Principled and Efficient Bilevel Optimization for Machine Learning

Riccardo Grazzi

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

October 2, 2023

I, Riccardo Grazzi, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Automatic differentiation (AD) is a core element of most modern machine learning libraries that allows to efficiently compute derivatives of a function from the corresponding program. Thanks to AD, machine learning practitioners have tackled increasingly complex learning models, such as deep neural networks with up to hundreds of billions of parameters, which are learned using the derivative (or gradient) of a loss function with respect to those parameters. While in most cases gradients can be computed exactly and relatively cheaply, in others the exact computation is either impossible or too expensive and AD must be used in combination with approximation methods. Some of these challenging scenarios arising for example in meta-learning or hyperparameter optimization, can be framed as bilevel optimization problems, where the goal is to minimize an objective function that is evaluated by first solving another optimization problem, the lower-level problem. In this work, we study efficient gradient-based bilevel optimization algorithms for machine learning problems. In particular, we establish convergence rates for some simple approaches to approximate the gradient of the bilevel objective, namely the *hypergradient*, when the objective is smooth and the lower-level problem consists in finding the fixed point of a contraction map. Leveraging such results, we also prove that the projected inexact hypergradient method achieves a (near) optimal rate of convergence. We establish these results for both the deterministic and stochastic settings. Additionally, we provide an efficient implementation of the methods studied and perform several numerical experiments on hyperparameter optimization, meta-learning, data-poisoning and equilibrium models, which show that our theoretical results are good indicators of the performance in practice.

Impact Statement

Gradient-based bilevel optimization methods have recently started to be more popular in machine learning research, while they are still rarely employed in industrial applications. The bilevel formulation allows to cover a wide variety of problems, but the complexity and high time and memory cost of gradient-based bilevel methods make them often less appealing than simpler and cheaper alternatives based on heuristics or tailored to specific applications. For these reasons, we foresee that this work will affect almost exclusively the academic world, at least until either a suitable industrial application is found or bilevel methods become easier to set up and/or cheaper to run.

This thesis can be seen as an effort to build a quantitative theoretical foundation for gradient-based bilevel optimization methods. The scale of modern bilevel optimization problems combined with the complexity of gradient-based methods used to solve them makes it difficult and expensive to perform comprehensive experimental evaluations, and our analysis provides an alternative way to compare different bilevel methods in terms of iteration and sample complexity. This may be useful to researchers and practitioners for designing methods which are provably better than established ones, but also to better understand strength and weaknesses of each method and when it is suitable to apply.

The experimental part of this work validates the theory on different small and medium scale problems in a variety of scenarios, and is accompanied by open-source code designed to be (and that has been) used by researchers to develop new methods and to solve custom bilevel problems.

Acknowledgements

I would like to thank my supervisor Massimiliano Pontil, and Saverio Salzo, which helped me immensely in finding and carving a good research path. In particular, I greatly thank Massimiliano for allowing me to explore freely many research directions and for giving me confidence during meetings with his positive attitude, and Saverio for encouraging me to tackle more theoretical aspects of optimization and for the numerous hours spent double-checking and correcting errors in my sometimes convoluted reasoning.

Thanks to all former and present members of the CSML-IIT gang. Although the group is scattered, I had the opportunity to share many meaningful moments with really wonderful people, some of whom became friends. Special thanks to Luca, for being my mentor at the beginning of the PhD and for the many drinks and illuminating discussions we had around the world. Thanks to Giulia, Jordan, Michele, Arya, Isak, Dimitri, Leonardo, Andrea, Pietro and Vladimir for all the math discussions and for the time spent together, mostly divided between pubs and hikes, with the occasional board game.

During these years, I had the great opportunity to do a four months internship at Amazon AWS in Berlin. Many thanks to Cédric Archambeau, Matthias Seeger and Valentin Flunkert, for supporting me during this period and even after and for being wonderful collaborators. Thanks also to Julien, my buddy intern and dear friend, especially for the meals shared outside in the cold autumn, too often followed by a warming Glühwein mit Schuss.

I wish to thank also my examiners Alain Zemkoho and Jin Bangti for scrutinizing this manuscript and for their helpful feedback, corrections, and words of

appreciation during the defence.

Un caloroso grazie to all my hometown friends, for being present during my sudden appearances in Valdarno and for coming to visit me in Genoa. I am also enormously grateful to my parents Andrea and Anna and to my brother Alberto for being always supportive. Last but not least, I would like to greatly thank Melanie, for sharing her life with me and managing to brighten even my darkest days.

Contents

List of Abbreviations

- AD: Automatic Differentiation.
- FMAD: Forward Mode Automatic Differentiation.
- RMAD: Reverse Mode Automatic Differentiation.
- ITD: Iterative Differentiation.
- AID: Approximate Implicit Differentiation.
- SID: Stochastic Implicit Differentiation.
- UL: Upper-Level.
- LL: Lower-Level.
- LS: Linear System.
- GD: Gradient Descent.
- SGD: Stochastic Gradient Descent.
- HPO: Hyperparameter Optimization.
- w.r.t.: with respect to.
- i.i.d.: independent and identically distributed.
- LHS: Left Hand Side.
- RHS: Right Hand Side.

List of Figures

List of Tables

Chapter 1

Introduction

The complexity of machine learning problems and algorithms is rapidly increasing thanks to the exponential rise in computational power. Nowadays, practitioners can develop and test new and existing methods with unprecedented ease, aided by powerful and reliable software libraries. A core element of recent machine learning libraries such as PyTorch (Paszke et al., 2019a), JAX, (Bradbury et al., 2018) and TensorFlow (Abadi et al., 2015) is automatic differentiation (AD) (Griewank and Walther, 2008), which is a way to automatically and efficiently compute exact derivatives from a program which evaluates a mathematical function. For instance, reverse mode AD (or backpropagation) allows to compute the gradient of the loss w.r.t. the parameters of deep learning models (Goodfellow et al., 2016) such as Convolutional Networks (Krizhevsky et al., 2012), LSTMs (Hochreiter and Schmidhuber, 1997), and Transformers (Vaswani et al., 2017), which would be hard and expensive to evaluate using other methods. The loss function is a scalar measure of the error the model commits on a given task, and its gradient is exploited by gradient-based optimization methods to learn the parameters minimizing the loss. Gradients are fundamental to learn models with a high number of parameters, since the cost of learning by relying only on loss values is generally too high.

In some situations however, even with the help of AD, computing derivatives is more challenging. For example in gradient-based hyperparameter optimization the dataset is split into training and validation, and we want to compute the gradient with respect to the hyperparameters of the validation loss evaluated on the optimal model

parameters, i.e. the ones minimizing the training loss. However, when this gradient is well-defined it is often very expensive or impossible to compute exactly since the optimal model parameters are the solution of another optimization problem. A similar hurdle is present also in meta-learning, where the goal is to learn a meta-model capable to adapt quickly to new tasks. Moreover, other similarly challenging settings studied in this work are poisoning adversarial attacks and learning models whose internal representation is the fixed point of a certain parametrized map. Instances of the latter are some kind of recurrent neural networks (Almeida, 1987; Pineda, 1987), graph neural networks (Scarselli et al., 2008) and more recently deep equilibrium models (Bai et al., 2019).

Most of these challenging scenarios fit into the mathematical framework of Bilevel Optimization (Franceschi et al., 2018; Franceschi, 2021), where the goal (or *upper-level problem*) is to minimize a function f where some variables are the solution of another optimization problem (*the lower-level problem*).

Bilevel optimization problems have a long history and have been originally referred to as Stackelberg Games (Von Stackelberg, 2010), where upper and lower level problems are solved by two players, the leader and follower respectively. A large amount of literature of bilevel optimization has been published by the operations research community and initially focused on linear bilevel problems: having linear constraints and linear upper and lower level objectives. However, even bilevel problem of this simple kind are usually non-convex and are often solved via global optimization algorithms like branch and bound (Hansen et al., 1992), that are less suited for high-dimensional problems.

In contrast, many bilevel problems in machine learning are high-dimensional, have usually a weaker structure for the lower-level objective, which can be quadratic, strongly convex, or even non-convex, and have either few or no constraints at the lower-level. Machine learning practitioners often use simple black-box approaches that ignore the bilevel structure such as grid or random search, which are intuitive and easy to implement. More sophisticated alternatives are Bayesian optimization methods (Snoek et al., 2012), which use past function evaluations to decide which

points to evaluate next, balancing exploration and exploitation. However, these black-box strategies struggle to optimize the large number of upper-level parameters present for example in some meta-learning problems, where e.g. the upper-level parameters are the weights of a neural network.

Recent works have instead shown the advantage of gradient-based[1] bilevel methods that rely on approximations of the so called *hypergradient*, i.e. the gradient of the upper-level objective, to solve bilevel optimization problems. In hyperparameter optimization for example, Pedregosa (2016) showed that gradient-based methods can find the best regularization parameters faster than random search and Bayesian optimization, while Lorraine et al. (2020) showed that these methods can also optimize millions of hyperparameters. In meta-learning the success of gradient-based methods is even more striking (Finn et al., 2017; Rajeswaran et al., 2019; Denevi et al., 2019a), since we usually deal with many meta-parameters. Other examples of the success of gradient-based optimization methods are in neural architecture search Liu et al. (2018) and dataset poisoning attacks (Biggio et al., 2012; Muñoz-González et al., 2017).

1.1 The Bilevel Fixed-Point Framework

In this section, we describe the mathematical bilevel framework that is the subject of this manuscript. Let $\Lambda \subseteq \mathbb{R}^n$ be closed and convex, $E\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}$ be the upper-level objective and $\Phi\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}^d$ be the lower-level fixed-point map. We consider the following bilevel problem

$$\begin{aligned} &\min_{\lambda \in \Lambda} f(\lambda) := E(w(\lambda), \lambda) \\ &\quad \text{with} \;\; w(\lambda) = \Phi(w(\lambda), \lambda). \end{aligned}$$

The *upper-level* problem is that of finding the minimizer of f, while the lower-level problem is that of computing $w(\lambda)$, i.e. the fixed point of Φ. This fixed-point formulation is different from the classical bilevel formulation where we have

[1] Notice that such methods may not be first-order methods, since they may rely on the second derivative of the lower-level objective.

a minimization problem at the lower-level, i.e. $w(\lambda) = \text{argmin}_{w \in \mathbb{R}^d} \mathcal{L}(w, \lambda)$ with $\mathcal{L} : \mathbb{R}^d \times \Lambda \to \mathbb{R}$. However, when $\mathcal{L}$ is differentiable and strictly convex, since $w(\lambda)$ is the only point satisfying $\nabla_1 \mathcal{L}(w(\lambda), \lambda) = 0$, if we set the fixed-point map to be the one-step gradient descent update, i.e. $\Phi(w, \lambda) = w - \eta \nabla_1 \mathcal{L}(w, \lambda)$ for any step size $\eta > 0$, then the two problems are equivalent.

We focus on the fixed-point formulation because it has two main advantages. Firstly, it simplifies the analysis when the lower-level problem can be solved by the repeated application of the map Φ. Secondly, it allows to handle more naturally problems where the lower-level is commonly written as a fixed-point problem: e.g. in the case of equilibrium models (Bai et al., 2019).

Note also that to ensure that $\nabla f(\lambda)$ is well defined we implicitly assume that the solution of the lower-level problem is unique. When this is not the case, the optimistic (or pessimistic) bilevel reformulation is often used, where at the upper-level there is an additional minimization (maximization) over the solution set at the lower-level. Optimistic bilevel problems are usually transformed into value function, Karush-Kuhn-Tucker (KKT) or mathematical program with equilibrium constraints (MPEC) reformulations (Dempe and Zemkoho, 2020).

Throughout this manuscript, we will assume that E and Φ are differentiable and indicate with $\nabla_i E$ and $\partial_i \Phi$ the gradient and Jacobian with respect to the i-th component of E and Φ respectively. If $w(\lambda)$ is unique and $I - \partial_1 \Phi(w(\lambda), \lambda)$ is invertible, then, thanks to the implicit function theorem, both $w(\lambda)$ and $f(\lambda)$ are differentiable, and we have

$$
\begin{aligned}
w'(\lambda) &= (I - \partial_1 \Phi(w(\lambda), \lambda))^{-1} \partial_2 \Phi(w(\lambda), \lambda) \\
\nabla f(\lambda) &= \nabla_2 E(w(\lambda), \lambda) + w'(\lambda)^\top \nabla_1 E(w(\lambda), \lambda),
\end{aligned}
$$

where $\nabla f(\lambda)$ is called the *hypergradient* (see Lemma 3.3.2 in Chapter 3). Note that computing $\nabla f(\lambda)$ accurately can be difficult because $w(\lambda)$ is usually the solution of an optimization problem, and also evaluating, storing and inverting the Jacobians of Φ might be too costly especially when d and n are large. Moreover, in many machine learning applications, E and Φ are either sums over many examples or integrals over

the data distributions, which makes even their evaluation expensive. To study more efficient strategies in this case, we consider the stochastic setup where

$$E(w,\lambda) = \mathbb{E}[\hat{E}(w,\lambda,\xi)], \quad \Phi(w,\lambda) = \mathbb{E}[\hat{\Phi}(w,\lambda,\xi)],$$

with $\hat{E}\colon \mathbb{R}^d \times \Lambda \times \Xi \to \mathbb{R}$, $\hat{\Phi}\colon \mathbb{R}^d \times \Lambda \times Z \to \mathbb{R}^d$ being cheaper to evaluate than E and Φ, and ξ, ζ being two independent random variables with values in Ξ and Z, respectively.

For the reasons described above, computing the exact hypergradient is often not feasible, and gradient-based bilevel methods use an approximation which is usually done with one of the following two strategies.

- *Iterative Differentiation (ITD)* (Maclaurin et al., 2015; Franceschi et al., 2017), which works by differentiating the steps of the solver for the lower-level problem.

- *Approximate implicit differentiation (AID)* (Pedregosa, 2016), which works as follows. Firstly, it computes an approximate solution to the lower-level problem using a given solver. Secondly, it uses another solver to compute the approximate solution of a *linear system* arising from the implicit expression of the hypergradient.

Thanks to reverse mode AD, used to compute upper-level gradients and lower-level Jacobian or hessian vector products, AID and ITD can be implemented efficiently, because the complexity of approximating the hypergradient is of the same order as that of approximating the objective f. However, prior to Grazzi et al. (2020), there was little focus on establishing theoretical guarantees for these methods and for gradient-based bilevel methods in general.

Deterministic Iterative differentiation was previously studied by the automatic differentiation community (Griewank and Walther, 2008, Chapter 15), which proved asymptotic linear rates for the approximation error. Asymptotic results for the convergence of the minimizers of the approximate bilevel problem (i.e. where the $w(\lambda)$ is replaced by an approximate solution) where given for ITD (Franceschi et al.,

2018). Concerning AID instead, the asymptotic convergence of the hypergradient and of a deterministic bilevel procedure using approximate hypergradients was proved by Pedregosa (2016), while hypergradient error rates in the special case of meta-learning with biased regularization where given by Rajeswaran et al. (2019). The sample complexity for some gradient-based bilevel procedure based on AID where provided in (Couellan and Wang, 2016; Ghadimi and Wang, 2018) for specific deterministic and stochastic algorithms. Multiple research developments have occurred within our field of study since we began our work on this subject. We discuss some of them in detail in the related works section of Chapters 4 and 5.

1.2 Outline of the Thesis

This work studies how well AID and ITD approximate the hypergradient and establishes convergence guarantees for a whole bilevel optimization procedure relying on AID. In particular, under the assumptions that the functions involved are smooth and the fixed-point equation at the lower-level is a contraction, we present a theoretical analysis of gradient-based bilevel optimization techniques used in machine learning together with experimental evidence of their efficacy. The main focus will be on establishing rates of convergence, which provide a principled way of comparing different bilevel optimization algorithms.

1.2.1 Summary of Contributions

- We propose to study an alternative formulation of bilevel optimization where the minimization problem at the lower-level is replaced by a fixed-point equation. This formulation naturally covers applications like equilibrium models and some graph and recurrent neural networks and allows us to analyse more naturally ITD approaches.

- We provide iteration complexity results for two deterministic approaches to approximate the hypergradient: iterative differentiation (ITD) and approximate implicit differentiation (AID). Both methods converge linearly to the true hypergradient, although AID has a better error upper bound, $O(q^t)$ versus $O(tq^t)$ of ITD where q is the contraction constant and t is the number of

iteration of the lower-level and linear system solvers, and is usually faster also in practice.

- We introduce the Stochastic Implicit Differentiation (SID) method to compute the hypergradient. SID modifies AID to include stochastic estimators. We prove that the mean square error of SID goes to zero when both the mini-batch sizes used to estimate $\nabla_1 E$, $\partial_2 \Phi$ and the number of iterations of the solvers for the lower-level problem and linear system go to infinity, and provide $O(1/t)$ convergence rates when both solvers are simple stochastic fixed-point iterations.

- As a byproduct of the Analysis of SID, we prove rates of convergence for stochastic fixed-point iteration methods when the expected fixed-point map is a contraction. These results are rather straightforward extensions of the ones for the stochastic gradient method on strongly convex and Lipschitz smooth objectives reported in Bottou et al. (2018) and can be of interest even beyond bilevel optimization.

- Finally, we establish the sample complexity of a bilevel procedure using inexact projected gradient descent at the upper-level, with hypergradient computed via SID. In particular, we recover the optimal $O(\varepsilon^{-2})$ and near-optimal $O(\varepsilon^{-1}\log(\varepsilon^{-1}))$ sample complexity to reach an ε-stationary point in the stochastic and deterministic setting respectively. Contrary to previous work, this result is achieved without warm-starting the lower-level problem, i.e. without using previously computed lower-level solutions as starting points for future steps of the bilevel algorithm. This makes our approach more suited to bilevel problems where the lower-level can be decomposed into many subproblems, such as meta-learning and equilibrium models, as we also show empirically.

1.2.2 Scope

The primary focus of this work is on theoretical aspects of gradient-based bilevel optimization algorithms, while black-box and higher order methods are secondary

topics. Deep learning models are sometimes used for the empirical evaluation, but they are not the primary subject of research.

All theoretical results that we propose assume that the upper-level objective E and the fixed point map Φ in the bilevel formulation are smooth. We will not study the issues arising when this is not the case. Furthermore, to be able to derive complexity results on the hypergradient approximation, we assume that the map in the fixed-point equation at the lower-level is a contraction. This is a strong assumption which guarantees that the lower-level problem has only one solution and the existence of the hypergradient. The reader is referred to the works by Mairal et al. (2011); Ochs et al. (2015); Bertrand et al. (2020, 2022b), which present algorithms for hyperparameter optimization on non-smooth lower-level problems, and to Liu et al. (2020, 2022); Arbel and Mairal (2022) which tackle the case where the lower-level problem has multiple solutions.

The bilevel optimization point of view that we adopt does not take into account possible issues concerning generalization. For example in hyperparameter optimization, when the number of hyperparameters is large we can more easily overfit to the validation set, if this is too small. This is an important problem which is seldom addressed by the literature from a theoretical perspective. A notable exception is the work by Bao et al. (2021), which provides generalization bounds leveraging stability. We will partially address this issue through experiments measuring the performance on a hold-out test set without any theoretical claims.

Related Publications

- R. Grazzi, M. Pontil, and S. Salzo. Bilevel optimization with a lower-level contraction: Optimal sample complexity without warm-start. *Journal of Machine Learning Research*, 24(167):1–37, 2023.

- R. Grazzi, M. Pontil, and S. Salzo. Convergence properties of stochastic hypergradients. In *International Conference on Artificial Intelligence and Statistics*, pages 3826–3834. PMLR, 2021b.

- R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo. On the iteration complex-

ity of hypergradient computation. In *International Conference on Machine Learning*, pages 3748–3758. PMLR, 2020.

Other Works Done During the PhD

- R. Grazzi, A. Akhavan, J. I. T. Falk, L. Cella, and M. Pontil. Group meritocratic fairness in linear contextual bandits. *NeurIPS*, 2022.

- R. Grazzi, V. Flunkert, D. Salinas, T. Januschowski, M. Seeger, and C. Archambeau. Meta-forecasting by combining global deep representations with local adaptation. *arXiv preprint arXiv:2111.03418*, 2021a.

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

1.2.3 Structure

The remaining part of this thesis is structured as follows.

1. Chapter 2 presents the background material. Section 2.1 explains the notation we use, while Section 2.2 introduces foundational mathematical concepts for our analysis. In Section 2.3, we discuss scalable optimization methods including theoretical results used in our analysis. In Section 2.4, we describe the inner workings and time/space complexity of automatic differentiation. Section 2.5 presents several important bilevel applications which will be explored in following chapters.

2. In Chapter 3 we derive deterministic iteration complexity results for the hypergradient computed using approximate implicit differentiation and iterative differentiation methods. The content of this chapter is adapted from Grazzi et al. (2020).

3. In Chapter 4 we illustrate a principled stochastic method to approximate the hypergradient, namely SID, and prove iteration complexity results for such method. The content of this chapter is mainly adapted from Grazzi et al.

(2021b), but we include the improved algorithm and analysis from Grazzi et al. (2023).

4. In Chapter 5 we derive the sample complexity for a bilevel procedure using SID to compute the hypergradient. The content of this chapter is adapted from Grazzi et al. (2023).

5. In Chapter 6 we derive general conclusions and discuss potential future works.

Our theoretical results are in Chapters 3 to 5. Each one of these chapters also contains a section with experiments on several bilevel problems arising in machine learning.

Chapter 2

Background

2.1 Notation

We outline part of the notation used throughout the thesis. $\mathbb{R}_+$ and $\mathbb{R}_{++}$ are used to denote the non-negative (either positive or zero) and positive reals respectively. We denote by $\|\cdot\|$ either the Euclidean norm or the spectral norm (when applied to matrices). For a differentiable function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ we denote by $f'(x) \in \mathbb{R}^{m\times n}$ the derivative of f at x. When $m = 1$, we denote by $\nabla f\colon \mathbb{R}^n \to \mathbb{R}^n$ the gradient of f. For a real-valued function $g\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ we denote by $\nabla_1 g(x,y) \in \mathbb{R}^n$ and $\nabla_2 g(x,y) \in \mathbb{R}^m$ the partial derivatives w.r.t. the first and second variable respectively. We also denote by $\nabla_1^2 g(x,y) \in \mathbb{R}^{n\times n}$ and $\nabla_{12}^2 g(x,y) \in \mathbb{R}^{n\times m}$ the second derivative of g w.r.t. the first variable and the mixed second derivative of g w.r.t. the first and second variable. For a vector-valued function $h\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^k$ we denote, by $\partial_1 h(x,y) \in \mathbb{R}^{k\times n}$ and $\partial_2 h(x,y) \in \mathbb{R}^{k\times m}$ the partial Jacobians w.r.t. the first and second variable respectively at $(x,y) \in \mathbb{R}^m \times \mathbb{R}^n$. The transpose and the inverse of a given matrix A, is denoted by $A^\top$ and A^{-1} respectively. We use the shorthand notation $\partial h^\top v$ to denote the Jacobian-vector product $h'(x,y)^\top v$ for some x and y. For a random variable X we denote by $\mathbb{E}[X]$ and $\mathbb{V}[X]$ its expectation and variance respectively. Finally, given two random variables X and Y, the conditional variance of X given Y is $\mathbb{V}[X \mid Y] := \mathbb{E}[\|X - \mathbb{E}[X \mid Y]\|^2 \mid Y]$ where $\mathbb{E}[X \mid Y]$ is the conditional expectation of X given Y. We sometimes use $\mathrm{Lip}(h)$ to refer to the Lipschitz constant of the function h.

2.2 Foundational Mathematical Concepts

This section delves into the foundational concepts that form the basis of this research. In particular, we will focus on convexity, strong convexity, Lipschitz continuity, Lipschitz smoothness, and contraction within the scope of functions from $\mathcal{X} \subseteq \mathbb{R}^n$ to $\mathbb{R}^d$ where $\mathcal{X}$ is convex. These definitions are standard in the optimization literature, see e.g. (Boyd and Vandenberghe, 2004; Ryu and Boyd, 2016), and are used to define specific classes of problems and to obtain the rate of convergence of algorithms.

Definition 2.2.1 (Convexity)**.** *A function $f : \mathcal{X} \to \mathbb{R}$ is convex if for all $x, y \in \mathcal{X}$, and for any $t \in [0,1]$, we have:*

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

if f is also differentiable in an open set containing $\mathcal{X}$, we can equivalently write

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x).$$

Definition 2.2.2 (Strong convexity)**.** *A function $f : \mathcal{X} \to \mathbb{R}$ is said to be μ-strongly convex if it satisfies the inequality:*

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y) - \frac{t(1-t)\mu}{2}\|y - x\|^2,$$

with $\mu > 0$, for all $x, y \in \mathcal{X}$ and for any $t \in [0,1]$. If f is also differentiable in an open set containing $\mathcal{X}$, we can equivalently write

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2}\|x - y\|^2.$$

Definition 2.2.3 (Lipschitz continuity)**.** *A vector-valued function $g : \mathcal{X} \to \mathbb{R}^d$ is said to be Lipschitz continuous if there exists a constant $L > 0$ such that, for all $x, y \in \mathcal{X}$,*

$$\|g(x) - g(y)\| \leq L\|x - y\|.$$

If g is also differentiable in an open set containing $\mathcal{X}$, we can equivalently write

$$\|g'(x)\| < L.$$

Definition 2.2.4 (Lipschitz smoothness)**.** *A function $g : \mathcal{X} \to \mathbb{R}^d$ is said to be Lipschitz smooth if g is differentiable in an open set containing $\mathcal{X}$ and its derivative g' is Lipschitz continuous.*

Definition 2.2.5 (Contraction)**.** *If a function $g : \mathcal{X} \to \mathcal{X}$ is Lipschitz continuous with constant $q < 1$, then g is a contraction.*

A direct application of the Banach fixed-point theorem (Banach, 1922) shows that if g is a contraction with constant q, then it has a unique fixed point $x^* \in \mathcal{X}$, i.e. such that $g(x^*) = x^*$, and that starting from any $x_0 \in \mathcal{X}$, the sequence $(x_i)_{i\in\mathbb{N}}$ where $x_i = g(x_{i-1})$ converges linearly to x^*, i.e. $\|x^* - x_k\| \leq q^k \|x^* - x_0\|$.

2.3 Scalable Optimization Methods

Mathematical Optimization studies methods to select the best element from a set according to some criterion. Optimization problems arise in all quantitative disciplines from computer science to economics. Moreover, the problem of learning from data is essentially an optimization problem. In this chapter, we consider the minimization problem

$$\min_{w\in\mathcal{W}\subseteq\mathbb{R}^d} f(w), \tag{2.1}$$

where $f : \mathcal{W} \mapsto \mathbb{R}$ is called the *objective function*.

In the bilevel optimization framework subject of this thesis, the upper-level problem is formulated as a minimization problem similar to (2.1). In contrast, the lower-level problem is that of finding the fixed-point of a vector valued map. Despite being seemingly different, these two formulations are related, since the minima of a smooth and/or convex function f are also fixed point of a certain operator related to the subgradient of f. This perspective allows to unify the analysis of several optimization methods (Ryu and Boyd, 2016).

Parametric Supervised Learning. The goal of supervised learning is to minimize the *expected risk*

$$\mathbb{E}_{(x,y)\sim\rho_{x,y}}[\mathcal{E}(h_w(x),y)]$$

where (x,y) are random variables representing the data and following the distribution $\rho_{x,y}$, where $x \in \mathcal{X} \subseteq \mathbb{R}^m$ is the *input* while $y \in \mathcal{Y}$ is the *target (or label)*, $h_w : \mathcal{X} \mapsto \mathcal{Y}$ is the machine learning model, also called hypothesis, in this case parameterized by $w \in \mathcal{W}$ and $\mathcal{E} : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ is the error measure for the task, measuring the error between predictions and targets (e.g. the square loss $\mathcal{E}(\hat{y},y) := (\hat{y}-y)^2$ for regression problems or the zero-one loss $\mathbf{1}\{\hat{y} = y\}$ for classification).

Since the expected risk cannot be directly computed due to the inaccessibility of $\rho_{x,y}$, supervised learning algorithms usually minimize the *(regularized) empirical risk*

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(h_w(x_i),y_i) + \beta\mathcal{R}(w), \tag{2.2}$$

where $((x_i,y_i))_{i=1}^n$ are the training examples sampled i.i.d. from $\rho_{x,y}$, $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ is the loss function, which is a usually continuous and convex on the first argument and acts as a surrogate for $\mathcal{E}$ (e.g. if $\mathcal{E}$ is the zero-one loss, $\mathcal{L}$ can be the cross-entropy loss), and $\mathcal{R} : \mathcal{W} \mapsto \mathcal{W}$ is a regularizer generally used to prevent *overfitting* to the training data (low empirical risk but high expected risk) and to encode prior knowledge about the problem (e.g. the L_p regularizer $\mathcal{R}(w) = \|w\|_p^p$). The regularizer is multiplied by the hyperparameter $\beta \in \mathbb{R}_+$ ($\beta = 0$ means no regularization is used).

The parameter space $\mathcal{W}$ is generally convex and closed and has a simple structure: either $\mathcal{W} = \mathbb{R}^d$, or is defined by simple constraints (e.g. box or ball), which allow for a relatively cheap projection of points in $\mathbb{R}^d$ onto $\mathcal{W}$. Furthermore, $\mathcal{L}(\cdot,y_i)$ and $\mathcal{R}$ are usually convex and differentiable almost everywhere. The majority of the complexity usually resides on the model h_w, which is usually differentiable almost everywhere, but can vary from a linear model $h_w(x) = w^\top x$ to deep neural networks with a hundred billion weights, as in large language models (Brown et al., 2020; Chowdhery et al., 2022). For some machine learning problems, the input

space $\mathcal{X}$ can be high-dimensional, in the order of thousands or millions (e.g. in the case of images it is the number of pixels times the number of color channels) and the number of training examples n can also range from the hundreds to the billions.

In the following, we primarily focus on optimization methods that are scalable, meaning that they can be applied when d and n are large, which is common in large-scale machine-learning applications. First-order methods, and in particular stochastic gradient descent and its several variants, have been proven very effective to tackle these problems.

2.3.1 Stochastic Gradient Descent

Let's assume that, in Problem (2.1), $\mathcal{W} = \mathbb{R}^d$ and the objective function f is smooth (in general f is smooth almost everywhere). *Stochastic gradient descent* (SGD) is an iterative procedure which starts from $w_1 \in \mathbb{R}^d$ and at each iteration $t \in \mathbb{N}$ computes

$$w_{t+1} = w_t - \eta_t \nabla_1 \hat{f}(w_t, \xi_t), \tag{2.3}$$

where $\{\eta_t\}$ is a sequence of step-sizes (also called learning rates), $\hat{f} : \mathcal{W} \times \Xi \mapsto \mathbb{R}$ is differentiable w.r.t. the first variable, $\{\xi_t\}$ is a sequence of i.i.d. random variables with values in Ξ and $\mathbb{E}[\nabla_1 \hat{f}(w, \xi_t)] = \nabla f(w)$.

In the case of the supervised learning objective in (2.2) a typical example is $\hat{f}(w,i)$ being either the regularized loss on the i-th training example, i.e. $\hat{f}(w,i) = \mathcal{L}(h_w(x_i), y_i) + \beta\mathcal{R}(w)$ with ξ_t uniform random on $\{1,\dots,n\}$, or on a small group of examples, called a mini-batch. When n is large, computing $\nabla_1 \hat{f}$ can be much cheaper than computing the full gradient ∇f while still yielding a descent direction in expectation. We note that SGD can also be used to minimize the expected risk if we set $\hat{f}(w,(x,y)) = \mathcal{L}(h_w(x), y)$ with $(x,y) \sim \rho_{x,y}$. Gradient Descent (GD) is obtained by setting $\hat{f} = f$.

In the following two theorems, we state some convergence results for SGD that hold in expectation under some mild assumption on the variance of $\nabla_1 \hat{f}$. These results are special cases of (Bottou et al., 2018, Theorems 4.6 to 4.10) with unbiased stochastic gradients (i.e. by setting $\mu = \mu_G = 1$ in the cited work).

Theorem 2.3.1 (SGD asymptotic convergenece)**.** *Let f be L-smooth and bounded from below, i.e. there exists $f^* \in \mathbb{R}$ such that $f^* = \min_{w\in\mathbb{R}^d} f(w)$. Let also $\mathbb{E}[\nabla_1 \hat{f}(w,\xi_t)] = \nabla f(w)$ and $\mathbb{V}[\nabla_1 \hat{f}(w,\xi_t)] \leq \sigma_1 + \sigma_2\|\nabla f(w)\|^2$ for every $t \in \mathbb{N}$. If SGD in* (2.3) *is applied with decreasing step-sizes such that*

$$\sum_{t=1}^{\infty} \eta_t = \infty, \qquad \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

Then the expected weighted average of squared gradient norms satisfies

$$\lim_{T\to\infty} \mathbb{E}\left[\frac{\sum_{t=1}^{T} \eta_t \|\nabla f(w_t)\|^2}{\sum_{t=1}^{\infty} \eta_t}\right] = 0,$$

and hence $\liminf_{t\to\infty} \mathbb{E}[\|\nabla f(w_t)\|^2] = 0$.

Theorem 2.3.2 (SGD Rates)**.** *Let f be L-smooth and bounded from below. Let also $\mathbb{E}[\nabla_1 \hat{f}(w,\xi_t)] = \nabla f(w)$ and $\mathbb{V}[\nabla_1 \hat{f}(w,\xi_t)] \leq \sigma_1 + \sigma_2\|\nabla f(w)\|^2$ for every $t \in \mathbb{N}$. The following statements hold.*

(i) If f is τ-strongly convex and SGD in (2.3) *is applied with constant step-size $\eta_t = \eta$ such that $0 < \eta \leq \frac{1}{L(\sigma_2+1)}$. Then the expected optimality gap satisfies*

$$\mathbb{E}[f(w_t) - f^*] \leq \frac{\sigma_1 L\eta}{2\tau} + (1-\eta\tau)^{t-1}(f(w_1) - f^*).$$

(ii) If SGD in (2.3) *is applied with constant step-size $\eta_t = \eta$ such that, as in the previous case $0 < \eta \leq \frac{1}{L(\sigma_2+1)}$. Then the expected average of squared gradient norms satisfies*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \|\nabla f(w_t)\|^2\right] \leq \sigma_1 L\eta + \frac{2(f(w_1) - f^*)}{T\eta}.$$

(iii) If f is τ-strongly convex and SGD in (2.3) *is applied with decreasing step-size*

$$\eta_t = \frac{\alpha}{\gamma + t}, \quad \text{for some} \quad \beta > \frac{1}{\tau} \quad \text{and} \quad \gamma > 0 \quad \text{such that} \quad \eta_1 < \frac{1}{L(\sigma_2+1)}$$

then the expected optimality gap satisfies

$$\mathbb{E}[f(w_t) - f^*] \leq \max\left\{\frac{\sigma_1 L\beta^2}{2(\beta\tau - 1)}, (\gamma+1)(f(w_1) - f^*)\right\} \Big/ (\gamma + t).$$

To summarize the above results, Theorem 2.3.1 states that for square summable but not summable stepsizes, SGD reaches a stationary point in expectation. Instead, Theorem 2.3.2 establishes the rates $O(exp(-t)+c)$ (with constant stepsize on strongly convex objectives), $O(1/t+c)$ (with constant stepsize on possibly non-convex objectives) and $O(1/t)$ (with decreasing stepsize on strongly convex objectives), where c is a problem and step-size specific constant which is zero when the variance parameter σ_1 is equal to zero (this implies that $c = 0$ for GD). $O(1/\sqrt{t})$ non-convex rates are proved when $\sigma_2 = 0$ in Ghadimi and Lan (2013) for randomized SGD, i.e. when final parameters are chosen at random among the iterates $\{w_t\}$ with probability which depends on the step-sizes, which are set as $\eta_t = \Theta(1/\sqrt{t})$.

Non-convex objectives and neural networks. Without the convexity assumption on f, the strongest SGD guarantees in the theorems above concern only the expected average gradient norm, while when f is strongly convex, we have rates on the expected optimality gap in function values, i.e. $\mathbb{E}[f(w_t) - f^*]$. These rates transfer directly to the expected squared norm of the optimality gap on the iterates $\mathbb{E}[\|w_t - w^*\|^2]$, since if f is τ-strongly-convex and $w^* = \text{argmin}_{w\in\mathbb{R}^d} f(w)$, we have $f(w_t) - f^* \geq \frac{\tau}{2}\|w_t - w^*\|^2$. When training a neural network, the objective function is non-convex and the optimization could get "trapped" in a saddle-point or a bad local-minima. However, stochastic gradient descent can provably escape certain types of saddle points (Ge et al., 2015), and even gradient descent can do so with a proper random initialization (Lee et al., 2016). Furthermore, for over-parameterized models, i.e. where $n < d$, (stochastic) gradient descent on some common deep neural architectures converges polynomially to a global minimizer with zero training error (Du et al., 2019; Allen-Zhu et al., 2019).

Constant vs decreasing step-sizes. From Theorem 2.3.2, we see that SGD with decreasing step-sizes converges properly, i.e. either the optimality gap (if f is

strongly-convex) or the expected norm of gradients (if f is not convex) are zero in the limit when the number of iterations goes to infinity. In contrast, SGD with constant step-sizes converges faster, but only up to a region which is proportional to the product of the step-size η, the Lipschitz constant L and a component of the variance of the stochastic gradients σ_1. A good strategy suggested by Bottou et al. (2018) would then be to start with a constant step-size until such region is reached and then proceed with decreasing step-sizes. However, for machine learning problems, going beyond a given precision might not bring any significant advantage or even be detrimental. This is because, as we previously showed, the objective f is a proxy for the expected risk and minimizing it accurately can cause overfitting. Indeed, limiting the precision of the optimization, by e.g. limiting the number of SGD iterations, is considered as an implicit form of regularization because it prevents overfitting.

Gradient descent and projection. We can recover the rates for gradient descent by setting $\sigma_1 = \sigma_2 = 0$ in Theorem 2.3.2. When $\sigma_1 = 0$, decreasing step-sizes are no longer necessary to achieve proper convergence, and we recover the $O(1/t)$ and linear rates for non-convex and strongly-convex objectives respectively. Furthermore, when $\mathcal{W} \neq \mathbb{R}^d$ is convex and closed, convergence guarantees similar to the case $\mathcal{W} = \mathbb{R}^d$ can also be obtained for the projected stochastic gradient descent method, which projects the iterates onto $\mathcal{W}$ after the stochastic gradient update.

Bias and bilevel optimization. The results in Theorems 2.3.1 and 2.3.2 can be extended to biased stochastic gradients where $\mathbb{E}[\nabla_1 \hat{f}(w_t, \xi_t)]$ is a direction of sufficient descent with norm comparable to that of the gradient (Bottou et al., 2018). However, we cannot apply this extension to bilevel optimization because, despite the stochastic upper-level gradients being biased, they are usually not descent directions in expectation. Therefore, to obtain convergence guarantees for gradient-based bilevel methods, the bias has to vanish when the number of upper-level iterations goes to infinity.

2.3.2 Accelerated Methods

Accelerated methods are gradient-based methods that also make use of previously computed gradients in their update rule.

Stochastic Gradient Descent with Momentum (SGDM) uses the updates

$$v_{t+1} = \gamma_t v_t + \nabla_1 \hat{f}(w_t, \xi_t),$$
$$w_{t+1} = w_t - \eta_t v_{t+1},$$

where $w_1 \in \mathbb{R}^d$, $v_1 = 0$. v_t contains the running average of past gradients and $\{\eta_t\}, \{\gamma_t\}$ are the sequence of step-sizes and momentum factors respectively. The deterministic variant (GDM) is obtained by setting $\nabla_1 \hat{f}(w_t, \xi_t) = \nabla f(w_t)$

On L-smooth and τ-strongly convex quadratic objectives GDM with fixed step-size and momentum factor, also known as the heavy-ball method, converges faster than gradient descent: the optimality gap at iteration t is $O(\exp(-t/\sqrt{\kappa}))$ (Polyak, 1964) against $O(\exp(-t/\kappa))$ for GD where $\kappa := L/\tau$ is called the condition number. Furthermore, the GDM variant proposed by Nesterov (1983) is shown to achieve the optimal rate of $O(1/t^2)$ for convex objectives, improving the $O(1/t)$ rate of GD. However, the stochastic variant SGDM does not improve the theoretical rates despite still outperforming SGD in some applications (Sutskever et al., 2013).

ADAM (Kingma and Ba, 2015) is arguably the most popular optimization method for deep learning after SGD. It has the following update rule.

$$v_{t+1} = \gamma_1 v_{t-1} + (1-\gamma_1)\nabla_1 \hat{f}_t(w_t, \xi_t)$$
$$m_{t+1} = \gamma_2 m_{t-1} + (1-\gamma_2)\nabla_1 \hat{f}_t(w_t, \xi_t)^2$$
$$w_{t+1} = w_t - \eta \frac{\sqrt{1-\gamma_2^t}}{1-\gamma_1^t} \frac{v_t}{\sqrt{m_t + \varepsilon}},$$

where squares and divisions for vectors are element-wise, $\gamma_1, \gamma_2 \in [0,1)$ control the exponential decay of the moving averages of gradients (v_t) and squared gradients (m_t), and η is the learning rate. The recommended defaults $\gamma_1 = 0.9$, $\gamma_2 = 0.999$ and $\eta = 10^{-3}$ perform quite well for several deep learning problems, which contributed to the success of the method. The method uses a combination of momentum and element-wise learning rates (given by the element-wise division by $\sqrt{m_t + \varepsilon}$) which help in situations when the ranges of gradients vary greatly across different parameters.

Despite its popularity, ADAM does not converge in general even on some convex objectives (Reddi et al., 2019).

2.3.3 Second-Order Methods

While stochastic gradient descent and its variants like SGDM and ADAM use only first-order information, second-order methods try to speed up the convergence by leveraging second-order derivatives, which are helpful especially when the Hessian of f is ill-conditioned. The most prominent example is the (deterministic) *Newton method*

$$w_{t+1} = w_t - H_t^{-1} \nabla f(w_t),$$

where f is twice differentiable and $H_t = \nabla^2 f(w_t)$ is its Hessian matrix. On quadratic strictly convex objectives of the form $w^\top A w + b^\top w + c$ with $x^\top A x > 0$ for every $x \in \mathbb{R}^d$, it is easy to see that this method converges in one iteration from any starting point $w_1 \in \mathbb{R}^d$. More generally, the Newton method can exhibit a local quadratic rate of convergence on strongly convex objectives (Nesterov, 2003). However, convergence of the Newton method (and many second-order methods) has usually stronger requirements than that of first-order methods. For example, it requires that the Hessian can always be inverted and, to achieve global convergence, to be coupled with line search or trust-region methods (Nocedal and Wright, 1999, Chapter 3 and 4), which are often too costly for large scale applications and not directly applicable to the stochastic setting. Furthermore, the cost of each Newton iteration is dominated by that of computing $H_t^{-1} \nabla f(w_t)$, which is equivalent to that of solving the linear system $\nabla^2 f(w_t) x = \nabla f(w_t)$. This can be solved without fully inverting the Hessian either through matrix factorization methods (for small d), or iterative methods which do not even require to compute and store the entire Hessian but only Hessian-vector products. Computing such products is usually much more efficient than computing the full hessian matrix and can also be done automatically and efficiently thanks to automatic differentiation. One of the commonly used iterative methods is conjugate gradient, which converges exactly after d iterations (and has fast rates even before that) when the Hessian is positive definite. Despite this, the cost

of computing a full Newton step is usually not feasible for large scale applications when d is large (it is usually $O(d^2)$ versus $O(d)$ for first-order methods). Therefore, several approximation have been proposed to reduce the cost per iteration and make second-order methods competitive with first-order approaches.

Hessian-free, quasi and stochastic Newton. To alleviate the cost of the Newton update, Hessian-free Newton methods solve the linear system associated with the Newton step approximately by stopping the iterative solver before convergence. Despite the approximation, these methods still have local superlinear convergence when the linear system is solved with increasing accuracy (Dembo et al., 1982). Alternatively, quasi-Newton methods like BFGS and L-BFGS (Fletcher, 2013) approximate the Hessian inverse using past gradients. BFGS has local superlinear convergence but high memory cost, while L-BFGS reduces the memory cost but is only provably linearly convergent. When the exact gradient or Hessian-vector product is too expensive to compute, stochastic algorithms, which replace these quantities with random estimators, have also been proposed (Martens and Grosse, 2015; Agarwal et al., 2017). However, stochastic second-order methods only improve the constants in the sublinear convergence rates w.r.t. stochastic first-order methods.

Second-order methods in deep learning. Second-order methods are seldom used to optimize deep neural networks. Despite some promising early proposals outperforming gradient descent on very deep (at the time) networks (Martens, 2010; Martens and Sutskever, 2011), SGD variants like ADAM combined with specific components in network architectures such as batch normalization (Ioffe and Szegedy, 2015) and residual connections (He et al., 2016), which simplify the optimization problem, have now become standard approaches capable of training very deep networks on huge datasets. Designing scalable second-order methods for deep learning is still an active area of research with some recent proposals outperforming first-order methods in some tasks (Yao et al., 2021; Frantar et al., 2021).

In this work we focus primarily on first-order methods, since they are currently the standard methods used to tackle large scale machine learning problems.

2.4 Automatic Differentiation

Automatic Differentiation (AD) (Griewank and Walther, 2008) is a technique to augment a given function evaluation program, i.e. a program whose inputs and outputs are numerical values and that evaluates a given mathematical function, with the ability of computing derivatives. It relies on the fact that good function evaluation programs are essentially clever compositions of simple operations for which exact derivatives are well known and that can be evaluated efficiently. As we will see, symbolic derivatives of some programs, obtained by applying the chain rule to the symbolic expression of the function computed by the program, can be complicated, uninformative and inefficient to evaluate without implementations that avoid re-computing the same quantities. AD instead takes advantage of the implementation of the function evaluation program: it also uses the chain rule, but applied directly to numerical values and following the program's order of execution. For this reason, evaluating derivatives, or more precisely Jacobian-vector products, with AD has a cost comparable to that of evaluating the original program.

In this section, we will present "exact" AD, i.e. assuming that the evaluation program works with infinite precision the corresponding derivative program will output the exact derivative. The main subject of this thesis and of (Griewank and Walther, 2008, Chap. 15) is instead that of computing approximate AD derivatives, where the evaluation program contains some approximation method used to compute certain implicit quantities. As we will show in the following chapters, in such case exact AD is still used to compute intermediate values.

To better motivate AD, we now compare it with two other techniques commonly used to compute derivatives.

Finite differences. A common technique to approximate the derivative of a function $f : \mathcal{X} \subseteq \mathbb{R}^d \to \mathbb{R}^m$ at a point x in the direction h is to use

$$\frac{f(x+\varepsilon h) - f(x)}{\varepsilon} \qquad \text{or} \qquad \frac{f(x+\varepsilon h) - f(x-\varepsilon h)}{2\varepsilon},$$

with $\varepsilon > 0$. These expressions however, might not provide an accurate derivative

estimate: when ε is too small the cancellation error due to finite precision arithmetic might reduce the number of significant digits of the estimate, but when ε is too large there might be truncation errors as we are further away from the derivative definition in which $\varepsilon \to 0$. Even when ε is optimal, these issues can be problematic in practice and become even more pronounced for higher-order derivatives. Furthermore, to approximate the full Jacobian of f, we would need to approximate d directional derivatives for a total cost of at least d function evaluations. In contrast, AD does not require to set a parameter, is not afflicted by truncation errors, and reverse mode AD on functions with scalar output has usually a cost never greater than that of 5 function evaluations for any d.

Symbolic differentiation, similarly to differentiation by hand, applies the chain rule to the symbolic expression of a given function. A simple problematic example where this approach might not be ideal is

$$f(x) = \prod_{i=1}^{d} x_i = x_1 x_2 \cdots x_d, \qquad \nabla f(x)_i = \prod_{j \neq i} x_j = x_1 \cdots x_{i-1} x_{i+1} \cdots x_d$$

where $\nabla f(x)_i$ is the i-th component of the gradient of f. The symbolic expression for the full $\nabla f(x)$ is quite cumbersome and contains several repeated sub-expressions that can result in wasted computation if some intermediate values are not stored and reused during the evaluation. AD instead does not differentiate the symbolic expression of f but its program, which is assumed to provide and efficient implementation for the evaluation of f. In this example, if the program evaluating f computes the values of the partial products $x_1 \cdots x_k$ internally, these are saved by the AD augmented program and then used to evaluate the derivative.

2.4.1 Function Evaluation Program

Let

$$f : \mathcal{X} \subseteq \mathbb{R}^d \to \mathbb{R}^m, \qquad Df : \mathcal{X} \subseteq \mathbb{R}^d \mapsto \mathbb{R}^m \times \mathbb{R}^d,$$

be respectively the differentiable mathematical function we wish to differentiate and its first-order derivative, i.e. its Jacobian. To perform symbolic differentiation,

we need the analytic expression of f, while for finite differences f is treated as a black box, and we only need to be able to evaluate f at any given point. In contrast, automatic differentiation requires the knowledge of a program which evaluates f at any point $x \in \mathcal{X}$. Therefore, we will assume that $f(x)$ can be written as

$$f(x) = g_l \circ g_{l-1} \circ \cdots \circ g_1(x), \tag{2.4}$$

where $w_0 = x$ and for any $i \in \{1, \dots l-1\}$

$$g_i(w_{i-1}) = \begin{pmatrix} w_{i-1} \\ \psi_i(w_{i-1}) \end{pmatrix}, \quad g_l(w_{l-1}) = \psi_l(w_{l-1})$$

and ψ_i are simple possibly vector-valued differentiable operations (such as element-wise addition, multiplication, reciprocal or affine transformations) called *elementals*, belonging to a predefined library Ψ. The RHS of eq. (2.4) represents a computer program evaluating f, where we use the convention that each operation g_i is executed sequentially from right to left and the partial computation is saved in $l-1$ auxiliary vector variables

$$v_i = \psi_i\big((v_j)_{j=0}^{i-1}\big), \quad \text{where} \quad (v_j)_{j=0}^{i-1} = \text{concat}\big((v_j)_{j=0}^{i-1}\big), \quad v_0 = x,$$

and we overloaded the notation $(v_j)_{j=0}^{i-1}$ to also be the concatenation of the vectors $v_0, \dots, v_{i-1}$ when needed. In practice, ψ_i usually depends on just one or two variables. To capture this aspect, we can also frame the evaluation of f at x as a directed acyclic graph where nodes are the auxiliary vector variables $\{v_i\}_{i=0}^{l}$, and edges represent direct dependencies between nodes: there is an edge from v_j to v_i, denoted with $j \prec i$, when v_j has a direct dependence on v_i. For instance, an indirect dependence would be when the output variable v_l depends on the input variable x but only through the intermediate variable v_{l-1}. With this notation we can write the expression for v_i more compactly by excluding unused variables as follows.

$$v_i = \psi_i(u_i), \quad \text{where} \quad u_i = (v_j)_{j \prec i} = \text{concat}\big((v_j)_{j \prec i}\big)$$

For example, for the product function $f(x) = \prod_{i=1}^{d} x_i$ we could set v_i to be equal to the partial product $\prod_{j=1}^{i+1} x_j$, i.e. set $v_0 = x$ and for every $i \in \{2, \dots d-1\}$[1]

$$v_1 = \psi_1(v_0) = x_1 * x_2, \quad v_i = \psi_i(v_{i-1}, v_0) = v_{i-1} * x_{i+1},$$

which yields the following computational graph, where we set $d = 4$.

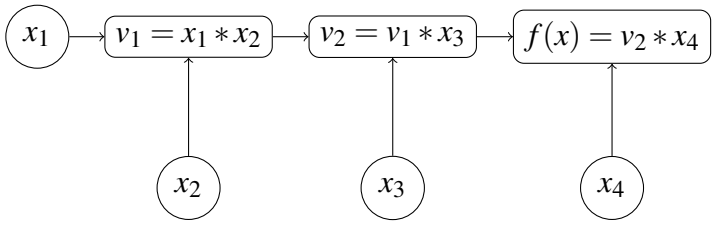

Figure 2.1: A possible computational graph for $f(x) = \prod_{i=1}^{4} x_i$

For every elemental $\psi \in \Psi$ AD uses a program that computes $\psi(u)$, and another for the Jacobian-vector products $D\psi(u)\,\dot{u}$ (for forward mode AD) and $D\psi(u)^\top \bar{u}$ (for reverse mode AD), where $u, \dot{u}, \bar{u}$ are vectors of appropriate dimensions. A minimal set of elementals that can approximate a large variety of functions is the so-called *polynomial core*, which contains only addition, multiplication, unary sign switch and constant assignment. However, a richer Ψ can usually speed up the computation. For example the library of elementals in the deep learning framework PyTorch (Paszke et al., 2019b) contains several vector and matrix operations whose programs for function evaluation and (reverse) Jacobian-vector products have fast C++ implementations which may take advantage of GPU parallelism. Furthermore, if we want our program to contain branches, the resulting function will in general not be differentiable, but AD can still be applied to recover some generalized derivatives.

For a given library Ψ of elementals, more than one program might correspond to a given function f and some operations g_i in eq. (2.4) might also be carried out in parallel if the order of application does not change the final result. In general, AD assumes that the decomposition we use to evaluate f gives us an efficient program and exploits such decomposition to compute derivatives.

[1]For a simple exposition and since since v_0 is a vector, we reduce the number of variables by setting ψ_i as a composition of two other elemental operations: coordinate selection (of the input x) and binary product.

2.4.2 Forward and Reverse Mode Automatic Differentiation

Forward and Reverse mode AD provide an efficient way to compute the Jacobian-vector products

$$Df(x)\,\dot{x} \in \mathbb{R}^m \quad \text{(forward)}, \qquad Df(x)^\top \bar{y} \in \mathbb{R}^d \quad \text{(reverse)},$$

for any $x \in \mathcal{X}$, $\dot{x} \in \mathbb{R}^d$ and $\bar{y} \in \mathbb{R}^m$. By applying the chain rule to eq. (2.4) we obtain

$$Df(x)\,\dot{x} = Dg_l(w_{l-1})\,Dg_{l-1}(w_{l-2}) \cdots Dg_1(w_0)\,\dot{x}, \tag{2.5}$$

$$Df(x)^\top \bar{y} = Dg_1(w_0)^\top Dg_2(w_1)^\top \cdots Dg_l(w_{l-1})^\top \bar{y}, \tag{2.6}$$

where for $i \in \{1,\ldots,l-1\}$

$$Dg_i(w_{i-1}) = \begin{pmatrix} I \\ D\psi_i(w_{l-1}) \end{pmatrix} \quad \text{and} \quad Dg_l(w_{l-1}) = D\psi_l(w_{l-1}).$$

Forward and Reverse AD can be thought of as computing the products in eq. (2.5) and eq. (2.6) respectively, from right to left, so that each computation step reduces to *one* (forward or reverse) Jacobian-vector product involving one elemental. A common implementation is outlined in Algorithms 2.4.1 and 2.4.2.

Algorithm 2.4.1 Forward Mode AD

Input: $x, \dot{x} \in \mathbb{R}^d$
Output: $f(x), Df(x)\,\dot{x}$
1: $v_0 = x$, $\dot{v}_0 = \dot{x}$
2: **for** $i = 1,\ldots,l$ **do**
3: $\quad v_i = \psi_i(u_i)$, $\dot{v}_i = D\psi_i(u_i)\,\dot{u}_i$
4: **return** v_l, $\dot{v}_l$

Algorithm 2.4.2 Reverse Mode AD

Input: $x \in \mathbb{R}^d$, $\bar{y} \in \mathbb{R}^m$
Output: $f(x), Df(x)^\top \bar{y}$
1: $v_0 = x$, $\bar{v}_l = \bar{y}$, $\bar{v}_i = 0$ for $i = 1,\ldots,l-1$
2: **for** $i = 1,\ldots,l$ **do**
3: $\quad v_i = \psi_i(u_i)$
4: **for** $i = l,\ldots,1$ **do**
5: $\quad \bar{u}_i = \bar{u}_i + D\psi_i(u_i)^\top \bar{v}_i$
6: **return** v_l, $\bar{v}_1$

Figure 2.2: Forward and Reverse Mode AD. We set $u_i = (v_j)_{j \prec i}$, $\dot{u}_i = (\dot{v}_j)_{j \prec i}$, $\bar{u}_i = (\bar{v}_j)_{j \prec i}$. $\{\dot{v}_i\}$, $\{\bar{v}_i\}$ are called the *dotted* and *adjoint* variables respectively. Computations on the same line and, depending on the structure of the evaluation graph of $f(x)$, also some iterations in the for cycles can be performed in parallel.

As one can predict from eq. (2.5), the auxiliary dotted variables $\{\dot{v}_i\}$ in forward mode AD are computed together with the intermediate variables $\{v_i\}$ in one so-called *forward pass* (going forward from inputs x to outputs $f(x)$), which also evaluates $f(x)$. On the other hand, from eq. (2.6) we observe that reverse mode AD requires first to compute and save all the intermediate $\{v_i\}$ during the *forward pass* and then proceed, in the so-called *reverse or backward pass* (going backwards from output to inputs), to compute the auxiliary adjoint variables $\{\overline{v}_i\}$ by plugging-in the saved $\{v_i\}$ in the (reverse) Jacobian-vector products. Hence, reverse mode AD has usually a greater memory cost than forward mode AD.

Expanding the computation of dotted and adjoint variables in Algorithms 2.4.1 and 2.4.2, and letting $\partial_{v_j}\psi_i(u_i)$ be the partial derivative of $\psi_i(u_i)$ w.r.t. v_j, we have

$$\dot{v}_i = \sum_{j \prec i} \partial_{v_j}\psi_i(u_i)\,\dot{v}_j, \qquad \overline{v}_j = \overline{v}_j + \partial_{v_j}\psi_i(u_i)^\top \overline{v}_i \quad \forall j \prec i.$$

This shows that dotted variables $\{\dot{v}_i\}$ are set in one step, while adjoint variables $\{\overline{v}_i\}$ are set incrementally during the reverse pass using backward pointers from each intermediate node v_i to its parents $\{v_j\}_{j \prec i}$. Adjoint variables can also be set in one step by using instead forward pointers from each node v_i to its direct descendants $\{v_j\}_{i \prec j}$. This is done by replacing the incremental update in line 5 of Algorithm 2.4.2 with $\overline{v}_i = \sum_{i \prec j} \partial_{v_j}\psi_i(u_i)^\top \overline{v}_i$. However, since forward pointers are less natural to implement, they are rarely used in AD frameworks.

Checkpointing. Forward and reverse AD can also be applied recursively if each elemental $\psi \in \Psi$ is seen as a composition of sub-elementals and so on. This is useful to save memory in reverse mode AD in exchange for an increased computation time. Indeed, a strategy to save memory called *checkpointing* is that of grouping some operations to be treated as higher level elementals. This hierarchical structure yields less auxiliary variables $\{v_i\}$ to be saved in the forward pass at each level, but increases the time complexity of the backward pass, which needs to re-compute the forward pass of the higher level elementals.

Hessian-vector products. Some optimization methods, including bilevel ones,

may require computing Hessian-vector products $\nabla^2 f(x)v$ where $f : \mathbb{R}^d \mapsto \mathbb{R}$ and $v \in \mathbb{R}^d$ is a given vector. By applying reverse mode AD twice we can see that the cost of a Hessian-vector product is of the same order as that of computing the function value. However, this can be done more efficiently by first computing $\nabla f(x)$ using reverse mode AD and then computing $\partial_x(x \mapsto \nabla f(x))\, v = \nabla^2 f(x)\, v$ with forward mode. This *forward-over-reverse* approach (Pearlmutter, 1994) requires less memory than a double application of reverse mode AD. The same result can be achieved also by applying forward mode AD first to compute $\nabla f(x)\, v$ and then reverse AD to get $\partial_x(x \mapsto \nabla f(x)\, v) = \nabla^2 f(x)\, v$, but this has increased time and space complexity since reverse is applied to a larger computational graph.

Reverse mode AD in deep learning frameworks. In deep learning applications, Reverse mode AD, also called **backpropagation**, is used to compute the gradient of a scalar loss function w.r.t. the potentially large number parameters of a deep neural network. This can be done by setting $\overline{y} = 1 \in \mathbb{R}$ in Algorithm 2.4.2. In the PyTorch library, AD is performed by saving the backward pointers to the arguments and a link to the reverse Jacobian-vector product operation $D\psi(u)^\top \overline{u}$ together with the result of an elemental operation ψ, during the forward pass. Instead, during the backward pass the evaluation graph is traversed backwards to compute the adjoints variables. Pytorch uses a dynamic graph that is constructed at runtime and is by default discarded after the backward pass. While pytorch only implements reverse mode AD, other frameworks like JAX (Bradbury et al., 2018) also implement forward mode AD which, as we saw, can be useful for fast Hessian-vector products.

2.4.3 Computational Complexity

Let Time and MEM be two time and space complexity measures such that for any function g

$$\text{TIME}\big[g(x)\big], \qquad \text{MEM}\big[g(x)\big]$$

are respectively the time and memory required for the evaluation of g at x, given by a certain fixed program. More precisely, $\text{MEM}\big[g(x)\big]$ measures the number of floating point allocations required, while $\text{TIME}\big[g(x)\big]$ can be thought of as a weighted sum of

the number of additions, multiplications, memory moves and non-linear operations performed, where each weight represents the (hardware-dependent) time required by each operation type. When $Dg(x)\,\dot{x}$ or $Dg(x)^\top\,\bar{y}$ are added to their arguments, TIME and MEM measure the time and memory complexity of forward or reverse AD respectively.

Time complexity and the cheap gradient principle. To study the time complexity we note that according to the decomposition of f in elemental functions we can reasonably assume that

$$\text{Time}[f(x)] = \sum_{i=1}^{l} \text{Time}[\psi_i(x)].$$

From Algorithms 2.4.1 and 2.4.2 we observe that for each elemental ψ_i, Forward mode AD computes $\psi_i(u_i)$ and $D\psi_i(u_i)\,\dot{u}_i$, while Reverse mode AD computes $\psi_i(u_i)$ and $\bar{u}_i + D\psi_i(u_i)^\top\,\bar{v}_i$. Therefore, we assume the following *forward/reverse sub-additivity* property

$$\text{Time}[f(x), Df(x)\,\dot{x}] \leq \sum_{i=1}^{l} \text{Time}[\psi_i(x), D\psi_i(u_i)\,\dot{u}_i],$$
$$\text{Time}[f(x), Df(x)^\top\,\bar{y}] \leq \sum_{i=1}^{l} \text{Time}[\psi_i(x), \bar{u}_i + D\psi_i(u_i)^\top\,\bar{v}_i].$$

We can easily show that when the library of elemental Ψ contains scalar functions in the polynomial core, i.e. addition, multiplication, unitary sign switch and constant assignment, then for any $\psi \in \Psi$ and vectors $u, \dot{u}, \bar{v}$ we get

$$\text{Time}[\psi(u), \psi(u)\,\dot{u}] \leq C_f \text{Time}[\psi(u)],$$
$$\text{Time}[\psi(u), \bar{u} + \psi(u)^\top\,\bar{v}] \leq C_r \text{Time}[\psi(u)],$$

where $C_f \in [2, 5/2]$ and $C_r \in [2, 5]$. These constants can also be improved for *vector mode* forward and reverse AD, i.e. when $\dot{x}$ and $\bar{y}$ are replaced by the matrices $\dot{X} \in \mathbb{R}^{d\times p}$ and $\bar{Y} \in \mathbb{R}^{m\times p}$. In this case a lower number of additions and memory accesses are performed. For a richer Ψ, e.g. containing vector-valued elementals a

similar linearly bounded complexity holds but with different constants. Combining the above inequalities with the forward/reverse sub-additivity we finally obtain

$$\text{Time}[f(x), Df(x)\,\dot{x}] \leq C_f \text{Time}[f(x)],$$
$$\text{Time}[f(x), Df(x)^\top \bar{y}] \leq C_r \text{Time}[f(x)].$$

Which means that the time complexity of forward and reverse Jacobian-vector products is at most linear in the time of evaluating the function. When applying reverse mode with $\bar{y} = 1 \in \mathbb{R}$ to a scalar valued function f we obtain

$$\text{Time}[f(x), \nabla f(x)] \leq C_r \text{Time}[f(x)],$$

which is referred to as the *cheap gradient principle*. Computing the same quantity with forward AD would require instead d forward Jacobian-vector products, which is less convenient already if $d > 5$. This fundamental and somewhat counterintuitive result is one of the root causes of the widespread use of reverse mode AD in deep learning to compute the gradient of the loss w.r.t. the numerous parameters of a neural network.

Computing the full jacobian. We can easily see that since $Df(x) = \sum_{i=1}^{d} Df(x)\, e_i^d = \sum_{i=1}^{m} e_i^{m\top} Df(x)$ where e_i^d and e_i^m are elements of the canonical basis of $\mathbb{R}^d$ and $\mathbb{R}^m$ respectively, computing the full Jacobian requires either d forward mode AD computations or m reverse AD computations and hence its time complexity is proportional to either d or m times that of evaluating the function. However, notice that the forward pass for the function evaluation can be reused for each Jacobian-vector product to save time. Computing the full Jacobian is inherently slower than a single Jacobian-vector product, but can be useful when such Jacobian is repeatedly used, for example when Jacobian-vector products are repeatedly computed with the same Jacobian to approximate the Newton step for a quadratic minimization problem.

Memory allocation. In the Algorithms 2.4.1 and 2.4.2, we defined auxiliary variables $\{v_i\}$, $\{\dot{v}_i\}$ and $\{\bar{v}_i\}$ which are used to compute the function value, forward

mode AD and reverse mode AD respectively. To save memory, some of these auxiliary variables can be overwritten by others, i.e. share the same memory location, when this does not change the final result. In particular to evaluate $f(x)$, v_i can be overwritten by any other variable v_k if v_i is never an argument of ψ_j with $j > k$, i.e. if $k \geq \max_{i \prec j} j$. The same also applies to $\{\dot{v}_i\}$ in forward mode AD, which can mimic the same overwrite protocol of $\{v_i\}$: when v_k overwrites v_i, $\dot{v}_k$ also overwrites $\dot{v}_i$. Therefore, since the dimension of v_i is the same as that of $\dot{v}_i$ we have

$$\mathrm{MEM}\big[f(x), Df(x)\,\dot{x}\big] = 2\mathrm{MEM}\big[f(x)\big].$$

For reverse mode AD instead, since each v_i is accessed sequentially in the backward pass from v_l to v_0 to compute the adjoints variables $\{\overline{v}_i\}$, their values must be saved separately during the forward pass to be able to retrieve them even in case of overwriting. It may be convenient in case of sequential execution if this separate storage is a *last in first out* (LIFO) queue, also called *tape*, for faster retrieval during the backward pass. Therefore, if we denote with $\dim(v_i)$ the dimension of the vector v_i we obtain

$$\mathrm{MEM}\big[f(x), Df(x)^\top \overline{y}\big] \sim \sum_{i=0}^{l} 2\dim(v_l) \geq 2\,\mathrm{MEM}\big[f(x)\big].$$

In general, for programs allowing several overwrites (e.g. programs evaluating feed forward neural networks or exectuing optimization algorithms), the memory cost of reverse mode AD can be largely greater than that of forward mode AD. For this reason, there are several techniques that aim at decreasing this cost like the aforementioned *checkpointing*. Another strategy is that of recomputing v_i during the backward pass when this can be done cheaply using the inverse of elemental functions, e.g. in the case of some linear operations. This allows to save memory and potentially decrease the computation time when re-computing is cheaper than retrieving from memory, but can result in catastrophic cancellations and other numerical problems arising from finite precision arithmetic since v_i in the backward pass is computed differently and possibly less accurately than in the forward pass.

2.5 Bilevel Problems in Machine Learning

In this section we describe important machine learning applications that can be framed as bilevel optimization problems. In particular, we will cover hyperparameter optimization in Section 2.5.1, meta-learning in Section 2.5.2, equilibrium models in Section 2.5.4 and poisoning adversarial attacks in Section 2.5.3. In the following Chapters, we show experimental results in all of these scenarios. We note that the bilevel framework is not limited to such applications.

2.5.1 Hyperparameter Optimization

Hyperparameter optimization (HPO) is the problem of finding the best value for the parameters controlling the behavior of a learning algorithm. These parameters are called *hyperparameters* to distinguish them from the parameters set through learning.

Consider the supervised learning problem[2] of finding a hypothesis $h : \mathcal{X} \mapsto \mathcal{Y}$ which minimizes

$$\mathbb{E}_{(x,y)\sim\rho_{x,y}}[\mathcal{E}(h(x),y)], \tag{2.7}$$

where $\rho_{x,y}$ is the unknown data distribution while $\mathcal{E}$ is a suitable error measure for the problem, which in this case might even be discontinuous (e.g. the classification error). This problem is generally solved through an algorithm $\mathcal{A}$ (possibly randomized) which takes as input some training dataset $D = \{(x_i, y_i)\}_{i=1}^n$ of i.i.d. examples each sampled from $\rho_{x,y}$ and some hyperparameters $\lambda \in \Lambda$ and returns the hypothesis h: $\mathcal{A}(D,\lambda) = h$. For example, if we let h be parameterized by w we can have

$$\mathcal{A}(D,\lambda) = h_{w^*}, \quad \text{where} \quad w^* \in \operatorname*{argmin}_{w\in\mathbb{R}^d} \frac{1}{n} \sum_{(x,y)\in D} \mathcal{L}(h_w(x),y) + \mathcal{R}(w,\lambda),$$

with $\mathcal{L}$ being a loss function surrogate for $\mathcal{E}$ and usually continuous and convex on the first argument, $\mathcal{R}$ is some regularization (e.g. $\lambda \in \mathbb{R}_{++}$ and $\mathcal{R}(w,\lambda) = \lambda\|w\|^2$) and the model h_w could be e.g. a linear model or a more complex deep neural network.

The definition of $\mathcal{A}$ can vary greatly depending on the desired level of abstraction. In the above example it outputs a minimizer of the regularized training loss,

[2] Note that hyperparameter optimization can also be applied to reinforcement and unsupervised learning.

which does not generally have a closed for solution and could be too expensive to compute accurately. Another approach could be for $\mathcal{A}$ to output the mapping obtained after some steps of (stochastic) gradient descent or another first-order optimization method on the regularized loss. In addition to regularization hyperparameters, λ could contain parameters involving the loss (e.g. the weights of different losses), design hyperparameters (e.g. the ones that chooses which network architecture and how many layers and neurons to use), optimization hyperparameter (e.g. the learning rate schedule and number of iterations) and others.

Hyperparameter optimization aims at solving the minimization problem

$$\min_{\lambda \in \Lambda} \{ f(\lambda) := \mathcal{C}(\mathcal{A}(D, \lambda)) \} \tag{2.8}$$

where $\mathcal{C}$ provides a good substitute for the expected error in eq. (2.7). Note that when $\mathcal{A}$ is defined as a minimizer of the training error, the problem in eq. (2.8) is a bilevel optimization problem. The complexity of $\mathcal{A}$ and the hyperparameter space Λ defines how large is the space of hypothesis. However, the larger the hypothesis space the higher the computational complexity of the problem. In general, choosing the appropriate Λ is a difficult problem which we do not tackle in this thesis. We refer the interested reader to (Perrone et al., 2019), which studies a transfer learning approach.

One of the most popular choices for $\mathcal{C}$ is the *validation loss*

$$\mathcal{C}(h) = \frac{1}{n'} \sum_{(x,y) \in D'} \hat{\mathcal{L}}(h(x), y), \tag{2.9}$$

where $D' = (x_{n+i}, y_{n+i})_{i=1}^{n'}$ is the validation set, containing i.i.d. samples from $\rho_{x,y}$ independent of D and $\hat{\mathcal{L}}$ is a surrogate loss for $\mathcal{E}$, which might be different from $\mathcal{L}$ and sometimes even equal to $\mathcal{E}$. While here we only consider a train and a validation set, usually obtained by splitting in two the original training set, we can instead take several non-overlapping training-validation splits of the same dataset using K-fold cross-validation (Stone, 1974). This will be more expensive but will also reduce the variance in the estimated quantities, which can be beneficial with small datasets.

Once the final hyperparameters and corresponding model (λ^*, h^*) are chosen, the final performance is measured on a hold out *test set* $D'' = (x_{n+n'+i}, y_{n+n'+i})_{i=1}^{n''}$ containing i.i.d. samples from $\rho_{x,y}$ independent from D and D' by measuring the *test set error* $n^{-1}\sum_{(x,y)\in D''} \mathcal{E}(h^*(x), y)$. This also allows to detect if the hyperparameter optimization procedure has overfitted to the validation set D' (low $\mathcal{C}(h^*)$ but high test set error), which may happen more often with a large number of hyperparameters and/or small datasets.

Other choices for $\mathcal{C}$ which do not rely on a validation dataset are e.g. Stein's unbiased risk estimate (Stein, 1981) and Akaike information criterion (Akaike, 1998). However, these usually require additional statistical modeling assumptions and are therefore less generally applicable than eq. (2.9).

Challenges. Solving the hyperparameter optimization problem in Equation (2.8) presents several challenges. The HPO objective function f is usually non-convex, can be non-smooth or even discontinuous and very expensive to compute, especially when dealing with deep neural networks and large datasets. The hyperparameters might be heterogeneous (e.g. $\lambda_1 \in \mathbb{R}_{++}$ is the learning rate and $\lambda_2 \in \mathbb{N}$ is the number of layers in the neural network) and of different types (reals, naturals, and even categorical). Moreover, there might be many hyperparameters whose choice greatly affects the performance of the algorithm (this happens more the more complex the model is).

Hyperparameter optimization methods

In the following, we describe commonly used hyperparameter optimization methods highlighting advantages and disadvantages. Since the literature in the field is vast, here we focus on describing only a few simple and established methods without covering possible variations and extensions which may also lie in between the proposed categorization. We refer to each coordinate of λ, a value $\lambda \in \Lambda$, and f in Equation (2.8) as hyperparameter, hyperparameter configuration and HPO objective respectively.

Manual search, also jokingly called "grad student descent", consists in the human practitioner iteratively deciding which hyperparameter configuration λ to

try next. The decision is usually based on the value of the HPO objective for the previous configurations, together with heuristics based on the previous experience and insights on the problem of the practitioner. Manual search is easy to implement and potentially cheap to execute. However, since it relies on human decisions, it is very hard to reproduce and, if done improperly, might even be biased by leaking information from the test set D'', which instead should only be used for the final evaluation of the method. Despite the downsides, manual search is widely used, especially for very large scale problems, mainly due its practically non-existent implementation overhead.

Grid search consists in choosing a priori a grid of hyperparameter configurations to evaluate. More specifically, for each hyperparameter, some candidates are chosen and all the configurations obtained as combinations of candidates are evaluated. Then, the configuration achieving the best HPO objective is selected. Usually, for categorical hyperparameters, all values are candidates, while for real and natural ones, candidates are chosen equally spaced in the domain or in the logarithm of the domain (Bengio, 2012). Grid search can be parallelized: each configuration can be evaluated in parallel since the set of configurations is chosen a priori. However, optimizing more than 3 hyperparameters becomes prohibitive due to the number of configurations growing exponentially with the number of hyperparameters.

Random search, like grid-search, also consists in constructing a set of hyperparameters configurations, then evaluating them and finally picking the one with the lowest $f(\lambda)$. However, each configuration is drawn randomly from the same distribution over the search space Λ. Usually, the distribution of each hyperparameter is independent of the others and popular choices are simple distributions as the uniform or the log-uniform (or reciprocal) distribution. For example if λ_1 is the number of layers of a neural network, λ_2 is the learning rate and λ_3 is the regularization parameter we can have each configuration $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ drawn from the distribution

$$\mathcal{U}[\{1, \dots, 100\}] \times \log\mathcal{U}[10^{-4}, 10^{3}] \times \log\mathcal{U}[10^{-4}, 10^{3}]$$

where $\mathcal{U}$ and $\log\mathcal{U}$ are the uniform and log-uniform (or reciprocal) distributions respectively. Random search is easy to implement and to parallelize (as grid search). Furthermore, it can be particularly effective even with many hyperparameters when they have a low *effective dimensionality* (Bergstra and Bengio, 2012), e.g. if only a few have a significant impact on the HPO objective. For these reasons, Bergstra and Bengio (2012) propose to use random search instead of grid search as the default hyperparameter optimization method. Despite the advantages, random search in high dimensions is affected by the curse of dimensionality: even when the number of configurations is large, only a small portion of the search space is explored when Λ is high-dimensional. This means that if the effective dimensionality of the search space is proportional to the number of hyperparameters n_λ, the probability of finding a good configuration drops exponentially with n_λ. This makes it difficult to handle many important hyperparameters, as in the case of grid search.

Model-based methods are iterative methods which choose the next hyperparameter configuration(s) to evaluate with the aid of a surrogate model of f, namely $\hat{f}$, constructed by using information obtained from previously evaluated configurations $\{(\lambda_i, f(\lambda_i))\}$. The surrogate model should be substantially cheaper to evaluate than f itself and usually measures also the uncertainty on its estimates for the values of f.

Bayesian Optimization (Bergstra et al., 2011; Hutter et al., 2011; Snoek et al., 2012) is one of the most popular and principled model-based approaches. It relies on Bayesian statistics, in particular on Gaussian process regression, to model the surrogate $\hat{f}$. In Gaussian process regression, f is assumed to be continuous and for every $\lambda \in \Lambda$, $\hat{f}(\lambda)$ is modeled as a Gaussian random variable whose mean and covariance depend on previously evaluated points. At the start of the procedure, $\hat{f}(\lambda)$ has the prior distribution $\mathcal{N}(\mu_0(\lambda), \sigma_0(\lambda))$, where μ_0 and σ_0 are simple mean and covariance functions (e.g. $\mu_0(\lambda) = 0$, $\sigma_0(\lambda) = \alpha_0 > 0$) that can encode prior knowledge on the problem. Furthermore, we set $\sigma_0(\lambda) = \kappa(\lambda, \lambda)$ where κ is a *kernel function* measuring the similarity between its arguments (e.g. $\lambda_1, \lambda_2 \in \Lambda$, $\kappa(\lambda_1, \lambda_2) = \alpha_0 \exp(-\|\lambda_1 - \lambda_2\|^2)$). At iteration s, let $\{\lambda_i\}_{i=1}^{s}$ be the previously evaluated points,

then the posterior distribution of $\hat{f}(\lambda) \mid \{\lambda_i, f(\lambda_i)\}_{i=1}^s$ is $\mathcal{N}(\mu_s(\lambda), \sigma_s(\lambda))$ where

$$\mu_s(\lambda) = \mu_0(\lambda) + v_\kappa(\lambda)^\top K^{-1}(y - \hat{y}), \quad \sigma_s^2(\lambda) = \sigma_0(\lambda) - v_\kappa(\lambda)^\top K^{-1} v_\kappa(\lambda),$$

K is the kernel matrix such that $K_{ij} = \kappa(\lambda_i, \lambda_j)$ for every $i, j \in \{1, \dots, s\}$ and

$$v_\kappa(\lambda) = \{\kappa(\lambda, \lambda_i)\}_{i=1}^s, \qquad y = \{f(\lambda_i)\}_{i=1}^s, \qquad \hat{y}_0 = \{\mu_0(\lambda_i)\}_{i=1}^s.$$

Then, the next point to evaluate, i.e. λ_{s+1}, is the one which maximizes a so-called *acquisition function* balancing exploitation and exploration. A common choice for the acquisition function is the *expected improvement*:

$$a(\lambda) = \mathbb{E}[\max\{f_{min} - \hat{f}(\lambda), 0\} \mid \{\lambda_i, f(\lambda_i)\}_{i=1}^s],$$

where $f_{min} = \min_{i \in \{1,s\}} f(\lambda_i)$, that has a closed form when the posterior is Gaussian. The procedure is usually warm-started by choosing some initial points via random search. Note that the cost of the kernel matrix inversion (K^{-1}), which scales cubically in the number of evaluated points, but this is usually not a big issue for hyperparameter optimization, since evaluating f is usually very expensive and dominates the total cost. Compared to random and grid search, Bayesian Optimization is still a black box technique (only requiring function values), but is less parallelizable, far more complex and introduces a number of design choices, e.g. which prior μ_0, σ_0, kernel κ and which acquisition function a to use, that may limit its accessibility (despite defaults being usually provided in popular libraries). Furthermore, it struggles to optimize more than 20 hyperparameters (Frazier, 2018).

Population based methods are iterative methods that, at each round, generate a population of hyperparameter configurations to evaluate which depends on the population evaluated at previous rounds. They usually require a greater number of function evaluations than Bayesian optimization, but at each round, the evaluation for different configurations can be executed in parallel. Evolutionary strategies and some bandit algorithms fall into this category.

Evolutionary strategies are heuristics methods which take inspiration from natural evolution. For example, the population in the next round may be formed by selecting, mutating (by adding random perturbation to the features) and/or combining (by mimicking sexual reproduction) the best individuals (according to their *fitness score*, $f(\lambda)$ in our case) of the current population. The evolutionary strategy CMA-ES has been applied to optimize the parameters of deep neural network (Loshchilov and Hutter, 2016). More recently, evolutionary strategies have been successfully applied for neural architecture search (Real et al., 2019, 2020).

Bandit approaches instead cast hyperparameter optimization as a multi-armed bandit problem (Lattimore and Szepesvári, 2020). One example is the *successive halving* strategy proposed by Jamieson and Talwalkar (2016), which consists in starting from a large population of configurations which is partially evaluated in parallel, i.e. by stopping the iterative learning algorithm $\mathcal{A}$ after some fixed number of iterations, and then resuming $\mathcal{A}$ only for the best half of the configurations (according to this partial evaluation). This procedure is then recursively applied to the remaining half until $\mathcal{A}$ is executed completely only for a few configurations, from which the best one is finally chosen. This procedure assumes that algorithms which learn fast will also perform better at the end of training, which is often the case, and exploits the iterative nature of learning algorithms to save compute time.

Gradient-based methods exploit (approximations of) the gradient of the HPO objective f in eq. (2.8), also called the hypergradient. Usually they are iterative methods that update λ as in (stochastic) gradient descent, but where the gradient of the HPO objective is often replaced by an approximation, due to its high computational complexity. Contrary to most hyperparameter optimization methods, gradient based approaches are usually *white-box*, requiring knowledge on the structure of f, which is combined with automatic differentiation to compute accurate and efficient derivatives. The main advantage of using gradients is that it enables in principle to optimize a very large number of hyperparameters, as shown by Lorraine et al. (2020), while also being competitive even on problems with a few hyperparameters (Pedregosa, 2016). On the other hand, gradient-based methods require the objective f to

be differentiable and the hyperparameters to be real-valued. The former assumption can be easily satisfied by using commonly used surrogates in place of non-smooth error measures: e.g. using the cross-entropy loss in place of the classification error. Moreover, discrete hyperparameter can be replaced by continuous (Franceschi et al., 2017; Liu et al., 2018) or probabilistic (Franceschi et al., 2019; Niepert et al., 2021) relaxations, although this may require also to change the underlying learning algorithm. Main disadvantages of gradient-based methods are that they are not directly parallelizable and have a large implementation overhead. Indeed, f is usually defined via either a gradient-based optimization routine or the minimizer of some function, and in most cases the automatic differentiation technique used should be able to handle second-order derivatives. In some cases computing the hypergradient could also have a prohibitive memory cost (Maclaurin et al., 2015; Franceschi et al., 2018; Grazzi et al., 2020). Finally, we mention that the capability of optimizing a large number (potentially millions) of hyperparameters increases the chances of overfitting to the validation set (Franceschi et al., 2018).

2.5.2 Meta-Learning

Humans still surpass machines in their ability to transfer their learned skills and knowledge. Indeed, when a football player starts playing basketball for the first time, they do not learn it from scratch, but they may, for example, rely on their improved awareness of the other players developed playing football, to learn faster how team play works. Many scientists believe that this transfer learning ability across different activities or tasks is necessary to achieve artificial general intelligence.

The meta-learning framework represents this concept by defining the objective to learn, from a group of tasks, a learning algorithm capable of adapting to new related tasks faster, i.e. with fewer examples. More formally, focusing on supervised learning and inspired by Denevi et al. (2019a), we define each task $\mathcal{T}$ as a set containing an error measure $\mathcal{E}$, a distribution $\rho_{x,y}$ and the number of training example n, i.e. $\mathcal{T} = \{\mathcal{E}, \rho_{x,y}, n\}$. We also define the task distribution $\rho_{\mathcal{T}}$, which encodes how the task are related. The goal of single-task learning is finding the hypothesis h

which minimizes

$$\mathbb{E}_{(x,y)\sim\rho_{x,y}}[\mathcal{E}(h(x),y)],$$

through a learning algorithm that uses n training examples sampled form $\rho_{x,y}$. By contrast, the goal of meta-learning is to find a learning algorithm $\mathcal{A}$, taking n training examples as input and returning an hypothesis as output, which aims at minimizing

$$\mathbb{E}_{\mathcal{T}\sim\rho_{\mathcal{T}}}\mathbb{E}_{D\sim\rho_{x,y}^{n}}\mathbb{E}_{(x,y)\sim\rho_{x,y}}[\mathcal{E}(\mathcal{A}(D)(x),y)], \tag{2.10}$$

using a set of training datasets, called *meta-training* dataset, each belonging to a task sampled from the task distribution $\rho_{\mathcal{T}}$. In light of eq. (2.10), it might be useful to think about meta-learning as a "lifted" form of learning, where hypotheses are replaced by learning algorithms and examples are replaced by tasks. We also note that if $\mathcal{A}(\mathcal{D})$ is defined as the minimizer of some loss function over D, the meta-learning framework exhibits a bilevel structure with an upper-level learning problem across tasks and several lower-level single-task learning problems.

Meta-learning algorithms work well in a few-shot learning scenario where the meta-training dataset contains many tasks, each with only a few (usually less than 50) examples, and where the evaluation is done exclusively on new tasks not present in the meta-training dataset. This is in contrast with *multi-task* learning (Caruana, 1997), where both training and testing is done on the same usually small group of tasks, each containing a larger number of examples. Mini-Imagenet (Ravi and Larochelle, 2017), Omniglot (Lake et al., 2011) and more recently Meta-Dataset (Triantafillou et al., 2019) are popular few-shot learning benchmarks in the domain of image classification. However, it must be noted that a simple last-layer adaptation strategy has been shown to outperform several meta-learning methods on these benchmarks (Tian et al., 2020), although by exploiting additional and commonly inaccessible information (Wang et al., 2021).

In the following, we categorize and outline the most popular meta-learning methods.

Model-based meta-learning methods exploit the power of very expressive

models such as deep neural networks which can take sets or sequences as inputs, to transform the entire training datasets D of each task together with one or multiple test inputs into prediction for the test outputs. This class of methods ignores the bilevel structure of the meta-learning framework: single-task learning is entirely performed in the forward pass of the network while meta-learning is done through (stochastic) gradient updates of the network's weights (the meta-parameters), where the objective function is the sum of the losses of the meta-training tasks. Some model-based meta-learning methods are based e.g. on LSTMs (Hochreiter et al., 2001), neural augmented networks (Santoro et al., 2016) and temporal convolution plus attention Mishra et al. (2018). More recently, transformers have been shown to perform well in the meta-learning setting on language modeling tasks (Brown et al., 2020; Schick and Schütze, 2021) and to classify tabular data (Hollmann et al., 2022).

Algorithmic meta-learning methods directly exploit the bilevel structure of the meta-learning framework by augmenting established single-task learning algorithms e.g. logistic regression or SVM, with meta-parameters learned across meta-training tasks. This provides additional inductive bias often enabling generally faster and more data efficient meta-learning and less meta-parameters compared to model-based methods.

Several algorithmic meta-learning methods developed in the late 90s and 2000s learn to select from a pool of single-task learning algorithms, the best one for each task (Vilalta and Drissi, 2002; Smith-Miles, 2009). For methods of this kind, learning is usually done in two stages. First, each learning algorithm in the pool is executed on every meta-training task. Then, the algorithm selection problem is treated as a supervised learning problem having as inputs some features of the task, called meta-features, and as outputs either the cross-validation errors or the index of the best performing algorithm computed in the first stage. Meta-features are often hand-engineered statistics, although a more recent approach (Edwards and Storkey, 2016) uses learned dataset representations output of a variational autoencoders. An alternative to just selecting one algorithm is using an ensemble of all the algorithms in the pool. This can improve the performance, although it increases the computational

cost.

In recent times, several end-to-end meta-learning approaches learn meta-parameters, usually weights of a neural network, that augment a single-task learning algorithm. Fore example, some methods augment simple non-parametric learning algorithms like k-nearest neighbor with meta-learned distance metrics (Sung et al., 2018) and example-specific (Vinyals et al., 2016) or class-specific (Snell et al., 2017) embeddings. Other methods meta-learn either entire single-task optimizers (Ravi and Larochelle, 2017; Andrychowicz et al., 2016), or embeddings to be used in linear or logistic regression (Bertinetto et al., 2018; Franceschi et al., 2018) or SVM (Lee et al., 2019). Instead, the approaches by Finn et al. (2017); Rajeswaran et al. (2019) meta-learn the parameters of a neural network at initialization. As highlighted by Franceschi et al. (2018), several algorithmic meta-learning approaches learn through first-order bilevel optimization techniques, i.e. exploiting the gradient of the validation loss of a mini-batch of tasks w.r.t. the meta-parameters, where such loss is computed after the execution of the single-task learning algorithm on each task of the batch. We also note that for this kind of methods, the number of meta-parameters is usually in the order of thousands or millions and often even outnumbers task-specific parameters. Therefore, black-box optimization methods, which only rely on function evaluations and not gradients, are much less effective.

2.5.3 Poisoning Adversarial Attacks

In poisoning adversarial attacks, a malicious agent or attacker aims at corrupting some examples in a dataset so that a model trained on such dataset will perform worse or will behave in a way that the attacker can exploit when deployed. A recent comprehensive survey is provided by Cinà et al. (2022). The rise in complexity and scale of modern machine learning models trained with distributed and federated optimization techniques and data often coming from untrusted sources, has greatly increased the potential harm of such attacks, which are considered a major threat by industry organizations (Kumar et al., 2020).

An effective albeit computationally expensive way of carrying out such attacks is by solving a bilevel optimization problem with the objective of finding the poisoned

examples that maximize the error on the validation set (indiscriminate attacks) or to maximize the error only on a certain class of examples (targeted attacks). More formally, the bilevel problem for indiscriminate attacks can be written as

$$\min_{\mathcal{D}_p \in \mathcal{C}} -\mathcal{L}(w(\mathcal{D}_p); \mathcal{D}'), \qquad w(\mathcal{D}_p) \in \operatorname*{argmin}_w \mathcal{L}(w; \mathcal{D} \cup \mathcal{D}_p) + \lambda \mathcal{R}(w),$$

where $\mathcal{L}$ is the loss for the task (e.g. MSE, cross-entropy), w is the parameter vector of the model, $\lambda > 0$, $\mathcal{R}$ is a regularizer (e.g. $\mathcal{R}(w) = \|w\|^2$), $\mathcal{D}$ and $\mathcal{D}'$ are the train and validation datasets respectively, while $\mathcal{D}_p$ is the set of poisoned examples and $\mathcal{C}$ is a usually compact set representing the constraints of the problem (e.g. features of poisoned examples should not be too far from some training examples so that there are higher chances that the attack is undetected). This class of attacks has been pioneered by Biggio et al. (2012) for SVM models, while Muñoz-González et al. (2017) have developed a more efficient algorithm that can be applied to deep neural networks with potentially many weights. Note also that $\mathcal{D}_p$ can be very high-dimensional, since it is usually of dimension $n_p \times (d+1)$ where n_p is the number of poisoned examples (which can be up to 30% of the training dataset) while d is the number of features of each example (number of pixels times number of channels in the case of images). To reduce the dimensionality of the problem (Yang et al., 2017; Feng et al., 2019) propose to optimize the parameters of a generative network which outputs $\mathcal{D}_p$, instead of $\mathcal{D}_p$ directly.

Carrying out bilevel optimization for poisoning can be hard and computationally expensive. Alternatives approaches are e.g. heuristic methods based on label flipping (Biggio et al., 2011; Xiao et al., 2012, 2015), where only the example's labels are modified, or feature collision (Shafahi et al., 2018), where noise is added to the features of the poisoned image to make its latent representation similar to a given target image at test time. Furthermore, Cinà et al. (2021) showed that a simple heuristic can be effective in attacking linear models, and also demonstrated how the bilevel problem in data poisoning can be difficult to optimize with gradient-based methods.

2.5.4 Equilibrium Models

Equilibrium models are machine learning models where part of the computation does not have an analytic expression but is instead defined implicitly as the solution of a parametric equilibrium problem. For example let $h_w : \mathcal{X} \mapsto \mathbb{R}$ be an equilibrium model acting on the input space $\mathcal{X}$ with parameters $w = (w_1, w_2)$, we can write

$$h_w(x) = w_1^\top z(x, w_2), \qquad z(x, w_2) = \phi(z(x, w_2), w_2, x), \tag{2.11}$$

where ϕ is a parametric map which depends both on the input $x \in \mathcal{X}$ and weights w_2 and $z(x, w_2)$ is the equilibrium representation associated to x. Note that optimizing w on a given training set is equivalent to solving a bilevel optimization problem with one equilibrium problem per training example at the lower-level.

This class of models has been originally explored in some recurrent and graph neural networks (Almeida, 1987; Pineda, 1987; Scarselli et al., 2008) and has been recently popularized by Bai et al. (2019), who introduced deep equilibrium models (DEQs), where the map in the equilibrium problem is structured as one or multiple layers in a deep neural network. As noted in the paper, these "equilibrium layers" are equivalent to having an infinite sequence of weight-tied standard neural layers whose activation converge to a fixed-point. The advantage of the equilibrium formulation is that the forward pass can be sped-up using faster root-finding algorithms, and derivatives can be taken implicitly by applying another root-finding method. Using implicit derivatives reduces the memory cost, which contrary to standard backpropagation, does not increase with depth (or with the number of root-finding iterations in this case). DEQs have been motivated by observing that the hidden activations of many deep sequence models converge towards an equilibrium. However, there are also evidences that even weight-tied deep sequence models have oscillating activations (Lan et al., 2019). The first application of DEQs was language modeling (Bai et al., 2019), but they have also been applied to vision tasks (Bai et al., 2020) and on graph neural networks (Gu et al., 2020).

Bai et al. (2019) utilize the Broyden method, a quasi Newton method, both to compute the fixed point in the forward pass and to solve the linear system in

the backward pass. This method requires to store the approximation of the inverse Jacobian also during inference of $\phi(\cdot, w_2, x)$, which could be problematic for large states z. However, in principle, the Broyden method can be substituted by any converging and potentially cheaper root-finding method. Vanilla DEQs exhibit an increasing number of root-finding solver iterations during training, which is a symptom of instability and greatly increases training and inference time compared to standard deep networks. To address this issue, Winston and Kolter (2020) use monotone operator theory to construct ϕ to have a unique fixed point that can be found via operator splitting methods with linear iteration complexity, while Bai et al. (2021) add a regularizer term involving the Jacobian of ϕ.

Chapter 3

Deterministic Hypergradient Approximation

3.1 Introduction

The principal goal of this chapter is to study the degree of approximation to the hypergradient of certain deterministic iterative schemes based on iterative or implicit differentiation. In the rest of the introduction we present the bilevel framework, alongside some relevant examples in machine learning. We then outline the gradient approximation methods and highlight our main contributions.

The bilevel framework. We consider the following bilevel problem.

$$\begin{aligned} &\min_{\lambda \in \Lambda} f(\lambda) := E(w(\lambda), \lambda) \\ &\quad \text{with } \; w(\lambda) = \Phi(w(\lambda), \lambda), \end{aligned} \tag{3.1}$$

where Λ is a closed convex subset of $\mathbb{R}^n$ and the functions $E\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}$ and $\Phi\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}^d$ are continuously differentiable on open sets containing their domain. We assume that the lower-level problem in (3.1) (which is a fixed point-equation) admits a unique solution. However, in general, explicitly computing such solution is either impossible or expensive. When f is differentiable, this issue affects the evaluation of the hypergradient $\nabla f(\lambda)$, which at best can only be approximately computed.

A prototypical example of the bilevel problem (3.1) is

$$\begin{aligned} &\min_{\lambda \in \Lambda} f(\lambda) := E(w(\lambda), \lambda)) \\ &\quad \text{with } \; w(\lambda) = \operatorname*{argmin}_{u \in \mathbb{R}^d} \mathcal{L}(u, \lambda), \end{aligned} \tag{3.2}$$

where $\mathcal{L}\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}$ is a loss function, twice continuously differentiable and strictly convex w.r.t. the first variable. Indeed, since $w(\lambda)$ is the unique point which satisfies the equation $\nabla_1 \mathcal{L}(w(\lambda), \lambda) = 0$, then if we let $\Phi(w, \lambda) = w - \alpha(\lambda)\nabla_1 \mathcal{L}(w, \lambda)$, i.e. the gradient descent update where $\alpha\colon \Lambda \to \mathbb{R}_{++}$ is any differentiable function that outputs the step size, then problem (3.2) and problem (3.1) are equivalent. Specific examples of problem (3.2), which include hyperparameter optimization and meta-learning, are discussed in Section 3.4.1.

Other instances of the bilevel problem (3.1), which are not of the special form of problem (3.2), arise in the context of so-called equilibrium models (EQM). Notably, these comprise some types of connectionist models employed in domains with structured data. Stable recurrent neural networks (Miller and Hardt, 2019), graph neural networks (Scarselli et al., 2008) and the formulations by Bai et al. (2019) belong to this class. EQM differ from standard (deep) neural networks in that the internal representations are given by fixed points of learnable dynamics rather than compositions of a finite number of layers. The learning problem for such type of models can be written as

$$\begin{aligned} &\min_{\lambda = (\gamma, \theta) \in \Lambda} f(\lambda) := \sum_{i=1}^{n} E_i(w_i(\gamma), \theta), \\ &\quad \text{with } w_i(\gamma) = \phi_i(w_i(\gamma), \gamma), \text{ for } 1 \leq i \leq n, \end{aligned} \tag{3.3}$$

where the operators $\phi_i : \mathbb{R}^d \times \Lambda \to \mathbb{R}^d$ (here $\Phi = (\phi_i)_{i=1}^n$) are associated to the training points x_i's, and the error functions E_i are the losses incurred by a standard supervised algorithm on the transformed dataset $\{w_i(\gamma), y_i\}_{i=1}^n$. A specific example is discussed in Section 3.4.2.

In this chapter, we present a unified analysis which allows to quantitatively com-

pare popular methods to approximate $\nabla f(\lambda)$ in the general setting of problem (3.1). The strategies we consider can be divided in two categories:

1. *Iterative Differentiation (ITD)* (Maclaurin et al., 2015; Franceschi et al., 2017, 2018; Finn et al., 2017). One defines the sequence of functions $f_t(\lambda) = E(w_t(\lambda), \lambda)$, where $w_t(\lambda)$ are the fixed-point iterates generated by the map $\Phi(\cdot, \lambda)$. Then $\nabla f(\lambda)$ is approximated by $\nabla f_t(\lambda)$, which in turn is computed using forward (FMAD) or reverse (RMAD) mode automatic differentiation (Griewank and Walther, 2008).

2. *Approximate Implicit Differentiation (AID)* (Pedregosa, 2016; Rajeswaran et al., 2019; Lorraine et al., 2020). First, an (implicit) equation for $\nabla f(\lambda)$ is obtained through the implicit function theorem. Then, this equation is approximately solved by using a two stage scheme. We analyse two specific methods in this class: the *fixed-point method* (Lorraine et al., 2020), also referred to as recurrent backpropagation in the context of recurrent neural networks (Almeida, 1987; Pineda, 1987), and the *conjugate gradient method* (Pedregosa, 2016).

Both schemes can be efficiently implemented using automatic differentiation (Griewank and Walther, 2008; Baydin et al., 2018) achieving similar cost in time, while ITD has usually a larger memory cost than AID[1].

Contributions. Although there is a vast amount of literature on the two hypergradient approximation strategies previously described, it remains unclear whether it is better to use one or the other. In this chapter, we shed some light over this issue both theoretically and experimentally. Specifically our contributions are the following:

- We provide iteration complexity results for ITD and AID when the mapping defining the fixed point equation is a contraction. In particular, we prove non-asymptotic linear rates for the approximation errors of both approaches.

- We make a theoretical and numerical comparison among different ITD and AID strategies considering several experimental scenarios.

[1]This is true when ITD is implemented using RMAD, which is the standard approach when λ is high dimensional.

We note that, to the best of our knowledge, non-asymptoptic rates of convergence for AID were only recently given in the case of meta-learning (Rajeswaran et al., 2019). Furthermore, we are not aware of any previous results about non-asymptotic rates of convergence for ITD.

3.2 Related Work

Iterative differentiation for functions defined implicitly has been extensively studied in the automatic differentiation literature. In particular (Griewank and Walther, 2008, Chap. 15) derives asymptotic linear rates for ITD under the assumption that $\Phi(\cdot, \lambda)$ is a contraction. Another attempt to theoretically analyse ITD is made by Franceschi et al. (2018) in the context of the bilevel problem (3.2). There, the authors provide sufficient conditions for the asymptotic convergence of the set of minimizers of the approximate problem to the set of minimizers of the bilevel problem. In contrast, in this chapter, we give rates for the convergence of the approximate hypergradient $\nabla f_t(\lambda)$ to the true one (i.e. $\nabla f(\lambda)$). ITD is also considered in Shaban et al. (2019) where $\nabla f_t(\lambda)$ is approximated via a procedure which is reminiscent of truncated backpropagation. The authors bound the norm of the difference between $\nabla f_t(\lambda)$ and its truncated version as a function of the truncation steps. This is different from our analysis which directly considers the problem of estimating the gradient of f.

In the case of AID, an asymptotic analysis is presented in Pedregosa (2016), where the author proves the convergence of an inexact gradient projection algorithm for the minimization of the function f defined in problem (3.2), using increasingly accurate estimates of $\nabla f(\lambda)$. Whereas Rajeswaran et al. (2019) present complexity results in the setting of meta-learning with biased regularization. Here, we extend this line of work by providing complexity results for AID in the more general setting of problem (3.1).

We also mention the papers by Amos and Kolter (2017) and Amos (2019), which present techniques to differentiate through the solutions of quadratic and cone programs respectively. Using such techniques allows one to treat these optimization problems as layers of a neural network and to use backpropagation for the end-to-end

training of the resulting learning model. In the former work, the gradient is obtained by implicitly differentiating through the KKT conditions of the lower-level problem, while the latter performs implicit differentiation on the residual map of Minty's parametrization.

A different approach to solve bilevel problems of the form (3.2) is presented by Mehra and Hamm (2019), who consider a sequence of "single level" objectives involving a quadratic regularization term penalizing violations of the lower-level first-order stationary conditions. The authors provide asymptotic convergence guarantees for the method, as the regularization parameter tends to infinity, and show that it outperforms both ITD and AID on different settings where the lower-level problem is non-convex.

All previously mentioned works except Griewank and Walther (2008) consider bilevel problems of the form (3.2). Another exception is Liao et al. (2018), which proposes two improvements to recurrent backpropagation, one based on conjugate gradient on the normal equations, and another based on Neumann series approximation of the inverse.

3.3 Iteration Complexity Analysis

In this section we establish non-asymptotic bounds on the hypergradient (i.e. $\nabla f(\lambda)$) approximation errors for both ITD and AID schemes. In particular, after proving some basic results on the gradient of f and $w(\cdot)$ in problem (3.1), In Section 3.3.1 we address the iteration complexity of ITD, while in Section 3.3.2, after giving a general bound for AID, we focus on two popular implementations of the AID scheme: one based on the conjugate gradient method and the other on the fixed-point method. Major results are proved in Appendix A.1.

Referring to problem (3.1), we group the assumptions as follows. Assumption 3.3.1 is general while 3.3.5 and 3.3.9 are specific to ITD and AID respectively.

All these assumptions are satisfied for several important machine learning settings. In particular, they are satisfied for problems in form (3.2), where Assumption 3.3.1(iv) is satisfied, $\mathcal{L}(\cdot, \lambda)$ is strongly convex and Lipschitz smooth, and

$\nabla_1^2\mathcal{L}(\cdot,\lambda)$, $\nabla_{12}\mathcal{L}(\cdot,\lambda)$, are Lipschitz Continuous. This is true for many problems with smooth objectives at both upper and lower levels and L2 regularization at the lower-level, and in all the experiments in Section 3.4.1. Furthermore, all the assumptions are verified for some of the experiments with equilibrium models in Section 3.4.2.

Assumption 3.3.1. *For every $\lambda \in \Lambda$,*

(i) *$w(\lambda)$ is the unique fixed point of $\Phi(\cdot,\lambda)$.*

(ii) *$I - \partial_1\Phi(w(\lambda),\lambda)$ is invertible.*

(iii) *$\partial_1\Phi(\cdot,\lambda)$, $\partial_2\Phi(\cdot,\lambda)$ are Lipschitz continuous with constants $\nu_{1,\lambda}, \nu_{2,\lambda}$ respectively.*

(iv) *$\nabla_1 E(\cdot,\lambda)$, $\nabla_2 E(\cdot,\lambda)$ are Lipschitz continuous with constants $\mu_{1,\lambda}$, $\mu_{2,\lambda}$ respectively.*

A direct consequence of Assumption 3.3.1(i)-(ii) and of the implicit function theorem is that $w(\cdot)$ and $f(\cdot)$ are differentiable.

Lemma 3.3.2. *(Differentiability of f). Consider problem* (3.1) *and suppose that Assumption 3.3.1(i)-(ii) hold. Then $w(\cdot)$ and $f(\cdot)$ are differentiable on an open set containing Λ and, $\forall \lambda \in \Lambda$*

$$w'(\lambda) = (I - \partial_1\Phi(w(\lambda),\lambda))^{-1}\partial_2\Phi(w(\lambda),\lambda), \tag{3.4}$$

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda),\lambda) + w'(\lambda)^\top \nabla_1 E(w(\lambda),\lambda). \tag{3.5}$$

Proof. The function $G(w,\lambda) := w - \Phi(w,\lambda)$ is continuously differentiable on an open set containing $\mathbb{R}^d \times \Lambda$. Then, we have for any $\lambda \in \Lambda$

$$\partial_1 G(w(\lambda),\lambda) = I - \partial_1\Phi(w(\lambda),\lambda),$$

which is invertible due to Assumption 3.3.1(ii). Thus, since $G(w(\lambda),\lambda) = 0$, the implicit function theorem (Lang, 2012, Theorem 5.9) yields that $w(\lambda)$ is continuously

differentiable with derivative

$$\begin{aligned} w'(\lambda) &= \partial_1 G(w(\lambda),\lambda)^{-1}\partial_2 G(w(\lambda),\lambda) \\ &= (I-\partial_1\Phi(w(\lambda),\lambda))^{-1}\partial_2\Phi(w(\lambda),\lambda). \end{aligned}$$

Finally, (3.4) follows from the chain rule for differentiation. □

Corollary 3.3.3. *Suppose that in problem* (3.2)*, the function* $\mathcal{L}\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}$ *is twice continuously differentiable and strongly convex w.r.t. the first variable. Let* $\alpha\colon \Lambda \to \mathbb{R}_{++}$ *be a differentiable function. Then Lemma 3.3.2 holds and*

$$(\forall \lambda \in \Lambda) \quad w'(\lambda) = -\nabla_1^2\mathcal{L}(w,\lambda)^{-1}\nabla_{21}^2\mathcal{L}(w(\lambda),\lambda).$$

Proof. Define $\Phi(w,\lambda) = w - \alpha(\lambda)\nabla_1\mathcal{L}(w,\lambda)$. Then, Fermat's rule for the lower-level problem in (3.2) yields that $w(\lambda)$ is the fixed point for $\Phi(\cdot,\lambda)$, while $I-\partial_1\Phi(w(\lambda),\lambda)$ is invertible since $I-\partial_1\Phi(w(\lambda),\lambda) = \alpha(\lambda)\nabla_1^2\mathcal{L}(w,\lambda)$ with $\alpha(\lambda)\neq 0$ and $\nabla_1^2\mathcal{L}(w,\lambda)$ positive definite due to the strong convexity of $\mathcal{L}(\cdot,\lambda)$. Therefore, Theorem 3.3.2 applies and, since $\partial_2\Phi(w(\lambda),\lambda) = -\alpha(\lambda)\nabla_{21}^2\mathcal{L}(w(\lambda),\lambda)$, (3.4) yields $w'(\lambda) = -\frac{\alpha(\lambda)}{\alpha(\lambda)}\nabla_1^2\mathcal{L}(w,\lambda)^{-1}\nabla_{21}^2\mathcal{L}(w(\lambda),\lambda)$. □

Before starting with the study of ITD and AID, we give a lemma which introduces three additional constants that will occur in the complexity bounds.

Lemma 3.3.4. *Let* $\lambda \in \Lambda$ *and* $D_\lambda = \|w(\lambda)\|$*. Then there exist* $L_{E,\lambda}, L_{\Phi,\lambda} \in \mathbb{R}_+$ *s.t.*

$$\sup_{\|w\|\leq 2D_\lambda} \|\nabla_1 E(w,\lambda)\| \leq L_{E,\lambda}, \quad \sup_{\|w\|\leq 2D_\lambda} \|\partial_2\Phi(w,\lambda)\| \leq L_{\Phi,\lambda}$$

Proof. The proof follows by noting that, since the closed ball of radius $2D_\lambda$ is compact, its image through the continuous functions $\nabla_1 E(\cdot,\lambda)$ and $\partial_2\Phi(\cdot,\lambda)$ is also compact, and so bounded. □

3.3.1 Iterative Differentiation

In this section we replace $w(\lambda)$ in (3.1) by the t-th iterate of $\Phi(\cdot,\lambda)$, for which we additionally require the following.

Assumption 3.3.5. *For every $\lambda \in \Lambda$, $\Phi(\cdot, \lambda)$ is a contraction with constant $q_\lambda \in (0,1)$.*

The approximation of the hypergradient $\nabla f(\lambda)$ is then obtained as follows.

Algorithm 3.3.1 Iterative Differentiation (ITD)

1. Let $t \in \mathbb{N}$, set $w_0(\lambda) = 0$, and compute,
$$\begin{array}{l} \text{for } i = 1, 2, \dots t \\ \quad w_i(\lambda) = \Phi(w_{i-1}(\lambda), \lambda). \end{array}$$
2. Set $f_t(\lambda) = E(w_t(\lambda), \lambda)$.
3. Compute $\nabla f_t(\lambda)$ using automatic differentiation.

Assumption 3.3.5 looks quite restrictive, however it is satisfied in a number of interesting cases:

(a) In the setting of the bilevel optimization problem (3.2), suppose that for every $\lambda \in \Lambda$, $\mathcal{L}(\cdot, \lambda)$ is $\tau_{\mathcal{L}}(\lambda)$-strongly convex and $L_{\mathcal{L}}(\lambda)$-Lipschitz smooth for some continuously differentiable functions $\tau_{\mathcal{L}} \colon \Lambda \to \mathbb{R}_{++}$, and $L_{\mathcal{L}} \colon \Lambda \to \mathbb{R}_{++}$. Set $\kappa(\lambda) = L_{\mathcal{L}}(\lambda)/\tau_{\mathcal{L}}(\lambda)$,
$$\alpha(\lambda) = \frac{2}{\tau_{\mathcal{L}}(\lambda) + L_{\mathcal{L}}(\lambda)}, \text{ and } q_\lambda = \frac{\kappa(\lambda) - 1}{\kappa(\lambda) + 1}. \tag{3.6}$$
Then, $\Phi(w, \lambda) = w - \alpha(\lambda)\nabla_1 \mathcal{L}(w, \lambda)$ is a contraction w.r.t. w with constant q_λ (see Appendix A.2).

(b) For strongly convex quadratic functions, accelerated methods like Nesterov's (Nesterov, 1983) or heavy-ball (Polyak, 1987) can be formulated as fixed-point iterations of a contraction in the norm defined by a suitable positive definite matrix.

(c) In certain graph and recurrent neural networks of the form (3.3), where the transition function is assumed to be a contraction (Scarselli et al., 2008; Almeida, 1987; Pineda, 1987).

The following lemma is a simple consequence of the theory on Neumann series and shows that Assumption 3.3.5 is stronger than Assumption 3.3.1(i)-(ii).

Lemma 3.3.6. *Let Assumption 3.3.5 be satisfied. Then, for every $\lambda \in \Lambda$, $\Phi(\cdot, \lambda)$ has a unique fixed point and, for every $w \in \mathbb{R}^d$, $I - \partial_1 \Phi(w, \lambda)$ is invertible and*

$$\|(I - \partial_1 \Phi(w, \lambda))^{-1}\| \leq \frac{1}{1 - q_\lambda}.$$

In particular, (i) and (ii) in Assumption 3.3.1 hold.

Proof. Let $\lambda \in \Lambda$ and $w \in \mathbb{R}^d$. Since $\Phi(\cdot, \lambda)$ is Lipschitz continuous with constant q_λ, it follows that

$$\|\partial_1 \Phi(w, \lambda)\| \leq q_\lambda < 1.$$

Therefore,

$$\sum_{k=0}^{\infty} \|\partial_1 \Phi(w, \lambda)\|^k \leq \sum_{k=0}^{\infty} q_\lambda^k = \frac{1}{1 - q_\lambda}.$$

Thus, $I - \partial_1 \Phi(w, \lambda)$ is invertible, $\sum_{k=0}^{\infty} \partial_1 \Phi(w, \lambda)^k = (I - \partial_1 \Phi(w, \lambda))^{-1}$ and the bound follows. □

With Assumption 3.3.5 in force and if $w_t(\lambda)$ is defined as at point 1 in Algorithm 3.3.1, we have the following proposition that is essential for the final bound. The proof is given in Appendix A.1.

Proposition 3.3.7. *Suppose that Assumptions 3.3.1(iii) and 3.3.5 hold and let $t \in \mathbb{N}$, with $t \geq 1$. Moreover, for every $\lambda \in \Lambda$, let $w_t(\lambda)$ be computed by Algorithm 3.3.1 and let D_λ and $L_{\Phi,\lambda}$ be as in Lemma 3.3.4. Then, $w_t(\cdot)$ is differentiable and, for every $\lambda \in \Lambda$,*

$$\|w_t'(\lambda) - w'(\lambda)\| \leq \left(\nu_{2,\lambda} + \nu_{1,\lambda} \frac{L_{\Phi,\lambda}}{1 - q_\lambda} \right) D_\lambda t q_\lambda^{t-1} + \frac{L_{\Phi,\lambda}}{1 - q_\lambda} q_\lambda^t. \tag{3.7}$$

Leveraging Proposition 3.3.7, we give the main result of this section, which is also proven in Appendix A.1.

Theorem 3.3.8. *(ITD bound) Suppose that Assumptions 3.3.1(iii)-(iv) and 3.3.5 hold and let $t \in \mathbb{N}$ with $t \geq 1$. Moreover, for every $\lambda \in \Lambda$, let $w_t(\lambda)$ and f_t be defined according to Algorithm 3.3.1 and let $D_\lambda, L_{E,\lambda}$, and $L_{\Phi,\lambda}$ be as in Lemma 3.3.4. Then,*

f_t is differentiable and, for every $\lambda \in \Lambda$,

$$\|\nabla f_t(\lambda) - \nabla f(\lambda)\| \leq \Big(c_1(\lambda) + c_2(\lambda)\frac{t}{q_\lambda} + c_3(\lambda)\Big) q_\lambda^t, \tag{3.8}$$

where

$$\begin{aligned} c_1(\lambda) &= \left(\mu_{2,\lambda} + \frac{\mu_{1,\lambda} L_{\Phi,\lambda}}{1-q_\lambda}\right) D_\lambda, \\ c_2(\lambda) &= \left(\nu_{2,\lambda} + \frac{\nu_{1,\lambda} L_{\Phi,\lambda}}{1-q_\lambda}\right) L_{E,\lambda}\, D_\lambda, \\ c_3(\lambda) &= \frac{L_{E,\lambda}\, L_{\Phi,\lambda}}{1-q_\lambda}. \end{aligned}$$

In this generality this is a new result that provides a non-asymptotic linear rate of convergence for the gradient of f_t towards that of f.

3.3.2 Approximate Implicit Differentiation

In this section we study another approach to approximate the gradient of f. We derive from (3.4) and (3.5) that

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda), \lambda) + \partial_2 \Phi(w(\lambda), \lambda)^\top v(\lambda) \tag{3.9}$$

where $v(\lambda)$ is the solution of the linear system

$$(I - \partial_1 \Phi(w(\lambda), \lambda)^\top) v = \nabla_1 E(w(\lambda), \lambda). \tag{3.10}$$

However, in the above formulas $w(\lambda)$ is usually not known explicitly or is expensive to compute exactly. To solve this issue $\nabla f(\lambda)$ is estimated as in Algorithm 3.3.2.

Note that, unlike ITD, this procedure is agnostic about the algorithms used to compute the sequences $w_t(\lambda)$ and $v_k(\lambda)$. Interestingly, in the context of problem (3.2), choosing $\Phi(w, \lambda) = w - \nabla_1 \mathcal{L}(w, \lambda)$ in Algorithm 3.3.2 yields the same procedure studied by Pedregosa (2016).

The number of iterations t and k in Algorithm 3.3.2 give a direct way of trading off accuracy and speed. To quantify this trade off we consider the following

Algorithm 3.3.2 Approximate Implicit Differentiation (AID)

1. Let $t \in \mathbb{N}$ and compute $w_t(\lambda)$ by t steps of an algorithm converging to $w(\lambda)$, starting from $w_0(\lambda) = 0$.
2. Compute $v_k(\lambda)$ after k steps of a solver for the system
$$(I - \partial_1\Phi(w_t(\lambda), \lambda)^\top)v = \nabla_1 E(w_t(\lambda), \lambda), \tag{3.11}$$
with solution $\hat{v}(\lambda)$.
3. Compute the approximate gradient as
$$\hat{\nabla} f(\lambda) := \nabla_2 E(w_t(\lambda), \lambda) + \partial_2\Phi(w_t(\lambda), \lambda)^\top v_k(\lambda).$$

assumptions.

Assumption 3.3.9. *Let $w_t(\lambda), v_k(\lambda), \hat{v}(\lambda)$ be as in Algorithm 3.3.2. For every $\lambda \in \Lambda$*

(i) $\forall w \in \mathbb{R}^d$, $I - \partial_1\Phi(w, \lambda)$ *is invertible.*

(ii) $\|w_t(\lambda) - w(\lambda)\| \leq \rho_\lambda(t)\|w(\lambda)\|$, $\rho_\lambda(t) \leq 1$, *and* $\rho_\lambda(t) \to 0$ *as* $t \to +\infty$.

(iii) $\|v_k(\lambda) - \hat{v}(\lambda)\| \leq \sigma_\lambda(k)\|\hat{v}(\lambda)\|$ *and* $\sigma_\lambda(k) \to 0$ *as* $k \to +\infty$.

If Assumption 3.3.9(i) holds, then, for every $\lambda \in \Lambda$, since the map $w \mapsto \|(I - \partial_1\Phi(w, \lambda))^{-1}\|$ is continuous, we have

$$\sup_{\|w\| \leq 2D_\lambda} \|(I - \partial_1\Phi(w, \lambda))^{-1}\| \leq \frac{1}{\hat{\mu}_\lambda} < +\infty, \tag{3.12}$$

for some $\hat{\mu}_\lambda > 0$. We note that, in view of Lemma 3.3.6, Assumption 3.3.5 implies Assumption 3.3.9(i) (which in turn implies Assumption 3.3.1(ii)) and in (3.12) one can take $\hat{\mu}_\lambda = 1 - q_\lambda$. We stress that, Assumption 3.3.9(ii)-(iii) are general and do not specify the type of algorithms solving the fixed-point equation $w = \Phi(w, \lambda)$ and the liner system (3.11). It is only required that such algorithms have explicit rates of convergence $\rho_\lambda(t)$ and $\sigma_\lambda(k)$ respectively. Finally, we note that Assumption 3.3.9(ii) is less restrictive than Assumption 3.3.5 and encompasses the procedure at point 1 in Algorithm 3.3.1: indeed in such case 3.3.9(ii) holds with $\rho_\lambda(t) = q_\lambda^t$.

It is also worth noting that the AID procedure requires only to store the last lower-level iterate, i.e. $w_t(\lambda)$. This is a considerable advantage over ITD, which

instead requires to store the entire lower-level optimization trajectory $(w_i(\lambda))_{0\leq i\leq t}$, if implemented using RMAD.

The iteration complexity bound for AID is given below. This is a general bound which depends on the rate of convergence $\rho_\lambda(t)$ of the sequence $(w_t(\lambda))_{t\in\mathbb{N}}$ and the rate of convergence $\sigma_\lambda(k)$ of the sequence $(v_k(\lambda))_{k\in\mathbb{N}}$.

Theorem 3.3.10. *(AID bound) Suppose that Assumptions 3.3.1(i)(iii)(iv) and 3.3.9(i)–(iii) hold. Let $\lambda \in \Lambda$, $t \in \mathbb{N}$, $k \in \mathbb{N}$. Let $D_\lambda, L_{E,\lambda}$, and $L_{\Phi,\lambda}$ be as in Lemma 3.3.4 and let $\hat{\mu}_\lambda$ be defined according to* (3.12)*. Let $\hat{\nabla} f(\lambda)$ be defined as in Algorithm 3.3.2 and let $\hat{\Delta} = \|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|$. Then,*

$$\hat{\Delta} \leq \left(\mu_{2,\lambda} + \frac{\mu_{1,\lambda} L_{\Phi,\lambda}}{\hat{\mu}_\lambda} + \frac{\nu_{2,\lambda} L_{E,\lambda}}{\hat{\mu}_\lambda} + \frac{\nu_{1,\lambda} L_{\Phi,\lambda} L_{E,\lambda}}{\hat{\mu}_\lambda^2}\right) D_\lambda \rho_\lambda(t) + \frac{L_{\Phi,\lambda} L_{E,\lambda}}{\hat{\mu}_\lambda} \sigma_\lambda(k). \tag{3.13}$$

Furthermore, if Assumption 3.3.5 holds, then $\hat{\mu}_\lambda = 1 - q_\lambda$ and

$$\hat{\Delta} \leq \left(c_1(\lambda) + \frac{c_2(\lambda)}{1-q_\lambda}\right)\rho_\lambda(t) + c_3(\lambda)\sigma_\lambda(k), \tag{3.14}$$

where $c_1(\lambda)$, $c_2(\lambda)$ and $c_3(\lambda)$ are defined in Theorem 3.3.8.

Theorem 3.3.10 provides a non-asymptotic rate of convergence for $\hat{\nabla} f$ which contrasts with the asymptotic result given in Pedregosa (2016). In this respect, making Assumption 3.3.9(i) instead of the weaker Assumption 3.3.1(ii) is critical.

Depending on the choice of the solver for the linear system (3.11) different AID methods are obtained. In the following we consider two cases.

AID with the Conjugate Gradient Method (AID-CG). For the sake of brevity we set $\hat{A}_\lambda = I - \partial_1 \Phi(w_t(\lambda), \lambda)^\top$ and $\hat{b}_\lambda = \nabla_1 E(w_t(\lambda), \lambda)$. Then, the linear system (3.11) is equivalent to the following minimization problem

$$\min_{v\in\mathbb{R}^d} \frac{1}{2}\|\hat{A}_\lambda v - \hat{b}_\lambda\|^2, \tag{3.15}$$

which, if $\partial_1\Phi(w_t(\lambda),\lambda)$ is symmetric (so that $\hat{A}_\lambda$ is also symmetric) is in turn equivalent to

$$\min_{v\in\mathbb{R}^d} \frac{1}{2} v^\top \hat{A}_\lambda v - v^\top \hat{b}_\lambda. \tag{3.16}$$

Several first order methods solving problems (3.15) or (3.16) satisfy assumption 3.3.9(iii) with linear rates and require only Jacobian-vector products. In particular, for the symmetric case (3.16), the *conjugate gradient* method features the following linear rate

$$\|v_k(\lambda) - v_t(\lambda)\| \leq 2\sqrt{\kappa(\hat{A}_\lambda)} \left(\frac{\sqrt{\kappa(\hat{A}_\lambda)}-1}{\sqrt{\kappa(\hat{A}_\lambda)}+1} \right)^k \|v_0(\lambda) - v_t(\lambda)\|, \tag{3.17}$$

where $\kappa(\hat{A}_\lambda)$ is the condition number of $\hat{A}_\lambda$. In the setting of case (a) outlined in Section 3.3.1, $\hat{A}_\lambda = \alpha(\lambda)\nabla_1^2\mathcal{L}(w_t(\lambda),\lambda)$ and

$$\tau_{\mathcal{L}}(\lambda) I \preccurlyeq \nabla_1^2\mathcal{L}(w_t(\lambda),\lambda) \preccurlyeq L_{\mathcal{L}}(\lambda) I.$$

Therefore, the condition number of $\hat{A}_\lambda$ satisfies $\kappa(\hat{A}_\lambda) \leq L_{\mathcal{L}}(\lambda)/\tau_{\mathcal{L}}(\lambda) = \kappa(\lambda)$ and hence

$$\frac{\sqrt{\kappa(\hat{A}_\lambda)}-1}{\sqrt{\kappa(\hat{A}_\lambda)}+1} \leq \frac{\sqrt{\kappa(\lambda)}-1}{\sqrt{\kappa(\lambda)}+1} \leq \frac{\kappa(\lambda)-1}{\kappa(\lambda)+1} = q_\lambda. \tag{3.18}$$

AID with the Fixed-Point Method (AID-FP). In this paragraph we make a specific choice for the sequence $(v_k(\lambda))_{k\in\mathbb{N}}$ in Assumption 3.3.9(iii). We let Assumption 3.3.5 be satisfied and consider the following algorithm. For every $\lambda \in \Lambda$ and $t \in \mathbb{N}$, we choose $v_0(\lambda) = 0 \in \mathbb{R}^d$ and,

$$\begin{aligned} &\text{for } k = 1,2,\dots \\ &\quad v_k(\lambda) = \partial_1\Phi(w_t(\lambda),\lambda)^\top v_{k-1}(\lambda) + \nabla_1 E(w_t(\lambda),\lambda). \end{aligned} \tag{3.19}$$

In such case the rate of convergence $\sigma_\lambda(k)$ is linear. More precisely, since

$\|\partial_1\Phi(w_t(\lambda),\lambda)\| \leq q_\lambda < 1$ (from Assumption 3.3.5), then the mapping

$$T\colon v \mapsto \partial_1\Phi(w_t(\lambda),\lambda)v + \nabla_1 E(w_t(\lambda),\lambda)$$

is a contraction with constant q_λ. Moreover, the fixed-point of T is the solution of (3.11). Therefore, $\|v_k(\lambda) - \hat{v}(\lambda)\| \leq q_\lambda^k \|v_0(\lambda) - \hat{v}(\lambda)\|$. In the end the following result holds.

Theorem 3.3.11. *If Assumption 3.3.5 holds and $(v_k(\lambda))_{k\in\mathbb{N}}$ is defined according to* (3.19)*, then Assumption 3.3.9(iii) is satisfied with $\sigma_\lambda(k) = q_\lambda^k$.*

Now, plugging the rate $\sigma_\lambda(k) = q_\lambda^k$ into the general bound (3.14) yields

$$\hat{\Delta} \leq \Big(c_1(\lambda) + \frac{c_2(\lambda)}{1-q_\lambda}\Big)\rho_\lambda(t) + c_3(\lambda)q_\lambda^k. \tag{3.20}$$

However, an analysis similar to the one in Section 3.3.1 shows that the above result can be slightly improved as follows.

Theorem 3.3.12. *(AID-FP bound) Suppose that Assumptions 3.3.1(i)(iii)(iv) and Assumption 3.3.5 hold. Suppose also that* (3.19) *holds. Let $\hat{\nabla} f(\lambda)$ be defined according to Algorithm 3.3.2 and $\hat{\Delta} = \|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|$. Then, for every $t,k \in \mathbb{N}$,*

$$\hat{\Delta} \leq \Big(c_1(\lambda) + c_2(\lambda)\frac{1-q_\lambda^k}{1-q_\lambda}\Big)\rho_\lambda(t) + c_3(\lambda)q_\lambda^k, \tag{3.21}$$

where $c_1(\lambda)$, $c_2(\lambda)$ and $c_3(\lambda)$ are given in Theorem 3.3.8.

We end this section with a discussion about the consequences of the presented results.

3.3.3 Remarks

Theorem 3.3.10 shows that Algorithm 3.3.2 computes an approximate gradient of f with a linear convergence rate (in t and k), provided that the solvers for the lower-level problem and the linear system converge linearly. Furthermore, under Assumption 3.3.5, both AID-FP and ITD converge linearly. However, if in Algorithm 3.3.2 we define $w_t(\lambda)$ as at point 1 in Algorithm 3.3.1 (so that $\rho_\lambda(t) = q_\lambda^t$),

and take $k = t$, then the bound for AID-FP (3.21) is lower than that of ITD (3.8), since $q_\lambda(1-q_\lambda^t)/(1-q_\lambda) = \sum_{i=1}^t q_\lambda^i < t$ for every $t \geq 1$. This suggests that AID-FP converges faster than ITD.

We now discuss the choice of the algorithm to solve the linear system (3.11) in Algorithm 3.3.2. Theorem 3.3.12 provides a bound for AID-FP, which considers procedure (3.19). However, we see from (3.14) in Theorem 3.3.10 that a solver for the linear system with rate of convergence $\sigma_\lambda(k)$ faster than q_λ^k may give a better bound. The above discussion, together with (3.17) and (3.18), proves that AID-CG has a better asymptotic rate than AID-FP for instances of problem (3.2) where the lower-level objective $\mathcal{L}(\cdot, \lambda)$ is Lipschitz smooth and strongly convex (case (a) outlined in Section 3.3.1).

Finally, we note that both ITD and AID consider the initialization $w_0(\lambda) = 0$. However, in a gradient-based bilevel optimization algorithm, it might be more convenient to use a warm start strategy where $w_0(\lambda)$ is set based on previous upper-level iterations. Our analysis can be applied also in this case, but the related upper bounds will depend on the upper-level dynamics. This aspect makes it difficult to theoretically analyse the benefit of a warm start strategy, as noted in Chapter 5.

3.4 Experiments

In the first part of this section we focus on the hypergradient approximation error and show that the upper bounds presented in the previous section give a good estimate of the actual convergence behavior of ITD and AID strategies on a variety of settings. In the second part we present a series of experiments pertaining optimization on both the settings of hyperparameter optimization, as in problem (3.2), and learning equilibrium models, as in problem (3.3). The algorithms have been implemented[2] in PyTorch (Paszke et al., 2019a). In the following, we shorthand AID-FP and AID-CG with FP and CG, respectively.

[2]The code is freely available at the following link.
`https://github.com/prolearner/hypertorch`

3.4.1 Hypergradient Approximation

In this section, we consider several problems of type (3.2) with synthetic generated data (see Appendix A.3.1 for more details) where $D = (X, y)$ and $D' = (X', y')$ are the training and validation sets respectively, with $X \in \mathbb{R}^{n_e \times p}$, $X' \in \mathbb{R}^{n'_e \times p}$, being n_e, n'_e the number of examples in each set and p the number of features. Specifically we consider the following settings, which are representative instances of relevant bilevel problems in machine learning.

Logistic Regression with $\mathcal{L}_2$ Regularization (LR). This setting is similar to the one in Pedregosa (2016), but we introduce multiple regularization parameters:

$$
\begin{aligned}
f(\lambda) &= \sum_{(x,y)\in D'} \psi(yx^\top w(\lambda)), \\
w(\lambda) &= \operatorname*{argmin}_{w\in\mathbb{R}^p} \sum_{(x,y)\in D} \psi(yx^\top w) + \frac{1}{2} w^\top \mathrm{d}iag(\lambda) w,
\end{aligned}
$$

where $\lambda \in \mathbb{R}^p_{++}$, $\psi(x) = \log(1 + e^{-x})$ and $\mathrm{d}iag(\lambda)$ is the diagonal matrix formed by the elements of λ.

Kernel Ridge Regression (KRR). We extend the setting presented by Pedregosa (2016) considering a p-dimensional Gaussian kernel parameter γ in place of the usual one:

$$
\begin{aligned}
f(\beta, \gamma) &= \frac{1}{2} \|y' - K'(\gamma) w(\beta, \gamma)\|^2, \\
w(\beta, \gamma) &= \operatorname*{argmin}_{w\in\mathbb{R}^{n_e}} \frac{1}{2} w^\top \left(K(\gamma) + \beta I\right) w - w^\top y,
\end{aligned}
\tag{3.22}
$$

where $\beta \in (0, \infty)$, $\gamma \in \mathbb{R}^p_{++}$ and $K'(\gamma)$, $K(\gamma)$ are respectively the validation and training kernel matrices (see Appendix A.3.1).

Biased Regularization (BR). Inspired by Denevi et al. (2019a); Rajeswaran et al. (2019), we consider the following.

$$
\begin{aligned}
f(\lambda) &= \frac{1}{2} \|X' w(\lambda) - y'\|^2, \\
w(\lambda) &= \operatorname*{argmin}_{w\in\mathbb{R}^p} \frac{1}{2} \|Xw - y\|^2 + \frac{\beta}{2} \|w - \lambda\|^2,
\end{aligned}
$$

where $\beta \in \mathbb{R}_{++}$ and $\lambda \in \mathbb{R}^p$.

Hyper-representation (HR). The last setting, reminiscent of (Franceschi et al., 2018; Bertinetto et al., 2019), concerns learning a (common) linear transformation of the data and is formulated as

$$f(H) = \frac{1}{2}\|X'Hw(H) - y'\|^2$$
$$w(H) = \operatorname*{argmin}_{w\in\mathbb{R}^d} \frac{1}{2}\|XHw - y\|^2 + \frac{\beta}{2}\|w\|^2$$

where $H \in \mathbb{R}^{p\times d}$ and $\beta \in \mathbb{R}_{++}$.

LR and KRR are high dimensional extensions of classical hyperparameter optimization problems, while BR and HR, are typically encountered in multi-task/meta-learning as single task objectives[3]. Note that Assumption 3.3.5 (i.e. $\Phi(\cdot,\lambda)$ is a contraction) can be satisfied for each of the aforesaid scenarios, since they all belong to case (a) of Section 3.3.1 (KRR, BR and HR also to case (b)).

We solve the lower-level problem in the same way for both ITD and AID methods. In particular, in LR we use the gradient descent method with optimal step size as in case (a) of Section 3.3.1, while for the other cases we use the heavy-ball method with optimal step size and momentum constants. Note that this last method is not a contraction in the original norm, but only in a suitable norm depending on the lower-level problem itself. To compute the exact hypergradient, we differentiate $f(\lambda)$ directly using RMAD for KRR, BR and HR, where the closed form expression for $w(\lambda)$ is available, while for LR we use CG with $t = k = 2000$ in place of the (unavailable) analytic gradient.

Figure 3.1 shows how the approximation error is affected by the number of lower-level iterations t. As suggested by the iteration complexity bounds in Section 3.3, all the approximations, after a certain number of iterations, converge linearly to the true hypergradient[4]. Furthermore, in line with our analysis (see Section 3.3.3), CG gives the best gradient estimate (on average), followed by FP, while ITD performs the worst. For HR, the error of all the methods increases significantly at the

[3] In multi-task/meta-learning the upper-level objectives are averaged over multiple tasks and the hypergradient is simply the average of the single task one.

[4] The asymptotic error can be quite large probably due to numerical errors (more details in Appendix A.3).

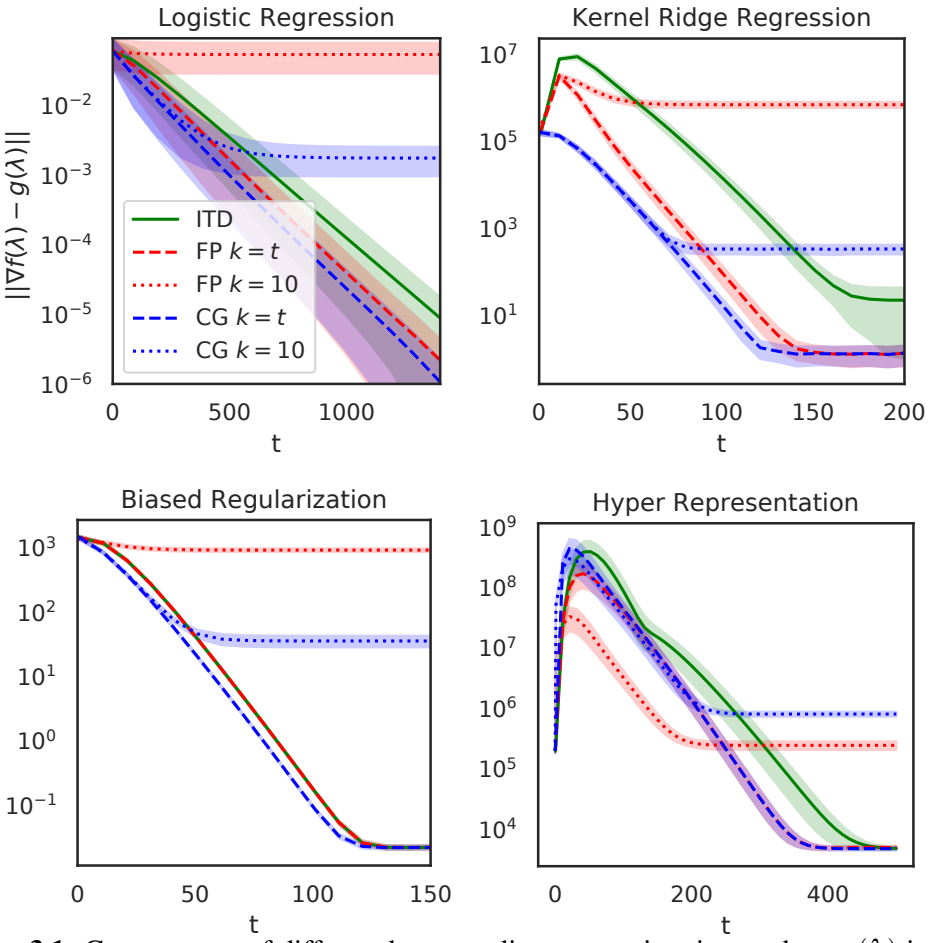

Figure 3.1: Convergence of different hypergradient approximations, where $g(\lambda)$ is equal to $\nabla f_t(\lambda)$ for ITD and to $\hat{\nabla} f(\lambda)$ for CG and FP. Mean and standard deviation (shaded areas) are computed over 20 values of λ sampled uniformly from $[\lambda_{\min}, \lambda_{\max}]^n$.

beginning, which can be explained by the fact that the heavy ball method is not a contraction in the original norm and may diverge at first. CG $k = 10$ outperforms FP $k = 10$ on 3 out of 4 settings, but both remain far from convergence.

3.4.2 Bilevel Optimization

In this section, we aim to solve instances of the bilevel problem (3.1) in which λ has a high dimensionality.

Kernel Ridge Regression on Parkinson. We take $f(\beta, \gamma)$ as defined in problem (3.22) where the data is taken from the UCI Parkinson dataset (Little et al., 2008), containing 195 biomedical voice measurements (22 features) from people with Parkinson's disease. To avoid projections, we replace β and γ respectively

with $\exp(\beta)$ and $\exp(\gamma)$ in the RHS of the two equations in (3.22). We split the data randomly into three equal parts to make the train, validation and test sets.

Logistic Regression on 20 Newsgroups[5]**.** This dataset contains 18000 news divided in 20 topics and the features consist in 101631 tf-idf sparse vectors. We split the data randomly into three equal parts for training, validation and testing. We aim to solve the bilevel problem

$$\begin{aligned} &\min_{\lambda \in \mathbb{R}^p} \mathrm{CE}(X' w(\lambda), y') \\ &w(\lambda) = \operatorname*{argmin}_{w \in \mathbb{R}^{p \times c}} \mathrm{CE}(Xw, y) + \frac{1}{2cp} \sum_{i=1}^{c} \sum_{j=1}^{p} \exp(\lambda_j) w_{ij}^2 \end{aligned}$$

where CE is the average cross-entropy loss, $c = 20$ and $p = 101631$. To improve the performance, we use warm-starts on the lower-level problem, i.e. we take $w_0(\lambda_i) = w_t(\lambda_{i-1})$ for all methods, where $(\lambda_i)_{i=1}^s$ are the upper-level iterates.

Training Data Optimization on Fashion MNIST. Similarly to Maclaurin et al. (2015), we optimize the features of a set of 10 training points, each with a different class label on the Fashion MNIST dataset (Xiao et al., 2017). More specifically we define the bilevel problem as

$$\begin{aligned} &\min_{X \in \mathbb{R}^{c \times p}} \mathrm{CE}(X' w(X), y') \\ &w(X) = \arg \min_{w \in \mathbb{R}^{p \times c}} \mathrm{CE}(Xw, y) + \frac{\beta}{2cp} \|w\|^2 \end{aligned}$$

where $\beta = 1$, $c = 10$, $p = 784$, $y = (0, \ldots, c)^\top$ and (X', y') contains the training set.

We solve each problem using (hyper)gradient descent with fixed step size selected via grid search (additional details are provided in Appendix A.3.2). The results in Table 3.1 show the upper-level objective and test accuracy both computed on the approximate lower-level solution $w_t(\lambda)$ after bilevel optimization[6].

For Parkinson and Fashion MNIST, there is little difference among the methods for a fixed t. For 20 newsgroup, CG $k = t$ reaches the lowest objective value, followed

[5] http://qwone.com/ jason/20Newsgroups/

[6] For completeness, we also report in the Appendix (Table A.1) the upper-level objective and test accuracy both computed on the exact lower-level solution $w(\lambda)$.

Table 3.1: Objective (test accuracy) values after s gradient descent steps where s is 1000, 500 and 4000 for Parkinson, 20 newsgroup and Fashion MNIST respectively. Test accuracy values are in %. $k_r = 10$ for Parkinson and 20 newsgroup while for Fashion MNIST $k_r = 5$.

	Parkinson		**20 newsgroup**		
	$t = 100$	$t = 150$	$t = 10$	$t = 25$	$t = 50$
ITD	2.39 (75.8)	2.11 (69.7)	1.08 (61.3)	0.97 (62.8)	0.89 (64.2)
FP $k = t$	2.37 (81.8)	2.20 (77.3)	1.03 (62.1)	1.02 (62.3)	0.84 (64.4)
CG $k = t$	2.37 (78.8)	2.20 (77.3)	0.93 (63.7)	0.78 (63.3)	0.64 (63.1)
FP $k = k_r$	2.71 (80.3)	2.60 (78.8)	–	0.94 (63.6)	0.97 (63.0)
CG $k = k_r$	2.33 (77.3)	2.02 (77.3)	–	0.82 (64.3)	0.75 (64.2)

	Fashion MNIST	
	$t = 5$	$t = 10$
ITD	0.41 (84.1)	0.43 (83.8)
FP $k = t$	0.41 (84.1)	0.43 (83.8)
CG $k = t$	0.42 (83.9)	0.42 (84.0)
FP $k = k_r$	–	0.42 (83.9)
CG $k = k_r$	–	0.42 (84.0)

by CG $k = 10$. We recall that for ITD we have cost in memory which is linear in t and that, in the case of 20 newsgroups for some t between 50 and 100, this cost exceeded the 11GB on the GPU. AID methods instead, require little memory and, by setting $k < t$, yield similar or even better performance at a lower computation time. Finally, we stress that since the upper-level objective is non-convex, possibly with several minima, gradient descent with a more precise estimate of the hypergradient may get more easily trapped in a bad local minimum.

Equilibrium Models. This experiment investigates the behavior of the hypergradient approximation methods on a simple instance of EQM (see problem (3.3)) on non-structured data. EQM are an attractive class of models due to their mathematical simplicity, enhanced interpretability and memory efficiency. A number of works (Miller and Hardt, 2019; Bai et al., 2019) have recently shown that EQMs can perform on par with standard deep nets on a variety of complex tasks, renewing the interest in these kinds of models.

We use a subset of $n_e = 5000$ instances randomly sampled from the MNIST dataset as training data and employ a multiclass logistic classifier paired with a

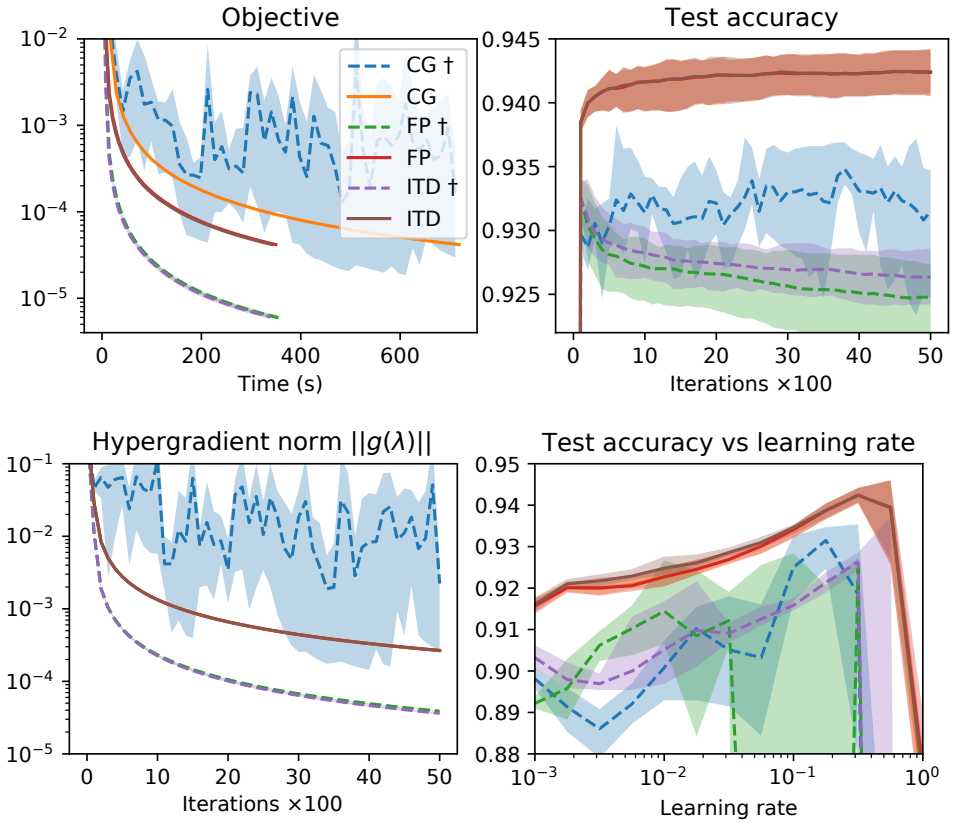

Figure 3.2: Experiments on EQM problems. Mean (solid or dashed lines) and point-wise minimum-maximum range (shaded regions) across 5 random seeds that only control the initialization of λ. The estimated hypergradient $g(\lambda)$ is equal to $\nabla f_t(\lambda)$ for ITD and $\hat{\nabla} f(\lambda)$ for AID. We used $t = k = 20$ for all methods and Nesterov momentum for optimizing λ, applying a projection operator at each iteration except for the methods marked with †. When performing projection, the curves produced by the three approximation schemes mostly overlap, indicating essentially the same performance (although at a different computational cost).

cross-entropy loss. We picked a small training set and purposefully avoided stochastic optimization methods to better focus on issues related to the computation of the hypergradients itself, avoiding the introduction of other sources of noise. We parametrize ϕ_i as

$$\phi_i(w_i, \gamma) = \tanh(Aw_i + Bx_i + c) \quad \text{for } 1 \leq i \leq n_e \tag{3.23}$$

where $x_i \in \mathbb{R}^p$ is the i-th example, $w_i \in \mathbb{R}^h$ and $\gamma = (A, B, C) \in \mathbb{R}^{h \times h} \times \mathbb{R}^{h \times p} \times \mathbb{R}^h$. Such a model may be viewed as a (infinite layers) feed-forward neural network with tied weights or as a recurrent neural network with static inputs. Imposing $\|A\| < 1$

ensures that the transition functions (3.23), and hence Φ, are contractions. This can be achieved during optimization by projecting the singular values of A onto the interval $[0, 1-\varepsilon]$ for $\varepsilon > 0$. We note that regularizing the norm of $\partial_1 \phi_i$ or adding L_1 or L_∞ penalty terms on A may encourage, but does not strictly enforce, $\|A\| < 1$.

We conducted a series of experiments to ascertain the importance of the contractiveness of the map Φ, as well as to understand which of the analysed methods is to be preferred in this setting. Since here $\partial_1 \Phi$ is not symmetric, the conjugate gradient method must be applied on the normal equations of problem (3.15). We set $h = 200$ and use $t = 20$ fixed-point iterations to solve the lower-level problem in all the experiments. The first three plots of Figure 3.2 report training objectives, test accuracies and norms of the estimated hypergradient for each of the three methods, either applying or not the constraint on A, while the last explores the sensitivity of the methods to the choice of the learning rate. Unconstrained runs are marked with †. Referring to the bottom right plot, it is clear (large shaded regions) that not constraining the spectral norm results in unstable behavior of the "memory-less" AID methods (green and blue lines) for all but a few learning rates, while ITD (violet), as expected, suffers comparatively less. On the contrary, when $\|A\| < 1$ is enforced, all the approximation methods are successful and stable, with FP to be preferred being faster than CG on the normal equations and requiring substantially less memory than ITD. As a side note, referring to the two plots at the top of Figure 3.2, we observe that projecting onto the spectral ball acts as powerful regularizer, in line with the findings of Sedghi et al. (2019).

Equilibrium Models with Convolutions. At last, we report a series of experiments on equilibrium models quite similar to those of the last paragraph, but with convolutional and max-pooling operators in place of the affinities of Equation (3.23). In particular, we model the learnable dynamics with parameters $\gamma = (K, K', c)$ as

$$\phi_i(w_i, \gamma) = \tanh\left(K \star w_i + \mu_{2\times 2}(K' \star x_i) + c\right) \tag{3.24}$$

where $w_i \in \mathbb{R}^{h\times 14\times 14}$ are the state feature maps, $\star$ denotes multichannel bi-dimensional cross-correlation, K and K' contain h 3×3 convolutional kernels each

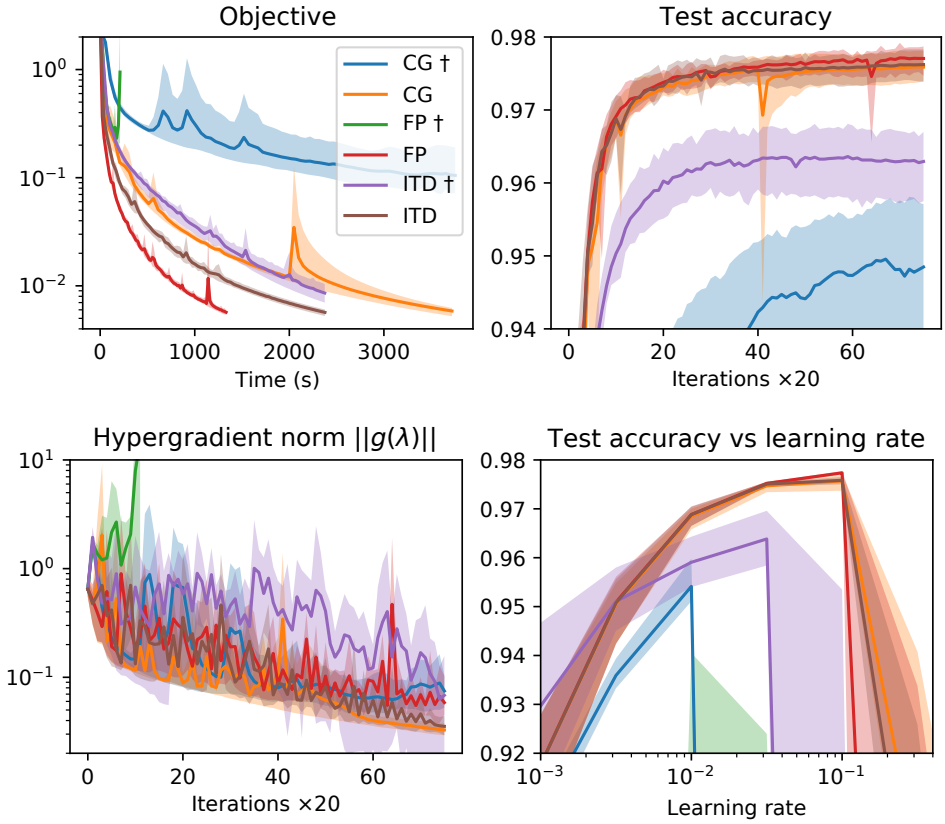

Figure 3.3: Experiments with convolutional EQMs. Mean (solid line) and point-wise minimum-maximum range (shaded region) across 5 random seeds. The seed only controls the initialization of λ. The estimated hypergradient $g(\lambda)$ is equal to $\nabla f_t(\lambda)$ for ITD and $\hat{\nabla} f(\lambda)$ for AID. We used $t = k = 20$ for all methods and Nesterov momentum (1500 iterations) for optimizing λ, applying a projector operator at each iteration except for the methods marked with †. Note that in the first three plots the step-size for the unconstrained experiments is smaller, to prevent divergence.

and $\mu_{2\times 2}$ denotes the max-pooling operator with a 2×2 field and stride of 2. The state feature maps are passed through a max-pooling operator before being flattened and fed to a multiclass logistic classifier. We set $h = 10$ for all the experiments. We use the results and the code of Sedghi et al. (2019) to efficiently perform the projection of the linear operator associated to K into the unit spectral ball[7]. Data and optimization method for the upper objective are the same as the previous experiments.

The results, reported in Figure 3.3, show similar behaviors of those of the previous experiments with non-convolutional fixed point map, albeit with larger differences among the methods, especially for the experiments without projection

[7]Specifically, we project onto $\|\mathrm{c}(K)\| \leq 0.999$, where $\mathrm{c}(K)$ is an $h \times h$ matrix of doubly block circulant matrices; see Sedghi et al. (2019) for details.

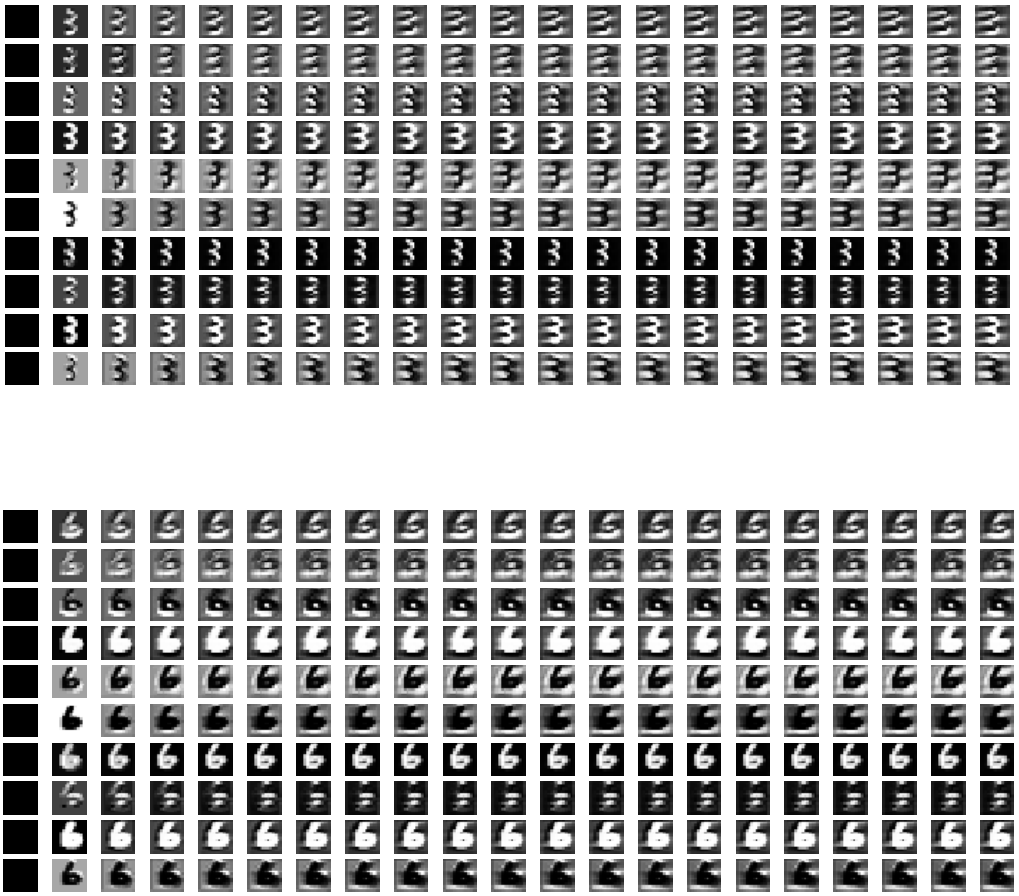

Figure 3.4: Images of two samples of the states filter maps $w_i \in \mathbb{R}^{10\times 14\times 14}$ for a three and a six from the MNIST dataset, learned with the fixed-point method and with projection. Each of the ten rows represents a filter and the x-axis proceeds with the iterations of the EQM dynamics (for a total of $t = 20$ iterations). The states are initialized to 0 (black images on the left) and then the mapping (3.24) is iterated 20 times to approximately reach the fixed point representation (rightmost images).

(denoted by † in the figure). The statistical performances of the contractive convolutional EQM exceeds abundantly those given by simpler dynamics of (3.23), with the fixed-point method (red line) being slightly better than the others. We show some visual examples of the learned dynamics in Figure 3.4, where we plot the 10 bi-dimensional state filter maps as the iterations of (3.24) proceed. Interestingly, when the projection is not performed, optimization with the fixed-point scheme to compute the hypergradient (akin to recurrent backpropagation, see green shaded region in the

rightmost plot of Figure 3.3) does not reliably converge for all the probed values of the step-size, indicating once more the importance of the contractiveness assumption for AID methods. Finally, note that regularizing the norm of $\partial_1 \phi_i$ or adding L_1 or L_∞ penalty terms on the matrix of the state-wise linear transformation may encourage, but does not strictly enforce, such condition. This may in part explain some difficulties encountered in training EQM-like models, e.g. in the context of relational learning (graph neural networks).

3.5 Discussion

In this chapter, we studied a general class of bilevel problems where at the lower-level we seek for a solution to a parametric fixed point equation. This formulation encompasses several learning algorithms recently considered in the literature. We established results on the iteration complexity of two strategies to compute the hypergradient (ITD and AID) under the assumption that the fixed point equation is defined by a contraction mapping. Our practical experience with these methods on a number of bilevel problems indicates that there is a trade-off between them, with AID based on the conjugate gradient method being preferable due to a potentiality better approximation of the hypergradient and lower space complexity. When the contraction assumption is not satisfied, however, our experiments on equilibrium models suggest that ITD is more reliable than AID methods.

Chapter 4

Stochastic Hypergradient Approximation

4.1 Introduction

In this chapter we study the following stochastic bilevel problem.

$$\begin{aligned} &\min_{\lambda \in \Lambda} f(\lambda) := \mathbb{E}[\hat{E}(w(\lambda), \lambda, \xi)] \\ &\text{with } \; w(\lambda) = \mathbb{E}[\hat{\Phi}(w(\lambda), \lambda, \zeta)], \end{aligned} \tag{4.1}$$

where $\Lambda \subseteq \mathbb{R}^n$ is closed and convex, $\hat{E}\colon \mathbb{R}^d \times \Lambda \times \Xi \to \mathbb{R}$, $\hat{\Phi}\colon \mathbb{R}^d \times \Lambda \times Z \to \mathbb{R}^d$ and ξ, ζ are two independent random variables with values in Ξ and Z, respectively. We also define

$$E(w, \lambda) := \mathbb{E}[\hat{E}(w, \lambda, \xi)], \quad \Phi(w, \lambda) := \mathbb{E}[\hat{\Phi}(w, \lambda, \xi)],$$

In dealing with problem (4.1), one critical issue is to devise efficient algorithms to compute the (hyper) gradient of the function f, so as to allow using gradient based approaches to find a solution. The computation of the hypergradient via approximate implicit differentiation (AID) (Pedregosa, 2016) requires one to solve two subproblems: (i) the lower-level problem in (4.1) and (ii) a linear system which arises from the implicit expression for $\nabla f(\lambda)$. However, especially in large scale scenarios, solving those subproblems exactly might either be impossible or too expensive,

hence, iterative approximation methods are often used. In Chapter 3, under the assumption that, for every $\lambda \in \Lambda$, the mapping $\Phi(\cdot, \lambda)$ in (4.1) is a contraction, a comprehensive analysis of the iteration complexity of the hypergradient computation for several popular deterministic algorithms was provided. Here instead, we address such iteration complexity for stochastic methods. This study is of fundamental importance since in many practical scenarios $\Phi(w, \lambda)$ and/or $E(w, \lambda)$ are expensive to compute, e.g., when they have a sum structure with many terms. In this situation stochastic approaches become the method of choice. For example, in large scale hyperparameter optimization and neural architecture search (Maclaurin et al., 2015; Lorraine et al., 2020; Liu et al., 2018), solving the lower-level problem requires minimizing a training objective over a large dataset, which is usually done approximately through SGD and its extensions. This chapter's contributions can be summarized as follows.

- We devise a stochastic estimator $\hat{\nabla} f(\lambda)$ of the true gradient, based on the AID technique, together with an explicit bound for the related mean square error. The bound is agnostic with respect to the stochastic methods solving the related subproblems, so that can be applied to several algorithmic solutions; see Theorem 4.4.5.

- We study the convergence of a general stochastic fixed-point iteration method which extends and improves previous analysis of SGD for strongly convex functions and can be applied to solve both subproblems associated to the AID approach. These results, which are interesting in their own right, are given in Theorems 4.5.2 and 4.5.3.

Proofs of the results presented in the chapter can be found in Appendix B.

4.2 Related Work

Pedregosa (2016) introduced an efficient class of deterministic methods to compute the hypergradient through AID together with asymptotic convergence results. Rajeswaran et al. (2019); Grazzi et al. (2020) extended this analysis providing iteration

complexity bounds. AID methods require to iteratively evaluate Φ and its derivatives. In this chapter, we extend these methods by replacing those exact evaluations with unbiased stochastic approximations and provide iteration complexity bounds in this scenario.

Another class of methods (ITD) computes the hypergradient by differentiating through the inner optimization scheme (Maclaurin et al., 2015; Franceschi et al., 2017, 2018). Iteration complexity results for the deterministic case are given in Grazzi et al. (2020). Here, we focus entirely on AID methods, leaving the investigation of stochastic ITD methods for future work.

An interesting special case of the bilevel problem (4.1) is when $f(\lambda) = \min_w E(w, \lambda)$. This scenario occurs for example in regularized meta-learning, where the properties of a simple stochastic hypergradient estimator have been studied extensively (Denevi et al., 2019a,b; Zhou et al., 2019). In this setting, Ablin et al. (2020) analyze, among others, implicit differentiation techniques for approximating the gradient of f, including stochastic approaches. However, the proposed estimator assumes to solve the related linear system exactly, which is often impractical. In this chapter, we focus on the more general setting of bilevel problem (4.1), devising algorithmic solutions that are fully stochastic, in the sense that also the subproblem involving the linear system is solved by a stochastic method.

Finally, stochastic algorithms for hypergradient computation in bilevel optimization problems have been studied in (Couellan and Wang, 2016; Ghadimi and Wang, 2018). There, the authors provide convergence rates for a whole bilevel optimization procedure using stochastic oracles both from the upper-level and the lower-level objectives. In particular, the method used by Ghadimi and Wang (2018) to approximate the hypergradient can be seen as a special case of our method with two particular choices of the stochastic solvers.[1]

[1] Specifically they use SGD with decreasing step sizes for the lower-level problem (which is a minimization problem) and, for the linear system, a stochastic routine derived from the Neumann series approximation of the matrix inverse.

4.3 Stochastic Implicit Differentiation

In this section we describe a general method for generating a stochastic approximation of the (hyper)gradient of f in (4.1). A special case of (4.1), which often occurs in machine learning, is

$$\begin{aligned} &\min_{\lambda\in\Lambda} f(\lambda) := E(w(\lambda),\lambda) \\ &\text{with } \; w(\lambda) = \operatorname{argmin}_w \mathbb{E}[\hat{\mathcal{L}}(w,\lambda,\zeta)], \end{aligned} \tag{4.2}$$

where $w \mapsto \mathbb{E}[\hat{\mathcal{L}}(w,\lambda,\zeta)]$ is strongly convex and Lipschitz smooth for every $\lambda \in \Lambda$. Indeed, (4.2) follows from (4.1) and the definition of E by choosing $\hat{\Phi}(w,\lambda,\zeta) = w - \alpha_\lambda \nabla \hat{\mathcal{L}}(w,\lambda,\zeta)$, for any $\alpha_\lambda > 0$. Consider the following assumptions.

Assumption 4.3.1. *The set $\Lambda \subseteq \mathbb{R}^m$ is closed and convex and the mappings $\Phi\colon \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$ and $E\colon \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$ are differentiable. For every $\lambda \in \Lambda$, we assume*

(i) *$\Phi(\cdot,\lambda)$ is a contraction, i.e., $\|\partial_1 \Phi(w,\lambda)\| \le q_\lambda$ for some $q_\lambda < 1$ $\forall w \in \mathbb{R}^d$.*

(ii) *$\partial_1\Phi(\cdot,\lambda), \partial_2\Phi(\cdot,\lambda)$ are Lipschitz continuous with constants $\nu_{1,\lambda}, \nu_{2,\lambda}$ respectively.*

(iii) *$\nabla_1 E(\cdot,\lambda), \nabla_2 E(\cdot,\lambda)$ are Lipschitz continuous with constants $\mu_{1,\lambda}, \mu_{2,\lambda}$ respectively.*

(iv) *$E(\cdot,\lambda)$ is Lipschitz continuous with constant $L_{E,\lambda}$.*

Similar assumptions are also considered and discussed in Chapter 3. Here we add Assumption 4.3.1(iv), which is necessary to bound $\|\nabla_1 E(w_t(\lambda),\lambda)\|$, where $w_t(\lambda)$ is given by a stochastic algorithm without making assumptions on the algorithms iterates. *Assumption* 4.3.1(*iv*) is satisfied e.g. for the cross-entropy loss but not for the square loss at the upper-level.

As we showed in Chapter 3, under Assumption 4.3.1, $\Phi(\cdot,\lambda)$ has a unique fixed point $w(\lambda)$ and the hypergradient is given by

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda),\lambda) + \partial_2 \Phi(w(\lambda),\lambda)^\top v(w(\lambda),\lambda), \tag{4.3}$$

where

$$v(w,\lambda) := \left(I - \partial_1\Phi(w,\lambda)^\top\right)^{-1} \nabla_1 E(w,\lambda). \tag{4.4}$$

This formula follows by differentiating the fixed point conditions for the lower-level problem and noting that, because of Assumption 4.3.1(i), $I - \partial_1\Phi(w,\lambda)^\top$ is invertible (see Lemma 3.3.6). We also consider the following properties for $\hat{\Phi}$ and $\hat{E}$.

Assumption 4.3.2. *The random variables ζ and ξ take values in measurable spaces Ξ and Z and $\hat{\Phi} : \mathbb{R}^d \times \Lambda \times Z \mapsto \mathbb{R}^d$, $\hat{E} : \mathbb{R}^d \times \Lambda \times \Xi \mapsto \mathbb{R}$ are measurable functions, differentiable w.r.t. the first two arguments in an open set containing $\mathbb{R}^d \times \Lambda$, and, for all $w \in \mathbb{R}^d$, $\lambda \in \Lambda$:*

(i) $\mathbb{E}[\hat{\Phi}(w,\lambda,\zeta)]=\Phi(w,\lambda)$, $\mathbb{E}[\hat{E}(w,\lambda,\xi)]=E(w,\lambda)$ *and we can exchange derivatives with expectations when taking derivatives on both sides.*

(ii) $\mathbb{V}[\hat{\Phi}(w,\lambda,\zeta)] \leq \sigma_{1,\lambda} + \sigma_{2,\lambda}\|\Phi(w,\lambda) - w\|^2$ *for some* $\sigma_{1,\lambda}, \sigma_{2,\lambda} \geq 0$.

(iii) $\mathbb{V}[\partial_1\hat{\Phi}(w,\lambda,\zeta)] \leq \sigma'_{1,\lambda}$, $\mathbb{V}[\partial_2\hat{\Phi}(w,\lambda,\zeta)] \leq \sigma'_{2,\lambda}$ *for some* $\sigma'_{1,\lambda}, \sigma'_{2,\lambda} \geq 0$.

(iv) $\mathbb{V}[\nabla_1\hat{E}(w,\lambda,\xi)] \leq \hat{\sigma}_{1,\lambda}$, $\mathbb{V}[\nabla_2\hat{E}(w,\lambda,\xi)] \leq \hat{\sigma}_{2,\lambda}$ *for some* $\hat{\sigma}_{1,\lambda}, \hat{\sigma}_{2,\lambda} \geq 0$.

Assumption 4.3.2 is satisfied in many cases. In particular, if $\hat{\Phi}$ is the gradient descent map on a few examples and $\hat{E}$ is the loss on a few validation examples and ζ, ξ select the indices of those examples (over a finite set of indices), the assumption is generally satisfied. In this scenario, Assumption 4.3.2(ii)(iii) map directly to equivalent assumptions on the variance of the stochastic estimator of the gradient and hessian of the loss function with $\sigma'_{2,\lambda} = 0$ if λ is a regularization parameter. The assumption is satisfied for every experiment in Sections 4.7 and 4.8.

To approximate $\nabla f(\lambda)$, we use mini-batch estimators $\nabla\bar{E}_J$ and $\partial_2\bar{\Phi}_J$ each with J samples, to estimate ∇E and $\partial_2\Phi$ respectively[2]. In particular, $\nabla_i\bar{E}_J(w_t(\lambda),\lambda) = \frac{1}{J}\sum_{j=1}^J \nabla_i\hat{E}(w_t(\lambda),\lambda,\xi_j)$ for $i \in \{1,2\}$ and $\partial_2\bar{\Phi}_J(w_t(\lambda),\lambda) = \frac{1}{J}\sum_{j=1}^J \partial_2\hat{\Phi}(w_t(\lambda),\lambda,\zeta'_j)$. Furthermore, motivated by (4.3)-(4.4), we consider to have at our disposal two stochastic solvers exploiting $\hat{\Phi}$: one for the lower-level

[2]For simplicity we use the same number of samples (J) for all the estimators. However, we observe that if e.g. λ is the regularization parameter at the lower-level, then $\sigma'_{2,\lambda} = 0$ and hence only one sample ($J = 1$) is needed to estimate $\partial_2\Phi$.

problem in (4.1) which generates a stochastic process $w_t(\lambda)$ estimating $w(\lambda)$ and another for the linear system

$$(I - \partial_1 \Phi(w,\lambda)^\top)v = \nabla_1 \bar{E}_J(w,\lambda), \quad \text{with } w \in \mathbb{R}^d, \tag{4.5}$$

generating a stochastic process $v_k(w,\lambda)$ approximating the solution $\bar{v}(w,\lambda)$ of (4.5). Then, the stochastic approximation to the hypergradient is defined as

$$\hat{\nabla} f(\lambda) := \nabla_2 \bar{E}_J(w_t(\lambda),\lambda) + \partial_2 \bar{\Phi}_J(w_t(\lambda),\lambda)^\top v_k(w_t(\lambda),\lambda). \tag{4.6}$$

Algorithm 4.3.1 Stochastic Implicit Differentiation (SID)

Requires: t, k, J, λ.

1. Let $t \in \mathbb{N}$ and compute $w_t(\lambda)$ by t steps of a stochastic algorithm that approximates $w(\lambda)$.
2. Compute $\nabla_i \bar{E}_J(w_t(\lambda),\lambda) = \frac{1}{J}\sum_{j=1}^J \nabla_i \hat{E}(w_t(\lambda),\lambda,\xi_j)$, where $(\xi_j)_{1\le j\le J}$ are i.i.d. copies of ξ and $i \in \{1,2\}$.
3. Let $k \in \mathbb{N}$ and Compute $v_k(w_t(\lambda),\lambda)$ by k steps of a stochastic solver for the linear system
$$(I - \partial_1 \Phi(w_t(\lambda),\lambda)^\top)v = \nabla_1 \bar{E}_J(w_t(\lambda),\lambda). \tag{4.7}$$
4. Compute the approximate hypergradient as
$$\hat{\nabla} f(\lambda) := \nabla_2 \bar{E}_J(w_t(\lambda),\lambda) + \partial_2 \bar{\Phi}_J(w_t(\lambda),\lambda)^\top v_k(w_t(\lambda),\lambda).$$
where $\partial_2 \bar{\Phi}_J(w_t(\lambda),\lambda) = \frac{1}{J}\sum_{j=1}^J \partial_2 \hat{\Phi}(w_t(\lambda),\lambda,\zeta'_j)$ and $(\zeta'_j)_{1\le j\le J}$ are i.i.d. copies of ζ.

The procedure, which we call SID, is summarized in Algorithm 4.3.1. In Section 4.6 we will give a way to generate the stochastic processes $(w_t(\lambda))_{t\in\mathbb{N}}$ and $(v_k(w,\lambda))_{k\in\mathbb{N}}$.

4.4 Convergence of Stochastic Implicit Differentiation

In this section, we provide an upper bound to the mean squared error of the hypergradient approximation:

$$\mathrm{MSE}_{\hat{\nabla} f} := \mathbb{E}[\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|^2], \tag{4.8}$$

where $\hat{\nabla} f(\lambda)$ is given by SID (Algorithm 5.2.1). In particular, we show that when the mini-batch size J, the number of lower-level and linear system iterations t and k tend to infinity, and the algorithms to solve the lower-level problem and the linear system converge in mean square error, then the mean square error of $\hat{\nabla} f(\lambda)$ tends to zero. Moreover, using stochastic fixed-point iteration solvers with decreasing stepsizes and setting $t = k = J$ we have $\mathrm{MSE}_{\hat{\nabla} f} = O(1/t)$.

We require the stochastic procedures at point 1 and 3 of Algorithm 4.3.1 to have non-asymptotic convergence rates in mean square[3]. This is the content of the following assumption.

Assumption 4.4.1. *For every $w \in \mathbb{R}^d$, $\lambda \in \Lambda$, $t,k,J \geq 1$, $j \in \{1,\dots,J\}$, the random variables $v_k(w,\lambda)$, $w_t(\lambda)$, ζ_j', are mutually independent, $w_t(\lambda)$ is independent of ξ_j and*

$$\mathbb{E}[\|w_t(\lambda) - w(\lambda)\|^2] \leq \rho_\lambda(t), \qquad \mathbb{E}[\|v_k(w,\lambda) - \bar{v}(w,\lambda)\|^2] \leq \sigma_\lambda(k),$$

where $\rho_\lambda : \mathbb{N} \mapsto \mathbb{R}_+$ and $\sigma_\lambda : \mathbb{N} \mapsto \mathbb{R}_+$.

In order to analyze the quantity in (4.8), we start with the standard bias-variance decomposition (see Lemma B.4.3) as follows

$$\mathrm{MSE}_{\hat{\nabla} f} = \underbrace{\|\mathbb{E}[\hat{\nabla} f(\lambda)] - \nabla f(\lambda)\|^2}_{\text{bias}} + \underbrace{\mathbb{V}[\hat{\nabla} f(\lambda)]}_{\text{variance}}. \tag{4.9}$$

[3]Exact rates are derived in Section 4.5

Then, using the law of total variance, we can write the useful decomposition

$$\mathbb{V}[\hat{\nabla} f(\lambda)] = \underbrace{\mathbb{E}[\mathbb{V}[\hat{\nabla} f(\lambda) \mid w_t(\lambda)]]}_{\text{variance I}} + \underbrace{\mathbb{V}[\mathbb{E}[\hat{\nabla} f(\lambda) \mid w_t(\lambda)]]}_{\text{variance II}}. \tag{4.10}$$

These decompositions allows us to analyze bias and variance separately and to decouple the randomness of the lower level solver from the one on the linear system and minibatch estimators, thus simplifying the analysis.

In the following three theorems we will bound the bias and the variance terms of the MSE. After that we state the final MSE bound in Theorem 4.4.5.

Theorem 4.4.2 (**Bias upper bounds**). *Suppose that Assumptions 4.3.1,4.3.2 and 4.4.1 are satisfied. Let $\lambda \in \Lambda$, $t,k \in \mathbb{N}$. Let $\hat{\Delta}_w := \|w_t(\lambda) - w(\lambda)\|$, then the following hold.*

(i) $\left\|\mathbb{E}[\hat{\nabla} f(\lambda) \mid w_t(\lambda)] - \nabla f(\lambda)\right\| \leq c_1 \hat{\Delta}_w + L_{\Phi,\lambda} \sqrt{\sigma_\lambda(k)} + \nu_{2,\lambda} \hat{\Delta}_w \sqrt{\sigma_\lambda(k)}.$

(ii) $\|\mathbb{E}[\hat{\nabla} f(\lambda)] - \nabla f(\lambda)\| \leq c_1 \sqrt{\rho_\lambda(t)} + L_{\Phi,\lambda} \sqrt{\sigma_\lambda(k)} + \nu_{2,\lambda} \sqrt{\rho_\lambda(t)} \sqrt{\sigma_\lambda(k)},$

where

$$c_1 = \mu_{2,\lambda} + \frac{\mu_{1,\lambda} L_{\Phi,\lambda} + \nu_{2,\lambda} L_{E,\lambda}}{1 - q_\lambda} + \frac{\nu_{1,\lambda} L_{E,\lambda} L_{\Phi,\lambda}}{(1-q_\lambda)^2}, \quad L_{\Phi,\lambda} = \partial_2 \Phi(w(\lambda), \lambda).$$

The proof is in Appendix B.1.1.

Theorem 4.4.3 (**Variance I bound**). *Suppose that Assumptions 4.3.1,4.3.2 and 4.4.1 are satisfied. Let $\lambda \in \Lambda$, $t,k \in \mathbb{N}$. Then*

$$\begin{aligned}\mathbb{E}[\mathbb{V}[\hat{\nabla} f(\lambda) \mid w_t(\lambda)]] \leq & \left(\hat{\sigma}_{2,\lambda} + 4 \frac{\sigma'_{2,\lambda}(L^2_{E,\lambda} + \hat{\sigma}_{1,\lambda}) + L^2_{\Phi,\lambda} \hat{\sigma}_{1,\lambda}}{(1-q_\lambda)^2} \right) \frac{2}{J} \\ & + 8(L^2_{\Phi,\lambda} + \sigma'_{2,\lambda}) \sigma_\lambda(k) + 8 \nu^2_{2,\lambda} \rho_\lambda(t) \left(\sigma_\lambda(k) + \frac{\hat{\sigma}_{1,\lambda}}{J(1-q_\lambda)^2} \right),\end{aligned}$$

where $L_{\Phi,\lambda}$ is defined as in Theorem 4.4.2.

The proof is in Appendix B.1.2.

Theorem 4.4.4 (Variance II bound). *Suppose that Assumptions 4.3.1,4.3.2 and 4.4.1 are satisfied. Let $\lambda \in \Lambda$, and $t,k \in \mathbb{N}$. Then*

$$\mathbb{V}[\mathbb{E}[\hat{\nabla} f(\lambda) \,|\, w_t(\lambda)]] \leq 3\big(c_1^2 \rho_\lambda(t) + L_{\Phi,\lambda}^2 \sigma_\lambda(k) + \nu_{2,\lambda}^2 \rho_\lambda(t)\sigma_\lambda(k)\big),$$

where c_1 and $L_{\Phi,\lambda}$ are defined as in Theorem 4.4.2.

Proof. We get $\mathbb{V}[\mathbb{E}[\hat{\nabla} f(\lambda) \,|\, w_t(\lambda)]] \leq \mathbb{E}\big[\|\mathbb{E}[\hat{\nabla} f(\lambda) \,|\, w_t(\lambda)] - \nabla f(\lambda)\|^2\big]$ from the property of the variance (Lemma B.4.3(ii)). The statement follows from Theorem 4.4.2(i), the inequality $(a+b+c)^2 \leq 3(a^2+b^2+c^2)$, then taking the total expectation and finally using Assumption 4.4.1. □

Theorem 4.4.5 (MSE bound for SID). *Suppose that Assumptions 4.3.1,4.3.2 and 4.4.1 are satisfied. Let $\lambda \in \Lambda$, and $t,k,J \in \mathbb{N}$. Then, if we use Algorithm 5.2.1, we have*

$$MSE_{\hat{\nabla} f} \leq \left(\hat{\sigma}_{2,\lambda} + 4\frac{\sigma'_{2,\lambda}(L_{E,\lambda}^2 + \hat{\sigma}_{1,\lambda}) + L_{\Phi,\lambda}^2 \hat{\sigma}_{1,\lambda}}{(1-q_\lambda)^2}\right)\frac{2}{J} + \left(6c_1^2 + \frac{8\nu_{2,\lambda}^2 \hat{\sigma}_{1,\lambda}}{(1-q_\lambda)^2}\right)\rho_\lambda(t)$$
$$+ \left(14L_{\Phi,\lambda}^2 + 8\sigma'_{2,\lambda}\right)\sigma_\lambda(k) + 14\nu_{2,\lambda}^2 \rho_\lambda(t)\sigma_\lambda(k),$$

where c_1 and $L_{\Phi,\lambda}$ are defined in Theorem 4.4.2. In particular, if $\lim_{t\to\infty}\rho_\lambda(t) = \lim_{k\to\infty}\sigma_\lambda(k) = 0$, then

$$\lim_{t,k,J\to\infty} MSE_{\hat{\nabla} f} = 0$$

Proof. Follows from (4.9)-(4.10) and summing bounds in Theorems 4.4.2(ii), 4.4.3, and 4.4.4. □

We will show in Section 4.6 that by using the stochastic fixed point iteration solvers with carefully chosen decreasing stepsizes, we have $\rho_\lambda(t) = O(1/t)$ and $\sigma_\lambda(k) = O(1/k)$ and hence, by setting $t = k = J$ we can achieve $\mathrm{MSE}_{\hat{\nabla} f} = O(1/t)$ (Corollary 4.6.2).

4.5 Stochastic Fixed-Point Iterations

In this section we address the convergence of stochastic fixed-point iteration methods which can be applied similarly to solve both subproblems in Algorithm 4.3.1 (see

Section 4.6). We consider the general situation of computing the fixed point of a contraction mapping which is accessible only through a stochastic oracle. The results are inspired by the analysis of the SGD algorithm for strongly convex and Lipschitz smooth functions given in (Bottou et al., 2018), but extended to our more general setting. Indeed, by a more accurate computation of the contraction constant of the gradient descent mapping, we are able to improve the convergence rates and increase the stepsizes given in the above cited paper. See Corollary 4.5.7 and the subsequent remark. We stress that the significance of the results presented in this section goes beyond the bilevel setting (4.1) and may be of interest per se. Proofs are in Appendix B.2

In line with the assumptions in Bottou et al. (2018) for the case of stochastic minimization of a strongly convex and Lipschitz smooth function, we make the following assumption.

Assumption 4.5.1. *Let ζ be a random variable with values in a measurable space $\mathcal{Z}$. Let $T : \mathbb{R}^d \mapsto \mathbb{R}^d$ and $\hat{T} : \mathbb{R}^d \times \mathcal{Z} \mapsto \mathbb{R}^d$ be such that*

(i) $\forall w_1, w_2 \in \mathbb{R}^d$, $\|T(w_1) - T(w_2)\| \leq q\|w_1 - w_2\|$, *with* $q < 1$.

(ii) $\forall w \in \mathbb{R}^d$, $\mathbb{E}[\hat{T}(w, \zeta)] = T(w)$

(iii) $\forall w \in \mathbb{R}^d$, $\mathbb{V}[\hat{T}(w, \zeta)] \leq \sigma_1 + \sigma_2\|T(w) - w\|^2$.

Since T is a contraction, there exists a unique $w^* \in \mathbb{R}^d$ such that

$$w^* = T(w^*). \tag{4.11}$$

We consider the following random process which corresponds to a stochastic version of the Krasnoselskii-Mann iteration for contractive operators. Let $(\zeta_t)_{t\in\mathbb{N}}$ be a sequence of independent copies of ζ. Then, starting from $w_0 \in \mathbb{R}^d$ we set

$$(\forall t \in \mathbb{N}) \quad w_{t+1} = w_t + \eta_t(\hat{T}(w_t, \zeta_t) - w_t). \tag{4.12}$$

The following two results provide non-asymptotic convergence rates for the procedure (4.12) for two different strategies about the step-sizes η_t.

Theorem 4.5.2 (Constant step-size)**.** *Let Assumption 4.5.1 hold and suppose that* $\eta_t = \eta \in \mathbb{R}_{++}$*, for every* $t \in \mathbb{N}$*, and that*

$$\eta \leq \frac{1}{1+\sigma_2}.$$

Let $(w_t)_{t\in\mathbb{N}}$ *be generated according to algorithm* (4.12) *and set* $MSE_{w_t} := \mathbb{E}[\|w_t - w^*\|^2]$*. Then, for all* $t \in \mathbb{N}$*,*

$$MSE_{w_t} \leq (1-\eta(1-q^2))^t \left(MSE_{w_0} - \frac{\eta\sigma_1}{1-q^2} \right) + \frac{\eta\sigma_1}{1-q^2}. \tag{4.13}$$

In particular, $\lim_{t\to\infty} MSE_{w_t} \leq \eta\sigma_1/(1-q^2)$*.*

Theorem 4.5.3 (Decreasing step-sizes)**.** *Let Assumption 4.5.1 hold and suppose that for every* $t \in \mathbb{N}$

$$\eta_t \leq \frac{1}{1+\sigma_2}, \quad \sum_{t=1}^{\infty} \eta_t = \infty, \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty. \tag{4.14}$$

Let $(w_t)_{t\in\mathbb{N}}$ *be generated according to Algorithm* (4.12)*. Then*

$$w_t \to w^* \qquad \mathbb{P}\text{-}a.s.$$

Moreover, if $\eta_t = \beta/(\gamma+t)$*, with* $\beta > 1/(1-q^2)$ *and* $\gamma \geq \beta(1+\sigma_2)$*, then we have*

$$\mathbb{E}[\|w_t - w^*\|^2] \leq \frac{1}{\gamma+t} \max\left\{ \gamma\mathbb{E}[\|w_0 - w^*\|^2], \frac{\beta^2\sigma_1}{\beta(1-q^2)-1} \right\} \tag{4.15}$$

We will now comment on the choice of the stepsizes in algorithm (4.12). Theorem 4.5.2 and 4.5.3 suggest that it may be convenient to start the algorithm with a constant stepsize. Then, once reached a mean square error approximately less than $\eta\sigma_1/(1-q^2)$, the stepsizes should change regime and start decreasing according to Theorem 4.5.3. More precisely, in the first phase it is recommended to set $\eta = 1/(1+\sigma_2)$ in order to maximize the stepsize. Then, the second phase should be initialized with w_0 such that $\mathrm{MSE}_{w_0} \leq \sigma_1/[(1+\sigma_2)(1-q^2)]$ and $\gamma = \beta(1+\sigma_2)$ so

that

$$\gamma \mathbb{E}[\|w_0 - w^*\|^2] \leq \frac{\beta^2 \sigma_1}{\beta(1-q^2)-1}.$$

In this situation, c will be dominated by its second term, which is minimized when $\beta = 2/(1-q^2)$. Similar suggestions are made in (Bottou et al., 2018).

In the following, partly inspired by the analysis of Nguyen et al. (2019), we show that, with an additional Lipschitz assumption on $\hat{T}$, which is commonly verified in practice, Assumption 4.5.1(iii) on the variance of the estimator is satisfied. The following Assumption 4.5.4 is an extension of Assumption 2 in (Nguyen et al., 2019).

Assumption 4.5.4. *There exists $L_{\hat{T}} \geq 0$ such that, for every $w_1, w_2 \in \mathbb{R}^d$ and $\forall z \in \mathcal{Z}$*

$$\|\hat{T}(w_1, z) - \hat{T}(w_2, z)\| \leq L_{\hat{T}} \|w_1 - w_2\|.$$

Theorem 4.5.5. *Suppose that Assumption 4.5.4 and Assumption 4.5.1(i)(ii) hold. Then Assumption 4.5.1(iii) holds. In particular, for every $w \in \mathbb{R}^d$,*

$$\mathbb{V}[\hat{T}(w, \zeta)] \leq \underbrace{2\mathbb{V}[\hat{T}(w^*, \zeta)]}_{\sigma_1} + \underbrace{2\frac{L_{\hat{T}}^2 + q^2}{(1-q)^2}}_{\sigma_2} \|T(w) - w\|^2.$$

We now discuss the popular case of SGD and make a comparison with the related results by Bottou et al. (2018). We assume that $\hat{T}(w, \zeta) = w - \alpha \nabla \hat{\mathcal{L}}(w, \zeta)$, for a suitable $\alpha > 0$. With this choice, algorithm (4.12) becomes

$$(\forall t \in \mathbb{N}) \quad w_{t+1} = w_t - \eta_t \alpha \nabla_1 \mathcal{L}(w_t, \zeta_t), \tag{4.16}$$

which is exactly stochastic gradient descent. We have the following assumption.

Assumption 4.5.6. *$\hat{\mathcal{L}} : \mathbb{R}^d \times \mathcal{Z} \to \mathbb{R}$ is twice continuously differentiable w.r.t. the first variable. Let $\mathcal{L}(w) := \mathbb{E}[\hat{\mathcal{L}}(w, \zeta)]$.*

(i) $\mathcal{L}(w)$ is τ strongly convex and L-smooth

(ii) $\forall w \in \mathbb{R}^d$, $\mathbb{V}[\nabla \hat{\mathcal{L}}(w,\zeta)] \leq \sigma_1' + \sigma_2' \|\nabla \mathcal{L}(w)\|^2$.

Corollary 4.5.7. *Let Assumption 4.5.6 hold and let* $(w_t)_{t\in\mathbb{N}}$ *be generated according to algorithm* (4.16) *with* $\eta_t = \eta \leq 1/(1+\sigma_2')$. *Then*

$$\mathbb{E}[\|w_t - w^*\|^2] \leq r_1^t \left(\mathbb{E}[\|w_0 - w^*\|^2] - r_2\right) + r_2, \tag{4.17}$$

where

$$r_1 := \begin{cases} 1 - \dfrac{\eta\tau}{L}\left(2 - \dfrac{\tau}{L}\right) & \text{if } \alpha = 1/L \\ 1 - 4\dfrac{\eta\tau L}{(L+\tau)^2} & \text{if } \alpha = 2/(L+\tau). \end{cases}$$

$$r_2 := \begin{cases} = \dfrac{\eta\sigma_1'}{\tau(2L-\tau)} & \text{if } \alpha = 1/L \\ \dfrac{\eta\sigma_1'}{\tau L} & \text{if } \alpha = 2/(L+\tau). \end{cases}$$

Moreover, let $\eta_t = \beta/(\gamma+t)$, *where*

$$\beta > \begin{cases} \dfrac{L^2}{\tau(2L-\tau)} & \text{if } \alpha = 1/L \\ \dfrac{(L+\tau)^2}{4\tau L} & \text{if } \alpha = 2/(L+\tau) \end{cases} \tag{4.18}$$

and $\gamma \geq \beta(1+\sigma_2')$. *Then, for all* $t \in \mathbb{N}$, *we have*

$$\mathbb{E}[\|w_t - w^*\|^2] \leq \frac{\max\{\gamma \mathbb{E}[\|w_0 - w^*\|^2], r_3\}}{\gamma + t}, \tag{4.19}$$

where

$$r_3 := \begin{cases} \dfrac{\beta^2\sigma_1'}{\beta\tau(2L-\tau) - L^2} & \text{if } \alpha = 1/L \\ \dfrac{4\beta^2\sigma_1'}{4\beta\tau L - (L+\tau)^2} & \text{if } \alpha = 2/(L+\tau). \end{cases}$$

Remark 4.5.8. *In (Bottou et al., 2018), under Assumption 4.5.6 a rate equal to*

(4.17) *is obtained, but with* $\alpha = 1/L$ *and*

$$r_1 = 1 - \eta\frac{\tau}{L} \quad \textit{and} \quad r_2 = \frac{\eta\sigma_1'}{2\tau}. \tag{4.20}$$

We see then, that Corollary 4.5.7 provides better rates. Also, our analysis allows choosing the larger (and optimal) stepsize $2/(L+\tau)$.

Remark 4.5.9. *In Assumption 4.5.6, suppose that* ζ *takes values in* $\mathcal{Z} = \{1,\dots,n\}$ *with uniform distribution and that for every* $i \in \{1,\dots,n\}$, $\hat{\mathcal{L}}(\cdot,i)$ *is strongly convex with modulus* τ. *This is, for instance, the case of the regularized empirical risk functional,*

$$\hat{\mathcal{L}}(w,i) = \psi(y_i w^\top x_i) + \frac{\tau}{2}\|w\|^2, \tag{4.21}$$

where $(x_i,y_i)_{1\le i\le n} \in (\mathbb{R}^d \times \{1,2\})^n$ *is the training set. Then, if the loss function* ψ *is Lipschitz continuous with constant* $\mathrm{Lip}(\psi)$, *as is the case, e.g., of the logistic loss, we have*

$$\mathbb{V}[\nabla\hat{\mathcal{L}}(w,i)] = \mathbb{V}_{i\sim U[\mathcal{Z}]}[\psi'(y_i w^\top x_i)y_i x_i] \le \mathrm{Lip}(\psi)^2\mathbb{E}_{i\sim U[\mathcal{Z}]}[\|x_i\|^2],$$

so that Assumption 4.5.6(ii) is satisfied with $\sigma_2' = 0$.

4.6 Solvers for Lower-Level Problem and Linear System

We are now ready to show how to generate the sequences $w_t(\lambda)$ and $v_k(w_t(\lambda),\lambda)$ required by Algorithm 4.3.1. Let ζ, ξ be random variables with values in Z and Ξ. Let $(\zeta_t)_{t\in\mathbb{N}}$ and $(\hat{\zeta}_t)_{t\in\mathbb{N}}$ be independent copies of ζ and let $(\eta_{\lambda,t})_{t\in\mathbb{N}}$ be a sequence of stepsizes.

For every $w \in \mathbb{R}^d$ we let $v_0(w,\lambda) = 0$, $w_0 : \Lambda \to \mathbb{R}^d$, and, for $k,t \in \mathbb{N}$,

$$w_{t+1}(\lambda) := w_t(\lambda) + \eta_{\lambda,t}(\hat{\Phi}(w_t(\lambda),\lambda,\zeta_t) - w_t(\lambda)), \tag{4.22}$$

$$v_{k+1}(w,\lambda) := v_k(w,\lambda) + \eta_{\lambda,k}(\hat{\Psi}_w(v_k(w,\lambda),\lambda,\hat{\zeta}_k) - v_k(w,\lambda)), \tag{4.23}$$

where $\hat{\Psi}_w(v,\lambda,z) := \partial_1\hat{\Phi}(w,\lambda,z)^\top v + \nabla_1\bar{E}_J(w,\lambda)$, $\bar{E}_J(w,\lambda) = (1/J)\sum_{j=1}^J \hat{E}_(w,\lambda,\xi_j)$ and $(\xi_j)_{1\le j\le J}$ being i.i.d. copies of the random variable $\xi \in \Xi$.

Note that to reduce the number of hyperparameters of the method, we use the same sequence of stepsizes $(\eta_{\lambda,t})_{t\in\mathbb{N}}$ for both the LL and LS problems. This choice might not be optimal and results in more conservative step sizes. If $\nabla_1\bar{E}_J(w,\lambda)$ is computed once beforehand, and the Jacobian-vector product in $\hat{\Psi}_w$ is computed using reverse mode automatic differentiation, the costs of evaluating $\hat{\Psi}_w$ and $\hat{\Phi}$ are of the same order of magnitude. Furthermore, thanks to the definition of $\hat{\Psi}$, we can solve both subproblems in Algorithm 4.3.1 using the procedure described in Section 4.5. In particular, as illustrated by the following theorem, we can obtain similar convergence guarantees for both (4.22) and (4.23).

Theorem 4.6.1. *Let Assumption 4.3.1(i), 4.3.2 hold. Let $w_t(\lambda)$ and $v_k(w,\lambda)$ be defined as in* (4.22) *and* (4.23)*. Assume $\sum_{t=0}^\infty \eta_{\lambda,t} = +\infty$ and $\sum_{t=0}^\infty \eta_{\lambda,t}^2 < +\infty$. Then, for every $\lambda \in \Lambda$, $w \in \mathbb{R}^d$, we have*

$$\lim_{t\to\infty} w_t(\lambda) = w(\lambda), \quad \lim_{k\to\infty} v_k(w,\lambda) = \bar{v}(w,\lambda) \quad \mathbb{P}\text{-a.s.}$$

Moreover, let $\tilde{\sigma}_2 := \max\{2\sigma'_{1,\lambda}/(1-q)^2, \sigma_{2,\lambda}\}$ and $\eta_{\lambda,t} := \beta/(\gamma+t)$ with $\beta > 1/(1-q_\lambda^2)$ and $\gamma \ge \beta(1+\tilde{\sigma}_2)$. Then for every $w \in \mathbb{R}^d$, $t,k > 0$

$$\mathbb{E}[\|w_t(\lambda) - w(\lambda)\|^2] \le \frac{d_{w,\lambda}}{\gamma+t} \qquad \mathbb{E}[\|v_k(w,\lambda) - \bar{v}(w,\lambda)\|^2] \le \frac{d_{v,\lambda}}{\gamma+k} \tag{4.24}$$

where

$$d_{w,\lambda} := \max\left\{\gamma\|w(\lambda)\|^2, \frac{\beta^2\sigma_{1,\lambda}}{\beta(1-q_\lambda^2)-1}\right\},$$
$$d_{v,\lambda} := \max\left\{\frac{L_{E,\lambda}^2 + \hat{\sigma}_{1,\lambda}}{(1-q_\lambda)^2}\gamma, \frac{2(L_{E,\lambda}^2+\hat{\sigma}_{1,\lambda})\sigma'_{1,\lambda}}{(1-q_\lambda)^2}\frac{\beta^2}{\beta(1-q_\lambda^2)-1}\right\}$$

Alternatively, with constant step size $\eta_t = \eta \leq 1/(1+\tilde{\sigma}_2)$ *we have*

$$\mathbb{E}[\|w_t(\lambda) - w(\lambda)\|^2] \leq (1-\eta(1-q_\lambda^2))^t \|w(\lambda)\|^2 + \frac{\eta\sigma_{1,\lambda}}{1-q_\lambda^2} \tag{4.25}$$

$$\mathbb{E}[\|v_k(w,\lambda) - \bar{v}(w,\lambda)\|^2] \leq (1-\eta(1-q_\lambda^2))^k \frac{L_{E,\lambda}^2 + \hat{\sigma}_{1,\lambda}}{(1-q_\lambda)^2} \tag{4.26}$$

$$+ \frac{\eta}{1-q_\lambda^2} \frac{2(L_{E,\lambda}^2 + \hat{\sigma}_{1,\lambda})\sigma'_{1,\lambda}}{(1-q_\lambda)^2} \tag{4.27}$$

Proof is in Appendix B.3.1.

Corollary 4.6.2. *Suppose that Assumptions 4.3.1,4.3.2 are satisfied and suppose that* $\hat{\nabla} f(\lambda)$ *is computed via Algorithm 5.2.1 with* $t = k = J \in \mathbb{N}$ *and LL/LS stepsizes* $(\eta_j)_{j\in\mathbb{N}}$ *chosen according to the decreasing case of Theorem 4.6.1. Then, we obtain*

$$MSE_{\hat{\nabla} f} \leq \frac{c_b + c_v}{t}, \tag{4.28}$$

where

$$\begin{aligned} c_b &= 3c_1^2 d_{w,\lambda} + 3L_{\Phi,\lambda}^2 d_{v,\lambda} + 3\nu_{2,\lambda}^2 d_{w,\lambda} d_{v,\lambda} \\ c_v &= \hat{\sigma}_{2,\lambda} + 8\frac{\sigma'_{2,\lambda}(L_{E,\lambda}^2 + \hat{\sigma}_{1,\lambda}) + L_{\Phi,\lambda}^2 \hat{\sigma}_{1,\lambda}}{(1-q_\lambda)^2} + \left(3c_1^2 + \frac{8\nu_{2,\lambda}^2 \hat{\sigma}_{1,\lambda}}{(1-q_\lambda)^2}\right) d_{w,\lambda} \\ &\quad + (11L_{\Phi,\lambda}^2 + 8\sigma'_{2,\lambda}) d_{v,\lambda} + 11\nu_{2,\lambda}^2 d_{v,\lambda} d_{w,\lambda}, \end{aligned} \tag{4.29}$$

and $d_{w,\lambda}$, $d_{v,\lambda}$ *are defined in Theorem 4.6.1, while* c_1 *and* $L_{\Phi,\lambda}$ *are defined in Theorem 4.4.2.*

Crucially, typical bilevel problems in machine learning come in the form of (4.2), where the lower-level objective $\mathcal{L}(w,\lambda) := \mathbb{E}[\hat{\mathcal{L}}(w,\lambda,\zeta)]$ is Lipschitz smooth and strongly convex w.r.t. w. In this scenario, there is a vast amount of stochastic methods in literature (see e.g. Bottou et al. (2018) for a survey) achieving convergence rates in expectations of the kind provided in Theorem 4.6.1 or even better. For example, when $\mathcal{L}(w,\lambda)$ has a finite sum structure, as in the case of the regularized empirical risk, exploiting variance reduction techniques makes the convergence rate $\rho_\lambda(t)$ linear. One can easily see that since the linear system can be seen as positive definite

quadratic optimization problem, the same techniques can be applied also to this problem obtaining linear $\sigma_\lambda(k)$.

4.7 Preliminary Experiment

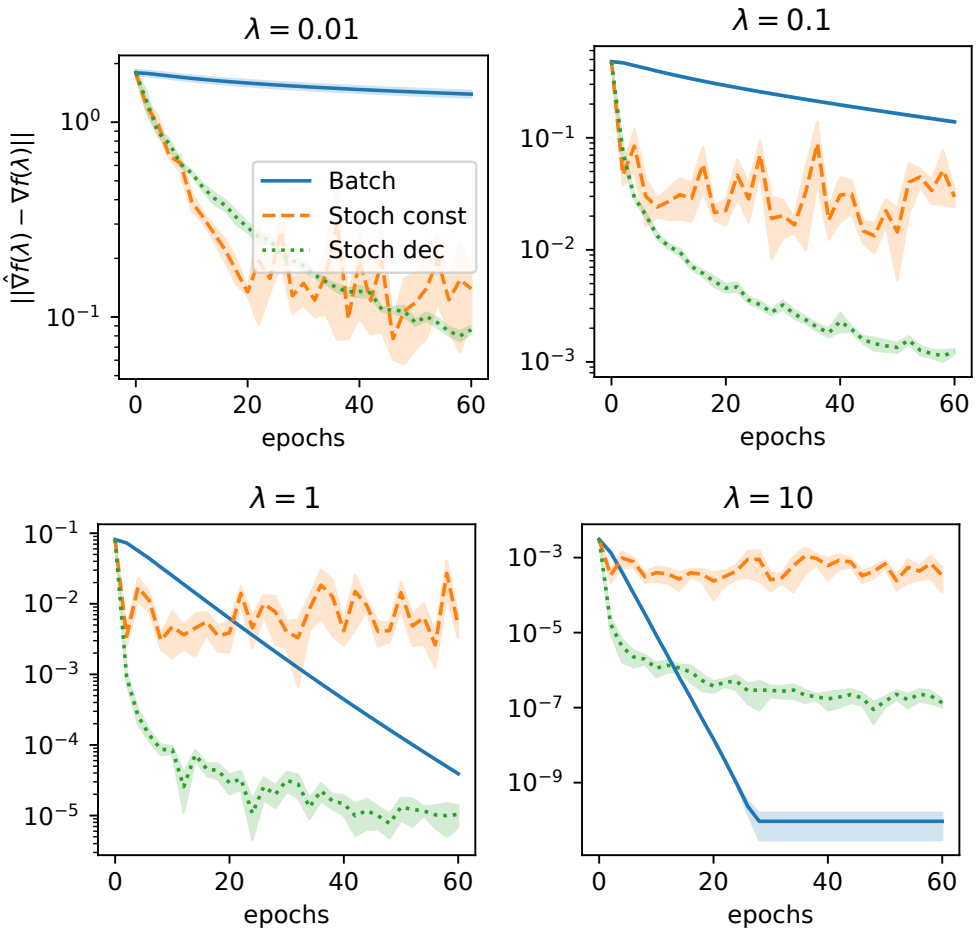

Figure 4.1: Experiment with a single regularization parameter. Convergence of three variants of SID for 4 choices of the regularization hyperparameter $\lambda \in \mathbb{R}_{++}$. Here, 2 epochs refer, in the Batch version, to one iteration on the lower-level problem plus one iteration on the linear system, whereas, in the Stochastic versions, they refer to 100 iterations on the lower-level problem plus 100 iterations on the linear system. The plot shows mean (solid lines) and std (shaded regions) over 5 runs, which vary the train/validation splits and, for the stochastic methods, the order and composition of the mini-batches.

In this section we present a preliminary experiment evaluating the effectiveness of the SID method for estimating the hypergradient of f in a real data scenario. In Section 4.8.1 we provide additional experiments on more realistic scenarios and with additional SID variants. We focus on a hyperparameter optimization problem where

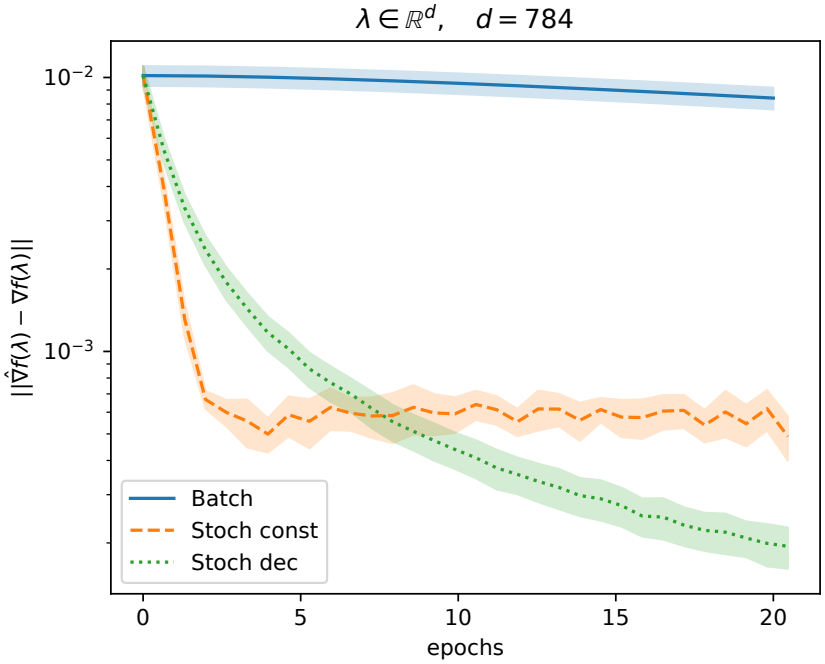

Figure 4.2: Experiment with multiple regularization parameters. Convergence of three variants of SID for several choices of the regularization hyperparameter $\lambda \in \mathbb{R}^d_{++}$. The plot shows mean (solid lines) and std (shaded regions) over 10 runs. For each run, $\lambda_i = e^{\varepsilon_i}$, where $\varepsilon_i \sim \mathcal{U}[-2,2]$ for every $i \in \{1,\dots,d\}$. Epochs are defined as in Figure 4.1.

we want to optimize the regularization parameter(s) in regularized logistic regression. Specifically, we consider a binary classification problem with the aim to distinguish between odd and even numbers in the MNIST dataset. Referring to problem (4.2), we set

$$f(\lambda) = \sum_{i=n_{\text{tr}}+1}^{n_{\text{tr}}+n_{\text{val}}} \psi(y_i x_i^\top w(\lambda)),$$
$$w(\lambda) = \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n_{\text{tr}}} \psi(y_i x_i^\top w) + R(w,\lambda),$$

where $\psi(u) = \log(1+e^{-u})$ is the logistic loss, $(x_i, y_i)_{1 \le i \le n_{\text{tr}}+n_{\text{val}}} \in (\mathbb{R}^p \times \{0,1\})^{n_{\text{tr}}+n_{\text{val}}}$ are training and validation examples, and $R(w,\lambda)$ is set according to either of the two situations below

- *one regularization parameter*:

$R(w,\lambda) = \frac{\lambda}{2}\|w\|^2$, $\lambda \in \mathbb{R}_{++}$

- *multiple regularization parameters*:

  $R(w,\lambda) = \frac{1}{2} w^\top \mathrm{d}iag(\lambda) w$ where $\mathrm{d}iag(\lambda)$ is the diagonal matrix formed by the elements of $\lambda \in \mathbb{R}^d_{++}$.

We set $n_{\text{tr}} = n_{\text{val}} = 5000$, i.e., we pick 10000 examples from the MNIST training set, and we group them into a training and a validation set of equal size. We set Φ to be the full gradient descent map on the lower-level objective, with optimal choice for the stepsize[4]. This map is a contraction because the lower objective is strongly convex and Lipschitz-smooth. We test the following three variants of SID (Algorithm 4.3.1), where we always solve the lower-level problem with t iterations of the procedure (4.22) and the linear system with $k = t$ iterations of the algorithm (4.23). However, we make different choices for $\eta_{\lambda,t}$ and the estimator $\hat{\Phi}$.

Batch. This variant of Algorithm 4.3.1 corresponds to the (deterministic) gradient descent algorithm with constant stepsize. We set $t_{\text{Batch}} = k_{\text{Batch}} = 30$ and, for every $t = 0, \ldots, t_{\text{Batch}}$, $\eta_{\lambda,t} = \eta_{\lambda,k} = 1$ and $\hat{\Phi}(w,\lambda,\zeta) = \Phi(w,\lambda)$.

Stoch const. For this variant, $\hat{\Phi}(w,\lambda,\zeta)$ corresponds to one step of stochastic gradient descent on a randomly sampled minibatch of 50 examples. Thus, $t_{\text{Stoch const}} = t_{\text{Batch}} \times 100$, $k_{\text{Stoch const}} = k_{\text{Batch}} \times 100$, and we pick $\eta_{\lambda,t} = \eta_{\lambda,k} = 1$, for $t = 0, \ldots, t_{\text{Stoch const}}$.

Stoch dec. For this variant the estimator $\hat{\Phi}$ is the same as for the *Stoch const* strategy, but we use decreasing stepsizes. More precisely, $\eta_{\lambda,t} = \eta_{\lambda,k} = \beta_\lambda/(\gamma_\lambda + t)$ with $\beta_\lambda = 2/(1-q_\lambda^2)$ and $\gamma_\lambda = \beta_\lambda$. Moreover, as before, $t_{\text{Stoch dec}} = t_{\text{Batch}} \times 100$, $k_{\text{Stoch dec}} = k_{\text{Batch}} \times 100$.

We note that the *Batch* strategy is exactly the *fixed point method* described in Chapter 3, which converges linearly to the true hypergradient. Moreover, for the stochastic versions, we set $\hat{E}(w,\lambda,\xi) = E(w,\lambda)$, and we can write $\hat{\Phi}(w,\lambda,\zeta) = \Phi_1(w,\zeta) + \Phi_2(w,\lambda)$, so that we are in the case discussed in Remark 4.5.9 and hence, $\sigma_2 = \sigma'_{2,\lambda} = 0$. In this situation we can see that the *Stoch dec* version of

[4]We set the stepsize equal to two divided by the sum of the Lipschitz and strong convexity constants of the lower-level objective. This gives the optimal contraction rate q_λ.

Algorithm 4.3.1 converges in expectation to the true hypergradient even when $J = 1$ which is the value we use, whereas, according to Corollary 4.5.7, the *Stoch const* version can possibly approach the true hypergradient in a first phase (at linear rate), but ultimately might not converge to it.

In Figures 4.1 and 4.2 we show the squared error between the approximate and the true hypergradient ($\hat{\nabla} f(\lambda)$ and $\nabla f(\lambda)$ respectively) for the two regularization choices described above[5]. In both figures we can see the effectiveness of the proposed SID method (and especially the *Stoch dec* variant) against its deterministic version (AID) studied in Chapter 3.

4.8 Experiments

In this section we provide additional experiments in two of the settings outlined in Chapter 3. In addition to the three methods considered in Section 4.7, we also test variants of the algorithm which use a mixed Stochastic/Batch strategy for the solution of the two subproblems as well as variants for which $t \neq k$. To have a fair comparison, each method computes the approximate hypergradient using the same number of epochs. We report the differences among the methods in Table 4.1.

Table 4.1: Differences among the methods used in the experiments. The column **% epochs**, provides percentages of epochs used to solve the lower level problem (LL) and the linear system (LS), while the column **algorithm** indicates which method is used for each of the two subproblems: gradient descent (GD), stochastic gradient descent with constant step size (SGD const) and stochastic gradient descent with decreasing step sizes (SGD dec).

Method	**% epochs (LL, LS)**	**algorithm (LL, LS)**
Batch	50, 50	GD, GD
Stoch const	50, 50	SGD const, SGD const
Stoch dec	50, 50	SGD const, SGD const
Stoch/Batch	50, 50	SGD dec, GD
Batch/Stoch	50, 50	GD, SGD dec
Batch 75%/25%	75, 25	GD, GD
Stoch const 75%/25%	75, 25	SGD const, SGD const
Stoch dec 75%/25%	75, 25	SGD dec, SGD dec

[5] Since for regularized logistic regression, the hypergradient is not available in closed form, we compute it by using the Batch version with $t = k = 2000$ (4000 epochs in total).

For each method we set $J = 1$ and the number of iterations for the lower-level problem and linear system (t and k) in Algorithm 4.3.1 as follows.

$$t = \text{round}\left(\frac{\%\text{ epochs LL}}{100} \times \text{total \# of epochs} \times n_{tr} \div \text{batch size LL}\right)$$
$$k = \text{round}\left(\frac{\%\text{ epochs LS}}{100} \times \text{total \# of epochs} \times n_{tr} \div \text{batch size LS}\right)$$

where % epochs LL/LS is the corresponding value in Table 4.1, n_{tr} is the number of examples in the training set and batch size LL (batch size LS) is the batch size used to solve the lower-level problem (linear system). The total number of epochs and n_{tr} depend on the setting and are the same for all methods.

4.8.1 Hypergradient Approximation on MNIST

We consider the following multinomial logistic regression setting on the MNIST dataset.

$$f(\lambda) = \sum_{i=n_{\text{tr}}+1}^{n_{\text{tr}}+n_{\text{val}}} \text{CE}(y_i, W(\lambda)x_i),$$
$$W(\lambda) = \arg\min_{W \in \mathbb{R}^{c\times d}} \sum_{i=1}^{n_{\text{tr}}} \text{CE}(y_i, Wx_i) + R(w,\lambda),$$

where c is the number of classes (10 for MNIST), CE is the cross entropy loss, $(x_i, y_i)_{1\leq i\leq n_{\text{tr}}+n_{\text{val}}} \in (\mathbb{R}^d \times \{1,\dots,c\})^{n_{\text{tr}}+n_{\text{val}}}$ are training and validation examples, and $R(w,\lambda)$ is set according to either of the two situations below

- *one regularization parameter*:

  $R(w,\lambda) = \frac{\lambda}{2}\|w\|^2$, $\lambda \in \mathbb{R}_{++}$

- *multiple regularization parameters (one per feature)*:

  $R(w,\lambda) = \frac{1}{2}\sum_{i=1}^{c}\sum_{j=1}^{d}\lambda_j w_{ij}^2$ where $\lambda \in \mathbb{R}^d_{++}$.

In this scenario we take into account the whole MNIST training set containing 60 thousand examples, which we split in half to make the train and validation sets, i.e. $n_{\text{tr}} = n_{\text{val}} = 30000$. The batch size for the stochastic variants is 300. Figure 4.3 shows the results. Even in this setting, the pure stochastic variants have a clear advantage

over the Batch algorithm. We also note that the mixed strategies perform worse than the pure stochastic strategies and that there is no particular gain in allocating more epochs to solve the lower-level problem.

4.8.2 Bilevel Optimization on Twenty Newsgroups

Here we replicate the setting of Chapter 3 where multiple regularization parameters are optimized on the twenty newsgroup dataset. In particular, the lower-level objective is the $\mathcal{L}_2$ regularized cross-entropy loss with one regularization parameter per feature computed on the training set, while the upper-level objective is the unregularized cross-entropy loss computed on the validation set.

Differently from the previous experiments, which focused only on hypergradients, in this case we address the problem of minimizing the upper-level objective $f(\lambda)$. To minimize $f(\lambda)$ we use the SGD optimizer provided by PyTorch setting the learning rate to 10^3. The approximate hypergradient is provided by one of the methods in Table 4.1 with a total budget of 20 epochs, meaning that each method exploits approximately 20 times the number of examples in the training set to compute the hypergradient. Similarly to the experiment in Chapter 3, we also warm-start the lower-level problem with the solution found at the previous upper-level iteration, which significantly improves the performance. We note that each method starts by computing an approximation of $w(\lambda_0)$ which may provide different values of the considered metrics, even at the beginning of the procedure (see Figure 4.4).

We halve the original training set to generate the training and validation sets, i.e. $n_{\text{tr}} = n_{\text{val}} = 5657$, and we use mini-batches of dimension 50 for the stochastic variants. Moreover, we use the provided test set to compute the test performance metrics.

The performances varying the number of upper-level iterations are shown in Figure 4.4. We can see that the pure stochastic variants outperform both the Batch and mixed methods. Furthermore, using the same number of epochs to solve the lower-level problem and the linear system appears to be the best strategy. In Table 4.2 we present the performance of the three main methods after completion of the bilevel optimization procedure.

Table 4.2: Final performance metrics on the twenty newsgroup dataset, averaged over 5 trials. The metrics for the first three rows are obtained after 100 iterations of SGD on the upper-level objective. The last row is the result for the conjugate gradient method obtained in Chapter 3 where we select the best upper-level learning rate and perform 500 upper-level iterations.

Method	upper-level iter.	val. loss	test acc. (%)
Batch $k = t = 10$	100	1.30	57.5
Stoch dec. $k = t = 1131$	100	0.92	64.1
Stoch const. $k = t = 1131$	100	0.91	64.1
Batch $k = t = 10$	500	0.93	63.7

4.9 Discussion

In this chapter we studied a stochastic method for the approximation of the hypergradient in bilevel problems defined through a fixed-point equation of a contraction mapping. Specifically, we presented a stochastic version of the approximate implicit differentiation technique (AID), which is one of the most effective solutions for hypergradient computation as shown in Chapter 3. Our strategy (SID) estimates the hypergradient with the aid of two stochastic solvers in place of the deterministic solvers used in AID, and can use large mini-batches to estimate ∇E and $\partial_2 \Phi$. We presented a formal description and a theoretical analysis of SID, ultimately providing a bound for the mean square error of the corresponding hypergradient estimator. As a byproduct of the analysis, we provided an extension of the SGD algorithm for stochastic fixed-point equations. We have also conducted numerical experiments which confirm that using stochastic instead of deterministic solvers in SID can indeed yield a more accurate hypergradient approximation.

We believe that our analysis of stochastic fixed-point algorithms can be further extended to include variance reduction strategies and other advances commonly used for SGD. A good starting point for this extension can be the work by Gorbunov et al. (2020), which provides a unified theory for SGD methods in the strongly convex setting.

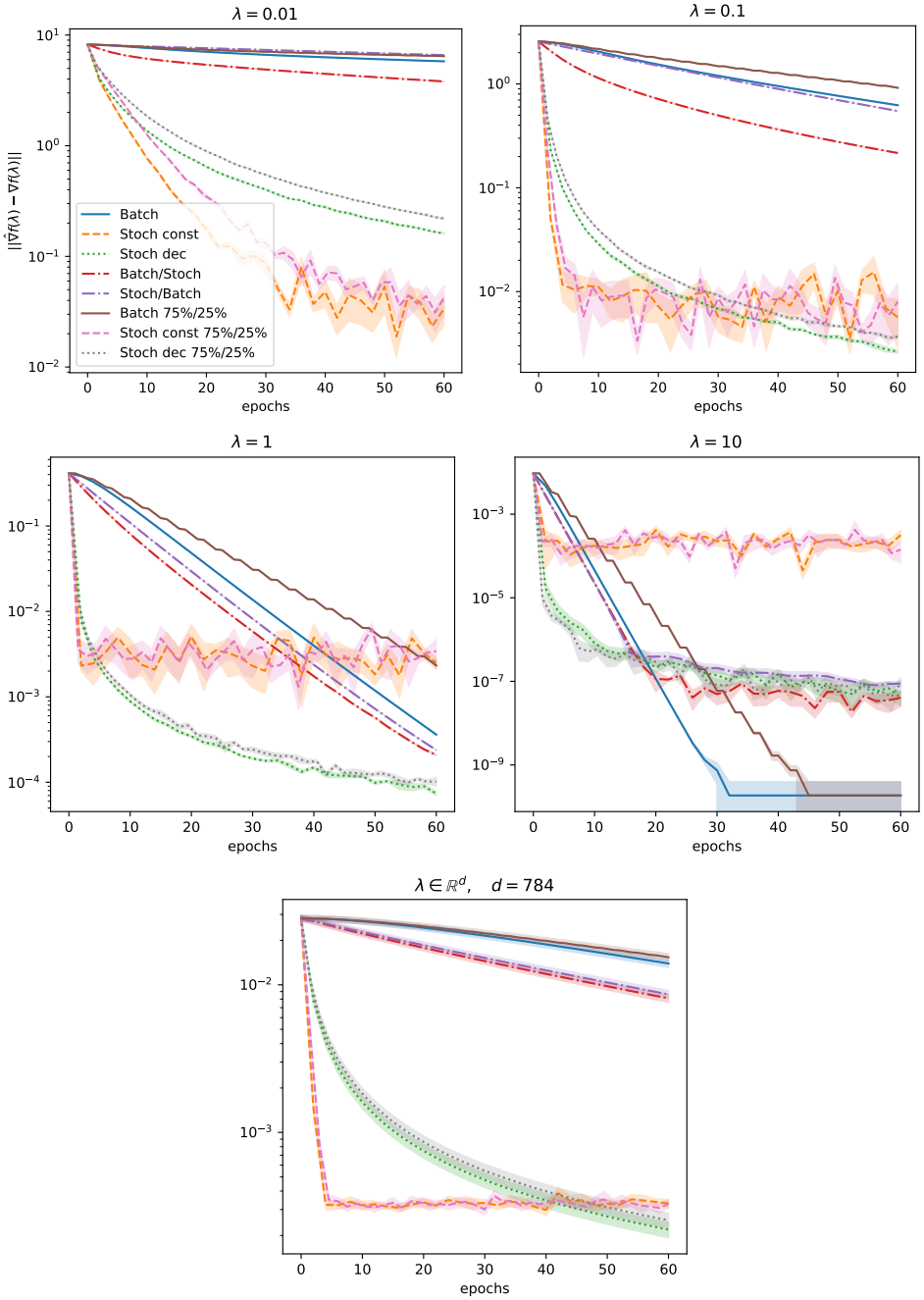

Figure 4.3: Experiments with a single (first 4 images) and multiple (last image) regularization parameters. The plots show mean (solid lines) and std (shaded regions) over 5 (first 4 images) and 10 (last image) runs. Each run varies the train/validation splits and, for the stochastic methods, the order and composition of the mini-batches. In addition, for each run in the last image, $\lambda_i = e^{\varepsilon_i}$, where $\varepsilon_i \sim \mathcal{U}[-2,2]$ for every $i \in \{1,\dots,d\}$. All methods use the same total computational budget. The first five use the same total number of epochs for solving the lower-level problem and the associated linear system. Whereas the last three methods – labeled with $75\%/25\%$ – dedicate $3/4$ of epochs to solve the lower-level problem and only $1/4$ for the linear system.

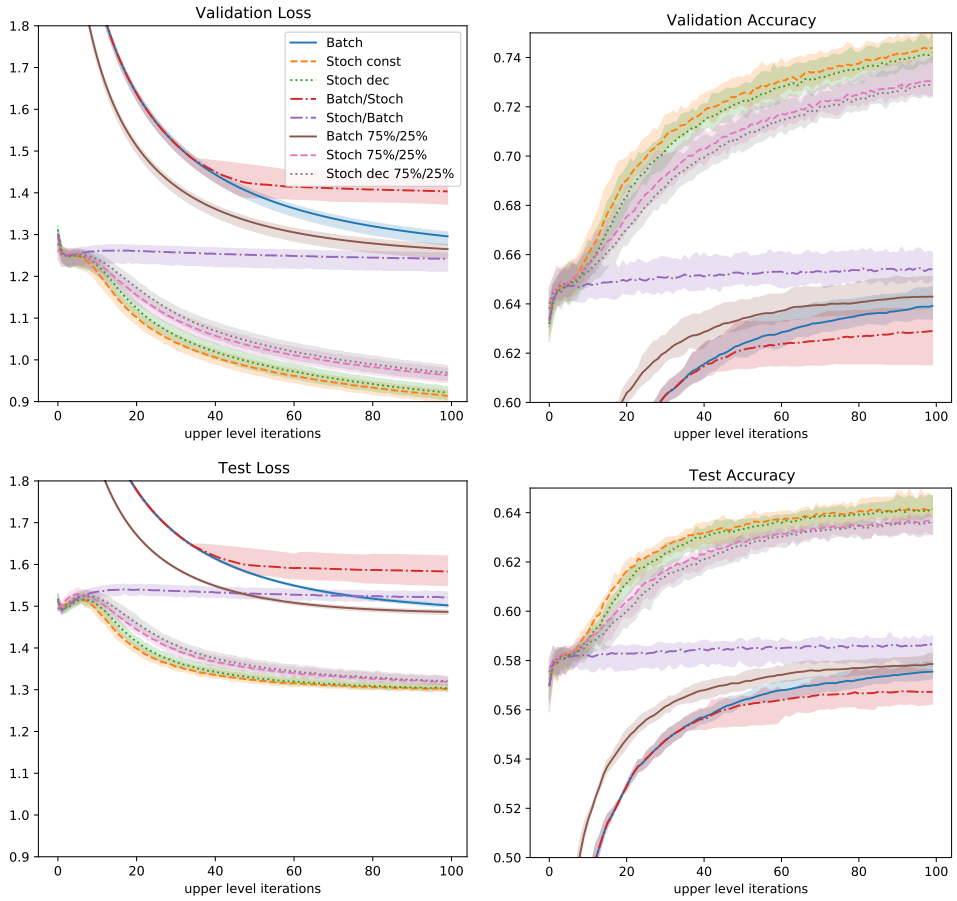

Figure 4.4: Performance metrics for multinomial logistic regression on twenty newsgroups. All methods compute the hypergradient in 20 epochs: methods labeled as 75%/25% compute the lower-level solution in 15 epochs and the solution for the linear system in 5, while the others solve both problems in 10 epochs. The plots show mean (solid line) and max-min (shaded region) over 5 runs varying both the train validation split and the mini-batch sampling of the stochastic algorithms. The starting point is the same for all methods and is set to $\lambda_0 = 0$ as in Chapter 3.

Chapter 5

Optimal Sample Complexity for a Gradient-Based Bilevel Method without Warm-Start

5.1 Introduction

In this chapter, as in Chapter 4, we consider the following stochastic bilevel problem

$$\begin{aligned} &\min_{\lambda\in\Lambda} f(\lambda) := \mathbb{E}[\hat{E}(w(\lambda),\lambda,\xi)] \\ &\text{with } \; w(\lambda) = \mathbb{E}[\hat{\Phi}(w(\lambda),\lambda,\zeta)], \end{aligned} \tag{5.1}$$

where $\Lambda \subseteq \mathbb{R}^n$ is closed and convex, $\hat{E}\colon \mathbb{R}^d \times \Lambda \times \Xi \to \mathbb{R}$ and $\hat{\Phi}\colon \mathbb{R}^d \times \Lambda \times Z \to \mathbb{R}^d$, and ξ and ζ are two independent random variables with values in Ξ and Z, respectively. In the following we refer to the problem of finding the fixed point $w(\lambda)$ of (5.1) as the *lower-level* (LL) problem, whereas we call the *upper-level* (UL) problem, that of minimizing f. We also define

$$E(w,\lambda) := \mathbb{E}[\hat{E}(w,\lambda,\xi)], \quad \Phi(w,\lambda) := \mathbb{E}[\hat{\Phi}(w,\lambda,\xi)],$$

and we assume that $\Phi(\cdot,\lambda)$ is a contraction, i.e. Lipschitz continuous with Lipschitz constant less than one. An important special case of the LL problem in (5.1), which

is the one usually considered in the related literature, is when

$$w(\lambda) = \operatorname*{argmin}_{w \in \mathbb{R}^d} \mathbb{E}[\hat{\mathcal{L}}(w, \lambda, \zeta)]. \tag{5.2}$$

In this case, provided that the objective $\mathcal{L}(w, \lambda) := \mathbb{E}[\hat{\mathcal{L}}(w, \lambda, \zeta)]$ is strongly convex and Lipschitz smooth, there always exists a sufficiently small $\eta > 0$ such that the gradient descent map

$$\Phi(w, \lambda) := w - \eta \nabla_1 \mathcal{L}(w, \lambda), \tag{5.3}$$

is a contraction with respect to w.

In dealing with Problem (5.1), we analyse gradient-based methods which exploit approximations of the hypergradient, i.e. the gradient of f in (5.1). As shown in Chapter 3, the contraction assumption guarantees that $\Phi(\cdot, \lambda)$ has a unique fixed point $w(\lambda)$ and the hypergradient, thanks to the implicit function theorem (Lang, 2012, Theorem 5.9), always exists and is given by

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda), \lambda) + \partial_2 \Phi(w(\lambda), \lambda)^\top v(w(\lambda), \lambda), \tag{5.4}$$

where $v(w, \lambda)$ is the solution of the linear system

$$(I - \partial_1 \Phi(w, \lambda)^\top) v = \nabla_1 E(w, \lambda), \tag{LS}$$

which is given by $v(w, \lambda) := \left(I - \partial_1 \Phi(w, \lambda)^\top\right)^{-1} \nabla_1 E(w, \lambda)$.

Computing the hypergradient exactly can be impossible or very expensive since it requires to compute the LL and LS solutions $w(\lambda)$ and $v(w(\lambda), \lambda)$. This is especially true in large-scale machine learning applications where the number of UL and LL parameters m and d can be very large. Furthermore, in cases such as hyperparameter optimization, where E is the average loss over the validation set while Φ is defined in (5.3) with $\mathcal{L}$ being the loss over the training set, if the dataset is large, E, Φ and their derivatives can become very expensive to compute. For this reason, relying on stochastic estimators ($\hat{E}$ and $\hat{\Phi}$) using only a mini-batch of examples becomes crucial for devising scalable methods.

To address these issues, *approximate implicit differentiation* (AID) methods (Pedregosa, 2016; Rajeswaran et al., 2019; Lorraine et al., 2020), compute the hypergradient by using approximate solutions for the LL and LS problems. *Iterative differentiation methods* (ITD) (Maclaurin et al., 2015; Franceschi et al., 2017, 2018; Finn et al., 2017) instead directly differentiate the lower-level solver. The convergence of those methods to the true hypergradient has been addressed in Chapter 3 for AID and ITD methods in the deterministic case and in Chapter 4 for stochastic AID methods.

By contrast, in this chapter we study the convergence rate of a full bilevel procedure to solve Problem (5.1), based on SID: the general AID stochastic method presented in Chapter 4. Such type of study was started by Ghadimi and Wang (2018) and was later followed by several works which we discuss more in detail in Section 5.3. Concerning ITD-based methods, we note that similar results were proved only in the deterministic setting (Ji et al., 2021, 2022).

Warm-start. A common procedure to improve the overall performance of bilevel algorithms is that of using as a starting point for the LL (or LS) solver at the current UL iteration, the LL (or LS) approximate solution found at the previous UL iteration (Hong et al., 2020; Guo and Yang, 2021; Huang and Huang, 2021; Chen et al., 2021). This strategy, which is called *warm-start*, reduces the number of LL (or LS) iterations needed by the bilevel procedure and is thought to be fundamental to achieve the optimal sample complexity (Arbel and Mairal, 2021). Moreover, warm-start is sometimes accompanied by the use of *large mini-batches* (Ji et al., 2021; Arbel and Mairal, 2021), i.e. averages of many samples, to estimate gradients or Jacobians. Large mini-batches allow to reduce the number of UL iteration but increase the cost per iteration and ultimately achieve the same sample complexity up to log terms.

In spite of the above advantages, warm-start presents a major downside: it is not suitable in applications where it is expensive to store the whole LL solution, such as meta-learning. Indeed, meta-learning consists in leveraging "common properties" between a set of learning tasks in order to facilitate the learning process.

We consider a *meta-training* set of T tasks. Each task $i \in \{1,\dots,T\}$ relies on a training and a validation set which we denote by D_i^{tr} and D_i^{val}, respectively. The meta-learning optimization problem is a bilevel problem where the UL objective has the form $f(\lambda) = \sum_{i=1}^{T} f_i(\lambda)$ with $f_i(\lambda) := \mathcal{L}(w^i(\lambda), \lambda; D_i^{\text{val}})$ and the LL solution can be written as

$$w(\lambda) = \operatorname*{argmin}_{w \in \mathbb{R}^{T\times d}} \sum_{i=1}^{T} \mathcal{L}(w^i, \lambda; D_i^{\text{tr}}), \tag{5.5}$$

where $\mathcal{L}$, λ and w^i (the i-th row of w) are the loss function, the meta-parameters, and task-specific parameters of the i-th task, respectively. For example, in (Franceschi et al., 2018) w^i and λ are the parameters of the last linear layer and the representation part of a neural network, respectively. Note that the minimization in (5.5) can be performed separately for each task. Therefore, when T is large, a common strategy is that of solving, at each UL iteration, only a small random subset of tasks.

In this context using warm-start is problematic. Indeed, if task j is sampled at iteration s, applying warm-start consistently would require using, as a starting point for the LL optimization, the solution for that same task j at iteration $s-1$. However, the task j might not be among the sampled tasks at iteration $s-1$. A possible remedy would be to warm-start by using the last available approximate solution of the LL problem for task j, which may have been computed too many iterations before the current one, ultimately making the warm-start procedure ineffective (see experiments in Section 5.6.2). In addition, the above strategy would need to keep the approximate solutions for all the previous tasks in memory and eventually for all the T tasks, which might be too costly when T and d are large. Indeed, in Section 5.6.2 we consider a problem in which the variable w occupies 122 GB of memory. Moreover, from the theoretical point of view, this requires a novel analysis to handle the related delays. This discussion suggests that the warm-start strategy currently considered in literature is not well suited for meta-learning, and indeed is never used in meta-learning experiments.

We note that similar issues arise also for equilibrium models when dealing with large datasets. Indeed, in the bilevel formulation of equilibrium models (see Chapter 3) the LL problem consists in finding a fixed point representation for each

training example and ultimately yields a separable structure as in meta-learning.

Contributions. In this chapter we show that a bilevel procedure that does not rely on warm-start can achieve optimal sample complexity, improving that by Ghadimi and Wang (2018). Specifically, we make the following contributions.

- *We analyse the sample complexity of the bilevel procedure in Algorithm 5.2.2 (BSGM) which combines projected inexact gradient descent with the hypergradient estimator computed via SID, i.e. the procedure described in Chapter 4.* In particular, we prove, without any convexity assumptions on f, that BSGM achieves the optimal and near-optimal sample complexities of $O(\varepsilon^{-2})$ (with a finite horizon) and $\tilde{O}(\varepsilon^{-2})$, to reach an ε-stationary point of Problem (5.1). In addition, it obtains near-optimal complexity of $\tilde{O}(\varepsilon^{-1})$ for the deterministic case. We stress that these results are achieved without warm-start, although with a reasonable additional assumption (see Remark 5.4.4(iv) and Remark 5.5.9).

- *We extend previous theoretical analyses by considering the more general case where the LL problem is a fixed-point equation instead of a minimization problem and relaxing some of the assumptions.* In particular, we cover the case where λ is subject to constraints (i.e. when $\Lambda \neq \mathbb{R}^m$), which are often needed to satisfy the other assumptions of the analysis, but neglected by some of the previous works. We also extend the scope of applicability of the method by including e.g. non-Lipschitz LL losses, like the square loss, in problems of type (5.2).

- *We evaluate the empirical performance of our method against other methods using warm-start on three instances of the bi-level problem* (5.1). Specifically, we provide experiments on equilibrium models and meta-learning showing that warm-start is ineffective and increases the memory cost. We also perform a data poisoning experiment which shows that warm-start can be beneficial, although our method remains competitive. We provide the code at https://github.com/CSML-IIT-UCL/bioptexps

Organization. In Section 5.2 we describe the bilevel procedure. We discuss closely related works in Section 5.3. In Section 5.4 we state our assumptions and some properties of the bilevel problem. In Section 5.5 we first study the convergence of the projected inexact gradient method with controllable mean square error on the gradient, and then combine this analysis with the one in Section 4.4 to derive the desired complexity results for Algorithm 5.2.2. We present the experiments in Section 5.6.

5.2 Bilevel Stochastic Gradient Method (BSGM)

We study the simple double-loop procedure in Algorithm 5.2.2 (BSGM). BSGM uses projected inexact gradient updates for the UL problem, where the (biased) hypergradient estimator is provided by Algorithm 5.2.1 (SID), which is a specialization of Algorithm 4.3.1 in Chapter 4, with stochastic fixed point iterations used to solve the LL and LS problems. SID computes the hypergradient by first solving the LL problem (Step 1), then it computes the estimator of the partial gradients of the UL function E using mini-batches of size J (Step 2). After this it computes an approximate solution to the LS (Step 3). Finally, it combines the LL and LS solutions together with min-batch estimators of $\nabla_2 E$ and $\partial_2 \Phi$ computed using a mini-batch of size J to give the final hypergradient estimator (Step 4). We remark that the samplings performed at all the four steps have to be mutually independent. Moreover, to solve the LL and LS problems we use simple stochastic fixed-point iterations which reduce to stochastic gradient descent in LL problems of type (5.2). We use the same sequence of step sizes η_i for both the LL and LS solvers and the same batch size J for both ∇E and $\partial_2 \Phi$ to simplify the analysis and to reduce the number of configuration parameters of the method. While this choice still achieves optimal sample complexity, it may be suboptimal in practice.

5.3 Comparison with Related Work

Several gradient-based algorithms, together with sample complexity rates have been recently introduced for stochastic bilevel problems with LL of type (5.2). They all follow a structure similar to Algorithm 5.2.2, where each UL update uses one (or

Algorithm 5.2.1 Stochastic Implicit Differentiation (SID)

Requires: $t, k, J, \lambda, w_0, (\eta_i)_{i=0}^{\infty}$.

1. **LL Solver:**

$$\begin{array}{l} \text{for } i = 0, 1, \ldots t-1 \\ \quad w_{i+1}(\lambda) = w_i(\lambda) + \eta_i(\hat{\Phi}(w_i(\lambda), \lambda, \zeta_i) - w_i(\lambda)) \end{array} \tag{5.6}$$

   where $(\zeta_i)_{0 \le i \le t-1}$ are i.i.d. copies of ζ.

2. Compute $\nabla_i \bar{E}_J(w_t(\lambda), \lambda) = \frac{1}{J} \sum_{j=1}^{J} \nabla_i \hat{E}(w_t(\lambda), \lambda, \xi_j)$, where $(\xi_j)_{1 \le j \le J}$ are i.i.d. copies of ξ and $i \in \{1, 2\}$.

3. **LS Solver:**

$$\begin{array}{l} \text{for } i = 0, 1, \ldots k-1 \\ \quad \left\lfloor \begin{array}{l} v_{i+1}(w_t(\lambda), \lambda) = v_i(w_t(\lambda), \lambda) + \eta_i(\hat{\Psi}_{w_t(\lambda)}(v_i(w_t(\lambda), \lambda), \lambda, \hat{\zeta}_i) \\ \qquad\qquad -v_i(w_t(\lambda), \lambda)) \end{array} \right. \end{array} \tag{5.7}$$

   where $\hat{\Psi}_w(v, \lambda, z) := \partial_1 \hat{\Phi}(w, \lambda, z)^\top v + \nabla_1 \bar{E}_J(w, \lambda)$, $(\hat{\zeta}_i)_{0 \le i \le k-1}$ are i.i.d. copies of ζ.

4. Compute the approximate hypergradient as

$$\hat{\nabla} f(\lambda) := \nabla_2 \bar{E}_J(w_t(\lambda), \lambda) + \partial_2 \bar{\Phi}_J(w_t(\lambda), \lambda)^\top v_k(w_t(\lambda), \lambda).$$

   where $\partial_2 \bar{\Phi}_J(w_t(\lambda), \lambda) = \frac{1}{J} \sum_{j=1}^{J} \partial_2 \hat{\Phi}(w_t(\lambda), \lambda, \zeta_j')$ and $(\zeta_j')_{1 \le j \le J}$ are i.i.d. copies of ζ.

Algorithm 5.2.2 Bilevel Stochastic Gradient Method (BSGM)

Requires: $\lambda_0, w_0, \alpha, \{\eta_j\}, \{t_s\}, \{J_s\}$.
for $s = 0, 1, \ldots, S-1$

1. Compute $\hat{\nabla} f(\lambda_s)$ using Algorithm 5.2.1 (SID) with $t = t_s, k = t_s, J = J_s, \lambda = \lambda_s$, $\{\eta_i\} = \{\eta_j\}$, and $w_0 = w_0$, $v_0 = 0$ (no warm-start).
2. $\lambda_{s+1} = P_\Lambda(\lambda_s - \alpha \hat{\nabla} f(\lambda_s))$

more for variance reduction methods) hypergradient estimator computed using a variant of Algorithm 5.2.1 with different LL and LS solvers. The algorithms mainly differ in how they compute the LL, LS and UL updates (e.g. in the choice of the step sizes $\eta_{t,s}, \alpha_s$, mini-batch sizes, and whether they use variance reduction techniques),

Table 5.1: Sample complexity (**SC**) of stochastic bilevel optimization methods for finding an ε-stationary point of Problem (5.1) with LL of type (5.2). **BS-LL** is the LL mini-batch size, i.e. the one used to approximate Φ in the LL solver. **WS** indicates the use of warm-start, e.g. Y, N means that warm-start is used for the LL problem but not for the LS. t_s and k_s denote the number of iterations for the LL and LS problems respectively, while α_s and $\eta_{t,s}$ are the stepsize respectively for the UL and LL problems at the s-th UL iteration and t-th LL iteration. L_f is the Lipschitz constant of ∇f, S is the total number of UL iteration and ESI means that the LS estimator is given by an exact single sample LL hessian inverse. The last 7 results are obtained under additional expected smoothness assumptions (Arjevani et al., 2022).

Algorithm	**SC**	**BS-LL**	**WS**	t_s	k_s	α_s	$\eta_{t,s}$
BSA (Ghadimi and Wang, 2018)	$O(\varepsilon^{-3})$	$\Theta(1)$	N, N	$\Theta(\sqrt{s})$	$\Theta(\log(\sqrt{s}))$	$\Theta(1/\sqrt{S})$	$\Theta(1/t)$
TTSA (Hong et al., 2020)	$\tilde{O}(\varepsilon^{-2.5})$	$\Theta(1)$	Y, N	1	$\Theta(\log(\sqrt{s}))$	$\Theta(S^{-2/5})$	$\Theta(S^{-3/5})$
stocBiO (Ji et al., 2021)	$\tilde{O}(\varepsilon^{-2})$	$\Theta(S)$	Y, N	$\Theta(1)$	$\Theta(\log(\sqrt{s}))$	$\leq 1/4L_f$	$\Theta(1)$
SMB (Guo et al., 2021)	$\tilde{O}(\varepsilon^{-2})$	$\Theta(1)$	Y, N	1	$\Theta(\log(\sqrt{s}))$	$\Theta(1/\sqrt{S})$	$\Theta(1/\sqrt{S})$
saBiAdam (Huang and Huang, 2021)	$\tilde{O}(\varepsilon^{-2})$	$\Theta(1)$	Y, N	1	$\Theta(\log(\sqrt{s}))$	$\Theta(1/\sqrt{s})$	$\Theta(1/\sqrt{s})$
ALSET (Chen et al., 2021)	$\tilde{O}(\varepsilon^{-2})$	$\Theta(1)$	Y, N	1	$\Theta(\log(\sqrt{S}))$	$\Theta(1/\sqrt{S})$	$\Theta(1/\sqrt{S})$
Amigo (Arbel and Mairal, 2021)	$O(\varepsilon^{-2})$	$\Theta(S)$	Y, Y	$\Theta(1)$	$\Theta(1)$	$\leq 1/L_f$	$\Theta(1)$
BSGM-1 (Ours)	$\tilde{O}(\varepsilon^{-2})$	$\Theta(1)$	N, N	$\Theta(s)$	$\Theta(s)$	$\leq 1/L_f$	$\Theta(1/t)$
BSGM-2 (Ours)	$O(\varepsilon^{-2})$	$\Theta(1)$	N, N	$\Theta(S)$	$\Theta(S)$	$\leq 1/L_f$	$\Theta(1/t)$
STABLE (Chen et al., 2022)	$O(\varepsilon^{-2})$	$\Theta(1)$	Y, N	1	ESI	$\Theta(1/\sqrt{S})$	$\Theta(1/\sqrt{S})$
FSLA (Li et al., 2022)	$O(\varepsilon^{-2})$	$\Theta(1)$	Y, Y	1	1	$\Theta(1/\sqrt{s})$	$\Theta(1/\sqrt{s})$
STABLE-VR (Guo and Yang, 2021)	$\tilde{O}(\varepsilon^{-1.5})$	$\Theta(1)$	Y, N	1	ESI	$\Theta(s^{-1/3})$	$\Theta(s^{-1/3})$
SUSTAIN (Khanduri et al., 2021)	$\tilde{O}(\varepsilon^{-1.5})$	$\Theta(1)$	Y, N	1	$\Theta(\log(\sqrt{s}))$	$\Theta(s^{-1/3})$	$\Theta(s^{-1/3})$
VR-saBiAdam (Huang and Huang, 2021)	$\tilde{O}(\varepsilon^{-1.5})$	$\Theta(1)$	Y, N	1	$\Theta(\log(\sqrt{s}))$	$\Theta(s^{-1/3})$	$\Theta(s^{-1/3})$
MRBO (Yang et al., 2021)	$\tilde{O}(\varepsilon^{-1.5})$	$\Theta(1)$	Y, N	1	$\Theta(\log(S))$	$\Theta(s^{-1/3})$	$\Theta(s^{-1/3})$
VRBO (Yang et al., 2021)	$\tilde{O}(\varepsilon^{-1.5})$	$\Theta(\sqrt{S})$	Y, N	$\Theta(1)$	$\Theta(\log(\sqrt{S}))$	$\Theta(1)$	$\Theta(1)$

in the number of LL and LS iterations t_s, k_s, and in the use of warm-start. These differences are summarized in Table 5.1.

Ghadimi and Wang (2018) introduce the first convergence analysis for a simple double-loop procedure, both in the deterministic and stochastic settings. Their algorithm uses (stochastic) gradient descent both at the upper and lower levels (SGD-SGD) and approximates the LS solution using an estimator based on truncated Neumann series (with k_s elements). In the stochastic setting, this procedure needs $O(\varepsilon^{-3})$ samples to reach an ε-stationary point. This sample complexity is achieved by increasing the number of LL and LS iterations, i.e. at the s-th UL iteration it sets $t_s = \Theta(\sqrt{s})$ and $k_s = \Theta(\log(\sqrt{s}))$.

Differently from this seminal work, all subsequent ones warm-start the LL problem to improve the sample complexity, since this allows them to choose $t_s = \Theta(1)$ or even $t_s = 1$, the latter case is also referred to as *single-loop*. Warm-start combined

with the simple SGD-SGD strategy can improve the $O(\varepsilon^{-3})$ sample complexity by carefully selecting the UL and LL stepsize, i.e. using two timescale (Hong et al., 2020) or single timescale (Chen et al., 2021) stepsizes, or by employing larger and ε-dependent mini-batches (Ji et al., 2021). Warm-starting also the LS can further improve the sample-complexity to $O(\varepsilon^{-2})$ (Arbel and Mairal, 2021). The complexity $O(\varepsilon^{-2})$ is optimal, since the optimal sample complexity of methods using unbiased stochastic gradient oracles with bounded variance on smooth functions is $\Omega(\varepsilon^{-2})$, and this lower bound is also valid for bilevel problems of type (5.1)[1] (also with LL of type (5.2)).

Chen et al. (2022); Khanduri et al. (2021); Guo and Yang (2021); Huang and Huang (2021) achieve the best-known sample complexity of $\tilde{O}(\varepsilon^{-1.5})$ using variance reduction techniques[2]. Li et al. (2022) introduce the first fully single loop algorithm where both the LL and LS are warm-started and solved with one iteration, although it achieves a sample complexity of $O(\varepsilon^{-2})$ while using variance reduction. Variance reduction techniques require additional algorithmic parameters and need expected smoothness assumptions to guarantee convergence (Arjevani et al., 2022). Furthermore, they increase the cost per iteration compared to the SGD-SGD strategy since they require two stochastic samples per iteration to estimate gradients instead of one. For these reasons, we do not investigate these kinds of techniques in the present work.

Except for Chen et al. (2022); Guo and Yang (2021), all methods discussed in this section and ours are also computationally efficient, since they only require gradients and Hessian-vector products. Hessian-vector products have a cost comparable to gradients thanks to automatic differentiation. Chen et al. (2022); Guo and Yang (2021) further rely on operations like inversions and projections of the LL Hessian. These can be too costly with a large number (d) of LL variables, which can make it impractical even to compute the full Hessian.

All the aforementioned works study smooth bilevel problems with LL of

[1]We can easily see this when $E(w,\lambda) = g(\lambda)$ and $\hat{E}(w,\lambda,\xi) = \hat{g}(\lambda,\xi)$ where $g : \Lambda \mapsto \mathbb{R}$ is Lipschitz smooth and $\hat{g}$ is an unbiased estimate of g whose gradient w.r.t. λ has bounded variance.

[2]Chen et al. (2022) uses variance reduction only on the LL Hessian updates (see eq. (12)).

type (5.2) and with a twice differentiable and strongly convex LL objective. At last, we mention two lines of work which consider different bilevel formulations: (Bertrand et al., 2020, 2022a), which study the error of hypergradient approximation methods for certain non-smooth bilevel problems, and (Liu et al., 2020, 2022; Arbel and Mairal, 2022), which analyse algorithms to tackle bilevel problems with more than one LL solution.

The sample complexity improvement that our method achieves compared to Ghadimi and Wang (2018), i.e. from $O(\varepsilon^{-3})$ to $O(\varepsilon^{-2})$, is possible because our hypergradient estimator (SID) uses mini-batches of size $\Theta(\varepsilon^{-1})$ (instead of $\Theta(1)$) to estimate ∇E and $\partial_2\Phi$ and a stochastic solver with decreasing step-sizes (instead of the truncated Neumann series inverse estimator) also to solve the LS problem (similar to the LL solver). This allows SID to have $O(\varepsilon^{-1})$ mean squared error (see Corollary 4.6.2), while the hypergradient estimator in Ghadimi and Wang (2018) achieves $O(\varepsilon^{-1})$ only for the bias, while the variance does not vanish. Consequently, we can use a more aggressive UL step-size (constant instead of decreasing), which reduces the number of UL iterations from $O(\varepsilon^{-2})$ to $O(\varepsilon^{-1})$.

Among the methods using warm-start, *Amigo* (Arbel and Mairal, 2021) is the most similar to ours. Indeed, it achieves the same $O(\varepsilon^{-2})$ optimal sample complexity as BSGM. Also, the number of UL iterations and the size of the mini-batch to estimate ∇E and $\partial_2\Phi$ is $O(\varepsilon^{-1})$, as for our method. The main differences with respect to BSGM are in the use of (i) the warm-start procedure in the LL and LS problems, which in general decreases the complexity, (ii) mini-batch sizes of the order of $\Theta(\varepsilon^{-1})$ to estimate Φ (in the LL), $\partial_1\Phi$ (in the LS), which increase the complexity, contrasting with our choice of taking just one sample for estimating the same quantities. Overall, (i)-(ii) balance out and ultimately give the same total complexity.

We note that our improvement over point (ii) is necessary to achieve the optimal sample complexity. Indeed, if one istead carries out the analysis by using (ii), constant step-sizes for the LS and LL, and setting $k_s, t_s = \Theta(\log(S))$, only suboptimal complexity of $O(\varepsilon^{-2}\log(\varepsilon^{-1}))$ is achieved, because mini-batches of size $\Theta(\varepsilon^{-1})$

are used $2S(1+\log(S))$ (instead of just $2S$) times in S UL iterations.

For the deterministic case, we improve the rate of Ghadimi and Wang (2018) from $O(\varepsilon^{-5/4})$ to $O(\varepsilon^{-1}\log(\varepsilon^{-1}))$ by setting $t_s = \Theta(\kappa\log(s))$ (and also k_s) instead of $t_s = \lceil (s+1)^{1/4}/2 \rceil$, where $\kappa = (1-q)^{-1}$ and q is the contraction constant defined in Assumption 5.4.1(i). Ji et al. (2021); Arbel and Mairal (2021) have an improved complexity of $O(\varepsilon^{-1})$, obtained by using warm-start and setting $t_s, k_s = \tilde{\Theta}(\kappa)$, where κ corresponds to the LL condition number.

Finally, note that warm-start makes it possible to set t_s and k_s with no dependence on ε both in the deterministic and stochastic settings, improving the sample complexity (by removing a log factor) in the former case. However, in the stochastic case the complexity does not improve because solving the LL and LS problems cannot have lower complexity than $O(\varepsilon^{-1})$, which is that of the sample mean estimation error. Such complexity is already achieved by our stochastic fixed-point iteration solvers with decreasing step-sizes and no warm-start.

5.4 Assumptions and Preliminary Results

We hereby state all the assumptions used for the analysis, discuss them and outline in a lemma some useful smoothness properties of the bilevel problem. Note that differently from Chapters 3 and 4 here we have assumptions holding uniformly for every $\lambda \in \Lambda$. Therefore, when such assumptions are satisfied, the constants might be much larger than in the assumptions of the previous chapters.

Assumption 5.4.1. *The set $\Lambda \subseteq \mathbb{R}^m$ is closed and convex and the mappings $\Phi\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}^d$ and $E\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}$ are differentiable in an open set containing $\mathbb{R}^d \times \Lambda$. For every $\lambda \in \Lambda$:*

(i) $\Phi(\cdot,\lambda)$ *is a contraction, i.e.,* $\|\partial_1\Phi(w,\lambda)\| \le q$ *for* $q < 1$ *and* $\forall w \in \mathbb{R}^d$.

(ii) $\|\partial_i\Phi(w(\lambda),\lambda) - \partial_i\Phi(w,\lambda)\| \le \nu_i\|w(\lambda)-w\|$ *for* $i \in \{1,2\}$, $\forall w \in \mathbb{R}^d$.

(iii) $\|\nabla_i E(w(\lambda),\lambda) - \nabla_i E(w,\lambda)\| \le \mu_i\|w(\lambda)-w\|$ *for* $i \in \{1,2\}$, $\forall w \in \mathbb{R}^d$.

(iv) $E(\cdot,\lambda)$ *is Lipschitz cont. on* $\mathbb{R}^d$ *with constant* L_E.

Assumption 5.4.2. *Let $w_0 : \Lambda \to \mathbb{R}^d$. For every $w^* \in \{w(\lambda) \,|\, \lambda \in \Lambda\}$, $\lambda \in \Lambda$:*

(i) $\nabla_1 E(w^*,\cdot), \nabla_2 E(w^*,\cdot)$ *are Lipschitz cont. with constants* $\bar{\mu}_1, \bar{\mu}_2$ *respectively.*

(ii) $\partial_1\Phi(w^*,\cdot)$, $\partial_2\Phi(w^*,\cdot)$ *are Lipschitz cont. with constants* $\bar{\nu}_1$, $\bar{\nu}_2$ *respectively.*

(iii) $\|w(\lambda) - w_0(\lambda)\| \leq B$ *for some* $B \geq 0$.

(iv) $\|\partial_2\Phi(w(\lambda),\lambda)\| \leq L_\Phi$ *for some* $L_\Phi \geq 0$.

Assumption 5.4.3. *The random variables* ζ *and* ξ *take values in measurable spaces* Ξ *and* Z *and* $\hat{\Phi} : \mathbb{R}^d \times \Lambda \times Z \mapsto \mathbb{R}^d$, $\hat{E} : \mathbb{R}^d \times \Lambda \times \Xi \mapsto \mathbb{R}$ *are measurable functions, differentiable w.r.t. the first two arguments in an open set containing* $\mathbb{R}^d \times \Lambda$*, and, for all* $w \in \mathbb{R}^d$, $\lambda \in \Lambda$*:*

(i) $\mathbb{E}[\hat{\Phi}(w,\lambda,\zeta)]=\Phi(w,\lambda)$, $\mathbb{E}[\hat{E}(w,\lambda,\xi)]=E(w,\lambda)$ *and we can exchange derivatives with expectations when taking derivatives on both sides.*

(ii) $\mathbb{V}[\hat{\Phi}(w,\lambda,\zeta)] \leq \sigma_1 + \sigma_2\|\Phi(w,\lambda) - w\|^2$ *for some* $\sigma_1, \sigma_2 \geq 0$.

(iii) $\mathbb{V}[\partial_1\hat{\Phi}(w,\lambda,\zeta)] \leq \sigma_1'$, $\mathbb{V}[\partial_2\hat{\Phi}(w,\lambda,\zeta)] \leq \sigma_2'$ *for some* $\sigma_1', \sigma_2' \geq 0$.

(iv) $\mathbb{V}[\nabla_1\hat{E}(w,\lambda,\xi)] \leq \hat{\sigma}_1$, $\mathbb{V}[\nabla_2\hat{E}(w,\lambda,\xi)] \leq \hat{\sigma}_{2,E}$ *for some* $\hat{\sigma}_1, \hat{\sigma}_{2,E} \geq 0$.

Assumptions 5.4.1, 5.4.2 and 5.4.3 are similar to the ones in (Ghadimi and Wang, 2018) and subsequent works, but extended to the bilevel fixed point formulation and sometimes weakened. Assumptions 5.4.1 and 5.4.3 are sufficient to obtain meaningful upper bounds on the mean square error of the SID estimator (Algorithm 5.2.1), while Assumption 5.4.2 enables us to derive the convergence rates of the bilevel procedure in Algorithm 5.2.2. The deterministic case can be studied by setting, in Assumption 5.4.3, $\sigma_1 = \sigma_2 = \sigma_1' = \sigma_2' = \hat{\sigma}_1 = \hat{\sigma}_{2,E} = 0$.

Remark 5.4.4.

(i) Although the majority of recent works set $\Lambda = \mathbb{R}^m$*, many bilevel problems satisfy the assumptions above only when* $\Lambda \neq \mathbb{R}^m$*. E.g., when* λ *is a scalar regularization parameter in the LL objective and* Φ *is the gradient descent map,* λ *has to be bounded from below away from zero for* $\Phi(\cdot,\lambda)$ *to always be a contraction (Assumption 5.4.1(i)). Also, when* Λ *and* $\{w_0(\lambda) \,|\, \lambda \in \Lambda\}$ *are bounded and closed, and Assumption 5.4.1(i) is satisfied, then 5.4.2(iii)(iv) are*

satisfied because $w(\cdot)$ is continuous in Λ. Our analysis directly considers the case $\Lambda \subseteq \mathbb{R}^m$, which includes the others.

(ii) The Lipschitz assumption on E (5.4.1(iv)) is needed to upper bound $\|\nabla_1 E(w_t(\lambda),\lambda)\|$. Otherwise, this is difficult to achieve since, in the stochastic setting, we have no control on the LL iterates $w_t(\lambda)$. This assumption can be relaxed in the deterministic case.

(iii) Assumption 5.4.2(iv) is weaker than the one commonly used in related works, which requires the partial Jacobian $\partial_2\Phi(w,\lambda)$ to be bounded uniformly on $\mathbb{R}^d \times \Lambda$. By contrast, we assume only the boundedness on the solution path $\{(w(\lambda),\lambda) \,|\, \lambda \in \Lambda\}$. This allows to extend to scope of applicability of the method. For example, when $\lambda \in [\lambda_{min}, \lambda_{max}]$ is the L_2-regularization parameter multiplying $(1/2)\|w\|^2$ in the LL objective, Φ is the gradient descent map and $w_0(\lambda) = 0$, then $\|\partial_2\Phi(w,\lambda)\| = \|w\|$ which is unbounded, while $\|\partial_2\Phi(w(\lambda),\lambda)\| = \|w(\lambda)\|$ is bounded since $w(\cdot)$ is differentiable (from 5.4.1(i)) and therefore continuous in $[\lambda_{min}, \lambda_{max}]$ which is a bounded and closed set.

(iv) Assumption 5.4.2(iii) uniformly bounds the distance of the LL solution $w(\lambda)$ from the starting point of the LL solver $w_0(\lambda)$. A similar assumption (with $w_0(\lambda) = 0$) is stated implicitly also in (Ghadimi and Wang, 2018) (See e.g. definition of M in eq. (2.28)). 5.4.2(iii) is not needed when using warm-start (see also Remark 5.5.9), although it is satisfied when Λ and $\{w_0(\lambda) \,|\, \lambda \in \Lambda\}$ are bounded and closed and 5.4.1(i) holds, but also in some cases where Λ is unbounded. For example in meta-learning, when λ is the bias in the LL regularization, i.e. $\Lambda = \mathbb{R}^d$, $\Phi(w,\lambda) = (1-\eta\gamma)w - \eta\nabla\mathcal{L}(w) + \eta\gamma\lambda$ with $\mathcal{L}$ L-smooth, $w_0(\lambda) = \lambda$ and $\eta > 0$ being the LL step-size, we have $w(\lambda) = \lambda - \gamma^{-1}\nabla\mathcal{L}(w(\lambda))$ which implies $\sup_{\lambda\in\mathbb{R}^d}\|w(\lambda)\| = \infty$ while $\sup_{\lambda\in\mathbb{R}^d}\|w(\lambda) - w_0(\lambda)\| \leq \gamma^{-1}L$.

(v) Assumption 5.4.3(ii) is more general than the corresponding one in (Ghadimi and Wang, 2018), which is a bound on the variance on the LL gradient estimator recovered by setting $\sigma_2 = 0$ and $\hat{\Phi}(w,\lambda,\xi) = w - \nabla_1\hat{\mathcal{L}}(w,\lambda,\xi)$

with $\nabla_1 \hat{\mathcal{L}}(w,\lambda,\xi)$ being an unbiased estimator of the LL gradient. Having $\sigma_2 > 0$ allows the variance to grow away from the fixed point, which occurs for example when the unregularized loss in the LL Problem (5.2) *is not Lipschitz (like for the square loss).*

Remark 5.4.5. *Variance reduction methods (Chen et al., 2022; Guo and Yang, 2021; Khanduri et al., 2021; Huang and Huang, 2021) require also an* expected smoothness assumption *on $\nabla\hat{E}$, $\hat{\Phi}$ and $\partial\hat{\Phi}$ (often satisfied in practice). See (Arjevani et al., 2022). A random function $g(\cdot,\xi)$, where ξ is the random variable, meets the expected smoothness assumption if $\mathbb{E}[\|g(x_1,\xi)-g(x_2,\xi)\|]^2 \leq \tilde{L}_g^2\|x_1-x_2\|^2$, for every x_1, x_2, where $\tilde{L}_g \geq 0$.*

The existence of the hypergradient $\nabla f(\lambda)$ is guaranteed by the fact that Φ and E are differentiable and that $\Phi(\cdot,\lambda)$ is a contraction (Assumption 5.4.1(i)). Furthermore, we have the following properties for the bilevel problem.

Lemma 5.4.6 (**Smoothness properties of the bilevel problem**). *Under Assumptions 5.4.1 and 5.4.2(i)(ii)(iv) the following statements hold.*

(i) $\|w'(\lambda)\| \leq L_w := \frac{L_\Phi}{1-q}$ *for every* $\lambda \in \Lambda$.

(ii) $w'(\cdot)$ *is Lipschitz continuous with constant*

$$L_{w'} = \frac{\bar{\nu}_2}{1-q} + \frac{L_\Phi}{(1-q)^2}\Big(\nu_2 + \bar{\nu}_1 + \frac{\nu_1 L_\Phi}{1-q}\Big).$$

(iii) $\nabla f(\cdot)$ *is Lipschitz continuous with constant*

$$L_f = \bar{\mu}_2 + L_E L_{w'} + \frac{L_\Phi}{1-q}\Big(\mu_2 + \bar{\mu}_1 + \frac{\mu_1 L_\Phi}{1-q}\Big).$$

The proof is in Appendix C.1. See Lemma 2.2 in Ghadimi and Wang (2018) for the special case of Problem (5.1) with LL of type (5.2).

5.5 Convergence of BSGM

In this section, we first derive convergence rates of the projected inexact gradient method for L-smooth possibly non-convex objectives (Section 5.5.1). Then, we

combine this result with the mean square error upper bounds in Chapter 4 (Sections 4.4 and 4.6) to obtain in Section 5.5.2, the desired convergence rate and sample complexity for BSGM (Algorithm 5.2.2).

5.5.1 Projected Inexact Gradient Method

Let $f : \Lambda \mapsto \mathbb{R}$, be an L-smooth function on the convex set $\Lambda \subseteq \mathbb{R}^m$. We consider the following *projected inexact gradient descent* algorithm

$$\begin{aligned} &\lambda_0 \in \Lambda \\ &\text{for } s = 0, 1, \dots \\ &\lfloor \ \lambda_{s+1} = P_\Lambda \left(\lambda_s - \alpha \hat{\nabla} f(\lambda_s) \right), \end{aligned} \tag{5.8}$$

where P_Λ is the projection onto Λ, $\alpha > 0$ is the step-size and $\hat{\nabla} f(\lambda_s)$ is s stochastic estimator of the gradient. We stress that we do not assume that $\hat{\nabla} f(\lambda_s)$ is unbiased.

Definition 5.5.1 (**Proximal Gradient Mapping**). *The proximal gradient mapping of f is*

$$G_\alpha(\lambda) := \alpha^{-1} \left(\lambda - P_\Lambda(\lambda - \alpha \nabla f(\lambda)) \right).$$

The above gradient mapping is commonly used in constrained non-convex optimization as a replacement of the gradient for the characterization of stationary points (see e.g. (Drusvyatskiy and Lewis, 2018)). Indeed, λ^* is a stationary point if and only if $G_\alpha(\lambda^*) = 0$ and in the unconstrained case (i.e. $\Lambda = \mathbb{R}^m$) we have $G_\alpha(\lambda) = \nabla f(\lambda)$. Since the algorithm is stochastic we provide guarantees in expectation. In particular, we bound $\frac{1}{S} \sum_{s=0}^{S-1} \mathbb{E}[\|G_\alpha(\lambda_s)\|^2]$. Note that this quantity is always greater than or equal to $\min_{s<S} \mathbb{E}[\|G_\alpha(\lambda_s)\|^2]$ (at least one of the iterates satisfies the bound).

The following theorem and corollary provide such upper bounds which have a linear dependence on the average MSE of $\hat{\nabla} f(\lambda_s)$. A similar setting is studied also by Dvurechensky (2017) where they consider inexact gradients but with a different error model. Schmidt et al. (2011) provide a similar result in the convex case.

Theorem 5.5.2. *Let $\Lambda \subseteq \mathbb{R}^m$ be convex and closed, $f : \Lambda \mapsto \mathbb{R}$ be L-smooth and $\{\lambda_s\}_s$ be a sequence generated by Algorithm* (5.8)*. Furthermore, let $\Delta_f := f(\lambda_0) -$*

$\min_\lambda f(\lambda)$, $c>0$, $\delta_s := \|\nabla f(\lambda_s) - \hat{\nabla} f(\lambda_s)\|$ *and* $0<\alpha<2/[L(1+c)]$. *Then* $\forall S \in \mathbb{N}$

$$\frac{1}{S}\sum_{s=0}^{S-1}\|G_\alpha(\lambda_s)\|^2 \leq \frac{1}{S}\left[\frac{4\Delta_f}{c_\alpha L(1+c))} + 2\left(1+\frac{1}{c_\alpha Lc}\right)\sum_{s=0}^{S-1}\delta_s^2\right],$$

where $c_\alpha = \alpha(2-\alpha L(1+c))$.

Proof. Since the projection is a *firmly non-expansive* operator, i.e. $\forall \gamma, \beta \in \mathbb{R}^n$,

$$\|P_\Lambda(\gamma) - P_\Lambda(\beta)\|^2 + \|\gamma - P_\Lambda(\gamma) - \beta + P_\Lambda(\beta)\|^2 \leq \|\gamma-\beta\|^2,$$

which yields, by expanding the second term in the Left hand side

$$2\|P_\Lambda(\gamma) - P_\Lambda(\beta)\|^2 + \|\gamma-\beta\|^2 - 2(\gamma-\beta)^\top(P_\Lambda(\gamma) - P_\Lambda(\beta)) \leq \|\gamma-\beta\|^2,$$

and, after simplifying

$$\|P_\Lambda(\gamma) - P_\Lambda(\beta)\|^2 \leq (\gamma-\beta)^\top(P_\Lambda(\gamma) - P_\Lambda(\beta)).$$

In particular, substituting $\gamma = \lambda_s$ and $\beta = \lambda_s - \alpha\hat{\nabla} f(\lambda_s)$ we get

$$\|\lambda_s - \lambda_{s+1}\|^2 \leq \alpha\hat{\nabla} f(\lambda_s)^\top(\lambda_s - \lambda_{s+1}). \tag{5.9}$$

Now, it follows from the Lipschitz smoothness of f that for every $\gamma, \beta \in \Lambda$

$$f(\beta) \leq f(\gamma) + \nabla f(\gamma)^\top(\beta-\gamma) + \frac{L}{2}\|\beta-\gamma\|.$$

Then substituting $\gamma = \lambda_s$ and $\beta = \lambda_{s+1}$, and letting $c' = Lc$ with $c>0$, we obtain

$$\begin{aligned}
f(\lambda_{s+1}) &\leq f(\lambda_s) - (\nabla f(\lambda_s) \mp \hat{\nabla} f(\lambda_s))^\top(\lambda_s - \lambda_{s+1}) + \frac{L}{2}\|\lambda_s - \lambda_{s+1}\|^2 \\
&\leq f(\lambda_s) - (\nabla f(\lambda_s) - \hat{\nabla} f(\lambda_s))^\top(\lambda_s - \lambda_{s+1}) + \left(\frac{L}{2} - \frac{1}{\alpha}\right)\|\lambda_s - \lambda_{s+1}\|^2 \\
&\leq f(\lambda_s) + \frac{1}{2c'}\|\nabla f(\lambda_s) - \hat{\nabla} f(\lambda_s))\|^2 + \left(\frac{L+c'}{2} - \frac{1}{\alpha}\right)\|\lambda_s - \lambda_{s+1}\|^2 \\
&\leq f(\lambda_s) + \frac{1}{2c'}\|\nabla f(\lambda_s) - \hat{\nabla} f(\lambda_s)\|^2 - \eta\|\lambda_s - \lambda_{s+1}\|^2,
\end{aligned}$$

where we used eq. (5.9) for the second line, the Young inequality $a^\top b \leq (1/2c')\|a\|^2 + (c'/2)\|b\|^2$ in the third line, and the definition $\eta := 1/\alpha - (L+c')/2$, which is positive due to the assumption on α, in the last line. Rearranging the terms we get

$$\|\lambda_s - \lambda_{s+1}\|^2 \leq \frac{1}{\eta}\left(f(\lambda_s) - f(\lambda_{s+1}) + \frac{1}{2c'}\|\nabla f(\lambda_s) - \hat{\nabla} f(\lambda_s)\|^2\right). \tag{5.10}$$

Furthermore, let $\bar{\lambda}_s := P_\Lambda(\lambda_s - \alpha\nabla f(\lambda_s))$. Then, we have that

$$\begin{aligned}\|\lambda_{s+1} - \bar{\lambda}_s\|^2 &= \|P_\Lambda(\lambda_s - \alpha\hat{\nabla} f(\lambda_s)) - P_\Lambda(\lambda_s - \alpha\nabla f(\lambda_s))\|^2 \\ &\leq \alpha^2\|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|^2,\end{aligned} \tag{5.11}$$

where we used the fact that the projection is 1-Lipschitz.

Now, recalling the definition of $G_\alpha(\lambda)$ we have that $G_\alpha(\lambda_s) = \alpha^{-1}(\lambda_s - \bar{\lambda}_s)$ and hence, using the inequalities (5.10) and (5.11), we have

$$\begin{aligned}\|G_\alpha(\lambda_s)\|^2 &= \alpha^{-2}\|\lambda_s \mp \lambda_{s+1} - \bar{\lambda}_s\|^2 \\ &\leq 2\alpha^{-2}\left(\|\lambda_s - \lambda_{s+1}\|^2 + \|\lambda_{s+1} - \bar{\lambda}_s\|^2\right) \\ &\leq \frac{2}{\eta\alpha^2}\left(f(\lambda_s) - f(\lambda_{s+1}) + \frac{1}{2c'}\|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|^2\right) \\ &\quad + 2\|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|^2 \\ &= \frac{2}{\eta\alpha^2}\left(f(\lambda_s) - f(\lambda_{s+1})\right) + \left(2 + (\eta c')^{-1}\alpha^{-2}\right)\|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|^2.\end{aligned}$$

Summing the inequalities over s and noting that $-f(\lambda_s) \leq -\min_\lambda f(\lambda)$ we get

$$\sum_{s=0}^{S-1}\|G_\alpha(\lambda_s)\|^2 \leq \frac{2\Delta_f}{\eta\alpha^2} + \left(2 + (\eta c')^{-1}\alpha^{-2}\right)\sum_{s=0}^{S-1}\|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|^2.$$

Finally, dividing both sides of the above inequality by S, recalling the definition of η, δ_s and c', (5.5.3) follows. □

Corollary 5.5.3. *Under the same assumptions of Theorem 5.5.2 we have*

$$\frac{1}{S}\sum_{s=0}^{S-1}\mathbb{E}[\|G_\alpha(\lambda_s)\|^2] \leq \frac{1}{S}\left[\frac{4\Delta_f}{c_\alpha L(1+c))} + 2\left(1+\frac{1}{c_\alpha L c}\right)\sum_{s=0}^{S-1} MSE_{\hat{\nabla} f(\lambda_s)}\right],$$

where $c_\alpha = \alpha(2-\alpha L(1+c))$. *Consequently, setting* $c = 1/2$, *for any* $\alpha \leq 1/L$ *we have*

$$\frac{1}{S}\sum_{s=0}^{S-1}\mathbb{E}[\|G_\alpha(\lambda_s)\|^2] \leq \frac{1}{S\alpha}\left[8\Delta_f + \frac{10}{L}\sum_{s=0}^{S-1} MSE_{\hat{\nabla} f(\lambda_s)}\right].$$

We recall that $MSE_{\hat{\nabla} f} := \mathbb{E}[\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|^2]$.

Proof. Follows by taking expectation of the inequality in the statement of Theorem 5.5.2 □

Remark 5.5.4. *Note that if the error term* $\sum_{s=0}^{S-1} MSE_{\hat{\nabla} f(\lambda_s)}$ *grows sub-linearly with S, Corollary 5.5.3 provides useful convergence rates. In particular, when* $\sum_{s=0}^{\infty} MSE_{\hat{\nabla} f(\lambda_s)} < \infty$, *we have a convergence rate of* $O(1/S)$, *which matches the optimal rate of (exact) gradient descent on smooth and possibly non-convex objectives.*

5.5.2 Bilevel Convergence Rates and Sample Complexity

Here, we finally prove the convergence and sample complexity of Algorithm 5.2.2 by combining the results of the previous section with the bounds on the MSE of the hypergradient estimator obtained in Section 4.4.

Definition 5.5.5 (**Sample Complexity**). *An algorithm which solves the stochastic bilevel problem in* (5.1) *has sample complexity N if the total number of samples of* ζ *and* ξ *is equal to N. For Algorithm 5.2.2, this corresponds to the total number of evaluations of* $\nabla\hat{E}, \hat{\Phi}, \partial\hat{\Phi}^\top v$.

In the following theorem we establish the sample complexity of Algorithm 5.2.2 for $t_s = \lceil c_3(s+1)\rceil$ and $t_s = \lceil c_3 S\rceil$ (finite horizon), where $c_3 > 0$ is an additional hyperparameter that can be tuned empirically.

Theorem 5.5.6 (**Stochastic BSGM**). *Suppose that* $\Lambda \subseteq \mathbb{R}^m$ *and Assumptions 5.4.1, 5.4.2, 5.4.3 are satisfied. Assume that the bilevel Problem* (5.1) *is solved by Algorithm 5.2.2 with* $\alpha \leq 1/L_f$ *and* $(\eta_j)_{j\in\mathbb{N}}$ *are decreasing and chosen according*

to Theorem 4.6.1, where L_f is defined in Lemma 5.4.6. Let $\lambda_0 \in \Lambda$, $G_\alpha(\lambda) := \alpha^{-1}(\lambda - P_\Lambda(\lambda - \alpha\nabla f(\lambda)))$ be the proximal gradient mapping, $c_3 > 0$, and c_b and c_v be the defined in Corollary 4.6.2. Then the following hold.

(i) Suppose that for every $s \in \mathbb{N}$ $t_s = k_s = J_s = \lceil c_3(s+1)\rceil$. Then for every $S \in \mathbb{N}$ we have

$$\frac{1}{S}\sum_{s=0}^{S-1}\mathbb{E}[\|G_\alpha(\lambda_s)\|^2] \le \frac{1}{S\alpha}\left[8\Delta_f + \frac{10}{L_f}\frac{c_b + c_v}{c_3}(\log(S)+1)\right].$$

Moreover, after $\tilde{O}(\varepsilon^{-2})$ samples there exists $s^* \le S-1$ such that $\mathbb{E}[\|G_\alpha(\lambda_{s^*})\|^2] \le \varepsilon$.

(ii) Finite horizon. Let $S \in \mathbb{N}$, and suppose that for $s = 0, \dots, S-1$, $t_s = k_s = J_s = \lceil c_3 S\rceil$. Then we have

$$\frac{1}{S}\sum_{s=0}^{S-1}\mathbb{E}[\|G_\alpha(\lambda_s)\|^2] \le \frac{1}{S\alpha}\left[8\Delta_f + \frac{10}{L_f}\frac{c_b + c_v}{c_3}\right].$$

Moreover, after $O(\varepsilon^{-2})$ samples there exists $s^* \le S-1$ such that $\mathbb{E}[\|G_\alpha(\lambda_{s^*})\|^2] \le \varepsilon$.

Proof. We first compute N, i.e. the total number of samples used in S iterations. At the s-th iteration, Algorithm 5.2.2 requires executing Algorithm 5.2.1 which uses $t_s + k_s + J_s$ copies of ζ, for evaluating $\hat{\Phi}$, $\partial_1\hat{\Phi}^\top v$, and $\partial_2\hat{\Phi}^\top v$, and additional J_s copies of ξ for evaluating $\nabla\hat{E}$. Thus, the s-th UL iteration requires $4\lceil c_3(s+1)\rceil$ and $4\lceil c_3 S\rceil$ samples for case (i) and (ii) respectively. Hence, we have

$$(i): \quad 2c_3S^2 \le N = 4\sum_{s=0}^{S-1}\lceil c_3(s+1)\rceil \le 4(c_3+1)S^2.$$
$$(ii): \quad 4c_3S^2 \le N = 4\lceil c_3 S\rceil\sum_{s=0}^{S-1} 1 \le 4(c_3+1)S^2.$$

This implies that in both cases $N = \Theta(S^2)$ or equivalently $S = \Theta(\sqrt{N})$.

(i): Corollary 4.6.2, with $t_s = \lceil c_3(s+1) \rceil$, yields

$$\sum_{s=0}^{S-1} \mathrm{MSE}_{\hat{\nabla} f(\lambda_s)} \leq (c_b + c_v) \sum_{s=0}^{S-1} \frac{1}{c_3(s+1)} \leq \frac{c_b + c_v}{c_3} (\log(S) + 1).$$

Since ∇f is L_f-Lipschitz continuous, thanks to Lemma 5.4.6 we can apply Corollary 5.5.3 and obtain (i). Therefore, we have $\frac{1}{S}\sum_{s=0}^{S-1} \mathbb{E}[\|G_\alpha(\lambda_s)\|^2] \leq \varepsilon$ in a number of UL iterations $S = \tilde{O}(\varepsilon^{-1})$. Since we proved $N = \Theta(S^2)$, the sample complexity result for case (i) follows.

(ii): Similarly to the case (i), we apply Corollary 4.6.2 with $t_s = \lceil c_3 S \rceil$ obtaining

$$\sum_{s=0}^{S-1} \mathrm{MSE}_{\hat{\nabla} f(\lambda_s)} \leq (c_b + c_v) \sum_{s=0}^{S-1} \frac{1}{c_3 S} = \frac{c_b + c_v}{c_3}.$$

Since ∇f is L_f-Lipschitz, thanks to Lemma 5.4.6, we derive (ii) from Corollary 5.5.3.

Therefore, in this case we have $\frac{1}{S}\sum_{s=0}^{S-1} \mathbb{E}[\|G_\alpha(\lambda_s)\|^2] \leq \varepsilon$ in a number of UL iterations $S = O(\varepsilon^{-1})$. Since $N = \Theta(S^2)$, the sample complexity result for case (ii) follows. □

In the following theorem we derive rates for Algorithm 5.2.2 in the deterministic case, i.e. when the variance of $\hat{\Phi}$ $\partial\hat{\Phi}$ and $\nabla\hat{E}$ is zero. In this case we will show that the LL and LS solvers in Algorithm 5.2.1 can be implemented with constant step size and with $J_s = 1$, to obtain the near-optimal sample complexity of $\tilde{O}(\varepsilon^{-1})$.

Theorem 5.5.7 (Deterministic BSGM). *Suppose that $\Lambda \subseteq \mathbb{R}^m$ and Assumptions 5.4.1, 5.4.2, 5.4.3 are satisfied with $\sigma_1 = \sigma_2 = \sigma_1' = \sigma_2' = \hat{\sigma}_1 = \hat{\sigma}_{2,E} = 0$, hence $\hat{\Phi} = \Phi$ and $\hat{E} = E$. Assume that the bilevel Problem* (5.1) *is solved by Algorithm 5.2.2 with $\alpha \leq 1/L_f$ with L_f defined in Lemma 5.4.6, $\eta_j = 1$, $t_s = k_s = \lceil c_3 \log(s+1) \rceil$ and $J_s = 1$, and $c_3 \geq 1/\log(1/q) > 0$. Let $\lambda_0 \in \Lambda$ and $G_\alpha(\lambda) := \alpha^{-1}(\lambda - P_\Lambda(\lambda - \alpha\nabla f(\lambda)))$ be the proximal gradient mapping. Then*

$$\frac{1}{S}\sum_{s=0}^{S-1} \|G_\alpha(\lambda_s)\|^2 \leq \frac{1}{S\alpha}\left[8\Delta_f + \frac{5C\pi^2}{3L_f}\right],$$

where

$$C := 3\left(\mu_2 + \frac{\mu_1 L_\Phi + \nu_2 L_E}{1-q} + \frac{\nu_1 L_E L_\Phi}{(1-q)^2}\right)^2 B^2 + 3L_\Phi^2 \frac{L_E^2}{(1-q)^2} + 3\nu_2^2 \frac{B^2 L_E^2}{(1-q)^2}.$$

Also, after $O(\varepsilon^{-1}\log(\varepsilon^{-1}))$ *samples there exists* $s^* \leq S-1$ *such that* $\|G(\lambda_{s^*})\|^2 \leq \varepsilon$.

Proof. Similarly to the proof of Theorem 5.5.6, but with $J_s = 1$, we obtain a number of samples in S iterations which is $N = \sum_{s=0}^{S-1} 2(t_s+1) = 2\sum_{s=1}^{S} \lceil c_3 \log(s) \rceil + 1$. Hence, if $S > 1$

$$N \geq 2c_3 \sum_{s=\lceil S/2 \rceil}^{S} \log(s) \geq c_3 (S/2-1)\log(S/2),$$
$$N \leq 2c_3 S \log\left(\frac{1}{S}\sum_{s=1}^{S} s\right) + 4S \leq 4S\left[c_3 \log\left(\frac{S+1}{2}\right) + 1\right].$$

Therefore, $N = \Theta(S\log(S)) = \tilde{\Theta}(S)$.

Since in the deterministic case $\mathbb{V}[\hat{\nabla} f(\lambda)] = 0$ and $\mathbb{E}[\hat{\nabla} f(\lambda)] = \hat{\nabla} f(\lambda)$, Theorem 4.4.2(ii) and setting $J = 1$ yields

$$\begin{aligned} &\|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|^2 \\ &\leq 3\left(\mu_2 + \frac{\mu_1 L_\Phi + \nu_2 L_E}{1-q} + \frac{\nu_1 L_E L_\Phi}{(1-q)^2}\right)^2 \rho_\lambda(t_s) + 3L_\Phi^2 \sigma_\lambda(k_s) + 3\nu_2^2 \rho_\lambda(t_s)\sigma_\lambda(k_s). \end{aligned} \tag{5.12}$$

Now we note that, in view of last result of Theorem 4.6.1, we have

$$\rho_\lambda(t_s) = q^{2t_s} B^2, \qquad \sigma_\lambda(k_s) = q^{2k_s} \frac{L_E^2}{(1-q)^2}.$$

Consequently, since $t_s = k_s$ and $q^{2x} \leq q^x$ with $x \geq 1$, we get

$$\|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|^2 \leq C q^{2t_s},$$

where C incorporates all the constants occurring in (5.12).

Recall that $t_s = \lceil c_3 \log(s+1) \rceil$ and $c_3 \geq 1/\log(1/q) > 0$. From the change of

base formula we have

$$t_s \geq c_3 \log(1/q) \log_q(1/(s+1)) \geq \log_q(1/(s+1)),$$

since $\log_q(1/(s+1)) \geq 0$ due to $q < 1$, $s \geq 0$. Consequently,

$$q^{2t_s} \leq q^{2\log_q(1/(s+1))} = \frac{1}{(s+1)^2}.$$

Hence, we can bound the sum of squared errors as follows.

$$\sum_{s=0}^{S-1} \|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|^2 \leq \sum_{s=0}^{S-1} \frac{C}{(s+1)^2} \leq \sum_{s=1}^{S} \frac{C}{s^2} \leq \frac{C\pi^2}{6}.$$

Using this result in combination with Corollary 5.5.3 we obtain (5.5.7). Therefore, we have $\frac{1}{S}\sum_{s=0}^{S-1} \mathbb{E}[\|G_\alpha(\lambda_s)\|^2] \leq \varepsilon$ in a number of UL iterations $S = O(\varepsilon^{-1})$. Since we proved that $N = \Theta(S\log(S)) = \tilde{\Theta}(S)$ we obtain the final sample complexity. □

Remark 5.5.8 (Dependency on the contraction constant)**.** *By setting for the stochastic case $\eta_t = \beta/(\gamma+t)$ with $\beta = 2/(1-q^2)$ and $\gamma = \beta(1+\tilde{\sigma}_2)$ in Algorithm 5.2.1 and $\alpha = 1/L_f$, $c_3 = \Theta(1)$ in Algorithm 5.2.2, and for the deterministic case $\alpha = 1/L_f$, $c_3 = \Theta(\kappa)$ in Algorithm 5.2.2, we obtain a sample complexity of $O(\varepsilon^{-2}\kappa^{16})$ and $O(\varepsilon^{-1}\log(\varepsilon^{-1})\kappa^5)$ respectively for the stochastic case of Theorem 5.5.6(i) and the deterministic case of Theorem 5.5.7 where $\kappa = (1-q)^{-1}$. For LL problems of type* (5.2) *with Lipschitz smooth and strongly convex loss, by appropriately setting η in* (5.3)*, κ is proportional to the condition number of the LL problem. In comparison, Amigo (Arbel and Mairal, 2021) reaches a sample complexity of $O(\varepsilon^{-2}\kappa^9)$ (stochastic) and $O(\varepsilon^{-1}\kappa^4)$ (deterministic). However, we note that for the deterministic case by setting $t_s = k_s = \Theta(\kappa\log(\kappa s))$ we obtain a sample complexity of $O(\varepsilon^{-1}\kappa^4\log(\kappa\varepsilon^{-1}))$, which is worse than warm-start only for the log factor. Finally, we note that (Arbel and Mairal, 2021) have a stronger assumption, which in our setting can be formulated as $\|\partial_2\Phi(w,\lambda)\| \leq L_\Phi \quad \forall w \in \mathbb{R}^d, \lambda \in \Lambda$. If we make such assumption (which implies Assumption 5.4.2(iv)), use different stepsizes in the LL and LS, i.e. $\eta_t = \beta/(\gamma+t)$ with $\beta = 2/(1-q^2)$, $\gamma = \beta(1+\sigma_2')$ (LL) or $\gamma = \beta(1+\tilde{\sigma}_2)$*

(LS) in Algorithm 5.2.1 and set $t_s = k_s = J_s = \Theta(\kappa^3 S)$ (i.e. $c_3 = \Theta(\kappa^3)$) we also obtain a stochastic complexity of $O(\varepsilon^{-2}\kappa^9)$.

Remark 5.5.9 (An advantage of warm-start)**.** *Our sample complexity results as well as those in Ghadimi and Wang (2018) depend on the constant B, defined in Assumption 5.4.2(iii) such that $\|w_0(\lambda) - w(\lambda)\| \leq B \; \forall \lambda \in \Lambda$. Instead, warm-start complexity bounds do not require such assumption and instead depend only on the quantity $\|w_0(\lambda_0) - w(\lambda_0)\|$, which can be much smaller than B; see e.g. (Arbel and Mairal, 2021). Although our method matches the sample complexity of warm-start approaches in the parameter ε, this aspect may lead to better bounds for warm-start, thus explaining why it is generally advantageous in practice.*

5.6 Experiments

We design the experiments with the following goals. Firstly, we assess the difficulties of applying warm-start and the effect of different upper-level batch sizes in a classification problem involving equilibrium models and in a meta-learning problem. In both settings the lower-level problem can be divided into several smaller sub-problems. Secondly, we compare our method with others achieving near-optimal sample complexity in a data poisoning experiment. All methods have been implemented in PyTorch (Paszke et al., 2019b) and the experiments have been executed on a GTX 1080 Ti GPU with 11GB of dedicated memory.

5.6.1 Equilibrium models

We consider a variation of the equilibrium models experiment presented in Section 3.4.2. In particular, we consider a multi-class classification problem with the following bilevel formulation:

$$\begin{aligned} &\min_{\lambda \in \Lambda} \sum_{i=1}^{n} \mathrm{CE}(\theta w(\lambda)^i + b, y_i) \\ &\text{with} \;\; w(\lambda)^i = \tanh(A w(\lambda)^i + B X_i + c) \quad \forall i \in \{1, \dots, n\} \end{aligned} \tag{5.13}$$

where CE is the cross-entropy loss, $(X, y) \in \mathbb{R}^{n \times p} \times \{1, \dots, c\}^n$ is the training set, $\lambda = (\theta, b, A, B, c)$, $\Lambda = \{\theta \in \mathbb{R}^{c \times d} : \|\theta\|_\infty \leq 1\} \times \mathbb{R}^c \times \{A \in \mathbb{R}^{d \times d} : \|A\| \leq 0.5\} \times$

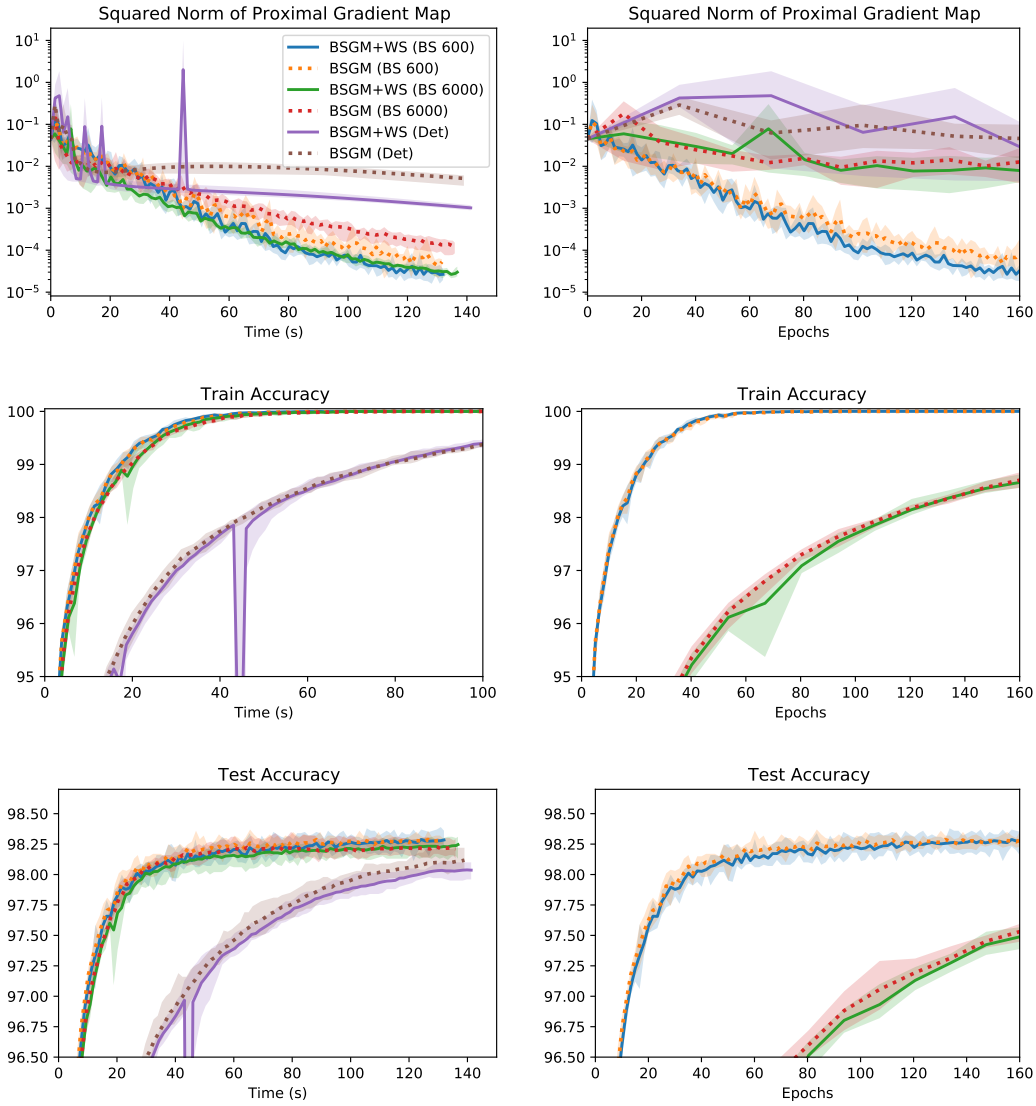

Figure 5.1: Equilibrium Models on MNIST. Results show mean (solid, dashed and dotted lines) and max-min (shaded region) over 5 seeds varying the randomness in the mini-batches and the initialization. BSGM is the method in Algorithm 5.2.2 while BSGM+WS is the variant with warm-start on the LL. BS indicates the mini-batch size used while methods with Det in the name use the whole training set of 60K examples.

$\mathbb{R}^{d\times p}\times\mathbb{R}^d$ and $w(\lambda)^i\in\mathbb{R}^d$ is the fixed point representation for i-th training example. The constraint on A, guarantees that for all i, the map $w\mapsto\tanh(Aw+Bx_i+c)$ is a contraction with Lipschitz constant not greater than 0.5. We perform this experiments using the whole MNIST training set, hence $n=6\times10^4, p=784, c=10$, and set $d=200$.

We compare variants of BSGM (Algorithm 5.2.2) with different batch sizes (J_s in Algorithm 5.2.2), which in this case indicates the number of training examples used to estimate the gradients of the UL objective. We also evaluate an extension of BSGM which uses warm-start only on the LL problem (similar to StochBIO (Ji et al., 2021)). Note that when using warm-start, all the fixed point representations computed by the algorithm are stored in memory to be used in the future. When the ratio between the number of examples n and the batch size is large, this can greatly increase the memory cost of the algorithm compared to the procedure without warm-start. For this particular problem, this cost is manageable since it amounts to storing a total of $nd = 12 \times 10^6$ floats, which correspond to 48 MB of memory, but for higher values of d and n it quickly becomes prohibitive, as we show in the meta-learning experiment.

Let $\lambda_0 = (\theta_0, b_0, A_0, B_0, c_0)$ be the hyperparameters at initialization, we set $b_0 = 0$, and we sample each coordinate of θ_0, A_0, B_0, and c_0 from a Gaussian distribution with zero mean and standard deviation 0.01. In Algorithm 5.2.2 we also set $w_0(\lambda) = 0$, $t_s = k_s = 2$, and $\alpha = 0.5$. Since computing the map $w \mapsto \tanh(Aw + Bx_i + c)$ is relatively cheap, we use deterministic solvers with step-size 1 for the LL and LS of each training example. To evaluate the UL parameters found by the algorithms, we compute an accurate approximation of the LL solution and the hypergradient on all training examples by running the LL and LS solver for 20 steps. The proximal gradient map is computed according to (5.5.1) with $\alpha = 1$.

Results are shown in Figure 5.1, where we compare three key performance measures of the different methods versus time and number of epochs. When comparing methods using the same batch size we can see that using warm-start improves the performance in terms of the norm of the proximal gradient map, i.e. the quantity that we can control theoretically. However, this effect decreases with smaller batch sizes since more UL iterations can pass until the same example is sampled twice. Furthermore, train and test accuracy are similar for methods with the same batch size, regardless of the use of warm-start. Finally, we note that decreasing the mini-batch consistently improves the performance in terms of number of epochs while, thanks

to the parallelism of the GPU, the performance with batch size equal to 600 and 6000 are similar.

5.6.2 Meta-learning

We perform a meta-learning experiment on Mini-Imagenet (Vinyals et al., 2016), a popular few-shot classification benchmark. Mini-Imagenet contains 100 classes from Imagenet which are split into 64, 16, 20 for the meta-train, meta-validation and meta-test sets respectively. A task is constructed by selecting some images from c randomly selected classes. Each image is downsampled to 84×84 pixels. Similarly to Franceschi et al. (2018), we evaluate an hyper-representation model where the UL parameters are the parameters of the representation layers of a convolutional neural network (CNN), shared across tasks, while the task-specific LL parameters are the parameters of the last linear layer. The CNN is composed by stacking 4 blocks, each made by a 3×3 convolutions with 32 output channels followed by a batch normalization layer.

We evaluate the performance of Algorithm 5.2.2 where the network parameters λ_0 are initialized using the default random initialization in PyTorch, $w_0(\lambda) = 0$, $\alpha = 0.2$, $\eta_j = 0.05$, $t_s = 10$, and different batch sizes $J_s = \{8, 16, 32\}$. The batch size in this case corresponds to the number of tasks at each UL iteration. Using warm start in this setting may require to save the last linear layer for all tasks, hence $n \times d \times c$ floats, where n is the number of tasks and $d \times c$ are the number of weights in the last linear layer. A meta-training task is constructed by selecting $c = 5$ classes out of 64, hence the number of tasks is $n = 7{,}624{,}512$. Moreover, we set $d = 800$. Thus, storing the last layer for all tasks would require 122GB of storage, which largely exceeds our GPU memory. Furthermore, the ratio between n and batch size is very high and this is likely to make make the effect of using warm-start negligible.

Results are shown in Figure 5.2, where we see that methods with smaller batch-sizes converge faster despite requiring a higher number of UL iterations. Furthermore, since during meta-training we see only $50,000$ tasks, we also implemented the method using warm-start by storing the approximate solutions to all previously sampled tasks to be used as initialization when they are sampled again. We run

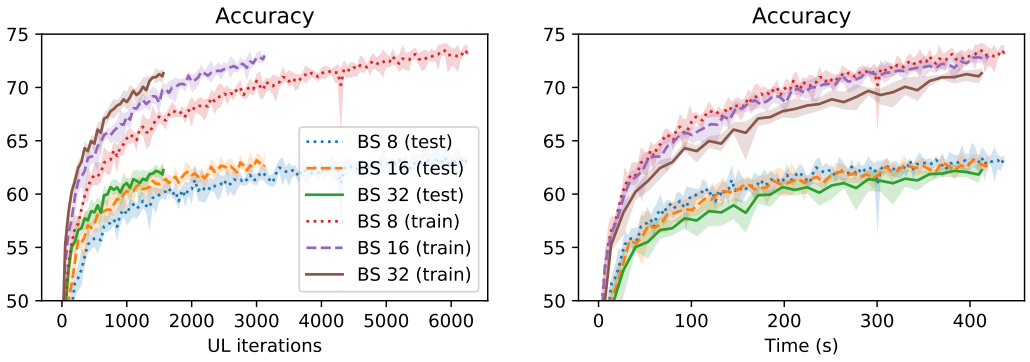

Figure 5.2: 5-way 5-shot classification on Mini-Imagenet. The plot show mean (solid lines) and max − min (shaded region) over 5 runs. Values are the average accuracy over 1000 meta-train/meta-test tasks computed after 10 steps of the LL solver. At the end of training all methods have seen a total of 50K tasks.

the method with mini-batch size equal to 8 and for 5 seeds and observed that all metrics essentially overlap the ones without warm-start, while the memory cost increases by 0.8 GB. These experiments suggest that warm-start may be ineffective in meta-learning problems, as mentioned in the introduction. Indeed, in this setting we observed that each task is sampled at most 3 times in a total of $6,250$ iterations.

5.6.3 Data poisoning

We consider the *data poisoning* scenario where a malicious agent or *attacker* aims at decreasing the performance of a machine learning model by corrupting its training dataset. In particular, the attacker adds noise to some of the training examples. However, this noise must be small in magnitude to avoid for the attack to be uncovered.

Specifically, we consider an image classification problem on the MNIST dataset where $(X, y) \in \mathbb{R}^{n \times p} \times \{1, \dots, c\}^n$, and $(X', y') \in \mathbb{R}^{n' \times p} \times \{1, \dots, c\}^n$ are the training and validation sets, and $p = 784$, $c = 10$, $n = 45{,}000$ and $n' = 15{,}000$ are the number of features, classes, training examples and validation examples respectively. Furthermore, we randomly select $\mathcal{I} \subseteq \{1, \dots, n\}$ to be the indices of the corrupted training examples such that $|\mathcal{I}| = 9{,}000$. The attacker finds the noise λ by solving

the following bilevel optimization problem.

$$\begin{aligned} &\max_{\lambda \in \Lambda} \frac{1}{n'} \sum_{i=i}^{n'} \mathrm{CE}(w(\lambda)^\top X'_i, y'_i) \\ \text{with } \; w(\lambda) = &\operatorname*{argmin}_{w \in \mathbb{R}^{p \times c}} \frac{1}{n} \sum_{i=1}^{n} \mathrm{CE}(w^\top (X_i + \lambda_i), y_i) + \frac{0.1}{p} \|w\|^2, \end{aligned} \tag{5.14}$$

where CE is the cross-entropy loss, $\Lambda = \{\lambda \in \mathbb{R}^{n \times p} \mid \lambda_i \in \mathcal{B}_2(0,5)\ \forall i \in \mathcal{I}, \lambda_i = 0\ \forall i \in \{1,\dots,n\}/\mathcal{I}\}$ and $\mathcal{B}_2(0,5)$ is the p-dimensional L2-ball centered in 0 with radius 5. Note that the LL problem is both strongly convex and Lipschitz smooth.

Baselines. We compare our method with StochBIO (Ji et al., 2021), Amigo (Arbel and Mairal, 2021), ALSET (Chen et al., 2022), which achieve (near) optimal sample complexity. We also consider ALSET†, i.e. a variant of ALSET where the LS problem is solved using warm-start and only one iteration. All baselines have been implemented as extensions to Algorithm 5.2.2 specialized to LL problems of type (5.2), which differ only in the use of warm-start and in the number of iterations and batch-sizes used. Except for ALSET†-DET, which is the deterministic version of ALSET† and computes the LL objective exactly, all other methods use mini-batches of size 90 to estimate the LL objective and its derivatives. We found this value to be sufficiently large for Amigo and StochBIO to perform well. The UL objective is instead always computed using all 15K validation examples. To fairly evaluate the different bilevel optimization methods, the linear model used for the final evaluation is trained by 1000 steps of gradient descent on the LL objective

$$\frac{1}{n} \sum_{i=1}^{n} \mathrm{CE}(w^\top (X_i + \lambda^*), y_i) + \frac{0.1}{p} \|w\|^2,$$

where λ^* is the output of the bilevel optimization method.

Random Search. Bilevel optimization methods have several configuration parameters which greatly affect the performance, e.g. the number of iterations for the LL and LS solvers, step sizes for the UL, LL and LS. Theoretical values for these parameters are often too conservative, hence they are usually set via manual search which is hard to reproduce and may be suboptimal. Thus, for a better comparison,

we set a total budget of 2M single-sample gradients and Hessian-vector products, so that each algorithm uses the same number of samples[3], and perform a random search with 200 random configuration parameters to select the configurations achieving the lowest accuracy on the validation set. Values and ranges of the random search are shown in Table 5.2. Note that to reduce the number of configuration parameters we keep them unchanged across UL and LL/LS iterations. For our method, we observed that using fixed instead of decreasing stepsizes for the LL/LS does not affect the top performances after the random search. Furthermore, we set $k = t$ and $\eta_{\text{LL}} = \eta_{\text{LS}}$ only for our method and all the others which use warm-start both for the LL and LS problems, which seems to improve performance[4].

Results. In Table 5.3 we show the results. Our method (BSGM) outperforms all the single-loop bilevel optimization methods (ALSET† and ALSET). However, methods using warm-start only in the LL (StochBIO) and both in LL and LS (Amigo) outperform BSGM, albeit not by a large margin. To aid reproducibility, we report in Table 5.4 the best configuration parameters of each method.

Method	WS	t	k	J	α	η_{LL}	η_{LS}
StochBIO	Y,N	$[10:10^4]$	$[10:10^4]$	k	$[10^3:10^9]$	$[10^{-4}:10]$	$[10^{-4}:10]$
Amigo	Y,Y	$[10:10^4]$	t	t	$[10^3:10^9]$	$[10^{-4}:10]$	η_{LL}
BSGM (ours)	N,N	$[10:10^4]$	t	t	$[10^3:10^9]$	$[10^{-4}:10]$	η_{LL}
ALSET†-DET	Y,Y	1	1	1	$[10^3:10^9]$	$[10^{-4}:10]$	η_{LL}
ALSET†	Y,Y	1	1	1	$[10^3:10^9]$	$[10^{-4}:10]$	η_{LL}
ALSET	Y,N	1	$[10:10^4]$	1	$[10^3:10^9]$	$[10^{-4}:10]$	$[10^{-4}:10]$

Table 5.2: Configurations parameters for the random search. The WS column indicates whether warm-start is used (Y) or not (N) for the LL (first entry) and LS (second entry). t, k and J are respectively the number of iteration for the LL and LS and the batch size, while α, η_{LL}, and η_{LS} are the step sizes for the UL, LL and LS respectively. Configuration parameters are sampled according to the log-uniform distribution over the specified ranges. For all methods we set $\lambda_0 = 0$.

[3]We do not account for the difference in computational cost between gradients and Hessian vector-products. The latter are usually more costly in practice.

[4]Indeed, we observed that using $k \neq t$ and $\eta_{\text{LL}} \neq \eta_{\text{LS}}$ for BSGM and Amigo does not improve and usually decreases the performance of the best methods, while setting $k = t$ and $\eta_{\text{LL}} = \eta_{\text{LS}}$ decreases the performance of StochBiO.

Method	Test (Val) Best	Test (Top 10)	Val (Top 10)
StochBIO	76.78 (73.57)	79.97 ± 1.92	77.33 ± 2.28
Amigo	78.01 (75.09)	79.29 ± 0.94	76.27 ± 0.93
BSGM (ours)	78.05 (75.05)	80.90 ± 1.33	78.16 ± 1.48
ALSET†-DET	83.03 (80.30)	86.13 ± 1.38	84.10 ± 1.73
ALSET†	90.75 (89.99)	90.66 ± 0.13	90.19 ± 0.15
ALSET	90.89 (90.49)	90.99 ± 0.11	90.65 ± 0.10

Table 5.3: Data-poisoning Accuracy (Lower is better). We report values for best and top 10 best performing parameter configurations selected via random search. For the top 10 results we report mean $\pm$ standard deviation. ALSET†-DET is the best performing deterministic method, all the others are stochastic.

Method	Test (Val) Acc	t	k	J	α	η_{LL}	η_{LS}
StochBIO	76.78 (73.57)	418	2477	k	1.0×10^{6}	5.4×10^{-3}	1.3×10^{-2}
Amigo	78.01 (75.09)	155	t	t	1.0×10^{7}	1.1×10^{-2}	LL sz
BSGM (ours)	78.05 (75.05)	287	t	t	4.0×10^{8}	9.0×10^{-2}	LL sz
ALSET†-DET	83.03 (80.30)	1	1	1	1.8×10^{5}	5.6×10^{-1}	LL sz
ALSET†	90.75 (89.99)	1	1	1	1.6×10^{6}	5.3×10^{-2}	3.9×10^{-1}
ALSET	90.89 (90.49)	1	85	1	5.5×10^{8}	2.0×10^{-2}	2.7×10^{-1}

Table 5.4: Best configuration parameters. Configuration parameters with lowest validation accuracy among 200 random configurations for each method.

5.7 Discussion

In this chapter, we studied bilevel optimization problems where the upper-level objective is smooth and the lower-level solution is the fixed point of a smooth contraction mapping. In particular, we presented BSGM (Algorithm 5.2.2), a bilevel optimization procedure based on inexact gradient descent, where the inexact gradient is computed via SID (Algorithm 5.2.1). SID uses stochastic fixed-point iterations to solve both the lower-level problem and the linear system and estimates ∇E and $\partial_2 \Phi$ using large mini-batches. We proved that, even without the use of warm-start on the lower-level problem and the linear system, BSGM achieves optimal and near-optimal sample complexity in the stochastic and deterministic bilevel setting respectively. We stress that in recent literature, warm-start was thought to be crucial to achieve the optimal sample complexity. We also showed that, when compared to methods using

warm-start, our approach yields a simplified and modular analysis which does not deal with the interactions between upper-level and lower-level iterates. Moreover, we showed empirically the inconvenience of the warm-start strategy on equilibrium models and meta-learning. Finally, we compared our method with several bilevel methods relying on warm-start on a data-poisoning experiment.

Chapter 6

Conclusions

In this thesis, we studied the theoretical properties of efficient bilevel optimization methods. In particular, we considered the bilevel framework where the aim is to find the minimum of a function which depends on the solution of a parametric fixed-point equation. Such framework provides a bird's eye view of several learning problems such as hyperparameter optimization, meta-learning, poisoning attacks and equilibrium models, allowing us to focus on general (hyper)gradient-based methods that rely only on exact gradients and Jacobian-vector products, which can be efficiently computed via automatic differentiation. The presence of an implicit function in the objective makes computing gradients and performing gradient-based optimization a rather difficult task, which requires additional approximations compared to more standard objectives for which exact derivatives can be easily computed. Our main contribution was to derive rates of convergence for deterministic and stochastic hypergradient approximation methods and finally for an entire bilevel optimization procedure, when the fixed-point equation is a contraction and all functions involved are smooth.

In particular, in Chapter 3, we proved deterministic non-asymptotic convergence rates for two hypergradient approximation techniques: iterative differentiation (ITD) and approximate implicit differentiation (AID). In particular, we showed that both converge linearly to the true hypergradient although the upper bound for the ITD error converges slower than that of the AID error and this translates into slower convergence also in practice. Furthermore, we showed that in deep equilibrium

models when the contraction assumption is not satisfied, ITD is generally more stable than AID.

In Chapter 4, we studied stochastic implicit differentiation (SID), the stochastic version of AID, which relies on unbiased estimators of gradients and Jacobian-vector products in place of the exact quantities. We proved that using large mini-batch sizes twice per upper-level iteration and decreasing step-sizes at the lower-level we achieve a rate of convergence of $O(1/t)$ where t is the size of the mini-batch and the number of iterations for the lower-level and linear system solvers.

Finally, in Chapter 5, we proved that a simple projected inexact gradient descent algorithm with the hypergradient estimated using SID achieves optimal non-convex sample complexity to solve the whole bilevel problem. A key feature of our algorithm is that it does not warm-start the lower-level problem and the linear system, which was previously thought essential to achieve optimal sample complexity. Also, we showed that for meta-learning and deep equilibrium models, applying warm-start is either not practical or not beneficial, while for data poisoning attacks it can be advantageous, although our method remains competitive.

6.1 Future Work

In the future, it would be valuable to extend the ideas presented here to other challenging machine learning scenarios not covered by our theoretical analysis. One of the most interesting directions would be to cover the important case where a neural network is trained at the lower-level, which yields a non-convex lower-level minimization problem. To tackle this scenario, the contraction assumption must be removed and the lower-level problem might have multiple solutions, which makes the hypergradient not well-defined. Some recent works design principled methods for bilevel problems having (just) a convex minimization problem (hence with possibly multiple solutions forming a convex set) at the lower-level (Liu et al., 2020, 2022; Sow et al., 2022), which is not the case for neural networks, while Vicol et al. (2022) study the effect of the implicit bias originated by the use of warm-start and

approximate hypergradients in over-parametrized[1] quadratic bilevel problems. We believe that further exploration of over-parametrized bilevel problems, taking also into account the specific structure and weights initialization of neural networks, could be an interesting research direction. A possible starting point could be the work by Allen-Zhu et al. (2019) (and references therein) on the linear convergence of SGD and GD on over-parameterized and randomly initialized deep neural networks.

As we mentioned also in the experimental section of Chapter 5, gradient-based bilevel optimization methods are applied to very different problems and usually depend heavily on at least 3 algorithmic parameters often chosen by hand, e.g. the upper-level, lower-level and linear system step sizes and optionally the number of iterations. This makes it difficult to fairly compare bilevel strategies empirically. Since the topic has gained substantial popularity, we believe it would be useful to design more standardized benchmarks for bilevel methods, covering a wide range of applications in machine learning and with a fairly simple and reproducible procedure to select the algorithmic parameters, such as random search. At the beginning, it might be advantageous to include only settings where the lower-level is strongly convex, since problems of this kind have been studied the most.

[1] Having more parameters than training/validation examples.

Appendix A

Appendix for Chapter 3

This Appendix is organized as follows.

- Appendix A.1 presents the proofs of the results stated in Section 3.3.
- In Appendix A.2 we specialize the bounds in Section 3.3 in the case where the lower-level solution can be written as the fixed point of a one step gradient descent map.
- Appendix A.3 presents the details of the experiments in Section 3.4 and additional results.

A.1 Proofs of the Results in Section 3.3

In this section we provide complete proofs of the results presented in the main body, which are restated here for the convenience of the reader. We also report few necessary additional results.

In the following technical lemma we give two results which are fundamental for the proofs of the ITD bound (Theorem 3.3.8) and the AID-FP bound (Theorem 3.3.12). The first result is standard (see Polyak (1987) Lemma 1, Section 2.2).

Lemma A.1.1. *Let $(u_k)_{k\in\mathbb{N}}$ and $(\tau_k)_{k\in\mathbb{N}}$ be two sequences of real non-negative numbers and let $q \in [0,\infty)$. Suppose that, for every $k \in \mathbb{N}$, with $k \geq 1$,*

$$u_k \leq q u_{k-1} + \tau_{k-1}. \tag{A.1}$$

Then, the following hold.

(i) If $(\tau_k)_{k\in\mathbb{N}} \equiv \tau$, *then* $u_k \leq q^k u_0 + \tau(1-q^k)/(1-q)$.

(ii) If, for every integer $k \geq 1$, $\tau_k \leq q\tau_{k-1}$, *then* $u_k \leq q^k u_0 + kq^{k-1}\tau_0$.

Proof. Let $k \in \mathbb{N}$, with $k \geq 1$. Then, we have

$$\begin{aligned} u_k &\leq qu_{k-1} + \tau_{k-1} \\ &\leq q(qu_{k-2} + \tau_{k-2}) + \tau_{k-1} \\ &= q^2 u_{k-2} + (\tau_{k-1} + q\tau_{k-2}) \\ &\vdots \\ &\leq q^k u_0 + \sum_{i=0}^{k-1} q^i \tau_{k-1-i}. \end{aligned} \tag{A.2}$$

(i): Suppose that $(\tau_k)_{k\in\mathbb{N}} \equiv \tau$. Then it follows from (A.2) that $u_k \leq q^k u_0 + \tau\sum_{i=0}^{k-1} q^i = q^k u_0 + \tau(1-q^k)/(1-q)$.

(ii): Suppose that, for every integer $k \geq 1$, $\tau_k \leq q\tau_{k-1}$. Then, for every integer k, i with $i \leq k-1$, we have $\tau_{k-1-i} \leq q^{k-1-i}\tau_0$, which substituted into (A.2) yields

$$u_k \leq q^k u_0 + \sum_{i=0}^{k-1} q^i q^{k-1-i}\tau_0$$

and (ii) follows. □

Proposition 3.3.7. *Suppose that Assumptions 3.3.1(iii) and 3.3.5 hold and let* $t \in \mathbb{N}$, *with* $t \geq 1$. *Moreover, for every* $\lambda \in \Lambda$, *let* $w_t(\lambda)$ *be computed by Algorithm 3.3.1 and let* D_λ *and* $L_{\Phi,\lambda}$ *be as in Lemma 3.3.4. Then,* $w_t(\cdot)$ *is differentiable and, for every* $\lambda \in \Lambda$,

$$\|w_t'(\lambda) - w'(\lambda)\| \leq \left(\nu_{2,\lambda} + \nu_{1,\lambda}\frac{L_{\Phi,\lambda}}{1-q_\lambda}\right) D_\lambda t q_\lambda^{t-1} + \frac{L_{\Phi,\lambda}}{1-q_\lambda} q_\lambda^t. \tag{3.7}$$

Proof. We assume that $(w_t(\lambda))_{t\in\mathbb{N}}$ is defined through the iteration

$$w_t(\lambda) = \Phi(w_{t-1}(\lambda), \lambda) \tag{A.3}$$

starting from $w_0(\lambda) = 0 \in \mathbb{R}^d$. Let $t \in \mathbb{N}$ with $t \geq 1$. Then, the mapping $\lambda \mapsto w_t(\lambda)$ is differentiable since, in view of (1), it is a composition of differentiable functions, whereas $w'(\lambda)$ exists due to Theorem 3.3.2. Differentiating the lower-level equation in (3.1) and the recursive equation in (A.3), we get

$$\begin{aligned} w_t'(\lambda) &= \partial_1\Phi(w_{t-1}(\lambda),\lambda)w_{t-1}'(\lambda) + \partial_2\Phi(w_{t-1}(\lambda),\lambda) \\ w'(\lambda) &= \partial_1\Phi(w(\lambda),\lambda)w'(\lambda) + \partial_2\Phi(w(\lambda),\lambda). \end{aligned} \tag{A.4}$$

Therefore, we get

$$\begin{aligned} \|w_t'(\lambda) - w'(\lambda)\| \leq & \|\partial_1\Phi(w_{t-1}(\lambda),\lambda) - \partial_1\Phi(w(\lambda),\lambda)\|\|w'(\lambda)\| \\ & + \|\partial_1\Phi(w_{t-1}(\lambda),\lambda)\|\|w_{t-1}'(\lambda) - w'(\lambda)\| \\ & + \|\partial_2\Phi(w_{t-1}(\lambda),\lambda) - \partial_2\Phi(w(\lambda),\lambda)\| \end{aligned}$$

and hence, we derive from Assumption 3.3.1(iii), Assumption 3.3.5, equation (3.4) and Lemmas 3.3.4 and 3.3.6, that

$$\begin{aligned} \|w_t'(\lambda) - w'(\lambda)\| \leq & (\nu_{2,\lambda} + \nu_{1,\lambda}L_{\Phi,\lambda}/(1-q_\lambda))\|w_{t-1}(\lambda) - w(\lambda)\| \\ & + q_\lambda\|w_{t-1}'(\lambda) - w'(\lambda)\|. \end{aligned}$$

Then, setting $p := \nu_{2,\lambda} + \nu_{1,\lambda}L_{\Phi,\lambda}/(1-q_\lambda)$, $\Delta_t := \|w_t(\lambda) - w(\lambda)\|$ and $\Delta_t' := \|w_t'(\lambda) - w'(\lambda)\|$, we get

$$\Delta_t \leq q_\lambda\Delta_{t-1} \quad \text{and} \quad \Delta_t' \leq q_\lambda\Delta_{t-1}' + p\Delta_{t-1}.$$

Therefore, it follows from Lemma A.1.1(ii) (with $u_t = \Delta_t'$ and $\tau_t = p\Delta_t$) that

$$\Delta_t' \leq q_\lambda^t\Delta_0' + tq_\lambda^{t-1}p\Delta_0 \leq \frac{L_{\Phi,\lambda}}{1-q_\lambda}q_\lambda^t + pD_\lambda tq_\lambda^{t-1},$$

where in the last inequality we used the bounds (see (A.4) and Lemmas 3.3.4 and

3.3.6)

$$\Delta_0 = \|w(\lambda) - w_0(\lambda)\| = \|w(\lambda)\| = D_\lambda$$
$$\Delta_0' = \|w'(\lambda) - w_0'(\lambda)\| = \|w'(\lambda)\| \leq \frac{L_{\Phi,\lambda}}{1-q_\lambda}. \tag{A.5}$$

Recalling the definitions of p and Δ_t', (3.7) follows. □

Theorem 3.3.8. *(ITD bound) Suppose that Assumptions 3.3.1(iii)-(iv) and 3.3.5 hold and let $t \in \mathbb{N}$ with $t \geq 1$. Moreover, for every $\lambda \in \Lambda$, let $w_t(\lambda)$ and f_t be defined according to Algorithm 3.3.1 and let $D_\lambda, L_{E,\lambda}$, and $L_{\Phi,\lambda}$ be as in Lemma 3.3.4. Then, f_t is differentiable and, for every $\lambda \in \Lambda$,*

$$\|\nabla f_t(\lambda) - \nabla f(\lambda)\| \leq \left(c_1(\lambda) + c_2(\lambda)\frac{t}{q_\lambda} + c_3(\lambda)\right) q_\lambda^t, \tag{3.8}$$

where

$$c_1(\lambda) = \left(\mu_{2,\lambda} + \frac{\mu_{1,\lambda} L_{\Phi,\lambda}}{1-q_\lambda}\right) D_\lambda,$$
$$c_2(\lambda) = \left(\nu_{2,\lambda} + \frac{\nu_{1,\lambda} L_{\Phi,\lambda}}{1-q_\lambda}\right) L_{E,\lambda} D_\lambda,$$
$$c_3(\lambda) = \frac{L_{E,\lambda} L_{\Phi,\lambda}}{1-q_\lambda}.$$

Proof. It follows from the definitions of f_t and f in Algorithm 3.3.1 and (3.1) respectively and the chain rule for differentiation that

$$\nabla f_t(\lambda) = \nabla_2 E(w_t(\lambda), \lambda) + w_t'(\lambda)^\top \nabla_1 E(w_t(\lambda), \lambda)$$
$$\nabla f(\lambda) = \nabla_2 E(w(\lambda), \lambda) + w'(\lambda)^\top \nabla_1 E(w(\lambda), \lambda).$$

Therefore,

$$\begin{aligned}
&\|\nabla f_t(\lambda) - \nabla f(\lambda)\| \\
&\quad \leq \|\nabla_2 E(w_t(\lambda),\lambda) - \nabla_2 E(w(\lambda),\lambda)\| \\
&\qquad + \|w'(\lambda)\|\|\nabla_1 E(w_t(\lambda),\lambda) - \nabla_1 E(w(\lambda),\lambda)\| \\
&\qquad + \|w_t'(\lambda) - w'(\lambda)\|\|\nabla_1 E(w_t(\lambda),\lambda)\|.
\end{aligned}$$

Now, we note that $\|w_t(\lambda)\| \leq \|w_t(\lambda) - w(\lambda)\| + \|w(\lambda)\| \leq (q_\lambda^t + 1)\|w(\lambda)\| \leq 2D_\lambda$. Therefore, it follows from Assumption 3.3.1(iv) and Lemmas 3.3.4 and 3.3.6 that

$$\begin{aligned}
\|\nabla f_t(\lambda) - \nabla f(\lambda)\| \leq &\left(\mu_{2,\lambda} + \mu_{1,\lambda} L_{\Phi,\lambda}/(1-q_\lambda)\right) q_\lambda^t D_\lambda \\
&+ L_{E,\lambda}\|w'(\lambda) - w_t'(\lambda)\|,
\end{aligned}$$

where we used $\|w_t(\lambda) - w(\lambda)\| \leq q_\lambda^t \|w_0(\lambda) - w(\lambda)\| = q_\lambda^t \|w(\lambda)\| = q_\lambda^t D_\lambda$. Then, (3.8) follows from Proposition 3.3.7. □

Now we address the proofs related to the AID method described in Section 3.3.2.

Theorem 3.3.10. *(AID bound) Suppose that Assumptions 3.3.1(i)(iii)(iv) and 3.3.9(i)–(iii) hold. Let $\lambda \in \Lambda$, $t \in \mathbb{N}$, $k \in \mathbb{N}$. Let $D_\lambda, L_{E,\lambda}$, and $L_{\Phi,\lambda}$ be as in Lemma 3.3.4 and let $\hat{\mu}_\lambda$ be defined according to* (3.12)*. Let $\hat{\nabla} f(\lambda)$ be defined as in Algorithm 3.3.2 and let $\hat{\Delta} = \|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|$. Then,*

$$\hat{\Delta} \leq \left(\mu_{2,\lambda} + \frac{\mu_{1,\lambda} L_{\Phi,\lambda}}{\hat{\mu}_\lambda} + \frac{\nu_{2,\lambda} L_{E,\lambda}}{\hat{\mu}_\lambda} + \frac{\nu_{1,\lambda} L_{\Phi,\lambda} L_{E,\lambda}}{\hat{\mu}_\lambda^2}\right) D_\lambda \rho_\lambda(t) + \frac{L_{\Phi,\lambda} L_{E,\lambda}}{\hat{\mu}_\lambda} \sigma_\lambda(k). \tag{3.13}$$

Furthermore, if Assumption 3.3.5 holds, then $\hat{\mu}_\lambda = 1 - q_\lambda$ and

$$\hat{\Delta} \leq \left(c_1(\lambda) + \frac{c_2(\lambda)}{1-q_\lambda}\right)\rho_\lambda(t) + c_3(\lambda)\sigma_\lambda(k), \tag{3.14}$$

where $c_1(\lambda)$, $c_2(\lambda)$ and $c_3(\lambda)$ are defined in Theorem 3.3.8.

Proof. For the sake of brevity we set

$$\begin{aligned}
\hat{A}_\lambda &= I - \partial_1\Phi(w_t(\lambda),\lambda)^\top, & A_\lambda &= I - \partial_1\Phi(w(\lambda),\lambda)^\top,\\
\hat{Z}_\lambda &= \partial_2\Phi(w_t(\lambda),\lambda), & Z_\lambda &= \partial_2\Phi(w(\lambda),\lambda),\\
\hat{b}_\lambda &= \nabla_1 E(w_t(\lambda),\lambda), & b_\lambda &= \nabla_1 E(w(\lambda),\lambda),\\
\hat{c}_\lambda &= \nabla_2 E(w_t(\lambda),\lambda), & c_\lambda &= \nabla_2 E(w(\lambda),\lambda).
\end{aligned}$$

It follows form Assumption 3.3.9(ii) that $\|w_t(\lambda) - w(\lambda)\| \leq \rho_\lambda(t)\|w(\lambda)\| = \rho_\lambda(t)D_\lambda \leq D_\lambda$ and hence $\|w_t(\lambda)\| \leq \|w_t(\lambda) - w(\lambda)\| + \|w(\lambda)\| \leq 2D_\lambda$. Then the following upper bounds related to the above quantities follow from Assumptions 3.3.1(iii)-(iv), equation (3.12) and Lemma 3.3.4.

$$\begin{aligned}
&\|\hat{A}_\lambda - A_\lambda\| \leq \nu_{1,\lambda}\rho_\lambda(t)D_\lambda,\ \|\hat{Z}_\lambda - Z_\lambda\| \leq \nu_{2,\lambda}\rho_\lambda(t)D_\lambda,\\
&\|\hat{b}_\lambda - b_\lambda\| \leq \mu_{1,\lambda}\rho_\lambda(t)D_\lambda,\ \|\hat{c}_\lambda - c_\lambda\| \leq \mu_{2,\lambda}\rho_\lambda(t)D_\lambda,\\
&\|\hat{A}_\lambda^{-1}\|, \|A_\lambda^{-1}\| \leq \frac{1}{\hat{\mu}_\lambda},\ \|\hat{Z}_\lambda\| \leq L_{\Phi,\lambda},\ \|\hat{b}_\lambda\|, \|b_\lambda\| \leq L_{E,\lambda}.
\end{aligned}$$

Now, setting $\hat{v}(\lambda) = \hat{A}_\lambda^{-1}\hat{b}_\lambda$ and $v(\lambda) = A_\lambda^{-1}b_\lambda$, $\hat{\nabla}f(\lambda)$[1] and (3.9) can be written as

$$\hat{\nabla}f(\lambda) = \hat{c}_\lambda + \hat{Z}_\lambda^\top v_k(\lambda),\ \nabla f(\lambda) = c_\lambda + Z_\lambda^\top v(\lambda).$$

[1] See point 3 in Algorithm 3.3.2.

Then, we have

$$\begin{aligned}
\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\| &= \|\hat{c}_\lambda + \hat{Z}_\lambda^\top v_k(\lambda) - c_\lambda - Z_\lambda^\top v(\lambda)\| \\
&\leq \|\hat{c}_\lambda - c_\lambda\| \\
&\quad + \|\hat{Z}_\lambda^\top v_k(\lambda) - \hat{Z}_\lambda^\top v(\lambda) + \hat{Z}_\lambda^\top v(\lambda) - Z_\lambda^\top v(\lambda)\| \\
&\leq \|\hat{c}_\lambda - c_\lambda\| \\
&\quad + \|\hat{Z}_\lambda\| \|v_k(\lambda) - v(\lambda)\| + \|\hat{Z}_\lambda - Z_\lambda\| \|v(\lambda)\| \\
&\leq \|\hat{c}_\lambda - c_\lambda\| \\
&\quad + \|\hat{Z}_\lambda\| \|v_k(\lambda) - v(\lambda)\| + \|\hat{Z}_\lambda - Z_\lambda\| \|A_\lambda^{-1}\| \|b_\lambda\| \\
&\leq \left(\mu_{2,\lambda} + \frac{\nu_{2,\lambda} L_{E,\lambda}}{\hat{\mu}_\lambda}\right) \rho_\lambda(t) D_\lambda + L_{\Phi,\lambda} \|v_k(\lambda) - v(\lambda)\|.
\end{aligned}$$

Moreover, it follows from 3.3.9(iii) that

$$\begin{aligned}
\|v_k(\lambda) - v(\lambda)\| &\leq \|v_k(\lambda) - \hat{v}(\lambda)\| + \|\hat{v}(\lambda) - v(\lambda)\| \\
&\leq \sigma_\lambda(k) \frac{L_{E,\lambda}}{\hat{\mu}_\lambda} + \|\hat{v}(\lambda) - v(\lambda)\|.
\end{aligned}$$

Finally, we have

$$\begin{aligned}
\|\hat{v}(\lambda) - v(\lambda)\| &\leq \|\hat{A}_\lambda^{-1} \hat{b}_\lambda - \hat{A}_\lambda^{-1} b_\lambda + \hat{A}_\lambda^{-1} b_\lambda - A_\lambda^{-1} b_\lambda\| \\
&\leq \|\hat{A}_\lambda^{-1}\| \|\hat{b}_\lambda - b_\lambda\| + \|b_\lambda\| \|\hat{A}_\lambda^{-1} - A_\lambda^{-1}\| \\
&\leq \frac{\mu_{1,\lambda} \rho_\lambda(t) D_\lambda}{\hat{\mu}_\lambda} + L_{E,\lambda} \|\hat{A}_\lambda^{-1} - A_\lambda^{-1}\| \\
&\leq \frac{\mu_{1,\lambda} \rho_\lambda(t) D_\lambda}{\hat{\mu}_\lambda} + L_{E,\lambda} \|\hat{A}_\lambda^{-1}\| \|A_\lambda - \hat{A}_\lambda\| \|A_\lambda^{-1}\| \\
&\leq \frac{\mu_{1,\lambda} \rho_\lambda(t) D_\lambda}{\hat{\mu}_\lambda} + \frac{L_{E,\lambda} \nu_{1,\lambda} \rho_\lambda(t) D_\lambda}{\hat{\mu}_\lambda^2}.
\end{aligned}$$

Combining all together we get (3.13). As regards the second part of the statement, if Assumption 3.3.5 is satisfied, then, in view of Lemma 3.3.6, we can take $\hat{\mu}_\lambda = 1 - q_\lambda$ in (3.12) and obtain (3.14). □

The following two propositions allow us to derive the refined iteration complex-

ity bound for AID-FP.

Proposition A.1.2. *Suppose that* (3.19) *holds. Let* $\lambda \in \Lambda, t \in \mathbb{N}$. *Let* $u_{t,0}(\lambda) = 0 \in \mathbb{R}^{d\times n}$ *and for every integer* $k \geq 1$,

$$u_{t,k}(\lambda) = \partial_1\Phi(w_t(\lambda),\lambda)u_{t,k-1}(\lambda) + \partial_2\Phi(w_t(\lambda),\lambda).$$

Then, for every $k \in \mathbb{N}$,

$$u_{t,k}(\lambda)^\top \nabla_1 E(w_t(\lambda),\lambda) = \partial_2\Phi(w_t(\lambda),\lambda)^\top v_k(\lambda). \tag{A.6}$$

Proof. We set $Y = \partial_1\Phi(w_t(\lambda),\lambda) \in \mathbb{R}^{d\times d}$, $C = \partial_2\Phi(w_t(\lambda),\lambda) \in \mathbb{R}^{d\times n}$, and $b = \nabla_1 E(w_t(\lambda),\lambda) \in \mathbb{R}^d$. Let $k \in \mathbb{N}$, $k \geq 1$. Then,

$$\begin{aligned}
u_{t,k}(\lambda) &= Yu_{t,k-1}(\lambda) + C \\
&= Y^2 u_{t,k-2}(\lambda) + (1+Y)C \\
&\vdots \\
&= Y^k u_{t,0}(\lambda) + \sum_{i=0}^{k-1} Y^i C \\
&= \sum_{i=0}^{k-1} Y^i C.
\end{aligned}$$

In the same way, it follows from (3.19) that $v_k(\lambda) = Y^\top v_{t,k-1}(\lambda) + b = \sum_{i=0}^{k-1}(Y^\top)^i b$. Therefore, we have

$$\begin{aligned}
u_{t,k}(\lambda)^\top b &= C^\top \left(\sum_{i=0}^{k-1} Y^i\right)^\top b \\
&= C^\top \sum_{s=i}^{k-1} (Y^\top)^i b \\
&= C^\top v_k(\lambda)
\end{aligned}$$

and the statement follows. □

Using Proposition A.1.2, for AID-FP we can write

$$\hat{\nabla} f(\lambda) = \nabla_2 E(w_t(\lambda), \lambda) + u_{t,k}(\lambda)^\top \nabla_1 E(w_t(\lambda), \lambda). \tag{A.7}$$

Then a result similar to Proposition 3.3.7 can be derived.

Proposition A.1.3. *Suppose that Assumption 3.3.1(i)(iii) and Assumption 3.3.5 hold. Let $\lambda \in \Lambda$ and $(u_{t,k}(\lambda))_{k\in\mathbb{N}}$ be defined as in Proposition A.1.2. Then, for every $t,k \in \mathbb{N}$, with $t \geq 1$,*

$$\|u_{t,k}(\lambda) - w'(\lambda)\| \leq \left(\nu_{2,\lambda} + \nu_{1,\lambda}\frac{L_{\Phi,\lambda}}{(1-q_\lambda)}\right)\frac{D_\lambda(1-q_\lambda^k)}{1-q_\lambda}\rho_\lambda(t) + \frac{L_{\Phi,\lambda}}{1-q_\lambda}q_\lambda^k.$$

Proof. Let $t,k \in \mathbb{N}$, with $t,k \geq 1$. Recalling that

$$\begin{aligned} u_{t,k}(\lambda) &= \partial_1\Phi(w_t(\lambda),\lambda)u_{t,k-1}(\lambda) + \partial_2\Phi(w_t(\lambda),\lambda) \\ w'(\lambda) &= \partial_1\Phi(w(\lambda),\lambda)w'(\lambda) + \partial_2\Phi(w(\lambda),\lambda) \end{aligned}$$

we can bound the norm of the difference as follows

$$\begin{aligned} &\|u_{t,k}(\lambda) - w'(\lambda)\| \\ &\quad\leq \|\partial_1\Phi(w_t(\lambda),\lambda) - \partial_1\Phi(w(\lambda),\lambda)\|\|w'(\lambda)\| \\ &\qquad + \|\partial_1\Phi(w_t(\lambda),\lambda)\|\|u_{t,k-1}(\lambda) - w'(\lambda)\| \\ &\qquad + \|\partial_2\Phi(w_t(\lambda),\lambda) - \partial_2\Phi(w(\lambda),\lambda)\| \\ &\quad\leq (\nu_{2,\lambda} + \nu_{1,\lambda}L_{\Phi,\lambda}/(1-q_\lambda))\|w_t(\lambda) - w(\lambda)\| \\ &\qquad + q_\lambda\|u_{t,k-1}(\lambda) - w'(\lambda)\|, \end{aligned}$$

which gives a recursive inequality. Then, setting $p := \nu_{2,\lambda} + \nu_{1,\lambda}L_{\Phi,\lambda}/(1-q_\lambda)$, $\Delta_t := \|w_t(\lambda) - w(\lambda)\|$ and $\hat{\Delta}'_k := \|u_{t,k}(\lambda) - w'(\lambda)\|$, we have

$$\hat{\Delta}'_k \leq q_\lambda\hat{\Delta}'_{k-1} + p\Delta_t.$$

Therefore, it follows from Lemma A.1.1(i) with $\tau = p\Delta_t$, that

$$\begin{aligned}\hat{\Delta}'_k &\leq q_\lambda^k \hat{\Delta}'_0 + p\Delta_t \frac{1-q_\lambda^k}{1-q_\lambda} \\ &\leq \frac{L_{\Phi,\lambda}}{1-q_\lambda} q_\lambda^k + pD_\lambda \rho_\lambda(t) \frac{1-q_\lambda^k}{1-q_\lambda},\end{aligned}$$

where in the last inequality we used Assumption 3.3.9(ii) and (see (A.5)) $\hat{\Delta}'_0 = \|u_{t,0}(\lambda) - w'(\lambda)\| = \|w'(\lambda)\| \leq L_{\Phi,\lambda}/(1-q_\lambda)$. The statement follows. □

Theorem 3.3.12. *(AID-FP bound) Suppose that Assumptions 3.3.1(i)(iii)(iv) and Assumption 3.3.5 hold. Suppose also that* (3.19) *holds. Let* $\hat{\nabla} f(\lambda)$ *be defined according to Algorithm 3.3.2 and* $\hat{\Delta} = \|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|$. *Then, for every* $t, k \in \mathbb{N}$,

$$\hat{\Delta} \leq \Big(c_1(\lambda) + c_2(\lambda) \frac{1-q_\lambda^k}{1-q_\lambda} \Big) \rho_\lambda(t) + c_3(\lambda) q_\lambda^k, \tag{3.21}$$

where $c_1(\lambda)$, $c_2(\lambda)$ *and* $c_3(\lambda)$ *are given in Theorem 3.3.8.*

Proof. Let $t \in \mathbb{N}$ with $t \geq 1$ and let $(u_{t,k}(\lambda))_{k\in\mathbb{N}}$ be defined as in Proposition A.1.2. Then, the difference between exact and approximate gradients can be bound as follows

$$\begin{aligned}&\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\| \\ &\quad \leq \|\nabla_2 E(w_t(\lambda), \lambda) - \nabla_2 E(w(\lambda), \lambda)\| \\ &\qquad + \|w'(\lambda)\| \|\nabla_1 E(w_t(\lambda), \lambda) - \nabla_1 E(w(\lambda), \lambda)\| \\ &\qquad + \|w'(\lambda) - u_{t,k}(\lambda)\| \|\nabla_1 E(w_t(\lambda), \lambda)\|.\end{aligned}$$

Now note that $\|w_t(\lambda)\| \leq \|w_t(\lambda) - w(\lambda)\| + \|w(\lambda)\| \leq (\rho_\lambda(t) + 1)\|w(\lambda)\| \leq 2D_\lambda$. Then it follows from the assumptions and Lemmas 3.3.6 and 3.3.4 that

$$\begin{aligned}\|\nabla f(\lambda) - \hat{\nabla} f(\lambda)\| \leq &\left(\mu_{2,\lambda} + \frac{\mu_{1,\lambda} L_{\Phi,\lambda}}{1-q_\lambda} \right) \rho_\lambda(t) D_\lambda \\ &+ L_{E,\lambda} \|u_{t,k}(\lambda) - w'(\lambda)\|,\end{aligned}$$

and the last term can be bounded using Proposition A.1.3. □

A.2 Gradient Descent as a Contraction Map

Consider problem (3.2) and take

$$\Phi(w,\lambda) = w - \alpha(\lambda)\nabla_1\mathcal{L}(w,\lambda),$$

where $\ell\colon \mathbb{R}^d \times \Lambda \to \mathbb{R}$ is twice continuously differentiable and, for every $\lambda \in \Lambda$,

(i) $\ell(\cdot,\lambda)$ is $\tau_{\mathcal{L}}(\lambda)$-strongly convex and $L_{\mathcal{L}}(\lambda)$-Lipschitz smooth, with $\tau_{\mathcal{L}}(\lambda) > 0$ and $L_{\mathcal{L}}(\lambda) > 0$.

(ii) $\alpha\colon \Lambda \subset \mathbb{R}^n \to \mathbb{R}_{++}$ is differentiable.

Then, if $\alpha(\lambda) \in (0, 2/L_{\mathcal{L}}(\lambda))$, $\Phi(\cdot,\lambda)$ is a contraction with constant $q_\lambda = \max\{1 - \alpha(\lambda)\tau_{\mathcal{L}}(\lambda), \alpha(\lambda)L_{\mathcal{L}}(\lambda) - 1\}$. The optimal choice of the step-size leads to set $\alpha(\lambda) = 2/(L_{\mathcal{L}}(\lambda) + \tau_{\mathcal{L}}(\lambda))$ giving

$$q_\lambda = \frac{L_{\mathcal{L}}(\lambda) - \tau_{\mathcal{L}}(\lambda)}{L_{\mathcal{L}}(\lambda) + \tau_{\mathcal{L}}(\lambda)} = \frac{\kappa(\lambda) - 1}{\kappa(\lambda) + 1},$$

where $\kappa(\lambda) = L_{\mathcal{L}}(\lambda)/\tau_{\mathcal{L}}(\lambda)$ is the condition number of the lower-level problem in (3.2). Note that, for every $t \in \mathbb{N}$ and $\lambda \in \Lambda$,

$$\tau_{\mathcal{L}}(\lambda) I \preccurlyeq \nabla_1^2 \ell(w_t(\lambda),\lambda) \preccurlyeq L_{\mathcal{L}}(\lambda) I \tag{A.8}$$

hence the condition number of $\nabla_1^2\ell(w_t(\lambda),\lambda)$ is smaller than $\kappa(\lambda)$.

We can write the derivatives of Φ as:

$$\partial_2\Phi(w,\lambda) = -\nabla_1\mathcal{L}(w,\lambda)\nabla\alpha(\lambda)^\top - \alpha(\lambda)\nabla_{21}^2\mathcal{L}(w,\lambda) \tag{A.9}$$

$$\partial_1\Phi(w,\lambda) = I - \alpha(\lambda)\nabla_1^2\mathcal{L}(w,\lambda) \tag{A.10}$$

Remark A.2.1. *When evaluated in* $(w(\lambda),\lambda)$*, one does not need* $\nabla\alpha(\lambda)$ *for* (A.9)*, because the first term on the r.h.s of eq.* (A.9) *is* 0*:*

$$\partial_2\Phi(w(\lambda),\lambda) = -\alpha(\lambda)\nabla_{21}^2\mathcal{L}(w(\lambda),\lambda).$$

From the remark it follows that $\|\partial_2\Phi(w(\lambda),\lambda)\| = \alpha(\lambda)\|\nabla^2_{21}\mathcal{L}(w(\lambda),\lambda)\|$

Furthermore, if we assume $\nabla^2_1\mathcal{L}(\cdot,\lambda)$ is $\hat{\rho}_{1,\lambda}$-Lipschitz and $\nabla^2_{21}\mathcal{L}(\cdot,\lambda)$ is $\hat{\rho}_{2,\lambda}$-Lipschitz then, calling $\Delta_{\partial_1\Phi} := \|\partial_1\Phi(w_1,\lambda) - \partial_1\Phi(w_2,\lambda)\|$ and $\Delta_{\partial_2\Phi} := \|\partial_2\Phi(w_1,\lambda) - \partial_2\Phi(w_2,\lambda)\|$ we have:

$$\Delta_{\partial_1\Phi} = \|\alpha(\lambda)\left(\nabla^2_1\mathcal{L}(w_1,\lambda) - \nabla^2_1\mathcal{L}(w_2,\lambda)\right)\| \leq \alpha(\lambda)\hat{\rho}_{1,\lambda}\|w_1 - w_2\|,$$

and

$$\begin{aligned}\Delta_{\partial_2\Phi} &= \|(\nabla_1\mathcal{L}(w_1,\lambda) - \nabla_1\mathcal{L}(w_2,\lambda))\,\nabla\alpha(\lambda)^\top + \alpha(\lambda)\left(\nabla^2_{21}\mathcal{L}(w_1,\lambda) - \nabla^2_{21}\mathcal{L}(w_2,\lambda)\right)\| \\ &\leq (L_{\mathcal{L}}(\lambda)\|\nabla\alpha(\lambda)\| + \alpha(\lambda)\hat{\rho}_{2,\lambda})\|w_1 - w_2\|.\end{aligned}$$

Thus, Assumption 3.3.1(iii) holds with $\nu_{1,\lambda} = \alpha(\lambda)\hat{\rho}_{1,\lambda}$ and $\nu_{2,\lambda} = L_{\mathcal{L}}(\lambda)\|\nabla\alpha(\lambda)\| + \alpha(\lambda)\hat{\rho}_{2,\lambda}$. Moreover, if we pick $L_{\mathcal{L},\lambda}$ such that $\|\nabla^2_{21}\mathcal{L}(w(\lambda),\lambda)\| \leq L_{\mathcal{L},\lambda}$, then Theorems 3.3.8,3.3.10 and 3.3.12 hold with

$$\begin{aligned} q_\lambda &= \max\{1 - \alpha(\lambda)\tau_{\mathcal{L}}(\lambda), \alpha(\lambda)L_{\mathcal{L}}(\lambda) - 1\} \\ c_1(\lambda) &:= \left(\mu_{2,\lambda} + \frac{\mu_{1,\lambda}\alpha(\lambda)L_{\mathcal{L},\lambda}}{1 - q_\lambda}\right) D_\lambda \\ c_2(\lambda) &:= \left(L_{\mathcal{L}}(\lambda)\|\nabla\alpha(\lambda)\| + \alpha(\lambda)\hat{\rho}_{2,\lambda}\right) L_{E,\lambda}\, D_\lambda \\ &\quad + \frac{\hat{\rho}_{1,\lambda}\alpha(\lambda)^2 L_{\mathcal{L},\lambda}L_{E,\lambda}\, D_\lambda}{1 - q_\lambda} \\ c_3(\lambda) &:= \frac{L_{E,\lambda}\,\alpha(\lambda)\|\nabla^2_{21}\mathcal{L}(w(\lambda),\lambda)\|}{(1 - q_\lambda)}. \end{aligned}$$

Given Remark A.2.1 and to avoid additional complexity of the algorithm, we can consider replacing $\partial_2\Phi(w,\lambda)$ with $\hat{\partial}_2\Phi(w,\lambda) = -\alpha(\lambda)\nabla^2_{21}\mathcal{L}(w,\lambda)$ in the expression for both $\nabla f_t(\lambda)$ and $\hat{\nabla} f(\lambda)$. We apply this change in all the experiments of case (3.2).

A.3 Additional Details on the Experiments

A.3.1 Hypergradient Approximation

In this section we provide additional details for the experiments in Section 3.4.1.

We define the train and validation kernel matrices in (3.22) as follows:

$$K'(\gamma)_{i,j} = \exp\left[-\left(X'_i - X_j\right)^\top \mathrm{d}iag(\gamma)\left(X'_i - X_j\right)\right]$$
$$K(\gamma)_{i,j} = \exp\left[-\left(X_i - X_j\right)^\top \mathrm{d}iag(\gamma)\left(X_i - X_j\right)\right].$$

We generate synthetic data by sampling each element of X and X' from a normal distribution. y (and in the same way y') is subsequently obtained in the following ways for the different settings outlined in Section 3.4.1.

$$y = sign(Xw_* + m\varepsilon) \qquad \text{(LR)}$$
$$y = Xw_* + m\varepsilon \qquad \text{(KRR)}$$
$$y = X(w_* + b_*) + m\varepsilon \qquad \text{(BR)}$$
$$y = XH_*w_* + m\varepsilon \qquad \text{(HR)}$$

where *sign* is the element wise sign function, each element of ε, w_* and H_* is sampled from a normal distribution, $b_* = \mathbb{1}$[2], and $m = 0.1$. X,X' have dimension 50×100 while H is a 100×200 matrix. The results in Figure 3.1 report mean and std over 20 values of λ such that $\lambda_i \sim \mathcal{U}(\lambda_{\min}, \lambda_{\max})$ for $1 \leq i \leq n$ where $\mathcal{U}$ is the uniform distribution on the interval $[\lambda_{\min}, \lambda_{\max}]$ which is $[0.01, 10]$, $[0.0005, 0.005]$, $[-5, 5]$ and $[-1, 1]$ respectively for LR, KRR, BR, HR. Furthermore, we set $\beta = 1$ for BR and $\beta = 10$ for HR. $\lambda_{\min}$, $\lambda_{\max}$ and β are selected as to make the expected lower-level problem difficult (q_λ close to 1).

We note that in Figure 3.1 the asymptotic error for KRR, BR and HR is considerably large. We suspect that this is due to the numerical error made by the hypergradient approximation procedures being larger than the one made when computing the exact hypergradient using the closed form expression of $w(\lambda)$. Indeed,

[2]Where $\mathbb{1} \in \mathbb{R}^d$ is a vector with all its components set to one.

we have observed that using double precision halves the asymptotic error, but we did not investigate further. Our theoretical analysis does not take this source of error into account since it assumes infinite precision arithmetic.

A.3.2 Bilevel Optimization

This section contains the details and some additional results on the experiments in Section 3.4.2 on problems of type (3.2)[3].

The average cross-entropy in 20 newsgroups and Fashion MNIST is defined as

$$\mathrm{CE}(Z,y) = -\frac{1}{|y|}\sum_{k=1}^{|y|}\sum_{i=1}^{c}\delta_{i,y_k}\log\left(\frac{e^{Z_{ki}}}{\sum_{j=1}^{c}e^{Z_{kj}}}\right)$$

where $Z_k \in \mathbb{R}^c$ $y_k \in \{0,\dots,c\}$ are respectively the prediction scores and the class label for the k-th example, δ_{i,y_k} equals 1 when $i = y_k$ and 0 otherwise and $|y|$ is the number of examples.

To solve the upper-level problem we use gradient descent with fixed step-size where the gradient is estimated using ITD or AID methods. In particular, we generate the sequence $(\lambda_i)_i^s$ as follows:

$$\lambda_i = \lambda_{i-1} - \zeta g(\lambda_{i-1})$$

where $g(\lambda) = \nabla f_t(\lambda)$ for ITD and $g(\lambda) = \hat{\nabla} f(\lambda)$ for AID are computed respectively using Algorithm 3.3.1 and Algorithm 3.3.2 with t and k fixed throughout the optimization.

All methods compute $w_t(\lambda)$ using t-steps of the same algorithm solving the lower-level problem in (3.2). In particular, we use heavy ball with optimal constants for Parkinson and gradient descent with step-size manually chosen for the other two settings where it is harder to compute the optimal one. Specifically, we set the step-size to 10^3 for 20 newsgroups and 10^4 for Fashion MNIST[4].

The initial parameter λ_0 is set to $(\beta_0, \gamma_0) = (0, -\log(p)\mathbb{1})$[5] for Parkinson, $0 \in$

[3]This includes all the settings except equilibrium models.

[4]Note that in this case the step-size is constant w.r.t. λ whereas the optimal one would vary with λ.

[5]Where $\mathbb{1} \in \mathbb{R}^p$ is a vector with all its components set to one.

$\mathbb{R}^p$ for 20 newsgroup and $X_0 = 0 \in \mathbb{R}^{c \times p}$ for Fashion MNIST. Furthermore, the regularization parameter β is set to 1 for Fashion MNIST.

We choose the step-size ζ with a grid search over 30 values in a suitable interval for each problem, choosing the one bringing the lowest value of the approximate objective $f_t(\lambda_s) = E(w_t(\lambda_s), \lambda_s)$ where s is equal to 1000, 500, 4000 for Parkinson, 20 newsgroups and Fashion MNIST respectively. The grid search values are spaced evenly in log scale inside the intervals $[10^{-6}, 10]$, $[10^{-4}, 10^4]$ and $[10^{-10}, 10^{-2}]$ respectively for Parkinson, 20 newsgroups and Fashion MNIST.

We note that the results In Table 3.1 report the value of the approximate objective $f_t(\lambda_s) = E(w_t(\lambda_s), \lambda_s)$ and the test accuracy (computed on $w_t(\lambda_s)$). For completeness, in Table A.1 we report $f(\lambda_s) = E(w(\lambda_s), \lambda_s))$ and the test accuracy (computed on $w(\lambda_s)$) where $w(\lambda_s)$ is computed using RMAD (exploiting the closed form of $w(\lambda_s)$) for Parkinson and using 2000 steps of gradient descent starting from $w_0(\lambda) = 0$ for 20 newsgroups and Fashion MNIST.

Table A.1: The values of $f(\lambda_s)$ and test accuracy (in percentage) are displayed after s gradient descent steps, where s is 1000, 500 and 4000 for Parkinson, 20 news and Fashion MNIST respectively. $k_r = 10$ for Parkinson and 20 news while for Fashion MNIST $k_r = 5$.

Parkinson

	$t = 100$	$t = 150$
ITD	2.39 (75.8)	2.11 (69.7)
FP $k = t$	2.36 (81.8)	2.19 (77.3)
CG $k = t$	2.20 (78.8)	2.19 (77.3)
FP $k = k_r$	2.71 (80.3)	2.60 (78.8)
CG $k = k_r$	2.17 (78.8)	1.99 (77.3)

20 newsgroup

	$t = 10$	$t = 25$	$t = 50$
ITD	1.155 (59.4)	1.082 (61.1)	1.058 (61.6)
FP $k = t$	1.155 (59.5)	1.083 (61.1)	1.058 (61.6)
CG $k = t$	0.983 (62.9)	0.955 (62.9)	0.946 (63.5)
FP $k = k_r$	1.155 (59.5)	1.078 (61.7)	1.160 (59.1)
CG $k = k_r$	0.983 (62.9)	0.989 (62.6)	1.001 (62.3)

Fashion MNIST

	$t = 5$	$t = 10$
ITD	0.497 (84.1)	0.431 (83.8)
FP $k = t$	0.497 (84.1)	0.431 (83.8)
CG $k = t$	0.522 (83.8)	0.424 (84.0)
FP $k = k_r$	0.497 (84.1)	0.426 (83.9)
CG $k = k_r$	0.522 (83.8)	0.424 (84.0)

Appendix B

Appendix for Chapter 4

This appendix is organized as follows. Appendices B.1 and B.2 contain the proofs for the results presented in the chapter. In Appendix B.4 we provide statements and proofs for some standard lemmas which are instrumental for the main results.

B.1 Proofs of Section 4.4

We start by proving two lemmas.

Lemma B.1.1. *Let Assumption 4.3.1 be satisfied. Then, for every $w \in \mathbb{R}^d$*

$$\|v(w(\lambda),\lambda) - v(w,\lambda)\| \leq \left(\frac{\nu_{1,\lambda} L_{E,\lambda}}{(1-q_\lambda)^2} + \frac{\mu_{1,\lambda}}{1-q_\lambda}\right) \|w(\lambda) - w\|. \tag{B.1}$$

Proof. Let $A_1 := (I - \partial_1 \Phi(w(\lambda),\lambda)^\top)$ and $A_2 = (I - \partial_1 \Phi(w,\lambda)^\top)$. Then it follows from Lemma B.4.8 that

$$\begin{aligned}
&\|v(w(\lambda),\lambda) - v(w,\lambda)\| \\
&\qquad \leq \|\nabla_1 E(w(\lambda),\lambda)\| \|A_1^{-1} - A_2^{-1}\| + \mu_{1,\lambda} \|A_2^{-1}\| \|w(\lambda) - w\| \\
&\qquad \leq \|\nabla_1 E(w(\lambda),\lambda)\| \|A_1^{-1}(A_2 - A_1)A_2^{-1}\| + \frac{\mu_{1,\lambda}}{1-q_\lambda} \|w(\lambda) - w\| \\
&\qquad \leq \left(\frac{\nu_{1,\lambda}}{(1-q_\lambda)^2} \|\nabla_1 E(w(\lambda),\lambda)\| + \frac{\mu_{1,\lambda}}{1-q_\lambda}\right) \|w(\lambda) - w\|.
\end{aligned}$$

Moreover, Assumption 4.3.1 yields that $\|\nabla_1 E(w(\lambda),\lambda)\| \leq L_{E,\lambda}$. Hence, the statement follows. □

Lemma B.1.2. *Let Assumption 4.3.1 be satisfied. Then, for every $w \in \mathbb{R}^d$*

$$\|v(w,\lambda)\| \leq \|(I - \partial_1\Phi(w,\lambda)^\top)^{-1}\|\|\nabla_1 E(w,\lambda)\| \leq \frac{L_{E,\lambda}}{1-q_\lambda}. \tag{B.2}$$

Proof. It follows from the definition of $v(w,\lambda)$ and Assumptions 4.3.1(i) and 4.3.1(iv) □

B.1.1 Proof of Theorem 4.4.2

Proof. (i): Using the definition of $\hat{\nabla} f(\lambda)$ and the fact that ζ_j' and $v_k(w_t(\lambda),\lambda)$ are independent random variables, we get

$$\mathbb{E}[\hat{\nabla} f(\lambda) \,|\, w_t(\lambda)] = \nabla_2 E(w_t(\lambda),\lambda) + \partial_2\Phi(w_t(\lambda),\lambda)^\top \mathbb{E}[v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)].$$

Consequently, recalling the hypergradient equation, we have,

$$\begin{aligned}
&\big\|\mathbb{E}[\hat{\nabla} f(\lambda) \,|\, w_t(\lambda)] - \nabla f(\lambda)\big\| \\
&\quad \leq \|\nabla_2 E(w(\lambda),\lambda) - \nabla_2 E(w_t(\lambda),\lambda)\| \\
&\qquad + \big\|\partial_2\Phi(w(\lambda),\lambda)^\top v(w(\lambda),\lambda) - \partial_2\Phi(w_t(\lambda),\lambda)^\top \mathbb{E}[v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)]\big\| \\
&\quad \leq \|\nabla_2 E(w(\lambda),\lambda) - \nabla_2 E(w_t(\lambda),\lambda)\| \\
&\qquad + \|\partial_2\Phi(w(\lambda),\lambda)\|\|v(w(\lambda),\lambda) - \mathbb{E}[v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)]\| \\
&\qquad + \|\partial_2\Phi(w(\lambda),\lambda) - \partial_2\Phi(w_t(\lambda),\lambda)\|\|\mathbb{E}[v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)]\|.
\end{aligned} \tag{B.3}$$

Now, concerning the term $\|v(w(\lambda),\lambda) - \mathbb{E}[v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)]\|$ in the above inequality, we have

$$\begin{aligned}
&\|v(w(\lambda),\lambda) - \mathbb{E}[v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)]\| \\
&\quad \leq \|v(w(\lambda),\lambda) - v(w_t(\lambda),\lambda)\| + \|v(w_t(\lambda),\lambda) - \mathbb{E}[v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)]\|.
\end{aligned} \tag{B.4}$$

Since $\mathbb{E}[\bar{v}(w_t(\lambda),\lambda) \,|\, w_t(\lambda)] = v(w_t(\lambda),\lambda)$ we have

$$\|v(w_t(\lambda),\lambda) - \mathbb{E}[v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)]\| = \|\mathbb{E}[\bar{v}(w_t(\lambda),\lambda) - v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)]\|$$

Moreover, using Jensen inequality and Assumption 4.4.1 we obtain

$$\begin{aligned}
&\|\mathbb{E}[\bar{v}(w_t(\lambda),\lambda) - v_k(w_t(\lambda),\lambda)\,|\,w_t(\lambda)]\| \\
&\qquad = \sqrt{\|\mathbb{E}[\bar{v}(w_t(\lambda),\lambda) - v_k(w_t(\lambda),\lambda)\,|\,w_t(\lambda)]\|^2} \qquad \text{(B.5)} \\
&\qquad \leq \sqrt{\mathbb{E}[\|\bar{v}(w_t(\lambda),\lambda) - v_k(w_t(\lambda),\lambda)\|^2\,|\,w_t(\lambda)]} \\
&\qquad \leq \sqrt{\sigma_\lambda(k)}. \qquad \text{(B.6)}
\end{aligned}$$

Therefore, using Lemma B.1.1, (B.4) yields

$$\begin{aligned}
\|v(w(\lambda),\lambda) - \mathbb{E}[v_k(w_t(\lambda),\lambda)\,|\,w_t(\lambda)]\| \leq \left(\frac{\nu_{1,\lambda} L_{E,\lambda}}{(1-q_\lambda)^2} + \frac{\mu_{1,\lambda}}{1-q_\lambda}\right)\|w(\lambda) - w_t(\lambda)\| \\
+ \sqrt{\sigma_\lambda(k)}. \qquad \text{(B.7)}
\end{aligned}$$

In addition, it follows from (B.4)-(B.6) and lemma B.1.2 that

$$\begin{aligned}
&\|\mathbb{E}[v_k(w_t(\lambda),\lambda)\,|\,w_t(\lambda)]\| \\
&\qquad \leq \|v(w_t(\lambda),\lambda)\| + \|v(w_t(\lambda),\lambda) - \mathbb{E}[v_k(w_t(\lambda),\lambda)\,|\,w_t(\lambda)]\| \\
&\qquad \leq \frac{L_{E,\lambda}}{1-q_\lambda} + \sqrt{\sigma_\lambda(k)}. \qquad \text{(B.8)}
\end{aligned}$$

Finally, combining (B.3), (B.7), and (B.8), and using Assumption 4.3.1, (i) follows. Then, since

$$\begin{aligned}
\|\mathbb{E}[\hat{\nabla} f(\lambda)] - \nabla f(\lambda)\| &= \big\|\mathbb{E}\big[\mathbb{E}[\hat{\nabla} f(\lambda)\,|\,w_t(\lambda)] - \nabla f(\lambda)\big]\big\| \\
&\leq \mathbb{E}\big[\big\|\mathbb{E}[\hat{\nabla} f(\lambda)\,|\,w_t(\lambda)] - \nabla f(\lambda)\big\|\big],
\end{aligned}$$

(ii) follows by taking the expectation in (i), using Assumption 4.4.1 and that $\mathbb{E}[\hat{\Delta}_w] = \sqrt{(\mathbb{E}[\hat{\Delta}_w])^2} \leq \sqrt{\mathbb{E}[\hat{\Delta}_w^2]} \leq \sqrt{\rho_\lambda(t)}$.

□

B.1.2 Proof of Theorem 4.4.3

Proof. Let $\tilde{\mathbb{E}}[\cdot] := \mathbb{E}[\cdot \,|\, w_t(\lambda)]$, $\tilde{\mathbb{V}}[\cdot] := \mathbb{V}[\cdot \,|\, w_t(\lambda)]$, $b_2 := \tilde{\mathbb{V}}[\nabla_2 \bar{E}_J(w_t(\lambda),\lambda)]$ and $b_1 := \partial_2\Phi(w_t(\lambda),\lambda)^\top v_k(w_t(\lambda),\lambda)$. Then,

$$
\begin{aligned}
&\tilde{\mathbb{V}}[\hat{\nabla} f(\lambda)] \\
&= \tilde{\mathbb{E}}\big[\|\hat{\nabla} f(\lambda) - \tilde{\mathbb{E}}[\hat{\nabla} f(\lambda)]\|^2\big] \\
&\leq 2\tilde{\mathbb{E}}\big[\|\partial_2\Phi(w_t(\lambda),\lambda)^\top \tilde{\mathbb{E}}[v_k(w_t(\lambda),\lambda)] \mp b_1 - \partial\bar{\Phi}_J(\lambda)^\top v_k(w_t(\lambda),\lambda)\|^2\big] + 2b_2 \\
&\leq 2\|\partial_2\Phi(w_t(\lambda),\lambda)\|^2 \tilde{\mathbb{E}}\big[\|v_k(w_t(\lambda),\lambda) - \tilde{\mathbb{E}}[v_k(w_t(\lambda),\lambda)]\|^2\big] \\
&\quad + 2\tilde{\mathbb{E}}\big[\|v_k(w_t(\lambda),\lambda)\|^2\big]\tilde{\mathbb{E}}\big[\|\partial\bar{\Phi}_J(\lambda) - \partial_2\Phi(w_t(\lambda),\lambda)\|^2\big] + 2b_2. \\
&= 2\underbrace{\|\partial_2\Phi(w_t(\lambda),\lambda)\|^2}_{a_1}\underbrace{\tilde{\mathbb{V}}[v_k(w_t(\lambda),\lambda)]}_{a_2} + 2\underbrace{\tilde{\mathbb{E}}\big[\|v_k(w_t(\lambda),\lambda)\|^2\big]}_{a_3}\tilde{\mathbb{V}}[\partial_2\bar{\Phi}_J(\lambda)] + 2b_2,
\end{aligned}
$$

where for the last inequality we used that $\zeta_i' \perp\!\!\!\perp v_k(w_t(\lambda),\lambda) \,|\, w_t(\lambda)$ and, in virtue of Lemma B.4.7, that

$$
\tilde{\mathbb{E}}\big[\Delta_v^\top \partial_2\Phi(w_t(\lambda),\lambda)(\partial_2\bar{\Phi}_J(w_t(\lambda),\lambda,\zeta) - \partial_2\Phi(w_t(\lambda),\lambda))^\top v_k(w_t(\lambda),\lambda)\big] = 0,
$$

where $\Delta_v := v_k(w_t(\lambda),\lambda) - \tilde{\mathbb{E}}[v_k(w_t(\lambda),\lambda)]$. In the following, we will bound each term of the inequality in order.

$$
\begin{aligned}
a_1 &= \|\partial_2\Phi(w_t(\lambda),\lambda) \mp \partial_2\Phi(w(\lambda),\lambda)\|^2 \\
&\leq 2\|\partial_2\Phi(w(\lambda),\lambda)\|^2 + 2\|\partial_2\Phi(w(\lambda),\lambda) - \partial_2\Phi(w_t(\lambda),\lambda)\|^2 \\
&\leq 2L_{\Phi,\lambda}^2 + 2\nu_{2,\lambda}^2\|w(\lambda) - w_t(\lambda)\|^2.
\end{aligned}
$$

Then, applying Assumption 4.4.1, and Lemma B.4.3(ii)

$$
\begin{aligned}
a_2 = \tilde{\mathbb{V}}[v_k(w_t(\lambda),\lambda)] &\leq \tilde{\mathbb{E}}[\|v_k(w_t(\lambda),\lambda) \mp \bar{v}(w_t(\lambda),\lambda) - v(w_t(\lambda),\lambda)\|^2] \\
&\leq 2\sigma_\lambda(k) + 2\frac{\hat{\sigma}_{1,\lambda}}{J(1-q_\lambda)^2},
\end{aligned}
$$

where in the last inequality, recalling Assumption 4.3.2(iv), we used

$$\begin{gathered}\tilde{\mathbb{E}}\big[\|v(w_t(\lambda),\lambda)-\bar{v}(w_t(\lambda),\lambda)\|^2\big]\leq\\ \|(I-\partial_1\Phi(w_t(\lambda),\lambda)^\top)^{-1}\|^2\tilde{\mathbb{E}}[\|\nabla_1 E(w_t(\lambda),\lambda)-\nabla_1\bar{E}_J(w_t(\lambda),\lambda)\|^2]\leq\\ \|(I-\partial_1\Phi(w_t(\lambda),\lambda)^\top)^{-1}\|^2\tilde{\mathbb{V}}[\nabla_1\bar{E}_J(w_t(\lambda),\lambda)]\leq\\ \frac{\hat{\sigma}_{1,\lambda}}{J(1-q_\lambda)^2}.\end{gathered}\tag{B.9}$$

Furthermore, exploiting Assumption 4.3.1 and 4.4.1, and Lemma B.1.2,

$$\begin{aligned}a_3&=\tilde{\mathbb{E}}\big[\|v_k(w_t(\lambda),\lambda)\mp\bar{v}(w_t(\lambda),\lambda)\mp v(w_t(\lambda),\lambda)\|^2\big]\\&\leq 2\|v(w_t(\lambda),\lambda)\|^2+4\tilde{\mathbb{E}}\big[\|v(w_t(\lambda),\lambda)-\bar{v}(w_t(\lambda),\lambda)\|^2\big]\\&\quad+4\tilde{\mathbb{E}}\big[\|\bar{v}(w_t(\lambda),\lambda)-v_k(w_t(\lambda),\lambda)\|^2\big]\\&\leq 2\frac{L_{E,\lambda}^2}{(1-q_\lambda)^2}+4\frac{\hat{\sigma}_{1,\lambda}}{J(1-q_\lambda)^2}+4\sigma_\lambda(k),\end{aligned}$$

where we used (B.9) in the last inequality. Using the formula for the variance of the sum of independent random variables and Assumption 4.3.2 we have

$$\tilde{\mathbb{V}}[\partial\bar{\Phi}_J(\lambda)]\leq\frac{\sigma'_{2,\lambda}}{J},\quad\tilde{\mathbb{V}}[\nabla_2\bar{E}_J(w_t(\lambda),\lambda)]\leq\frac{\hat{\sigma}_{2,\lambda}}{J}.$$

Combining the previous bounds together and defining $\hat{\Delta}_w:=\|w(\lambda)-w_t(\lambda)\|$ and simplifying some terms knowing that $J>1$ we get that

$$\begin{aligned}\tilde{\mathbb{V}}[\hat{\nabla}f(\lambda)]\leq&\left(\hat{\sigma}_{2,\lambda}+4\frac{\sigma'_{2,\lambda}(L_{E,\lambda}^2+\hat{\sigma}_{1,\lambda})+L_{\Phi,\lambda}^2\hat{\sigma}_{1,\lambda}}{(1-q_\lambda)^2}\right)\frac{2}{J}+8(L_{\Phi,\lambda}^2+\sigma'_{2,\lambda})\sigma_\lambda(k)\\&+8v_{2,\lambda}^2\Delta_w^2\left(\sigma_\lambda(k)+\frac{\hat{\sigma}_{1,\lambda}}{J(1-q_\lambda)^2}\right).\end{aligned}$$

The proof is completed by taking the total expectation on both sides of the inequality above. □

B.2 Proofs of Section 4.5

Theorem 4.1 (Constant step-size)**.** *Let Assumption 4.5.1 hold and suppose that* $\eta_t = \eta \in \mathbb{R}_{++}$*, for every* $t \in \mathbb{N}$*, and that*

$$\eta \leq \frac{1}{1+\sigma_2}.$$

Let $(w_t)_{t\in\mathbb{N}}$ *be generated according to algorithm* (4.12) *and set* $MSE_{w_t} := \mathbb{E}[\|w_t - w^*\|^2]$*. Then, for all* $t \in \mathbb{N}$*,*

$$MSE_{w_t} \leq (1-\eta(1-q^2))^t \left(MSE_{w_0} - \frac{\eta\sigma_1}{1-q^2} \right) + \frac{\eta\sigma_1}{1-q^2}. \tag{16}$$

In particular, $\lim_{t\to\infty} MSE_{w_t} \leq \eta\sigma_1/(1-q^2)$.

Proof. Let $\mathfrak{W}_t$ be the σ-algebra generated by $w_0, w_1, \cdots, w_t$. Then

$$\begin{aligned}
\mathbb{E}[\|w_{t+1} - w^*\|^2 \,|\, \mathfrak{W}_t] &= \mathbb{E}[\|w_t - w^* + \eta(\hat{T}(w_t,\zeta_t) - w_t)\|^2 \,|\, \mathfrak{W}_k] \\
&= \|w_t - w^*\|^2 + \eta^2 \mathbb{E}[\|(\hat{T}(w_t,\zeta_t) \mp T(w_t) - w_t)\|^2 \,|\, \mathfrak{W}_k] \\
&\quad + 2\eta(w_t - w^*)^\top (T(w_t) - w_t) \\
&= \|w_t - w^*\|^2 + \eta^2 \|T(w_t) - w_t\|^2 + \eta^2 \mathbb{V}[\hat{T}(w_t,\zeta_t) \,|\, \mathfrak{W}_t] \\
&\quad + 2\eta(w_t - w^*)^\top (T(w_t) - w_t) \\
&\leq \|w_t - w^*\|^2 + \eta^2(1+\sigma_2)\|T(w_t) \mp w^* - w_t\|^2 + \eta^2\sigma_1 \\
&\quad + 2\eta(w_t - w^*)^\top (T(w_t) \mp w^* - w_t) \\
&\leq (1 - 2\eta + \eta^2(1+\sigma_2))\|w_t - w^*\|^2 \\
&\quad + \eta^2(1+\sigma_2)\|T(w_t) - w^*\|^2 + \eta^2\sigma_1 \\
&\quad + \eta(1 - \eta(1+\sigma_2))2(w_t - w^*)^\top (T(w_t) - w^*).
\end{aligned}$$

Furthermore, since $\|T(w_t) - w^*\| \leq q\|w_t - w^*\|$ and $2ab \leq a^2 + b^2$, we have that

$$2(w_t - w^*)^\top (T(w_t) - w^*) \leq 2\|w_t - w^*\|\|T(w_t) - w^*\| \leq (1+q^2)\|w_t - w^*\|^2.$$

From the upper bound on the step size we have that $\eta(1-\eta(1+\sigma_2)) \geq 0$, hence:

$$\begin{aligned}\mathbb{E}[\|w_{t+1}-w^*\|^2 \,|\, \mathfrak{W}_t] &\leq (1-2\eta)\|w_t-w^*\|^2 + \eta^2(1+\sigma_2)(1+q^2)\|w_t-w^*\|^2 \\ &\quad + \eta^2\sigma_1 + \eta(1+q^2)\|w_t-w^*\|^2 \\ &\quad - \eta^2(1+\sigma_2)(1+q^2)\|w_t-w^*\|^2 \\ &\leq (1-\eta(1-q^2))\|w_t-w^*\|^2 + \eta^2\sigma_1.\end{aligned}$$

Taking total expectations we get

$$\mathbb{E}[\|w_{t+1}-w^*\|^2] \leq (1-\eta(1-q^2))\mathbb{E}[\|w_t-w^*\|^2] + \eta^2\sigma_1$$

and subtracting $\eta\sigma_1/(1-q^2)$ from both sides we obtain

$$\mathbb{E}[\|w_{t+1}-w^*\|^2] - \frac{\eta\sigma_1}{1-q^2} \leq (1-\eta(1-q^2))\left(\mathbb{E}[\|w_t-w^*\|^2] - \frac{\eta\sigma_1}{1-q^2}\right).$$

Now the statement follows by applying the above inequality recursively. □

Theorem 4.2 (Decreasing step-sizes)**.** *Let Assumption 4.5.1 hold and suppose that for every $t \in \mathbb{N}$*

$$\eta_t \leq \frac{1}{1+\sigma_2}, \quad \sum_{t=1}^{\infty}\eta_t = \infty, \quad \sum_{t=1}^{\infty}\eta_t^2 < \infty. \tag{20}$$

Let $(w_t)_{t\in\mathbb{N}}$ be generated according to Algorithm (4.12)*. Then*

$$w_t \to w^* \qquad \mathbb{P}\text{-a.s.}$$

Moreover, if $\eta_t = \beta/(\gamma+t)$, with $\beta > 1/(1-q^2)$ and $\gamma \geq \beta(1+\sigma_2)$, then we have

$$\mathbb{E}[\|w_t-w^*\|^2] \leq \frac{c}{\gamma+t}, \tag{21}$$

where

$$c := \max\left\{\gamma\mathbb{E}[\|w_0-w^*\|^2], \frac{\beta^2\sigma_1}{\beta(1-q^2)-1}\right\}.$$

Proof. As in the proof of Theorem 4.5.2 we get

$$(\forall t \in \mathbb{N}) \quad \mathbb{E}[\|w_{t+1} - w^*\|^2 \,|\, \mathfrak{W}_t] \leq (1 - \eta_t(1-q^2))\|w_t - w^*\|^2 + \eta_t^2 \sigma_1. \quad \text{(B.10)}$$

Taking total expectations we obtain

$$(\forall t \in \mathbb{N}) \quad \mathbb{E}[\|w_{t+1} - w^*\|^2] \leq (1 - \eta_t(1-q^2))\mathbb{E}[\|w_t - w^*\|^2] + \eta_t^2 \sigma_1, \quad \text{(B.11)}$$

which can be equivalently written as

$$(\forall t \in \mathbb{N}) \quad (1-q^2)\eta_t \mathbb{E}[\|w_t - w^*\|^2] \leq \mathbb{E}[\|w_t - w^*\|^2] - \mathbb{E}[\|w_{t+1} - w^*\|^2] + \eta_t^2 \sigma_1.$$

Since the right-hand side is summable (being the sum of a telescopic series and a summable series), we have

$$(1-q^2) \sum_{t=0}^{\infty} \eta_t \mathbb{E}[\|w_t - w^*\|^2] \leq \mathbb{E}[\|w_0 - w^*\|^2] + \sigma_1 \sum_{t=0}^{+\infty} \eta_t^2 < +\infty. \quad \text{(B.12)}$$

Now, it follows from (B.10) that $(\|w_t - w^*\|^2)_{t\in\mathbb{N}}$ is an almost supermartingale (in the sense of Robbins and Siegmund (1971)), hence $\|w_t - w^*\|^2 \to \zeta$ $\mathbb{P}$-a.s. for some positive random variable ζ. Since $\sum_{t=0}^{+\infty} \eta_t = +\infty$, it follows from (B.12) that $\liminf_{t\to+\infty} \mathbb{E}[\|w_t - w^*\|^2] = 0$. Then Fatou's lemma yields that $\mathbb{E}[\zeta] \leq \liminf_{t\to+\infty} \mathbb{E}[\|w_t - w^*\|^2] = 0$. Thus, since ζ is positive, $\zeta = 0$ $\mathbb{P}$-a.s. and hence $x_t \to x_*$ $\mathbb{P}$-a.s.

Concerning the second part of the statement, it is easy to see that the sequence $(\eta_t)_{t\in\mathbb{N}}$ satisfies the assumptions (4.14). We can thus apply eq. (B.11) at each iteration. Let $\Delta_t := \mathbb{E}[\|w_t - w^*\|^2]$, from the definition of c we have that for $t = 0$ $\Delta_0 \leq c/\gamma$. Now, suppose that (4.15) holds at step t. We want to prove that it holds at

$t+1$. Defining $\xi := \gamma + t$, it follows from (B.11) that

$$\begin{aligned}\Delta_{t+1} &\leq \frac{\xi - \beta(1-q^2)}{\xi}\frac{c}{\xi} + \frac{\beta^2\sigma_1}{\xi^2} \\ &= \frac{(\xi-1)}{\xi^2}c - \underbrace{\frac{(\beta(1-q^2)-1)c - \beta^2\sigma_1}{\xi^2}}_{\geq 0 \text{ by the definition of } c \text{ and } \beta} \\ &\leq \frac{c}{\xi+1} = \frac{c}{\gamma+t+1},\end{aligned}$$

where the last inequality derives from $\xi^2 \geq (\xi-1)(\xi+1)$. □

Theorem 4.5.5. *Suppose that Assumption 4.5.4 and Assumption 4.5.1(i)(ii) hold. Then Assumption 4.5.1(iii) holds. In particular, for every $w \in \mathbb{R}^d$,*

$$\mathbb{V}[\hat{T}(w,\zeta)] \leq \underbrace{2\mathbb{V}[\hat{T}(w^*,\zeta)]}_{\sigma_1} + \underbrace{2\frac{L_{\hat{T}}^2 + q^2}{(1-q)^2}}_{\sigma_2}\|T(w) - w\|^2.$$

Proof. Let $w \in \mathbb{R}^d$. Then by Assumption 4.5.1-(ii) and the inequality $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$ we get

$$\begin{aligned}\mathbb{V}[\hat{T}(w,\zeta)] &= \mathbb{E}[\|\hat{T}(w,\zeta) \mp \hat{T}(w^*,\zeta) - T(w)\|^2] \\ &\leq 2\mathbb{E}[\|\hat{T}(w,\zeta) - \hat{T}(w^*,\zeta)\|^2] + 2\mathbb{E}[\|\hat{T}(w^*,\zeta) - T(w)\|^2] \\ &\leq 2\mathbb{E}[\|\hat{T}(w,\zeta) - \hat{T}(w^*,\zeta)\|^2] + 2\mathbb{V}[\hat{T}(w^*,\zeta)] + 2\|T(w^*) - T(w)\|^2.\end{aligned}$$

Therefore, leveraging Assumption 4.5.1-(i) and Assumption 4.5.4 we have

$$\mathbb{V}[\hat{T}(w,\zeta)] \leq 2(L_{\hat{T}}^2 + q^2)\|w - w^*\|^2 + 2\mathbb{V}[\hat{T}(w^*,\zeta)].$$

Finally, note that $\|w-w^*\| \leq \|w - T(w)\| + \|T(w) - w^*\| = \|w - T(w)\| + \|T(w) - T(w^*)\| \leq \|w - T(w)\| + q\|w - w^*\|$. Hence, $\|w - w^*\| \leq \|w - T(w)\|/(1-q)$. The statement follows. □

B.3 Proofs of Section 4.6

B.3.1 Proof of Theorem 4.6.1

Proof. The statement follows by applying Theorem 4.5.2 and 4.5.3 with $\hat{T} = \hat{\Phi}(\cdot,\lambda,\cdot)$ and $\hat{T} = \hat{\Psi}_w(\cdot,\lambda,\cdot)$. To that purpose, in view of those theorems it is sufficient to verify Assumption 4.5.1. This is immediate for $\hat{\Phi}(\cdot,\lambda,\cdot)$, due to Assumptions 4.3.1(i) and 4.3.2. Further, applying 4.3.2(ii) gives the first inequality in (4.24) and (4.25). Concerning $\hat{\Psi}_w(\cdot,\lambda,\cdot)$, let $\tilde{\mathbb{E}}[\cdot] = \mathbb{E}[\cdot \,|\, (\xi_j)_{1\le j\le J}]$ and $\tilde{\mathbb{V}}[\cdot] = \mathbb{V}[\cdot \,|\, (\xi_j)_{1\le j\le J}]$. It follows from Assumptions 4.3.1(i) and 4.3.2(i), that,

$$\tilde{\mathbb{E}}[\hat{\Psi}_w(v,\lambda,\zeta)] = \partial_1\Phi(w,\lambda)v + \nabla_1\bar{E}_J(w,\lambda) =: \Psi_w(v,\lambda).$$

Since $\|\partial_1\Psi_w(v,\lambda)\| = \|\partial_1\Phi(w,\lambda)\| \le q_\lambda$, $\Psi_w(\cdot,\lambda)$ is a contraction with constant q_λ and Assumptions 4.5.1(i)-(ii) are satisfied. From Assumption 4.3.2

$$\tilde{\mathbb{V}}[\hat{\Psi}_w(v,\lambda,\zeta)] \le \|v\|^2\sigma'_{1,\lambda}, \tag{B.13}$$

and

$$\begin{aligned}\|v\| &\le \|\Psi_w(v,\lambda) - v\| + \|\Psi_w(v,\lambda)\| \\ &\le \|\Psi_w(v,\lambda) - v\| + \|\partial_1\Phi(w,\lambda)^\top v + \nabla_1\bar{E}_J(w,\lambda)\| \\ &\le \|\Psi_w(v,\lambda) - v\| + q_\lambda\|v\| + \|\nabla_1\bar{E}_J(w,\lambda)\|.\end{aligned}$$

It follows that

$$\|v\| \le \frac{1}{1-q_\lambda}\left(\|\Psi_w(v,\lambda) - v\| + \|\nabla_1\bar{E}_J(w,\lambda)\|\right). \tag{B.14}$$

Hence, combining (B.13) and (B.14) we obtain

$$\tilde{\mathbb{V}}[\Psi_w(v,\lambda,\zeta)] \le \frac{2\sigma'_{1,\lambda}}{(1-q_\lambda)^2}\|\Psi_w(v,\lambda) - v\|^2 + \frac{2\|\nabla_1\bar{E}_J(w,\lambda)\|^2\sigma'_{1,\lambda}}{(1-q_\lambda)^2},$$

which satisfies Assumption 4.5.1(iii). Thus, we can apply Theorem 4.5.2 and 4.5.3 to obtain results on $v_k(w,\lambda)$ which hold conditioned to $(\xi_j)_{j=1}^J$. The bounds in the second inequality of (4.24) and in (4.27) are finally obtained by taking the total expectation and noting that

$$\mathbb{E}[\|\nabla_1 \bar{E}_J(w,\lambda)\|^2] = \|\nabla_1 E(w,\lambda)\|^2 + \mathbb{V}[\nabla_1 \bar{E}_J(w,\lambda)] \leq L_{E,\lambda}^2 + \hat{\sigma}_{1,\lambda}/J \leq L_{E,\lambda}^2 + \hat{\sigma}_{1,\lambda}.$$

□

B.4 Standard Lemmas

Lemma B.4.1. *Let X be a random vector with values in $\mathbb{R}^d$ and suppose that $\mathbb{E}[\|X\|^2] < +\infty$. Then $\mathbb{E}[X]$ exists in $\mathbb{R}^d$ and $\|\mathbb{E}[X]\|^2 \leq \mathbb{E}[\|X\|^2]$.*

Proof. It follows from Hölder's inequality that $\mathbb{E}[\|X\|] \leq \mathbb{E}[\|X\|^2]$. Therefore, X is Bochner integrable with respect to $\mathbb{P}$ and $\|\mathbb{E}[X]\| \leq \mathbb{E}[\|X\|]$. Hence, using Jensen's inequality we have $\|\mathbb{E}[X]\|^2 \leq (\mathbb{E}[\|X\|])^2 \leq \mathbb{E}[\|X\|^2]$ and the statement follows. □

Definition B.4.2. *Let X be a random vector with value in $\mathbb{R}^d$ such that $\mathbb{E}[\|X\|^2] < +\infty$. Then the variance of X is*

$$\mathbb{V}[X] := \mathbb{E}[\|X - \mathbb{E}[X]\|^2] \tag{B.15}$$

Lemma B.4.3 (Properties of the variance)**.** *Let X and Y be two independent random variables with values in $\mathbb{R}^d$ and let A be a random matrix with values in $\mathbb{R}^{n\times d}$ which is independent on X. We also assume that X, Y, and A have finite second moment. Then the following hold.*

(i) $\mathbb{V}[X] = \mathbb{E}[\|X\|^2] - \|\mathbb{E}[X]\|^2$,

(ii) *For every* $x \in \mathbb{R}^d$, $\mathbb{E}[\|X-x\|^2] = \mathbb{V}[X] + \|\mathbb{E}[X]-x\|^2$. *Hence,* $\mathbb{V}[X] = \min_{x\in\mathbb{R}^d} \mathbb{E}[\|X-x\|^2]$,

(iii) $\mathbb{V}[X+Y] = \mathbb{V}[X] + \mathbb{V}[Y]$,

(iv) $\mathbb{V}[AX] \leq \mathbb{V}[A]\mathbb{V}[X] + \|\mathbb{E}[A]\|^2\mathbb{V}[X] + \|\mathbb{E}[X]\|^2\mathbb{V}[A]$.

Proof. (i)-(ii): Let $x \in \mathbb{R}^d$. Then, $\|X-x\|^2 = \|X-\mathbb{E}[X]\|^2 + \|\mathbb{E}[X]-x\|^2 + 2(X-\mathbb{E}[X])^\top(\mathbb{E}[X]-x)$. Hence, taking the expectation we get $\mathbb{E}[\|X-x\|^2] = \mathbb{V}[X] + \|\mathbb{E}[X]-x\|^2$. Therefore, $\mathbb{E}[\|X-x\|^2] \geq \mathbb{V}[X]$ and for $x = \mathbb{E}[X]$ we get $\mathbb{E}[\|X-x\|^2] = \mathbb{V}[X]$. Finally, for $x = 0$ we get (i).

(iii): Let $\bar{X} := \mathbb{E}[X]$ and $\bar{Y} := \mathbb{E}[Y]$, we have

$$\begin{aligned}
\mathbb{V}[X+Y] &= \mathbb{E}[\|X-\bar{X}+Y-\bar{Y}\|^2] \\
&= \mathbb{E}[\|X-\bar{X}\|^2] + \mathbb{E}[\|Y-\bar{Y}\|^2] + 2\mathbb{E}[X-\bar{X}]^\top \mathbb{E}[Y-\bar{Y}] \\
&= \mathbb{E}[\|X-\bar{X}\|^2] + \mathbb{E}[\|Y-\bar{Y}\|^2]
\end{aligned}$$

Recalling the definition of $\mathbb{V}[X]$ the statement follows.

(iv): Let $\bar{A} := \mathbb{E}[A]$ and $\bar{X} := \mathbb{E}[X]$. Then,

$$\begin{aligned}
\mathbb{V}[AX] &= \mathbb{E}[\|AX - \mathbb{E}[A]\mathbb{E}[X]\|^2] \\
&= \mathbb{E}[\|AX - A\bar{X} + A\bar{X} - \bar{A}\bar{X}\|^2] \\
&= \mathbb{E}[\|A(X-\bar{X}) + (A-\bar{A})\bar{X}\|^2] \\
&= \mathbb{E}[\|A(X-\bar{X})\|^2] + \mathbb{E}[\|(A-\bar{A})\bar{X}\|^2] \\
&\quad + 2\mathbb{E}[(X-\bar{X})^\top A^\top (A-\bar{A})\bar{X}] \\
&= \mathbb{E}[\|A(X-\bar{X})\|^2] + \mathbb{E}[\|(A-\bar{A})\bar{X}\|^2] \\
&\quad + 2\mathbb{E}[(X-\bar{X})^\top]\mathbb{E}[A^\top (A-\bar{A})\bar{X}] \\
&= \mathbb{E}[\|(A-\bar{A}+\bar{A})(X-\bar{X})\|^2] + \mathbb{E}[\|(A-\bar{A})\bar{X}\|^2] \\
&= \mathbb{E}[\|(A-\bar{A})(X-\bar{X})\|^2] + \mathbb{E}[\|(A-\bar{A})\bar{X}\|^2] + \mathbb{E}[\|\bar{A}(X-\bar{X})\|^2] \\
&\quad + 2\mathbb{E}[(X-\bar{X})^\top (A-\bar{A})^\top \bar{A}(X-\bar{X})] \\
&= \mathbb{E}[\|(A-\bar{A})(X-\bar{X})\|^2] + \mathbb{E}[\|(A-\bar{A})\bar{X}\|^2] + \mathbb{E}[\|\bar{A}(X-\bar{X})\|^2] \\
&\quad + 2\mathbb{E}[(X-\bar{X})^\top \mathbb{E}[A-\bar{A} \mid X]^\top \bar{A}(X-\bar{X})] \\
&\leq \mathbb{E}[\|A-\bar{A}\|^2]\mathbb{E}[\|X-\bar{X}\|^2] \\
&\quad + \mathbb{E}[\|A-\bar{A}\|^2]\|\bar{X}\|^2 + \|\bar{A}\|^2 \mathbb{E}[\|X-\bar{X})\|^2]
\end{aligned}$$

In the above equalities we have used the independence of A and X in the formulas

$\mathbb{E}[AX] = \mathbb{E}[A]\mathbb{E}[X]$, $\mathbb{E}[(X-\bar{X})^\top A^\top (A-\bar{A}\bar{X})] = \mathbb{E}[(X-\bar{X})^\top]\mathbb{E}[A^\top(A-\bar{A}\bar{X})]$, and $\mathbb{E}[(X-\bar{X})^\top (A-\bar{A})^\top \bar{A}(X-\bar{X})\,|\,X] = (X-\bar{X})^\top \mathbb{E}[(A-\bar{A})^\top\,|\,X]\bar{A}(X-\bar{X})$. □

Lemma B.4.4. *Let $f : \mathcal{Z} \subset \mathbb{R}^n \mapsto \mathbb{R}^m$ be an L-Lipschitz function, with $L > 0$, meaning that*

$$\|f(x) - f(y)\| \leq L\|x-y\| \qquad \forall x, y \in \mathcal{Z}$$

Let X be a random variable with finite variance. Then, we have that

$$\mathbb{V}[f(X)] \leq L^2 \mathbb{V}[X] \tag{B.16}$$

Proof. We have

$$\begin{aligned}
\mathbb{V}[f(X)] &= \mathbb{E}[\|f(X) - \mathbb{E}[f(X)]\|^2] \\
&= \mathbb{E}[\|f(X) - f(\mathbb{E}[X])\|^2] - \|f(\mathbb{E}[X]) - \mathbb{E}[f(X)]\|^2 \\
&\leq \mathbb{E}[\|f(X) - f(\mathbb{E}[X])\|^2] \\
&\leq L^2 \mathbb{E}[\|X - \mathbb{E}[X]\|^2] = L^2 \mathbb{V}[X].
\end{aligned}$$

□

Definition B.4.5. *(Conditional Variance). Let X be a random variable with values in $\mathbb{R}^d$ and Y be a random variable with values in a measurable space $\mathcal{Y}$. We call* conditional variance *of X given Y the quantity*

$$\mathbb{V}[X\,|\,Y] := \mathbb{E}[\|X - \mathbb{E}[X\,|\,Y]\|^2\,|\,Y].$$

Lemma B.4.6. *(Law of total variance) Let X and Y be two random variables, we can prove that*

$$\mathbb{V}[X] = \mathbb{E}[\mathbb{V}[X\,|\,Y]] + \mathbb{V}[\mathbb{E}[X\,|\,Y]] \tag{B.17}$$

Proof.

$$\begin{aligned}
\mathbb{V}[X] &= \mathbb{E}[\|X - \mathbb{E}[X]\|^2] \\
\text{(var. prop.)} \implies &= \mathbb{E}[\|X\|^2] - \|\mathbb{E}[X]\|^2 \\
\text{(tot. expect.)} \implies &= \mathbb{E}[\mathbb{E}[\|X\|^2 \,|\, Y]] - \|\mathbb{E}[\mathbb{E}[X \,|\, Y]]\|^2 \\
\text{(var. prop.)} \implies &= \mathbb{E}[\mathbb{V}[X \,|\, Y] + \|\mathbb{E}[X \,|\, Y]\|^2] - \|\mathbb{E}[\mathbb{E}[X \,|\, Y]]\|^2 \\
&= \mathbb{E}[\mathbb{V}[X \,|\, Y]] + \left(\mathbb{E}[\|\mathbb{E}[X \,|\, Y]\|^2] - \|\mathbb{E}[\mathbb{E}[X \,|\, Y]]\|^2\right)
\end{aligned}$$

recognizing that the term inside the parenthesis is the conditional variance of $\mathbb{E}[X \,|\, Y]$ gives the result. □

Lemma B.4.7. *Let ζ and η be two independent random variables with values in $\mathcal{Z}$ and $\mathcal{Y}$ respectively. Let $\psi\colon \mathcal{Y} \to \mathbb{R}^{m\times n}, \phi\colon \mathcal{Z} \to \mathbb{R}^{n\times p}$, and $\varphi\colon \mathcal{Y} \to \mathbb{R}^{p\times q}$ matrix-valued measurable functions. Then*

$$\mathbb{E}[\psi(\eta)(\phi(\zeta) - \mathbb{E}[\phi(\zeta)])\varphi(\eta)] = 0 \tag{B.18}$$

Proof. Since, for every $y \in \mathcal{Y}$, $B \mapsto \psi(y)B\varphi(y)$ is linear and ζ and η are independent, we have

$$\mathbb{E}[\psi(\eta)(\psi(\zeta) - \mathbb{E}[\psi(\zeta)])\varphi(\eta) \,|\, \eta] = \psi(\eta)\mathbb{E}\big[\phi(\zeta) - \mathbb{E}[\phi(\zeta)]\big]\varphi(\eta) = 0.$$

Taking the expectation the statement follows. □

Lemma B.4.8. *Let A be a square matrix such that $\|A\| \le q < 1$ Then, $I - A$ is invertible and*

$$\|(I-A)^{-1}\| \le \frac{1}{1-q}.$$

Proof. Since $\|A\| \le q < 1$,

$$\sum_{k=0}^{\infty} \|A\|^k \le \sum_{k=0}^{\infty} q^k = \frac{1}{1-q}.$$

Thus, the series $\sum_{k=0}^{\infty} A^k$ is convergent, say to B, and

$$(I-A)\sum_{i=0}^{k} A^i = \sum_{i=0}^{k} A^i(I-A) = \sum_{i=0}^{k} A^i - \sum_{i=0}^{k+1} A^i + I \to I, \tag{B.19}$$

so that $(I-A)B = B(I-A) = I$. Therefore, $I-A$ is invertible with inverse B and hence $\|(I-A)^{-1}\| \leq \sum_{k=0}^{\infty} \|A\|^k \leq 1/(1-q)$. □

Appendix C

Appendix for Chapter 5

C.1 Proof of Lemma 5.4.6

To prove (i), recall that $w'(\lambda) = \big(I - \partial_1\Phi(w(\lambda),\lambda)\big)^{-1}\partial_2\Phi(w(\lambda),\lambda)$, hence

$$\begin{aligned}
\|w'(\lambda)\| &= \|\big(I - \partial_1\Phi(w(\lambda),\lambda)\big)^{-1}\partial_2\Phi(w(\lambda),\lambda)\| \\
&\leq \|\big(I - \partial_1\Phi(w(\lambda),\lambda)\big)^{-1}\|\|\partial_2\Phi(w(\lambda),\lambda)\| \\
&\leq \sum_{i=0}^{\infty}\|\partial_1\Phi(w(\lambda),\lambda)\|^i\|\partial_2\Phi(w(\lambda),\lambda)\| \\
&\leq \sum_{i=0}^{\infty} q^i L_\Phi = \frac{L_\Phi}{1-q},
\end{aligned}$$

where in the second inequality we used the properties of Neumann series and in the last inequality we used Assumption 5.4.1(i) and 5.4.2(iv).

Next we prove (ii). Let $A(\lambda) = I - \partial_1\Phi(w(\lambda),\lambda)$ For every $\lambda \in \Lambda$

$$\begin{aligned}
\|A(\lambda_1) - A(\lambda_2)\| &= \|\partial_1\Phi(w(\lambda_1),\lambda_1) - \partial_1\Phi(w(\lambda_2),\lambda_2)\| \\
&\leq \|\partial_1\Phi(w(\lambda_2),\lambda_1) - \partial_1\Phi(w(\lambda_2),\lambda_2)\| \\
&\quad + \|\partial_1\Phi(w(\lambda_1),\lambda_1) - \partial_1\Phi(w(\lambda_2),\lambda_1)\| \\
&\leq \bar{\nu}_1\|\lambda_1 - \lambda_2\| + \nu_1\|w(\lambda_1) - w(\lambda_2)\| \\
&\leq \Big(\bar{\nu}_1 + \frac{\nu_1 L_\Phi}{1-q}\Big)\|\lambda_1 - \lambda_2\|,
\end{aligned}$$

where we used Assumption 5.4.1(ii) and 5.4.2(ii) in the second inequality and (i) in

the last inequality. Consequently, for every $\lambda_1, \lambda_2 \in \Lambda$

$$\begin{aligned}
\|w'(\lambda_1) - w'(\lambda_2)\| &\leq \|A(\lambda_1)^{-1}\| \|\partial_2 \Phi((w(\lambda_1), \lambda_1) - \partial_2 \Phi((w(\lambda_2), \lambda_2)\| \\
&\quad + \|\partial_2 \Phi((w(\lambda_1), \lambda_1)\| \|A(\lambda_1)^{-1}\| \|A(\lambda_1) - A(\lambda_2)\| \|A(\lambda_2)^{-1}\| \\
&\leq \|A(\lambda_1)^{-1}\| \|\partial_2 \Phi((w(\lambda_1), \lambda_2) - \partial_2 \Phi((w(\lambda_2), \lambda_2)\| \\
&\quad + \|A(\lambda_1)^{-1}\| \|\partial_2 \Phi((w(\lambda_1), \lambda_1) - \partial_2 \Phi((w(\lambda_1), \lambda_2)\| \\
&\quad + \|\partial_2 \Phi((w(\lambda_1), \lambda_1)\| \|A(\lambda_1)^{-1}\| \|A(\lambda_1) - A(\lambda_2)\| \|A(\lambda_2)^{-1}\| \\
&\leq \left[\frac{\nu_2 L_\Phi / (1-q) + \bar{\nu}_2}{1-q} + \frac{L_\Phi}{(1-q)^2} \left(\bar{\nu}_1 + \frac{\nu_1 L_\Phi}{1-q} \right) \right] \|\lambda_1 - \lambda_2\|.
\end{aligned}$$

To prove (iii) instead, let

$$\bar{\nabla} f(w, \lambda) := \nabla_2 E(w, \lambda) + \partial_2 \Phi(w, \lambda) \left[I - \partial_1 \Phi(w, \lambda)^\top \right]^{-1} \nabla_1 E(w, \lambda) \tag{C.1}$$

Note that $\nabla f(\lambda) = \bar{\nabla} f(w(\lambda), \lambda)$. We have that for every $\lambda_1, \lambda_2 \in \Lambda$

$$\|\nabla f(\lambda_1) - \nabla f(\lambda_2)\| \leq \|\nabla f(\lambda_1) - \bar{\nabla} f(w(\lambda_1), \lambda_2)\| + \|\nabla f(\lambda_2) - \bar{\nabla} f(w(\lambda_1), \lambda_2)\| \tag{C.2}$$

We bound the two terms of the RHS of (C.2) as follows.

$$\begin{aligned}
\|\nabla f(\lambda_1) - \bar{\nabla} f(w(\lambda_1), \lambda_2)\| &\leq \|\nabla_2 E(w(\lambda_1), \lambda_1) - \nabla_2 E(w(\lambda_1), \lambda_2))\| + \\
&\quad + \|w'(\lambda_1)\| \|\nabla_1 E(w(\lambda_1), \lambda_1) - \nabla_1 E(w(\lambda_1), \lambda_2))\| \\
&\leq \left(\bar{\mu}_2 + \frac{L_\Phi \bar{\mu}_1}{1-q} \right) \|\lambda_1 - \lambda_2\|,
\end{aligned}$$

$$\begin{aligned}
\|\nabla f(\lambda_2) - \bar{\nabla} f(w(\lambda_1), \lambda_2)\| &\leq \|\nabla_2 E(w(\lambda_2), \lambda_2) - \nabla_2 E(w(\lambda_1), \lambda_2))\| \\
&\quad + \|w'(\lambda_2)\| \|\nabla_1 E(w(\lambda_2), \lambda_2) - \nabla_1 E(w(\lambda_1), \lambda_2))\| \\
&\quad + \|\nabla_1 E(w(\lambda_1), \lambda_2)\| \|w'(\lambda_2) - w'(\lambda_1)\| \\
&\leq \left(L_E L_{w'} + \frac{\mu_2 L_\Phi}{1-q} + \frac{\mu_1 L_\Phi^2}{(1-q)^2} \right) \|\lambda_1 - \lambda_2\|.
\end{aligned}$$

Summing the two inequalities above we obtain the final result.

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

P. Ablin, G. Peyré, and T. Moreau. Super-efficiency of automatic differentiation for functions defined as a minimum. *arXiv preprint arXiv:2002.03722*, 2020.

N. Agarwal, B. Bullins, and E. Hazan. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18 (1):4148–4187, 2017.

H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected papers of hirotugu akaike*, pages 199–213. Springer, 1998.

Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings, 1st First International Conference on Neural Networks*, volume 2, pages 609–618. IEEE, 1987.

B. Amos. *Differentiable optimization-based modeling for machine learning*. PhD thesis, PhD thesis. Carnegie Mellon University, 2019.

B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR. org, 2017.

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

M. Arbel and J. Mairal. Amortized implicit differentiation for stochastic bilevel optimization. In *International Conference on Learning Representations*, 2021.

M. Arbel and J. Mairal. Non-convex bilevel games with critical point selection maps. *arXiv preprint arXiv:2207.04888*, 2022.

Y. Arjevani, Y. Carmon, J. C. Duchi, D. J. Foster, N. Srebro, and B. Woodworth. Lower bounds for non-convex stochastic optimization. *Mathematical Programming*, 305:1–50, 2022.

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pages 688–699, 2019.

S. Bai, V. Koltun, and J. Z. Kolter. Multiscale deep equilibrium models. *Advances in Neural Information Processing Systems*, 33:5238–5250, 2020.

S. Bai, V. Koltun, and J. Z. Kolter. Stabilizing equilibrium models by jacobian regularization. *arXiv preprint arXiv:2106.14342*, 2021.

S. Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1):133–181, 1922.

F. Bao, G. Wu, C. Li, J. Zhu, and B. Zhang. Stability and generalization of bilevel programming in hyperparameter optimization. *Advances in Neural Information Processing Systems*, 34:4529–4541, 2021.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18 (153), 2018.

Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

L. Bertinetto, J. F. Henriques, P. H. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

L. Bertinetto, J. F. Henriques, P. H. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. *ICLR*, 2019.

Q. Bertrand, Q. Klopfenstein, M. Blondel, S. Vaiter, A. Gramfort, and J. Salmon. Implicit differentiation of lasso-type models for hyperparameter optimization. In *International Conference on Machine Learning*, pages 810–821. PMLR, 2020.

Q. Bertrand, Q. Klopfenstein, M. Massias, M. Blondel, S. Vaiter, A. Gramfort, and J. Salmon. Implicit differentiation for fast hyperparameter selection in non-smooth convex learning. *Journal of Machine Learning Research*, 23(149):1–43, 2022a.

Q. Bertrand, Q. Klopfenstein, M. Massias, M. Blondel, S. Vaiter, A. Gramfort, J. Salmon, J. Chevalier, T. Nguyen, B. Thirion, et al. Implicit differentiation for fast hyperparameter selection in non-smooth convex learning. *Journal of Machine Learning Research*, 23(149):1–43, 2022b.

B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In *Asian conference on machine learning*, pages 97–112. PMLR, 2011.

B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1467–1474, 2012.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

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

T. Chen, Y. Sun, Q. Xiao, and W. Yin. A single-timescale method for stochastic bilevel optimization. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *PMLR*, pages 2466–2488, 2022.

A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

A. E. Cinà, S. Vascon, A. Demontis, B. Biggio, F. Roli, and M. Pelillo. The hammer and the nut: Is bilevel optimization really needed to poison linear classifiers? In

2021 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2021.

A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *arXiv preprint arXiv:2205.01992*, 2022.

N. Couellan and W. Wang. On the convergence of stochastic bi-level gradient methods. *Optimization*, 2016.

R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact newton methods. *SIAM Journal on Numerical analysis*, 19(2):400–408, 1982.

S. Dempe and A. Zemkoho. *Bilevel Optimization*. Springer, 2020.

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

G. Denevi, D. Stamos, C. Ciliberto, and M. Pontil. Online-within-online meta-learning. In *Advances in Neural Information Processing Systems*, pages 13110–13120, 2019b.

D. Drusvyatskiy and A. S. Lewis. Error bounds, quadratic growth, and linear convergence of proximal methods. *Mathematics of Operations Research*, 43(3): 919–948, 2018.

S. Du, J. Lee, H. Li, L. Wang, and X. Zhai. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*, pages 1675–1685. PMLR, 2019.

P. Dvurechensky. Gradient method with inexact oracle for composite non-convex optimization. *arXiv preprint arXiv:1703.09180*, 2017.

H. Edwards and A. Storkey. Towards a neural statistician. *arXiv preprint arXiv:1606.02185*, 2016.

J. Feng, Q.-Z. Cai, and Z.-H. Zhou. Learning to confuse: generating training time adversarial data with auto-encoder. *Advances in Neural Information Processing Systems*, 32, 2019.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

R. Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

L. Franceschi. *A Unified Framework for Gradient-based Hyperparameter Optimization and Meta-learning*. PhD thesis, UCL (University College London), 2021.

L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1165–1173. JMLR. org, 2017.

L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1563–1572, 2018.

L. Franceschi, M. Niepert, M. Pontil, and X. He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pages 1972–1982. PMLR, 2019.

E. Frantar, E. Kurtic, and D. Alistarh. M-fac: Efficient matrix-free approximations of second-order information. *Advances in Neural Information Processing Systems*, 34:14873–14886, 2021.

P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, pages 797–842. PMLR, 2015.

S. Ghadimi and G. Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

S. Ghadimi and M. Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

E. Gorbunov, F. Hanzely, and P. Richtárik. A unified theory of sgd: Variance reduction, sampling, quantization and coordinate descent. In *International Conference on Artificial Intelligence and Statistics*, pages 680–690. PMLR, 2020.

R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo. On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, pages 3748–3758. PMLR, 2020.

R. Grazzi, V. Flunkert, D. Salinas, T. Januschowski, M. Seeger, and C. Archambeau. Meta-forecasting by combining global deep representations with local adaptation. *arXiv preprint arXiv:2111.03418*, 2021a.

R. Grazzi, M. Pontil, and S. Salzo. Convergence properties of stochastic hypergradients. In *International Conference on Artificial Intelligence and Statistics*, pages 3826–3834. PMLR, 2021b.

R. Grazzi, A. Akhavan, J. I. T. Falk, L. Cella, and M. Pontil. Group meritocratic fairness in linear contextual bandits. *NeurIPS*, 2022.

R. Grazzi, M. Pontil, and S. Salzo. Bilevel optimization with a lower-level contraction: Optimal sample complexity without warm-start. *Journal of Machine Learning Research*, 24(167):1–37, 2023.

A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui. Implicit graph neural networks. *Advances in Neural Information Processing Systems*, 33:11984–11995, 2020.

Z. Guo and T. Yang. Randomized stochastic variance-reduced methods for stochastic bilevel optimization. *arXiv preprint arXiv:2105.02266*, 2021.

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.

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.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International conference on artificial neural networks*, pages 87–94. Springer, 2001.

N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. In *NeurIPS 2022 First Table Representation Workshop*, 2022.

M. Hong, H.-T. Wai, Z. Wang, and Z. Yang. A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic. *arXiv preprint arXiv:2007.05170*, 2020.

F. Huang and H. Huang. BiAdam: Fast Adaptive Bilevel Optimization Methods. *arXiv e-prints*, art. arXiv:2106.11396, June 2021.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pages 240–248. PMLR, 2016.

K. Ji, J. Yang, and Y. Liang. Bilevel optimization: Convergence analysis and enhanced design. In *International Conference on Machine Learning*, pages 4882–4892. PMLR, 2021.

K. Ji, M. Liu, Y. Liang, and L. Ying. Will bilevel optimizers benefit from loops. *arXiv preprint arXiv:2205.14224*, 2022.

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. *Advances in Neural Information Processing Systems*, 34:30271–30283, 2021.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

R. S. S. Kumar, M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia. Adversarial machine learning-industry perspectives. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 69–75. IEEE, 2020.

B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

S. Lang. *Fundamentals of differential geometry*, volume 191. Springer Science & Business Media, 2012.

T. Lattimore and C. Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht. Gradient descent only converges to minimizers. In *Conference on learning theory*, pages 1246–1257. PMLR, 2016.

K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10657–10665, 2019.

J. Li, B. Gu, and H. Huang. A fully single loop algorithm for bilevel optimization without hessian inverse. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7426–7434, 2022.

R. Liao, Y. Xiong, E. Fetaya, L. Zhang, K. Yoon, X. Pitkow, R. Urtasun, and R. Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pages 3088–3097, 2018.

M. Little, P. McSharry, E. Hunter, J. Spielman, and L. Ramig. Suitability of dysphonia measurements for telemonitoring of parkinson's disease. *Nature Precedings*, pages 1–1, 2008.

H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

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 *International Conference on Machine Learning*, pages 6305–6315. PMLR, 2020.

R. Liu, P. Mu, X. Yuan, S. Zeng, and J. Zhang. A general descent aggregation framework for gradient-based bi-level optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020.

I. Loshchilov and F. Hutter. Cma-es for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.

D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

J. Mairal, F. Bach, and J. Ponce. Task-driven dictionary learning. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):791–804, 2011.

J. Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimization. In *ICML*, 2011.

A. Mehra and J. Hamm. Penalty method for inversion-free deep bilevel optimization. *arXiv preprint arXiv:1911.03432*, 2019.

J. Miller and M. Hardt. Stable recurrent models. *ICLR*, 2019.

N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017.

Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

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

L. M. Nguyen, P. H. Nguyen, P. Richtárik, K. Scheinberg, M. Takáč, and M. van Dijk. New convergence aspects of stochastic gradient algorithms. *Journal of Machine Learning Research*, 20(176):1–49, 2019.

M. Niepert, P. Minervini, and L. Franceschi. Implicit mle: backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34:14567–14579, 2021.

J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

P. Ochs, R. Ranftl, T. Brox, and T. Pock. Bilevel optimization with nonsmooth lower level problems. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 654–665. Springer, 2015.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019a.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019b. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-li pdf`.

B. A. Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6 (1):147–160, 1994.

F. Pedregosa. Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning*, pages 737–746, 2016.

V. Perrone, H. Shen, M. W. Seeger, C. Archambeau, and R. Jenatton. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. *Advances in Neural Information Processing Systems*, 32, 2019.

F. J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229, 1987.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

B. T. Polyak. *Introduction to Optimization*. Optimization Software Inc. Publication Division, New York, NY, USA, 1987.

A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, pages 113–124, 2019.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. *ICLR*, 2017.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

E. Real, C. Liang, D. So, and Q. Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, pages 8007–8019. PMLR, 2020.

S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

H. Robbins and D. Siegmund. A convergence theorem for non negative almost supermartingales and some applications. *Optimizing Methods in Statistics*, pages 233–257, 1971.

E. K. Ryu and S. Boyd. Primer on monotone operator methods. *Appl. Comput. Math*, 15(1):3–43, 2016.

A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

T. Schick and H. Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, 2021.

M. Schmidt, N. Roux, and F. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

H. Sedghi, V. Gupta, and P. M. Long. The singular values of convolutional layers. *ICLR*, 2019.

A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots. Truncated back-propagation for bilevel optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1723–1732, 2019.

A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *Advances in neural information processing systems*, 31, 2018.

K. A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):1–25, 2009.

J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

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

C. M. Stein. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pages 1135–1151, 1981.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)*, 36(2):111–133, 1974.

F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1199–1208, 2018.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola. Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision*, pages 266–282. Springer, 2020.

E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

P. Vicol, J. P. Lorraine, F. Pedregosa, D. Duvenaud, and R. B. Grosse. On implicit bias in overparameterized bilevel optimization. In *International Conference on Machine Learning*, pages 22234–22259. PMLR, 2022.

R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

H. Von Stackelberg. *Market structure and equilibrium*. Springer Science & Business Media, 2010.

R. Wang, M. Pontil, and C. Ciliberto. The role of global labels in few-shot classification and how to infer them. *Advances in Neural Information Processing Systems*, 34:27160–27170, 2021.

E. Winston and J. Z. Kolter. Monotone operator equilibrium networks. *Advances in neural information processing systems*, 33:10718–10728, 2020.

H. Xiao, H. Xiao, and C. Eckert. Adversarial label flips attack on support vector machines. In *ECAI 2012*, pages 870–875. IOS Press, 2012.

H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli. Support vector machines under adversarial label contamination. *Neurocomputing*, 160:53–62, 2015.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

C. Yang, Q. Wu, H. Li, and Y. Chen. Generative poisoning attack method against neural networks. *arXiv preprint arXiv:1703.01340*, 2017.

J. Yang, K. Ji, and Y. Liang. Provably faster algorithms for bilevel optimization. *Advances in Neural Information Processing Systems*, 34:13670–13682, 2021.

Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney. Adahessian: An adaptive second order optimizer for machine learning. In *proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10665–10673, 2021.

P. Zhou, X. Yuan, H. Xu, S. Yan, and J. Feng. Efficient meta learning via minibatch proximal update. In *Advances in Neural Information Processing Systems*, pages 1534–1544, 2019.