TONGJI UNIVERSITY

School of Electronics and Information Engineering

装

订

线

Reproduction of Provably Faster Algorithms for Bilevel Optimization

Stochastic Process

2022 年 6 月 24 日

目录

装 订 线

装 订 线

1 论文概述

本论文主要论述了作者针对双层优化问题提出的两种改进算法，并对其应用结果进行介绍。

双层优化算法目前已经广泛应用于机器学习领域，例如元学习、超参数优化、强化学习。而双层优化思想，几乎是很多元学习问题的核心。

作者总结了若干种已有的基于动量的算法，分析了它们在双层优化问题方面的优劣。例如在时间复杂性方面，这些算法并没有达到比随机梯度下降算法（SGD）更好的时间复杂度。

随后作者介绍了选择 inner-loop 与 outer-loop 算法设计的基础方法（基于梯度，Jacobian 和 Hessian 矩阵等），接着介绍了本文讨论问题的范畴，重点强调了非凸-强凸等相关假设。

继而作者提出了两种对于随机双层优化的改进方法。包括基于动量的迭代递归方法（MRBO）和基于嵌套循环中的递归梯度估计方差的方法（VRBO），并对两种算法的算法思想进行全面阐述，对训练结果的收敛性和时间复杂度进行了分析（其时间复杂度优于现有的所有双层优化算法）。

2 研究领域与研究现状

在论文的 1 Introduction 部分，作者总结了近年来被提出的若干种优化算法的思想、原理、复杂度、应用场景。其中不乏大量的专业术语（我们之前并未接触过），而这对我们巩固数学基础、了解相关领域的研究现状、进而理解论文内容有着必要作用。

因此，我们在论文阅读初期进行了大量的资料搜集与文献学习工作，以填补我们在双层优化、双回路与单回路算法设计、元学习、凸优化等相关领域的空白。最终，在报告的本章节中，我们以“关键术语”作为子标题，依据作者的行文思路展开，在原文的基础上补充了相应的资料与我们的个人理解。

2.1 总述

双层优化（Bilevel Optimization）是本文研究的核心问题，作为近年来机器学习一个常用的优化算法，它广泛应用于超参数优化算法、多任务和元学习、网络架构搜索、生成对抗学习、深度强化学习等研究方向。

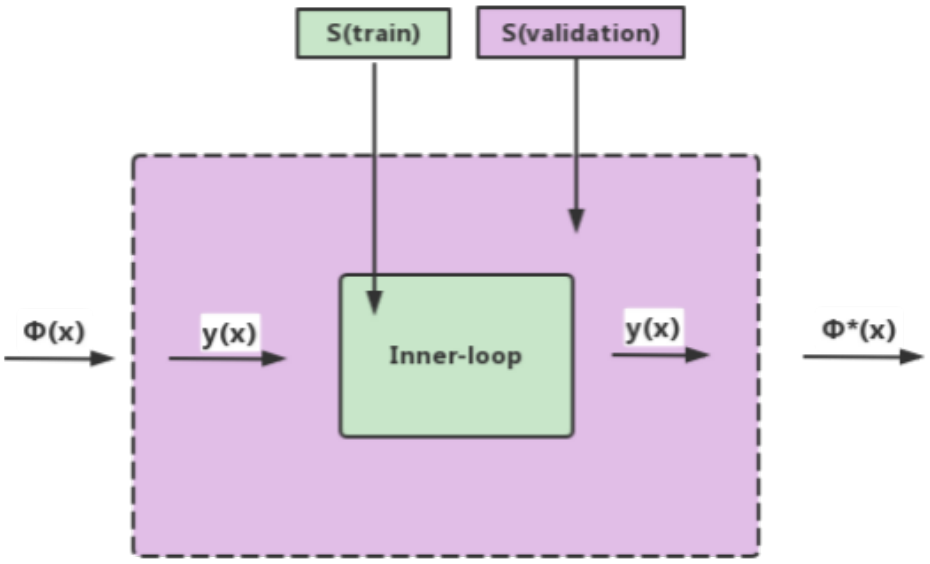
我们在本次复现过程中，首先对双层优化相关文献进行了阅读和探究。通过理解双层优化的运行逻辑来尝试分析它的优点以及改进方向。在原理层面，我们探究了随机性方法（Stochastic Method）与确定性方法（Deterministic Method）的联系与区别，双层优化器的单回路设计（Single-loop Design）与双回路设计（Double-loop Design）的复杂性对比与性能对比。而基于动量的递归法（Momentum-based Recursive Approaches）与方差降低法（Variance reduction Approaches）既是双层优化领域的两种重要策略，也是作者后续进行优化和创新的方向。

随后在其应用端去了解可应用方向，主要了解包括元学习（Meta-Learning），超参数优化（hyperparameter optimization）等。在应用问题的约束上，初步了解了凸优化问题，以适配双层优化可运行环境。最终现有算法的运行逻辑，以理解作者对双层优化改进的具体方式，并通过代码进行具体分析与结果模拟。

以下主要介绍在复现学习过程中对部分方法的知识学习与理解思考。

2.2 双层优化：Bilevel Optimization

双层优化，即包含两层优化任务，其中一层作为约束嵌套在另一层中。内部（Inner-loop）和外部（Outer-loop）优化任务通常分别称为低级子问题和高级子问题，利用训练集训练、优化内层模型，然后令外层学习内层。内层是实际的学习者，而外层可以很好地在内层的基础上加以进步。其核心思想可由如下流程图所表述：



(a) 双层优化

原文开头便提到，双层优化一般可以表述为如下最小化问题：

$$\min_{x\in\mathbb{R}^p}\Phi(x) := f(x,y^*(x)) \quad \text{s.t. } y^*(x) = \arg\min_{y\in\mathbb{R}^q} g(x,y).$$

外部函数的数值取决于变量和内循环函数的优化器，因此，相比于最小优化或极大极小优化，双层优化算法设计更加复杂。例如，如果采用基于梯度的方法，那么 Outer-loop 函数的梯度（即超梯度）必然涉及 Inner-loop 函数的 Jacobian 矩阵与 Hessian 矩阵，这对计算的时间复杂度提出了更高挑战。

2.3 随机双层优化算法：StocBiO

2.3.1 原文内容总结

随机双层优化器已被提出，以便在数据量很大或需要在算法运行时对大量新数据进行采样的大规模场景中实现比确定性方法更好的效率。这是随机性方法（Stochastic Method）与确定性方法（Deterministic Method）的重要区别。

这类问题采用的函数表述通常是：

$$\Phi(x) := f(x,y^*(x)) := \mathbb{E}_\xi\left[F(x,y^*(x);\xi)\right], \quad g(x,y) := \mathbb{E}_\xi[G(x,y;\xi)]$$

其中外部函数和内部函数分别取样本 ξ 和的 ς 期望值。

在此基础上，有相关文献（Bilevel optimization: Nonasymptotic analysis and faster algorithms）提出了一种随机梯度下降（SGD）类型的优化器（stocBiO），并表明 stocBiO 能达到 $\widetilde{\mathcal{O}}\left(\varepsilon^{-2}\right)$ 的计算复杂度并且可以达到准确的静止点。

论文在原有随机双层优化的基础上展开后续算法创新。

2.3.2 资料搜集与拓展研究

我们阅读了被引文献，对 stocBiO 算法流程有了更全面的了解。

针对确定性双层优化问题，stocBiO 分别基于近似隐式微分（AID）和迭代微分（ITD）两种常用算法进行了优化。算法使用高效的 Jacobian-Matrix 和 Hessian-Vector 计算提出了一个样本高效的超梯度估计。

该算法发展了整形器理论和可证明的快速算法，分别用于非凸、强凸双层确定性、随机优化问题。首先为基于 ITD 和 AID 的双层优化器提供了统一的收敛速度和复杂度分析。AID-BiO 认为需要不断增加内环步数来实现的保证，stocBiO 允许实践中经常使用的内环步数不变。此外，有研究中为内环更新和外环超梯度估计引入了一个热启动初始化，这允许反向传播跟踪误差到之前的循环，并产生了改进的计算复杂度。

2.4 单回路设计（single-loop design）与双回路设计（double-loop design）

2.4.1 单回路法与 MRBO

在以往的研究中，诸多基于随机梯度下降（SGD）方法的优化器都遵循单回路设计，即同时更新 x 和 y。

MSTSA 算法，基于动量的方法进行递归更新，并创新了其中的计算策略；类似地，SEMA 算法通过动量递归技术来一起更新 x 和 y；算法 STABLE，应用动量策略来更新 Hessian-Matrix，但涉及复杂的矩阵逆计算而非超梯度近似循环，因此增加了计算的复杂性。

作为 SGD 型 stocBiO 算法，上述 SEMA、MSTSA 和 STABLE 实现了相同的 $\widetilde{\mathcal{O}}\left(\varepsilon^{-2}\right)$ 复杂度，即动量技术在这些算法没有表现出理论上的优势。综上，作者思考了如下问题：

“能否设计一个更快的单回路的基于动量的递归双层优化器，相比 *SGD* 类型的 *stocBiO*（和所有其他基于动量的算法），能实现了按顺序降低计算复杂度，并且也能很容易地用高效的矩阵向量积来实现？”

这便引申出了作者提出的第一种优化算法——基于单回路动量的递归二层优化器（MRBO）。相比于以上算法，MRBO 在计算时同时更新 x 和 y，为梯度和超梯度构建低方差低的小批量梯度下降样本（随机选取数据中一定量的数据进行训练，然后计算损失函数更新梯度），分析时采用 Hessian-vector type 来分析动量的递归估计器的估计属性。

2.4.2 双回路法与 VRBO

尽管现有的加速双层优化算法的研究方向大多专注于单回路设计，但研究表明，双回路双层算法（如 BSA 和 stocBiO）比单回路算法（如 TTSA）实现了更好的性能。

有诸多文献指出，方差减少法能够与双回路算法相配合，提高优化性能，例如 SVRG 、SARAH 和 SPIDER，它们通常可以产生可证明的更低复杂度。基本思想是使用周期性高精度大批量的梯度评估来构造低方差梯度估计器。

到目前为止，还没有任何关于使用方差减少来加速双环双层优化算法的研究。

综上，作者思考了如下问题：

“能否开发一个双循环方差减少的双层优化器，其计算复杂度比 *SGD* 类型的 *stocBiO*（和所有其他现有算法）有所提高？如果可以，这种双循环算法在双层优化中何时会优于单回路算法？”

这便引申出了作者提出的第二种优化算法——基于方差减少策略的递归二层优化器（VRBO）。VRBO 尝试在每个外循环中加入大批量梯度，抛弃原有极大极小优化方法，采取递归方差减少，也达到了近似最优的复杂度。在分析时也观察方差减少情况进行特殊处理。

2.5 元学习：Meta Learning

元学习（Meta Learning），也称为 Learning to Learn (也就是学习学习) 的目标是能够学习新技能或适应新事物的设计模型，通过一些训练示例以获取原理图。在应用元学习时，其可以对几乎任意的可调整参数进行学习，寻找最优解。

元表示，是指我们需要通过元学习学习的东西，可以是内层训练出模型，也可以超参数网络结构、损失、误差等等。而元优化，主要指双层优化中的 Outer-loop 采用的优化方式，通常采用的方法有三种——Gradient，RL 及 Evolution。元目标（Meta-Objective），即元学习的具体目标，不同的应用会有不同的目标。

综上，元学习的主要框架建立在双层优化问题上，两者有着相似的理论基础与研究目标。

在应用过程中，元学习是 Task Level 的，而一般的任务会包含训练集（Support Set）和测试集（Query Set）。正常训练的流程就是先使用训练集进行训练，然后再使用测试集测试。因此，在 Meta-Training 的过程中，我们可以构造双层优化过程，在 Inner-loop 即内层中使用训练集更新模型，然后再外层基于更新后的模型优化 Meta Knowledge

2.6 超参数优化：Hyperparameter Optimization

机器学习算法中的参数是用来调整数据的。超参数优化就是指这个问题找出超参数的最优集合光靠训练数据是无法得出的。我们无法通过直接的一层优化方式得到训练的模型，那么我们就需要进行特殊调整，调整的数据则是整个训练过程的变量。

比如在学习和视觉领域，设计正则化模型或者通常推荐使用支持向量机超参数的选择方法。深度神经网络由许多神经元组成，输入的数据通过这些神经元来传输到输出端。在神经网络训练时候，每个神经元的权重会随着损失函数的值来优化从而减小损失函数的值。这些训练好的参数就是这个算法的模型。超参数是用来调节整个网络训练过程的，例如神经网络的隐藏层的数量，核函数的大小，数量等等。超参数并不直接参与到训练的过程中，他们只是配置变量。需要注意的是在训练过程中，参数是会更新的而超参数往往都是恒定的。

想要对超参数进行调整的这些方法首先表示为双层优化问题，然后进行转换转化为单级优化问题（即 Inner-loop）及其最优性条件。由于计算成本高，尤其是在高维的情况下超参数空间，这些原始方法甚至可以不能保证得到局部最优解。因此尝试利用双层优化来对超参数优化问题进行嵌套优化，令 Outer-loop 目标的目标是最小化验证集关于超参数的损失（如权重衰减），和 LL 目标需要通过最小化与模型参数 (如权重和偏差) 相关的训练损失来输出一个学习算法。这样可以得到想要的超参数调整方法。

想要对超参数进行调整的这些方法首先表示为双层优化问题，然后进行转换转化为单级优化问题内层子问题及其最优性条件。由于计算成本高，尤其是在高维的情况下超参数空间，这些原始方法甚至可以不能保证得到局部最优解。因此尝试利用双层优化来对超参数优化问题进行嵌套优化，令外层目标的目标是最小化验证集关于超参数的损失 (如权重衰减)，和内层目标需要通过最小化与模型参数 (如权重和偏差) 相关的训练损失来输出一个学习算法。这样可以得到想要的超参数调整方法。

2.7 凸优化：Convex Optimization

2.7.1 原文内容总结

在原文的双层算法优化中，解决的主要是凸优化（Convex）和强凸优化问题（Strongly-convex），这表示相关函数具备良好的可优化性质。论文的 Introduction 中提到，已证明更快的双层优化算法主要研究非凸和强凸两种设置情况，其中外部函数相对于 x 是非凸的，并且内部函数 g(x,y) 对任何关于 x 的 y 是强凸的。

论文中列举了两种实际问题中的常见情况：

（1）在超参数优化中，通常是非凸的，其中 x 表示神经网络的超参数，但是内部函数通过在 y 上包含强凸正则化器，它就可以相对于是强凸的。

（2）在少量的元学习中，内部函数通常采用强凸的二次型正则化器。

为了能够有效解决“双层优化等式”中的确定性问题，有学者提出了各种不同的双层优化算法，其中包括两类流行的确定性的基于梯度的方法，分别是基于近似隐式微分（AID）和迭代微分（ITD）。

2.7.2 资料搜集与拓展研究

在此部分，我们对凸优化与双层优化领域的研究问题进行了拓展探究。

凸优化，是数学最优化的一个子领域，研究定义于凸集中的凸函数最小化的问题。虽然条件苛刻，但应用广泛，具有重要价值，主要体现在凸优化本身具有很好的性质。一来，凸问题的局部最优解就是全局最优解，基于这个性质，只要设计一个较为简单的局部算法，例如贪婪算法（Greedy Algorithm）或梯度下降法（Gradient Decent），收敛求得的局部最优解即为全局最优。因此求解凸优化问题相对来说是比较高效的。这也是为什么机器学习中凸优化的模型非常多，毕竟机器学习处理大数据，需要高效的算法。。二来，凸优化理论中的 Lagrange 对偶，为凸优化算法的最优性与有效性提供了保证。近些年来关于凸问题的研究非常透彻，以至于只要把某一问题抽象为凸问题，就可以近似认为这个问题已经解决了。

同时，凸优化具有很强扩展性对于非凸问题，通过一定的手段，要么可以等价地化归为凸问题，要么可以用凸问题去近似、逼近得到边界。例如，几何规划、整数规划，虽然本身是非凸的，但是可以借助凸优化手段去解，这就极大地扩张了凸优化的应用范围。

以深度学习来说，其中关键的反向传播（Back Propagation）算法，本质就是凸优化算法中的梯度下降法，即使问题极度非凸，梯度下降还是有很好的表现，凸优化的应用十分广泛，如线性回归、范数逼近、插值拟合、参数估计，以及许多的几何问题等都可以尝试求解。

装 订 线

3 MRBO 算法设计

作者提出了一种新的基于单回路动量的递归双层优化器（MRBO），并且证明了它比现有的随机双层优化器具有更低的时间复杂复杂度。

3.1 算法描述

基于动量的递归双层优化器（MRBO）以单回路方式更新变量，并采用动量递归技术 STORM 在每次迭代时同步更新 x 和 y。

首先，为更新 y，在第 k 轮迭代中，MRBO 基于当前 $\nabla_y G(x_k, y_k; \mathscr{B}_y)$ 和先前的 $\nabla_y G(x_{k-1}, y_{k-1}; \mathscr{B}_y)$，使用小批量 B_y 样本构造基于动量的梯度估计器 u_k。其中，超参数 β_k 在每次迭代中都会减少，因此梯度估计器 u_k 更多地由先前的 u_{k-1} 决定，特别是当 y_k 临近最优点时，这提高了梯度估计的稳定性。随着迭代轮数增加，步长 η_k 衰减，从而减少收敛误差。

之后，用类似的方法更新 x，在第 k 轮迭代中，MRBO 首先基于当前的 $\widehat{\nabla}\Phi(x_k; \mathscr{B}_x)$ 和之前的 $\widehat{\nabla}\Phi(x_{k-1}; \mathscr{B}_x)$ 使用小批量 B_x 样本构造基于动量的递归超梯度估计器 v_k。超参数 α_k 在每次迭代时减小，因此新的梯度估计 v_k 更多地由临近的 v_{k-1} 确定，尤其是在 x_k 最佳点附近，从而提升了梯度估计的稳定性。

如原文所述，超梯度评估器 $\widehat{\nabla}\Phi(x_k; \mathscr{B}_x)$ 的设计基于以下形式:

$$\widehat{\nabla}\Phi(x_k; \mathscr{B}_x) = \nabla_x F(x_k, y_k; \mathscr{B}_F)$$

$$- \quad \nabla_x \nabla_y G(x_k, y_k; \mathscr{B}_G)\, \eta \sum_{q=-1}^{Q-1} \prod_{j=Q-q}^{Q} \left(I - \eta \nabla_y^2 G(x_k, y_k; \mathscr{B}_j)\right) \nabla_y F(x_k, y_k; \mathscr{B}_F)$$

另外，为提高计算效率，作者在 MRBO 算法中使用 Hessian 向量代替 Hessians。

Algorithm 1: 基于动量的递归双层优化器 (MRBO)

Input: 步幅 $\lambda, \gamma > 0$, 系数 α_0, β_0, 初始值 x_0, y_0,Hessian 评估数 Q,Batch Size S, 常数 $c_1, c_2, m, d > 0$

for $k = 0, 1, \ldots, K$ **do**
  以 Batch Size S 从 $\mathcal{B}_y, \mathcal{B}_x = \{\mathcal{B}_j (j = 1, \ldots, Q), \mathcal{B}_F, \mathcal{B}_G\}$ 中对每个成分抽样
  if $k = 0$ **then**
    $v_k = \widehat{\nabla}\Phi(x_k; \mathcal{B}_x), u_k = \nabla_y G(x_k, y_k; \mathcal{B}_y)$;
  else
    $v_k = \widehat{\nabla}\Phi(x_k; \mathcal{B}_x) + (1 - \alpha_k)(v_{k-1} - \widehat{\nabla}\Phi(x_{k-1}; \mathcal{B}_x))$
    $u_k = \nabla_y G(x_k, y_k; \mathcal{B}_y) + (1 - \beta_k)(u_{k-1} - \nabla_y G(x_{k-1}, y_{k-1}; \mathcal{B}_y))$
  end
  update : $\eta_k = \frac{d}{\sqrt[3]{m+k}}, \alpha_{k+1} = c_1 \eta_k^2, \beta_{k+1} = c_2 \eta_k^2$
  $x_{k+1} = x_k - \gamma \eta_k v_k, y_{k+1} = y_k - \lambda \eta_k u_k$
end

(a) MRBO

3.2 收敛性分析

为了分析 MRBO 收敛性，我们首先需要了解双层优化算法与传统优化方法的差异。首先，需要在双层优化的 Outer-loop 使用超梯度，而这种超梯度动量分析方法涉及一阶梯度与 Hessian 向量，因而更加复杂。其次，由于 MRBO 将基于动量的递归方法同时应用于 Inner-loop 与 Outer-loop 的迭代，故需要考虑内外层循环梯度评估器的交互，并基于此分析算法的收敛性。

为了后续收敛性分析，作者在论文中提出了三点假设，包括：假设内部函数的强凸性、假设双层优化中的自由度与有界方差（以及假设函数满足 lipschitz 连续条件）、假设外目标函数的梯度有界。这是后续 MRBO 算法与 VRBO 算法收敛性分析的基础。

3.2.1 命题 1

若假设 1,2,3 成立且 $\eta < \frac{1}{L}$, 则关于 $\mathscr{B}_x$ 上 x 的超梯度评估器 $\widehat{\nabla}\Phi(x_k;\mathscr{B}_x)$ 具有有界方差

$$\mathbb{E}\left\|\widehat{\nabla}\Phi(x_k;\mathscr{B}_x)-\bar{\nabla}\Phi(x_k)\right\|^2 \leq G^2$$

其中 $G^2=\frac{2M^2}{S}+\frac{12M^2L^2\eta^2(Q+1)^2}{S}+\frac{4M^2L^2(Q+2)(Q+1)^2\eta^4\sigma^2}{S}$。令 $\bar{\varepsilon}_k = v_k - \bar{\nabla}\Phi(x_k)$, 其中 v_k 表示超梯度的动量评估器。则 v_k 的每次迭代方差满足

$$\mathbb{E}\|\bar{\varepsilon}_k\|^2 \leq \mathbb{E}\left[2\alpha_k^2G^2+2(1-\alpha_k)^2L_Q^2\|x_k-x_{k-1}\|^2\right.$$

$$\left.+2(1-\alpha_k)^2L_Q^2\|y_k-y_{k-1}\|^2+(1-\alpha_k)^2\|\bar{\varepsilon}_{k-1}\|^2\right]$$

其中 $L_Q^2=2L^2+4\tau^2\eta^2M^2(Q+1)^2+8L^4\eta^2(Q+1)^2+2L^2\eta^4M^2\rho^2Q^2(Q+1)^2$。

上述命题刻画了基于梯度动量递归评估器对迭代方差的影响，以及每次步骤 k 更新所产生的随机性影响。

3.2.2 命题 2

若假设 1,2,3 成立且 $\eta < \frac{1}{L}$ 和 $\gamma \leq \frac{1}{4L_\Phi\eta_k}$ 成立, 其中 $L_\Phi = L+\frac{2L^2+\tau M^2}{\mu}+\frac{\rho LM+L^3+\tau ML}{\mu^2}+\frac{\rho L^2M}{\mu^3}$ 。

则有

$$\mathbb{E}[\Phi(x_{k+1})] \leq \mathbb{E}[\Phi(x_k)]+2\eta_k\gamma\left(L'^2\|y_k-y^*(x_k)\|^2+\|\bar{\varepsilon}_k\|^2+C_Q^2\right)-\frac{1}{2\gamma\eta_k}\|x_{k+1}-x_k\|^2$$

其中 $C_Q=\frac{(1-\eta\mu)^{Q+1}ML}{\mu}, L'^2=\max\left\{\left(L+\frac{L^2}{\mu}+\frac{M\tau}{\mu}+\frac{LM\rho}{\mu^2}\right)^2, L_Q^2\right\}$.

上述命题描述了目标函数值是如何因变量 x（约束中的最后一项）的一次迭代更新 $\|x_{k+1}-x_k\|^2$ 而减少的（由 $\mathbb{E}[\Phi(x_{k+1})]-\mathbb{E}[\Phi(x_k)]$ 刻画）。这样的数值减少也受到了如下影响: 变量 y 的的跟踪误差 $\|y_k-y^*(x_k)\|^2$ （即 $y_k y^*(x_k)$）, 超梯度的动量评估器的方差 $\|\bar{\varepsilon}_k\|^2$, 以及关于超梯度的 Hessian 反近似误差 C_Q .

3.2.3 **定理** 1

定理 1 描述 MRBO 的收敛特征。

若假设 1,2,3 成立, 令超参数 $c_1 \geq \frac{2}{3d^3}+\frac{9\lambda\mu}{4}, c_2 \geq \frac{2}{3d^3}+\frac{75L'^2\lambda}{2\mu}, m \geq \max\left\{2, d^3, (c_1 d)^3, (c_2 d)^3\right\}, y_1 = y^*(x_1), \eta < \frac{1}{L}, 0 \leq \lambda \leq \frac{1}{6L}, 0 \leq \gamma \leq \min\left\{\frac{1}{4L_\Phi \eta_K}, \frac{\lambda^3}{\sqrt{150L'^2L^2/\mu^2+8\lambda\mu\left(L_Q^2+L^2\right)}}\right\}$,

$$\frac{1}{K}\sum_{k=1}^{K}\left(\frac{L'^2}{4}\|y^*(x_k)-y_k\|^2+\frac{1}{4}\|\bar{\varepsilon}_k\|^2+\frac{1}{4\gamma^2\eta_k^2}\|x_{k+1}-x_k\|^2\right) \leq \frac{M'}{K}(m+K)^{1/3}$$

其中 L'^2 由命 2 所定义， $M'=\frac{\Phi(x_1)-\Phi^*}{\gamma d}+\left(\frac{2G^2\left(c_1^1+c_2^2\right)d^2}{\lambda\mu}+\frac{2C_Q^2 d^2}{\eta_k^2}\right)\log(m+K)+\frac{2G^2}{S\lambda\mu d\eta_0}$

定理 1 刻画了变量 $x_k, y_k \|\bar{\varepsilon}_k\|$ 的同时收敛：跟踪误差 $\|y^*(x_k)-y_k\|$ 收敛至零, 且超梯度的动量递归评估器的方差 $\|\bar{\varepsilon}_k\|$ 也收敛为零, 这两点共同促成了 MRBO 算法的收敛性。

3.3 **复杂性分析**

通过调整超参数以满足定理 1 中的条件，得到 MRBO 计算复杂度满足如下:

在定理 1 的相同条件下, 选择 $K=\mathcal{O}\left(\varepsilon^{-1.5}\right), Q=\mathcal{O}\left(\log\left(\frac{1}{\varepsilon}\right)\right)$ 则算法 1 中的 MRBO 将找到一个 ε -稳定点, 其梯度复杂度为 $\mathcal{O}\left(\varepsilon^{-1.5}\right)$,Jacobian 向量与 Hessian 向量复杂度为 $\widetilde{\mathcal{O}}\left(\varepsilon^{-1.5}\right)$.

综上可知，MRBO 实现了 $\tilde{\mathcal{O}}\left(\varepsilon^{-1.5}\right)$ 的复杂度，这比目前的随机双层优化算法提升了 $\tilde{\mathcal{O}}\left(\varepsilon^{-0.5}\right)$, 同时这也第一次实现了单回路递归动量法比 SGD 方法在双层优化中表现更好的性能

3.4 **总结**

1. MRBO 是将动量递归算法引入双层优化的算法，通过动量完成对内层外层循坏迭代的影响，加入 Hessian 向量的计算分析，是全新的分析方法。

2. 在算法中，作者采用超梯度估计的方法调整方差约束，并基于此约束构建超梯度的动量递归评估器。

3. MRBO 相较于传统的双层优化算法，具有更好的时间复杂度，且超过了现有 SGD 方法在双层优化中的表现。

4 VRBO 算法设计

尽管所有现有的用于二层优化的动量算法（例如动量加速的 SGD 双阶优化器、MSTSA、SEMA 等）都遵循单回路设计，但 stoBiO 算法的提出，表明了双回路二层优化算法可以实现比单回路算法更好的性能。因此，本文作者提出了一种双回路优化算法，称为方差减少双阶优化器 (VRBO)。

4.1 算法描述

VRBO 采用 SARAH 算法与 SPIDER 算法中的方差减少技术进行双回路优化。VRBO 构建了用于更新 x 和 y 的递归方差减少的梯度估计器，在外循环中每次更新 x 之后都会有 (m+1) 次内循环更新 y。

VRBO 将外循环迭代划分为若干个 epoch，并在每个 epoch 开始时，基于一批独立样本 S_1 计算超梯度估计器 $\widehat{\nabla}\Phi(x_k,y_k;\mathscr{S}_1)$ 和梯度 $\nabla_y G(x_k,y_k;\mathscr{S}_1)$，以达到减少方差的目标。其中 $\widehat{\nabla}\Phi(x_k,y_k\mathscr{S}_1)$ 采取以下形式：

$$\widehat{\nabla}\Phi(x_k,y_k;\mathscr{S}_1)=\frac{1}{S_1}\sum_{i=1}^{S_1}(\nabla_x F(x_k,y_k;\xi_i)$$

$$-\nabla_x\nabla_y G(x_k,y_k;\zeta_i)\,\eta\sum_{q=-1}^{Q-1}\prod_{j=Q-q}^{Q}\left(I-\eta\nabla_y^2 G\left(x_k,y_k;\zeta_i^j\right)\right)\nabla_y F(x_k,y_k;\xi_i)$$

VRBO 在内层迭代过程中采用一组更小的样本 S_2，递归地更新梯度估计器 $\nabla_y G(\widetilde{x}_{k,t},\widetilde{y}_{k,t};\mathscr{S}_2)$ 和 $\nabla\Phi(\widetilde{x}_{k,t},\widetilde{y}_{k,t};\mathscr{S}_2)$

Algorithm 2: 方差缩减双层优化器 (VRBO)

Input: 步幅 $\beta,\alpha>0$, 初始值 x_0,y_0,Hessian 评估数 Q, 样本大小 S_1,S_2, 周期数 q

for $k=0,1,\ldots,K$ **do**
  if $k\%q==0$ **then**
    抽取独立同分布样本 S_1
    $u_k=\nabla_y G(x_k,y_k;\mathcal{S}_1),v_k=\widehat{\nabla}\Phi(x_k,y_k;\mathcal{S}_1)$
  else
    $u_k=\tilde{u}_{k-1,m+1},v_k=\tilde{v}_{k-1,m+1}$
  end
  $x_{k+1}=x_k-\alpha v_k$
  令 $\tilde{x}_{k,-1}=x_k,\tilde{y}_{k,-1}=y_k,\tilde{x}_{k,0}=x_{k+1},\tilde{y}_{k,0}=y_k,\tilde{v}_{k,-1}=v_k,\tilde{u}_{k,-1}=u_k$
  for $t=0,1,\ldots,m+1$ **do**
    抽取独立同分布样本 S_2
    $\tilde{v}_{k,t}=\tilde{v}_{k,t-1}+\widehat{\nabla}\Phi(\tilde{x}_{k,t},\tilde{y}_{k,t};\mathcal{S}_2)-\widehat{\nabla}\Phi(\tilde{x}_{k,t-1},\tilde{y}_{k,t-1};\mathcal{S}_2)$
    $\tilde{u}_{k,t}=\tilde{u}_{k,t-1}+\nabla_y G(\tilde{x}_{k,t},\tilde{y}_{k,t};\mathcal{S}_2)-\nabla_y G(\tilde{x}_{k,t-1},\tilde{y}_{k,t-1};\mathcal{S}_2)$
    $\tilde{x}_{k,t+1}=\tilde{x}_{k,t},\tilde{y}_{k,t+1}=\tilde{y}_{k,t}-\beta\tilde{u}_{k,t}$
  end
  $y_{k+1}=\tilde{y}_{k,m+1}$
end

(a) VRBO 算法

4.2 收敛性分析

为了分析 VRBO 的收敛性，作者首先描述了超梯度估计器的统计特性，其中梯度、Jacobian 向量和 Hessian 向量都有递归的方差减少形式。然后，作者阐述了内环跟踪误差如何影响外环超梯度估计误差，以确定整体收敛性。分析的复杂性主要是由于二层优化中的超梯度，这在以前的传统最小化和最小化优化的方差减小研究中是不存在的。

在下面的命题中，作者对超梯度估计器的方差进行了表征，并进一步利用这种约束来表征基于递归方差减少技术的超梯度和内循环梯度估计器在所有迭代中的累积方差。

4.2.1 命题 3

若假设 1,2,3 成立。令 $\eta < \frac{1}{L}$。那么超梯度估计器相对于 x，其方差是有界的。

$$\mathbb{E}\left\|\widehat{\nabla}\Phi(x_k,y_k;\mathscr{S}_1)-\bar{\nabla}\Phi(x_k)\right\|^2 \leq \frac{\sigma'^2}{S_1}$$

其中 $\sigma'^2 = 2M^2 + 28L^2M^2\eta^2(Q+1)^2$. Let $\Delta_k = \mathbb{E}\left(\left\|v_k - \bar{\nabla}\Phi(x_k)\right\|^2 + \left\|u_k - \nabla_y g(x_k,y_k)\right\|^2\right)$

v_k 和 u_k 分别表示超梯度和内环梯度的递归方差减少估计。那么，v_k 和 u_k 的累积方差被约束为

$$\sum_{k=0}^{K-1}\Delta_k \leq \frac{4\sigma'^2K}{S_1} + 22\alpha^2L_Q^2\sum_{k=0}^{K-2}\mathbb{E}\left\|v_k\right\|^2 + \frac{4}{3}\mathbb{E}\left\|\nabla_y g(x_0,y_0)\right\|^2$$

如上述公式所示，超梯度估计器的方差约束随着用于近似 Hessian 逆的 Hessian 向量的数量 Q 的增加而增加，也可以通过改变 S1 规模而减少。公式一并给出了关于超梯度和内环梯度估计器累积方差 $\sum_{k=0}^{K-1}\Delta_k$ 的上限

4.2.2 命题 4

若假设 1、2、3 成立。令 $\eta < \frac{1}{L}$。我们可以得到

$$\mathbb{E}\left[\Phi(x_{k+1})\right] \leq \mathbb{E}\left[\Phi(x_k)\right] + \frac{\alpha L'^2}{\mu^2}\mathbb{E}\left\|\nabla_y g(x_k,y_k)\right\|^2 + \alpha\mathbb{E}\left\|\widetilde{\nabla}\Phi(x_k) - v_k\right\|^2 - \left(\frac{\alpha}{2} - \frac{\alpha^2}{2}L_\Phi\right)\mathbb{E}\left\|v_k\right\|^2$$

其中 $L'^2 = \left(L + \frac{L^2}{\mu} + \frac{M\tau}{\mu} + \frac{LM\rho}{\mu^2}\right)^2$ 和 $\tilde{\nabla}\Phi(x_k)$

$$\widetilde{\nabla}\Phi(x_k) = \nabla_x f(x_k,y_k) - \nabla_x\nabla_y g(x_k,y_k)\left[\nabla_y^2 g(x_k,y_k)\right]^{-1}\nabla_y f(x_k,y_k)$$

命题 4 描述了目标函数值是如何由于变量 x 的一次 $\|v_k\|^2$ 迭代更新而减少的。这种减值也受到

梯度对 y 的矩和递归超梯度估计器的方差的影响。

4.2.3 定理 2

定理 2 描述 VRBO 的收敛特征。

若假设 1、2、3 成立。令 $\alpha=\frac{1}{20L_m^3},\beta=\frac{2}{13L_Q},\eta<\frac{1}{L},S_2\geq 2\left(\frac{L}{\mu}+\dot{1}\right)L\beta,m=\frac{16}{\mu\beta}-1,\dot{q}=\frac{\mu L\beta S_2}{\mu+L}$，其中 $L_m=\max\{L_Q,L_\Phi\}$。于是我们可以得到

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\|\nabla\Phi(x_k)\|^2\leq\mathscr{O}\left(\frac{Q^4}{K}+\frac{Q^6}{S_1}+Q^4(1-\eta\mu)^{2Q}\right)$$

定理 2 表明，VRBO 的收敛速度与迭代次数 K 呈亚线性失系。收敛误差由两个项组成。第一个误差项 $\frac{Q^6}{S_1}$ 是由外循环的小批量梯度和超梯度估计引起的，可以通过增加 S_1 批量大小来减少。第二个误差项是 $Q^4(1-\eta\mu)^{2Q}$ 由 Hessian 向量类型的超梯度估计的近似误差引起的，它随着 Q 以指数速度递减。

4.3 复杂性分析

定理 2 的相同条件下，选择 $S_1=\mathscr{O}\left(\varepsilon^{-1}\right),S_2=\mathscr{O}\left(\varepsilon^{-0.5}\right),Q=\mathscr{O}\left(\log\left(\frac{1}{\varepsilon^{0.5}}\right)\right),K=\mathscr{O}\left(\varepsilon^{-1}\right)$，然后，VRBO 找到一个 ε-稳态点，梯度复杂性为 $\widetilde{\mathscr{O}}\left(\varepsilon^{-1.5}\right)$，Hessian-vector 复杂度为 $\widetilde{\mathscr{O}}\left(\varepsilon^{-1.5}\right)$

通过在算法 2 中正确选择超参数，我们便可以得到 VRBO 复杂度系数为 $\widetilde{\mathscr{O}}\left(\varepsilon^{-1.5}\right)$。

4.4 总结

1. VRBO 是第一个采用递归方差减少法进行二层优化的方法。在 VRBO 中，内循环使用外循环中每个 epoch 计算的大批量的梯度估计值来构造方差降低的梯度和超梯度估计器。

2. 与 MRBO 类似，VRBO 涉及 Jacobian 和 Hessian 向量乘积的计算，而不是 Hessians 或 Hessians 的逆，从而一定程度上避免了繁杂的计算。

3. 同时我们可以通过后续实验中的数据结果，发现 VRBO 实现了较低的训练损失，表现出更快的收敛速度与更强的收敛稳定性。

4. 通过与 MRBO (以及其他基于单环动量的算法 MSTSA、STABLE 和 SEMA) 性能的比较，发现作为一种双环算法，VRBO 实际应用中明显优于所有现有的单回路动量算法（包括 MRBO）。

5. 与 MRBO 类似，作者通过严谨的数学推导，得出 VRBO 在计算的复杂度方面优于所有现有的随机双层优化算法。

5 代码复现

5.1 环境说明

系统	Windows 10
硬件	GTX 1650 和 Intel i7-10875H
语言	python 3.9.7
核心库	torch 1.8.0+cu111 torchvision 0.10.1 numpy 1.22.4

5.2 文件清单

5.2.1 目录

data：数据集存放，默认 MMIST

model：保存的部分训练模型

output：不同优化器的在训练过程中收敛速度、loss 变化等

results：模型在断点处参数的记录

5.2.2 辅助模块

paraminit.py：参数设定及初始化（借鉴了开源代码的实现思路）

preprocess.py：针对 MNIST 的数据预处理函数

grad.py：Inner-loop 中 fy、gy 梯度的相关运算

loss.py：损失函数的定义、针对训练集与测试集的损失计算

train.py：模型训练的若干通用函数

5.2.3 算法核心代码

stocBiO.py：一个随机梯度下降型优化器（stocBiO）

MRBO.py：作者在论文中所论述的第一种优化算法：基于单环动量的递归二层优化器

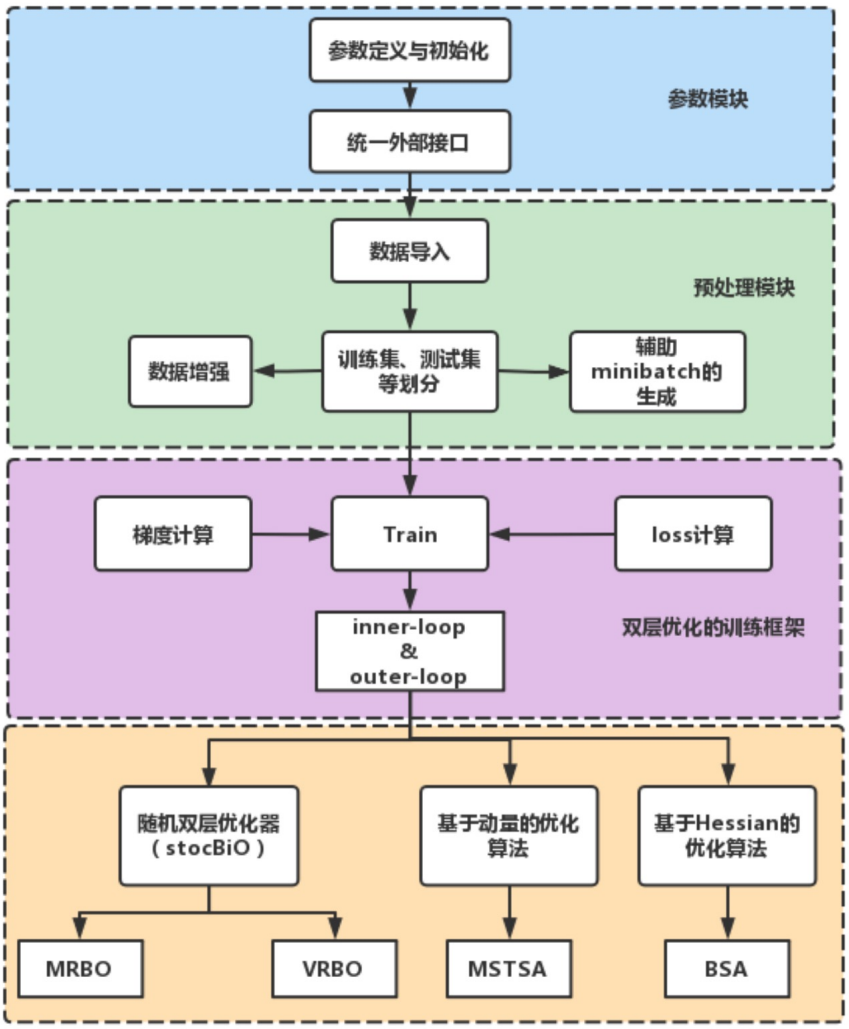
VRBO.py：作者在论文中所论述的第二种优化算法：基于方差减少技术的随机双层优化算法

MSTSA.py：Basiline 算法之一：MSTSA

BSA.py：Basiline 算法之一：BSA

5.2.4 模块设计

我们在代码复现的过程中，处于模块设计的综合性考虑，将代码部分主要划分为四个模块：参数设计模块、数据预处理模块、双层优化的训练框架、5 种核心算法的实现与测试模块。它们之间存在着高度的依赖与递进关系，功能与特性等可以由如下流程图所展示：



(a) 模块设计图

5.3 核心算法实现

5.3.1 随机双层优化器——stocBiO

根据论文的概述，在双层优化研究的早期，便有学者提出了一类基于约束的算法，试图用内部问题的最优性条件来惩罚外部函数。为了进一步简化基于约束的双层方法的实现，随后有学者提出了基于梯度的双层算法，而随机双层优化器 stocBiO 便是其中的重要成果之一，也是作者提出新算法的基础。因此，我们在复现阶段首先针对了 stocBiO 算法。

随机梯度下降

这个过程依赖于 Jacobian 矩阵、Hessian 矩阵以及 G 的梯度的计算，另外需要考虑随机变量、步长、迭代次数等相关参数的含义

```
# Hessian Matrix
z = []
ot = f_output(x[1], i_params)
Grad_gy = grad_gy(y[1], i_params, h_params, ot, f_regularize)
Grad_G = torch.reshape(i_params, [-1]) - args.eta * torch.reshape(Grad_gy, [-1])
for _ in range(Q):
    Jacobian = torch.matmul(Grad_G, v0)
    vnew = torch.autograd.grad(Jacobian, i_params)[0]
    v0 = torch.unsqueeze(torch.reshape(vnew, [-1]), 1).detach()
    z.append(v0)
vQ = args.eta * v0 + torch.sum(torch.stack(z), dim=0)
# G Gradient
ot = f_output(x[2], i_params)
Grad_gy = grad_gy(y[2], i_params, h_params, ot, f_regularize)
Grad_gy = torch.reshape(Grad_gy, [-1])
Grad_gyx = torch.autograd.grad(torch.matmul(Grad_gy, vQ.detach()), h_params, retain_graph=True)[0]
```

Inner-loop Outer-loop

构造两层的 optimization 优化过程，在 Inner-loop 即内层中使用 train set 更新 model，然后再 Outer-loop 基于更新后的 model 优化超参数。这是一个类似 meta-learning 的过程。

```
for epoch in range(args.epochs):
    i_grad = 0.0  # 一次 epoch 的 inner 梯度变化
    t_index = torch.randperm(args.batch_num) # 训练集索引
    for iter in range(args.iterations): # 迭代过程
        # 生成 minibatch 的数据....
        # 计算权重与一次迭代输出...
        weight = gamma[m_index * args.batch_size: (m_index + 1) * args.batch_size]

        output = f_output(images, i_params) # inner-loop 的一次输出
        # inner-loop 中相关更新值..
        i_params = i_params - args.inner_lr * i_update # inner-loop 参数更新


    v_data = create_val_data(args, v_index, x, y, device) # 生成验证集数据
    # 计算 outer 的超参数、outer-loop 中相关更新值...

    with torch.no_grad():

        weight = weight - args.outer_lr * torch.squeeze(o_update) # 更新权重
        gamma[o_index: o_index + args.batch_size] = weight # 将更新后权重赋值给超参数
        o_index = (o_index + args.batch_size) % args.train_size
    # 训练集与测试集 loss 计算
    train_loss = cal_train_loss(train_set, i_params, device, args.batch_num)
    test_loss = cal_test_loss(test_set, i_params, device)
```

5.3.2 基于单环动量的递归二层优化器——MRBO

以单循环方式进行，将动量递归技术应用于每次迭代时更新 x 和 y。在第 k 次迭代中，基于当前态 $G(y_k)$ 和先前态 $G(y_{k-1})$，使用小批量样本更新 y 与梯度估计器。步长 λ 在每次迭代时减小，以减小收敛误差。

为保证梯度估计的稳定性，根据迭代次数降低超参数的值，进而使得梯度估计量的计算权重更偏向于临近的先前态。

为降低算法的时间和空间复杂度，作者在算法中提出使用 Hessian 向量代替 Hessian 矩阵。论文

中给出了详细的算法描述，如下图所示：

Algorithm 1 Momentum-based Recursive Bilevel Optimizer (MRBO)

Input: Stepsize $\lambda, \gamma > 0$, Coefficients α_0, β_0, Initializers x_0, y_0, Hessian Estimation Number Q, Batch Size S, Constant $c_1, c_2, m, d > 0$
for $k = 0, 1, \ldots, K$ **do**
  Draw Samples $\mathcal{B}_y, \mathcal{B}_x = \{\mathcal{B}_j (j = 1, \ldots, Q), \mathcal{B}_F, \mathcal{B}_G\}$ with batch size S for each component
  if $k = 0$: **then**
    $v_k = \widehat{\nabla}\Phi(x_k; \mathcal{B}_x), u_k = \nabla_y G(x_k, y_k; \mathcal{B}_y)$
  else
    $v_k = \widehat{\nabla}\Phi(x_k; \mathcal{B}_x) + (1 - \alpha_k)(v_{k-1} - \widehat{\nabla}\Phi(x_{k-1}; \mathcal{B}_x))$
    $u_k = \nabla_y G(x_k, y_k; \mathcal{B}_y) + (1 - \beta_k)(u_{k-1} - \nabla_y G(x_{k-1}, y_{k-1}; \mathcal{B}_y))$
  end if
  update: $\eta_k = \frac{d}{\sqrt[3]{m+k}}, \quad \alpha_{k+1} = c_1 \eta_k^2, \quad \beta_{k+1} = c_2 \eta_k^2$
  $x_{k+1} = x_k - \gamma \eta_k v_k, \quad y_{k+1} = y_k - \lambda \eta_k u_k$
end for

(b) MRBO 算法

核心：Inner-loop Outer-loop

```
for epoch in range(args.epochs):
    i_grad = 0.0  # 一次 epoch 的 inner 梯度变化
    h_params = gamma[o_index: o_index + args.batch_size] # outer 的超参数
    # 生成验证集数据与 minibatch 的数据
    v_index = ..
    v_data = ..
    s_data, s_labels = v_data
```

```
    if not epoch: # 初始
        grad_x = stocBiO_sub(..) # inner-loop 的一次 x 梯度
        output = f_output(s_data[1], i_params) # inner-loop 的一次输出
        grad_y = grad_gy(..) # inner-loop 的一次 y 梯度
        # inner-loop 参数更新
        i_params, grad_y_old = i_params - args.inner_lr * eta_k * grad_y, grad_y
    else: # 2...k，此时迭代要考虑先前态
        # k与 k-1的 inner-loop 输出
        output = f_output(s_data[1], i_params)
        output_old = f_output(s_data[1], i_params_old)
        # k与 k-1的 update_y
```

```
        update_y = grad_gy(.., i_params, ..)
        update_y_old = grad_gy(.., i_params_old, ..)
        grad_y = update_y + (1-beta_k) * (grad_y_old-update_y_old)
        # inner-loop 参数更新
        i_params, grad_y_old = i_params, grad_y
        i_params = i_params - args.inner_lr * eta_k * grad_y
        # inner-loop 中相关更新值 ....

    # 结束本次迭代前，用变量记录内层参数
      i_params_old, h_params_old, grad_x_old, grad_y_old
            = i_params, h_params, grad_x, grad_y
      i_params = i_params - args.inner_lr * eta_k * grad_y,
      h_params = h_params - args.outer_lr * eta_k * grad_x
    o_update = torch.squeeze(o_update)
    i_grad = torch.norm(grad_y)
    # 结束本次迭代前，更新 weight、eta、alpha、beta
    weight = h_params
    eta_k = eta_k*(((epoch+m)/(epoch+m+1))**(1/3))
    alpha_k, beta_k = alpha_k*(eta_k**2), beta_k*(eta_k**2)
```

5.3.3 基于方差减少策略的递归二层优化器——VRBO

作者在论文中提到，虽然所有现有动量算法（参考文献 [2]、[22]、[11]）都是依据单回路方法设计的双层优化算法，而文献 [20] 中的经验结果表明，双回路两层算法可以获得比单循环算法更好的性能。

VRBO 采用文献 [4] 中提到 spider 方差减少技术进行二层优化，采用双回路的设计。VRBO 构造用于更新 x 和 y 的递归方差缩减梯度估计，其中 Outer-loop 中的 x 之后是 y 的 m+1 Inner-loop 更新。VRBO 划分 Outer-loop 迭代到各个 epoch，并在每个 epoch 开始计算超梯度估计量。之后，算法递归地更新 Gy 与梯度估计器。

与之前算法的显著不同在于，迭代过程需要两次划分 minibatch。

论文中给出了详细的算法描述，如下图所示：

Algorithm 2 Variance Reduction Bilevel Optimizer (VRBO)

Input: Stepsize $\beta, \alpha > 0$, Initializer x_0, y_0, Hessian Q, Sample Size S_1, S_2, Periods q
for $k = 0, 1, \ldots, K$ **do**
  if $\mathrm{mod}(k, q) = 0$: **then**
    Draw a batch $\mathcal{S}_1$ of i.i.d. samples
    $u_k = \nabla_y G(x_k, y_k; \mathcal{S}_1)$, $v_k = \widehat{\nabla}\Phi(x_k, y_k; \mathcal{S}_1)$
  else
    $u_k = \widetilde{u}_{k-1,m+1}$, $v_k = \widetilde{v}_{k-1,m+1}$
  end if
  $x_{k+1} = x_k - \alpha v_k$
  Set $\widetilde{x}_{k,-1} = x_k, \widetilde{y}_{k,-1} = y_k, \widetilde{x}_{k,0} = x_{k+1}, \widetilde{y}_{k,0} = y_k, \widetilde{v}_{k,-1} = v_k, \widetilde{u}_{k,-1} = u_k$
  for $t = 0, 1, \ldots, m+1$ **do**
    Draw a batch $\mathcal{S}_2$ of i.i.d samples
    $\widetilde{v}_{k,t} = \widetilde{v}_{k,t-1} + \widehat{\nabla}\Phi(\widetilde{x}_{k,t}, \widetilde{y}_{k,t}; \mathcal{S}_2) - \widehat{\nabla}\Phi(\widetilde{x}_{k,t-1}, \widetilde{y}_{k,t-1}; \mathcal{S}_2)$
    $\widetilde{u}_{k,t} = \widetilde{u}_{k,t-1} + \nabla_y G(\widetilde{x}_{k,t}, \widetilde{y}_{k,t}; \mathcal{S}_2) - \nabla_y G(\widetilde{x}_{k,t-1}, \widetilde{y}_{k,t-1}; \mathcal{S}_2)$
    $\widetilde{x}_{k,t+1} = \widetilde{x}_{k,t}, \widetilde{y}_{k,t+1} = \widetilde{y}_{k,t} - \beta \widetilde{u}_{k,t}$
  end for
  $y_{k+1} = \widetilde{y}_{k,m+1}$
end for

(c) VRBO 算法

VRBO 的单次过程需要精心考虑

```
def VRBO_iter(args, v_data, i_params, h_params, h_params_old, grad_x, grad_y):
    x, y = v_data[0]
    ot = f_output(x[1], i_params) # inner-loop 层输出
    # 初始 xy 的更新值
    dy = grad_gy(y[1], i_params, h_params, ot, f_regularize)
    dy_old = grad_gy(y[1], i_params, h_params_old, ot, f_regularize)
    dx = stocBiO_sub(.., h_params,..)
    dx_old = stocBiO_sub(.., h_params_old,..)
    # 构造用于更新 x 和 y 的递归方差减少梯度估计器
    v_t, u_t = grad_x + dx - dx_old, grad_y + dy - dy_old
    # 更新并记录参数信息
    i_params_new = i_params - args.inner_lr * u_t
    for t in range(args.iterations):
```

装 订 线

```
    # 生成 minibatch 的数据
    x, y = v_data[t + 1]
    ot = f_output(x[1], i_params_new) # inner-loop 层输出
    # 迭代中 x、y 的更新值，对应原文算法描述中的xk xk-1
    dy = grad_gy(y[1], i_params_new, h_params, ot, f_regularize)
    ot = f_output(x[1], i_params)
    dy_old = grad_gy(y[1], i_params, h_params, ot, f_regularize)
    dx = stocBiO_sub(.., i_params_new, ...)
    dx_old = stocBiO_sub(.., i_params, ...)
    # 构造用于更新 x 和 y 的递归方差减少梯度估计器
    v_t = v_t + dx - dx_old
    u_t = u_t + dy - dy_old
    i_params = i_params_new
    i_params_new = i_params - args.inner_lr * u_t # 更新并记录参数信息

# 返回最终的参数信息与梯度估计器
return i_params_new, v_t, u_t
```

Inner-loop Outer-loop

仍然是基于双层优化的框架进行设计，与 MRBO 相似，此处便不再赘述

5.3.4 其他 Baseline 算法——MSTSA 与 BSA

MSTSA

文献 [22](A momentum-assisted single-timescale stochastic approximation algorithm for bilevel optimization) 提出了一种通过文献 [3](Momentum-based variance reduction in non-convex SGD) 引入的基于动量的递归技术来更新 x 的算法 MSTSA。

我们阅读了被引用的相关文献，发现 MSTSA 算法巧妙之处在于更新 Outer-loop 梯度的过程。这也是复现过程中的易错点。

```
def MSTSA_sub(args, o_update_old, c_eta, v_data, i_params, i_params_new, h_params, h_p
    arams_old, f_output, f_regularize):
    """
    MSTSA 算法更新 outer 梯度的过程
    """
    grad_new = stocBiO_sub(.., i_params_new, h_params, ...)
    grad_pri = stocBiO_sub(.., i_params, h_params_old, ...)
    o_update = c_eta * grad_new + (1 - c_eta) * (o_update_old + grad_new - grad_pri)
    return o_update
```

BSA

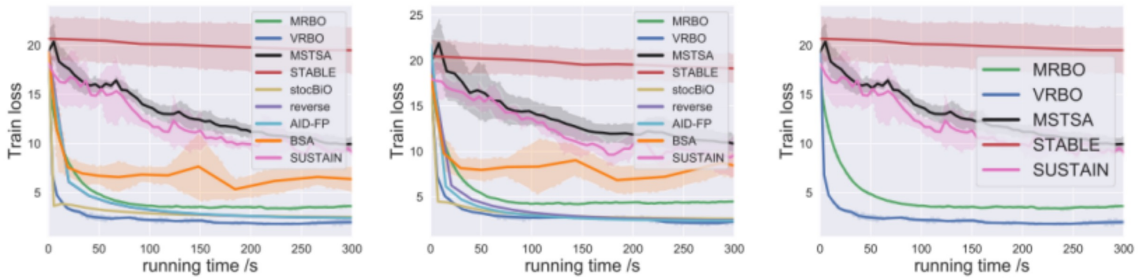
即双回路随机优化算法，算法本质上与 SGD 方法、stoBiO 算法类似，但优化了 Hessian 向量计算的有关步骤，降低了时间复杂度。

6 算法综合比较

6.1 原文的实验与结论

在原论文中的实验部分，作者将提出的两种优化算法 MRBO 与 VRBO 的性能与其他双层优化算法进行了综合比较：BSA（双环随机算法）、AID-FP（双环决策算法）、MSTSA（单回路随机算法），STABLE（带 Hessian 的单循环随机算法逆向计算）和 stocBiO（双环随机算法）。

实验最终数据得到原文中的 Figure1，如下图所示:



(a) Noise rate $p = 0.1$ (b) Noise rate $p = 0.15$ (c) Noise rate $p = 0.1$

Figure 1: training loss v.s. running time.

作者由此验证了 VRBO 相比于 SGD 型的 stocBiO 算法，有着更低的 training loss 与更稳定的收敛过程。并且 VRBO 的收敛速度明显快于所有基于动量的单回路方法。

此外，MRBO 实现了最快的所有基于单圈动量的算法的收敛速度，这与理论推导的结果一致。

综合上述结果，作者得到了如下重要结论：双回路算法往往比单回路算法有更快的收敛速度!

并由以下三点加以佐证：

（1）双回路 VRBO 在所有算法中表现最好算法；

（2）双回路 SGD 型 stocBiO、GD 型 reverse 和 AID-FP 比单回路动量加速随机算法性能更佳；

（3）双环 SGD 型 BSA 的收敛速度快于单环动量加速随机 MSTSA.

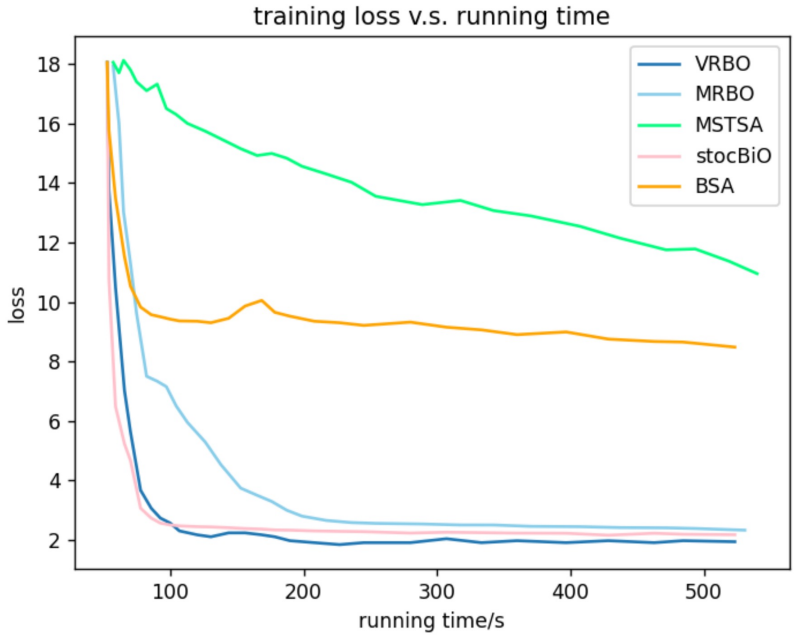
作者进行了如下的说明与解释：

这种现象仅在双层优化中观察到。相反地发生在极小化和极大极小问题中，单回路算法的性能明显优于双回路算法。原因可能是，双层优化中外环处的超梯度估计对 inner-loop 输出非常敏感。因此，对于每个 outer-loop 迭代，双回路结构中足够的 inner-loop 迭代提供了更精确的输出 y*(x)，因此有助于估计更准确的 outer-loop 中的超梯度。这进一步促进了更好的 outer-loop 迭代，并更快地生成总体趋同。

6.2 复现结果

我们在代码复现部分，共完成了 5 种算法：stocBiO（双层随机梯度下降算法）、作者提出的 MRBO 与 VRBO 算法、和作为 Baseline 的 MSTSA （单回路随机算法）、BSA（双回路随机算法）。

在本节，我们同样使用预处理后的 MNIST 数据集进行训练与测试。得到的训练数据如下图所示。虽然我们复现过程与作者存在细小差别，且硬件、系统等层面有一定差异，但仍然可以观察到，5 种算法的 loss 随时间下降的曲线，整体趋势与变动情况与作者所得结果一致。这也验证了我们复现过程的正确性。



(a) 基于 MNIST 数据集的算法性能比较

训练的参数设置

训练集规模	20000
测试集规模	500
验证集规模	5000
Batch 大小	480
外层 Epochs	40
内层 iterations	200
外层学习率	0.1
内层学习率	0.1
噪声比	0.1

VRBO、MRBO、stocBiO 算法的最终收敛结果基本一致。相比之下，VRBO 有着更稳定的收敛过程。相比知下，作为 Baseline 的 MSTSA、BSA 等算法，loss 变化与收敛速度表现较差，且在训练初期容易出现较大波动。

7 算法拓展改进

7.1 改进点简述

由于双层优化的嵌套结构，即使是基于梯度的方法，Outer-loop 的梯度（即超梯度）也需要涉及内环函数的 Jacobian 和 Hessian 向量进行一、二阶偏导的计算。无论是求导还是矩阵求逆运算，在实践中可能都是非常耗费计算资源的，且扩展性差。

以上述问题为切入点，我们搜集了相关资料，发现有学者提出了无 Hessian 的双层优化算法，其总体思路是使用零阶或一阶方法来近似双层问题的全部超梯度。然而，这种近似可能会导致方差的增大和训练结果的不稳定性，但是如果只估计响应的 Jacobian 矩阵作为超梯度的部分组成部分，结果证明是非常有效的。

在此基础上，我们尝试了一种新的无 Hessian 方法，采用类似零阶的处理，通过取两个优化路径之间的差值来逼近响应 Jacobian 矩阵，以降低计算的复杂度。

7.2 背景知识

7.2.1 超强辐射：Hypergradients

目前，基于梯度的双层优化器的关键步骤往往是估计超梯度（即目标相对于外部变量 x 的梯度）。它的形式如下：

$$\nabla\Phi(x) = \nabla_x f(x, y^*(x)) + \mathscr{J}_*(x)^\top \nabla_y f(x, y^*(x))$$

其中，Jacobian 向量 $\mathscr{J}_*(x) = \frac{\partial y^*(x)}{\partial x} \in \mathbb{R}^{d\times p}$ 根据 Lorraine 等人（2020）的研究，可以看出 $\mathscr{J}_*(x)$ 包含两个分量：直接梯度 $\nabla_x f(x, y^*(x))$ 和间接梯度 $\mathscr{J}_*(x)^\top \nabla_y f(x, y^*(x))$。其中，直接梯度可以用现有的自动微分技术进行有效的计算。然而，间接梯度的计算过程仍非常复杂，对于高维的二层优化问题（如神经网络变量），即使通过设计 Jacobian 向量和 Hessian 向量乘积来替代二阶计算，计算的代价仍是非常高。

7.2.2 零阶近似：Zeroth-Order Approximation

Zeroth-order 近似是一种强大的技术，当评估梯度不可行（如黑箱问题）或计算成本高时，可以根据函数值来估计函数的梯度。

Spokoiny(2017) 提出的零阶近似方法的思想是，使用仅基于函数的值来近似一般黑箱函数 h : $\mathbb{R}^n \to \mathbb{R}$ 的梯度

$$\widehat{\nabla} h(x;u) = \frac{h(x+\mu u) - h(x)}{\mu} u$$

这是一个高斯随机向量，$\mu > 0$ 是平滑参数。这样的 oracle 可以被证明是平滑函数 $\mathbb{E}_u[h(x+\mu u)]$ 的梯度的无偏估计。

7.3 算法设计

7.3.1 基于部分零阶的二层优化器

我们尝试使用 Jacobian 估计器设计了一个确定性的二层优化器，而其中零阶估计器只用于估计部分超梯度。在算法的每一步 k，算法运行 N 步长全 GD 来近似 $y_k^N(x_k+\mu u_{k,j})$。然后，算法对 Q 个高斯向量进行采样 $\{u_{k,j}\in\mathcal{N}(0,I), j=1,\ldots,Q\}$，对于每个样本 $u_{k,j}$ ，运行 N 步全 GD 来近似 yN (xk + ţuk,j)，然后计算 Jacobian 估计器 $\hat{\mathscr{J}}_N(x;u_{k,j})$。然后，Q 估计器的样本平均值被用来构建以下超梯度估计器，以更新外部变量 x。

$$\begin{aligned}\widehat{\nabla}\Phi(x_k) &= \nabla_x f\left(x_k, y_k^N\right) + \frac{1}{Q}\sum_{j=1}^{Q}\hat{\mathscr{J}}_N^T(x_k;u_{k,j})\nabla_y f\left(x_k, y_k^N\right)\\ &= \nabla_x f\left(x_k, y_k^N\right) + \frac{1}{Q}\sum_{j=1}^{Q}\left\langle\delta(x_k;u_{k,j}), \nabla_y f\left(x_k, y_k^N\right)\right\rangle u_{k,j}\end{aligned}$$

将该估计器的算法策略，与 MRBO 与 VRBO 等双层优化算法结合，能够省略复杂的 Hessian 向量或 Jacobian 向量的乘积运算，代之以梯度计算，因此理论上能够取得更高效率。

7.3.2 基于部分零阶的随机二层优化器

在机器学习应用中损失函数 f，g 经常采取有限和的形式表达给定的数据 $\mathscr{D}_{n,m} = \{\xi_i, \zeta_j, i=1,\ldots,n, j=1,\ldots,m\}$，形式如下：

$$f(x,y) = \frac{1}{n}\sum_{i=1}^{n}F(x,y;\xi_i), \quad g(x,y) = \frac{1}{m}\sum_{i=1}^{m}G(x,y;\zeta_i)$$

其中，样本量 n 和 m 通常非常大。对于这样一个大规模数据量的场景，我们联系了原论文中的观点—— “随机双层优化器，在数据量很大或需要在运行时对大量新数据进行采样的场景中实现比确定性方法更好的效率”，在原有确定性方法的基础上，尝试引入随机性的处理策略。

最终，我们尝试设计了一个部分零阶的随机二层优化器，使用 N 次随机梯度下降（SGD）步骤来寻找 $\left\{Y_k^N, Y_{k,1}^N, \ldots, Y_{k,Q}^N\right\}$ 的内部问题，并且将每个外部变量集设置为 $x_k+\mu u_{k,j}$。所有 SGD 的运行可遵循相同的批量采样路径，随后，送入 Jacobian 估计器。在外部层面，算法从内部数据集中独立抽取一个新的 batch$\mathscr{D}_F$ 来评估随机梯度 $\nabla_x F(x_k, Y_k^N;\mathscr{D}_F)$ 和 $\nabla_y F(x_k, Y_k^N;\mathscr{D}_F)$。

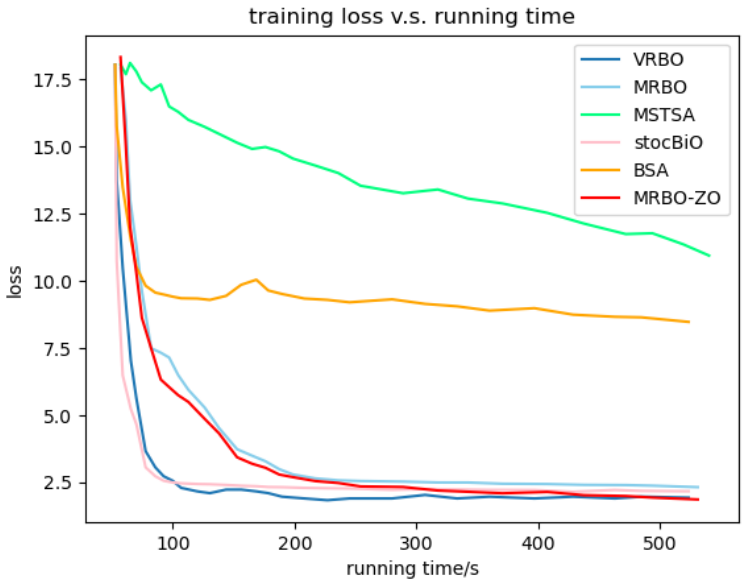
因此，超梯度的估计可采用如下方式:

$$\nabla_x F\left(x_k, Y_k^N; \mathscr{D}_F\right) \widehat{\nabla} \Phi\left(x_k\right)=\nabla_x F\left(x_k, Y_k^N; \mathscr{D}_F\right)+\frac{1}{Q} \sum_{j=1}^{Q}\left\langle \delta\left(x_k; u_{k, j}\right), \nabla_y F\left(x_k, Y_k^N; \mathscr{D}_F\right)\right\rangle u_{k, j}$$

7.4 算法测试

为了验证上述优化算法的有效性，我们尝试用代码，在已实现的二层随机优化算法框架的基础上，初步实现了基于部分零阶的随机二层优化器，并将其中的零阶近似等策略应用于 MRBO 算法的优化。为后续讨论方便，我们暂且将上述方法命名为 MRBO-ZO。

我们仍在在 MNIST 数据集上进行实验与测试，且数据的预处理方法、内层模型训练的相关参数与第六章相同，以排除其他无关因素的干扰。最后，我们综合比较了的 MRBO-ZO 和 5 个已复现的随机二层优化器，包括 stocBiO、MRBO 和 VRBO、MSTSA 和 BSA。训练过程中 loss 随时间变化如下图所示：



(a) 基于 MNIST 数据集的优化算法性能比较

由此可以看出我们改进后的算法 MRBO-ZO 相比于原始 MRBO，能够以更快的速度收敛，在训练了 40 个 epoch 后，loss 值略低于 MRBO、VRBO 等算法，且 loss 曲线在 40 个 epoch 时仍有一定的下降趋势，这反映了训练后期部分零阶方法对于优化效率的提升。

8 总结

应用随机过程是随机数学的一个重要分支，在课程学习过程中整体上以理论知识为主，理论抽象性比较强。对于我们专业的学习需求来说，大数据在提供海量信息的同时，也暴露了传统计算方法效率低的问题，且目前在大规模机器学习问题的求解中，随机优化算法占据着不可替代的地位。在本次论文复现过程中，我们初次通过前沿论文进行随机过程相关方法的实践。

在对于论文的具体实践过程中，我们首先进行了模块设计，将复现工作分为了参数设计模块、数据预处理模块、双层优化的训练框架、多种优化算法的实现与测试模块。之后，我们层层递进。首先基于 Jacobian 矩阵、Hessian 矩阵以及 G 梯度的计算，复现了随机双层优化算法 stocBiO。在此过程中，我们也对模型训练的 epoch、batch size、learning rate 等参数进行了有效调整。之后，我们反复研读、讨论论文中两种创新算法。在复现 MRBO 算法的过程中，我们加入单回路动量递归优化，每次同步更新 x 和 y，通过 Hessian 向量代替 Hessian 矩阵，降低时间复杂度。另外，我们也实现了基于方差减少策略的递归二层优化器 VRBO，通过划分外层迭代 epoch 并计算超梯度估计量，同样实现了对复杂度的较大优化。在实现原文两种算法的基础上，我们还广泛阅读了被引文献，并针对 MSTSA、BSA 等典型的 Baseline 算法进行了复现，进而完成了多种双层优化算法的综合对比分析。

时间复杂度表格如下所示：

算法	$\mathrm{Gc}(F,\varepsilon)$	$\mathrm{Gc}(G,\varepsilon)$	$\mathrm{JV}(G,\varepsilon)$	$\mathrm{HV}(G,\varepsilon)$	$\mathrm{Hyy}^{\mathrm{inv}}(G,\varepsilon)$
MSTSA	$O(\varepsilon^{-2})$	$O(\varepsilon^{-2})$	$O(\varepsilon^{-2})$	$\widetilde{\mathcal{O}}(\varepsilon^{-2})$	/
stocBiO	$O(\varepsilon^{-2})$	$O(\varepsilon^{-2})$	$O(\varepsilon^{-2})$	$\widetilde{\mathcal{O}}(\varepsilon^{-2})$	/
VRBO（我们复现的）	$\widetilde{\mathcal{O}}(\varepsilon^{-1.5})$	$\widetilde{\mathcal{O}}(\varepsilon^{-1.5})$	$\widetilde{\mathcal{O}}(\varepsilon^{-1.5})$	$\widetilde{\mathcal{O}}(\varepsilon^{-1.5})$	/
MRBO（我们复现的）	$O(\varepsilon^{-1.5})$	$O(\varepsilon^{-1.5})$	$O(\varepsilon^{-1.5})$	$\widetilde{\mathcal{O}}(\varepsilon^{-1.5})$	/
MRBO-ZO（我们优化的）	$O(\varepsilon^{-1.5})$	$O(\varepsilon^{-1.5})$	$O(\varepsilon^{-1.5})$	$O(\varepsilon^{-1.5})$	$O(\varepsilon^{-1.5})$
BSA	$O(\varepsilon^{-1.5})$	$O(\varepsilon^{-1.5})$	$O(\varepsilon^{-1.5})$	/	$O(\varepsilon^{-1.5})$

综上，我们在代码复现部分完成 stocBiO（双层随机梯度下降算法）、作者提出的 MRBO 与 VRBO 算法、作为 Baseline 的 MSTSA（单回路随机算法）、BSA（双回路随机算法）共五种算法。并利用预处理后的 MNIST 数据集进行训练测试，最终复现得到的数据结果与原文基本一致。

在算法创新方面，我们以超强辐射、零阶近似等原理为基础，尝试设计了基于部分零阶的二层优化器以及基于部分零阶的随机二层优化器。采用一种新的无 Hessian 方法，与类似零阶的处理，通过取两个优化路径之间的差值来逼近响应 Jacobian 矩阵，以降低计算的复杂度。通过测试发现，改进后的算法 MRBO-ZO 相比于原始 MRBO，能够以更快的速度收敛，在训练了 40 个 epoch 后，loss 值略低于 MRBO、VRBO 等算法，且 loss 曲线在 40 个 epoch 时仍有一定的下降趋势，这反映了训练后期部分零阶方法对于算法效率的提升。

总的来说，我们对于论文进行了系统的了解和学习，在学习过程中，我们依照作者的思路拾级而上，对此前已经较为成熟的算法进行了解，对相关背景进行知识补充。在对双层优化的理论基础和元学习的基本逻辑进行系统性的了解之后，我们深入理解凸优化问题的良好性质和解决逻辑，参数层面则了解了超参数优化在机器学习中的应用和调整方法。我们在文章学习过程中领悟了作者通

过对现有二层优化器的思考，引入单环动量的递归与方差减少技术，同时尝试将随机梯度下降的优化器与双重循环的优化器相结合，达到更好地加速效果。通过对论文的复现，我们不仅加强了对论文所涉及的双层优化等知识概念的认知，也深刻认识到在科研过程中，从论文中汲取经验的学习方法的意义价值。在往后我们也会更加努力深入地去拜读借鉴先进学者的理论与实践，进行更加深入的研究加深对专业知识的理解并提高自己的专业能力。

9 成员分工与自评

9.1 小组分工

胡孝博：主要负责代码复现工作，完成了报告中代码复现与算法综合比较部分。

徐奕：主要对项目各部分进行整合、调试，对部分算法进行优化并撰写报告。

张家瑞：主要负责对论文的行文逻辑梳理，对背景知识补充，对复现结果进行总结，撰写此部分报告。

9.2 成员自评

胡孝博：主要负责本次作业代码复现工作，包括模块设计、五种优化算法的实现与调试、模型训练、结果分析与可视化等。在此基础上，相应地完成了报告中第 5、6 部分的撰写。在本次随机过程课程大作业中，我以课程的理论知识为基础，尝试阅读顶会论文，通过搜集文献资料等方法开拓了自己的知识面，对科研工作、科研领域等有了初步了解。将抽象的算法描述复现为代码，需要我们对论文内容的深刻理解，更需要我们细致耐心、积极探索、勇于尝试。通过本次实验，我与队友分工协作，加深了对双层优化算法、基于动量或方差减少策略算法、随机化方法、单环设计与双环设计等概念的理解，提高了文献阅读能力、编程实践能力与自主探究能力。

徐奕：本次实验我主要负责对项目各部分进行整合、调试，对部分算法进行优化并撰写报告。在项目的整体整合和调试过程中，提高了我的动手实践的能力，也让我更加深刻的了解在算法背后的数理知识，在论文的复现过程中关注到了许多在实现过程中的细节，；在优化部分算法的过程中加深了对双层优化以及零阶近似等方面的认识了解。在整个实验进程中，能认真设计了自己负责的部分，同时也对实验的各个模块进行了合理的整合，调试过程中也始终保持耐心，关注每一个细节，根据运行情况对各个部分进行适应性的调整，并及时给其他两位组员反馈，共同修改，解决问题。

张家瑞：本次实验我在对项目逻辑进行梳理，在此过程中了解到更多基于现有科学成果进行科研的方法，提高了科研能力。对背景知识进行补充的过程中，了解到一些随机过程及机器学习相关问题定义和解决方法，丰富了知识储备。在实验总结过程中，了解论文行文逻辑和具体成果，加深了对论文研究成果的理解。在整个复现过程中，和队友积极沟通，对项目的团队设计有了更好的认识，共同完成了此次论文复现。

参考文献

[1] Yang J, Ji K, Liang Y. Provably faster algorithms for bilevel optimization[J]. Advances in Neural Information Processing Systems, 2021, 34: 13670-13682.

[2] Daouda Sow, Kaiyi Ji, Yingbin Liang On the Convergence Theory for Hessian-Free Bilevel Algorithms,arXiv:2110.07004.

[3] T. Chen, Y. Sun, and W. Yin. A single-timescale stochastic bilevel optimization method. arXiv preprint arXiv:2102.04671, 2021.

[4] K. Ji, J. Yang, and Y. Liang. Bilevel optimization: Nonasymptotic analysis and faster algorithms.In International Conference on Machine Learning (ICML), 2021.

[5] P. Khanduri, S. Zeng, M. Hong, H.-T. Wai, Z. Wang, and Z. Yang. A momentum-assisted single-timescale stochastic approximation algorithm for bilevel optimization. arXiv preprint arXiv:2102.07367v1, 2021.

[6] H. Rafique, M. Liu, Q. Lin, and T. Yang. Weakly-convex–concave min–max optimization:provable algorithms and applications in machine learning. Optimization Methods and Software,pages 1–35, 2021.

[7] Q. Tran-Dinh, N. H. Pham, D. T. Phan, and L. M. Nguyen. Hybrid stochastic gradient descent algorithms for stochastic nonconvex optimization. arXiv preprint arXiv:1905.05920, 2019.

[8] Z. Wang, K. Ji, Y. Zhou, Y. Liang, and V. Tarokh. Spiderboost: A class of faster variance-reduced algorithms for nonconvex optimization. arXiv preprint arXiv:1810.10690, 2018.

[9] R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo. On the iteration complexity of hypergradient computation. In International Conference on Machine Learning (ICML), pages 3748–3758,2020.

[10] Z. Guo, Y. Xu, W. Yin, R. Jin, and T. Yang. On stochastic moving-average estimators for non-convex optimization. arXiv preprint arXiv:2104.14840, 2021.

[11] P. Hansen, B. Jaumard, and G. Savard. New branch-and-bound rules for linear bilevel programming. SIAM Journal on Scientific and Statistical Computing, 13(5):1194–1217, 1992.

[12] G. M. Moore. Bilevel programming algorithms for machine learning model selection. Rensselaer Polytechnic Institute, 2010.

[13] C. Shi, J. Lu, and G. Zhang. An extended kuhn-Tucker approach for linear bilevel programming. Applied Mathematics and Computation, 162(1):51–63, 2005.

装 订 线