2024 年 5 月 计 算 数 学 第 46 卷第 2 期
May, 2024 MATHEMATICA NUMERICA SINICA Vol.46, No.2

doi: 10.12286/jssx.j2024-1170

青年评述

机器学习中的双层优化算法简介*

马士谦[1)]

(莱斯大学计算应用数学与运筹学系, 休斯顿 TX 77005, 美国)

摘 要

双层优化是近年来的一个热门研究方向. 这主要归功于机器学习的兴起和双层优化在机器学习中的许多重要应用. 本文对双层优化的算法、理论及应用最近几年的发展做一个简要的介绍. 内容主要包括双层优化的历史, 双层优化在电力系统, 超参优化, 元学习等领域的应用, 以及双层优化的算法设计和理论保证. 算法方面我们主要分两种情况: 下层问题是强凸问题和下层问题是一般凸问题. 这里我们会讨论梯度法和基于下层最优函数的方法. 我们也会重点讨论分布式网络中的双层优化, 包括去中心化的双层优化和联邦双层优化的算法和理论分析.

关键词: 双层优化; 一阶算法; 超参优化; 元学习; 去中心化计算; 联邦学习.

MR (2010) 主题分类: 90C30.

1. 引 言

双层优化是近年来优化领域的一个热门方向. 本文对机器学习中的乐观型双层优化做一个简要的介绍, 主要包括双层优化的历史, 在机器学习领域的应用, 算法设计和理论分析, 分布式网络中的双层优化问题等.

双层优化是指一个优化问题的约束集合是由另一个优化问题的最优解集来定义的. 也就是说, 在这个优化问题的约束里, 有另一个优化问题. 我们一般把约束中的这个优化问题称为下层问题, 把目标函数中的优化问题称为上层问题. 双层优化问题具有下列形式:

$$\min_{x\in\mathbb{R}^p} \Phi(x) = F(x, y^*(x)),\ \text{s.t.}, y^*(x) \in \operatorname*{argmin}_{y\in\mathbb{R}^q} f(x, y). \tag{1.1}$$

这里为简单起见, 我们假设上层和下层问题都是无约束问题. 并且我们假设 F 是一阶连续可微函数, f 是二阶连续可微函数. 这些假设是为了后面讨论算法的需要. 由于文献中一般假

* 2024 年 1 月 2 日收到.

[1)] 作者简介: 马士谦, 莱斯大学计算应用数学与运筹学系副教授. 他于北京大学数学科学学院获得学士学位, 中国科学院数学与系统科学研究院计算数学与科学工程计算研究所获得硕士学位, 哥伦比亚大学工业工程与运筹学系获得博士学位. 研究方向主要包括大规模优化问题的理论和算法, 及其在机器学习和深度学习中的应用. 2010 年获得 INFORMS Optimization Society Best Student Paper Prize, 2011 年获得 honorable mention in the INFORMS George Nicholson Student Paper Competition, 2016 年获得 Journal of the Operations Research Society of China Excellent Paper Award, 2018 年获得 Excellence Award for Visiting Scholar at Tencent AI Lab, 2023 年获得 SIAM Review SIGEST Award. 现担任 Journal of Machine Learning Research, Journal of Scientific Computing 和 Journal of Optimization Theory and Applications 编委, 并任 ICML, NeurIPS, ICLR 和 AISTATS 的领域主席. 马士谦教授为 2023 年 Texas Colloquium on Distributed Learning 大会报告人, 现为 Secretary/Treasurer of the INFORMS Optimization Society, 并任 2024 年 INFORMS Optimization Society Conference 大会主席.

设 F 是非凸函数, 所以 Φ 是非凸函数, 算法设计的目标是寻找一个近似稳定点, 即满足条件 $\|\nabla\Phi(x)\|_2 \le \epsilon$ 的点 x.

最早的双层优化问题见于 Stackelberg 1952 年所著 "Theory of the Market Economy" 一书 [45]. 书中用双层优化描述了一个经典的经济学中的博弈问题. 在这个问题中, 有一个 "Leader", 有一个 "Follower". Leader 先做决策 (在作出决策之前, Leader 知道 Follower 会观测到自己的决策), 然后 Follower 根据 Leader 的决策作出对自己最有利的决策. Leader 和 Follower 有自己不同的目标函数. 这样的问题可以用一个双层优化来刻画: 上层问题对应于 Leader 的决策问题, 下层问题对应于 Follower 的决策问题. 双层优化第一次出现在运筹学的领域是 1973 年 Bracken 和 McGill 的文章 [4]. 这篇文章用双层优化描述了一个军事领域的运筹学问题. 假设有红方和蓝方在交战. 红方要将自己的武器部署在一些指定的位置, 这些位置可以打击到蓝方的武器和物资. 所以红方的决策问题是将自己的武器部署在哪里和分别部署多少, 决策的目标是减小成本. 蓝方要做的决策是如何分布自己的武器和物资, 以将损失减少到最小.

2. 双层优化的应用

双层优化一个经典的应用场景, 是用于电网和电力系统中的决策问题. 双层优化在最近几年引起了更加广泛的关注, 是因为在机器学习领域发现了很多非常重要的应用. 下列是一些非常有影响力的工作, 比如超参优化 [14, 40], 元学习 [3, 41, 43], 以及强化学习 [14, 19]. 这里我们重点介绍超参优化和元学习两个应用.

2.1. 电力定价系统中的应用

电力系统的定价问题是双层优化的一个重要应用领域. 在欧美的电力市场中, 有很多家电力公司可以给用户供电. 每个公司都有各种不同的电价合约可供选择. 用户可以根据自己的实际需要 (比如用电量的大小) 来选择对自己最有利的电力公司和电价合约. 这个问题就是一个典型的双层优化问题. 这里电力公司是 "Leader", 用户是 "Follower". 电力公司要做的决策问题是如何制定各种不同的电价合约从而保持在市场中的竞争力, 而用户面临的决策问题是选择哪家公司和哪种合约从而在满足自己用电量需要的前提下花费最小.

2.2. 超参优化

很多统计学模型和机器学习模型都有超参数. 比如统计学中的 LASSO 模型

$$\min_{\beta} \sum_{i=1}^{m} (\beta^\top a^i - b_i)^2 + \mu\|\beta\|_1. \tag{2.1}$$

这里 (a^i, b_i), $i = 1, \ldots, m$ 是输入数据. 还有机器学习中的支撑向量机模型

$$\min_{w,c} \sum_{i=1}^{m} \max\{1 - b_i(w^\top a^i - c), 0\} + \mu\|w\|_2^2. \tag{2.2}$$

在 (2.1) 和 (2.2) 中, μ 都是超参数. 如何选取这个超参数在实际应用中非常的重要. 统计学和机器学习中常用的选取超参数的方法是 K-fold 交叉验证. 以 LASSO 模型为例, K-fold 交叉

验证的具体做法是把训练数据集 $D := \{(a^i, b_i)\}_{i=1}^m$ 分成 K 份 $D_k := \{(a^i, b_i)\}_{i\in T_k}$, 这里 T_k, $k = 1, \ldots, K$ 是指标集合 $T := \{1, \ldots, m\}$ 的一个划分. 对于一个固定的 μ 和每一个 k, 我们用 $D\backslash D_k$ 作为训练数据集求解训练模型 (2.1). 我们把得到的解记作 $\beta^k(\mu)$. 这样对于每一个 $k = 1, \ldots, K$ 做了同样的步骤以后, 我们就可以计算交叉验证误差:

$$CV(\mu) := \frac{1}{K}\sum_{k=1}^{K}\sum_{i\in T_k}\ell(\beta^k(\mu), a^i, b_i).$$

这里我们用 ℓ 来表示 LASSO 模型中的目标函数. 通过计算很多不同的 μ 所对应的误差 $CV(\mu)$, 我们最终可以选择最小的误差所对应的 μ 来作为最终的超参数. 可以看出, 这样的方法有两个缺点: (i) 我们需要求解 K 次 LASSO 模型. 这在 K 很大时会很花时间. (ii) 最终选取到的超参数不一定是最优的, 因为这个参数是从我们预先给定的 μ 的集合里面选出来的. 双层优化可以有效的解决这两个问题. 文献中已经有一系列的工作讨论如何用双层优化选择支撑向量机模型中的超参数 [1, 2, 24, 26, 27]. 下面我们用 LASSO 问题为例来讨论如何用双层优化做超参优化. LASSO 问题中的超参选择可以用下面的双层优化来解决:

$$\begin{aligned}\min_{\mu,\beta^1,\ldots,\beta^K}\ & \frac{1}{K}\sum_{k=1}^{K}\sum_{i\in T_k}\ell(\beta^k(\mu), a^i, b_i)\\ \text{s.t.},\ & \beta^k \in \operatorname*{argmin}_{z^k}\sum_{i\notin T_k}\ell(z^k, a^i, b_i) + \mu\|z^k\|_1, \quad \forall k = 1, \ldots, K.\end{aligned}$$

也可以等价的写成下面的形式:

$$\begin{aligned}\min_{\mu,\beta^1,\ldots,\beta^K}\ & \frac{1}{K}\sum_{k=1}^{K}\sum_{i\in T_k}\ell(\beta^k(\mu), a^i, b_i)\\ \text{s.t.},\ & (\beta^1, \ldots, \beta^K) \in \operatorname*{argmin}_{(z^1,\ldots,z^K)}\sum_{k=1}^{K}\sum_{i\notin T_k}\ell(z^k, a^i, b_i) + \mu\|z^k\|_1.\end{aligned}$$

这样的好处是, 只需求解一个双层优化问题就可以找到最优的超参数 μ.

这方面的一些最新进展参见 [5, 16, 28, 51].

2.3. 元学习

元学习是机器学习中的一个新兴课题. 元学习涵盖的范围非常宽泛. 这里重点介绍元学习中的一个基础任务: 超表示学习. 假设有 M 个元学习的任务: $\{\mathcal{T}_i\}_{i=1}^M$. 每一个任务 $\mathcal{T}_i$ 都有一组训练数据 D_i^{tr} 和一组测试数据 D_i^{test}. 元学习的目标是通过这些数据集对每个任务 $\mathcal{T}_i$ 学习它的模型参数 w_i, 与此同时学习所有任务共有的超表示 λ. 这个 λ 可以是神经网络的参数, 可以是求解任务所用的算法里面的参数, 也可以是用来描述所有模型共有的一些特征的参数. 通过元学习得到的 w_i 和 λ, 可以用来很好的处理类似的任务. 这个元学习的例子可以用下面的双层优化来描述:

$$\begin{aligned}\min_{\lambda}\ & \sum_{i=1}^{M}\mathcal{L}(w_i^*(\lambda), \lambda, D_i^{test})\\ \text{s.t.}\ & w^*(\lambda) \in \operatorname*{argmin}_{w}\sum_{i=1}^{M}\mathcal{L}(w_i, \lambda, D_i^{tr}).\end{aligned}$$

这里我们用 $\mathcal{L}$ 来表示训练损失函数.

3. 双层优化算法

近年来, 由于双层优化被成功应用于机器学习领域, 其相关的算法研究也引起了广泛的关注. 我们主要讨论两类双层优化问题的算法：下层问题是强凸优化问题, 和下层问题是一般凸优化问题. 由于近几年双层优化的算法主要应用于机器学习领域, 文献中一般考虑目标函数 F 和 f 是期望或者 finite-sum 的情况. 也就是说, (1.1) 中的 F 和 f 分别可以写成

$$F(x,y) := \mathbb{E}_\xi \bar{F}(x,y,\xi), \quad f(x,y) := \mathbb{E}_\zeta \bar{f}(x,y,\zeta), \tag{3.1}$$

或者

$$F(x,y) := \sum_{i=1}^m F_i(x,y), \quad f(x,y) := \sum_{i=1}^m f_i(x,y). \tag{3.2}$$

这里 m 可以看成是样本数据的个数. 因为计算梯度很困难, 对于这样的问题设计的算法都是随机算法. 也就是用随机梯度代替梯度. 所以, 在下面的讨论中, 如果涉及到样本采样复杂度, 方差等等概念, 我们默认讨论的是 (3.1) 或者 (3.2) 的情况.

3.1. 双层优化的等价形式：单层约束优化

因为双层优化问题是一个特殊的约束优化问题 (约束里面有另一个优化问题), 一个很自然的想法是, 是否可以把双层优化当作一个单层的约束优化问题来求解? 文献中有两种常用的方法把双层优化转化成单层优化问题. 一种是将下层的优化问题用它的 KKT 条件来替代, 从而把原来的双层优化问题转化成带约束的单层优化问题. 但是由于非线性约束的存在, 这样的单层优化问题有可能仍然不容易求解. 在这一节我们主要介绍下面的 value-function-based 方法. 这要求把双层优化 (1.1) 等价的写成下面的形式：

$$\min_{x,y} \ F(x,y), \quad \text{s.t.}, f(x,y) \le f^*(x). \tag{3.3}$$

这里 $f^*(x)$ 的定义是 $f^*(x) := \min_y f(x,y)$. 容易看出, 如果 (x,y) 满足 (3.3) 的约束条件, 那么 y 必须是 $\min_y f(x,y)$ 的最优解. 但是因为 $f^*(x)$ 有可能是非凸并且非光滑的, 所以求解 (3.3) 仍然是不容易的. 不过这个单层约束优化还是非常有用, 可以帮助我们设计一些其他的算法来求解双层优化问题. 这个我们在后面会仔细讨论.

3.2. 下层问题是强凸优化问题

最早的关于双层优化算法迭代复杂度的分析是 Ghadimi 和 Wang 2018 年在 arxiv 上面的预印本文章 [17]. 这个文章分析了下层问题是强凸问题时, 梯度算法求解双层优化问题 (1.1) 的算法迭代复杂度. 用梯度法求解 (1.1) 最自然的想法是用迭代公式：

$$x^{k+1} := x^k - \tau_{x,k} \nabla \Phi(x^k). \tag{3.4}$$

这里 $\tau_{x,k} > 0$ 是步长, $\nabla\Phi(x)$ 是上层函数 F 的超梯度. 因为 $y^*(x)$ 也是 x 的函数, 所以这里是用超梯度, 不同于传统的梯度. 这个超梯度 $\nabla\Phi(x)$ 有一个显式的计算公式如下：

$$\nabla\Phi(x) := \nabla_1 F(x,y^*(x)) - \nabla^2_{12} f(x,y^*(x))[\nabla^2_{22} f(x,y^*(x))]^{-1} \nabla_2 F(x,y^*(x)). \tag{3.5}$$

这里我们需要指出这个计算公式要用到隐函数定理. 该定理要求下层问题的最优解必须唯一. 这是我们要求下层问题强凸的原因. 所以, 如果我们能得到 $y^*(x^k)$ 的一个很好的近似, 那么就可以得到关于超梯度 $\nabla\Phi(x^k)$ 的一个很好的近似, 从而可以应用 (3.4) 来进行算法迭代. 那么如何得到 $y^*(x^k)$ 的一个很好的近似呢? 我们可以用梯度法来近似求解 $x = x^k$ 时的下层问题. 所以, 上层问题和下层问题都用梯度法求解的算法可以概括如下:

$$\begin{aligned}
&\text{for } k = 0, 1, \ldots, K-1\\
&\quad y^{k,0} = y^{k-1,T}\\
&\quad \text{for } t = 0, 1, \ldots, T-1\\
&\qquad y^{k,t+1} = y^{k,t} - \tau_{y,k}\nabla_2 f(x^k, y^{k,t}),\\
&\quad x^{k+1} = x^k - \tau_{x,k}\widetilde{\nabla\Phi}(x^k),
\end{aligned} \tag{3.6}$$

这里 $\widetilde{\nabla\Phi}(x^k)$ 是超梯度 $\nabla\Phi(x^k)$ 的近似, 其计算公式如下:

$$\widetilde{\nabla\Phi}(x^k) = \nabla_1 F(x^k, y^{k,T}) - \nabla^2_{12} f(x^k, y^{k,T})[\nabla^2_{22} f(x^k, y^{k,T})]^{-1}\nabla_2 F(x^k, y^{k,T}). \tag{3.7}$$

这个算法有两个显著的缺点和难点: (i) 两层循环. 可以看到, 每次 x^k 更新的时候, 总是要等下层问题运行很多次迭代. 这在实际中可能不是一个好的选择. (ii) 计算超梯度 (3.7) 的时候要求解一个方程组. 这在实际中可能非常的费时. 有很多工作着眼于解决这两个问题. 我们下面介绍这方面的一些代表工作. Hong et al.的文章 [19] 提出了 TTSA(Two-Timescale Stochastic Algorithm) 算法. TTSA 通过下面的迭代公式更新迭代点:

$$y^{k+1} := y^k - \beta_k h^k_f \tag{3.8a}$$

$$x^{k+1} := x^k - \alpha_k h^k_F. \tag{3.8b}$$

这里 h^k_f 和 h^k_F 分别是 $\nabla_2 f(x^k, y^k)$ 和超梯度 $\nabla\Phi(x^k)$ 的近似. 这个算法的优势在于 x 和 y 可以同步更新, 而且是一个单循环的算法. 如果我们选取步长 α_k, β_k 使得 $\lim_k \frac{\alpha_k}{\beta_k} = 0$, 再加上其他一些假设, 那么就可以证明 TTSA 的收敛性和收敛速度. 这里 $\lim_k \frac{\alpha_k}{\beta_k} = 0$ 当 α_k 和 β_k 都趋向于 0 时, 保证 α_k 收敛的速度更快. 也就是说 β_k 要远大于 α_k. 这样的步长的设置保证了 y^k 比 x^k 要收敛的更快. 这也符合梯度方法 (3.6) 的想法: 精确求解 y^k 以后再通过一次梯度步来更新 x^k. Chen et al. [7] 提出了另外一种单循环的算法. 该算法提升了 TTSA 的样本采样的复杂度, 不过代价是每次迭代的运算量有所提供. 所以该算法适用于采样比较花时间的情况. 关于如何处理超梯度中的方程组求解问题, 有两种常用的方法: approximate implicit differentiation (AID) 和 iterative differentiation (ITD). Ji et al.的文章 [22] 分析了基于 AID 和 ITD 的梯度方法的迭代复杂度分析, 其给出的结果改进了 [17] 和 [19] 的结果. 一些后续的工作 [18, 23, 47] 引入了 momentum 和方差缩减的技巧, 进一步加速和提升了这类求解双层优化问题的算法的收敛速度.

这里我们指出, 上述算法都没有很好的解决超梯度 (3.5) 计算中的方程组求解问题. AID 和 ITD 本质上都是应用迭代算法来近似的求解这个方程组. 这样的操作有可能会很费时. 针对这个问题, Dagreou et al. [11] 提出了 SOBA 算法. 该算法提出用如下的步骤来处理方程组求解问题. 首先, 求解方程组 $[\nabla^2_{22} f(x,y)]^{-1}\nabla_2 F(x,y)$ 等价于极小化一个二次函数, 即求解一个无约束的二次规划问题

$$\min_v \frac{1}{2} v^\top \nabla^2_{22} f(x,y) v - \nabla_2 F(x,y)^\top v. \tag{3.9}$$

这个二次规划问题可以用梯度法来迭代求解. 但是这样可能很费时. SOBA 算法提出只用一次梯度步来近似这个二次规划问题的解. 求解 (1.1) 的 SOBA 算法第 k 次迭代的公式可以描述如下：

$$y^{k+1} := y^k - \beta_k D_y^k, \tag{3.10a}$$

$$v^{k+1} := v^k - \eta_k D_v^k, \tag{3.10b}$$

$$x^{k+1} := x^k - \alpha_k D_x^k. \tag{3.10c}$$

这里的 α_k, β_k, η_k 都是步长, $D_y^k = \nabla_2 f(x^k, y^k)$, $D_v^k = -\nabla_2 F(x^k, y^k) + \nabla_{22}^2 f(x^k, y^k) v^k$ 是 (3.9) 的目标函数的梯度, D_x^k 是超梯度 (3.5) 的近似. 通过公式 (3.5) 可知,

$$D_x^k = \nabla_1 F(x^k, y^k) - \nabla_{12}^2 f(x^k, y^k) v^k.$$

总的来说, 对于下层问题是强凸优化问题的情况, 由于下层问题的最优解唯一, 我们可以用隐函数定理来获得计算超梯度的公式 (3.5). 所以在这种情况下绝大多数的算法都是围绕梯度法来设计的.

3.3. 下层问题是一般凸优化问题

由于上层问题需要下层问题的最优解, 那么在双层优化中首先要保证下层问题是凸问题, 这样才能保证可以设计算法来获得下层问题的最优解. 所以文献中关于双层优化的研究基本上都考虑下层问题是强凸或者一般凸的情况. 当下层问题是强凸的时候, 其最优解是唯一的. 这时我们可以应用隐函数定理来获得计算超梯度的公式 (3.5). 上一个小节所讨论的算法基本上都是基于超梯度的. 也就是说对于上层和下层问题都采用梯度算法. 但是, 当下层问题是一般凸优化问题时, 其最优解可能不唯一, 那么隐函数定理的条件不成立. 这时就不能计算超梯度了. 所以对于下层问题是一般凸优化问题的情况, 其算法设计跟下层问题是强凸问题的情况是有很大的区别的.

在机器学习的工作中, 一个很自然的想法是：是否可以给下层问题的目标函数加一个小的扰动项 $\epsilon\|y\|_2^2$ 使其变成强凸问题, 然后用上一节的算法去求解这个扰动后的问题, 以期近似的求解原问题. 但是, 这个方法是不可行的. Liu et al.的文章 [34] 和 Chen et al.的文章 [6] 对此给出了反例, 说明了下层函数 f 的轻微扰动, 可能会导致函数 Φ 的巨大变化. 所以, 加扰动项这个技巧在这里是无法应用的.

处理下层问题是一般凸优化问题的一个非常重要的方法是 value-function-based 方法. 这方面早期的工作包括 [12, 39, 52]. 这方面最近也有很多工作, 包括 [16, 31, 32, 35, 38, 42, 44, 51] 等一系列的文章. 其中 [31] 提出了下面的算法. 首先考虑单层约束优化的等价问题 (3.3). 由于约束规范化条件不成立, [31] 的作者提出给约束做一个轻微的扰动, 从而保证约束集合是有内点的. 扰动以后的问题如下：

$$\min_{x,y} \ F(x,y), \ \text{s.t.}, f(x,y) \le f_\mu^*(x), \tag{3.11}$$

这里

$$f_\mu^*(x) := \min_y f(x,y) + \frac{\mu_1}{2}\|y\|^2 + \mu_2. \tag{3.12}$$

然后使用内点算法来求解不等式约束问题 (3.11). 具体做法是首先考虑下面的 log barrier 函数

$$\Phi_{\mu,\theta,\tau}(x) := \min_y F(x,y) + \frac{\theta}{2}\|y\|^2 - \tau\log(f_\mu^*(x) - f(x,y)). \tag{3.13}$$

这里又增加了一个扰动项 $\frac{\theta}{2}\|y\|^2$ 是为了保证函数 $\Phi_{\mu,\theta,\tau}(x)$ 是可导的. 内点算法求解该问题的基本步骤是

Step 1. 计算 $\Phi_{\mu_k,\theta_k,\tau_k}(x)$ 在 x^k 点的梯度 $\nabla\Phi_{\mu_k,\theta_k,\tau_k}(x^k)$

Step 2. 更新 x^k: $x^{k+1} = x^k - \alpha_k\nabla\Phi_{\mu_k,\theta_k,\tau_k}(x^k)$

Step 3. 缩小罚因子 μ_k, θ, τ_k.

这里要注意第一步计算梯度 $\nabla\Phi_{\mu_k,\theta_k,\tau_k}(x^k)$ 并不简单. 其涉及到求解两个优化子问题 (3.12) 和 (3.13). 在这个工作之后, 很多其他的工作也设计了相关的算法来求解下层问题是一般凸问题的双层优化. 我们下面对这些工作做一个简单的介绍. Liu et al. [29] 提出的 BOME! 算法可以简要描述如下[1)] .

Step 1. 运行 T 次梯度步来近似求解下层问题:

$$y_k^{(t+1)} = y_k^{(t)} - \alpha\nabla_y f(x_k, y_k^{(t)}), \quad t = 0,\ldots,T-1.$$

这样得到的 $y_k^{(T)}$ 是 $y^*(x_k)$ 的一个近似.

Step 2. 定义 $\hat{q}(x,y) = f(x,y) - f(x,y_k^{(T)})$

Step 3. 计算

$$\lambda_k = \max\left\{\eta - \frac{\langle\nabla F(x_k,y_k), \nabla\hat{q}(x_k,y_k)\rangle}{\|\nabla\hat{q}(x_k,y_k)\|^2}, 0\right\}$$

Step 4. 更新 (x_k, y_k):

$$(x_{k+1}, y_{k+1}) := (x_k, y_k) - \xi(\nabla F(x_k,y_k) + \lambda_k\nabla\hat{q}(x_k,y_k)).$$

这个算法的原理如下. 首先, 考虑 (3.3) 的等价形式:

$$\min_{x,y}\ F(x,y),\ \text{s.t.}, q(x,y) := f(x,y) - f^*(x) \le 0. \tag{3.14}$$

注意我们的初始点 (x_0,y_0) 不一定满足这里的约束条件, 也就是说 $q(x_0,y_0)$ 可能是正数. 所以这里的思路是通过公式

$$(x_{k+1}, y_{k+1}) := (x_k, y_k) - \xi\delta_k \tag{3.15}$$

来更新 (x_k,y_k), 这里的 δ_k 的定义是:

$$\delta_k = \operatorname*{argmin}_\delta \|\nabla F(x_k,y_k) - \delta\|^2,\ \text{s.t.},\ \langle\nabla q(x_k,y_k), \delta\rangle \ge \phi_k. \tag{3.16}$$

[1)] 这里我们指出 [29] 虽然并未假设下层问题是凸问题, 但是仍然假设了下层函数 $f(x,\cdot)$ 有唯一极小点并且满足 PL 不等式.

这里 $\phi_k = \eta\|\nabla q(v_k, \theta_k)\|^2$ 给 $\langle \nabla q(x_k, y_k), \delta\rangle$ 提供一个正的下界. 当步长 ξ 足够小的时候, 这个正的下界保证 (3.15) 更新公式使得 q 会下降. 因为 q 是正的, 而我们希望 q 变成负的以满足 (3.14) 的约束条件. 所以我们希望 q 是下降的. 同时, (3.16) 保证 δ_k 足够接近 $\nabla F(x_k, y_k)$, 所以 (3.15) 更新公式也会使得 F 下降. 这个 δ_k 的定义公式 (3.16) 有一个显式解, 就是上面的算法描述中 Step 3 和 Step 4 给出的具体表达形式. 在实际操作中, 计算 $q(x_k, y_k)$ 和 $\nabla q(x_k, y_k)$ 都需要 $y^*(x_k)$. 在上面的算法描述中, 当我们用 $y_k^{(T)}$ 来近似 $y^*(x_k)$ 后, 我们在 Step 2 定义了 $\hat{q}$ 来代替 q.

3.4. 一阶算法

需要指出的是, 第 3.2 节所讨论的基于超梯度的算法中, 都会涉及到下层函数 f 的二阶导数. 而第 3.3 节所讨论的 value-function-based 算法, 由于不需要计算超梯度, 都只需要计算下层函数 f 的一阶导数即可. 在此基础上, Kwon et al.等的文章 [25] 提出了 F2SA(Fully First-order Stochastic Approximation) 算法. 这个算法的核心就是：如果对于上层和下层函数都只有一阶导数信息, 如何设计算法? F2SA 算法首先考虑把单层约束优化问题 (3.3) 用拉格朗日函数写成极小极大问题的等价形式：

$$\min_{x,y}\max_{\lambda}\mathcal{L}_\lambda(x,y) := F(x,y) + \lambda(f(x,y) - f^*(x)).$$

定义 $\mathcal{L}_\lambda^*(x) := \min_y \mathcal{L}_\lambda(x,y)$. 文章 [25] 证明了, 当 λ 足够大的时候, 下面的不等式成立

$$\|\nabla\Phi(x) - \nabla\mathcal{L}_\lambda^*(x)\| \le C/\lambda.$$

这里 $C > 0$ 是一个常数. 所以, 当 λ 足够大的时候, 我们可以用 $\nabla\mathcal{L}_\lambda^*(x)$ 来代替超梯度 $\nabla\Phi(x)$. 那么如何计算 $\nabla\mathcal{L}_\lambda^*(x)$? [25] 给出了计算公式：

$$\nabla_x\mathcal{L}_\lambda(x, y_\lambda^*(x)) = \nabla_x F(x, y_\lambda^*(x)) + \lambda(\nabla_x f(x, y_\lambda^*(x)) - \nabla_x f(x, y^*(x))).$$

这里 $y_\lambda^*(x)$ 的定义是 $y_\lambda^*(x) := \mathrm{argmin}_y \mathcal{L}_\lambda(x,y)$. 注意这里当 λ 足够大的时候, $\mathcal{L}_\lambda(x,y)$ 关于 y 是强凸函数. F2SA 算法设计的思路是：用 z^k 来逼近 $y^*(x^k)$, 用 y^k 来逼近 $y_\lambda^*(x)$, 然后再对 x 用超梯度做梯度下降. F2SA 算法可以描述如下：

for $k = 0, 1, \ldots, K-1$ **do**

$z_{k,0} := z_k, \quad y_{k,0} = y_k$

for $t = 0, 1, \ldots, T-1$ **do**

$z_{k,t+1} := z_{k,t} - \gamma_k h_{fz}^{k,t}$

$y_{k,t+1} := y_{k,t} - \alpha_k(h_{Fy}^{k,t} + \lambda_k h_{fy}^{k,t})$

end for

$z_{k+1} := z_{k,T}, \quad y_{k+1} := y_{k,T}$

$x_{k+1} := x_k - \xi\alpha_k(h_{Fx}^k + \lambda_k(h_{fxy}^k - h_{fxz}^k))$

$\lambda_{k+1} := \lambda_k + \delta_k$

end for

这里的 h 分别是相应的梯度的近似 (或者说随机梯度). 定义如下：

$$h_{fz}^{k,t} := \nabla_y f(x_k, z_{k,t}), \quad h_{Fy}^{k,t} := \nabla_y F(x_k, y_{k,t}),$$
$$h_{fy}^{k,t} := \nabla_y f(x_k, y_{k,t}), \quad h_{fxy}^{k} := \nabla_x f(x_k, y_{k+1}),$$
$$h_{Fx}^{k} := \nabla_x F(x_k, y_{k+1}), \quad h_{fxz}^{k} := \nabla_x f(x_k, z_{k+1}).$$

在这个算法中, 我们还可以设置 $T = 1$, 从而得到一个单循环的算法.

4. 去中心化双层优化

去中心化双层优化是一个非常重要的研究方向. 这方面最早的工作是 Chen et al.2022 年 6 月份的 arxiv 预印本文章 [8]. 该文章提出了最早的双层优化的去中心化算法. 这里我们考虑上层和下层函数都是 finite-sum(3.2) 的情况, 其中 m 是局部服务器的个数. 这些局部服务器通过网络联接起来. 为了保护隐私等需要, 局部服务器的数据不能共享给其他服务器. 这就要求局部服务器只能用自己的数据来运行算法和更新迭代点, 然后把迭代点的相关信息通过网络跟其他服务器分享. 当整个问题是一个双层优化问题时, 这个问题变得非常的复杂. Chen et al.的文章 [8] 考虑了下层函数是强凸的情况. 根据我们之前的讨论, 这种情况下一般是计算超梯度, 然后对上层函数做梯度下降法. 由于每个局部服务器只能通过自己的数据来计算超梯度, 那么在第 i 个服务器上面计算的超梯度应该是：

$$\nabla \Phi_i(x) = \nabla_1 F_i(x, y^*(x)) - \nabla_{12} f(x, y^*(x)) \left(\nabla_{22}^2 f(x, y^*(x))\right)^{-1} \nabla_2 F_i(x, y^*(x)). \tag{4.1}$$

这里需要注意, 虽然 $\nabla_1 F_i(x, y^*(x))$ 和 $\nabla_2 F_i(x, y^*(x))$ 可以通过局部数据得到, 但是 $\nabla_{12} f(x, y^*(x)) \left(\nabla_{22}^2 f(x, y^*(x))\right)^{-1}$ 是需要全局信息的. Chen et al.的文章 [8] 提出用下面的方法在每一个局部服务器来计算这个全局信息

$$Z^\top = \left(\sum_{i=1}^m \nabla_{12} f_i(x, y)\right) \left(\sum_{i=1}^m \nabla_{22}^2 f_i(x, y)\right)^{-1}. \tag{4.2}$$

我们引入下列记号：$J_i := \nabla_{12} f_i(x, y)$, $H_i = \nabla_{22}^2 f_i(x, y)$. 那么计算全局信息 Z 就等价于求解下面的二次规划问题：

$$\min_{Z \in \mathbb{R}^{q \times p}} \frac{1}{m} \sum_{i=1}^m \frac{1}{2} \mathrm{Tr}(Z^\top H_i Z) - \mathrm{Tr}(J_i Z). \tag{4.3}$$

这个二次规划问题可以用现有的去中心化的算法来求解. 当然这需要各局部服务器之前的相互协作和通信. 这就是 [8] 的基本框架. 在一篇后续的文章 [9] 中, Chen et al.引入了新的技巧, 从而不需要计算 Hessian 和 Jacobian 矩阵, 而只需要计算它们与向量的乘积. [9] 同时也引入了移动平均 (moving average) 的技巧. 这些都帮助提升了 [8] 文中算法的计算效率和复杂度.

很多其他的工作也讨论了去中心化双层优化问题的算法, 比如 [15, 36, 48]. 这里我们重点介绍一项最近的工作 [13]. 在这篇文章中, 作者 Dong et al.提出了一种单循环的去中心化双层优化算法 SLDBO. 这篇文章的基本想法是把 SOBA 算法 [11] 拓展到去中心化的情形. Dong et al. [13] 提出的 SLDBO 算法一个显著的特点是不需要异质性 (heterogeneity) 假设. 异质性假设是去中心化双层优化算法的一个常用的假设. 这个假设主要是为了保证各局部服务器上

面的数据是相似的, 从而比较容易证明去中心化算法的收敛性. 比如下列工作都做了这样的假设.

- (DSBO, [8] 的假设 2.4). 假设各个局部服务器上面的数据都是独立同分布的.
- (MA-DSBO, [9] 的假设 2.3). 假设存在常数 $\delta \geq 0$ 使得下式成立

$$\left\| \nabla_2 f_i(x,y) - \frac{1}{m}\sum\nolimits_{i=1}^{m} \nabla_2 f_i(x,y) \right\| \leq \delta, \quad \forall x, y.$$

- (SLAM, [37] 的定理 1). 假设存在常数 $L \geq 0$ 使得下式成立

$$\left\| \nabla^2_{22} f_i(x_i,y_i) - \frac{1}{m}\sum\nolimits_{i=1}^{m} \nabla^2_{22} f_i(x_i,y_i') \right\| \leq L\|y_i - y_i'\|, \quad \forall x_i, y_i, y_i'.$$

- (SimFBO, [49] 的假设 4). 假设存在常数 $\delta_1 \geq 1$ 和 $\delta_2 \geq 0$ 使得下式成立

$$\frac{1}{m}\sum_{i=1}^{m} \|\nabla_2 f_i(x,y)\|^2 \leq \delta_1^2 \left\| \frac{1}{m}\sum\nolimits_{i=1}^{m} \nabla_2 f_i(x,y) \right\|^2 + \delta_2^2, \quad \forall x, y.$$

基于此, 在不做任何异质性假设的情况下, [13] 把 SOBA 算法拓展到去中心化的情形是一个重大的突破. SLDBO 算法简要描述在算法 4.1 中.

算法 4.1 A Single-Loop Algorithm for DBO (SLDBO)

输入: K 是算法最大迭代次数; r_y 和 r_v 是两个给定的常数; 网络拓扑矩阵 $W = [w_{ij}]$; 步长 α, β, η

for $k = 0, 1, \ldots, K-1$ **do**

  for $i = 1, \ldots, m$ **do**

$$d_{y,i}^k = \nabla_2 f_i(x_i^k, y_i^k); \tag{4.4}$$

$$d_{v,i}^k = \nabla_2 F_i(x_i^k, y_i^k) - \nabla^2_{22} f_i(x_i^k, y_i^k) v_i^k; \tag{4.5}$$

$$d_{x,i}^k = \nabla_1 F_i(x_i^k, y_i^k) - \nabla^2_{12} f_i(x_i^k, y_i^k) v_i^k; \tag{4.6}$$

$$t_{y,i}^k = \sum\nolimits_{j=1}^{m} w_{ij} t_{y,j}^{k-1} + d_{y,i}^k - d_{y,i}^{k-1}, \quad y_i^{k+1} = \mathcal{P}_{r_y}\left[\sum\nolimits_{j=1}^{m} w_{ij}(y_j^k - \beta t_{y,j}^k)\right]; \tag{4.7}$$

$$t_{v,i}^k = \sum\nolimits_{j=1}^{m} w_{ij} t_{v,j}^{k-1} + d_{v,i}^k - d_{v,i}^{k-1}, \quad v_i^{k+1} = \mathcal{P}_{r_v}\left[\sum\nolimits_{j=1}^{m} w_{ij}(v_j^k + \eta t_{v,j}^k)\right]; \tag{4.8}$$

$$t_{x,i}^k = \sum\nolimits_{j=1}^{m} w_{ij} t_{x,j}^{k-1} + d_{x,i}^k - d_{x,i}^{k-1}, \quad x_i^{k+1} = \sum\nolimits_{j=1}^{m} w_{ij}(x_j^k - \alpha t_{x,j}^k). \tag{4.9}$$

  end for

end for

这里我们对 SLDBO(算法 4.1) 做一些简要的说明. 参数 r_y, r_v, α, β, η 在 [13] 文章中给出了具体的形式 (计算公式或者需要满足的条件). (4.7), (4.8), (4.9) 三个式子对应于 SOBA

算法 (3.10) 的迭代公式. 只是在计算梯度 $(t_{y,i}^k,\, t_{v,i}^k,\, t_{x,i}^k)$ 的时候用到了梯度追踪 (gradient tracking) 的技巧. 在 (4.7) 和 (4.8) 中的记号 $\mathcal{P}_r(z)$ 是指将 z 投影到半径为 r 的球里面. 这个投影是非常重要的步骤. 在去中心化双层优化算法中, 一个核心的问题就是如何保证迭代点是有界的. 这也是为什么文献中这方面的算法都要做异质性假设. [13] 通过巧妙的设计投影和投影半径, 很好的克服了这个问题. 这个投影的操作保证了 y_i^k 和 v_i^k 都是有界的. 这两个有界又可以推导出 x_i^k 是有界的, 所以我们不需要对 x_i^k 做投影. [13] 文中关于 SLDBO 的收敛性理论结果如下.

定理 4.1. 定义 $\bar{x}^k = \frac{1}{m}\sum_{i=1}^m x_i^k$, $\bar{y}^k = \frac{1}{m}\sum_{i=1}^m y_i^k$ and $\bar{v}^k = \frac{1}{m}\sum_{i=1}^m v_i^k$. 算法 4.1 的迭代序列满足下列式子.

(a) 一致性误差 (Consensus Error). 对于任意的整数 $1 \le k \le K$, 下列式子成立

$$\frac{1}{m}\sum_{i=1}^m \|x_i^k - \bar{x}^k\|^2 = O\left(\frac{1}{K}\right),$$

$$\frac{1}{m}\sum_{i=1}^m \|y_i^k - \bar{y}^k\|^2 = O\left(\frac{1}{K}\right), \quad \frac{1}{m}\sum_{i=1}^m \|v_i^k - \bar{v}^k\|^2 = O\left(\frac{1}{K}\right).$$

(b) 稳定点误差 (Stationarity). 对于任意的整数 $K \ge 1$, 下列式子成立

$$\min_{0\le k\le K-1} \|\nabla\Phi(\bar{x}^k)\|^2 = O\big(1/\sqrt{K}\big).$$

这说明在前 K 次迭代中, 至少有一个各服务器的平均点 $\bar{x}^k$, 其对应的超梯度的范数的平方是 $O\big(1/\sqrt{K}\big)$ 量级的. 这给出了 SLDBO 算法的收敛速度.

5. 联邦双层优化

联邦双层优化是另一个值得关注的研究方向. 联邦优化同样是关注网络上的优化问题, 只是网络结构与上一节所讲的去中心化网络结构不同. 在联邦优化中, 有一个中心服务器和很多局部服务器. 中心服务器和局部服务器中心可以进行通信来做信息交互, 但是局部服务器之间不做信息交互. 如何在联邦网络结构中求解双层优化是一个新的课题. 与之前去中心化的讨论类似, 在联邦双层优化中, 我们仍然考虑上层和下层函数都是 finite-sum(3.2) 的情况. 这方面最近的代表性工作是 Kaiyi Ji 课题组的一系列工作 [21, 46, 49]. 在最新的工作 [49] 中, Yang et al.提出了 SimFBO 算法. 该算法不需要子循环, 而且可以达到线性加速 (即当服务器个数增加时, 算法收敛速度也随之线性提高). SimFBO 简要描述如下 (见算法 5.1). 这里我们可以看到局部服务器和中心服务器都采用了类似于 SOBA 的方式来更新迭代点 x,v,y. 局部服务器将加权平均的梯度通过通信传送给中心服务器. 中心服务器将收到的梯度再次做加权平均, 并以此作为梯度在中心服务器上进行梯度方法的迭代. 这个算法里面最后中心服务器更新迭代点的时候对于 v 的更新也做了一个类似于 SLDBO 的投影. [49] 的作者证明了这个算法的线性加速性质, 也证明了更好的采样复杂度.

算法 5.1 SimFBO

输入: T 是算法最大迭代次数; $\tau_i^{(t)}$ 是第 i 个局部服务器第 t 次外层迭代时的最大迭代次数; 步长 γ_y, γ_v, γ_x; 投影半径 r

for $t = 0, 1, \ldots, T$ **do**

  for $i = 1, \ldots, m$ **do**

    $y_i^{(t,0)} = y^{(t)}, v_i^{(t,0)} = v^{(t)}, x_i^{(t,0)} = x^{(t)}$

    for $k = 0, 1, \ldots, \tau_i^{(t)} - 1$ **do**

      局部服务器用类似 SOBA(3.10) 和 SLDBO(Alg 4.1) 的方式更新 $y_i^{(t,k)}$,$v_i^{(t,k)}$,$x_i^{(t,k)}$

    end for

    局部服务器 i 对自己的这 $\tau_i^{(t)}$ 次迭代所有的梯度做一个加权平均, 记作 $q_{y,i}^{(t)}$, $q_{v,i}^{(t)}$, $q_{x,i}^{(t)}$ (分别对应于更新 y,v,x 所用到的梯度).

  end for

  中心服务器通过通信从局部服务器得到 $q_{y,i}^{(t)}$, $q_{v,i}^{(t)}$, $q_{x,i}^{(t)}$, 并再次做加权平均得到 $q_y^{(t)}$, $q_v^{(t)}$, $q_x^{(t)}$

  中心服务器更新迭代点 y,v,x

$$y^{(t+1)} = y^{(t)} - \gamma_y q_y^{(t)}, \quad v^{(t+1)} = \mathcal{P}_{r_v}\left(v^{(t)} - \gamma_v q_v^{(t)}\right), x^{(t+1)} = x^{(t)} - \gamma_x q_x^{(t)}.$$

end for

6. 总结与展望

双层优化是一个热门的研究方向. 关于双层优化的研究是一个非常活跃的领域. 近几年每年都有很多关于双层优化的理论, 算法, 应用等方面的工作. 除了本文所讨论的几类双层优化问题之外, 也有一些文章着重于研究带非光滑项的双层优化问题的算法, 比如 [10,20,38]. 对这些最新的进展给出一个全面的综述是非常有挑战性的. 并且本文主要讨论了机器学习中的双层优化的算法, 对于双层优化的理论, 感兴趣的读者可以参考 Jane J. Ye [50,52,53] 与合作者的一系列工作. 对于双层优化在其他领域的应用, 可参考 [12]. 感兴趣的读者也可以参考其他的一些双层优化的综述文章, 比如 [30,54] 对于双层优化在机器学习, 计算机视觉和信号处理领域的应用做了比较全面的介绍. 由于该领域发展迅猛, 本文因篇幅所限难免未能讨论很多其他优秀的工作. 本文旨在抛砖引玉, 对这个领域做一个简要的介绍. 欢迎各位同行批评指正.

参 考 文 献

[1] Bennett K, Hu J, Kunapuli G, and Pang J S. Classification model selection via bilevel programming. Optimization Methods and Software, 2008, 23: 475–489.

[2] Bennett K, Ji X, Hu J, Kunapuli G, and Pang J S. Model selection via bilevel programming. In Proceedings of the International Joint Conference on Neural Networks (IJCNN'06) Vancouver, B.C. Canada, July, 2006, 16–21, 1922–1929.

[3] Bertinetto L, Henriques J F, Torr P, and Vedaldi A. Meta-learning with differentiable closed-form solvers. In International Conference on Learning Representations (ICLR), 2018.

[4] Bracken J and McGill J T. Mathematical programs with optimization problems in the constraints. Operations Research, 1973, 21(1): 37–44.

[5] Chen H, Xu H, Jiang R, and So A M C. Lower-level duality based reformulation and majorization minimization algorithm for hyperparameter optimization. arXiv:2403.00314, 2024.

[6] Chen L, Xu J, and Zhang J. Bilevel optimization without lower-level strong convexity from the hyper-objective perspective. https://arxiv.org/abs/2301.00712, 2023.

[7] Chen T, Sun Y, Xiao Q, and Yin W. A single-timescale stochastic bilevel optimization method. In AISTATS, 2022.

[8] Chen X, Huang M, and Ma S. Decentralized bilevel optimization. Optimization Letters (special issue on Recent Advances in Bilevel Optimization and Its Applications), 2024.

[9] Chen X, Huang M, Ma S, and Balasubramanian K. Decentralized stochastic bilevel optimization with improved per-iteration complexity. In ICML, 2023.

[10] Chen Z, Kailkhura B, and Zhou Y. An accelerated proximal algorithm for regularized nonconvex and nonsmooth bi-level optimization. Machine Learning, 2023, 112(5): 1433–1463.

[11] Dagréou M, Ablin P, Vaiter S, and Moreau T. A framework for bilevel optimization that enables stochastic and global variance reduction algorithms. In NeurIPS, 2022, 35: 26698–26710.

[12] Dempe S and Zemkoho A. Bilevel Optimization. Springer, 2020.

[13] Dong Y, Ma S, Yang J, and Yin C. A single-loop algorithm for decentralized bilevel optimization. arxiv:2311.08945, 2023.

[14] Franceschi L, Frasconi P, Salzo S, Grazzi R, and Pontil M. Bilevel programming for hyperparameter optimization and meta-learning. In ICML, 2018, 80: 1568–1577.

[15] Gao H, Gu B, and Thai M T. Stochastic bilevel distributed optimization over a network. arXiv preprint arXiv:2206.15025, 2022.

[16] Gao L L, Ye J, Yin H, Zeng S, and Zhang J. Value function based difference-of-convex algorithm for bilevel hyperparameter selection problems. In ICML, 2022.

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

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

[19] Hong M, Wai H T, Wang Z, and Yang Z. A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic. SIAM Journal on Optimization, 2023, 33(1): 147–180.

[20] Huang F, Li J, Gao S, and Huang H. Enhanced bilevel optimization via Bregman distance. In NeurIPS, 2022.

[21] Huang M, Zhang D, and Ji K. Achieving linear speedup in non-IID federated bilevel learning. In ICML, 2023.

[22] Ji K, Yang J, and Liang Y. Bilevel optimization: Convergence analysis and enhanced design. In ICML, 2021, 139, 4882–4892,.

[23] Khanduri P, Zeng S, Hong M, Wai H T, Wang Z, and Yang Z. A near-optimal algorithm for stochastic bilevel optimization via double-momentum. In NeurIPS, 2021.

[24] Kunapuli G, Bennett K, Hu J, and Pang J S. Bilevel model selection for support vector machines. Centre de Recherches Mathématiques CRM Proceedings and Lecture Notes, 2008, 45: 129–158.

[25] Kwon J, Kwon D, Wright S J, and Nowak R D. A fully first-order method for stochastic bilevel optimization. ICML, 2023.

[26] Lee Y C, Mitchell J E, and Pang J S. An algorithm for global solution to bi-parametric linear complementarity constrained linear programs. Journal of Global Optimization, 2013, 62(2): 263–297.

[27] Lee Y C, Mitchell J E, and Pang J S. Global resolution of the support vector machine regression parameters selection problem with lpcc. EURO Journal on Computational Optimization, 2013, 3(3): 197–261.

[28] Li Q, Li Z, and Zemkoho A. Bilevel hyperparameteroptimization for support vector classification: theoretical analysis and a solution method. Mathematical Methods of Operations Research, 2022, 96: 315–350.

[29] Liu B, Ye M, Wright S, Stone P, and Liu Q. Bome! bilevel optimization made easy: A simple first-order approach. In NeurIPS, 2022.

[30] Liu R, Gao J, Zhang J, Meng D, and Lin Z. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021, 44(12): 10045–10067.

[31] Liu R, Liu X, Yuan X, Zeng S, and Zhang J. A value-function-based interior-point method for non-convex bi-level optimization. In ICML, 2021.

[32] Liu R, Liu X, Zeng S, Zhang J, and Zhang Y. Value-function-based sequential minimization for bi-level optimization. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2023, 45(12): 15930–15948.

[33] Liu R, Mu P, Yuan X, Zeng S, and Zhang J. A generic first-order algorithmic framework for bi-level programming beyond lower-level singleton. In ICML, 2020.

[34] Liu R, Mu P, Yuan X, Zeng S, and Zhang J. A general descent aggregation framework for gradient-based bi-level optimization. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2023, 45(1): 38–57.

[35] Lu S. SLM: A smoothed first-order Lagrangian method for structured constrained nonconvex optimization. In NeurIPS, 2024.

[36] Lu S, Cui X, Squillante M S, Kingsbury B, and Horesh L. Decentralized bilevel optimization for personalized client learning. In ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2022, 5543–5547.

[37] Lu S, Zeng S, Cui X, Squillante M, Horesh L, Kingsbury B, Liu J, and Hong M. A stochastic linearized augmented lagrangian method for decentralized bilevel optimization. In NeurIPS, 2022, 35, 30638–30650.

[38] Lu Z and Mei S. First-order penalty methods for bilevel optimization. arXiv:2301.01716, 2023.

[39] Outrata J V. On the numerical solution of a class of Stackelberg problems. Zeitschrift fur Operations Research, 1990.

[40] Pedregosa F. Hyperparameter optimization with approximate gradient. In ICML, 2016, 48, 737–746.

[41] Rajeswaran A, Finn C, Kakade S M, and Levine S. Meta-learning with implicit gradients. In NeurIPS, 2019, 32, 113–124.

[42] Shen H and Chen T. On penalty-based bilevel gradient descent method. In ICML, 2023.

[43] Snell J, Swersky K, and Zemel R. Prototypical networks for few-shot learning. In NeurIPS, 2017.

[44] Sow D, Ji K, Guan Z, and Liang Y. A constrained optimization approach to bilevel optimization-with multiple inner minima. arXiv:2203.01123, 2022.

[45] Stackelberg H v. Theory of the market economy. 1952.

[46] Xiao P and Ji K. Communication-efficient federated hypergradient computation via aggregated iterative differentiation. In ICML, 2023.

[47] Yang J, Ji K, and Liang Y. Provably faster algorithms for bilevel optimization. NeurIPS, 2021, 34: 13670–13682.

[48] Yang S, Zhang X, and Wang M. Decentralized gossip-based stochastic bilevel optimization over communication networks. In NeurIPS, 2022.

[49] Yang Y, Xiao P, and Ji K. SimFBO: Towards simple, flexible and communication-efficient federated bilevel learning. In NeurIPS, 2023.

[50] Ye J J. Constraint qualifications and kkt conditions for bilevel programming problems. Mathematics of Operations Research, 2006, 31(4): 811–824.

[51] Ye J J, Yuan X, Zeng S, and Zhang J. Difference of convex algorithms for bilevel programs with applications in hyperparameter selection. Mathematical Programming, 2023, 198(2): 1583–1616.

[52] Ye J J and Zhu D L. Optimality conditions for bilevel programming problems. Optimization, 1995.

[53] Ye J J, Zhu D L, and Zhu Q J. Exact penalization and necessary optimality conditions for generalized bilevel programming problems. SIAM Journal on Optimization, 1997, 7(2): 481–507.

[54] Zhang Y, Khanduri P, Tsaknakis I, Yao Y, Hong M, and Liu S. An introduction to bi-level optimization: Foundations and applications in signal processing and machine learning. arXiv:2308.00788, 2023.

A GENTLE INTRODUCTION TO ALGORITHMS FOR BILEVEL OPTIMIZATION FROM MACHINE LEARNING

Ma Shiqian

(*Department of Computational Applied Mathematics and Operations Research, Rice University, Houston TX 77005, USA*)

Abstract

Bilevel Optimization recently became a very active research area. This is mainly due to its important applications from machine learning. In this paper, we give a gentle introduction to algorithms, theory, and applications of bilevel optimization. In particular, we will discuss the history of bilevel optimization, its applications in power grid, hyper-parameter optimization, meta learning, as well as algorithms for solving bilevel optimization and their convergence properties. We will mainly discuss algorithms for solving two types of bilevel optimization problems: lower-level problem is strongly convex and lower-level problem is convex. We will discuss gradient methods and value-function-based methods. Decentralized and federated bilevel optimization will also be discussed.

Keywords: Bilevel Optimization; First-order Methods; Hyperparameter Optimization; Meta Learning; Decentralized Optimization; Federated Learning.

2010 Mathematics Subject Classification: 90C30.