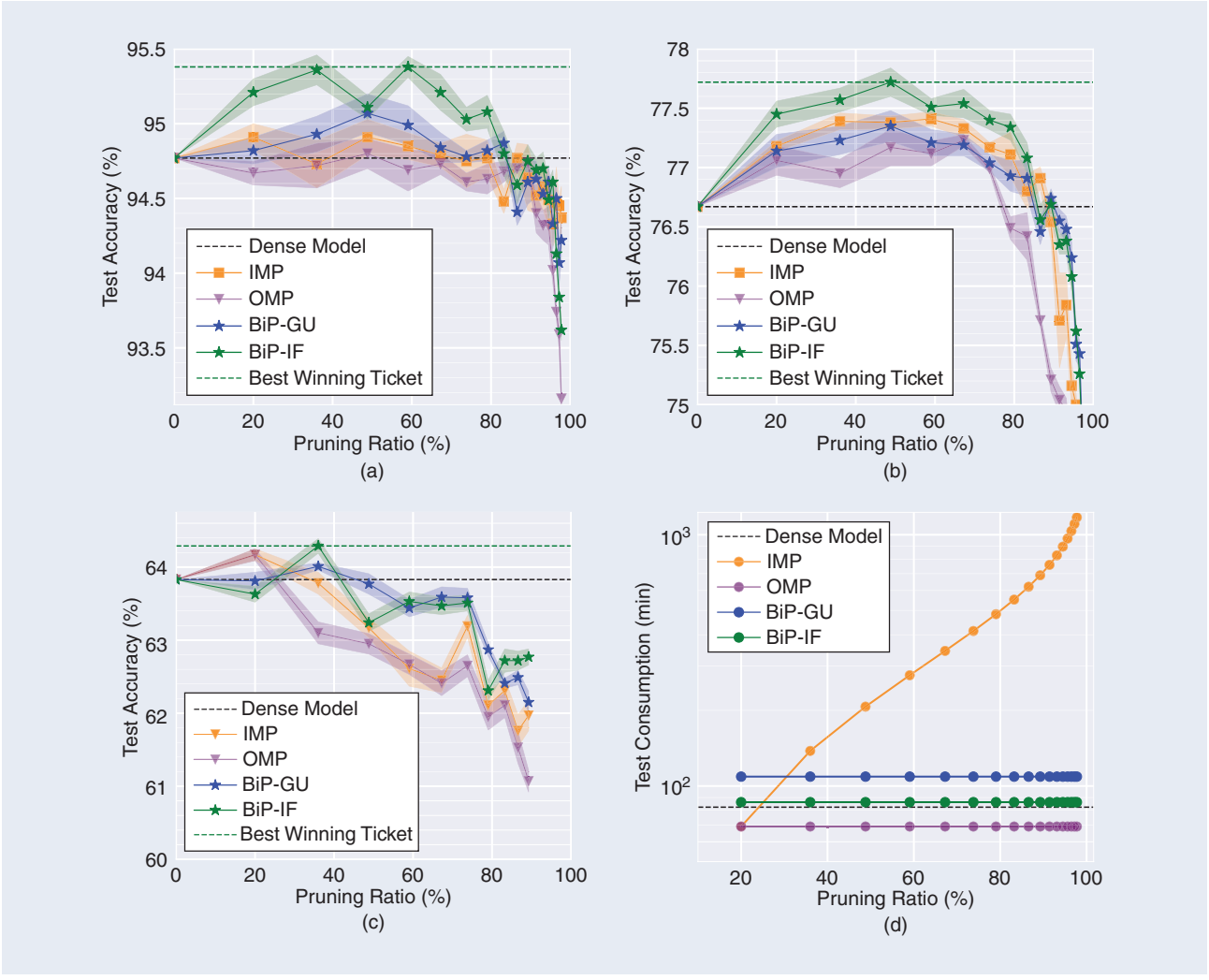


FIGURE 3. Experiment results of model pruning on different datasets under ResNet-18. (a)–(c) Pruning trajectory is given by test accuracy (%) versus sparsity (%) under different datasets. (a) CIFAR-10, (b) CIFAR-100, and (c) Tiny-ImageNet. (d) Efficiency comparison: the entire time consumption versus the pruning ratio.

is typically relaxed to a *single-level* optimization problem, known as IRMv1 [56]:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \sum_{i=1}^{E} \left[\ell_i(\boldsymbol{\theta}) + \gamma \left\| \nabla_{w|w=1.0} \ell_i(w \circ \boldsymbol{\theta}) \right\|_2^2 \right] \tag{20}$$

where $\gamma > 0$ is a regularization parameter and $\nabla_{w|w=1.0} \ell_i(w \odot \boldsymbol{\theta})$ denotes the gradient of ℓ_i with respect to w, computed at $w = 1$. In the preceding formulation, the identity mapping $w = 1$ is adopted, symbolizing a basic "imaginary" classification head. Meanwhile, $\boldsymbol{\theta}$ corresponds to the combination of the representation extractor and the actual invariant predictor. However, the preceding formulation is restricted to linear invariant prediction and penalizes the deviation of individual environment losses from stationarity to approach the lower-level optimality in IRM.

Beyond IRMv1, a consensus-constrained BLO method is developed in [58] to solve problem (IRM). The key idea is to introduce E auxiliary predictors $\{\boldsymbol{\phi}_i\}$ and explicitly enforce prediction invariance by infusing a *consensus prediction constraint* $C = \{\{\boldsymbol{\phi}_i\} \,|\, \boldsymbol{\phi} = \ldots = \boldsymbol{\phi}_E\}$ to the lower-level problem of (IRM) and promote the per-environment stationarity in its upper-level problem. This modifies (IRM) to an ordinary BLO problem:

$$\begin{aligned} &\underset{\boldsymbol{\theta}}{\text{minimize}} \sum_{i=1}^{E} \left[\ell_i(\boldsymbol{\phi}_i^*(\boldsymbol{\theta}) \circ \boldsymbol{\theta}) + \gamma \left\| \nabla_{\phi_i} \ell_i(\boldsymbol{\phi}_i^*(\boldsymbol{\theta}) \circ \boldsymbol{\theta}) \right\|_2^2 \right] \\ &\text{subject to } \{\boldsymbol{\phi}_i^*(\boldsymbol{\theta})\}_{i=1}^{E} = \underset{\{\boldsymbol{\phi}_i\} \in C}{\operatorname{argmin}} \sum_{i=1}^{E} \ell_i(\boldsymbol{\phi}_i \circ \boldsymbol{\theta}), \ \forall i \in [E], \end{aligned} \tag{IRM-BLO}$$

where $\gamma > 0$ is a regularization parameter, and $[E]$ denotes the integer set $\{1, 2, \ldots, E\}$. The advantage of converting (IRM) into the consensus-constrained (IRM-BLO) is that projection onto the consensus constraint yields a closed-form solution, i.e., $\mathcal{P}_C(\mathbf{a}) = \arg\min_{\{\boldsymbol{\phi}_i\} \in C} \sum_{i=1}^{E} \| \boldsymbol{\phi}_i - \mathbf{a}_i \|_2^2 = 1/E \sum_i \mathbf{a}_i$, where $\mathcal{P}_C(\mathbf{a})$ denotes the projection operation to project the point $\mathbf{a}$ onto the constraint C. It has been shown in [58] that problem (IRM-BLO) can be effectively solved using the GU approach, which approximates each individual lower-level solution using K-step GD unrolling together with the consensus projection. Thus, the lower-level solution becomes

$$\forall i, \boldsymbol{\phi}_i^*(\boldsymbol{\theta}) \approx \frac{1}{E} \sum_{i=1}^{E} \boldsymbol{\phi}_i^{(K)}, \ \boldsymbol{\phi}_i^{(k)} = \boldsymbol{\phi}_i^{(k-1)} - \beta \nabla_{\boldsymbol{\phi}_i} \ell_i(\boldsymbol{\phi}_i^{(k-1)} \circ \boldsymbol{\theta}), \quad \text{for } k \in [K] \tag{21}$$

where $\beta > 0$ is the lower-level learning rate. Based on this expression, AD can be called to compute the implicit gradient from $\boldsymbol{\phi}_i^{(K)}(\boldsymbol{\theta})$ to the variable $\boldsymbol{\theta}$ in the GU process.

Experiment results

We evaluate the performance of IRM-BLO with two commonly used image classification datasets, Colored-MNIST [56] and Colored-FashionMNIST [57], where spurious correlation between the image label and the background color is manually imposed in the datasets, which makes the conventional ERM training ineffective. To capture both the accuracy and the variance of invariant predictions across multiple testing environments, the *average accuracy* and the *accuracy gap* (the difference between the best-case and worst-case accuracy) are measured for IRM methods. Table 7 presents the resulting performance of IRM-BLO and compares it with those of ERM and two IRM baselines, IRMv1 [56] and IRM-Game [57]. Note that all IRM variants outperform ERM, which justifies the importance of IRM training to improve model generalization across diverse environments. Within the IRM training family, IRM-BLO outperforms others by achieving the highest average accuracy and the smallest accuracy gap across both datasets. This superior performance underscores the value of BLO compared to the suboptimal design IRMv1.

Table 7. Performance of different IRM training methods.

		Colored-MNIST		Colored-FashionMNIST	
Method	**BLO (Solver)**	**Average Accuracy**	**Accuracy Gap**	**Average Accuracy**	**Accuracy Gap**
ERM	N/A	49.19 ± 1.89	90.72 ± 2.08	49.77 ± 1.71	88.62 ± 2.49
IRMv1 [56]	N/A	68.33 ± 0.31	2.04 ± 0.05	68.76 ± 0.31	1.45 ± 0.09
IRM-Game [57]	N/A	67.73 ± 0.24	1.67 ± 0.14	67.49 ± 0.32	1.82 ± 0.13
IRM-BLO [58]	GU	**69.47** ± 0.24	**1.04** ± 0.07	**69.43** ± 0.21	**1.14** ± 0.11

The best performance per evaluation metric is highlighted in boldface, and the performance of the BLO-enabled method is marked in gray.

Discussion

BLO is a challenging but rapidly developing subject. Despite the recent progress discussed in this article, significant work is yet to be done to address various challenges, ranging from developing scalable algorithms to extending the applicability of BLO to a wider range of problems. Here we highlight several worthwhile future directions.

- *BLO algorithms*: First, the development and analysis of BLO algorithms for more general BLO problems, including those with complex lower-level constraints (e.g., nonlinear constraints), require additional exploration. The current focus has predominantly been on problems with linear constraints, and extending the BLO framework to handle nonlinear constraints is an important and challenging task. Additionally, exploring scenarios with coupled constraints between lower and upper levels, such as resource sharing among adversarial and normal agents, presents a complex area that is underexplored. Second, BLO formulations with NS lower-level solutions, such as (NS-BLO), lack theoretically grounded, scalable, and easy-to-implement algorithms. This aspect of BLO has received less attention from the community, making it an open topic for future investigation. Developing efficient algorithms that can handle NS lower-level solutions and provide convergence guarantees is a key research direction. Third, beyond the scope of BLO, exploring problems involving more than two levels, such as dataset pruning

for TL, represents an exciting frontier. These multilevel problems introduce additional complexity and challenges, requiring the design of novel algorithms to tackle the inherent hierarchical structure effectively. Finally, the large scale and distributed availability of data demand the development of decentralized and federated algorithms to solve these complex BLO problems, which also presents a compelling research direction for future exploration.

- *BLO theories*: First, while significant progress has been made in establishing theoretical guarantees for solving the basic BLO problem (LU-BLO), more attention is needed on exploring practical settings, including (coupled) lower-level constraints, nonconvex lower-level problems, and/or black-box settings, where one may not have access to upper-/lower-level parameters. These scenarios present unique challenges and complexities, making it difficult to analyze the problem and derive theoretical guarantees. Investigating the convergence properties and establishing theoretical foundations for solving BLO problems under these practical settings is an important avenue for future research. Second, when datasets become massive, it is crucial to develop theoretically grounded BLO algorithms that can adhere to practical requirements. Therefore, developing theoretical frameworks and analyzing the convergence properties of algorithms for solving large-scale BLO problems under realistic assumptions is also an important research topic. Finally, with the discovery of the phenomenon of double descent, theoretical analysis of standard ML algorithms on overparameterized neural networks has received significant attention from the research community. Theoretical investigation of BLO algorithms for such overparameterized problems is certainly an interesting research direction.
- *BLO applications*: First, in the context of mixture-of-experts (MoE) training, there is a complex interplay between the training of the gating network that selects experts and the training of the expert-oriented pathways used for final predictions. Exploring BLO techniques to effectively optimize the coupling between these two processes in MoE training can lead to improved performance and better utilization of emerging ML models like MoE. Second, prompt learning, a key technique used in today's foundation models, involves a crucial coupling between prompt pattern learning and label/feature mapping optimization. Leveraging BLO methods to model and optimize the interactions between prompt pattern learning and label/feature mapping can enhance the learning process and enable accurate and robust prompt generations. Third, BLO is highly applicable in (inverse) reinforcement learning. For instance, the actor/critic algorithm can be formulated as a BLO problem, with separate agents evaluating and optimizing the policy. In *inverse* reinforcement learning, the tasks involve inferring the agents' reward function and finding the optimal policy based on it. Applying BLO frameworks to these scenarios offers potential for novel insights and improved efficiency.

In summary, the interplay between the theoretical underpinnings of BLO and its practical applications promises a fertile ground for future exploration and innovation, pushing the boundaries of optimization theory and applications in SP and ML.

Authors

Yihua Zhang (zhan1908@msu.edu) received his bachelor's degree in automation from Huazhong University of Science and Technology. He is with the Computer Science and Engineering Department, Michigan State University, East Lansing, MI 48824-1312 USA. He is a Graduate Student Member of IEEE

Prashant Khanduri (khanduri.prashant@wayne.edu) received his Ph.D. degree in electrical engineering and computer science from Syracuse University. He is with the Computer Science Department, Wayne State University, Detroit, MI 48202 USA. He is a Member of IEEE.

Ioannis Tsaknakis (tsakn001@umn.edu) received his Ph.D. degree in electrical and computer engineering from the University of Minnesota. He is with the Department of Electrical and Computer Engineering, University of Minnesota, MN 55455 USA. He is a Graduate Student Member of IEEE.

Yuguang Yao (yaoyugua@msu.edu) received his bachelor's degree in automation from Tsinghua University. He is with the Computer Science and Engineering Department, Michigan State University, East Lansing, MI 48824-1312 USA. He is a Graduate Student Member of IEEE.

Mingyi Hong (mhong@umn.edu) received his Ph.D. degree in systems engineering from the University of Virginia. He is with the Department of Electrical and Computer Engineering, University of Minnesota, MN 55455 USA. He is a Senior Member of IEEE.

Sijia Liu (liusiji5@msu.edu) received his Ph.D. degree in electrical and computer engineering from Syracuse University. He is with the Computer Science and Engineering Department, Michigan State University, East Lansing, MI 48824-1312 USA, and the Massachusetts Institute of Technology-IBM Watson Artificial Intelligence Lab, Cambridge, MA, USA. He is a Senior Member of IEEE.

References

[1] S. Dempe, "Bilevel optimization: Theory, algorithms, applications and a bibliography," *Bilevel Optimization: Advances and Next Challenges*, S. Dempe and A. Zemkoho, Eds., Cham, Switzerland: Springer-Verlag, 2020, pp. 581–672.

[2] H. V. Stackelberg, *The Theory of the Market Economy.* London, U.K.: Oxford Univ. Press, 1952.

[3] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Ann. Oper. Res.*, vol. 153, pp. 235–256, Sep. 2007, doi: 10.1007/s10479-007-0176-2.

[4] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Trans. Evol. Comput.*, vol. 22, no. 2, pp. 276–295, Apr. 2018, doi: 10.1109/TEVC.2017.2712906.

[5] Y. Beck, I. Ljubić, and M. Schmidt, "A survey on bilevel optimization under uncertainty," *Eur. J. Oper. Res.*, vol. 311, no. 2, pp. 401–426, Dec. 2023, doi: 10.1016/j.ejor.2023.01.008.

[6] Y. Beck, D. Bienstock, M. Schmidt, and J. Thürauf, "On a computationally Ill-behaved bilevel problem with a continuous and nonconvex lower level," *J. Optim. Theory Appl.,* vol. 198, pp. 428–447, Jul. 2023, doi: 10.1007/s10957-023-02238-9.

[7] H. Sun, W. Pu, X. Fu, T. H. Chang, and M. Hong, "Learning to continuously optimize wireless resource in a dynamic environment: A bilevel optimization perspective," *IEEE Trans. Signal Process.*, vol. 70, pp. 1900–1917, Jan. 2022, doi: 10.1109/TSP.2022.3143372.

[8] S. Park, H. Jang, O. Simeone, and J. Kang, "Learning to demodulate from few pilots via offline and online meta-learning," *IEEE Trans. Signal Process.*, vol. 69, pp. 226–239, 2021, doi: 10.1109/TSP.2020.3043879.

[9] C. Crockett and J. A. Fessler, "Bilevel methods for image reconstruction," *Found. Trends Signal Process.*, vol. 15, nos. 2–3, pp. 121–289, 2022, doi: 10.1561/2000000111.

[10] Y. Zhang, G. Zhang, P. Khanduri, M. Hong, S. Chang, and S. Liu, "Revisiting and advancing fast adversarial training through the lens of bi-level optimization," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 26,693–26,712.

[11] A. Robey, F. Latorre, G. J. Pappas, H. Hassani, and V. Cevher, "Adversarial training should be cast as a non-zero-sum game," 2023, *arXiv:2306.11035.*

[12] M. Zhao, B. An, Y. Yu, S. Liu, and S. Pan, "Data poisoning attacks on multi-task relationship learning," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–8, doi: 10.1609/aaai.v32i1.11838.

[13] W. R. Huang, J. Geiping, L. Fowl, G. Taylor, and T. Goldstein, "MetaPoison: Practical general-purpose clean-label data poisoning," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 12,080–12,091, doi: 10.5555/3495724.3496737.

[14] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135, doi: 10.5555/3305381.3305498.

[15] C. Fan, P. Ram, and S. Liu, "Sign-MAML: Efficient model-agnostic meta-learning by signSGD," in *Proc. 5th Workshop Meta-Learn. Conf. Neural Inf. Process. Syst.*, 2021.

[16] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 1–12.

[17] Z. Borsos, M. Mutny, and A. Krause, "Coresets via bilevel optimization for continual learning and streaming," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, 2020, pp. 1–12.

[18] Y. Zhang, Y. Yao, P. Ram, P. Zhao, T. Chen, M. Hong, Y. Wang, and S. Liu, "Advancing model pruning via bi-level optimization," in *Proc. 36th Conf. Neural Inf. Process. Syst.*, 2022, pp. 1–18.

[19] C. Chen, X. Chen, C. Ma, Z. Liu, and X. Liu, "Gradient-based bi-level optimization for deep learning: A survey," 2022, *arXiv:2207.11719.*

[20] C. Holtz, T.-W. Weng, and G. Mishne, "Learning sample reweighting for accuracy and adversarial robustness," 2022, *arXiv:2210.11513.*

[21] R. Liu, P. Mu, X. Yuan, S. Zeng, and J. Zhang, "A generic first-order algorithmic framework for bi-level programming beyond lower-level singleton," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 6305–6315, doi: 10.5555/3524938.3525523.

[22] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots, "Truncated back-propagation for bilevel optimization," in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2019, pp. 1723–1732.

[23] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1568–1577.

[24] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1165–1173, doi: 10.5555/3305381.3305502.

[25] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *Proc. Int. Conf. Learn. Representations*, 2018.

[26] R. Liu, J. Gao, J. Zhang, D. Meng, and Z. Lin, "Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 10,045–10,067, Dec. 1, 2022, doi: 10.1109/TPAMI.2021.3132674.

[27] S. G. Krantz and H. R. Parks, *The Implicit Function Theorem: History, Theory, and Applications.* Berlin, Germany: Springer Science & Business Media, 2002.

[28] J. L. Nazareth, "Conjugate gradient method," *Wiley Interdisciplinary Rev., Comput. Statist.*, vol. 1, no. 3, pp. 348–353, 2009, doi: 10.1002/wics.13.

[29] S. P. Singh and D. Alistarh, "WoodFisher: Efficient second-order approximation for neural network compression," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 18,098–18,109, doi: 10.5555/3495724.3497243.

[30] S. Ghadimi and M. Wang, "Approximation methods for bilevel programming," 2018, *arXiv:1802.02246.*

[31] M. Hong, H.-T. Wai, Z. Wang, and Z. Yang, "A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic," *SIAM J. Optim.*, vol. 33, no. 1, pp. 147–180, 2023, doi: 10.1137/20M1387341.

[32] P. Khanduri, I. Tsaknakis, Y. Zhang, J. Liu, S. Liu, J. Zhang, and M. Hong, "Linearly constrained bilevel optimization: A smoothed implicit gradient approach," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 16,291–16,325.

[33] D. Sow, K. Ji, Ziwei Guan, and Y. Liang, "A constrained optimization approach to bilevel optimization with multiple inner minima," 2022, *arXiv:2203.01123.*

[34] H. Shen and T. Chen, "On penalty-based bilevel gradient descent method," 2023, *arXiv:2302.05185.*

[35] R. Liu, X. Liu, X. Yuan, S. Zeng, and J. Zhang, "A value-function-based interior-point method for non-convex bi-level optimization," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 6882–6892.

[36] B. Liu, M. Ye, S. Wright, P. Stone, and Q. Liu, "Bome! Bilevel optimization made easy: A simple first-order approach," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 17,248–17,262.

[37] Q. Xiao, H. Shen, W. Yin, and T. Chen, "Alternating projected SGD for equality-constrained bilevel optimization," in *Proc. 26th Int. Conf. Artif. Intel. Statist.*, 2023, pp. 987–1023.

[38] T. Chen, Y. Sun, Q. Xiao, and W. Yin, "A single-timescale method for stochastic bilevel optimization," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2022, pp. 2466–2488.

[39] M. Arbel and J. Mairal, "Amortized implicit differentiation for stochastic bilevel optimization," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–39.

[40] K. Ji, J. Yang, and Y. Liang, "Bilevel optimization: Convergence analysis and enhanced design," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 4882–4892.

[41] P. Khanduri, S. Zeng, M. Hong, H.-T. Wai, Z. Wang, and Z. Yang, "A near-optimal algorithm for stochastic bilevel optimization via double-momentum," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 30,271–30,283.

[42] T. Chen, Y. Sun, and W. Yin, "Tighter analysis of alternating stochastic gradient method for stochastic nested problems," 2021, *arXiv:2106.13781.*

[43] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM J. Optim.*, vol. 23, no. 4, pp. 2341–2368, 2013, doi: 10.1137/120880811.

[44] A. Cutkosky and F. Orabona, "Momentum-based variance reduction in non-convex SGD," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 15,236–15,245, doi: 10.5555/3454287.3455652.

[45] J. Yang, K. Ji, and Y. Liang, "Provably faster algorithms for bilevel optimization," in *Proc. 35th Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 1–13.

[46] M. Dagréou, P. Ablin, S. Vaiter, and T. Moreau, "A framework for bilevel optimization that enables stochastic and global variance reduction algorithms," in *Proc. 36th Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 1–13.

[47] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 2206–2216, doi: 10.5555/3524938.3525144.

[48] J. Kwon, D. Kwon, S. Wright, and R. Nowak, "A fully first-order method for stochastic bilevel optimization," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 18,083–18,113.

[49] Z. Akhtar, A. Singh Bedi, S. Teja Thomdapu, and K. Rajawat, "Projection-free algorithm for stochastic bi-level optimization," 2021, *arXiv:2110.11721.*

[50] Z. Guo, Q. Hu, L. Zhang, and T. Yang, "Randomized stochastic variance-reduced methods for multi-task stochastic bilevel optimization," 2021, *arXiv:2105.02266.*

[51] C. Fang, C. J. Li, Z. Lin, and T. Zhang, "SPIDER: Near-optimal non-convex optimization via stochastic path-integrated differential estimator," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 1–11.

[52] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 1–9.

[53] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 4, pp. 648–664, Dec. 2018, doi: 10.1109/TCCN.2018.2881442.

[54] M. Cicerone, O. Simeone, and U. Spagnolini, "Channel estimation for MIMO-OFDM systems by modal analysis/filtering," *IEEE Trans. Commun.*, vol. 54, no. 11, pp. 2062–2074, Nov. 2006, doi: 10.1109/TCOMM.2006.884849.

[55] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–42.

[56] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, "Invariant risk minimization," 2019, *arXiv:1907.02893.*

[57] K. Ahuja, K. Shanmugam, K. Varshney, and A. Dhurandhar, "Invariant risk minimization games," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 145–155.

[58] Y. Zhang, P. Sharma, P. Ram, M. Hong, K. Varshney, and S. Liu, "What is missing in IRM training and evaluation? Challenges and solutions," in *Proc. Int. Conf. Learn. Representations*, 2023.

[59] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Netw.*, vol. 113, pp. 54–71, May 2019, doi: 10.1016/j.neunet.2019.01.012.

[60] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–23.

[61] E. Wong, L. Rice, and J. Zico Kolter, "Fast is better than free: Revisiting adversarial training," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–23.

[62] M. Andriushchenko and N. Flammarion, "Understanding and improving fast adversarial training," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, 2020, pp. 16,048–16,059.

SP