Titre: Title:	Smooth and Nonsmooth Large Scale Optimization with Inexact Evaluations
Auteur: Author:	Nathan Allaire
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Allaire, N. (2025). Smooth and Nonsmooth Large Scale Optimization with Inexact Evaluations [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/71183/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Smooth and Nonsmooth Large Scale Optimization with Inexact Evaluations

NATHAN ALLAIRE
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques

Décembre 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Smooth and Nonsmooth Large Scale Optimization with Inexact Evaluations

présentée par **Nathan ALLAIRE**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Jonathan JALBERT, président
Sébastien LE DIGABEL, membre et directeur de recherche
Dominique ORBAN, membre et codirecteur de recherche
Vahid PARTOVI NIA, membre et codirecteur de recherche
Quentin CAPPART, membre
Alberto DE MARCHI, membre externe

DEDICATION

À la mémoire de mon grand-père
"Recherche la beauté, elle est en toute chose"

ACKNOWLEDGEMENTS

My first words of thanks go to my research supervisor, Sébastien Le Digabel. Without his trust and generosity, I would never have received a reply to that poorly written, overly polite email sent in the hope of finding an internship to end my engineering studies. Thank you, Sébastien, for your unwavering support from the very beginning to the very end of this adventure. You have been a captain both human, honest, and fair.

I would also like to express my deepest gratitude to my co-supervisors, Dominique Orban and Vahid Partovi Nia. Thank you, Dominique, for your scientific rigor and your kind but firm expectations, which taught me a great deal about myself through our collaboration. You are a true example of generosity at work and intellectual honesty. And thank you, Vahid, for offering me the opportunity to carry out that research internship at the Noah's Ark Lab, which opened up a new world of scientific possibilities and deeply shaped the course of my PhD. I will miss our philosophical discussions over coffee.

Thanks to the GERAD lab for its warm and caring welcome. Thanks to my colleagues at GERAD, especially Sacha and Théo, for our mischievous collaboration and all our wonderfully absurd stories. Thank you to Alexis Montoison and Dominique Monnet for their constant kindness and generosity throughout these four years. I also wish to thank the GERAD and the Fonds de recherche du Québec – Nature et technologies (FRQNT) for their financial support during this thesis.

I sincerely thank Professors Jonathan Jalbert, Quentin Cappart, Tarek Ould Bachir, and Alberto De Marchi for agreeing to serve on my jury.

Thank you to all my students from the MTH1008 course, as well as to my former teachers. You opened within me the door to the transmission of knowledge, and I hope it never closes.

To my friends Martin and Mohamed: I'd like to write you a thank-you for every laugh and ridiculous story, but I'd prefer to keep this thesis under a thousand pages.

Thank you to my family—to my parents for their unwavering love and support, to Mupsy for taking such tender care of me during my preparatory classes, and to my sisters, whose paths, though very different from mine, inspire my admiration.

And finally, thank you, Loan, for everything. Thank you for helping me become the man I am growing into. My greatest pride is to see how beautifully and naturally we evolve, hand in hand.

我爱你.

RÉSUMÉ

Dans cette thèse, nous étudions deux grandes classes de problèmes d'optimisation rencontrés en apprentissage automatique et en science des données, en nous concentrant sur les situations où certaines informations essentielles du problème ne sont disponibles qu'approximativement. La première classe concerne l'optimisation à grande échelle, et plus particulièrement l'entraînement de modèles d'apprentissage profond, où la fonction objectif s'exprime comme une espérance sur un vaste ensemble de données. Notre première contribution (Chapitre 4) est d'avoir démontré la faisabilité d'entraîner des modèles de langage au moyen de méthodes d'ordre zéro, posant ainsi les bases de l'optimisation à grande échelle sans rétropropagation du gradient, et permettant une réduction significative de la consommation mémoire. L'article correspondant a été publié dans les *Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods* et est disponible dans [4]. Dans la continuité de ces travaux, notre seconde contribution est *KronZO* (Chapitre 5), une méthode d'ordre zéro dans laquelle nous apportons des améliorations méthodologiques significatives. L'article correspondant a été soumis dans la revue *Springer Nature Computer Science* (SNCS) et est disponible dans [6]. Nous en établissons la convergence sous des hypothèses classiques et montrons, à travers des expériences numériques, que KronZO atteint des performances de pointe parmi les méthodes d'ordre zéro, tout en réduisant la consommation de mémoire.

La seconde classe de problèmes étudiés relève de l'optimisation non lisse, où la fonction objectif se compose d'un terme lisse et d'un terme de régularisation non-lisse. Une approche courante pour résoudre ce type de problème est la *méthode du gradient proximal*, qui repose sur le calcul d'*opérateurs proximaux*. Nous considérons ici le cas où la fonction objectif, son gradient et l'opérateur proximal sont évalués de manière inexacte. Nous introduisons *iR2N* (Chapitre 6), une méthode quasi-Newton modifiée pour les problèmes régularisés, spécifiquement conçue pour traiter ces évaluations inexactes. L'article correspondant a été soumis dans le *SIAM Journal on Scientific Computing* (SISC) et est disponible dans [5]. Nous établissons la convergence de iR2N sous un ensemble d'hypothèses adaptées et développons un cadre général pour construire des opérateurs proximaux inexacts garantissant la convergence théorique de iR2N. L'efficacité de la méthode est évaluée au moyen d'expériences numériques sur divers problèmes et mettent en évidence les gains significatifs liés à l'utilisation d'informations inexactes. L'implémentation de iR2N est disponible publiquement [14], et les opérateurs proximaux inexacts utilisés dans les expériences sont détaillés dans [2, 3].

ABSTRACT

In this thesis, we study two major classes of optimization problems that arise in machine learning and data science, focusing on situations where key problem information is only available approximately. The first class concerns large-scale optimization, and more specifically the training of deep learning models, where the objective function is expressed as an expectation over a large dataset. Our first contribution (Chapter 4) demonstrates the feasibility of training language models using zeroth-order methods, thereby laying the foundations of large-scale optimization without backpropagation and enabling a significant reduction in memory consumption. The corresponding paper was published in the *Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods* and is available in [4]. Building upon this work, our second contribution, *KronZO* (Chapter 5), introduces significant methodological improvements. The corresponding paper has been submitted to *Springer Nature Computer Science* (SNCS) and is available in [6]. We establish its convergence under standard assumptions and show through numerical experiments that KronZO achieves state-of-the-art performance among zeroth-order methods while reducing memory consumption.

The second class of problems studied falls within nonsmooth optimization, where the objective function consists of a smooth term and a nonsmooth regularization term. A common approach to such problems is the *proximal gradient method*, which relies on the computation of *proximal operators*. We consider here the case where the objective function, its gradient, and the proximal operator are evaluated inexactly. We introduce *iR2N* (Chapter 6), a quasi-Newton method for regularized problems specifically designed to handle such inexact evaluations. The corresponding manuscript was submitted to the *SIAM Journal on Scientific Computing* (SISC) and will be available in [5]. We establish the convergence of iR2N under suitable assumptions and develop a general framework for constructing inexact proximal operators that preserve its theoretical convergence. The effectiveness of the method is assessed through numerical experiments on various problems, which highlight the substantial efficiency gains obtained by leveraging inexact information. The implementation of iR2N is publicly available at [14], and the inexact proximal operators we use are detailed in [2, 3].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Inexact evaluations in optimization refer to situations where the information guiding the decision process, such as objective function values, gradients, or other quantities, is not computed with absolute precision, whether intentionally or not. Such approximations may arise from numerical errors, computational constraints, or reliance on estimations when exact evaluations are prohibitively expensive. For instance, in the context of *deep learning*, computing the exact gradient over the entire dataset can be unreasonably costly. Similarly, in complex simulations or stochastic optimization problems, obtaining perfectly accurate evaluations may be difficult, or even impossible, due to the stochastic nature or inherent complexity of the model. A final example is the use of low-precision floating-point arithmetic, which lightens computation at the cost of larger numerical errors. In such cases, the available information is then said to be *inexact*.

This research aims to provide a theoretical analysis of numerical optimization problems of the form

$$\min_{x\in\Omega\subseteq\mathbb{R}^n} F(x), \tag{1.1}$$

in which some components are evaluated inexactly. Our objectives are twofold: (i) to design and analyze methods whose convergence remains guaranteed despite such inaccuracies, and (ii) to implement them to assess their robustness and practical performance. With this in mind, we focus on two families of optimization problems, depending on the properties of the objective function.

The first family includes *smooth* problems (that is, the objective function is differentiable and its derivatives are continuous on $\mathbb{R}^n$) of high dimension, for which the exact evaluation of the function and its gradient is often too costly to be practical. The second concerns *nonsmooth* problems, where the objective function is expressed as the sum of a differentiable term and a nondifferentiable term, which requires specific tools such as the *proximal operator*.

In detail, the first family addresses large-scale smooth optimization problems, whose objective function can be written as

$$F(x) := \mathbb{E}_\xi[f(x;\xi)]\,, \tag{1.2}$$

where the expectation is generally not computable in closed form. In practice, it is approximated using a subsample of indices $\mathcal{I}$, of cardinality $m = \mathrm{card}(\mathcal{I})$, as follows:

$$F(x) \approx \tfrac{1}{m}\sum_{i=1}^{m} f_i(x) \quad \text{and} \quad \nabla F(x) \approx \tfrac{1}{m}\sum_{i=1}^{m} \nabla f_i(x), \tag{1.3}$$

where each f_i represents the contribution of the i-th sample, typically associated with a training datum in a large dataset. Thus, solving (1.2) relies on inexact approximations of F and its gradient ∇F. A visual representation of (1.3) is provided in Figure 1.1.

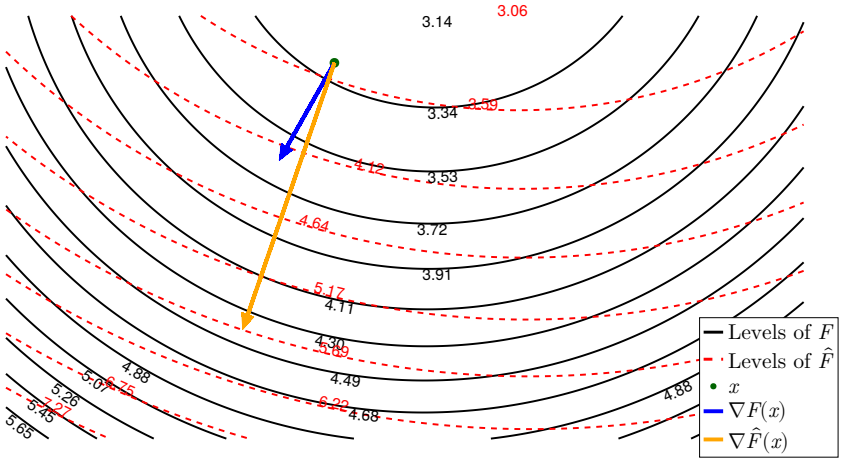

Figure 1.1 Comparison between a quadratic function F of two variables obtained as the sum of many functions f_i, and the same function sampled over a subset of indices $\mathcal{I}$.

Figure 1.1 illustrates the trade-off between computational efficiency and accuracy. Clearly, for the chosen value of m, F and its gradient differ significantly from their sampled versions. Increasing m in (1.3) would yield a more accurate approximation of the objective (and its gradient) but at a higher computational cost.

The second family concerns nonsmooth optimization problems, whose objective function takes the form

$$F(x) := f(x) + h(x), \tag{1.4}$$

where f is differentiable and h is nondifferentiable. In this context, we focus on cases where the objective function, its gradient, and the *proximal operator* are all evaluated inexactly.

1.1 Research Context

As we strive to numerically solve ever larger problems (particularly in deep learning [56], or more generally in tackling increasingly complex challenges) the number of required operations also grows. However, the energy consumption associated with each individual operation remains roughly constant; therefore, in the absence of efforts toward efficiency, solving a large-scale problem inevitably requires more energy than solving a smaller one. Similarly, for a given computational power, an increase in the number of operations directly translates into

longer computation times. Moreover, the risk of saturating the available memory by storing a large volume of data simultaneously also increases with problem size.

There are several ways to reduce the amount of energy required to solve such problems. A natural approach is to decrease the number of elementary operations, which directly reduces energy consumption. This can, for instance, take the form of an approximate evaluation of the objective function or its gradient: by using only a subset of indices $\mathcal{I}$ in (1.2), one limits the computational cost. This type of economy may also aim to reduce memory usage, which is particularly critical in deep learning. However, this reduction in computational load comes at the cost of partial information loss, which may affect result quality. Likewise, certain components of the problem may be computed with reduced precision or through iterative procedures stopped before full convergence. These approximations save resources but inevitably introduce a degree of inexactness into the information exploited by the algorithm.

To control memory impact when solving large-scale problems such as those in (1.2), various approaches have already proven effective, including *quantization* [46, 50], *pruning* [61], and *transfer learning* [92]. These strategies aim to reduce computational complexity and memory usage while preserving performance. Although effective, these methods mainly operate on the model architecture or representation. In this work, we adopt a complementary algorithmic perspective, focusing instead on the precision of the information being processed.

1.2 Research Objectives

In this thesis, we aim to solve problems (1.2) and (1.4) while reducing computational cost by exploiting inexact information. For problem (1.2), we also seek to limit the memory footprint of the methods, making them applicable to large-scale models. We develop and analyze methods whose robustness to inexactness is theoretically established under suitable convergence assumptions.

CHAPTER 2 LITERATURE REVIEW

In this chapter, we briefly recall a few essential notions of numerical optimization before diving into more complex topics. We assume basic familiarity with optimization and omit elementary details.

2.1 Fundamentals of Unconstrained Optimization

We consider the general unconstrained problem

$$\min_{x \in \mathbb{R}^n} f(x), \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function. This function f does not necessarily have a direct link with the objective functions (1.2) and (1.4): here, we simply consider a general context in which the objective is continuously differentiable in order to develop the content of this section. A classical approach to solve (2.1) consists in generating a sequence $\{x_k\}$ according to the rule

$$x_{k+1} = x_k + t_k d_k,$$

where d_k is a descent direction and $t_k > 0$ a step size.

A vector $d_k \in \mathbb{R}^n$ is a *descent direction* if

$$\nabla f(x_k)^T d_k < 0.$$

Indeed, a Taylor expansion of f around x_k gives

$$f(x_k + t d_k) \approx f(x_k) + t \nabla f(x_k)^T d_k,$$

so that for sufficiently small $t > 0$, the value of f decreases along d_k. Optimization methods based on descent directions are theoretically powerful. It is therefore important to compute the value of t_k carefully.

2.1.1 Line Search and Decrease Conditions

The choice of the step t_k is crucial: a step that is too large compromises stability, while one that is too small slows convergence. A common strategy consists in determining t_k through a line search, aiming to ensure a sufficient decrease of f.

Definition 1 (Armijo and Wolfe Conditions)**.** *Let $x_k \in \mathbb{R}^n$ and a descent direction d_k. A step $t_k > 0$ satisfies:*

- *the Armijo condition if there exists $\alpha \in (0,1)$ such that*

$$f(x_k + t_k d_k) \leq f(x_k) + \alpha t_k \nabla f(x_k)^T d_k,$$

- *the Wolfe conditions [99] if there exist $\alpha \in (0,1)$, $\beta \in (\alpha, 1)$ such that*

$$f(x_k + t_k d_k) \leq f(x_k) + \alpha t_k \nabla f(x_k)^T d_k \quad \textit{(Armijo condition),}$$

and

$$\nabla f(x_k + t_k d_k)^T d_k \geq \beta \nabla f(x_k)^T d_k \quad \textit{(curvature condition).}$$

The Armijo condition alone guarantees sufficient decrease but may hold for arbitrarily small t_k. The curvature condition prevents such choices by requiring significant progress along the descent direction. The Wolfe conditions are therefore central in the convergence analysis of descent methods.

The Wolfe conditions can be written, by defining $\varphi(t) = f(x_k + t d_k)$, as follows

$$\begin{cases} \varphi(t_k) \leq \varphi(0) + \alpha t_k \varphi'(0), \\ \varphi'(t_k) \geq \beta \varphi'(0), \end{cases}$$

We illustrate an example of the Wolfe conditions in Figure 2.1.

Figure 2.1 illustrates the search for a step t_k satisfying the Wolfe conditions for the function φ, for a given triplet (f, x_k, d_k). In the upper plot, the blue curve represents $\varphi(t)$, while the red line corresponds to the Armijo condition and the light red area indicates the steps satisfying the Armijo condition. The lower plot shows the derivative φ', with the dotted orange line corresponding to the bound $\beta\varphi'(0)$, and the light green area to the steps satisfying the curvature condition. The intersection of these two regions defines the purple zone in the upper plot, corresponding to the steps that simultaneously satisfy the Armijo and curvature

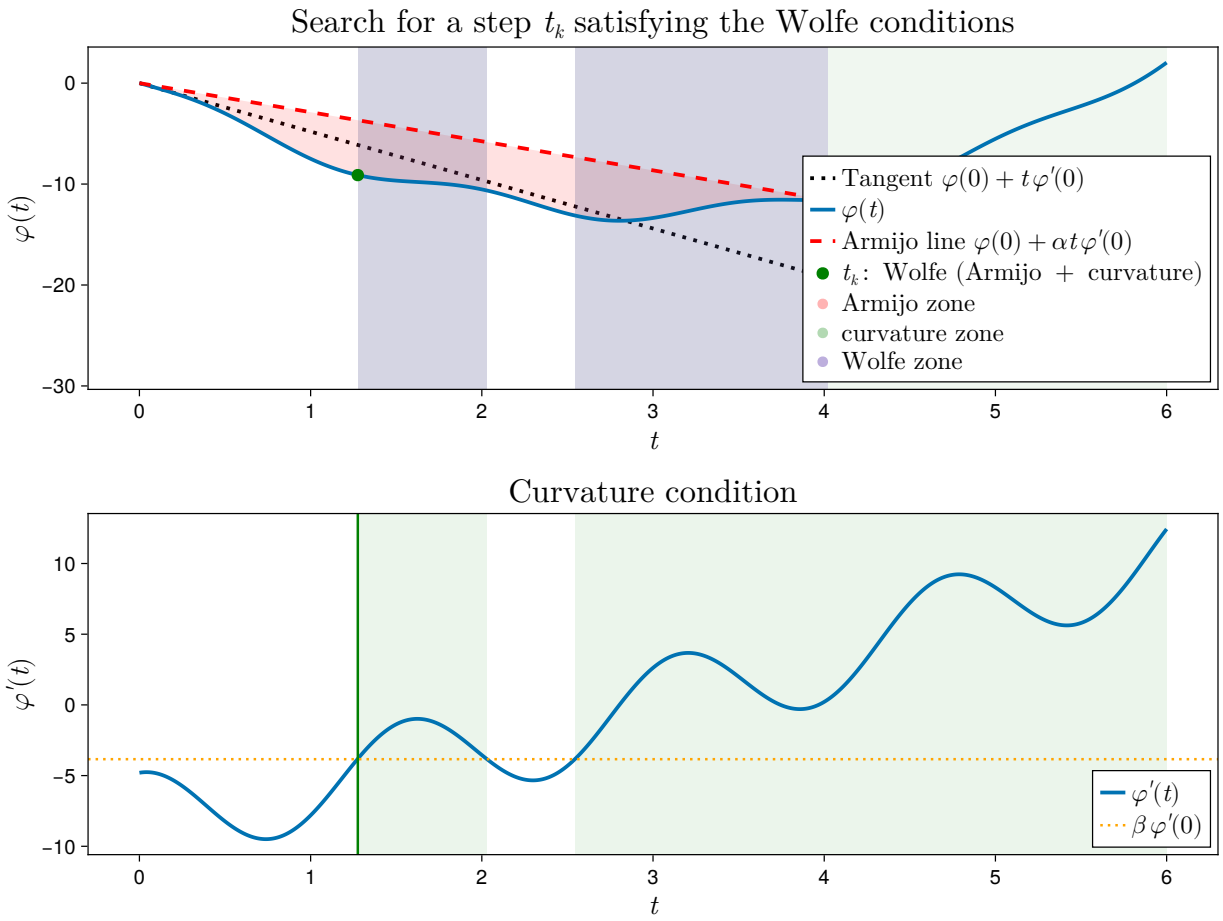

Figure 2.1 Wolfe conditions for a function $\varphi(t)$. The smallest step t_k satisfying the Wolfe conditions is shown in green.

conditions—that is, the *Wolfe conditions*. The smallest admissible step t_k balances stability (avoiding a step that is too large) and convergence speed (avoiding a step that is too small).

2.1.2 Backtracking Method

When the curvature condition is not explicitly verified, a common alternative is the progressive step reduction (*backtracking*) line search, which iteratively reduces the step size $t \leftarrow \rho t$ with $\rho \in (0, 1)$ until the Armijo condition is satisfied.

Lemma 1 (75, Alg. 3.1)**.** *If f is of class $\mathcal{C}^1$ and ∇f is L-Lipschitz continuous, then for any descent direction d_k and any standard choice of parameters (t_0, ρ, α), there exists a finite integer $m \geq 0$ such that $t_k = \rho^m t_0$ satisfies the Armijo condition.*

2.1.3 Global Convergence and Zoutendijk's Theorem

The following theorem links the properties of descent directions and steps satisfying the Wolfe conditions to the asymptotic behavior of the gradient.

Theorem 1 (75, Th. 3.2)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable, bounded below, and with L-Lipschitz continuous gradient. Consider a sequence $\{x_k\}$ generated by $x_{k+1} = x_k + t_k d_k$, where d_k is a descent direction and t_k satisfies the Wolfe conditions. Then*

$$\sum_{k=0}^{\infty} (\cos\theta_k)^2 \|\nabla f(x_k)\|^2 < +\infty,$$

where

$$\cos\theta_k := -\frac{\nabla f(x_k)^T d_k}{\|\nabla f(x_k)\| \|d_k\|}.$$

The finiteness of this sum implies that $(\cos\theta_k)^2\|\nabla f(x_k)\|^2 \to 0$. In other words, the gradients become small or the directions cease to be strictly descending—two situations consistent with convergence toward a stationary point.

Theorem 2 (75, § 3.2)**.** *Under the assumptions of Theorem 1, if the directions d_k satisfy $\cos\theta_k \geq c > 0$ for all k, then*

$$\lim_{k\to\infty} \|\nabla f(x_k)\| = 0.$$

2.1.4 Example: Steepest Descent Method

A simple method using descent directions is the gradient method, or steepest descent method. The steepest descent method chooses $d_k = -\nabla f(x_k)$: this direction is a descent direction because, if $\nabla f(x_k) \neq 0$ (i.e., if x_k is not stationary), then $d_k^T \nabla f(x_k) = -\|\nabla f(x_k)\|^2 < 0$.

Algorithm 1 Steepest Descent Method

1: Choose an initial point $x_0 \in \mathbb{R}^n$, an initial step size $t_0 > 0$, and a stopping tolerance $\varepsilon > 0$.
2: **While** $\|\nabla f(x_k)\| > \varepsilon$ **do**
3: Set $d_k = -\nabla f(x_k)$.
4: Find t_k satisfying the Wolfe conditions.
5: Update $x_{k+1} = x_k + t_k d_k$.
6: **end While**
7: **return** x_k

If f is μ-strongly convex and has L-Lipschitz continuous gradient, the convergence of this method is linear, with a factor depending on the conditioning $\kappa = L/\mu$ of the Hessian at the

optimal point:

$$\|x_{k+1} - x^\star\| \leq (1 - \mu/L)\|x_k - x^\star\|,$$

where $x^\star$ is the point minimizing f.

This result highlights the relative slowness of gradient descent for ill-conditioned problems, motivating the use of more sophisticated approaches such as Newton or quasi-Newton methods.

2.2 Newton and Quasi-Newton Methods

Newton's method exploits the local structure of the objective function to accelerate convergence. The main idea is to use information from the Hessian matrix (the matrix of second derivatives) to adjust the descent direction. At each iteration, the direction d_k is obtained by solving the linear system

$$\nabla^2 f(x_k) d_k = -\nabla f(x_k),$$

where $\nabla^2 f(x_k)$ denotes the Hessian matrix of f at x_k. The solution is then updated according to the classical rule

$$x_{k+1} = x_k + t_k d_k.$$

When the Hessian matrix $\nabla^2 f(x_k)$ is positive definite, it is symmetric and invertible. Thus

$$y^T (\nabla^2 f(x_k))^{-1} y > 0 \quad \text{for all } y \neq 0.$$

Then, since $d_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$, we have

$$\nabla f(x_k)^T d_k = -\nabla f(x_k)^T (\nabla^2 f(x_k))^{-1} \nabla f(x_k) < 0,$$

so d_k is indeed a descent direction.

Algorithm 2 Newton's Method

1: Choose an initial point $x_0 \in \mathbb{R}^n$, an initial step size $t_0 > 0$, and a stopping tolerance $\varepsilon > 0$.
2: **While** $\|\nabla f(x_k)\| > \varepsilon$ **do**
3: Solve $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$ to obtain d_k.
4: Find t_k satisfying the Wolfe conditions.
5: Update $x_{k+1} = x_k + t_k d_k$.
6: **end While**
7: **return** x_k

Under certain conditions, Newton's method achieves quadratic convergence.

Theorem 3 (75, Section 9.5)**.** *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable and that $x^\star$ is a stationary point such that $\nabla f(x^\star) = 0$ and $\nabla^2 f(x^\star)$ is positive definite. Then there exists a neighborhood of $x^\star$ such that, for any x_0 in this neighborhood, the sequence $\{x_k\}$ generated by Newton's method,*

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k),$$

converges to $x^\star$ quadratically, that is,

$$\|x_{k+1} - x^\star\| \leq C \|x_k - x^\star\|^2, \quad \text{for some constant } C > 0.$$

Figure 2.2 illustrates an example of minimizing a quadratic function using Algorithms 1 and 2. Each step t_k satisfies the Wolfe conditions.

In the upper plot of Figure 2.2, the faster convergence rate of Newton's method is clear. The lower plot confirms the quadratic convergence rate of Newton's method.

The assumption that $\nabla^2 f(x^\star)$ is positive definite ensures that $x^\star$ is a strict local minimum and that the Newton direction remains a descent direction in a neighborhood of $x^\star$. A major drawback of Newton's method is the cost of forming and storing the Hessian. Moreover, if the matrix is not positive definite, the resulting direction may fail to be a descent direction.

These limitations are addressed by *quasi-Newton methods* variants. These methods construct an approximation of the Hessian $B_k \approx \nabla^2 f(x_k)$ from successive gradients, avoiding its explicit computation. They also ensure that the approximated matrix remains positive definite, guaranteeing that the computed direction is always a descent direction.

Among the best-known are the BFGS method and its limited-memory version l-BFGS [67, 74]. The BFGS update is written as

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k},$$

with

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k), \quad s_k = x_{k+1} - x_k.$$

The above update guarantees that B_{k+1} remains positive definite as long as $y_k^T s_k > 0$. This condition is generally satisfied if the step t_k meets the Wolfe conditions [75, Section 6.1]. The

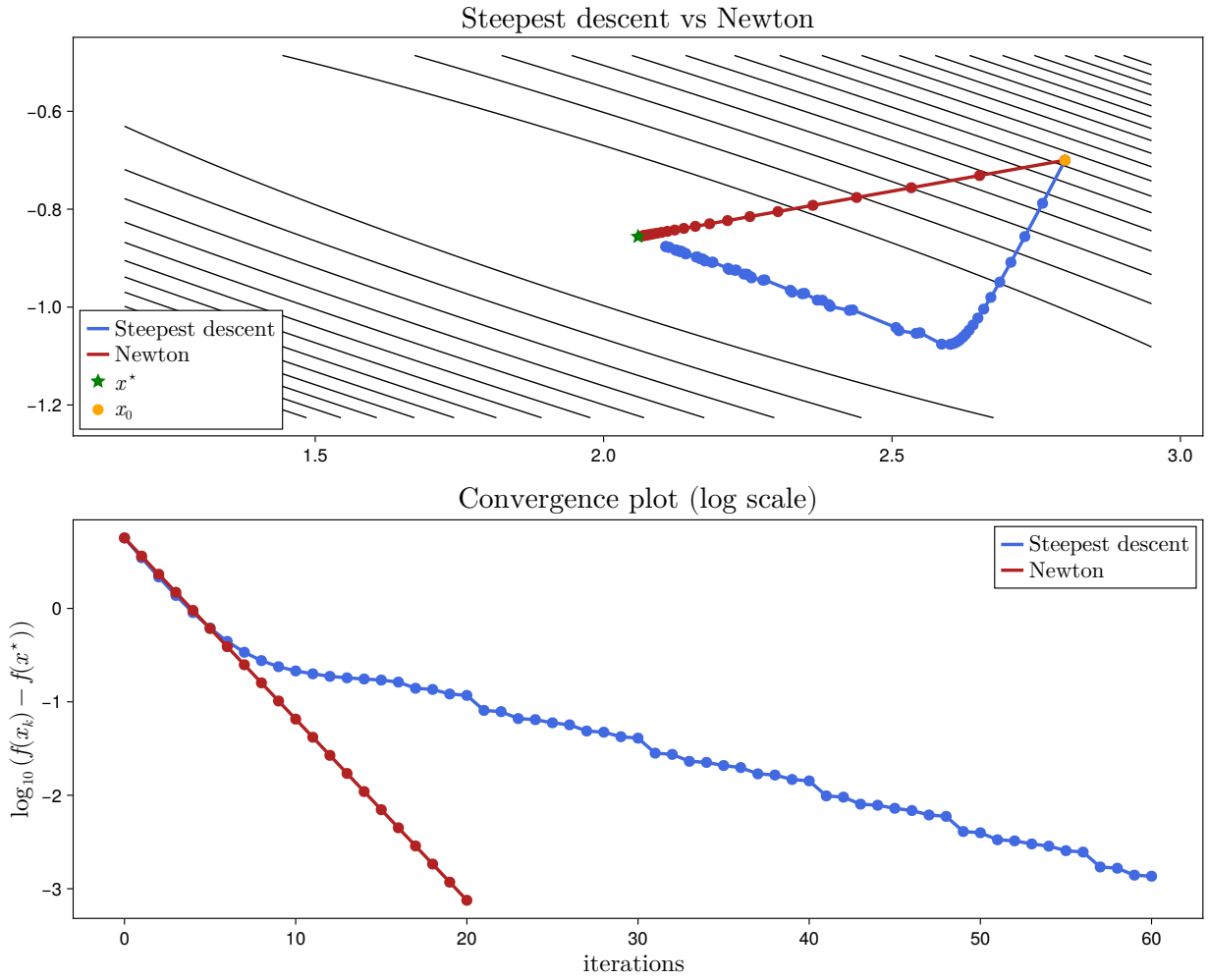

Figure 2.2 Algorithm 1 and Algorithm 2 applied to a quadratic function of two variables. Algorithm 1 is given a budget of 80 iterations, and 20 for Algorithm 2. Each point represents one iteration of the method. The starting point is denoted x_0 for both methods. The optimal point is $x^\star$.

descent direction is then given by

$$d_k = -B_k^{-1} \nabla f(x_k),$$

and ensures a decrease in the objective function. Quasi-Newton methods thus retain super-linear convergence while avoiding the cost of computing and storing an explicit Hessian [75, Theorem 6.6].

In the classical BFGS method, the update of B_{k+1} requires the full matrix B_k, which itself depends on B_{k-1}, and so on. This recursive dependence makes explicit storage and updates

Algorithm 3 BFGS Method

Choose $x_0 \in \mathbb{R}^n$, an initial positive definite matrix B_0, an initial step size $t_0 > 0$, and a stopping tolerance $\varepsilon > 0$.
While $\|\nabla f(x_k)\| > \varepsilon$ **do**
  Compute $d_k = -B_k^{-1}\nabla f(x_k)$.
  Find t_k satisfying the Wolfe conditions.
  Update $x_{k+1} = x_k + t_k d_k$.
  Compute $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.
  Update B_{k+1} using the BFGS formula.
end While
return x_k

of B_k rapidly expensive in high dimensions. The *l-BFGS* method [67] provides an efficient alternative: instead of storing the entire matrix, it keeps only the last l pairs (s_i, y_i) and reconstructs the action of B_k^{-1} on the gradient implicitly. This approach drastically reduces memory and computational costs while preserving the superlinear convergence observed in BFGS.

Another popular update is the *SR1* (Symmetric Rank-One) update [75, Chapter 6.2]. It approximates the Hessian $\nabla^2 f(x)$ from successive gradient differences, without enforcing positivity of the approximation. This lack of constraint allows it to capture local curvature more faithfully—including negative curvature—which makes it particularly suitable (though sometimes numerically unstable) for nonconvex problems.

2.3 Quadratic Regularization Method

The quadratic regularization algorithm (R2) [25] aims to solve (2.1) by minimizing, at each iteration, a regularized quadratic model of the objective function. This model takes the form

$$m_k(s) = f(x_k) + \nabla f(x_k)^T s + \tfrac{1}{2}\sigma_k \|s\|^2,$$

where the regularization parameter $\sigma_k > 0$ is dynamically adjusted at each iteration. The R2 method is detailed in Algorithm 4.

Algorithm 4 R2 Algorithm

Choose constants $0 < \eta_1 \leq \eta_2 < 1$, and $0 < \gamma_3 \leq 1 \leq \gamma_1 \leq \gamma_2$, stopping tolerance $\varepsilon > 0$.
Choose $x_0 \in \mathbb{R}^n$, $\sigma_0 > 0$, and compute $f(x_0)$.
For $k = 0, 1, \ldots$ **do**
  If $\|\nabla f(x_k)\| \leq \varepsilon$ **then**
    Return x_k.
  end If
  Define the model m_k as above and compute a point $s_k \in \operatorname{argmin}_s m_k(s)$.
  Compute the ratio

$$\rho_k := \frac{f(x_k) - f(x_k + s_k)}{m_k(0) - m_k(s_k)}.$$

  If $\rho_k \geq \eta_1$ **then**
    $x_{k+1} = x_k + s_k$.
  else
    $x_{k+1} = x_k$.
  end If
  Update σ_k according to

$$\sigma_{k+1} \in \begin{cases} [\gamma_3 \sigma_k, \sigma_k], & \text{if } \rho_k \geq \eta_2, \quad \text{(very successful iteration)}, \\ [\sigma_k, \gamma_1 \sigma_k], & \text{if } \eta_1 \leq \rho_k < \eta_2, \quad \text{(successful iteration)}, \\ [\gamma_1 \sigma_k, \gamma_2 \sigma_k], & \text{if } \rho_k < \eta_1, \quad \text{(unsuccessful iteration)}. \end{cases}$$

end For

With the quadratic model above, the step s_k is computed explicitly as

$$s_k = -\frac{1}{\sigma_k} \nabla f(x_k).$$

The R2 method achieves a stationary point in $\mathcal{O}(\varepsilon^{-2})$ iterations. This result is comparable to that of trust-region methods, to which it is closely related.

The R2 algorithm has given rise to many recent variants [8, 9, 11, 40, 61, 71]. We develop a new one in this manuscript, presented in Chapter 6. Moreover, the R2 method provides a natural transition to trust-region methods, introduced in the next section.

2.4 Trust-Region Methods

Trust-region methods approximate the objective function f by a local model, for example a linear model $m_k(s) := f(x_k) + \nabla f(x_k)^T s$ around the current point x_k. They define a region within which this model is assumed to represent f accurately. The step s_k is then determined

by solving the subproblem

$$\min_{s\in\mathbb{R}^n} m_k(s) \quad \text{subject to } \|s\| \le \Delta_k,$$

where $\Delta_k > 0$ is the trust-region radius at iteration k.

These methods are used in various contexts, such as smooth optimization [36], nonsmooth optimization [9], and derivative-free optimization [13, 37].

A general scheme is presented in Algorithm 5.

Algorithm 5 Trust-Region Method

Choose constants $0 < \eta_1 \le \eta_2 < 1$, and $0 < \gamma_3^{-1} \le \gamma_1 \le \gamma_2 < 1 < \gamma_3 \le \gamma_4$, stopping tolerance $\varepsilon > 0$.
Choose $x_0 \in \mathbb{R}^n$, $\Delta_0 > 0$, and compute $f(x_0)$.
For $k = 0, 1, \ldots$ **do**
  If $\|\nabla f(x_k)\| \le \varepsilon$ **then**
    Return x_k.
  end If
  Define a model m_k and compute a step s_k as an approximate solution to

$$\min_{\|s\|\le\Delta_k} m_k(s).$$

  Compute the ratio

$$\rho_k := \frac{f(x_k) - f(x_k + s_k)}{m_k(0) - m_k(s_k)}.$$

  If $\rho_k \ge \eta_1$ **then**
    $x_{k+1} = x_k + s_k$.
  else
    $x_{k+1} = x_k$.
  end If
  Update Δ_k according to

$$\Delta_{k+1} \in \begin{cases} [\gamma_3\Delta_k, \gamma_4\Delta_k], & \text{if } \rho_k \ge \eta_2, \quad \text{(very successful iteration)}, \\ [\gamma_2\Delta_k, \Delta_k], & \text{if } \eta_1 \le \rho_k < \eta_2, \quad \text{(successful iteration)}, \\ [\gamma_1\Delta_k, \gamma_2\Delta_k], & \text{if } \rho_k < \eta_1 \quad \text{(unsuccessful iteration)}. \end{cases}$$

end For

The update of the radius Δ_k follows an intuitive logic. During a very successful iteration ($\rho_k \ge \eta_2$), the model m_k proves to be a good approximation of f in the considered region. It is then appropriate to increase the trust-region size for the next iteration. Conversely, during an unsuccessful iteration ($\rho_k < \eta_1$), the predictive quality of the model is insufficient, and Δ_k

should be reduced to restrict the area where the model is considered reliable. This adaptive logic is analogous to the adjustment of σ_k^{-1} in the R2 algorithm.

Another common example of a model used is the quadratic model

$$m_k(s) = f(x_k) + \nabla f(x_k)^T s + \tfrac{1}{2} s^T B_k s,$$

where B_k is a quasi-Newton approximation of the Hessian $\nabla^2 f(x_k)$, such as l-BFGS or SR1. The identification $B_k = \sigma_k I$ in the previous model shows that the R2 algorithm can be viewed as a trust-region method with radius $\Delta_k = \sigma_k^{-1}$.

Under standard assumptions on f and the sequence $\{B_k\}_{k\geq 0}$, the trust-region subproblem (step 7 of Algorithm 5) admits a solution [36, Algorithm 4.3]. However, in practice, it is not necessary to solve it exactly: an approximate solution satisfying certain decrease conditions is sufficient to guarantee global convergence.

2.5 Cauchy Point

A Cauchy point ensures a so-called "Cauchy" decrease of the quadratic model m_k defined by $m_k(s) = f(x_k) + \nabla f(x_k)^T s + \frac{1}{2} s^T B_k s$ with B_k symmetric. It is a step $s_{k,\text{cp}} = -\tau_k \nabla f(x_k)$ computed along the steepest-descent direction, with

$$\tau_k = \operatorname*{argmin}_{t \in (0, \Delta_k / \|\nabla f(x_k)\|]} m_k(-t \nabla f(x_k)),$$

where $\Delta_k > 0$ is the trust-region radius. When $\nabla f(x_k)^T B_k \nabla f(x_k) > 0$, we have

$$\tau_k = \min\{\|\nabla f(x_k)\|^2 / (\nabla f(x_k)^T B_k \nabla f(x_k)), \Delta_k / \|\nabla f(x_k)\|\}$$

and otherwise $\tau_k = \Delta_k / \|\nabla f(x_k)\|$. Under standard assumptions (lower boundedness of f, symmetry of B_k), the decrease at the Cauchy point satisfies the classical bound $m_k(0) - m_k(s_{k,\text{cp}}) \geq \frac{1}{2} \|\nabla f(x_k)\| \min\{\|\nabla f(x_k)\| / \|B_k\|, \Delta_k\}$.

This bound provides an explicit lower bound on the model decrease and plays a central role in the analysis of trust-region methods. In practice, one first computes the Cauchy point, then attempts to improve it with a more sophisticated solver for the subproblem. Moreover, the R2 algorithm achieves $\mathcal{O}(\varepsilon^{-2})$ complexity for satisfying a first-order stationarity criterion as soon as the accepted step yields at least the Cauchy decrease, i.e., $m_k(s_k) \leq m_k(s_{k,\text{cp}})$ for all $k \geq 0$ [30, Chapter 2].

2.6 Proximal Gradient Method

In this section, we consider the nonsmooth problem $\min_{x\in\mathbb{R}^n} f(x)+h(x)$, where f is of class $\mathcal{C}^1$ and h is generally nonsmooth. We refer to standard references for usual definitions [18, Chapters 3, 6 and 10].

The lack of differentiability of h generally prevents the direct application of gradient descent to $f+h$. A classical approach is the proximal gradient method [18, Chapter 10], based on the proximal operator.

Definition 2. *Let $h:\mathbb{R}^n\to\mathbb{R}\cup\{+\infty\}$ be proper and lower semicontinuous, and let $\nu>0$. The proximal operator of h at x is defined by* $\operatorname{prox}_{\nu h}(x) := \operatorname{argmin}_{u\in\mathbb{R}^n} \frac{1}{2}\|u-x\|^2+\nu h(u)$.

If h is convex, $\operatorname{prox}_{\nu h}(x)$ is a singleton for all $\nu>0$ [18, Th. 6.3]. In the nonconvex case, if h is prox-bounded with threshold $\nu_h>0$, then for any $\nu\in(0,\nu_h)$, $\operatorname{prox}_{\nu h}(x)$ is nonempty and compact [84, Th. 1.25].

We provide an illustration of the proximal operator in Figure 2.3.

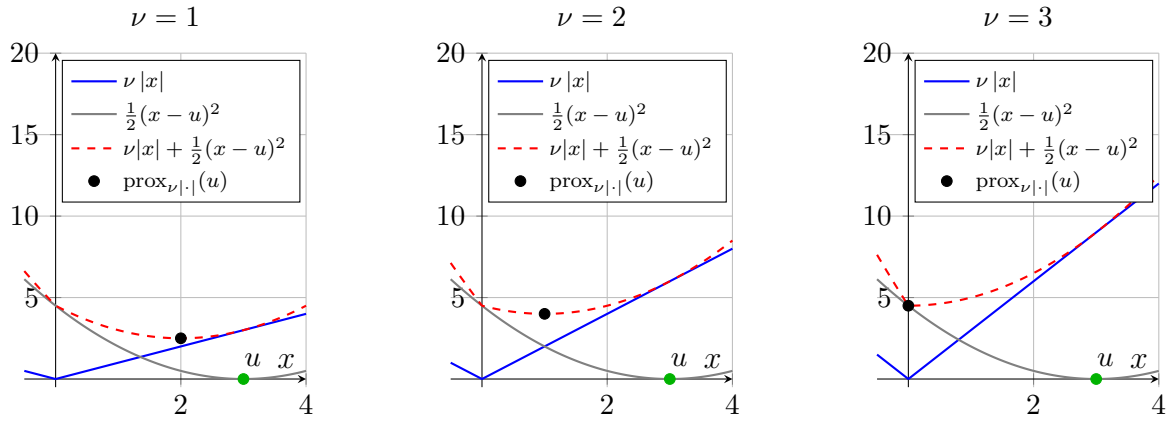

Figure 2.3 Comparison of $\operatorname{prox}_{\nu|\cdot|}(u)$ for three values of ν with $u=3$. The red dashed curve represents the objective of the proximal operator, and the black point denotes the minimizer of this objective.

As illustrated in Figure 2.3, where h is the absolute value function, the proximal operator seeks the best compromise between minimizing the quadratic term and the nonsmooth term. The relative importance of h compared to the quadratic term is controlled by the value of ν: the larger ν is, the more weight is given to minimizing the nonsmooth term.

Under the assumption that ∇f is L-Lipschitz continuous, the proximal gradient method is obtained by minimizing, around x_k, the model $s\mapsto f(x_k)+\nabla f(x_k)^T s+\frac{1}{2}\nu^{-1}\|s\|^2+h(x_k+s)$.

Setting $u = x_k + s$ yields the standard update $x_{k+1} \in \text{prox}_{\nu h}\big(x_k - \nu \nabla f(x_k)\big)$. Equivalently, $x_{k+1} - x_k \in \text{argmin}_{s \in \mathbb{R}^n} \nabla f(x_k)^T s + \frac{1}{2\nu}\|s\|^2 + h(x_k + s)$. Thus, the proximal gradient method can be seen as a steepest descent step $(x_k - \nu \nabla f(x_k))$, followed by a proximal step $x_{k+1} \in \text{argmin}_u \frac{1}{2}\|u - (x_k - \nu \nabla f(x_k))\|^2 + \nu h(u)$ or $s_k \in \text{argmin}_s \frac{1}{2}\|s + \nu \nabla f(x_k)\|^2 + \nu h(x_k + s)$.

If h is proper, lower semicontinuous, and bounded below, and if ∇f is L-Lipschitz continuous, then for any $\nu \in (0, 1/L)$, the sequence $\{x_k\}$ produced by the proximal gradient method satisfies the descent inequality $f(x_{k+1}) + h(x_{k+1}) - \big(f(x_k) + h(x_k)\big) \leq -\frac{1}{2}(\nu^{-1} - L)\|x_{k+1} - x_k\|^2$ [19, Lem. 2]. This relation generalizes the "Cauchy point" type decrease to the nonsmooth setting by controlling the reduction of $f + h$ through the displacement $x_{k+1} - x_k$.

The proximal operator is available in closed form for many common functions h [18, Chapter 6]. We will return later to the case where $\text{prox}_{\nu h}$ is not explicitly computable.

2.7 Quasi-Newton Method for Nonsmooth Optimization

This section develops a first trust-region algorithm based on a quasi-Newton update for nonsmooth optimization. This algorithm was initially proposed in [9] and serves as the starting point for the work presented in Chapter 6.

2.7.1 Some Notation

Before presenting the algorithm, we introduce some notation.

For $\nu > 0$ and $x \in \mathbb{R}^n$, the first-order models

$$\varphi_{\text{cp}}(s; x) := f(x) + \nabla f(x)^T s, \tag{2.2}$$

$$\psi(s; x) \approx h(x + s), \tag{2.3}$$

$$m_{\text{cp}}(s; x, \nu^{-1}) := \varphi_{\text{cp}}(s; x) + \tfrac{1}{2}\nu^{-1}\|s\|^2 + \psi(s; x), \tag{2.4}$$

serve to generalize the concept of a Cauchy point—hence the notation "cp"—and where we use "$\approx$" to indicate that the left-hand side is an approximation of the right-hand side. Models (2.2)–(2.4) are used both to define a sufficient decrease threshold at each iteration and to introduce a pseudo-measure of stationarity.

For $\sigma > 0$, $x \in \mathbb{R}^n$, and $B(x) = B(x)^T \in \mathbb{R}^{n \times n}$, the second-order models

$$\varphi(s; x) := f(x) + \nabla f(x)^T s + \tfrac{1}{2} s^T B(x) s, \tag{2.5}$$

$$m(s; x, \sigma) := \varphi(s; x) + \tfrac{1}{2}\sigma\|s\|^2 + \psi(s; x), \tag{2.6}$$

are used to compute a step improving upon the Cauchy point. We use the notation $s_{k,\text{cp}} \in \operatorname{argmin}_s m_{\text{cp}}(s; x_k, \nu_k^{-1})$ and $s_k \in \operatorname{argmin}_s m(s; x_k, \sigma_k)$. Finally, we define the decrease of $\varphi+\psi$ at $s_{k,\text{cp}}$ as the quantity

$$\xi_{\text{cp}}(s_{k,\text{cp}}, x_k, \nu_k^{-1}) := (\varphi_{\text{cp}} + \psi)(0; x_k) - (\varphi_{\text{cp}} + \psi)(s_{k,\text{cp}}; x_k). \tag{2.7}$$

2.7.2 Proximal Trust-Region Algorithm with Quasi-Newton Update

We omit here the assumptions made on f, h, as well as on models (2.4) and (2.6). For more details, the reader is referred to the corresponding works [9].

Algorithm 6 Nonsmooth Regularized Trust-Region Algorithm

Choose constants $0 < \eta_1 \le \eta_2 < 1$, $0 < \gamma_1 \le \gamma_2 < 1 < \gamma_3 \le \gamma_4$, and $\alpha > 0$, $\beta \ge 1$.
Choose $x_0 \in \mathbb{R}^n$, where h is finite, $\Delta_0 > 0$, and compute $f(x_0) + h(x_0)$.
For $k = 0, 1, \ldots$ **do**
  Choose $0 < \nu_k \le 1/\left(L + \alpha^{-1}\Delta_k^{-1}\right)$.
  Define the model $m_{\text{cp}}(s; x_k, \nu_k^{-1})$ as in (2.4).
  Define $m(s; x_k, \sigma_k)$ as in (2.6).
  Compute $s_{k,\text{cp}}$.
  Compute s_k, an approximate solution of $m(s; x_k, \sigma_k)$, such that

$$\|s_k\| \le \min(\Delta_k, \beta\|s_{k,\text{cp}}\|).$$

  Compute the ratio

$$\rho_k := \frac{f(x_k) + h(x_k) - \left(f(x_k + s_k) + h(x_k + s_k)\right)}{m(0; x_k, \sigma_k) - m(s_k; x_k, \sigma_k)}.$$

  If $\rho_k \ge \eta_1$ **then**
    $x_{k+1} = x_k + s_k$.
  else
    $x_{k+1} = x_k$.
  end If
  Update the trust-region radius as follows:

$$\Delta_{k+1} \in \begin{cases} [\gamma_3\Delta_k, \gamma_4\Delta_k], & \text{if } \rho_k \ge \eta_2, \quad \text{(very successful iteration)}, \\ [\gamma_2\Delta_k, \Delta_k], & \text{if } \eta_1 \le \rho_k < \eta_2, \quad \text{(successful iteration)}, \\ [\gamma_1\Delta_k, \gamma_2\Delta_k], & \text{if } \rho_k < \eta_1, \quad \text{(unsuccessful iteration)}. \end{cases}$$

end For

As explained in Section 2.6, computing $s_{k,\text{cp}}$ essentially amounts to performing one step of

the proximal gradient method. Without going into details, the authors of [9] also introduce a method equivalent to Algorithm 4 for the nonsmooth case. This method is used to compute s_k in step 8 of Algorithm 6, where the objective f is the quadratic model from (2.6).

A method similar to Algorithm 6 is R2N [40], which allows f to be only continuously differentiable and the sequence $\{B_k\}_{k\geq 0}$ to be unbounded under certain conditions.

Algorithm 7 R2N: Modified Proximal Quasi-Newton Method

1: Choose constants $0 < \theta_1 < 1 < \theta_2$, $0 < \eta_1 \leq \eta_2 < 1$, and $0 < \gamma_3 \leq 1 \leq \gamma_1 \leq \gamma_2$.
2: Choose $x_0 \in \mathbb{R}^n$ where h is finite, and $0 < \sigma_{\min} < \sigma_0$.
3: **For** $k = 0, 1, \ldots$ **do**
4: Choose $B_k := B(x_k) \in \mathbb{R}^{n\times n}$ such that $B_k = B_k^T$.
5: Compute $\nu_k := \theta_1/(\|B_k\| + \sigma_k)$.
6: Compute $s_{k,\text{cp}} \in \operatorname{argmin}_s m_{\text{cp}}(s; x_k, \nu_k^{-1})$ defined in (2.4) and $\xi_{\text{cp}}(x_k, \nu_k^{-1})$ defined in (2.7).
7: Compute a step s_k such that $m(s_k; x_k, \sigma_k) \leq m(s_{k,\text{cp}}; x_k, \sigma_k)$, with m defined in (2.6).
8: **If** $\|s_k\| > \theta_2\|s_{k,\text{cp}}\|$ **then** reset $s_k = s_{k,\text{cp}}$.
9: **end If**
10: Compute the ratio

$$\rho_k := \frac{f(x_k) + h(x_k) - \Big(f(x_k + s_k) + h(x_k + s_k)\Big)}{\varphi(0; x_k) + \psi(0; x_k) - \Big(\varphi(s_k; x_k) + \psi(s_k; x_k)\Big)}.$$

11: **If** $\rho_k \geq \eta_1$ **then** set $x_{k+1} = x_k + s_k$.
12: **else** set $x_{k+1} = x_k$.
13: **end If**
14: Update the regularization parameter as follows:

$$\sigma_{k+1} \in \begin{cases} [\gamma_3\sigma_k, \sigma_k], & \text{if } \rho_k \geq \eta_2, \quad \text{(very successful iteration)}, \\ [\sigma_k, \gamma_1\sigma_k], & \text{if } \eta_1 \leq \rho_k < \eta_2, \quad \text{(successful iteration)}, \\ [\gamma_1\sigma_k, \gamma_2\sigma_k], & \text{if } \rho_k < \eta_1, \quad \text{(unsuccessful iteration)}. \end{cases}$$

15: **end For**

In Chapter 6, we develop an extension of Algorithm 7 to the case where f, ∇f, and the proximal operator are potentially evaluated inexactly. Specifically, our method computes a step $\widehat{s}_{k,\text{cp}}$ such that $m_{\text{cp}}(\widehat{s}_{k,\text{cp}}; x_k, \nu_k^{-1}) \approx \min_s m_{\text{cp}}(s; x_k, \nu_k^{-1})$, as long as the decrease $\widehat{\xi}_{k,\text{cp}}$ associated with $\widehat{s}_{k,\text{cp}}$ is at least a positive fraction of that associated with the exact step $s_{k,\text{cp}}$.

2.8 Large-Scale Optimization

In this section, we review the fundamentals of large-scale smooth optimization—particularly neural network training—from which our work in Chapters 4 and 5 is developed.

2.8.1 Fundamental Concepts

When the problem to be solved involves a large number of optimization variables, it is generally numerically difficult to apply methods such as gradient descent Algorithm 1 or (quasi)-Newton methods Algorithm 2, due to the need to store a large amount of information (gradient, previous iterates, etc.). This is especially the case when training large neural networks to solve (1.2).

To address this, a very popular method is stochastic gradient descent (SGD) [7]. The principle is as follows: iteratively select a set of indices as in (1.3), compute the gradient of the objective over this subset, update the optimization parameters, and repeat. In the context of neural networks, this parameter update is known as "backpropagation."

In Algorithm 8, we consider a network with L layers indexed by $\ell = 1, \dots, L$, with layer sizes n_ℓ. For each layer ℓ, $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $b_\ell \in \mathbb{R}^{n_\ell}$ are respectively the weights and biases; $x_\ell = (W_\ell, b_\ell)$, and $x = \{x_\ell\}_{\ell=1}^L$. For an input e_i, we set $a_0 = e_i$, and for $\ell \geq 1$, $z_\ell = W_\ell a_{\ell-1} + b_\ell$ (pre-activation) and $a_\ell = \sigma_\ell(z_\ell)$ (activation), where σ_ℓ is the activation function (applied componentwise) and σ'_ℓ its derivative. The mini-batch is $B = \{(e_i, s_i)\}_{i \in \mathcal{I}}$ of size m. The per-sample loss is $\ell_i = f(a_L, s_i)$, and the average mini-batch loss is $f_B = \frac{1}{m} \sum_{i \in \mathcal{I}} \ell_i$. The quantities $\delta_\ell \in \mathbb{R}^{n_\ell}$ denote the backpropagated local derivatives, and $\odot$ denotes the Hadamard product. The per-sample gradients are $\nabla_{W_\ell} \ell_i = \delta_\ell a_{\ell-1}^T$ and $\nabla_{b_\ell} \ell_i = \delta_\ell$; the average gradients over B are obtained by averaging.

Algorithm 8 Backpropagation for an L-layer network

Inputs: mini-batch $B = \{(e_i, s_i)\}_{i\in\mathcal{I}}$, parameters $\{x_\ell = (W_\ell, b_\ell)\}_{\ell=1}^{L}$, activation functions $\{\sigma_\ell\}$, loss f.
(Forward pass) For each $i \in \mathcal{I}$:
  $a_0 \leftarrow e_i$.
For $\ell = 1, 2, \ldots, L$ **do**
  $z_\ell \leftarrow W_\ell a_{\ell-1} + b_\ell$.
  $a_\ell \leftarrow \sigma_\ell(z_\ell)$.
end For
Compute the loss $\ell_i \leftarrow f(a_L, s_i)$.
(Backward pass) Initialize the output derivative:
  $\delta_L \leftarrow \nabla_{a_L} f(a_L, s_i) \ \odot \ \sigma'_L(z_L)$.
For $\ell = L, L-1, \ldots, 1$ **do**
  $\nabla_{W_\ell}\ell_i \leftarrow \delta_\ell a_{\ell-1}^T, \quad \nabla_{b_\ell}\ell_i \leftarrow \delta_\ell$.
  If $\ell > 1$ **then**
    $\delta_{\ell-1} \leftarrow (W_\ell^T \delta_\ell) \ \odot \ \sigma'_{\ell-1}(z_{\ell-1})$.
  end If
end For
(Mini-batch aggregation)
$$\nabla_{W_\ell} f_B \leftarrow \frac{1}{m}\sum_{i\in\mathcal{I}} \nabla_{W_\ell}\ell_i, \quad \nabla_{b_\ell} f_B \leftarrow \frac{1}{m}\sum_{i\in\mathcal{I}} \nabla_{b_\ell}\ell_i.$$
Outputs: average gradients $\{\nabla_{W_\ell} f_B, \nabla_{b_\ell} f_B\}_{\ell=1}^{L}$.

Backpropagation uses the *chain rule* to propagate the network's error from the output layer back through the preceding layers. In the context of Algorithm 8, this corresponds to the recursive computation of the local derivatives δ_ℓ.

Indeed, the quantity

$$\delta_L = \nabla_{a_L} f(a_L, s_i) \odot \sigma'_L(z_L)$$

measures the sensitivity of the loss f with respect to the activations of the final layer. This error is then propagated backward layer by layer according to the relation

$$\delta_{\ell-1} = (W_\ell^T \delta_\ell) \odot \sigma'_{\ell-1}(z_{\ell-1}),$$

which directly follows from the chain rule applied to the composition of activation functions and linear transformations in the network.

Thus, at each iteration of the **For** loop in Algorithm 8, backpropagation efficiently computes the gradients $\nabla_{W_\ell}\ell_i = \delta_\ell a_{\ell-1}^T$ and $\nabla_{b_\ell}\ell_i = \delta_\ell$, without explicitly forming all the partial derivatives of the entire network. This approach exploits the hierarchical structure of the model to make the computation of the global gradient ∇f both fast and memory-efficient.

We describe the SGD procedure in Algorithm 9.

Algorithm 9 Stochastic Gradient Descent (SGD, mini-batch)

1: **Inputs:** initial parameters x_0, step sizes $\{\alpha_k\}_{k\geq 0}$, mini-batch size m.
2: **For** $k = 0, 1, \ldots$ **do**
3: Sample a mini-batch $\mathcal{I}_k$ of size m as in (1.3).
4: Estimate the mini-batch gradient as in Algorithm 8:

$$g_k \leftarrow \frac{1}{m} \sum_{i \in \mathcal{I}_k} \nabla f(x_k; e_i, s_i).$$

5: Update the parameters:

$$x_{k+1} \leftarrow x_k - \alpha_k g_k.$$

6: **(Optional)** Stopping criterion if $\|g_k\|$ or the loss variation is small.
7: **end For**

As a devoted defender of the kingdom of mathematics, it must be noted that despite its name, the SGD method is **not** a true descent method. Indeed, unlike methods where the direction $d_k = -\nabla f(x_k)$ guarantees by construction a local decrease in the objective function, the direction used in SGD is based on a noisy estimate of the gradient:

$$g_k = \frac{1}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f(x_k; e_i, s_i),$$

where $\mathcal{I}_k \subset \{1, \ldots, N\}$ is a random subset of sample indices. Thus, the step

$$x_{k+1} = x_k - \alpha_k g_k$$

does not ensure that $f(x_{k+1}) < f(x_k)$ and may even increase it locally. It is only *in expectation* over the distribution of subsamples $\mathcal{I}_k$ that the update direction is a descent direction:

$$\mathbb{E}[g_k] = \nabla f(x_k).$$

In other words, SGD does not follow a strictly descending trajectory but rather stochastically explores the landscape of f, giving it a useful ability to escape local minima and plateaus, at the cost of a noisy convergence.

2.8.2 ADAM

In practice, it is often beneficial to accelerate gradient descent by adding a form of inertia, called *momentum* [77]. The idea is to smooth successive updates by accumulating an ex-

ponential moving average of past gradients, which helps dampen oscillations and speed up convergence in elongated valleys of the optimization landscape. The *momentum* update is given by:

$$v_k = \beta v_{k-1} + (1-\beta)\nabla f(x_k), \qquad x_{k+1} = x_k - \alpha v_k,$$

where $\beta \in [0,1)$ controls the persistence of motion (typically $\beta = 0{,}9$). When the gradient maintains a consistent direction across iterations, v_k accumulates this information and amplifies the descent step, while in the presence of oscillations, the moving average reduces the effect of stochastic noise.

The ADAM method [59] (*Adaptive Moment Estimation*) is an advanced variant of stochastic gradient descent that adapts the learning rate for each parameter individually. It combines the ideas of two classical algorithms: *Momentum*, which accumulates a moving average of the gradient to smooth updates, and *RMSProp* [53], which normalizes steps according to the gradient variance. Thus, ADAM dynamically adjusts both the size and direction of the steps using biased estimators of the first and second moments of the gradient.

Algorithm 10 ADAM Method (*Adaptive Moment Estimation*)

Input: Initial learning rate $\alpha > 0$, exponential parameters $\beta_1, \beta_2 \in [0,1)$, and $\varepsilon > 0$.

1: Initialize $m_0 = 0$, $v_0 = 0$, and choose x_0.

2: **For** $k = 1, 2, \ldots$ **do**

3: Compute the stochastic gradient $g_k = \nabla f_{\mathcal{I}_k}(x_{k-1})$ using Algorithm 8.

4: Update the moving averages:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k, \qquad v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^{\odot 2}.$$

5: Correct the bias:

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k}, \qquad \hat{v}_k = \frac{v_k}{1-\beta_2^k}.$$

6: Update the parameters:

$$x_k = x_{k-1} - \alpha \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \varepsilon}.$$

7: **end For**

The combination of these two moving averages allows ADAM to benefit from both a more stable update direction and a learning rate scaled appropriately for each coordinate. In practice, it converges quickly and robustly, even in noisy or ill-conditioned settings. *ADAMW* [69] is a variant of ADAM that slightly modifies the update rule to avoid undesirable algorithmic coupling with the adaptive moment scaling. Empirically, this reformulation makes the optimizer more stable and often improves generalization, while retaining ADAM's bias corrections and per-coordinate adaptive step sizes.

For both Algorithm 9 and Algorithm 10, the Algorithm 8 requires storing the sampled gradient, activations, and possibly the first and second moment estimates. This storage represents a significant (often dominant, see Chapter 5) portion of the total memory used during model training.

It then becomes desirable to devise a method for training such models that does not rely on backpropagation.

2.9 Large-Scale Optimization Without Backpropagation

A well-known derivative-free optimization method for estimating a gradient along a given direction is the finite-difference method [45]:

$$\widehat{\nabla} f(x; Z, \epsilon, \xi) := \frac{f(x + \epsilon Z; \xi) - f(x - \epsilon Z; \xi)}{2\epsilon} Z, \tag{2.8}$$

where $Z \sim \mathcal{N}(0, I) \in \mathbb{R}^n$ and $\epsilon > 0$. As $\epsilon \to 0$, we have $\frac{f(x+\epsilon Z;\xi)-f(x-\epsilon Z;\xi)}{2\epsilon} Z \to Z^T \nabla f(x; \xi)$, so (2.8) can be seen as a projection of the gradient onto the direction Z. Since $Z \sim \mathcal{N}(0, I)$, (2.8) is an unbiased gradient estimator, that is, $\mathbb{E}_Z\left[\widehat{\nabla} f(x; Z, \epsilon, \xi)\right] = \nabla f(x; \xi)$.

Malladi et al. [70] were the first to propose training language models using (2.8) within the MeZO framework. More specifically, they *finetune* language models, which corresponds to solving a simpler optimization problem than full training from scratch. Later, Chen et al. [31] improved MeZO with LoZO, introducing a low-rank method that is more memory-efficient. In Chapters 4 and 5, we extend these works to the training of language models.

2.10 Pretraining versus finetuning

Along this document, we will use the terms *pretraining* and *finetuning* to refer to two distinct phases of training large language models (LLMs). *Pretraining* refers to the initial training phase of a language model on a large corpus of text data. During this phase, the model learns general language patterns, grammar, and contextual relationships by predicting missing words or generating text based on input prompts. This phase typically requires substantial computational resources and time, as it involves training on vast datasets to capture a wide range of linguistic features. On the other hand, *finetuning* is a subsequent phase where the pretrained model is further trained on a smaller, task-specific dataset. The goal of finetuning is to adapt the general language understanding acquired during pretraining to perform well on a particular task, such as sentiment analysis, question answering, or text classification. Finetuning usually requires less computational power and time compared

to pretraining, as it leverages the knowledge already embedded in the pretrained model. In Chapters 4 and 5, we focus on the pretraining of language models using derivative-free optimization methods. A specific section on finetuning is included in Section 5.6.

2.11 Inexact Evaluations

In the next section, we focus on various types of inexact evaluations, which are central to this thesis.

2.11.1 IEEE 754 Standard

The representation of numbers in computers, particularly floating-point numbers, is standardized by the IEEE 754 standard [1]. This standard, established to ensure computational consistency across different hardware systems, defines how numbers are stored and manipulated in memory.

It specifies several formats for representing floating-point numbers, the most common being single precision (32-bit) and double precision (64-bit). It also defines rounding rules, exception handling (such as overflows or undefined results), and basic arithmetic operations.

Floating-point numbers are represented by a base, an exponent, and a significand (also called a mantissa). The base is 2 for binary representations and 10 for decimal ones. The exponent determines the magnitude of the number, while the significand controls its precision. Higham [52] describes the principles of floating-point arithmetic as follows.

A floating-point system $\mathbb{F} \subset \mathbb{R}$ is a subset of real numbers whose elements have the form:

$$y = \pm m \times \beta^{e-t} \in \mathbb{F}. \tag{2.9}$$

The set $\mathbb{F}$ is thus characterized by four integer parameters:

- the base $\beta = 2$ for binary representations,
- the precision t,
- the exponent range $e_{\min} \leq e \leq e_{\max}$,
- the significand $0 \leq m \leq \beta^{t-1}$.

Table 2.1 compares 32-bit and 64-bit arithmetic.

Table 2.1 Comparison of parameters in 32-bit and 64-bit floating-point systems.

Parameter	32-bit	64-bit
Base (β)	2	2
Precision (t)	24	53
Exponent range (e)	-125 to 128	-1021 to 1024
Significand (m)	$0 \leq m \leq 2^{23}$	$0 \leq m \leq 2^{52}$

Thus, the smaller the precision t and the narrower the exponent range, the fewer elements the set $\mathbb{F}$ contains. $\mathbb{F}$ is therefore a discrete subset of $\mathbb{R}$. Representing a real number in floating-point arithmetic amounts to rounding it to an element of $\mathbb{F}$, introducing an inaccuracy known as rounding error. The minimum distance between two successive elements of $\mathbb{F}$ increases as precision decreases, leading to larger rounding errors in the worst case. As multiple floating-point operations accumulate, so do these errors, meaning that the final result of a computation is very likely not the exact theoretical value.

2.11.2 Other Inexact Evaluations

When evaluating one of the components of a problem requires substantial computational resources (numerical simulation, solving a system of differential equations, large-scale problem, etc.), it becomes relevant to simplify this evaluation. Concretely, this may involve stopping a procedure before optimality, using only part of the available information as in stochastic gradients [7], or employing a surrogate model [13]. By definition, the resulting evaluation is then *inexact*.

When designing robust optimization algorithms, it is essential to account for such sources of inexactness in the convergence analysis.

In Chapters 4 and 5, the inexactness arises from two sources: first, from the sampling of the objective (and therefore its gradient) in (1.2), and second, from the zeroth-order method used during the optimization process. Indeed, in (2.8), the optimization relies on an estimated gradient. In Chapter 6, inexactness may stem either from inexact evaluations of f and ∇f, or from inexact evaluations of the proximal operator (Definition 2).

CHAPTER 3 ORGANIZATION OF THE THESIS

This thesis is composed of eight chapters, two of which are dedicated to research papers summarizing the main contributions, and one chapter corresponding to a paper to be submitted. It is organized as follows.

Chapter 1 establishes the general framework and objectives of the research.

Chapter 2 presents the theoretical concepts required to understand the manuscript and reviews relevant works from the literature.

Chapter 4 addresses problem (1.2) and introduces a new method for training language models inspired by derivative-free optimization. This approach significantly reduces the memory footprint by removing the need to store intermediate activations and gradients used during backpropagation, thus enabling the training of large models on limited hardware resources. The associated paper [4] was presented at the *International Conference on Pattern Recognition Applications and Methods 2025* (ICPRAM 2025), where it received the Best Industrial Paper Award.

Chapter 5 extends the work of Allaire et al. [4] by introducing theoretical and methodological improvements to the previously presented method. In particular, the method generates search directions structured through Kronecker products, making them extremely lightweight. These directions are then compared in terms of objective descent and must satisfy a sufficient decrease condition. These developments are supported by a convergence analysis and experimental results illustrating their efficiency.

Finally, in Chapter 6, we present a modified quasi-Newton method for (1.4), designed to remain robust under certain inexact evaluations. In particular, we prove the convergence of the method when the objective, its gradient, and the proximal operator are evaluated inexactly. We also show that the complexity bound remains of the same order as in the exact versions. Numerical experiments demonstrate that using inexact evaluations can significantly reduce computational costs.

Chapter 7 concludes this thesis by summarizing the contributions, discussing their limitations, and proposing directions for improvement and future research.

CHAPTER 4 ARTICLE 1: ZEROTH ORDER OPTIMIZATION FOR PRETRAINING LANGUAGE MODELS

Nathan Allaire and Sébastien Le Digabel and Mahsa Ghazvini Nejad and Vahid Partovi Nia

Manuscript published on 23-25 February 2025, in the *Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods* (ICPRAM)

Abstract

The physical memory for training Large Language Models (LLMs) grow with the model size, and are limited to the GPU memory. In particular, back-propagation that requires the computation of the first-order derivatives adds to this memory overhead. Training extremely large language models with memory-efficient algorithms is still a challenge with theoretical and practical implications. Back-propagation-free training algorithms, also known as zeroth-order methods, are recently examined to address this challenge. Their usefulness has been proven in fine-tuning of language models. However, so far, there has been no study for language model pretraining using zeroth-order optimization, where the memory constraint is manifested more severely. We build the connection between the second order, the first order, and the zeroth order theoretically. Then, we apply the zeroth order optimization to pre-training light-weight language models, and discuss why they cannot be readily applied. We show in particular that the curse of dimensionality is the main obstacle, and pave the way towards modifications of zeroth order methods for pre-training such models.

4.1 INTRODUCTION

For the past decades, first order (FO) optimization has been the preferred choice in the machine learning community. Stochastic gradient descent (SGD) [7] was introduced as an efficient and robust method for training and fine-tuning language models (LMs). Later, the Adam optimizer [59] and its variants [69] have been a major improvement for those tasks by adding momentum and adaptive learning rate to SGD. However, second order (SO) optimization is less common than FO methods such as SGD and Adam in machine learning community due to its higher computational and memory costs. The SO optimization adds some precious information and often yields a faster convergence than FO [88], but it is still under progress for training deep learning models.

Recently, researchers have shown that larger models lead to a smaller loss value and therefore lead to a more accurate model [56]. In return, LLMs continue to grow in size and complexity, and the memory constraints imposed by traditional training methods present a significant hurdle. Since the introduction of the BERT model [96] common language models have grown from 340M to 70B $\sim 200\times$ while the GPU memories have grown from 16GB to 80GB almost $5\times$. Moreover, Malladi et al. [70] showed that the back-propagation of the OPT-13B model requires $12\times$ more memory than inference. Those observations prompted a reevaluation of approaches for more resource-efficient learning, specially targeting training, pre-training, and fine-tuning.

Zeroth-order (ZO) methods includes derivative-free optimization methods also known as black box optimization. However, in the machine learning community it is referred to the algorithms that approximate the full gradient via gradient estimators based only on the function evaluation in the forward pass [26], [90]. In the context of machine learning, ZO methods do not need to back-propagate and therefore, cut the memory required in training step. Back-propagation-free methods for fine-tuning LLMs were first introduced by Malladi et al. [70] that unveils the first *Memory efficient Zeroth-Order* algorithm. Several extensions and variants of this algorithm were disclosed in [44], and historically initiated in [68] to develop *memory efficient zeroth order stochastic variance reduction of the gradient* to tackle the high-variance issue inherent to ZO. Recently, [105] proposed a benchmark on ZO methods and showed their efficiency on several fine-tuning tasks. We observed that the variance inflation in zeroth order training is rather a blessing.

All these works focus on *fine-tuning* of a language model and avoid addressing the more complex *pre-training* step. We dive into the largely unexplored field of ZO optimization for pre-training focusing on the following question: *can language models be effectively pre-trained using ZO optimization, and if not, what are the underlying limitations and how can they be overcome?*

To answer this question first we try to understand the relationship between second order, first order, and zero order using simple but insightful theory. Then we aim to focus on two key model characteristics: the dimension of the problem and the variance associated to ZO optimization in language model pretraining. We establish our experiments using a light-weight 20M transformer model due to lack of computational resources. However, we expect a similar behaviour for light-weight language models under 1B parameters. The behaviour of model pretraining changes often after surpassing billion parameters, see for instance [102].

Our main *contributions* include

- We make a brief overview of the main (SO, FO, ZO) optimization methods and *disclose several theoretical results on the optimal value of the learning rate* to establish a concrete connection between SO and FO in particular,
- We pre-train the Llama2-20M model [93] with vanilla FO and ZO optimization and showcase the *effectiveness of ZO methods in this context*,
- Run several experiments on a controlled ZO gradient variance, and demonstrate that the *high variance of ZO is needed for pre-training light weight language models*.

4.2 OPTIMIZATION BACKGROUND

Consider the unconstrained optimization problem

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^n} \mathcal{L}(\boldsymbol{\theta}), \qquad \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_i(\boldsymbol{\theta}), \tag{4.1}$$

where $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ is a non-convex loss function, each $\mathcal{L}_i$ is the loss of a single training instance. This optimization problem encapsulates most of deep learning training problems, including LLMs pretraining and fine-tuning.

4.2.1 Second order

Optimization is a fundamental aspect of various scientific and engineering disciplines, where the goal is to find the best parameter setting from a set of feasible solutions. While FO optimization methods, such as gradient descent, are widely used in deep learning training thanks to their simplicity and efficiency, they often struggle with issues like slow convergence and sensitivity to the choice of the step size, learning rate, etc. SO optimization techniques leverage not only the gradient (first derivative) but also the Hessian matrix (second derivative) of the objective function. By incorporating the curvature information, these methods provide a more accurate descent and a faster convergence to the minimum.

One of the most prominent SO optimization methods is Newton's method. Newton's method uses both the gradient and the Hessian matrix to iteratively find the stationary points of a function, by assuming an approximate quadratic function over the loss $\mathcal{L}$. This is a common assumption in neural networks near the minimum

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) \approx \tfrac{1}{2}\boldsymbol{\theta}^\top \mathbf{H}\boldsymbol{\theta} + \mathbf{g}^\top\boldsymbol{\theta}, \tag{4.2}$$

where $\{\nabla^2\mathcal{L}\} = \mathbf{H}$ is the $n \times n$ Hessian and $\nabla\mathcal{L} = \mathbf{g}$ is the gradient vector of size n. In

general, the Newton update provides potential candidates for local minima, but gives an exact minimum in a single update if the function is quadratic and has a unique minimum, i.e., has a positive definite Hessian. The update rule for Newton's method is given by

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}^{-1}\mathbf{g}, \tag{4.3}$$

where $\boldsymbol{\theta}_k$ is the current update, $\mathbf{g} = \nabla\mathcal{L}(\boldsymbol{\theta}_k)$ is the gradient, and $\mathbf{H} = \nabla^2\mathcal{L}(\boldsymbol{\theta}_k)$ is the Hessian, both evaluated at $\boldsymbol{\theta}_k$. The Newton's method achieves quadratic convergence near the optimal solution, making it significantly faster than FO methods for approximately quadratic functions. However, in deep learning applications, applying the Newton's method is challenging, e.g., the computational cost of calculating and inverting the $n \times n$ Hessian matrix, especially for high-dimensional problems with a large n such as large language models. In deep learning practice, the second order algorithms approximate the Hessian using the squared gradient, i.e., $\mathbf{H} \approx \mathbf{g}\mathbf{g}^\top$. This approximation has originated from the Fisher information identity, where the second derivative of the negative log-likelihood equals the gradient square in expectation [65].

4.2.2 First Order

One of the most widely used FO optimization methods is the gradient descent. The main idea behind the gradient descent is to iteratively move in the direction of the steepest descent, which is determined by the negative gradient of the function. The update rule for gradient descent is given by

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\mathbf{g}, \tag{4.4}$$

where η is a positive learning rate. This formulation surprisingly resembles (4.3). Hereafter, we aim to provide a clearer connection between SO and FO methods using a theoretical study on the learning rate. The learning rate is widely studied in the literature with established theoretical results under different assumptions for $\mathcal{L}$, see for instance [78, 100].

The gradient step in (4.4) equals the Newton step (4.3) for a diagonal Hessian with constant positive diagonal elements $h_{ii} = \eta^{-1}, \forall i \in \{1, \dots, n\}$, see Figure 4.1. FO methods like gradient descent (GD) typically exhibit linear convergence, meaning the error decreases proportionally to the current error at each step. In contrast, Newton's method often achieves quadratic convergence near the optimal solution. In other words, the error decreases proportionally to the square of the current error, leading to much faster convergence when the step is taken close to the minimum.

The optimal learning rate is rarely studied in practice, however it is not difficult to derive it

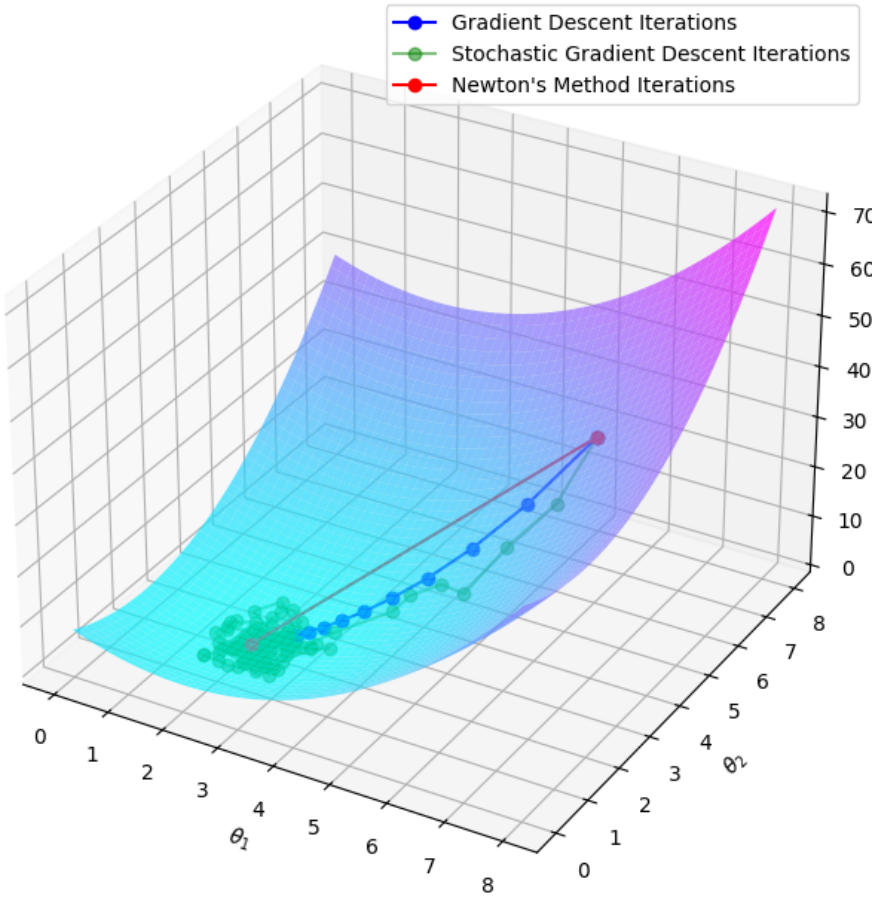

Figure 4.1 The Newton's method (red) compared with FO gradient descent (blue) and its stochastic variant (green).

in a quadratic problem. Note that the Newton's method moves to the minimum in a single iteration irrespective of the initial value. A quadratic problem is often formulated as

$$\mathcal{Q}(\boldsymbol{\theta}) = \tfrac{1}{2}\boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta} + \mathbf{b}^\top \boldsymbol{\theta},$$

where $\mathbf{A}$ is the matrix composed of the quadratic weights, and $\mathbf{b}$ is the linear component. We use align this notations with (4.2), and assume $\mathcal{Q} \equiv \hat{\mathcal{L}}$ is a quadratic approximation of a deep learning loss $\mathcal{L}$ for simplicity. Therefore, given a positive-definite Hessian, and a vector of gradients $\mathbf{g}$ the minimum of the quadratic approximation is attained at $\boldsymbol{\theta}^* = -\mathbf{H}^{-1}\mathbf{g}$.

Lemma 2. *Assume the quadratic problem*

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{H}\boldsymbol{\theta} + \mathbf{g}^\top \boldsymbol{\theta},$$

with a positive constant diagonal Hessian $\mathbf{H} = \lambda\mathbf{I}$*, where* $\mathbf{I}$ *is the identity matrix. The*

gradient descent attains the minimum with the optimal learning rate $\eta = \frac{1}{\lambda}$.

Proof. The proof is straightforward by assuring that the loss function attains its minimum in a single update, i.e., $\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t = -\eta \mathbf{g} = \boldsymbol{\theta}^* = -\mathbf{H}^{-1}\mathbf{g}$. Given $\mathbf{H} = \lambda \mathbf{I}$, the gradient descent updates reaches to the minimum in a single step if $\eta \mathbf{g} = \frac{1}{\lambda}\mathbf{g}$, equivalently $\eta = \frac{1}{\lambda}$. □

A constant diagonal Hessian is too restrictive, and a more general result can be developed.

Theorem 4. *Suppose the quadratic problem*

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{H} \boldsymbol{\theta} + \mathbf{g}^\top \boldsymbol{\theta},$$

with a diagonal Hessian $\mathbf{H} = \text{diag}(h_{ii})$ *where* diag *produces a diagonal matrix with* $h_{ii} > 0$, *as their main diagonal elements. The optimal learning rate is*

$$\eta^* = \frac{\sum_{i=1}^n g_i^2}{\sum_{i=1}^n h_{ii} g_i^2},$$

where $\mathbf{g}$ *is the gradient vector, i.e.,*

$$\eta^* = \frac{\sum_{i=1}^n (\nabla_i \mathcal{L})^2}{\sum_{i=1}^n h_{ii} (\nabla_i \mathcal{L})^2}$$

Proof. The approach in Lemma 2 that matches the Newton update with the gradient descent is an under-specified vector equation, and cumbersome to solve. Therefore, we directly minimize the loss in a single update, i.e., minimizing $\hat{\mathcal{L}}(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t)$ in terms of η

$$\mathcal{G}(\eta) = \hat{\mathcal{L}}(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) = \frac{\eta^2}{2}\mathbf{g}^\top \mathbf{H} \mathbf{g} - \eta \mathbf{g}^\top \mathbf{g}.$$

With $\mathcal{G}$ a quadratic univariate function minimized at $\eta^* = \frac{\mathbf{g}^\top \mathbf{g}}{\mathbf{g}^\top \mathbf{H} \mathbf{g}}$. Given that $\mathbf{H}$ is diagonal, the optimal learning rate simplifies to

$$\eta^* = \frac{\sum_{i=1}^n g_i^2}{\sum_{i=1}^n h_{ii} g_i^2}.$$

□

In high-dimensional settings such as deep learning loss optimization, the Hessian is highly non-diagonal, and the learning rate is usually tuned in practice by running several experiments. It is more important to specify a range in a general case.

Theorem 5. *Assume the quadratic problem*

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \tfrac{1}{2}\boldsymbol{\theta}^\top \mathbf{H}\boldsymbol{\theta} + \mathbf{g}^\top\boldsymbol{\theta},$$

with a positive definite Hessian. The optimal learning lies within the range $\frac{1}{\lambda_{\max}} \leq \eta^* \leq \frac{1}{\lambda_{\min}}$ *where* $\lambda_{\max}, \lambda_{\min}$ *are the largest and smallest eigenvalues of the Hessian, respectively.*

Proof. The proof follows along the result of Theorem 4,

$$\eta^* = \frac{\mathbf{g}^\top\mathbf{g}}{\mathbf{g}^\top\mathbf{H}\mathbf{g}}.$$

For a non-vanishing gradient we aim to minimize and maximize η^* given a Hessian, i.e.,

$$\min_{\mathbf{g}} \frac{\mathbf{g}^\top\mathbf{g}}{\mathbf{g}^\top\mathbf{H}\mathbf{g}} \leq \eta^* \leq \max_{\mathbf{g}} \frac{\mathbf{g}^\top\mathbf{g}}{\mathbf{g}^\top\mathbf{H}\mathbf{g}}.$$

The Hessian $\mathbf{H}$ is positive-definite so has a eigenvalue eigenvector decomposition of the form $\mathbf{H} = \mathbf{P}\boldsymbol{\Lambda}\mathbf{P}^\top$, with an orthogonal $\mathbf{P}$, i.e., $\mathbf{P}\mathbf{P}^\top = \mathbf{I}$ and a diagonal $\boldsymbol{\Lambda} = \text{diag}\{\lambda_i > 0\}$. One may re-write the optimizing function as

$$\frac{\mathbf{g}^\top\mathbf{g}}{\mathbf{g}^\top\mathbf{H}\mathbf{g}} = \frac{\mathbf{g}^\top\mathbf{P}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{\Lambda}^{-1}\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{P}^\top\mathbf{g}}{\mathbf{g}^\top\mathbf{P}\boldsymbol{\Lambda}^{\frac{1}{2}}\Lambda^{\frac{1}{2}}\mathbf{P}^\top\mathbf{g}}.$$

By exchanging the optimization direction from $\mathbf{g}$ to $\mathbf{x} = \boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{P}^\top\mathbf{g}$ one may simplify the minimization to

$$\min_{\mathbf{g}} \frac{\mathbf{g}^\top\mathbf{g}}{\mathbf{g}^\top\mathbf{H}\mathbf{g}} = \min_{\mathbf{x}} \frac{\mathbf{x}^\top\boldsymbol{\Lambda}^{-1}\mathbf{x}}{\mathbf{x}^\top\mathbf{x}} = \min_{\mathbf{x}} \frac{\sum_{i=1}^n \frac{x_i^2}{\lambda_i}}{\sum_{i=1}^n x_i^2}.$$

However, a lower bound for $\sum_{i=1}^n \frac{x_i^2}{\lambda_i}$ is achieved by factorizing $\lambda_{\max}$, i.e.,

$$\frac{1}{\lambda_{\max}} \leq \frac{\sum_{i=1}^n \frac{x_i^2}{\lambda_i}}{\sum_{i=1}^n x_i^2} \quad \forall \mathbf{x},$$

and this is a tight lower bound in the sense that the lower bound is attained for $\mathbf{x} = \mathbf{e}_{\max}$ where $\mathbf{e}_{\max}$ is the eigenvector associated to the largest eignevalue of $\mathbf{H}$.

A similar analogy applies to $\max_{\mathbf{g}} \frac{\mathbf{g}^\top \mathbf{g}}{\mathbf{g}^\top \mathbf{H}\mathbf{g}}$ that yields

$$\frac{\sum_{i=1}^{n} \frac{x_i^2}{\lambda_i}}{\sum_{i=1}^{n} x_i^2} \leq \frac{1}{\lambda_{\min}} \quad \forall \mathbf{x},$$

and the proof is complete. □

In practice two avenues are taken, i) either several learning rates are tried and the loss values after training are compared, ii) a non-constant with a specific scheduling on training η_k is tried, and the learning rate is decayed towards the origin as the training progresses.

Gradient descent typically yields a steady and predictable reduction in error for many optimization problems, particularly when the objective function is smooth and nearly quadratic. In contrast, stochastic gradient descent (SGD), updates parameters using a randomly selected subset of data points, i.e., takes a gradient step on the noisy batched version of (4.1) by re-writing

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{b=1}^{B} \sum_{i=1}^{m_b} \mathcal{L}_{bi}(\boldsymbol{\theta}),$$

where m_b is a batch size, with the total sample size $m = \sum_{b=1}^{B} m_b$. Each step for a batch minimizes $\sum_{i=1}^{m_b} \mathcal{L}_{bi}(\boldsymbol{\theta})$ periodically until all data are fed and one epoch is completed. Deep learning models are often trained with 10 to 400 epochs.

Batched samples introduce variability into the convergence process. While this stochastic nature can help SGD escape local minima and potentially find better solutions, it also means that the convergence path is noisier and less predictable. As a result, SGD often converges more slowly in terms of the number of iterations compared to GD, see Figure 4.2. However, because each iteration of SGD is computationally cheaper (processing only a mini-batch of data), it is more efficient in practice, especially for large-scale problems where the full gradient computation is prohibitive, but run on massively parallel processors such as GPUs.

4.2.3 Zeroth Order

Optimization is a fundamental aspect of machine learning and artificial intelligence, playing a crucial role in model training and parameter tuning. Among the various optimization techniques, ZO and FO methods are widely used due to their distinct advantages and applications.

ZO optimization methods, such as grid search, do not require gradient information to find

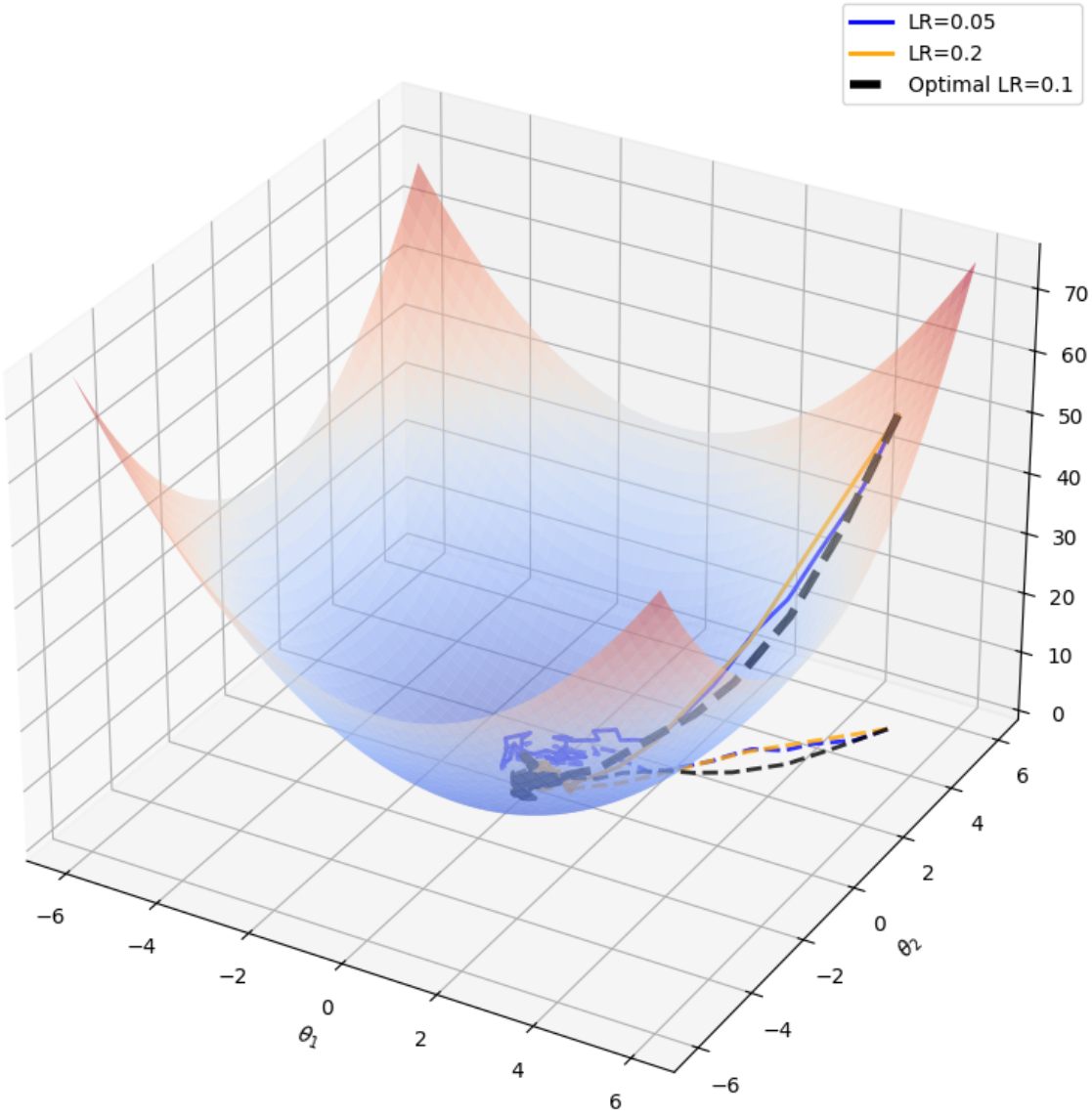

Figure 4.2 Stochastic gradient descent with different learning rates. The black solid refers to the optimal learning rate in stochastic setting.

the optimal parameters. Grid search, in particular, is a brute-force technique that evaluates a predefined set of hyperparameters to identify the best combination. This method is straightforward and easy to implement, making it a popular choice for hyperparameter tuning in scenarii where the objective function is complex or non-differentiable. However, grid search can be computationally expensive, especially as the dimensionality of the hyperparameter space increases. In contrast, FO optimization methods, such as gradient descent, leverage gradient information to iteratively update parameters in the direction that minimizes the objective function. These methods are generally more efficient than ZO methods, as they

can converge to the optimal solution faster by following the gradient. FO methods are particularly effective for large-scale optimization problems and are widely used in training deep neural networks.

We define the ZO gradient estimator, named Zeroth-Order SGD (ZO-SGD) as in [70]

$$\begin{aligned}\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}) &:= \sum_{i=1}^{q} \frac{\mathcal{L}(\boldsymbol{\theta}+\epsilon\mathbf{z}_i)-\mathcal{L}(\boldsymbol{\theta}-\epsilon\mathbf{z}_i)}{2\epsilon}\mathbf{z}_i,\\ \widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}) &\approx \frac{\partial\mathcal{L}(\boldsymbol{\theta};\mathbf{z})}{\partial\boldsymbol{\theta}} = \mathbf{z}^\top\nabla\mathcal{L}(\boldsymbol{\theta}),\end{aligned}$$

where the random variable $\mathbf{z}_i$ is a random vector sampled from a normal distribution $\mathbf{z}_i \sim \mathcal{N}(0,1)$. The infinitesimal constant ϵ is a small perturbation step size, also known as the *smoothing parameter*. After several tests, we claim that the value of this parameter has little impact on the models as long as $10^{-3} \leq \epsilon \leq 10^{-5}$. We selected the value 10^{-4} everywhere. The query budget q is half the number of times the loss is called to build $\widehat{\nabla}\mathcal{L}$.

As ϵ goes to zero and $q = 1$, the ZO estimator approaches the directional derivative of $\mathcal{L}$ at $\boldsymbol{\theta}$ along the direction $\mathbf{z}$. Note that since $\mathbb{E}_{\mathbf{z}}\left[\frac{\partial\mathcal{L}(\boldsymbol{\theta};\mathbf{z})}{\partial\boldsymbol{\theta}}\right] = \nabla\mathcal{L}(\boldsymbol{\theta})$, the ZO estimator $\widehat{\nabla}\mathcal{L}$ is an unbiased estimator of the FO gradient. This estimation improves as q increases. One expects to perform a more accurate ZO approximation towards FO with a higher query budget q. Although ZO methods are back-propagation free, they suffer from slow convergence. The time needed by ZO to reach the same accuracy as FO is roughly $\mathcal{O}(n)$ bigger, n being the problem dimension [73]. Most ZO methods have a variance in the range of $\mathcal{O}(nq^{-1})$, q being the number of queries to the loss [41]. This result also highlights a trade-off between the ZO gradient estimation and the query complexity. Table 4.1 sums up the different convergence rates of three optimization classes.

Table 4.1 For a given optimal solution θ^*, the convergence rate of the relative error varies. i) SO decreases the error quadratically with the iteration k near the optimum. ii) FO decreases linearly, with coefficient $r \in (0,1)$: r depends on the geometry and the conditioning of the objective function. The closer to zero, the better convergence rate. iii) Eventually, ZO decreases sub-linearly with coefficient $\alpha \in (0,1)$. The norm $\|\cdot\|$ indicates the Euclidean norm.

Relative Error	SO	FO	ZO
$\frac{\|\theta_{k+1}-\theta^*\|}{\|\theta_k-\theta^*\|}$	$\mathcal{O}(\|\theta_k-\theta^*\|)$	$\mathcal{O}(r)$	$\mathcal{O}\left(\frac{k}{k+1}\right)^{\alpha}$

We show that the high variance of ZO optimization could in fact be in favor of ZO methods

in the pre-training context. As Figure 4.3 suggests, the variance of ZO can be a convergence accelerator on a small problem. On the other hand, using without care can lead to disastrous results on a larger problem.

4.3 PRETRAINING

After positive results that show the behaviour of ZO-SGD is close to FO-SGD in the fine-tuning context, we explore the more complex pre-training task. Our goal is to find ways to pre-train LMs with ZO, which means to bring ZO close to FO in terms of final loss value. For this purpose, after understanding that dimension and variance are the key, we first try to reduce the dimension of the problem. Eventually, we address the ZO-variance issue by implementing two variance-reduction strategies and study their impact.

4.3.1 Vanilla Solver

The model trained is Llama2 with 20M parameters. This small model is built using the `config` file from HuggingFace. In order to remove as many degrees of freedom as we can, we train with a fixed learning rate, no weight decay, no momentum or Nesterov acceleration *vanilla* FO-SGD vs ZO-SGD. The training dataset is `cosmopedia-100k` [22] from HuggingFace. We run both experiments on two epochs on 8 V100 GPUs.

Table 4.2 Comparison of vanilla FO-SGD and ZO-SGD with two different query budget q for the pre-training task on Llama2-20M with 8 NVIDIA V100 GPUs.

Method	Loss (cross-entropy)	Train time (h)	Memory (GB)	Iteration time (s)
FO	8.02	2.5	21.2	0.66
$q = 1$	diverges	1.5	3.2	0.43
$q = 20$	8.17	30	3.3	7.14

As expected and shown in Table 4.2, the high dimension of the model leads ZO to behave poorly. For ZO-SGD to improve the loss, the minimal value is $q = 3$. With $q = 20$, the optimized ZO pre-training loss gets close to FO but the training time increases linearly with q while the progress in terms of loss are logarithmic with q. Though ZO can theoretically reach FO with very high budget q in the vanilla case, the high training time and the smaller memory savings make this solution undesirable. One way to improve ZO is to reduce the dimension of the problem.

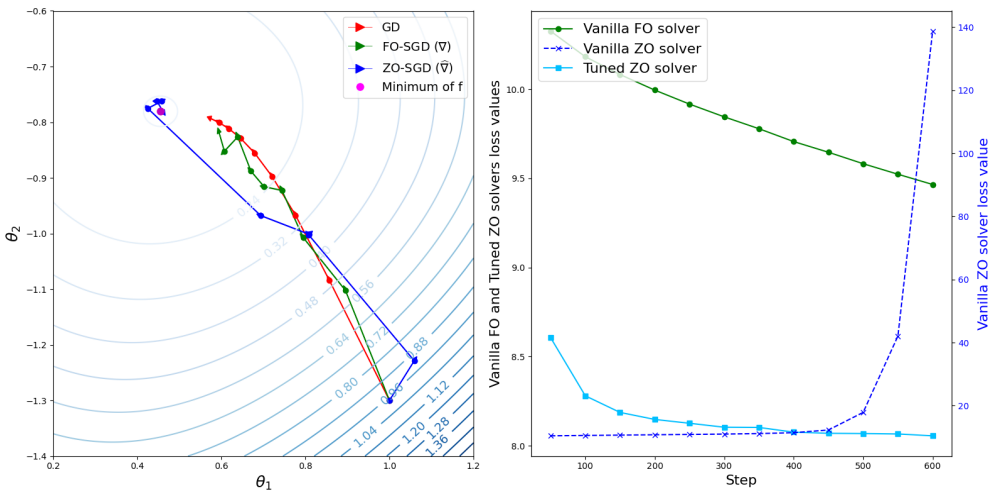

Figure 4.3 The behaviour of gradient descent (GD) in red, FO stochastic gradient descent (FO-SGD) in green, and its ZO approximation (ZO-SGD) in blue on a two-dimensional example. The optimal point is denoted by a pink blob (left panel). An example of pre-training of Llama2-20M with a ZO approximation. A vanilla FO solver in green (without momentum, learning-rate scheduling, any add-on to improve the solver), a vanilla ZO solver in dark blue which diverges, and a ZO solver on a smaller dimension with larger query budget q in light blue that eventually converges to the FO optimum loss (right panel).

4.3.2 Reduced Dimension

We try to improve the behaviour of ZO by reducing the dimension of the problem. Both FO-SGD and ZO-SGD were run under the same setup as Section 4.3.1. First, we load the FO-trained model from Section 4.3.1. Then, we freeze all the parameters of the model except the ones involved in the last MLP layer of Llama2-20M. After freezing, there are about 400k parameters remaining (2% of the initial model size). We reinitialize the unfrozen parameters, then compare FO-SGD versus ZO-SGD in terms of pre-training losses. Figure 4.5 shows a better behaviour of ZO compared to Section 4.3.1: with a higher query budget, ZO *can* reach FO on a smaller dimension problem. We suspect this is why ZO is efficient in fine-tuning. Following this observation, we propose a strategy for pre-training LMs. At the first step update only a part of the model, say *Block 1* with ZO, train for a few epochs until convergence. At the second step, re-run the ZO pre-training on another block, say *Block 2*, and so on until the whole model is trained. Details of our method can be found in Section 4.4.

On Figure 4.4, FO-variance is roughly 1 000 times smaller than ZO-variance with $q = 1$ and 100 times smaller for $q = 20$. As expected, one way to reduce the variance is to increase the budget q but this has a crucial impact on the training time. Training Llama2-20M on just 2 epochs with $q = 100$ takes around 6 days on our setup, versus a couple hours with FO-SGD,

so increasing q is not an option. We wonder if high-variance really is a flaw for pre-training LMs with ZO or is there a significant impact of this variance on the optimized pre-training loss. Table 4.3 shows that a smaller batch size (therefore a higher variance) has a positive impact on the loss. Following this observation, we implement a tunable variance-reduction strategy and observe the same effect there.

4.3.3 Variance Manipulation

We use the setup of Section 4.3.2 and consider the specific value $q = 2$. The gradient estimator can then be expressed as

$$\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}) = X_1 + X_2 \tag{4.5}$$
$$X_1 := \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon\mathbf{z}_1) - \mathcal{L}(\boldsymbol{\theta} - \epsilon\mathbf{z}_1)}{2\epsilon}\mathbf{z}_1 \quad ,$$
$$X_2 := \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon\mathbf{z}_2) - \mathcal{L}(\boldsymbol{\theta} - \epsilon\mathbf{z}_2)}{2\epsilon}\mathbf{z}_2.$$

Since $\mathbf{z}_1$ and $\mathbf{z}_2$ are independent, we can build X_2 to be negatively correlated to X_1. This strategy is a common trick in Monte-Carlo simulation [85] and leads the variance of $X_1 + X_2$ to be smaller than it should be if $\mathbf{z}_1$ and $\mathbf{z}_2$ were sampled independently.

For this purpose, at each step of the training, we build X_2 the following way:

Algorithm 11 Building X_2 negatively correlated to X_1

Input: $\alpha \in (-1, 1)$

1: Sample $\mathbf{z}_1$ and $\mathbf{z}_0$ from a normal distribution of mean 0 and variance 1
2: Define $\mathbf{z}_2 := \alpha\mathbf{z}_1 + \sqrt{1-\alpha^2}\mathbf{z}_0$
3: Compute $X_1 := \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon\mathbf{z}_1) - \mathcal{L}(\boldsymbol{\theta} - \epsilon\mathbf{z}_1)}{2\epsilon}\mathbf{z}_1$ and $X_2 = \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon\mathbf{z}_2) - \mathcal{L}(\boldsymbol{\theta} - \epsilon\mathbf{z}_2)}{2\epsilon}\mathbf{z}_2$ as in (4.5)

The parameter α tunes the correlation between $\mathbf{z}_1$ and $\mathbf{z}_2$: $\alpha = -1$ means $\mathbf{z}_1$ and $\mathbf{z}_2$ are perfectly negatively correlated, $\alpha = 0$ means $\mathbf{z}_1$ and $\mathbf{z}_2$ are uncorrelated (which is the case by default when sampling $\mathbf{z}_1$ and $\mathbf{z}_2$ independently), $\alpha = 1$ means $\mathbf{z}_1$ and $\mathbf{z}_2$ are perfectly positively correlated.

Regarding Table 4.3, we expect the loss to decrease as α goes from -1 to zero. Table 4.4 shows that this is indeed the case.

Table 4.3 ZO-SGD with $q = 1$ and varying batch size while pre-training LLama2-20M with reduced dimension.

Batch size	Loss (cross-entropy)	Train time (h)	Memory (GB)	Variance $\times 10^{-2}$
2	9.953	1.2	1.6	26.1
4	9.958	1.3	3.1	8.9
8	9.981	1.5	6.1	7.8

Table 4.4 Comparison of the gradients distribution with variance manipulation. The optimal loss value is **bold**.

α	Loss (cross-entropy)	Mean $\times 10^{-6}$	Variance $\times 10^{-2}$
0.0	**7.981**	31.3	9.8
-0.5	8.000	-11.1	8.7
-0.9	8.016	9.4	1.5

4.4 EXPERIMENTAL DETAILS

Figure 4.4 gives the gradients density related to the pre-training of Llama2-20M pre-trained on `cosmopedia-100k` data, FO versus ZO after one epoch of 6K steps. The gradients values of FO are more concentrated than ZO. A higher query budget q reduces the variance and the mean of ZO, still not reaching FO.

Figure 4.5 shows the pre-training loss trace plot of Llama2-20M model on `cosmopedia-100k` text with different query budget on a reduced dimension model. For this experiment, only the last MLP layer of Llama2-20M is trained, which corresponds to roughly 500K parameters. Figure 4.5 confirms that ZO is close to FO with $q = 1$ (8.02 for ZO and 7.98 for FO). Augmenting q allows ZO to reach FO but the training time increases linearly with q.

Figure 4.6 shows an extension of Figure 4.5 in pre-training Llama2-20M layer-by-layer over 600K parameters. When training a layer, the whole model but a single layer is frozen and this specific layer is reinitialized and retrained. Once this block training is performed, the trained layer is frozen, and the next layer is unfrozen and reinitialized. The first block is the embedding layer that includes 10M parameters, and is trained with vanilla FO in two epochs. The second block is trained for one epoch with vanilla FO versus vanilla ZO, and so on. Figure 4.5 shows that ZO can reach FO on smaller parameter setting. Moreover, this result offers ways to improve ZO for pre-training, for instance adding a momentum that suits ZO.

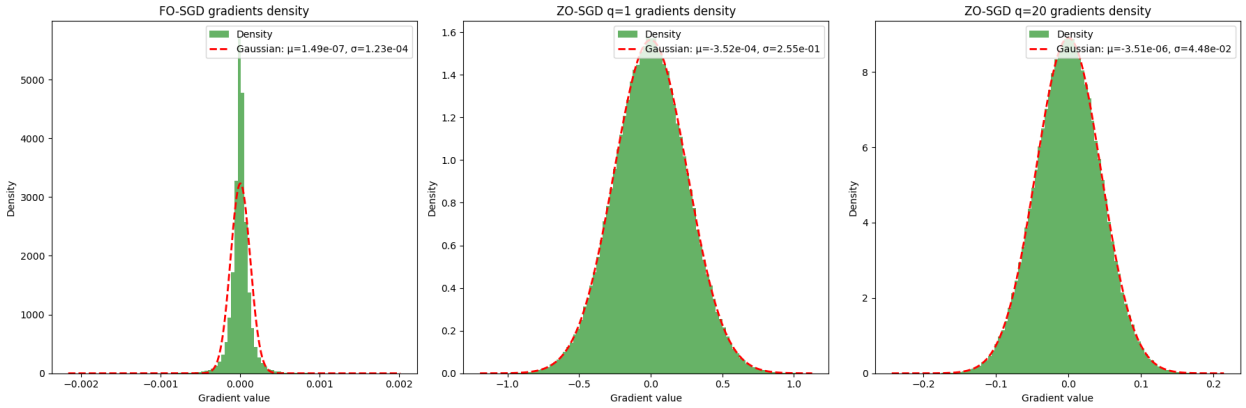

Figure 4.4 The gradient density is shown in green. The closest normal distribution is shown in red dashed curve. The mean and the variance values are mentioned in the legend. The mean and the variance for FO are lower than ZO. As q increases, both mean and variance decrease.

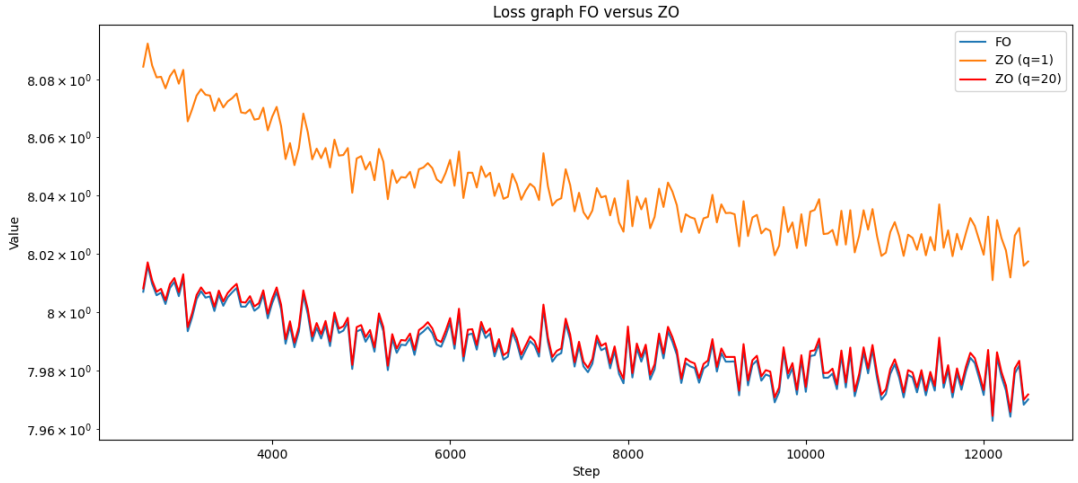

Figure 4.5 The loss curve of FO-SGD is shown in blue. Curves for ZO-SGD are in orange ($q = 1$) and red ($q = 20$). Increasing the query budget has a positive impact on the optimized training loss. With high enough budget, ZO can reach FO.

4.5 DISCUSSION AND FUTURE WORK

In this work, we followed several new avenues for pre-training LMs with ZO, such as variance reduction and working on a reduced dimension problem. Low memory cost is a key aspect of ZO training. Its use could be very relevant in applications such as on-device training or in situations where memory is limited. By reducing memory requirements, ZO training allows larger models to be trained on the same hardware at the cost of longer training times.

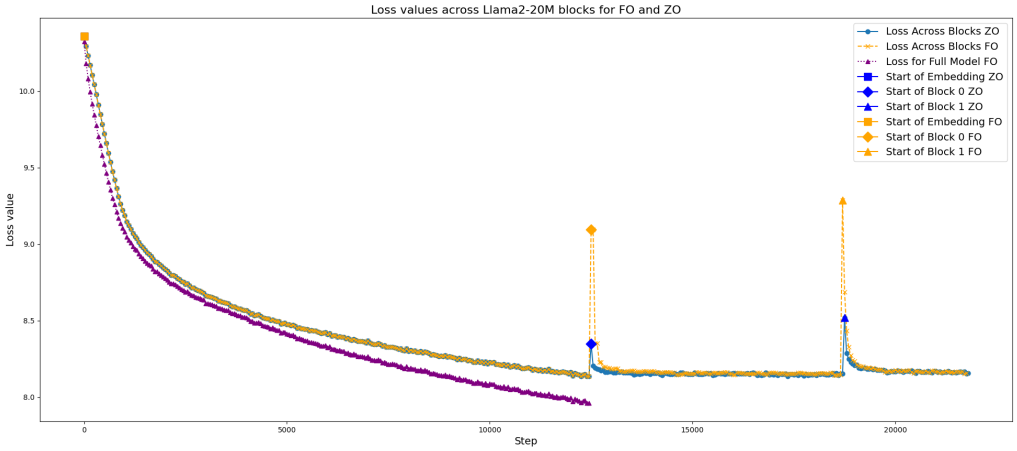

Figure 4.6 Layer-wise training of Llama2-20M for vanilla FO (yellow) versus vanilla ZO (blue). Training the full size model with vanilla FO is shown in purple.

Future investigations include scaling up blockwise ZO pre-training to larger models, on small chunks with a moderate query budget q. The training time will be longer than for FO, but depending on the goal, the memory savings may compensate. Ultimately, this work focused on untuned ZO training. Features such as adding momentum could improve its efficiency and robustness to scaling.

4.6 Conclusion

This exploratory work unveiled some recently unknown behaviour of ZO optimization in pre-training LMs. We established the connection between SO and FO by studying the optimal learning rate. We also provided a recipe for a successful application of ZO in pretraining. First reducing the dimension of the problem leads to the success of ZO to pre-train LMs in the sense that vanilla ZO converges to vanilla FO. Second, the high variance of ZO is not a disadvantage as it is often thought in the community, but rather is an asset during the pre-training. As a consequence, artificially reducing the variance leads to a higher loss value. We proposed to pre-train LMs using ZO optimization on a reduced dimension space like blocks of parameters, because it is the key to a successful pre-training.

CHAPTER 5 ARTICLE 2: ZEROTH ORDER KRONECKER OPTIMIZATION FOR PRETRAINING LANGUAGE MODELS

Nathan Allaire and Sébastien Le Digabel and Dominique Orban and Vahid Partovi Nia

Manuscript submitted on September 8, 2025, to *Springer Nature Computer Science* (SNCS); currently under review.

Abstract

Training language models (LMs) under tight GPU memory budgets rules out standard back-propagation and motivates *zeroth-order* (ZO) optimization. While ZO methods have proven effective for fine-tuning, their potential during the more memory-intensive pretraining stage has received little attention. We first revisit the singular-value spectra of layer gradients during pretraining and show that the gradient information is spread across many directions; low-rank ZO methods therefore potentially discard some informative components. Building on this insight, we introduce **KronZO**, a Kronecker-structured ZO optimizer that (i) explores a full-rank search subspace with state-of-the-art storage compression and (ii) employs a criterion-driven *directional* update that selectively keeps only informative steps. When pretraining GPT-2 Small from scratch on OpenWebText, KronZO achieves a markedly lower training loss than all previous ZO baselines while consuming less GPU memory. Although still trailing first-order methods in final loss, KronZO substantially narrows the gap at a fraction of their memory footprint, extending ZO optimization to larger models and longer runs and paving the way for memory-efficient pretraining on commodity hardware.

5.1 Introduction

Recent scaling laws demonstrate that, when trained with the same optimization method and data, a larger LM (i.e., with more parameters) achieves a lower training loss and better downstream accuracy than a smaller LM [56].

Consequently, the number of parameters exploded (from tens of millions in BERT-base [96] to hundreds of billions in modern LMs) while commodity GPU memory has risen from 16GB to only 80GB over the same period. The dominant memory bottleneck during training is *not* due to the model weights themselves, but to the activation tensors retained for back-propagation, along with optimizer-specific states, which together can increase the overall

memory footprint of training by up to 12 times on modern LMs compared to inference [70]. Even *parameter-efficient fine-tuning* (PEFT) techniques, such as *Low-Rank Adapters* like LoRA [54], leave these costs unchanged because they still rely on FO optimization (SGD [7], Adam [59]) and gradient computation in the backward pass.

Hardware-level approaches (model compression, quantization, pruning, knowledge distillation) alleviate inference cost but do not eliminate the back-propagation memory overhead during training. ZO algorithms provide an alternate solution: they approximate gradients from forward evaluations alone, bypass back-propagation entirely and eliminate activation storage for gradient computation [90]. Classical variants such as ZO-SGD [45] have matured theoretically [41, 73] and were recently adapted to LM fine-tuning through the *Memory-Efficient Zeroth-Order* (MeZO) algorithm [70]. Follow-up work introduced forward-gradient refinements [105] and the *Low-Rank Zeroth-Order* (LoZO) method [31], which exploits gradient low-rankness in fine-tuning to improve performance and reduce memory.

In contrast, ZO *pretraining* remains largely unexplored. Allaire et al. [4] provided first evidence that ZO *can* pretrain small LMs, but plain ZO-SGD falls short at larger scales. Building on this preliminary study, the present work scales ZO pretraining to models an order of magnitude larger and introduces **KronZO**, a new ZO method that

- exploits a Kronecker factorization to craft memory-efficient perturbations, markedly reducing the storage budget relative to state-of-the-art methods;
- employs high-dimensional perturbations to ensure broad exploration of the complex objective landscape;
- introduces a novel, criterion-driven *directional update* that stabilizes and improves convergence speed in highly stochastic regimes.

Our numerical experiments show that KronZO markedly outperforms prior ZO baselines during pretraining, achieving a substantially lower loss on GPT-2 Small with significantly reduced GPU memory consumption. Despite relying solely on forward evaluations, KronZO narrows the gap with FO optimizers while enabling training on hardware configurations where back-propagation is infeasible. Furthermore, downstream evaluation on the GLUE benchmark confirms that models pretrained with KronZO retain strong generalization capabilities, matching the transfer performance of SGD across diverse NLP tasks.

Related work

ZO optimization, or derivative-free optimization, addresses settings where gradients are unavailable, unreliable, or prohibitively expensive [13]. In machine learning, ZO algorithms estimate gradients from finite-difference queries [26, 90], and subsequently employ optimization methods based on these estimates. They have proved valuable for adversarial attacks [55, 95, 106], model interpretability [39], hyper-parameter search [60, 94], and more recently, LM training [4, 31, 70]. This field is currently under active research [103, 105].

Fine-tuning has become the de facto approach to adapt LMs for specific downstream applications [49]. Full fine-tuning has the same memory cost as regular pretraining since the number of trainable parameters is unchanged. Although PEFT reduces the number of trainable parameters, full back-propagation still dominates memory use. MeZO [70] showed that ZO methods can finetune LMs with accuracy comparable to FO methods while using significantly less memory; LoZO [31] further leverages gradient structure in fine-tuning to surpass both MeZO and LoRA in terms of memory efficiency and convergence performance on fine-tuning tasks. Building on [31, 70], we analyze their mechanisms and limitations in Section 5.2.2, which motivates our proposed KronZO method.

Pretraining poses additional challenges: extreme dimensionality, sharper non-convexity, batch-dependent noise. To the best of our knowledge, beyond the first study of Allaire et al. [4], ZO pretraining has not been explored in the literature. KronZO fills this gap, coupling Kronecker perturbations with innovative ZO updates to push the frontier of scalable, memory-efficient LM pretraining.

Notation

Let L be the total number of layers in an LM. Matrices in layer l have m^l rows and n^l columns, $l \in \{1, 2, \ldots, L\}$. The optimization proceeds over K iterations, indexed by k. Matrices are denoted $A^l, B^l, U^l, V^l, Z^l, \theta^l$, while sets of such matrices are written in bold, e.g., $\mathbf{Z} = \{Z^l\}_{l=1}^L$. The set of matrices $\boldsymbol{\theta} = \{\theta^l\}_{l=1}^L$ where $\theta^l \in \mathbb{R}^{m^l \times n^l}$ corresponds to the model parameters, or weights in layer l. The total number of parameters is $d = \sum_{l=1}^L m^l n^l$. For convenience, we occasionally treat matrices or sets of matrices as column vectors, e.g., $\mathbf{Z} \in \mathbb{R}^d$. The loss function to be minimized is denoted $\mathcal{L}$. Let ξ be a random variable representing the stochasticity in batch selection, and let ξ_k denote its realization at iteration k, i.e., the mini-batch sampled at iteration k. Finally, the cardinality of a set Ω is denoted $|\Omega|$.

We consider the optimization problem

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}) \quad \text{where} \quad \mathcal{L}(\boldsymbol{\theta}) := \mathbb{E}_{\xi}[\mathcal{L}(\boldsymbol{\theta};\xi)]\,. \tag{5.1}$$

In an LM context, we assume that $\mathcal{L}$ is differentiable and has Lipchitz-continuous gradient.

5.2 Background

In this section, we review the concepts relevant to our methodology. We first provide an overview of FO optimization. We then discuss existing ZO methods, notably MeZO [70] and LoZO [31].

5.2.1 First-order optimization

The classic way to solve (5.1) is to use the stochastic gradient method

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta};\xi) \tag{5.2}$$

where $\nabla\mathcal{L}(\boldsymbol{\theta};\xi)$ is the stochastic gradient of the loss with respect to $\boldsymbol{\theta}$ and $\alpha > 0$ is the step size, often referred to as the *learning rate*. At iteration k, a mini batch ξ_k is sampled from ξ, a forward pass gives the value $\mathcal{L}(\boldsymbol{\theta}_k;\xi_k)$, and the back-propagation computes $\nabla\mathcal{L}(\boldsymbol{\theta}_k;\xi_k)$ using the chain rule.

Back-propagation consumes more GPU memory than the forward pass because it must retain every layer output (activation) and store a gradient tensor for each weight matrix. For a batch of size $|\xi_k|$ and L layers, the principal memory terms are

$$\underbrace{\mathcal{M}_{\text{act}} = |\xi_k| \sum_{l=1}^{L} n^l}_{\text{activations}}, \qquad \underbrace{\mathcal{M}_{\text{grad}} = \sum_{l=1}^{L} m^l n^l}_{\text{weight gradients}}\,.$$

Most FO optimizers keep $s_{\text{FO}} \in \{0,1,2\}$ auxiliary buffers per parameter (e.g., momentum for SGD, first and second moments for AdamW [69]). Including the weights themselves, the memory consumed by the optimizer state is

$$\mathcal{M}_{\text{opt,FO}} = (1 + s_{\text{FO}}) \sum_{l=1}^{L} m^l n^l.$$

Hence the *peak FO footprint* is

$$\mathcal{M}_{\text{FO}} = |\xi_k| \sum_{l=1}^{L} n^l + (2 + s_{\text{FO}}) \sum_{l=1}^{L} m^l n^l. \tag{5.3}$$

5.2.2 Zeroth-Order optimization

ZO methods do not back-propagate the gradients, so they require neither $\mathcal{M}_{\text{act}}$ nor $\mathcal{M}_{\text{grad}}$. They only store the weights and a method-specific state (see sets of matrices $\mathbf{A}$ and $\mathbf{B}$ for KronZO in Section 5.4) whose size is at most a fraction $s_{\text{ZO}} \leq 1$ of the parameter count:

$$\mathcal{M}_{\text{ZO}} = (1 + s_{\text{ZO}}) \sum_{l=1}^{L} m^l n^l, \qquad s_{\text{ZO}} \leq 1. \tag{5.4}$$

Malladi et al. [70] report that training OPT-13B requires 12 times more memory than inference, a ratio consistent with (5.3) and with our own measurements, see Figure 5.6.

To sidestep back-propagation and its significant memory overhead, we turn to a ZO paradigm that relies exclusively on loss evaluations.

Finite differences

The most straightforward ZO technique is to estimate directional derivatives by *finite differences* [45], namely the Random Gradient Estimator (RGE)

$$\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}; \xi) := \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon \mathbf{Z}; \xi) - \mathcal{L}(\boldsymbol{\theta} - \epsilon \mathbf{Z}; \xi)}{2\epsilon} \mathbf{Z}, \tag{5.5}$$

where $\mathbf{Z} \sim \mathcal{N}(0, \mathbf{I}_d) \in \mathbb{R}^d$, and $\epsilon > 0$ is a small perturbation step size called the *smoothing parameter*. In order to reduce the variance inherent to RGE (5.5), it can be interesting to average the estimator over q randomly sampled directions $\mathbf{Z}_i$ with q-RGE

$$\widehat{\nabla}_q\mathcal{L}(\boldsymbol{\theta}; \xi) := \frac{1}{q} \sum_{i=1}^{q} \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon \mathbf{Z}_i; \xi) - \mathcal{L}(\boldsymbol{\theta} - \epsilon \mathbf{Z}_i; \xi)}{2\epsilon} \mathbf{Z}_i, \tag{5.6}$$

where each $\mathbf{Z}_i \sim \mathcal{N}(0, \mathbf{I}_d) \in \mathbb{R}^d$ and q is the *query budget*.

When $q = 1$, (5.6) reduces to (5.5). Since $\lim_{\epsilon \to 0} \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon \mathbf{Z}; \xi) - \mathcal{L}(\boldsymbol{\theta} - \epsilon \mathbf{Z}; \xi)}{2\epsilon} = \mathbf{Z}^\top \nabla \mathcal{L}(\boldsymbol{\theta}; \xi)$ in vectorized notation, (5.5) can be seen as the projection of the stochastic gradient on $\mathbf{Z}$. Since $\mathbf{Z}$

comes from a standard normal distribution, (5.5) and (5.6) are non-biased estimators of the stochastic gradient:

$$\mathbb{E}[\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta};\xi)] = \mathbb{E}[\widehat{\nabla}_q\mathcal{L}(\boldsymbol{\theta};\xi)] = \nabla\mathcal{L}(\boldsymbol{\theta};\xi).$$

At each iteration, the parameters are updated similarly to (5.2):

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}_k;\xi_k). \tag{5.7}$$

Malladi et al. [70] use (5.5) and (5.6) to finetune LMs with MeZO. For convenience, we also introduce the layer-wise notation of (5.5) and (5.6)

$$\widehat{\nabla}\mathcal{L}(\theta^l;\xi) := \frac{\mathcal{L}(\theta^l + \epsilon Z^l;\xi) - \mathcal{L}(\theta^l - \epsilon Z^l;\xi)}{2\epsilon} Z^l, \qquad l = 1, 2, \ldots, L \tag{5.8}$$

$$\widehat{\nabla}_q\mathcal{L}(\theta^l;\xi) := \frac{1}{q}\sum_{i=1}^{q} \frac{\mathcal{L}(\theta^l + \epsilon Z_i^l;\xi) - \mathcal{L}(\theta^l - \epsilon Z_i^l;\xi)}{2\epsilon} Z_i^l, \qquad l = 1, 2, \ldots, L \tag{5.9}$$

where $\theta^l \in \mathbb{R}^{m^l \times n^l}$ are the layer parameters. Thus Z^l and $Z_i^l \in \mathbb{R}^{m^l \times n^l}$. Without additional effort for storing Z^l efficiently, the memory required to run RGE or q-RGE is $\mathcal{O}(m^l n^l)$ for each layer l, which corresponds to $s_{\text{ZO}} = 1$ in (5.4). In MeZO, only the seed to generate Z^l is stored, and it can be regenerated from this seed *on-the-fly* during computations (thus $s_{\text{ZO}} \ll 1$). Despite reducing the memory costs, the regeneration process creates additional floating points operations.

Low-rank ZO

In MeZO, $Z^l \in \mathbb{R}^{m^l \times n^l}$ very likely has full rank, i.e., $\text{rank}(Z^l) \approx \min\{m^l, n^l\}$. Based on the observation that gradients exhibit a low-rank structure during fine-tuning, Chen et al. [31] introduced LoZO to align the structure of the stochastic gradient estimator with that of the actual stochastic gradient. In their work, $Z_i^l = U_i^l (V^l)^\top$, where $U^l \in \mathbb{R}^{m^l \times r}$ and $V^l \in \mathbb{R}^{n^l \times r}$ with $0 < r \ll \min\{m^l, n^l\}$, both sampled independently from a standard Gaussian distribution. The layer-wise gradient estimator is

$$\widehat{\nabla}_q^{\text{LoZO}}\mathcal{L}(\theta^l;\xi) := \sum_{i=1}^{q} \frac{\mathcal{L}(\theta^l + \epsilon U_i^l (V^l)^\top;\xi) - \mathcal{L}(\theta^l - \epsilon U_i^l (V^l)^\top;\xi)}{2\epsilon} \frac{U_i^l (V^l)^\top}{r}. \tag{5.10}$$

The scaling $1/r$ in (5.10) makes the gradient estimator unbiased. The matrix U_i^l is sampled q times at every iteration for each layer, while V^l is refreshed once every ν iteration ($\nu \in \mathbb{N}_{>0}$) for each layer. This way, (5.10) explores the subspace defined by V^l for ν steps. Since $\text{rank}(U^l) \approx r$ and $\text{rank}(V^l) \approx r$, $\text{rank}(U^l V^{l\top}) \approx r$, and the perturbation has low rank.

LoZO requires storing all U_i^l and V^l for each layer, for a memory cost of $\mathcal{O}(m^l r + n^l r)$. With a typical attention layer size of $784 \times 2{,}304$ (GPT-2 Small, BeRT-base), MeZO needs to store roughly 2×10^6 additional floats per layer, while LoZO only needs to store $784 \times r + 2{,}304 \times r$ floats, which is about 10^4 with the typical value $r = 4$. LoZO showed state-of-the-art fine-tuning efficiency on a large family of tasks and LMs.

In the next section, we analyze the singular values of both finetuning and pretraining gradients and show that they differ significantly, which motivates our design choice in KronZO to use full-rank perturbations, following the approach of MeZO.

5.3 Pretraining gradients analysis

Studies on *intrinsic dimension* in LMs [62, 66] show that stochastic gradients collapse into a low-dimensional subspace during fine-tuning, making low-rank ZO optimizers such as LoZO highly effective. These findings rely on analyzing the *singular-value spectrum* (the ordered set of singular values) of each layer's gradient matrix, which in fine-tuning is sharply peaked and therefore concentrates most of the information in only a few directions. Figure 5.1 confirms this behavior on GPT-2: as fine-tuning proceeds on the Shakespeare dataset, more singular values shrink while a handful remain large, signaling an increasingly low-rank structure.

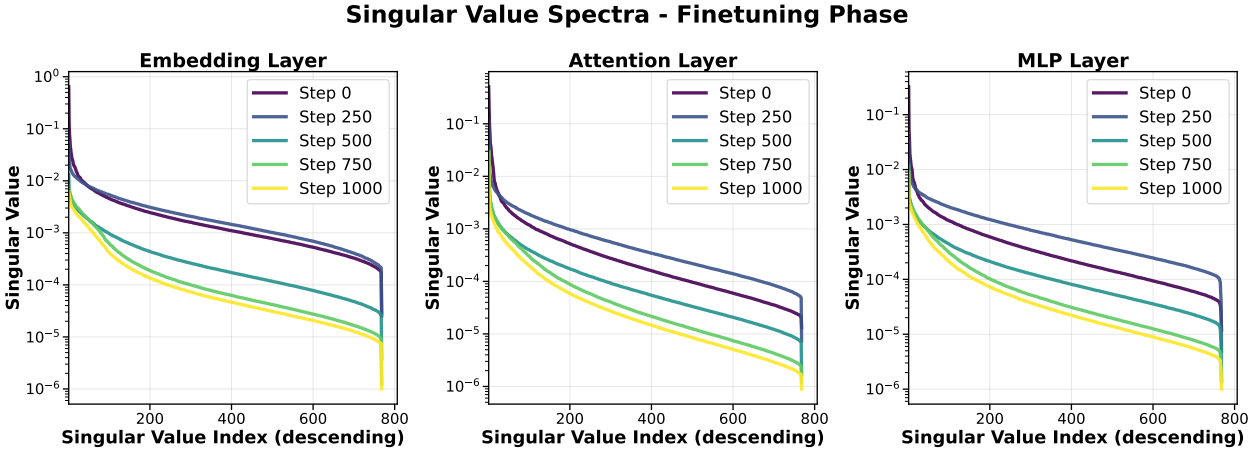

Figure 5.1 Evolution of stochastic gradients spectra during GPT-2 fine-tuning on the Shakespeare dataset. At early steps, the purple and dark-blue curves are relatively flat, with several high singular values. At step 1,000, the yellow curve still retains a handful of high singular values but drops off sharply: information concentrates rapidly in a few directions, evidencing low-rank collapse.

By contrast, Figure 5.2 shows that during *pretraining* the spectrum remains comparatively flat: although a handful of directions still carry the largest singular values, many others

exhibit mid-range magnitudes, indicating that information is spread across a wide set of modes.

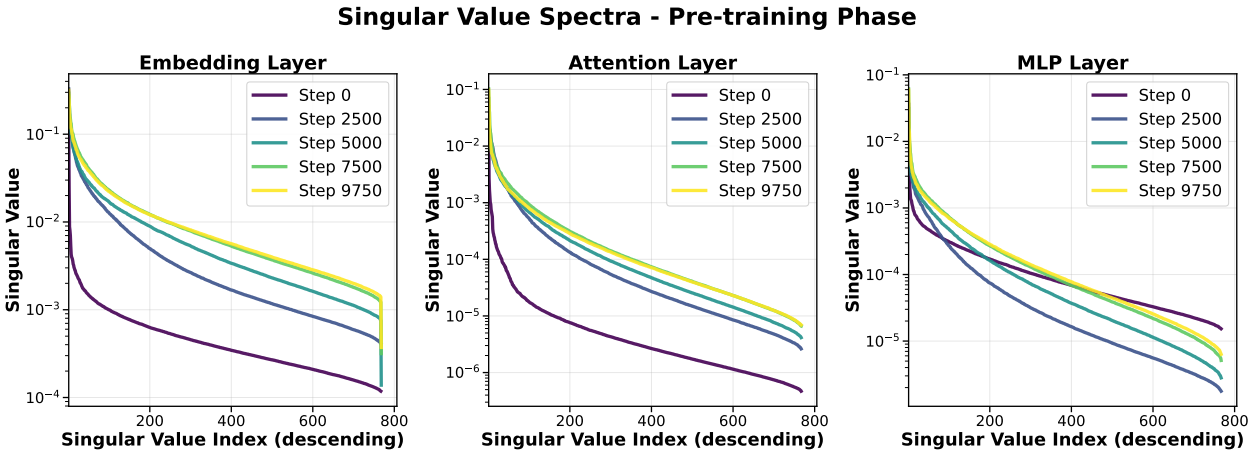

Figure 5.2 Evolution of stochastic gradients spectra in GPT-2 pretraining on the Shakespeare dataset. During early steps, the purple and dark-blue curves show a very sharp descent. At final steps, the yellow curve is flatter, indicating a broader spectrum.

This qualitative gap suggests that low-rank ZO methods, while effective for fine-tuning, may not be suitable for pretraining. In such settings, full-rank ZO methods are more appropriate, as supported by our results in Section 5.6. Since almost no singular value is exactly zero, the algebraic rank remains nearly full at each step. To gauge the true dimensionality of pretraining gradients, we turn to *effective-rank* measures instead. Next, we compare the widely used *stable rank* with the *participation ratio* (PR) and show that the former can underestimate dimensionality, whereas the latter more accurately captures the richness of LM stochastic gradients.

5.3.1 Stable rank

For a matrix $A \in \mathbb{R}^{m\times n}$ with singular values $\{\sigma_i\}_{i=1}^{\mu}$, where $\mu = \min(m,n)$ the stable rank is

$$\operatorname{srank}(A) \;=\; \frac{\|A\|_F^2}{\|A\|_2^2} \;=\; \frac{\sum_{i=1}^{\mu}\sigma_i^2}{\max_{i=1}^{\mu}\sigma_i^2}. \tag{5.11}$$

Throughout this section, we call $\lambda_i = \sigma_i^2$ the *energy* of singular direction i, and $(\lambda_1, \ldots, \lambda_\mu)$ the energy vector with total energy $\sum_{i=1}^{\mu}\lambda_i$.

Although continuous, the stable rank is driven by the *largest* singular value and can mask numerous medium-energy directions. Take $D = \operatorname{diag}\big(100, 60, \ldots, 60\big)$ where 60 is repeated 19 times; thus $\operatorname{rank}(D) = 20$. D has a structure similar to pretraining gradients: one

high singular value, and many medium-sized. Its energy vector is $\boldsymbol{\lambda} = (100^2, 19 \times 60^2)$ and $\sum_{i=1}^{\mu} \lambda_i = 78\,400$. Nineteen medium directions therefore carry $68\,400/78\,400 \approx 87\%$ of the energy, yet

$$\operatorname{srank}(D) = \frac{10\,000 + 19 \times 3\,600}{10\,000} \approx 7.8,$$

i.e., only 40% of the true rank. Figure 5.3 echoes this behavior: the stable-rank curve of GPT-2 gradients suggests an overly optimistic low rank during pretraining.

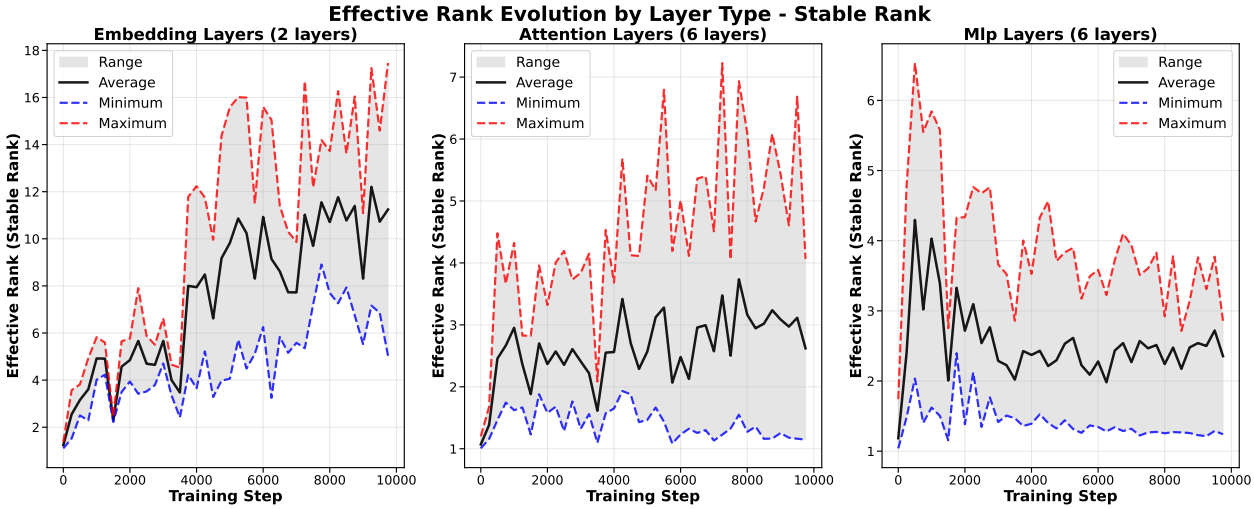

Figure 5.3 Stable rank (5.11) of GPT-2 gradients during pretraining. Because it tracks the largest singular values, it indicates a misleadingly low dimensionality across layer types.

5.3.2 Participation ratio

To capture all energetic directions, we adopt the *participation ratio* (PR) [43], defined on $\lambda_i = \sigma_i^2$ for $i = 1, 2, \ldots, \mu$:

$$\operatorname{pr}(A) = \frac{\left(\sum_{i=1}^{\mu} \lambda_i\right)^2}{\sum_{i=1}^{\mu} \lambda_i^2} = \frac{\left(\sum_{i=1}^{\mu} \sigma_i^2\right)^2}{\sum_{i=1}^{\mu} \sigma_i^4}. \tag{5.12}$$

As the inverse Herfindahl index [51], the PR estimates *how many singular directions carry non-negligible energy*. Apply (5.12) to matrix D from the previous section,

$$\operatorname{pr}(D) = \frac{(10\,000 + 19 \times 3\,600)^2}{10\,000^2 + 19 \times 3\,600^2} \approx 17.8,$$

far closer to the algebraic rank of 20 and more than double the stable-rank value. Figure 5.4 confirms this trend in real gradients: during pretraining, the PR rises to roughly 30% of the

layer width.

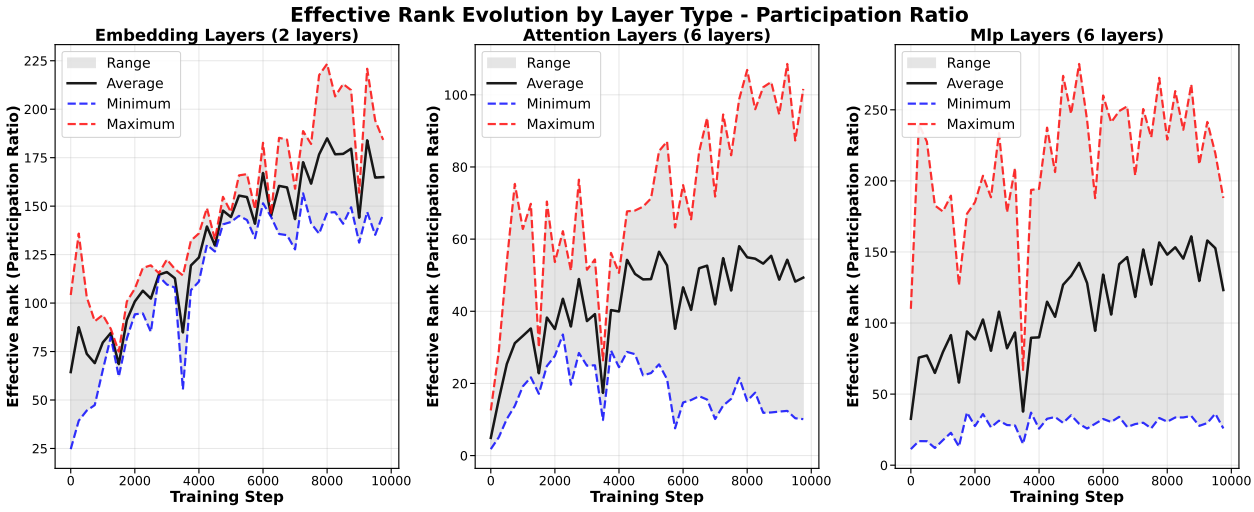

Figure 5.4 Participation ratio (5.12) of the same gradients used in Figure 5.3. Many more directions contribute significant energy, which contradicts the apparent low-rank structure suggested by the stable rank.

Because the PR remains sensitive to medium-sized singular values, it delivers a much more faithful effective rank for LM pretraining gradients, whose spectrum is broad. Although each subplot is noisy and not strictly monotonic, the overall upward trend of both the stable-rank and participation-ratio in Figures 5.3 and 5.4 shows that pretraining stochastic gradients do *not* have low-rank; they begin at a moderate rank and increase steadily as training progresses.

Consequently, ZO methods that limit updates to a tiny subspace (e.g., LoZO) are ill-suited to pretraining, whereas full-rank schemes such as MeZO are better aligned with the true dimensionality. This observation is consistent with the results reported in Section 5.6, where full-rank methods outperform their low-rank counterparts in pretraining. However, MeZO incurs a significant overhead: it must either store the entire matrix $\mathbf{Z}$ or regenerate it from the random seed at the cost of extra flops, an impractical choice for long-running tasks such as pretraining.

To cope with the broad pretraining subspace while keeping memory use and flops manageable, we turn to KronZO, a *full-rank* ZO optimizer that retains all informative directions through a lightweight Kronecker parameterization.

5.4 ZO Kronecker optimization (KronZO)

In this section, we present KronZO, our proposed ZO Kronecker optimization method for pretraining LMs. Recall that the Kronecker product of

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n_1} \\ a_{21} & a_{22} & \cdots & a_{1n_2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_1 1} & a_{m_1 2} & \cdots & a_{m_1 n_1} \end{bmatrix} \in \mathbb{R}^{m_1 \times n_1} \text{ and } B \in \mathbb{R}^{m_2 \times n_2} \text{ is defined as}$$

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n_1}B \\ a_{21}B & a_{22}B & \cdots & a_{1n_1}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_1 1}B & a_{m_1 2}B & \cdots & a_{m_1 n_1}B \end{bmatrix} \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}.$$

The Kronecker product possesses several useful algebraic properties; see [47] for a comprehensive overview. In this work, we focus on its rank-preserving property:

$$\operatorname{rank}(A \otimes B) = \operatorname{rank}(A) \times \operatorname{rank}(B). \tag{5.13}$$

The Kronecker product has many applications in machine learning, notably thanks to its very efficient and flexible factorizations in CNNs [50]. We use this formulation to generate compact, full-rank random perturbations in KronZO.

KronZO is innovative in two ways. First, its **compression** is more efficient than LoZO (see Algorithm 12), further reducing memory overhead, and the sampled directions are richer, which is key in pretraining. Second, its innovative, criterion-based **parameters update** makes it more efficient than previous ZO methods. In Algorithm 13, the matrix B^l is updated every ν iterations, which is the same behavior as ν for V^l in LoZO. The query budget q is the same as parameter q in (5.10): at each iteration, q random perturbations are sampled independently. However, the way those perturbations are managed is different from (5.6) and (5.10), see Section Parameters update.

Compression

For each layer l, KronZO generates and stores two Kronecker factors $A^l \in \mathbb{R}^{m_1^l \times n_1^l}$ and $B^l \in \mathbb{R}^{m_2^l \times n_2^l}$ such that $m_1^l m_2^l = m^l$ and $n_1^l n_2^l = n^l$ to form perturbation $Z^l = A^l \otimes B^l \in \mathbb{R}^{m^l \times n^l}$. The routine ChooseKronDims (Algorithm 12) enforces these dimensionality constraints: it calls BestPair, which returns the two divisors of an integer closest to its square root,

falling back to $(1, n)$ if the integer is prime (which is generally not the case in LMs). In short, ChooseKronDims selects the shapes of A^l and B^l so that each factor is as close to square as possible. For example, a weight matrix of size 768 × 2,304 yields $(m_1^l, m_2^l) = (24, 32)$ and $(n_1^l, n_2^l) = (48, 48)$. In Table 5.1, we compare the compression rates between the three methods on a typical attention layer of the GPT-2 model.

Table 5.1 Storage cost and compression rates for MeZO, LoZO and KronZO on a typical attention layer in GPT-2. For LoZO, we choose $r = 4$. Our method achieves better compression rates while retaining expressive perturbations.

Method	Sampling structure	Parameters stored	Compression	Example size
MeZO	Full matrix Z^l	mn	1	768×2304
LoZO	Low rank U^l, V^l	$mr + nr$	150	$U^l : 768\times4$, $V^l : 2304\times4$
KronZO	Kron. factors A^l, B^l	$m_1n_1 + m_2n_2$	650	$A^l : 24\times48$, $B^l : 32\times48$

As highlighted in Section 5.3, the pretraining task requires exploring the full optimization space. Thus, low-rank methods such as LoZO are less tailored for this task. Using (5.13), since A^l and B^l almost certainly have full rank, $A^l \otimes B^l$ also almost certainly has full rank. Similarly to MeZO, this allows the perturbation in KronZO to span the full optimization space. Therefore, KronZO outperforms the memory efficiency of LoZO, and gives insightful information on the complete optimization space, like MeZO.

Parameters update

The other major innovation in KronZO is in the way parameters are updated. In the regular q-RGE (5.6), q random perturbations are sampled, and the stochastic gradient estimators are computed and averaged, which results in $2q$ evaluations of $\mathcal{L}$ per iteration. Some inappropriate random perturbations can dominate promising ones, leading to a poor averaged estimator in practice.

In KronZO (Algorithm 13), we iteratively choose a perturbation that results in the largest decrease in the objective. For this purpose, at iteration k, we sample a fresh batch ξ_k (Line 4) and $\mathbf{B}$ if $k \,\%\, \nu = 0$ (Line 6). Then, we generate q random perturbations $\mathbf{A}_i$, $(i = 1, 2, \ldots, q)$ and compute the projection of the stochastic gradient on $\mathbf{Z}_i = \mathbf{A}_i \otimes \mathbf{B}$: $c_i\mathbf{Z}_i$ (Line 10) before computing $\mathcal{L}(\boldsymbol{\theta}_k - \alpha c_i \mathbf{Z}_i; \xi_k)$ (Line 11). We keep only the best of those q directions (Lines 12-14), i.e., the lowest of the evaluated losses. The way KronZO samples perturbations can be seen as a lucky RGE: we only use one direction to update, but this direction is competing against $q - 1$ others. It is therefore a biased perturbation, and may be a descent direction. This strategy costs $3q$ evaluations of the objective per iteration, which is more expensive than

Algorithm 12 Approximate Square Kronecker Factorization Strategy for KronZO

Function BestPair(n)
  root $\leftarrow \lfloor \sqrt{n} \rfloor$
  For offset $= 0$ **to** root **do**
    For cand $\in \{\text{root} - \text{offset}, \text{root} + \text{offset}\}$ **do**
      If cand ≥ 1 **and** n mod cand $= 0$ **then**
        return (cand, n/cand)
      end If
    end For
  end For
  return $(1, n)$ *Fallback for prime numbers*
end Function

Function ChooseKronDims(m^l, n^l)
  $(m_1, m_2) \leftarrow$ BestPair(m^l) *Row factorization*
  $(n_1, n_2) \leftarrow$ BestPair(n^l) *Column factorization*
  return (m_1, m_2, n_1, n_2)
end Function

Constraint: $m_1 \times m_2 = m^l$ and $n_1 \times n_2 = n^l$
Objective: Minimize $|m_1 - \sqrt{m^l}| + |m_2 - \sqrt{m^l}| + |n_1 - \sqrt{n^l}| + |n_2 - \sqrt{n^l}|$

(5.6) but gives richer directions. We illustrate this mechanism in Figure 5.5. With the same five random directions, the averaging rule blends useful and misleading signals, so its gradient estimator diverges from the true gradient. KronZO evaluates every candidate, picks the single direction that yields the largest objective decrease, and produces an estimate that tracks the true descent direction more closely. In high-dimensional spaces, e.g., pretraining, this dilution effect grows: the probability that informative directions are drowned out by many irrelevant ones rises, whereas KronZO still retains the best-performing direction. Although the stochastic gradient estimate used by KronZO is biased (whereas the q-RGE estimator in (5.6) is not), the perturbation-selection mechanism markedly improves convergence. As shown in Section 5.6, this strategy achieves a substantially lower loss than both (5.6) and the low-rank variant (5.10) under the same query budget.

Loss landscapes are highly non-convex, containing many local minima and exhibiting sensitivity to noise due to batch variability. Since the chosen perturbation direction depends on the current batch (which may differ significantly from others) consistently updating along the best batch-dependent direction can be unreliable. Moreover, we only sample a small number of perturbations ($q = 10$ to 50), which is negligible compared to the dimensionality of the model. As a result, the selected best perturbation may not accurately reflect the overall

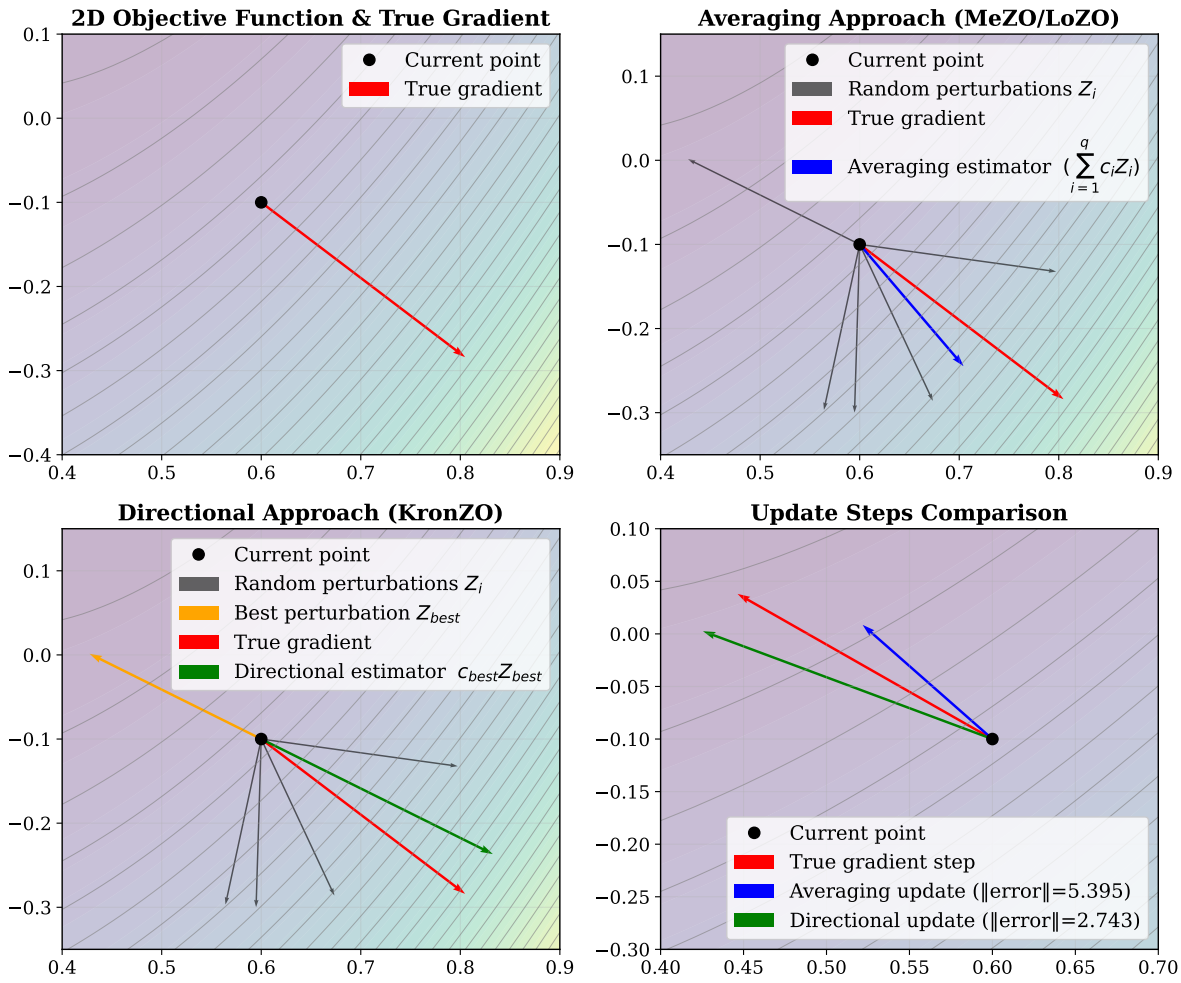

Figure 5.5 Directional versus averaging updates on a two dimensional Rosenbrock example (same five random directions in all panes). **Top-left:** objective contours and true gradient (red). **Top-right:** q-RGE/averaging—individual perturbations (black) and their mean estimator (blue). **Bottom-left:** KronZO keeps only the best perturbation (orange) and forms its directional estimator (green). **Bottom-right:** parameter-update vectors: the directional step lies much closer to the true gradient step than the averaged one.

landscape, potentially leading to suboptimal, or even *stale* updates that are uncorrelated with true descent.

To mitigate this, we compare the current best loss (over the current batch) to a sliding window of the past h losses over previous batches (Line 16). If it improves upon the worst loss in the window, we update the parameters (Line 17). Otherwise, we consider the iteration a failure and skip the update. Empirically, such failures occur in approximately 10% of iterations. In both cases, the window is refreshed with current best loss replacing the maximum loss of the window (Line 19). We term this strategy (perturbation selection and acceptance) a

directional update: parameters are advanced only when a single, carefully vetted direction, chosen for its within-batch effectiveness and cross-batch robustness, outperforms all q others, ensuring that the step aligns with the true descent geometry rather than averaging random perturbations.

Algorithm 13 Kronecker ZO optimization (KronZO)

Input: objective $\mathcal{L}$, step budget K, smoothing parameter ϵ, learning rate α, sampling interval ν, sampling strategy $S(m,n)$, query budget q, history length h.

Init: ChooseKronDims(m^l, n^l) for each layer l *see Algorithm 12*
For $k = 0, 1, \ldots, K-1$ **do**
  Sample fresh batch ξ_k
  Set $\mathcal{L}_{\text{best}} = \mathcal{L}(\boldsymbol{\theta}_k; \xi_k)$, $c_{\text{best}} = \emptyset$, $\mathbf{Z}_{\text{best}} = \emptyset$
  If $k \,\%\, \nu = 0$ **then** sample $\mathbf{B} = \{B^l\}_{l=1}^L : m_2^l \times n_2^l$ *refresh* $\mathbf{B}$
  end If
  For $i = 1, 2, \ldots q$ **do**
    Sample $\mathbf{A}_i = \{A_i^l\}_{l=1}^L : m_1^l \times m_2^l$ and define $\mathbf{Z}_i = \mathbf{A}_i \otimes \mathbf{B}$
    Compute $c_i = \dfrac{\mathcal{L}(\boldsymbol{\theta}_k + \epsilon \mathbf{Z}_i; \xi_k) - \mathcal{L}(\boldsymbol{\theta}_k - \epsilon \mathbf{Z}_i; \xi_k)}{2\epsilon}$
    Compute $\mathcal{L}_i = \mathcal{L}(\boldsymbol{\theta}_k - \alpha c_i \mathbf{Z}_i)$ *evaluate at candidate update (5.7)*
    If $\mathcal{L}_i < \mathcal{L}_{\text{best}}$ **then**
      Set $\mathcal{L}_{\text{best}} = \mathcal{L}_i$, $c_{\text{best}} = c_i$, $\mathbf{Z}_{\text{best}} = \mathbf{Z}_i$
    end If
  end For
  If $\mathcal{L}_{\text{best}} \leq \max\{\mathcal{L}_{k-1}, \mathcal{L}_{k-2}, \ldots, \mathcal{L}_{k-h}\}$ **then**
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha c_{\text{best}} \mathbf{Z}_{\text{best}}$
  end If
  Update history: $\max\{\mathcal{L}_{k-1}, \mathcal{L}_{k-2}, \ldots, \mathcal{L}_{k-\min\{h,k\}}\} \leftarrow \mathcal{L}_{\text{best}}$
end For

For each method, Table 5.2 indicates the *frozen component* that remains fixed for the next ν optimization steps for each layer, the corresponding set of admissible perturbations, i.e., the linear subspace $\mathcal{S}$ explored during that window, $\dim(\mathcal{S})$: the number of degrees of freedom, and the typical *algebraic rank* of a single update drawn from $\mathcal{S}$.

MeZO does not freeze anything, so each step is sampled from the *full* parameter space of size $m^l n^l$ for each layer, and rank $\min(m^l, n^l)$, which guarantees maximal directional coverage. This benefit comes at a cost: either MeZO stores the entire perturbation tensor (memory–intensive) or it keeps only the random seed and regenerates the tensor on demand (memory–light but requiring additional flops).

LoZO freezes a right factor $V^l \in \mathbb{R}^{n^l \times r}$. During the next ν steps every perturbation has the form $U_i^l (V^l)^\top$, confining optimization to a low-rank subspace of dimension $m^l r$. Because

Table 5.2 Comparison of the optimization subspaces induced by MeZO, LoZO and KronZO per layer of the model.

Method	Frozen for ν steps	Subspace $\mathcal{S}$	$\dim(\mathcal{S})$	Perturbation rank
MeZO	–	$\{Z^l \in \mathbb{R}^{m^l \times n^l}\}$	$m^l n^l$	$\min(m^l, n^l)$
LoZO	$V^l \in \mathbb{R}^{n^l \times r}$	$\{U^l V^{l\top}, U^l \in \mathbb{R}^{m^l \times r}\}$	$m^l r$	r
KronZO	$B^l \in \mathbb{R}^{m_2^l \times n_2^l}$	$\{A^l \otimes B^l, A^l \in \mathbb{R}^{m_1^l \times n_1^l}\}$	$m_1^l n_1^l$	$\operatorname{rank}(A^l)\operatorname{rank}(B^l)$

$\operatorname{rank}(U_i^l (V^l)^\top) \leq r \ll n^l$, LoZO strongly reduces variance and memory, yet necessarily ignores any gradient component not in $\operatorname{span}(V^l)$, a bias that can slow down pretraining where informative subspaces are high-dimensional.

KronZO freezes one Kronecker factor $B^l \in \mathbb{R}^{m_2^l \times n_2^l}$ and varies only $A^l \in \mathbb{R}^{m_1^l \times n_1^l}$. The admissible updates $Z^l = A^l \otimes B^l$ live in a structured subspace of dimension $m_1^l n_1^l \ll m^l n^l$ (strong compression) but still almost surely have *full rank* whenever A^l and B^l have full rank (5.13). This Kronecker compression keeps the memory overhead negligible while preserving rich directional coverage, offering a better bias–variance trade-off than LoZO and explaining KronZO's slightly faster convergence and lower final loss in pretraining, even without the directional update, see Section Comparison with state-of-the-art.

In short, both LoZO and KronZO optimize within a fixed lower-dimensional subspace for ν iterations: LoZO by explicit rank truncation, KronZO by Kronecker structure. Yet, only KronZO retains the potential for full-rank updates, which is crucial for thoroughly exploring the high-dimensional loss landscape encountered during large-scale pretraining.

5.5 Convergence results

In this section, we derive convergence results for KronZO. As noted in Section Parameters update, when $q > 1$, the stochastic-gradient estimator used by KronZO is biased. Although this bias speeds up the optimization process in practice, see Section Computational results, unbiasedness is required for our convergence arguments [31, 70]. Therefore, we restrict attention to the case $q = 1$, i.e., one sampled direction per iteration. For clarity, we also assume that every step is accepted; equivalently, the `if` branch at Line 16 of Algorithm 13 is never entered and the update in Line 17 always occur. Finally, we consider a single-layer model with $m \times n$ parameters, but the analysis still holds with multiple layers. Complete details can be found in Section 5.7.

Our convergence analysis builds upon the following standard regularity conditions.

Assumption 5.5.1. *The stochastic gradient $\nabla\mathcal{L}(\theta;\xi)$ satisfies:*

1. ***Lipschitz continuity.*** *There exists $L > 0$ such that for all $(\theta_1, \theta_2) \in \mathbb{R}^{m\times n}$,*

$$\left\|\nabla\mathcal{L}(\theta_1;\xi) - \nabla\mathcal{L}(\theta_2;\xi)\right\|_F \leq L\left\|\theta_1 - \theta_2\right\|_F.$$

2. ***Unbiasedness and bounded variance.*** *There exists $\sigma \geq 0$ such that for all θ,*

$$\mathbb{E}_\xi[\nabla\mathcal{L}(\theta;\xi)] = \nabla\mathcal{L}(\theta), \quad \mathbb{E}_\xi\left[\|\nabla\mathcal{L}(\theta;\xi) - \nabla\mathcal{L}(\theta)\|_{\mathrm{F}}^2\right] \leq \sigma^2.$$

In particular, AssumptionAssumption 5.5.1 ensures that $\mathcal{L}$ is continuously differentiable with a Lipschitz continuous gradient, which is a standard requirement in stochastic gradient analysis. Under the setup introduced in this section, the optimization problem reads

$$\theta^* \in \arg\min_{\theta\in\mathbb{R}^{m\times n}} \mathbb{E}_\xi[\mathcal{L}(\theta;\xi)]\,,$$

which is (5.1).

It is worth noting that (5.1) corresponds to the case of a fixed training dataset, whose randomness is represented by ξ. The complete formulation is

$$\theta^* \in \arg\min_{\theta\in\mathbb{R}^{m\times n}} \mathbb{E}_{X,Y}\left[\mathbb{E}_{\xi|(X,Y)}[\mathcal{L}(\theta;\xi,X,Y)]\right]$$

where the **outer expectation** is taken with respect to the *data-generating distribution* of (X, Y), that is, the unknown probability distribution on the input–output pairs from which training datasets are drawn. The **inner expectation** is then taken over any additional source of randomness ξ (such as minibatch sampling, data augmentation, or dropout) conditional on having fixed a particular dataset (X, Y). For simplicity, it is common to work only with the inner expectation, i.e., to solve the problem for a given dataset, which reduces to (5.1).

Consider $q = 1$ and let $A \in \mathbb{R}^{m_1\times n_1}$, $B \in \mathbb{R}^{m_2\times n_2}$ such that $m_1m_2 = m$, $n_1n_2 = n$ and $\epsilon > 0$. Denote $Z = A \otimes B \in \mathbb{R}^{m\times n}$, the KronZO estimator reads

$$\widehat{\nabla}\mathcal{L}(\theta;\xi,Z) = \frac{\mathcal{L}(\theta+\epsilon Z) - \mathcal{L}(\theta-\epsilon Z)}{2\epsilon}Z. \tag{5.14}$$

KronZO resamples B every ν iterations and A at each iteration, then updates

$$\theta \leftarrow \theta - \alpha\widehat{\nabla}\mathcal{L}(\theta;\xi,Z). \tag{5.15}$$

We first show in Lemma 3 that under this setup, the stochastic-gradient estimator (5.14) is unbiased, which is necessary for the analysis. Then, following the analysis in LoZO [31], we show that KronZO is a subspace minimization method that solves a sequence of K subproblems, each one for ν iterations. We bound the expected improvement on a single subproblem (5.26) and use it to establish Theorem 6.

Inequality (5.31) bounds a standard stationarity surrogate, namely the average squared gradient along the iterates:

$$G_K := \frac{1}{K}\sum_{k=0}^{K-1} \|\nabla\mathcal{L}(\theta_{k\nu})\|_{\mathrm{F}}^2 \le C_T$$

for some nonnegative quantity C_T that depends on $T = K\nu$ and on the algorithmic parameters. In the absence of specific choices for α and ϵ, the right-hand side need not vanish as $T \to \infty$, so convergence to stationarity is not guaranteed.

If $\alpha = \epsilon = T^{-1/2}$ and T is large enough to satisfy the step size condition $0 < \alpha = T^{-1/2} < \frac{-28+\sqrt{28^2+144}}{288\,\bar{L}m_1n_1\nu}$ (so that $\beta > 0$), then

$$\begin{aligned}\beta\,\frac{m_2n_2}{K}\sum_{k=0}^{K-1} \|\nabla\mathcal{L}(\theta_{k\nu})\|_{\mathrm{F}}^2 &\le \frac{\mathcal{L}(\theta_0)-\mathcal{L}^*}{T^{1/2}} + \frac{536\,\nu^2\,\bar{L}^4\,m_1^2n_1^2}{T^2} + \frac{24\,\nu^3\,\bar{L}^2\,\bar{\sigma}^2}{T^{3/2}} \\ &\quad + \frac{\bar{L}^2\,m_1n_1}{2T} + \frac{66\,\bar{L}^3\,m_1^3n_1^3}{T^{3/2}} + \frac{12\,\bar{\sigma}^2\,\bar{L}\,m_1n_1}{T^{1/2}} \\ &= \mathcal{O}\left(T^{-1/2}\right).\end{aligned} \tag{5.16}$$

Consequently, for sufficiently small α and ϵ, the average squared gradient norms of KronZO's iterates admit an upper bound that decays to zero at a sublinear rate as T grows, coinciding with the LoZO rate [31].

This decay has the usual stationarity implications. Let G_K be as above. From (5.16), $G_K \to 0$ as $T \to \infty$, hence

$$\min_{0\le k<K} \|\nabla\mathcal{L}(\theta_{k\nu})\|_{\mathrm{F}}^2 \le G_K \to 0,$$

so at least one iterate is asymptotically stationary. Moreover, by Markov's inequality, for any $\varepsilon > 0$,

$$\frac{1}{K}\left|\left\{0 \le k < K : \|\nabla\mathcal{L}(\theta_{k\nu})\|_{\mathrm{F}}^2 > \varepsilon\right\}\right| \le \frac{G_K}{\varepsilon} \to 0,$$

which shows that the algorithm spends an increasing fraction of iterations near stationarity. These guarantees are in the average sense and do not by themselves imply convergence of the entire sequence $\{\theta_t\}$. In practice, choosing ϵ small and a diminishing step size (e.g., $\alpha_t \propto t^{-1/2}$) ensures the average stationarity measure converges to zero.

5.6 Computational results

In this section, we present our setup and results on KronZO versus existing state-of-the-art ZO methods.

Experimental setup

We implement all methods in PyTorch [76] and base our code on nanoGPT [57]. All experiments run on a single NVIDIA A100 with 80GB of GPU memory. We pretrain GPT-2 Small (125M parameters) [80] on the OpenWebText corpus [24] with a step budget $K = 10,000$.

We choose a batch size of 32 and a context length of 1024, following the reference configuration in `nanoGPT` [57], with *no* dropout and *no* bias terms. Hyper-parameters are selected by a coarse grid search: $\epsilon \in [10^{-4}, 5 \times 10^{-4}]$; a cosine learning-rate schedule with initial value $\alpha = 3 \times 10^{-4}$. For LoZO, we keep the hyper-parameter choices of [31] (rank $r = 2$, refresh period $\nu = 50$). KronZO instead sets a shorter refresh period of $\nu = 5$, which introduces only a negligible overhead because the auxiliary matrix $\mathbf{B}$ is inexpensive to generate; it also maintains a history loss buffer of size $h = 10$.

Cost per step

MeZO and LoZO rely on antithetic sampling: one perturbation requires two objective evaluations, $\mathcal{L}(\boldsymbol{\theta} + \epsilon \mathbf{Z}_i)$ and $\mathcal{L}(\boldsymbol{\theta} - \epsilon \mathbf{Z}_i)$, so each step costs $2q$ forward passes to compute gradient estimator (5.6) and (5.10). KronZO re-uses the same pair to compute a scaling coefficient c_i for $i = 1, 2, \ldots, q$, then evaluates the candidate $\mathcal{L}(\boldsymbol{\theta} - \alpha c_i \mathbf{Z}_i)$; each sample therefore costs three forward passes, i.e., $3q$ per step. In Tables 5.3 and 5.4, all ZO methods are compared under an equivalent total budget of function evaluations, so that the higher per-iteration cost of the directional update is taken into account.

Directional versus averaging updates

Table 5.3 compares the classical *averaging* update (q-RGE (5.6)) with its *directional* counterpart that we introduce in Section Parameters update. We implement directional-update variants of MeZO and LoZO to isolate the effect of this novelty and assess its impact on convergence. To equalize wall-clock effort, the directional variants use $q' = \lceil 2q/3 \rceil$ samples so that the total number of forward passes remains identical. Both variants store a single perturbation vector, hence their memory footprint is the same.

Table 5.3 confirms that substituting the averaging step with our directional update system-

Table 5.3 Averaging vs. directional updates under a fixed forward-pass budget.

Method	Variant	Step budget q	Memory (GB)	Time/iter (s)	Final loss
MeZO	Averaging	50	1.0	6.7	8.62
	Directional	33	1.0	6.7	**7.01**
LoZO	Averaging	50	0.6	7.2	8.82
	Directional	33	0.6	7.2	**7.53**
KronZO	Averaging	50	0.5	7.4	8.63
	Directional	33	0.5	7.4	**6.95**

atically enhances optimization. Hence, directional update yields richer descent directions and markedly lower objectives across all methods with equal total evaluation budget. The restricted optimization subspace explored by LoZO does not adequately span the broad singular-value spectrum characteristic of pretraining, which accounts for its slightly reduced effectiveness. The extra run-time for LoZO and KronZO stems from their additional linear-algebra steps ($U^l V^{l\top}$ and $A^l \otimes B^l$, respectively); MeZO avoids this at the cost of higher memory.

Comparison with state-of-the-art

Table 5.4 collects the best state-of-the-art variant of each ZO method and contrasts it with SGD and AdamW. Starting from an untrained loss of about 11, MeZO and LoZO achieve a $\approx 20\%$ reduction, while KronZO reaches $\approx 37\%$ with an even smaller memory footprint than LoZO.

Table 5.4 Resource usage and final loss. KronZO matches LoZO's memory footprint while markedly improving optimization performance.

Method	Step budget q	Memory (GB)	Time/iter (s)	Final loss
AdamW (1^{st}-order)	—	1.5	0.4	3.18
SGD (1^{st}-order)	—	1.0	0.4	3.50
Untrained	—	—	—	11.0
MeZO	50	1.0	6.7	8.62
LoZO	50	0.6	7.2	8.82
KronZO	33	0.5	7.5	**6.95**
AdamW (GPT-2 XL 1.5B)	—	25.4	2.6	2.65
SGD (GPT-2 XL 1.5B)	—	18.7	2.6	3.25
KronZO (GPT-2 XL 1.5B)	33	6.4	66.1	8.21

FO methods converge more quickly and attain a lower loss, but consume two to three times more GPU memory than ZO methods, and this gap widens with model size. The last three rows of Table 5.4, which cover a 1.5-billion-parameter model confirm that KronZO still requires only a fraction of the FO memory. Its training efficiency, however, degrades at this scale, as expected: in very high dimensions the likelihood that a randomly sampled direction aligns well with the true stochastic gradient becomes vanishingly small. Designing more scalable direction-selection schemes to restore efficiency on billion-parameter LMs is therefore a key avenue for future work.

Figure 5.6 plots the memory footprint across the OPT family [104]: the KronZO over Adam memory ratio increases from 7 times on OPT-1.3B to roughly 11 times on OPT-30B. Results were extrapolated from [70], assuming $s_{\mathrm{ZO}} = 1$, corresponding to storing the full matrix $\mathbf{Z}$. These results make ZO methods such as KronZO a compelling choice whenever memory is the binding resource.

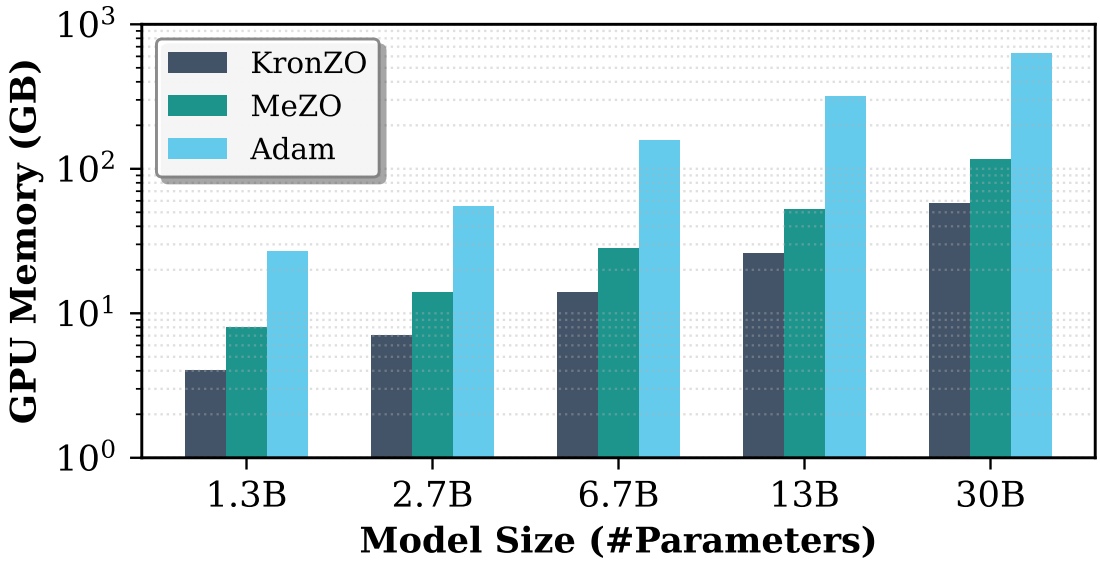

Figure 5.6 Memory usage for KronZO, MeZO and classic FO training with Adam on different sizes of OPT model in logarithmic scale.

Downstream Evaluation

This section assesses whether pretraining with KronZO yields transferable representations across diverse NLP tasks. We compare the downstream performance of two pretrained GPT-2 XL (1.5B) models: (i) pretrained with SGD, and (ii) pretrained with KronZO, both on OpenWebText under the same setup.

To ensure a fair comparison, we deliberately exclude additional training heuristics such as weight decay or momentum from (i), since these are not part of KronZO, which results in a simple baseline we refer to as *vanilla* SGD. The integration of these features into KronZO is left to future work.

We fully fine-tune both models on the GLUE [98] benchmark, including both the backbone and linear classification head, without freezing any layers.

Fine-tuning is performed with AdamW (β_1=0.9, β_2=0.999, weight decay 0.01). We fine-tune for 3 epochs on MNLI, QQP, QNLI, and SST-2, and for up to 10 epochs on MRPC, RTE, CoLA, and WNLI, with early stopping (patience=2) based on each task's primary development metric.

Evaluation metrics reported in Table 5.5 follow GLUE standards: accuracy for MNLI, QNLI, SST-2, RTE, and WNLI; accuracy and F1 for QQP and MRPC; and Matthews correlation coefficient (MCC) for CoLA. MNLI results are reported for both matched and mismatched splits.

Each experiment is repeated with three random seeds. Macro scores are computed as the mean across tasks for accuracy (classification tasks), F1 (QQP/MRPC only), and MCC (CoLA only).

Table 5.5 GLUE development results for GPT-2 XL pretrained with KronZO and vanilla SGD. We report accuracy for all tasks except QQP/MRPC (F1) and CoLA (MCC).

Method	MNLI_m	MNLI_{mm}	QQP	QNLI	SST-2	MRPC	RTE	CoLA	WNLI	**Avg.**
KronZO	0.589	**0.598**	**0.728**	**0.625**	0.807	0.816	**0.552**	**0.114**	0.563	**0.601**
SGD	**0.594**	0.597	0.722	0.615	**0.811**	**0.822**	0.527	0.000	0.563	0.583

In Table 5.5, KronZO matches vanilla SGD on all nine tasks, which shows consistent generalization on cross-domain natural language inference (MNLI-mm) and small-sample reasoning tasks such as RTE. On larger or more stable datasets (e.g., SST-2, MRPC), both methods perform similarly, while CoLA remains challenging for both models, as expected for causal language models. Overall, these results suggest that KronZO preserves downstream transferability while relying solely on ZO updates.

The goal of this experiment is not to maximize benchmark performance but to assess whether representations learned through ZO optimization transfer effectively to diverse downstream tasks. From this perspective, KronZO achieves downstream generalization on par with FO SGD, demonstrating that ZO pretraining can produce transferable representations at scale. Taken together, these results indicate that the representations learned during KronZO pre-

training transfer effectively to tasks that differ substantially from the OpenWebText pretraining distribution. These results also provide direct empirical evidence that KronZO-based pretraining yields features that generalize beyond the specific corpus used for pretraining.

5.7 Theoretical results

This section is dedicated to the convergence analysis of KronZO. We first prove the unbiasedness of the estimator (5.14), then show that KronZO can be seen as a subspace optimization problem before showing Theorem 6 from Convergence results.

5.7.1 Unbiasedness of KronZO

Definition 3 (Unbiased estimator)**.** *A gradient estimator with parameters P, $\widehat{\nabla}\mathcal{L}(\theta;\xi,P)$ is unbiased if*

$$\mathbb{E}_P\Big[\widehat{\nabla}\mathcal{L}(\theta;\xi;P)\Big] = \nabla\mathcal{L}(\theta;\xi).$$

Lemma 3. *The gradient estimator in KronZO is unbiased. For all $\theta \in \mathbb{R}^{m\times n}, \xi \in \mathbb{R}$,*

$$\mathbb{E}_Z\Big[\widehat{\nabla}\mathcal{L}(\theta;\xi,Z)\Big] = \nabla\mathcal{L}(\theta;\xi).$$

Proof. In vectorized notation,

$$\lim_{\epsilon\to 0} vec(\widehat{\nabla}\mathcal{L}(\theta;\xi,Z,\epsilon)) = vec(Z)vec(Z^\top)vec(\nabla\mathcal{L}(\theta;\xi)).$$

Take the expectation on Z,

$$\mathbb{E}_Z\Big[vec(\widehat{\nabla}\mathcal{L}(\theta;\xi))\Big] = \mathbb{E}_Z\Big[vec(Z)vec(Z^\top)vec(\nabla\mathcal{L}(\theta;\xi))\Big] = \Sigma \times vec(\nabla\mathcal{L}(\theta;\xi)),$$

with $\Sigma = \mathbb{E}_Z\Big[vec(Z)vec(Z)^\top\Big]$ the covariance matrix of Z. Each component of Z is sampled from $\mathcal{N}(0,1)$ therefore $\Sigma = I_{mn}$ which leads to

$$\mathbb{E}_Z\Big[\widehat{\nabla}\mathcal{L}(\theta;\xi,Z,\epsilon)\Big] = \nabla\mathcal{L}(\theta;\xi).$$

□

5.7.2 KronZO viewed as subspace minimization

Let $\phi : \mathbb{R}^{p\times q} \to \mathbb{R}^{m\times n}$ be a *fixed* linear map that defines the search subspace.

$$\phi(X) = \begin{cases} X\,V^\top & \text{for LoZO with fixed } U \\ X \otimes B & \text{for KronZO with fixed } B. \end{cases}$$

Define

$$\mathcal{G}_\theta(X;\xi) := \mathcal{L}\Big(\theta + \phi(X);\xi\Big), \qquad \mathcal{G}_\theta(X) := \mathbb{E}_\xi[\mathcal{G}_\theta(X;\xi)]\,.$$

For a search direction A we abbreviate the two–point estimator

$$\widehat{\nabla}\mathcal{G}_\theta(X;\xi,A) \;:=\; \frac{\mathcal{G}_\theta(X+\epsilon A;\xi) - \mathcal{G}_\theta(X-\epsilon A;\xi)}{2\epsilon}\,A.$$

To solve (5.1), KronZO iteratively solves the following subproblem for ν iterations with fixed matrix B:

$$X^* \in \arg\min_{X\in\mathbb{R}^{m_1\times n_1}} \mathcal{G}_\theta(X). \tag{5.17}$$

At subproblem k, (5.17) is solved by following the update rule

$$X_{k,s+1} = X_{k,s} - \alpha\widehat{\nabla}_X\mathcal{L}(\tilde{\theta}_k + X_{k,s}\otimes B_k;\xi_{k,s}), \quad s = 0,1,\ldots,\nu-1, \tag{5.18}$$

$$\tilde{\theta}_{k+1} = \tilde{\theta}_k + X_{k,\nu}\otimes B_k. \tag{5.19}$$

Here, $X_{k,0} = \mathbf{0}$, and

$$\widehat{\nabla}_X\mathcal{L}(\tilde{\theta}_k + X_{k,s}\otimes B_k;\xi_{k,s}) = \\ \frac{\mathcal{L}(\tilde{\theta}_k + (X_{k,s}+\epsilon A_{k,s})\otimes B_k;\xi_{k,s}) - \mathcal{L}(\tilde{\theta}_k + (X_{k,s}-\epsilon A_{k,s})\otimes B_k;\xi_{k,s})}{2\epsilon}A_{k,s},$$

where $A_{k,s}$ is sampled from a normal distribution every iteration. Let $Y_{k,s} := \tilde{\theta}_k + X_{k,s}\otimes B_k$. We use the update rule (5.18) to derive the update of Y

$$Y_{k,s+1} = \tag{5.20}$$

$$Y_{k,s} - \alpha\tfrac{\mathcal{L}(Y_{k,s}+\epsilon A_{k,s}\otimes B_k;\xi_{k,s}) - \mathcal{L}(Y_{k,s}-\epsilon A_{k,s}\otimes B_k;\xi_{k,s})}{2\epsilon}A_{k,s}\otimes B_k, \tag{5.21}$$

which is exactly (5.15). Details on the equivalence between methods such as LoZO and KronZO and the subspace minimization problem (5.17) can be found in [31, Section 4.2].

5.7.3 Convergence analysis

The analysis is divided in two parts. First, we derive a bound on the subproblem iterations in Lemma 9, then use it to derive an upper bound on the whole optimization process in Theorem 6. In the remainder of this section, we suppose Assumption 5.5.1 is satisfied.

Lemma 4 (Lipschitz continuity in the X-space)**.** *Let $B \in \mathbb{R}^{m_2 \times n_2}$ be fixed and define $\phi(X) = X \otimes B$. Under Assumption 5.5.1,*

$$X \longmapsto \nabla_X \mathcal{G}_\theta(X)$$

is $\widetilde{L}$–Lipschitz w.r.t. the Frobenius norm, where

$$\widetilde{L} \;=\; L \, \|B\|_F^2.$$

Where L is the Lipschitz constant of $\theta \mapsto \nabla_\theta \mathcal{L}(\theta)$ from Assumption 5.5.1.

Proof. Set $g(X) := \theta + X \otimes B$, so that $\mathcal{G}_\theta(X) = \mathcal{L}(g(X))$. For any perturbation ΔX, by the chain rule,

$$\mathrm{d}\mathcal{G}_\theta(X)[\Delta X] = \Big\langle \nabla_\theta \mathcal{L}\Big(g(X)\Big),\, \Delta X \otimes B \Big\rangle_F.$$

By Frobenius duality, $\nabla_X \mathcal{G}_\theta(X) = \mathcal{C}_B\Big(\nabla_\theta \mathcal{L}\Big(g(X)\Big)\Big)$, where the contraction $\mathcal{C}_B$ (adjoint of $X \mapsto X \otimes B$) satisfies $\|\mathcal{C}_B\|_{\mathrm{op}} \le \|B\|_F$.

Let X_1, X_2 and $Y_k = g(X_k)$. Then

$$\|\nabla_X \mathcal{G}_\theta(X_1) - \nabla_X \mathcal{G}_\theta(X_2)\|_F \le \|B\|_F \, \|\nabla_\theta \mathcal{L}(Y_1) - \nabla_\theta \mathcal{L}(Y_2)\|_F \le L \, \|B\|_F \, \|Y_1 - Y_2\|_F.$$

Since $\|Y_1 - Y_2\|_F = \|(X_1 - X_2) \otimes B\|_F = \|X_1 - X_2\|_F \, \|B\|_F$,

$$\|\nabla_X \mathcal{G}_\theta(X_1) - \nabla_X \mathcal{G}_\theta(X_2)\|_F \le L \, \|B\|_F^2 \, \|X_1 - X_2\|_F.$$

□

Lemma 5 (Variance bound of the projected gradient)**.** *Let $B \in \mathbb{R}^{m_2 \times n_2}$ be fixed and define $\phi(X) = X \otimes B$. Under Assumption 5.5.1, for any $X \in \mathbb{R}^{m_1 \times n_1}$,*

$$\mathbb{E}_\xi\Big[\|\nabla_X \mathcal{G}_\theta(X;\xi) - \nabla_X \mathcal{G}_\theta(X)\|_F^2\Big] \le \widetilde{\sigma}^2$$

where $\widetilde{\sigma}^2 := \sigma^2 \|B\|_F^2$ and σ^2 is the variance bound from Assumption 5.5.1.

Proof. With $\phi(X) = X \otimes B$ and the Frobenius duality (see the proof of Lemma 4) we have

$$\nabla_X \mathcal{G}_\theta(X;\xi) \;=\; \mathcal{C}_B\Big(\nabla_\theta \mathcal{L}(\theta + X \otimes B;\xi)\Big),$$

where the *contraction operator* $\mathcal{C}_B : \mathbb{R}^{(m_1 m_2)\times(n_1 n_2)} \longrightarrow \mathbb{R}^{m_1 \times n_1}$ is defined block–wise by

$$\Big[\mathcal{C}_B(T)\Big]_{ij} := \langle T_{(ij)},\, B\rangle_F, \qquad T_{(ij)} \in \mathbb{R}^{m_2 \times n_2} \text{ is the } (i,j)\text{–block of } T.$$

For any block matrix T

$$\Big\|\mathcal{C}_B(T)\Big\|_F^2 \le \|B\|_F^2 \sum_{i,j} \|T_{(ij)}\|_F^2 = \|B\|_F^2 \|T\|_F^2,$$

hence $\|\mathcal{C}_B\|_{\mathrm{op}} \le \|B\|_F$.

Define the zero-mean random matrix

$$\Delta(\xi) := \nabla_\theta \mathcal{L}(\theta + X \otimes B;\xi) - \nabla_\theta \mathcal{L}(\theta + X \otimes B).$$

By Assumption 5.5.1, $\mathbb{E}_\xi\Big[\|\Delta(\xi)\|_{\mathrm{F}}^2\Big] \le \sigma^2$.

$$\begin{aligned}
\mathbb{E}_\xi\Big[\|\nabla_X \mathcal{G}_\theta(X;\xi) - \nabla_X \mathcal{G}_\theta(X)\|_{\mathrm{F}}^2\Big] &= \mathbb{E}_\xi\Big[\Big\|\mathcal{C}_B\Big(\Delta(\xi)\Big)\Big\|_{\mathrm{F}}^2\Big] \\
&\le \|\mathcal{C}_B\|_{\mathrm{op}}^2\, \mathbb{E}_\xi\Big[\|\Delta(\xi)\|_{\mathrm{F}}^2\Big] \\
&\le \|B\|_{\mathrm{F}}^2\; \sigma^2,
\end{aligned}$$

which completes the proof. □

Lemma 6. *Under Assumption 5.5.1, for the gradient estimator $\widehat{\nabla}\mathcal{G}_\theta(X;\xi)$, the following bound holds:*

$$\mathbb{E}_A\Big[\Big\|\widehat{\nabla}\mathcal{G}_\theta(X;\xi)\Big\|_{\mathrm{F}}^2\Big] \le \frac{\tilde{L}(m_1 n_1 + 4)^3 \epsilon^2}{4} + 6 m_1^2 n_1^2 \,\|\nabla \mathcal{G}_\theta(X;\xi)\|_{\mathrm{F}}^2 \,.$$

Proof. First,

$$\begin{aligned}
\mathbb{E}_A\Big[\Big\|\widehat{\nabla}\mathcal{G}_\theta(X;\xi)\Big\|_{\mathrm{F}}^2\Big] \le\; & 2\mathbb{E}_A\Big[\Big\|\widehat{\nabla}\mathcal{G}_\theta(X;\xi) - \langle \nabla \mathcal{G}_\theta(X;\xi), A\rangle A\Big\|_{\mathrm{F}}^2\Big] \\
& + 2\mathbb{E}_A\Big[\|\langle \nabla \mathcal{G}_\theta(X;\xi), A\rangle A\|_{\mathrm{F}}^2\Big] \,.
\end{aligned} \tag{5.22}$$

For the first term in (5.22), we use the Taylor inequality along with Lemma 4:

$$|\mathcal{G}_\theta(X+\epsilon A;\xi)-\mathcal{G}_\theta(X)-\epsilon\langle\nabla\mathcal{G}_\theta(X;\xi),A\rangle|\le\frac{\tilde{L}\epsilon^2}{2}\|A\|_{\mathrm{F}}^2,$$
$$|\mathcal{G}_\theta(X-\epsilon A;\xi)-\mathcal{G}_\theta(X)+\epsilon\langle\nabla\mathcal{G}_\theta(X;\xi),A\rangle|\le\frac{\tilde{L}\epsilon^2}{2}\|A\|_{\mathrm{F}}^2.$$

The previous inequalities yield

$$|\mathcal{G}_\theta(X+\epsilon A;\xi)-\mathcal{G}_\theta(X-\epsilon A;\xi)-2\epsilon\langle\nabla\mathcal{G}_\theta(X;\xi),A\rangle|\le\tilde{L}\epsilon^2\|A\|_{\mathrm{F}}^2.$$

Divide both sides by 2ϵ, multiply by $\|A\|_F$ and take the square

$$\left\|\widehat{\nabla}\mathcal{G}_\theta(X;\xi)-\langle\nabla\mathcal{G}_\theta(X;\xi),A\rangle A\right\|_{\mathrm{F}}^2\le\frac{\tilde{L}\epsilon^2}{4}\|A\|_F^6.$$

Take the expectation on A and multiply by two give

$$2\mathbb{E}_A\left[\left\|\widehat{\nabla}\mathcal{G}_\theta(X;\xi)-\langle\nabla\mathcal{G}_\theta(X;\xi),A\rangle A\right\|_{\mathrm{F}}^2\right]\le\frac{\tilde{L}\epsilon^2}{2}\mathbb{E}_A\left[\|A\|_F^6\right].$$

Since $A:m_1\times n_1$ has i.i.d $\mathcal{N}(0,1)$ entries, the Frobenius norm satisfies $\|A\|_{\mathrm{F}}^2\sim\chi^2(m_1n_1)$. Then, $\mathbb{E}_A[\|A\|_F^6]=\mathbb{E}_A\left[(\|A\|_{\mathrm{F}}^2)^3\right]=\mathbb{E}_A[(\chi^2(m_1n_1))^3]=m_1n_1(m_1n_1+2)(m_1n_1+4)\le(m_1n_1+4)^3$. Finally,

$$2\mathbb{E}_A\left[\left\|\widehat{\nabla}\mathcal{G}_\theta(X;\xi)-\langle\nabla\mathcal{G}_\theta(X;\xi),A\rangle A\right\|_{\mathrm{F}}^2\right]\le\frac{\tilde{L}(m_1n_1+4)^3\epsilon^2}{4}.$$

The second term in Equation (5.22) can be calculated directly with the Cauchy-Schwarz inequality:

$2\mathbb{E}_A\left[\|\langle\nabla\mathcal{G}_\theta(X;\xi),A\rangle A\|_{\mathrm{F}}^2\right]\le2\|\nabla\mathcal{G}_\theta(X;\xi)\|_{\mathrm{F}}^2\,\mathbb{E}_A[\|A\|_F^4]\le6m_1^2n_1^2\|\nabla\mathcal{G}_\theta(X;\xi)\|_{\mathrm{F}}^2$, which completes the proof. □

We now introduce the *Gaussian smoothing function*:

$$\mathcal{G}_\theta^\epsilon(X)=\mathbb{E}_A[\mathcal{G}_\theta(X+\epsilon A)]=\frac{1}{\kappa}\int\mathcal{G}_\theta(X+\epsilon A)e^{-\frac{1}{2}}\|A\|_{\mathrm{F}}^2\,dA,$$

where κ is the normalization parameter: $\kappa=\int e^{-\frac{1}{2}\|A\|_{\mathrm{F}}^2}\,dA$.

Lemma 7. *For the Gaussian smoothing function $\mathcal{G}_\theta^\epsilon(X)$, the following properties hold:*

- $\mathbb{E}_{A,\xi}\left[\widehat{\nabla}\mathcal{G}_\theta(X;\xi)\right]=\nabla\mathcal{G}_\theta^\epsilon(X)$,

- $\mathcal{G}_\theta^\epsilon(X)$ *is* $\tilde{L}$*-smooth,*
- $|\mathcal{G}_\theta^\epsilon(X) - \mathcal{G}_\theta(X)| \leq \frac{\tilde{L} m_1 n_1 \epsilon^2}{2}$,
- $\|\nabla \mathcal{G}_\theta^\epsilon(X) - \nabla \mathcal{G}_\theta(X)\|_{\mathrm{F}}^2 \leq \tilde{L}^2 m_1 n_1 \epsilon^2$.

Proof. The three first properties can be found in [73, Section 2]. The proof of the last claim can be found in [31, Lemma B.3]. □

Lemma 8. *If the step size condition* $36\tilde{L}^2\alpha^2\nu m_1^2 n_1^2 \leq \frac{1}{\nu-1}$ *is satisfied, then for any* $t \leq \nu$*, the following bound holds:*

$$\begin{aligned}\mathbb{E}_{A,\xi}\left[\|X_t - X_0\|_{\mathrm{F}}^2\right] \leq\ & 144\alpha^2\nu^2 m_1^2 n_1^2 \mathbb{E}_{A,\xi}\left[\|\nabla \mathcal{G}_\theta(X_0;\xi_0)\|_{\mathrm{F}}^2\right] \\ & + 536\alpha^2\tilde{L}^2\epsilon^2 m_1^3 n_1^3 \nu^2 + 24\nu^2\alpha^2\tilde{\sigma}^2.\end{aligned} \tag{5.23}$$

Proof. Let $t \in [0, \nu-1]$. By definition of X_{t+1},

$$\begin{aligned}\mathbb{E}_{A,\xi}\left[\|X_{t+1} - X_0\|_{\mathrm{F}}^2\right] &= \mathbb{E}_{A,\xi}\left[\left\|X_t - \alpha\widehat{\nabla}\mathcal{G}_\theta(X_t;\xi_t) - X_0\right\|_{\mathrm{F}}^2\right] \\ &\leq \mathbb{E}_{A,\xi}\left[\|X_t - \alpha\nabla\mathcal{G}_\theta^\epsilon(X_t) - X_0\|_{\mathrm{F}}^2\right] \\ &\quad + \alpha^2\mathbb{E}_{A,\xi}\left[\left\|\nabla\mathcal{G}_\theta^\epsilon(X_t) - \widehat{\nabla}\mathcal{G}_\theta(X_t;\xi_t)\right\|_{\mathrm{F}}^2\right].\end{aligned} \tag{5.24}$$

Since the squared Frobenius norm is convex, for any function ϕ and for any $c \in (0,1)$,

$$\begin{aligned}\|\phi(x) + \phi(y)\|_{\mathrm{F}}^2 &= \left\|c\frac{\phi(x)}{c} + (1-c)\frac{\phi(y)}{1-c}\right\|_{\mathrm{F}}^2 \\ &\leq c\left\|\frac{\phi(x)}{c}\right\|_{\mathrm{F}}^2 + (1-c)\left\|\frac{\phi(y)}{1-c}\right\|_{\mathrm{F}}^2 \\ &= \frac{1}{c}\|\phi(x)\|_{\mathrm{F}}^2 + c\|\phi(y)\|_{\mathrm{F}}^2.\end{aligned}$$

For the first term in (5.24), the previous convexity inequality with $c = \frac{\nu-1}{\nu} \in (0,1)$ gives

$$\begin{aligned}\mathbb{E}_{A,\xi}\left[\|X_t - \alpha\nabla\mathcal{G}_\theta^\epsilon(X_t) - X_0\|_{\mathrm{F}}^2\right] \leq\ & \left(1 + \frac{1}{\nu-1}\right)\mathbb{E}_{A,\xi}\left[\|X_t - X_0\|_{\mathrm{F}}^2\right] \\ & + \nu\mathbb{E}_{A,\xi}\left[\|\alpha\nabla\mathcal{G}_{\theta^\epsilon}(X_t)\|_{\mathrm{F}}^2\right].\end{aligned}$$

For the second term, we reuse the convexity inequality to get

$$\alpha^2\mathbb{E}_{A,\xi}\Big[\big\|\nabla\mathcal{G}_\theta^\epsilon(X_t)-\widehat{\nabla}\mathcal{G}_\theta(X_t;\xi_t)\big\|_{\mathrm{F}}^2\Big]\leq\frac{\nu\alpha^2}{\nu-1}\mathbb{E}_{A,\xi}\Big[\big\|\widehat{\nabla}\mathcal{G}_\theta(X_t,\xi_t)\big\|_{\mathrm{F}}^2\Big]+\alpha^2\nu\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta^\epsilon(X_t)\|_{\mathrm{F}}^2\Big].$$

The two previous inequalities yield

$$\begin{aligned}\mathbb{E}_{A,\xi}\Big[\|X_{t+1}-X_0\|_{\mathrm{F}}^2\Big]\leq\Big(1+\frac{1}{\nu-1}\Big)\mathbb{E}_{A,\xi}\Big[\|X_t-X_0\|_{\mathrm{F}}^2\Big]&\\+2\alpha^2\nu\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta^\epsilon(X_t)\|_{\mathrm{F}}^2\Big]&\\+\alpha^2\Big(1+\frac{1}{\nu-1}\Big)\mathbb{E}_{A,\xi}\Big[\big\|\widehat{\nabla}\mathcal{G}_\theta(X_t,\xi_t)\big\|_{\mathrm{F}}^2\Big].&\end{aligned}\tag{5.25}$$

To bound the second term in (5.25), we add and subtract a compensatory term along with Jensen's inequality to get

$$\begin{aligned}&2\alpha^2\nu\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta^\epsilon(X_t)\|_{\mathrm{F}}^2\Big]\\&=2\alpha^2\nu\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta^\epsilon(X_t)-\nabla\mathcal{G}_\theta(X_t)+\nabla\mathcal{G}_\theta(X_t)-\nabla\mathcal{G}_\theta(X_t;\xi_t)+\nabla\mathcal{G}_\theta(X_t;\xi_t)\|_{\mathrm{F}}^2\Big]\\&\leq 6\nu\alpha^2\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta^\epsilon(X_t)-\nabla\mathcal{G}_\theta(X_t)\|_{\mathrm{F}}^2\Big]+6\nu\alpha^2\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_t)-\nabla\mathcal{G}_\theta(X_t;\xi_t)\|_{\mathrm{F}}^2\Big]\\&+6\nu\alpha^2\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_t;\xi_t)\|_{\mathrm{F}}^2\Big].\end{aligned}$$

Now, Lemmas 6 and 7 give

$$2\alpha^2\nu\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta^\epsilon(X_t)\|_{\mathrm{F}}^2\Big]\leq 6\nu\alpha^2\tilde{L}^2m_1n_1\epsilon^2+6\nu\alpha^2\tilde{\sigma}^2+6\nu\alpha^2\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_t;\xi_t)\|_{\mathrm{F}}^2\Big].$$

To bound the third term in (5.25), Lemma 6 along with $m_1^2n_1^2\geq m_1n_1\geq 1$ yield

$$\begin{aligned}\alpha^2\Big(1+\frac{1}{\nu-1}\Big)\mathbb{E}_{A,\xi}\Big[\big\|\widehat{\nabla}\mathcal{G}_\theta(X_t,\xi_t)\big\|_{\mathrm{F}}^2\Big]&\leq 2\alpha^2\mathbb{E}_{A,\xi}\Big[\big\|\widehat{\nabla}\mathcal{G}_\theta(X_t,\xi_t)\big\|_{\mathrm{F}}^2\Big]\\&\leq 2\alpha^2\frac{\tilde{L}(m_1n_1+4)^3\epsilon^2}{4}+12\alpha^2m_1^2n_1^2\,\|\nabla\mathcal{G}_\theta(X;\xi)\|_{\mathrm{F}}^2\\&\leq 128\alpha^2\tilde{L}^2\epsilon^2m_1^3n_1^3\nu+12\alpha^2m_1^2n_1^2\,\|\nabla\mathcal{G}_\theta(X;\xi)\|_{\mathrm{F}}^2.\end{aligned}$$

We can now bound the right term in (5.25)

$$\begin{aligned}\mathbb{E}_{A,\xi}\Big[\|X_{t+1}-X_0\|_{\mathrm{F}}^2\Big] \leq \Big(1+\frac{1}{\nu-1}\Big)\mathbb{E}_{A,\xi}\Big[\|X_t-X_0\|_{\mathrm{F}}^2\Big] \\ + (6\nu\alpha^2+12\alpha^2m_1^2n_1^2)\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_t;\xi_t)\|_{\mathrm{F}}^2\Big] \\ + 6\nu\alpha^2m_1^3n_1^3\epsilon^2+6\nu\alpha^2\tilde{\sigma}^2+128\alpha^2\tilde{L}^2\epsilon^2m_1^3n_1^3.\end{aligned}$$

The $\tilde{L}$-smoothness of $\nabla\mathcal{G}_\theta(X,\xi)$ from Lemma 4 and $\nu\geq 2$ induce

$$\begin{aligned}\mathbb{E}_{A,\xi}\Big[\|X_{t+1}-X_0\|_{\mathrm{F}}^2\Big] \leq \Big(1+\frac{1}{\nu-1}+2\tilde{L}^2(6\nu\alpha^2+12\alpha^2m_1^2n_1^2)\Big)\mathbb{E}_{A,\xi}\Big[\|X_t-X_0\|_{\mathrm{F}}^2\Big] \\ + (12\alpha^2m_1^2n_1^2\nu+24\alpha^2m_1^2n_1^2\nu)\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_0;\xi_0)\|_{\mathrm{F}}^2\Big] \\ + 6\nu\alpha^2\tilde{\sigma}^2+134\alpha^2\tilde{L}^2\epsilon^2m_1^3n_1^3\nu.\end{aligned}$$

We simplify the previous expression

$$\begin{aligned}\mathbb{E}_{A,\xi}\Big[\|X_{t+1}-X_0\|_{\mathrm{F}}^2\Big] \leq \Big(1+\frac{1}{\nu-1}+36\tilde{L}^2\alpha^2\nu m_1^2n_1^2\Big)\mathbb{E}_{A,\xi}\Big[\|X_t-X_0\|_{\mathrm{F}}^2\Big] \\ + 36\alpha^2\nu m_1^2n_1^2\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_0;\xi_0)\|_{\mathrm{F}}^2\Big] \\ + 134\alpha^2\tilde{L}^2\epsilon^2m_1^3n_1^3\nu+6\nu\alpha^2\tilde{\sigma}^2.\end{aligned}$$

The step size condition $36\tilde{L}^2\alpha^2\nu m_1^2n_1^2\leq\frac{1}{\nu-1}$ yields

$$\begin{aligned}\mathbb{E}_{A,\xi}\Big[\|X_{t+1}-X_0\|_{\mathrm{F}}^2\Big] \leq \Big(1+\frac{2}{\nu-1}\Big)\mathbb{E}_{A,\xi}\Big[\|X_t-X_0\|_{\mathrm{F}}^2\Big] \\ + 36\alpha^2\nu m_1^2n_1^2\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_0;\xi_0)\|_{\mathrm{F}}^2\Big] \\ + 134\alpha^2\tilde{L}^2\epsilon^2m_1^3n_1^3\nu+6\nu\alpha^2\tilde{\sigma}^2.\end{aligned}$$

By induction, note that $\sum_{s=0}^{t-1}\Big(1+\frac{2}{\nu-1}\Big)\leq 4\nu$ for $t\leq\nu$,

$$\begin{aligned}&\mathbb{E}_{A,\xi}\Big[\|X_t-X_0\|_{\mathrm{F}}^2\Big] \\ &\leq\sum_{s=0}^{t-1}\Big(1+\frac{2}{\nu-1}\Big)\Big(36\alpha^2\nu m_1^2n_1^2\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_0;\xi_0)\|_{\mathrm{F}}^2\Big]+134\alpha^2\tilde{L}^2\epsilon^2m_1^3n_1^3\nu+6\nu\alpha^2\tilde{\sigma}^2\Big) \\ &\leq 144\alpha^2\nu^2m_1^2n_1^2\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{G}_\theta(X_0;\xi_0)\|_{\mathrm{F}}^2\Big]+536\alpha^2\tilde{L}^2\epsilon^2m_1^3n_1^3\nu^2+24\nu^2\alpha^2\tilde{\sigma}^2,\end{aligned}$$

which concludes the proof. □

In Lemma 8, the step size condition $36\tilde{L}^2\alpha^2\nu m_1^2 n_1^2 \leq \frac{1}{\nu-1} \implies \alpha \leq \frac{1}{6\tilde{L}m_1 n_1(\nu-1)}$, which is a weaker condition in expectation than $\alpha < \frac{-28+\sqrt{28^2+144}}{288\tilde{L}m_1 n_1 \nu}$. Therefore, if α satisfies (5.30), the step condition in Lemma 8 is also satisfied.

Lemma 9. *The following bound holds for subproblem (5.17):*

$$\begin{aligned}\mathbb{E}_{A,\xi}[\mathcal{G}_\theta(X_\nu) - \mathcal{G}_\theta(X_0)] \leq \left(-\frac{\nu\alpha}{4} + 28\tilde{L}m_1 n_1 \nu^2\alpha^2 + 144\alpha^3\nu^3 m_1^2 n_1^2 \tilde{L}^2\right)\mathbb{E}_{A,\xi}\left[\|\nabla\mathcal{G}_\theta(X_0)\|_{\mathrm{F}}^2\right] \\ + 536\nu^3\tilde{L}^4\alpha^3\epsilon^2 m_1^2 n_1^2 + 24\nu^3\tilde{L}^2\alpha^3\tilde{\sigma}^2 \\ + \frac{\tilde{L}^2 m_1 n_1 \epsilon^2\alpha\nu}{2} + 66\tilde{L}^3 m_1^3 n_1^3\epsilon^2\nu\alpha^2 + 12\tilde{\sigma}^2\tilde{L}m_1 n_1\alpha^2\nu. \end{aligned} \tag{5.26}$$

Proof. The proof is very similar to [31, Lemma B.5], therefore we rewrite in our notation only the important steps of the proof. First, using $\tilde{L}$-smoothness of $\mathcal{G}_\theta^\epsilon$, and the inequality $X_\nu - X_0 = -\alpha\sum_{t=0}^{\nu-1}\widehat{\nabla}\mathcal{G}_\theta(X_t;\xi_t)$,

$$\begin{aligned}\mathbb{E}_{A,\xi}[\mathcal{G}_\theta(X_\nu) - \mathcal{G}_\theta(X_0)] &\leq \mathbb{E}_{A,\xi}[\langle\nabla\mathcal{G}_\theta^\epsilon(X_0), X_\nu - X_0\rangle] + \frac{\tilde{L}}{2}\mathbb{E}_{A,\xi}\left[\|X_\nu - X_0\|_{\mathrm{F}}^2\right] \\ &= -\alpha\sum_{t=0}^{\nu-1}\mathbb{E}_{A,\xi}\left[\langle\nabla\mathcal{G}_\theta^\epsilon(X_0), \widehat{\nabla}\mathcal{G}_\theta(X_t;\xi_t)\rangle\right] \\ &+ \frac{\alpha^2\tilde{L}}{2}\mathbb{E}_{A,\xi}\left[\left\|\sum_{t=0}^{\nu-1}\widehat{\nabla}\mathcal{G}_\theta()X_t;\xi_t\right\|_{\mathrm{F}}^2\right]. \end{aligned} \tag{5.27}$$

The first term in (5.27) can be bounded using $\langle a, b\rangle \leq \frac{\|a^2\|+\|b\|^2}{2}$ along with Lemma 4 and Lemma 7 to get

$$\begin{aligned}-\alpha\sum_{t=0}^{\nu-1}\mathbb{E}_{A,\xi}\left[\langle\nabla\mathcal{G}_\theta^\epsilon(X_0), \widehat{\nabla}\mathcal{G}_\theta(X_t;\xi_t)\rangle\right] \leq -\frac{\alpha\nu}{4}\mathbb{E}_{A,\xi}\left[\|\nabla\mathcal{G}_\theta(X_0)\|_{\mathrm{F}}^2\right] \\ + \frac{\tilde{L}^2\alpha}{2}\sum_{t=0}^{\nu-1}\mathbb{E}_{A,\xi}\left[\|X_t - X_0\|_{\mathrm{F}}^2\right] \\ + \frac{\tilde{L}^2 m_1 n_1\epsilon^2\alpha\nu}{2}. \end{aligned} \tag{5.28}$$

On the other hand, using Lemmas 6 and 7

$$\frac{\alpha^2\tilde{L}}{2}\mathbb{E}_{A,\xi}\left[\left\|\sum_{t=0}^{\nu-1}\widehat{\nabla}\mathcal{G}_\theta(X_t;\xi_t)\right\|_{\mathrm{F}}^2\right] \leq 28\tilde{L}m_1n_1\nu^2\alpha^2\mathbb{E}_{A,\xi}\left[\|\nabla\mathcal{G}_\theta(X_0)\|_{\mathrm{F}}^2\right] + 28\tilde{L}^3m_1n_1\nu\alpha^2\sum_{t=0}^{\nu-1}\mathbb{E}_{A,\xi}\left[\|X_t-X_0\|_{\mathrm{F}}^2\right] + 66\tilde{L}^3m_1^3n_1^3\epsilon^2\nu\alpha^2 + 12\tilde{\sigma}^2\tilde{L}m_1n_1\alpha^2\nu. \tag{5.29}$$

Equation (5.28) and (5.29) give

$$\begin{aligned}&\mathbb{E}_{A,\xi}[\mathcal{G}_\theta(X_\nu)-\mathcal{G}_\theta(X_0)]\\ &\leq\left(-\frac{\nu\alpha}{4}+28\tilde{L}m_1n_1\nu^2\alpha^2+144\alpha^3\nu^3m_1^2n_1^2\tilde{L}^2\right)\mathbb{E}_{A,\xi}\left[\|\nabla\mathcal{G}_\theta(X_0)\|_{\mathrm{F}}^2\right]\\ &+536\nu^3\tilde{L}^4\alpha^3\epsilon^2m_1^2n_1^2+24\nu^3\tilde{L}^2\alpha^3\tilde{\sigma}^2\\ &+\frac{\tilde{L}^2m_1n_1\epsilon^2\alpha\nu}{2}+66\tilde{L}^3m_1^3n_1^3\epsilon^2\nu\alpha^2+12\tilde{\sigma}^2\tilde{L}m_1n_1\alpha^2\nu.\end{aligned}$$

□

We have established the bound for solving the subproblem (5.17). Next, we investigate the impact of updating θ and resampling B and establish the convergence result for our proposed KronZO algorithm. This leads to the following theorem.

Theorem 6. *Let K the number of subproblems solved and $T = K\nu$ the total number of iterations. Let AssumptionAssumption 5.5.1 be satisfied. Denote $\tilde{L} = \|B\|_{\mathrm{F}}^2 L$ and $\bar{L} = \mathbb{E}_B\left[\tilde{L}\right] = m_2n_2L$. Denote $\tilde{\sigma}^2 = \|B\|_{\mathrm{F}}^2\sigma^2$ and $\bar{\sigma}^2 = \mathbb{E}_B[\tilde{\sigma}^2] = m_2n_2\sigma^2$. If the step condition*

$$0<\alpha<\frac{-28+\sqrt{28^2+144}}{288\bar{L}m_1n_1\nu} \tag{5.30}$$

holds, the iterates $\{\theta_t\}_{t\in[0,T]}$ generated by KronZO respect the following bound:

$$\beta\frac{m_2n_2}{K}\sum_{k=0}^{K-1}\|\nabla\mathcal{L}(\theta_{k\nu})\|_{\mathrm{F}}^2 \leq \frac{\mathcal{L}(\theta_0)-\mathcal{L}^*}{T\alpha} + 536\nu^2\bar{L}^4\alpha^2\epsilon^2m_1^2n_1^2+24\nu^3\bar{L}^2\alpha^3\bar{\sigma}^2 + \frac{\bar{L}^2m_1n_1\epsilon^2}{2}+66\bar{L}^3m_1^3n_1^3\epsilon^2\alpha+12\bar{\sigma}^2\bar{L}m_1n_1\alpha \tag{5.31}$$

where $\beta = \left(\frac{1}{4}-28\bar{L}m_1n_1\nu\alpha-144\alpha^2\nu^2m_1^2n_1^2\bar{L}^2\right)>0$.

Proof. Recalling the update rule for the subproblem, it follows that

$$\mathcal{G}_{\theta_{k\nu}}(X_\nu) = \mathcal{L}(\theta_{k\nu} + X_{(k,\nu)} \otimes B_k) = \mathcal{L}(\theta_{(k+1)\nu}), \quad \mathcal{G}_{\theta_{k\nu}}(X_0) = \mathcal{L}(\theta_{k\nu}).$$

Moreover, note that $\nabla\mathcal{G}_{\theta_{k\nu}}(X_0) = \nabla\mathcal{L}(\theta_{k\nu}) \otimes B_k$. Denote $\bar{L} = \mathbb{E}_B\big[\tilde{L}\big] = m_2 n_2 L$ and $\bar{\sigma}^2 = \mathbb{E}_B[\tilde{\sigma}^2] = m_2 n_2 \sigma^2$. Applying Lemma 9 gives

$$\begin{aligned}\mathbb{E}_{A,B,\xi}\Big[\mathcal{L}(\theta_{(k+1)\nu} - \mathcal{L}(\theta_{k\nu})\Big] \leq &\left(-\frac{\nu\alpha}{4} + 28\bar{L}m_1 n_1 \nu^2\alpha^2 + 144\alpha^3\nu^3 m_1^2 n_1^2 \bar{L}^2\right)\mathbb{E}_{A,\xi}\Big[\|\nabla\mathcal{L}(\theta_{k\nu}) \otimes B_k\|_{\mathrm{F}}^2\Big] \\ &+ 536\nu^3\bar{L}^4\alpha^3\epsilon^2 m_1^2 n_1^2 + 24\nu^3\bar{L}^2\alpha^3\bar{\sigma}^2 \\ &+ \frac{\bar{L}^2 m_1 n_1 \epsilon^2 \alpha\nu}{2} + 66\bar{L}^3 m_1^3 n_1^3 \epsilon^2 \nu\alpha^2 + 12\bar{\sigma}^2\bar{L}m_1 n_1 \alpha^2\nu.\end{aligned}$$

Using identity $\|P \otimes Q\|_{\mathrm{F}}^2 = \|P\|_{\mathrm{F}}^2 \|Q\|_{\mathrm{F}}^2$ and $\mathbb{E}\Big[\|B_k\|_{\mathrm{F}}^2\Big] = m_2 n_2$, dividing by $\nu\alpha K$ on both sides then summing over K and rearranging the terms give

$$\begin{aligned}\beta\frac{m_2 n_2}{K}\sum_{k=0}^{K-1}\|\nabla\mathcal{L}(\theta_{k\nu})\|_{\mathrm{F}}^2 \leq &\frac{\mathcal{L}(\theta_0) - \mathcal{L}^*}{T\alpha} + \\ &+ 536\nu^2\bar{L}^4\alpha^2\epsilon^2 m_1^2 n_1^2 + 24\nu^3\bar{L}^2\alpha^3\bar{\sigma}^2 \\ &+ \frac{\bar{L}^2 m_1 n_1 \epsilon^2}{2} + 66\bar{L}^3 m_1^3 n_1^3 \epsilon^2\alpha + 12\bar{\sigma}^2\bar{L}m_1 n_1\alpha,\end{aligned}$$

which is (5.31). Moreover, letting $a := \bar{L}m_1 n_1 \nu\alpha$, the quantity $\beta = \frac{1}{4} - 28a - 144a^2$ is a concave quadratic function of a with two real roots. Therefore, $\beta > 0$ if and only if

$$0 < a < \frac{-28 + \sqrt{28^2 + 144}}{288},$$

which is equivalent to the step condition

$$0 < \alpha < \frac{-28 + \sqrt{28^2 + 144}}{288\,\bar{L}m_1 n_1 \nu}.$$

□

5.8 Conclusion and future work

We analyzed pretraining stochastic gradients through *stable rank* and *participation ratio*, revealing that the gradients effective dimensionality is higher than suggested by the stable rank alone. Consequently, low-rank ZO methods ignore critical directions in a pretraining context. On the other hand, existing full-rank ZO methods add significant memory or flops overhead. KronZO bridges this gap by pairing (i) a Kronecker parameterization that preserves rich subspaces with (ii) a directional criterion-based update, leading to better convergence. The result is a ZO optimizer that, under equal forward-pass budgets, outperforms MeZO and LoZO while using less memory.

Compared to our preliminary study [4], which focused on small-scale feasibility, KronZO introduces a structured optimizer that, thanks to components (i) and (ii), reduces memory usage and improves convergence at larger scales for an identical step budget.

Although KronZO still trails SGD in final loss, we believe that ZO methods have the potential to reach performance comparable to FO methods during pretraining on medium-sized models, while preserving their exceptionally low memory footprint, at the cost of a higher training time. Downstream evaluation on billion-scale models shows that pretraining with KronZO achieves a level of generalization comparable to that of vanilla SGD.

Practical takeaways

- **Use participation ratio for pretraining.** It provides a faithful estimate of the number of active directions in LMs and should guide the choice of ZO rank.
- **Prefer directional update.** Replacing averaging by a directional test yields systematic gains.
- **Choose KronZO when RAM is scarce.** At fixed memory, it enables training large models more efficiently than full-rank ZO baselines, with strong downstream transferability.

Limitations and future work

KronZO increases the number of forward passes by 50% ($3q$ vs. $2q$ per iteration). While this overhead is acceptable on inexpensive computations (for example on small models), latency-critical scenarios may require further pruning, or better exploiting the $3q$ query budget per step, e.g., via *secant-constrained updates*.

The method is also highly sensitive to the hyper-parameters choice, which were selected via a coarse grid search. Automated tuning methods, such as the Mesh Adaptive Direct Search (MADS) [12], could markedly enhance efficiency.

A natural next step is to investigate techniques for closing the loss gap between FO and KronZO that widens at larger scales, such as ZO-Kronecker momentum, weight decay or ZO parameter ϵ schedule.

We did not evaluate KronZO in the fine-tuning setting; given its strong performance for ZO pretraining, it may also prove effective for fine-tuning tasks.

Finally, aspects such as systematic analysis of robustness to randomness and estimator bias, a clearer runtime profiling, and the exploration of hybrid bias–variance strategies (potentially combining ZO and FO information) all fall outside the scope of this paper but constitute natural directions for future work.

CHAPTER 6 ARTICLE 3: AN INEXACT MODIFIED QUASI-NEWTON METHOD FOR NONSMOOTH REGULARIZED OPTIMIZATION

Nathan Allaire and Sébastien Le Digabel and Dominique Orban

Manuscript submitted on December 16th, 2025, to *SIAM Journal on Scientific Computing* (SISC); currently under review.

Abstract

We introduce method iR2N, a modified proximal quasi-Newton method for minimizing the sum of a $\mathcal{C}^1$ function f and a lower semi-continuous prox-bounded h that permits inexact evaluations of f, ∇f and of the relevant proximal operators. Both f and h may be nonconvex. In applications where the proximal operator of h is not known analytically but can be evaluated via an iterative procedure that can be stopped early, or where the accuracy on f and ∇f can be controlled, iR2N can save significant computational effort and time. At each iteration, iR2N computes a step by approximately minimizing the sum of a quadratic model of f, a model of h, and an adaptive quadratic regularization term that drives global convergence. In our implementation, the step is computed using a variant of the proximal-gradient method that also allows inexact evaluations of the smooth model, its gradient, and proximal operators. We assume that it is possible to interrupt the iterative process used to evaluate proximal operators when the norm of the current iterate is larger than a fraction of that of the minimum-norm optimal step, a weaker condition than others in the literature. Under standard assumptions on the accuracy of f and ∇f, we establish global convergence in the sense that a first-order stationarity measure converges to zero and a worst-case evaluation complexity in $O(\epsilon^{-2})$ to bring said measure below $\epsilon > 0$. Thus, inexact evaluations and proximal operators do not deteriorate asymptotic complexity compared to methods that use exact evaluations. We illustrate the performance of our implementation on problems with ℓ_p-norm, ℓ_p total-variation and the indicator of the nonconvex pseudo p-norm ball as regularizers. On each example, we show how to construct an effective stopping condition for the iterative method used to evaluate the proximal operator that ensures satisfaction of our inexactness assumption. Our results show that iR2N offers great flexibility when exact evaluations are costly or unavailable, and highlight how controlled inexactness can reduce computational effort effectively and significantly.

6.1 Introduction

We consider the problem class

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\, f(x) + h(x), \tag{6.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable, $h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is proper, lower semi-continuous (lsc), and both may be nonconvex. In practice, h, called the *regularizer*, is designed to promote desirable properties in solutions, such as sparsity. We develop method iR2N, a variant of the modified proximal quasi-Newton algorithm R2N of Diouane et al. [40] that allows for inexact evaluations of f and ∇f, as well as of the relevant proximal operators. Among other applications, evaluations of f and ∇f are inexact when they result from the discretization of a differential or integral operator [15], from the sampling of a sum of a large number of terms, as in machine learning applications [81], or from using multiple floating-point systems [71]. Like R2N, iR2N computes a step at each iteration by approximately minimizing the sum of a quadratic model of f, a model of h, and an adaptive quadratic regularization term. The subproblem is solved with method iR2, which is to method R2 of Aravkin et al. [9] as iR2N is to R2N, i.e., proximal operators are evaluated inexactly. Method R2 may be viewed as a variant of the standard proximal-gradient method with adaptive step length, and is a special case of R2N. We consider settings where proximal operators do not have a closed-form expression, and one must thus rely on inexact evaluations. Specifically, we focus on scenarios where proximal operators can be evaluated by running a convergent algorithm that can be terminated early with appropriate guarantees detailed below. Special cases that fit our assumptions include choices of convex and nonconvex h, including the ℓ_p-norm total variation (TV), ℓ_p-norm regularizer and the indicator of the nonconvex ℓ_p-pseudo norm ball with $0 < p < 1$. Method iR2N reduces to R2N when f, ∇f and proximal operators are evaluated exactly. We establish global convergence of iR2N under standard assumptions on the inexactness of f and ∇f, and provided the inexact proximal operator yields a step whose norm is at least a fraction of the norm of an optimal step. We also establish that worst-case evaluation complexity of iR2N is of the same order as that of R2N. Thus, inexact evaluations do not degrade worst-case complexity. Our remaining assumptions are standard. To emphasize our assumptions on inexact evaluations, we simplify those assumptions of [40] that would complicate the analysis. In particular, we assume that ∇f is Lipschitz continuous, but its Lipschitz constant need not be known nor approximated. However, it should be clear that iR2N remains convergent under the more general assumptions of [40] with its worst-case complexity affected accordingly. It should also be clear that minor alterations of our approach would establish that the proximal quasi-Newton trust-region algorithm of Aravkin et al. [8, 10] remains convergent under inexact evaluations and its asymptotic worst-

case complexity is unchanged. Such minor alterations would also establish convergence and complexity of Levenberg-Marquardt variants in the vein of [11] that are useful when f is a least-squares residual.

We report computational experience with the proximal operator of the ℓ_p norm, the total variation in ℓ_p norm, and the indicator of the nonconvex ℓ_p-pseudo norm ball. Each of those proximal operators must be evaluated via an iterative procedure. For each, we devise a stopping condition that ensures satisfaction of our assumption on inexact proximal operator evaluations. Our results show that iR2N offers great flexibility in settings where exact evaluations are costly or unavailable, and highlight how controlled inexactness can be exploited to reduce computational effort effectively and significantly. We provide an efficient Julia implementation of iR2N as part of the open-source package `RegularizedOptimization.jl` [14].

Related Research

Most numerical methods for (6.1) require the evaluation of one or more proximal operators [72] at each iteration. The proximal operator of h with step size $\nu > 0$ at $q \in \mathbb{R}^n$ is

$$\operatorname*{prox}_{\nu h}(q) := \operatorname*{argmin}_{u \in \mathbb{R}^n} \tfrac{1}{2}\|u - q\|^2 + \nu h(u) \subseteq \mathbb{R}^n. \tag{6.2}$$

For given h and q, (6.2) can be empty, a singleton or contain multiple elements, one of which must be identified. Beck [18] and Chierchia et al. [33] summarize the closed-form of (6.2) for a large number of choices of h relevant in applications. The standard proximal-gradient method [42] along with most proximal methods in the literature assume that obtaining an element of (6.2) *exactly* is possible.

For certain choices of h, it is necessary to apply an iterative method to appoximate an element of (6.2), e.g., the total variation (TV) with ℓ_p regularization $h(x) = \|Dx\|_p$, where $p \geq 1$ and D is the upper bidiagonal finite-difference operator with a diagonal of negative ones and a superdiagonal of ones. Finding an element in (6.2) for the TV-ℓ_p can be achieved via the taut-string method [16] or the fast TV denoising method [34]. As in other methods in the literature for various choices of convex h [17, 38, 48], the latter monitor the duality gap between a convex problem and its dual. Those algorithms have guaranteed convergence properties and can be terminated early, i.e., short of optimality. In the above, the evaluation of (6.2) is inexact in the sense that a convergent process to identify a global minimizer is applied and can be stopped short of optimality according to an optimality criterion.

A somewhat more complicated scenario is the algorithm described by Yang et al. [101] for the

case where h is the indicator of the "ball" in pseudo-norm ℓ_p with $p \in (0, 1)$. The evaluation of the proximal operator requires solving a nonconvex problem to global optimality in that case, and their algorithm is not guaranteed to always succeed. We return to this problem in Section 6.4.

Other concepts of inexactness of the proximal operator appear in the literature. For convex h, Rockafellar [83] requires that an approximate solution of (6.2) be a certain distance from the optimal set. Still for convex h, Barré et al. [17] unveil multiple ways to define inexactness by finding a primal-dual point in a certain relaxed subdifferential. Salzo and Villa [86] define three approximations: they compute z such that either *(i)* $\|z - \operatorname{prox}_{\nu h}(q)\| \leq \epsilon$, *(ii)* $\nu^{-1}(q - z)$ lies in a relaxation of the subdifferential of h at z, or *(iii)* $z \in \operatorname{prox}_{\nu h}(q + e)$ with $\|e\| \leq \epsilon$ for some $\epsilon > 0$. Chen et al. [32] extend proximal inexactness by introducing the concept of $(\gamma, \delta, \varepsilon)$-proximal-gradient stationary point (PGSP) for convex h based on the Goldstein subdifferential. The PGSP generalizes the three concepts of [86] by jointly relaxing spatial and functional exactness and directly quantifying the first-order residual, thus also encompassing Rockafellar's [83] and relaxed subgradient formulations within a unified framework. For nonconvex h, Gu et al. [48] say that an element is an inexact solution of (6.2) if its objective value is within ϵ of its optimal value.

To cope with inexact evaluations of the proximal operator, classical schemes must be revised to preserve convergence guarantees. The seminal inexact proximal-point algorithm (iPPA) of Rockafellar [83] allows summably controlled errors in the resolvant computation of a maximal monotone operator and still ensures global convergence with linear/superlinear behavior under suitable parameter growth. Building on the accelerated estimate-sequence framework, Salzo and Villa [86] establish that the accelerated iPPA retains $O(1/k)$ decay under inexactness of type (i) above, and optimal $O(1/k^2)$ decay under inexactness of type (ii). Schmidt et al. [87] establish an $O(1/k)$ rate for proximal-gradient and an $O(1/k^2)$ rate for an accelerated variant under inexactness similar to (iii) above. Extensions include inertial, variable-metric forward–backward schemes with relative inner accuracy and uniform symmetric positive definite metrics [28]; nonconvex inexact (accelerate) proximal gradient with guarantees matching the exact counterparts under calibrated error schedules [48]; adaptive, implementable stopping rules that preserve $O(\epsilon^{-2})$ iteration complexity and enable support identification [38]; and accelerated proximal gradient under relative error criteria that maintain an $O(1/k^2)$ rate [21]. For nonconvex problems, the sequence generated by an inexact proximal-gradient (or splitting) method can still be shown to converge to a first-order critical point under an assumption of type (iii) above on the approximation errors [91]. Finally, for weakly convex functions, recent results establish global convergence for inexact proximal algorithms under inexactness of type (i) above, allowing controlled inexactness in the proximal

steps while maintaining convergence [58].

Notation

The Euclidean norm is $\|\cdot\|$. When required, other norms are denoted with different symbols. We use f, h, m, ϕ, φ, ξ and ψ for functions. Other lowercase Latin letters denote vectors in $\mathbb{R}^n$. Exceptions are p and q, which are standard to denote a pair of dual ℓ_p and ℓ_q norms, and r, which denotes a radius. Uppercase A and B are matrices, L is a Lipschitz constant, and O is used for the Landau notation. Lowercase Greek letters denote scalars. Calligraphic letters denote sets.

6.2 Background

6.2.1 Variational Analysis Concepts

We say that $h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is proper if $h(x) < +\infty$ for at least one $x \in \mathbb{R}^n$ and lower semi-continuous (lsc) at $\bar{x}$ if $\liminf_{x\to\bar{x}} h(x) = h(\bar{x})$. It is lsc if it is lsc at all $\bar{x} \in \mathbb{R}^n$. We say that h is prox-bounded at x if there is $\lambda > 0$ such that $w \mapsto h(w) + \frac{1}{2}\lambda^{-1}\|w - x\|^2$ is bounded below [84, Definition 1.23]. The threshold of prox-boundedness of h at x is the supremum of all such λ at x, and is denoted λ_x. We say that h is *uniformly prox-bounded* if there is $\lambda \in \mathbb{R}_+ \cup \{+\infty\}$ such that $\lambda_x \geq \lambda$ for all $x \in \mathbb{R}^n$.

For $\phi : \mathbb{R}^n \to \mathbb{R} \cup \{\pm\infty\}$ and $\bar{x} \in \mathrm{dom}(\phi)$, the Fréchet subdifferential of ϕ at $\bar{x}$ is

$$\widehat{\partial}\phi(\bar{x}) := \left\{ v \in \mathbb{R}^n \;\middle|\; \liminf_{x\to\bar{x}} \frac{\phi(x) - \phi(\bar{x}) - v^T(x - \bar{x})}{\|x - \bar{x}\|} \geq 0 \right\}.$$

The limiting subdifferential $\partial\phi(\bar{x})$ of ϕ at $\bar{x}$ is the set of elements $v \in \mathbb{R}^n$ such that there exists a sequence $\{x_k\} \to \bar{x}$ with $\{\phi(x_k)\} \to \phi(\bar{x})$, and there exists $v_k \in \widehat{\partial}\phi(x_k)$ for all k such that $\{v_k\} \to v$. It always holds that $\widehat{\partial}\phi(\bar{x}) \subseteq \partial\phi(\bar{x})$.

If ϕ is proper, we say that $\bar{x}$ is stationary for ϕ, or for the problem of minimizing ϕ, if $0 \in \widehat{\partial}\phi(\bar{x})$. If ϕ is proper and has a local minimum at $\bar{x}$, then $\bar{x}$ is stationary for ϕ. In the special case where $\phi = f + h$ with f continuously differentiable and h proper, then $\partial\phi(x) = \nabla f(x) + \partial h(x)$ [84, Theorem 10.1]. We say that $f : \mathbb{R}^n \to \mathbb{R}$ has Lipschitz-continuous gradient with Lipschitz constant $L \geq 0$ if for all x and $s \in \mathbb{R}^n$,

$$|f(x + s) - f(x) - \nabla f(x)^T s| \leq \tfrac{1}{2}L\|s\|^2. \tag{6.3}$$

6.2.2 Models

In this work, we focus on three sources of inexactness: the objective, its gradient and the proximal operator evaluations. We denote $\widehat{f}$ and $\widehat{\nabla} f$ the inexact counterparts of f and ∇f. At each iteration, R2N computes a step s_{cp} defined below that serves to define a stationarity measure and that results from a proximal operator evaluation. Accordingly, in iR2N, we denote its inexact counterpart $\widehat{s}_{\text{cp}}$. We follow [9, 11, 40] and structure the iterations of an algorithm around two sets of models, but, since the only information we have access to is inexact, those are based on $\widehat{f}$ and $\widehat{\nabla} f$.

For $\nu > 0$ and $x \in \mathbb{R}^n$, the first-order models

$$\varphi_{\text{cp}}(s; x) \coloneqq \widehat{f}(x) + \widehat{\nabla} f(x)^T s \tag{6.4}$$

$$\psi(s; x) \approx h(x + s) \tag{6.5}$$

$$m_{\text{cp}}(s; x, \nu^{-1}) \coloneqq \varphi_{\text{cp}}(s; x) + \tfrac{1}{2}\nu^{-1}\|s\|^2 + \psi(s; x) \tag{6.6}$$

serve to generalize the concept of Cauchy point, hence the subscript "cp", where we use the symbol "$\approx$" to mean that the left-hand side is an approximation of the right-hand side. We will be more specific in Assumption 6.3.3 below. The dual role of models (6.4)–(6.6) is to define a threshold for sufficient decrease at each iteration, and to define a measure of approximate stationarity.

For $\sigma > 0$, $x \in \mathbb{R}^n$ and $B(x) = B(x)^T \in \mathbb{R}^{n\times n}$, the second-order models

$$\varphi(s; x) \coloneqq \widehat{f}(x) + \widehat{\nabla} f(x)^T s + \tfrac{1}{2} s^T B(x) s \tag{6.7}$$

$$m(s; x, \sigma) \coloneqq \varphi(s; x) + \tfrac{1}{2}\sigma\|s\|^2 + \psi(s; x), \tag{6.8}$$

are used to compute a step. Because $\varphi_{\text{cp}}(\cdot; x)$ is linear and $\varphi(\cdot; x)$ is quadratic for fixed x, both have globally Lipschitz-continuous gradient.

We follow [9, 11, 40] and require that all models that we consider satisfy the following assumption.

Assumption 6.2.1. *For all $x \in \mathbb{R}^n$, $\psi(\cdot; x)$ is proper, lsc and uniformly prox-bounded. In addition, $\psi(0; x) = h(x)$ and $\partial\psi(0; x) \subseteq \partial h(x)$.*

6.2.3 The Proximal-Gradient Method

The direct generalization of the gradient method to nonsmooth regularized optimization is the proximal-gradient method [42]. For (6.1), the proximal-gradient iteration can be written

$$x_{k+1} = x_k + s_{k,\text{cp}} \tag{6.9}$$

$$\begin{aligned} s_{k,\text{cp}} &\in \operatorname*{argmin}_s \tfrac{1}{2}\nu_k^{-1}\|s + \nu_k \widehat{\nabla} f(x_k)\|^2 + \psi(s; x_k) \\ &= \operatorname*{argmin}_s \widehat{\nabla} f(x_k)^T s + \tfrac{1}{2}\nu_k^{-1}\|s\|^2 + \psi(s; x_k) \\ &= \operatorname*{argmin}_s m_{\text{cp}}(s; x_k, \nu_k^{-1}), \end{aligned} \tag{6.10}$$

where $\nu_k > 0$ is an appropriate step length, though it is typically used with $\psi(s; x_k) := h(x_k + s)$. We call $s_{k,\text{cp}}$ a Cauchy step. It turns out that $s_{k,\text{cp}}$ exists provided ν_k is sufficiently small.

Proposition 1 (84, Theorem 1.25)**.** *Let $\varphi_{\text{cp}}(\cdot; x)$ be as in (6.4), and $\psi(\cdot; x)$ be proper, lsc, prox-bounded with threshold $\lambda_x > 0$ and such that $\psi(0; x) = h(x)$. For any $0 < \nu < \lambda_x$, the set $\operatorname{argmin}_s m_{\text{cp}}(s; x, \nu^{-1})$ is nonempty and compact.*

We denote s_{cp} an element of $\operatorname{argmin}_s m_{\text{cp}}(s; x, \nu^{-1})$ when one exists. When s_{cp} is well defined, the quantity

$$\begin{aligned} \xi_{\text{cp}}(s_{\text{cp}}, x, \nu^{-1}) &:= (\varphi_{\text{cp}} + \psi)(0; x) - (\varphi_{\text{cp}} + \psi)(s_{\text{cp}}; x) \\ &= (\widehat{f} + h)(x) - (\varphi_{\text{cp}} + \psi)(s_{\text{cp}}; x) \end{aligned} \tag{6.11}$$

is central to the algorithm and the analysis, as it is in [9, 11, 40], where it plays the dual role of defining Cauchy decrease and serving as stationarity measure. Indeed, under standard assumptions, x is stationary for (6.1) if $\xi_{\text{cp}}(s_{\text{cp}}; x, \nu^{-1}) = 0$ [40, Lemma 3.5]. We diverge slightly from those references and, for reasons that become clear later, note that $\nu^{-1}\|s_{\text{cp}}\|$ can equally be used as stationarity measure.

Proposition 2. *Let $x \in \mathbb{R}^n$ and $\psi(\cdot; x)$ be proper, lsc, prox-bounded with threshold $\lambda_x > 0$ and such that $\partial\psi(0; x) \subseteq \partial h(x)$. Let $0 < \nu < \lambda_x$ and $s_{\text{cp}} \in \operatorname{argmin}_s m_{\text{cp}}(s; x, \nu^{-1})$. If $s_{\text{cp}} = 0$, then $0 \in \widehat{\nabla} f(x) + \partial h(x)$. If, in addition, $\widehat{\nabla} f(x) = \nabla f(x)$, then x is stationary for (6.1).*

Proof. If $s_{\text{cp}} = 0$, then $\xi_{\text{cp}}(s_{\text{cp}}; x, \nu^{-1}) = 0$ by (6.11). The rest of the proof is identical to that of [40, Lemma 3.5].

□

In the special case $h = 0$, i.e., smooth optimization, $s_{\text{cp}} = -\nu\nabla f(x)$. Thus, we normalize and use $\nu^{-1}\|s_{\text{cp}}\|$ as stationarity measure.

The identification of an s_{cp}, when one exists, coincides with the identification of an element in the image of a proximal operator (6.2), i.e., $s_{\text{cp}} \in \text{prox}_{\nu\psi(\cdot;x)}(-\nu\widehat{\nabla} f(x))$. It is the computation of an element in such a set that represents the main computational challenge in problems for which the set is not known analytically, so that one must resort to an iterative numerical method. In that case, the s_{cp} computed is inexact, and we refer to this situation as an inexact evaluation of the proximal operator.

The following result is hidden inside the proof of [27, Lemma 2].

Proposition 3. *Let f have Lipschitz-continuous gradient with Lipschitz constant $L \geq 0$ and let h be proper, lsc and prox-bounded at $x \in \mathbb{R}^n$ with threshold $\lambda_x > 0$. Let $0 < \nu < \min(1/L, \lambda_x)$, and let $s \in \mathbb{R}^n$ be such that*

$$f(x) + \nabla f(x)^T s + \tfrac{1}{2}\nu^{-1}\|s\|^2 + h(x+s) \leq (f+h)(x). \tag{6.12}$$

Then,

$$(f+h)(x) - (f+h)(x+s) \geq \tfrac{1}{2}(\nu^{-1} - L)\|s\|^2. \tag{6.13}$$

Proof. We inject $f(x) + \nabla f(x)^T s \geq f(x+s) - \frac{1}{2}L\|s\|^2$, which follows from (6.3), into (6.12) and obtain (6.13).

□

Proposition 3 applied to $\varphi_{\text{cp}}(\cdot;x)$, $\psi(\cdot;x)$ and $s_{\text{cp}} \in \text{prox}_{\nu\psi(\cdot;x)}(-\nu\widehat{\nabla} f(x))$, yields

$$\xi_{\text{cp}}(s_{\text{cp}}; x, \nu^{-1}) \geq \tfrac{1}{2}\nu^{-1}\|s_{\text{cp}}\|^2, \tag{6.14}$$

because the Lipschitz constant of $\nabla\varphi_{\text{cp}}(\cdot;x)$ is zero.

By contrast, we denote an approximate Cauchy step resulting from an *inexact* minimization of (6.6) as $\widehat{s}_{\text{cp}}$. We will be more specific about the meaning of inexactness in that context in Assumption 6.3.5. Accordingly, we define

$$\widehat{\xi}_{\text{cp}}(\widehat{s}_{\text{cp}}; x, \nu^{-1}) := (\varphi_{\text{cp}} + \psi)(0;x) - (\varphi_{\text{cp}} + \psi)(\widehat{s}_{\text{cp}}; x). \tag{6.15}$$

Proposition 3 states that (6.13) also holds for any s that produces simple decrease in (6.6); s need not be an exact minimizer. Thus, if we apply a descent procedure to minimize (6.6) starting from $s = 0$, any iterate, denoted generically as $\widehat{s}_{\text{cp}}$, generated by that procedure will satisfy (6.13), i.e.,

$$(\varphi_{\text{cp}} + \psi)(0;x) - (\varphi_{\text{cp}} + \psi)(\widehat{s}_{\text{cp}}; x) \geq \tfrac{1}{2}\nu^{-1}\|\widehat{s}_{\text{cp}}\|^2. \tag{6.16}$$

Thus, an exact minimizer in (6.10) would produce a Cauchy step $s_{k,\text{cp}}$ that satisfies (6.14). For brevity, we write $\xi_{k,\text{cp}} := \xi_{\text{cp}}(s_{k,\text{cp}}; x_k, \nu_k^{-1})$ and $\widehat{\xi}_{k,\text{cp}}$ instead of $\widehat{\xi}_{\text{cp}}(\widehat{s}_{k,\text{cp}}; x_k, \nu_k^{-1})$. The above shows that $\xi_{k,\text{cp}} \geq \frac{1}{2}\nu_k^{-1}\|s_{k,\text{cp}}\|^2$ and $\widehat{\xi}_{k,\text{cp}} \geq \frac{1}{2}\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\|^2$ provided $\widehat{s}_{k,\text{cp}}$ results in simple decrease in (6.6) from $s = 0$.

Proposition 2 indicates that one role of the first-order models (6.4)–(6.6), and hence of $\widehat{s}_{k,\text{cp}}$ and $\widehat{\xi}_{k,\text{cp}}$ is to determine approximate stationarity. The role of the second-order models (6.7)–(6.8) is to allow us to compute a step that improves upon the (inexact) Cauchy step. Minimizing the second-order model is a well-defined problem for all sufficiently large σ_k.

Proposition 4 (40, Proposition 3.3)**.** *Let $\varphi(\cdot; x)$ be defined as in (6.7), and let $\psi(\cdot; x)$ be proper, lsc and prox-bounded with threshold $\lambda_x > 0$ and such that $\psi(0; x) = h(x)$. For any $\sigma > \lambda_x^{-1} - \lambda_{\min}(B(x))$, the set $\operatorname{argmin}_s m(s; x, \sigma)$ is nonempty and compact, where $\lambda_{\min}$ represents the smallest eigenvalue.*

6.3 Algorithm and Convergence Analysis

Our algorithm is a modification of method R2N of Diouane et al. [40]. At a general iteration k, an approximate Cauchy step $\widehat{s}_{k,\text{cp}}$ is computed together with the corresponding value of $\widehat{\xi}_{k,\text{cp}}$ by minimizing (6.6) inexactly. If x_k is not approximately stationary, a step s_k is computed by approximately minimizing (6.8). Because only $\widehat{f}$, and not f, is available, we compute the ratio of achieved versus predicted decrease

$$\widehat{\rho}_k := \frac{\widehat{f}(x_k) + h(x_k) - (\widehat{f}(x_k + s_k) + h(x_k + s_k))}{\varphi(0; x_k) + \psi(0; x_k) - (\varphi(s_k; x_k) + \psi(s_k; x_k))} \tag{6.17}$$

to accept or reject s_k. Acceptance of s_k occurs when $\widehat{\rho}_k \geq \widehat{\eta}_1 > 0$, which indicates that sufficient decrease occurs in $\widehat{f} + h$. The parameters of the algorithm, specifically $\sigma_{\min}$, together with assumptions on the accuracy of $\widehat{f}$, are chosen so that acceptance of s_k also implies that sufficient decrease occurs in $f + h$. We then update σ_k accordingly, as in R2N. All that is required of s_k is that it satisfy a sufficient decrease condition—see Assumption 6.3.4 below. That can be achieved, for instance, by computing $\widehat{s}_{k,\text{cp}}$ from a single (inexact) proximal-gradient iteration on (6.8) with a well-chosen step length ν_k starting from $s = 0$, and computing s_k by continuing the (inexact) proximal-gradient iterations from $\widehat{s}_{k,\text{cp}}$. Should $\|s_k\|$ be much larger than $\|\widehat{s}_{k,\text{cp}}\|$, we reset s_k to $\widehat{s}_{k,\text{cp}}$ as in R2N. The procedure is formally stated as Algorithm 14. We refer the reader to [40] for more background.

Algorithm 14 iR2N

Given $\kappa_f > 0$, $\kappa_\nabla > 0$, choose constants $0 < \gamma_3 \leq 1 < \gamma_1 \leq \gamma_2$, $0 < \widehat{\eta}_1 \leq \widehat{\eta}_2 < 1$.
Choose $0 < \theta_1 < 1 < \theta_2$.
Choose $\sigma_{\min} > 4\kappa_f \theta_1 \theta_2^2 / (\widehat{\eta}_1(1-\theta_1))$ and $\sigma_0 \geq \sigma_{\min}$.
For $k = 0, 1, \ldots$ **do**
  Choose $B_k := B(x_k) \in \mathbb{R}^{n \times n}$ such that $B_k = B_k^T$.
  Set $\nu_k := \theta_1 / (\|B_k\| + \sigma_k)$.
  Repeat
    Compute $\widehat{s}_{k,\text{cp}}$ an approximate solution of $\min_s m_{\text{cp}}(s; x_k, \nu_k^{-1})$ and $\widehat{\xi}_{k,\text{cp}}$.
    Compute a step s_k such that $m(s_k; x_k, \sigma_k) \leq m(\widehat{s}_{k,\text{cp}}; x_k, \sigma_k)$.
    If $\|s_k\| > \theta_2 \|\widehat{s}_{k,\text{cp}}\|$ **then**
      Reset $s_k = \widehat{s}_{k,\text{cp}}$.
    end If
  until $\widehat{f}$ and $\widehat{\nabla} f$ satisfy Assumption 6.3.6.
  Compute the ratio $\widehat{\rho}_k$ as in (6.17).
  If $\widehat{\rho}_k \geq \widehat{\eta}_1$ **then**
    Set $x_{k+1} = x_k + s_k$.
  else
    Set $x_{k+1} = x_k$.
  end If
  Update the regularization parameter according to

$$\sigma_{k+1} \in \begin{cases} [\gamma_3 \sigma_k,\ \sigma_k] & \text{if } \widehat{\rho}_k \geq \widehat{\eta}_2, \quad \text{very successful iteration} \\ [\sigma_k,\ \gamma_1 \sigma_k] & \text{if } \widehat{\eta}_1 \leq \widehat{\rho}_k < \widehat{\eta}_2, \quad \text{successful iteration} \\ [\gamma_1 \sigma_k,\ \gamma_2 \sigma_k] & \text{if } \widehat{\rho}_k < \widehat{\eta}_1 \quad \text{unsuccessful iteration} \end{cases}$$

  Reset $\sigma_{k+1} = \max(\sigma_{k+1},\ \sigma_{\min})$.
end For

6.3.1 Assumptions

Intentionally, our assumptions are not the most general under which convergence of Algorithm 14 can be shown to occur. We have done so in order to highlight the influence of our assumptions on the inexactness of the objective, gradient and proximal operators evaluations on the analysis. We refer the interested reader to [40] for the current most general assumptions. Nonetheless, we expect that our convergence guarantees remain valid under the weaker assumptions, at the cost of a more intricate analysis.

Our first assumption concerns Lipschitz-continuity of the gradient. Technically, this assumption is only necessary for the complexity analysis; convergence can be guaranteed under continuous differentiability only.

Assumption 6.3.1. *∇f is Lipschitz-continuous with constant $L \geq 0$—see (6.3).*

We assume that $\{B_k\}$ is bounded; a common assumption in the literature. Under appropriate growth conditions, convergence is preserved even if $\{B_k\}$ is allowed to grow unbounded [40].

Assumption 6.3.2. *There exists $\kappa_B > 0$ such that $\|B_k\| \leq \kappa_B$ for all k.*

Assumption 6.3.2 is trivially satisfied when $B_k = 0$, as in [9, Algorithm 6.1]. It is also satisfied in [20] where the objective is strongly convex and the model Hessian is defined by a positive definite limited-memory quasi-Newton update. Under standard assumptions, the LBFGS and LSR1 updates satisfy Assumption 6.3.2 [8, 29].

Our next assumption bounds the discrepancy between h and its model ψ.

Assumption 6.3.3. *There exists $\kappa_h > 0$ such that $|\psi(x, s) - h(x + s)| \leq \kappa_h \|s\|^2$ for all x and $s \in \mathbb{R}^n$.*

The bound $\|s\|^2$ in Assumption 6.3.3 can be relaxed to $o(\|s\|)$ [40]. Assumption 6.3.3 is satisfied when $\psi(s; x) = h(x + s)$, and when $h(x) = g(c(x))$ where c is twice continuously differentiable with bounded second derivatives and g is globally Lipschitz continuous if we select $\psi(s; x) = g(c(x) + \nabla c(x)^T s)$.

The next assumption drives the convergence analysis and states that the step s_k computed at iteration k should result in a decrease at least comparable to that induced by the approximate Cauchy step in the first-order model.

Assumption 6.3.4. *There is $\theta_1 \in (0, 1)$ such that $\varphi(0; x) + \psi(0; x) - (\varphi(s_k; x) + \psi(s_k; x)) \geq (1 - \theta_1)\widehat{\xi}_{k,\text{cp}}$ for all k.*

As we now show, Assumption 6.3.4 holds for s_k computed as stated in Algorithm 14.

Lemma 10. *For $\theta_1 \in (0, 1)$ and s_k as in Algorithm 14, Assumption 6.3.4 holds.*

Proof. The proof of [40, Proposition 3] applies with $s = s_k$ and $\widehat{s}_{k,\text{cp}}$ in place of s_{cp}. Indeed, it remains valid for any $s \in \mathbb{R}^n$ and $s_{\text{cp}} \in \mathbb{R}^n$ as long as $m(s; x, \sigma) \leq m(s_{\text{cp}}; x, \sigma)$, which is guaranteed by step 7 of Algorithm 14.

□

We ensure that Step 7 in Algorithm 14 holds because the inexact Cauchy step $\widehat{s}_{k,\text{cp}}$ coincides with the first (inexact) step of the proximal gradient method applied to $m(s; x_k, \sigma_k)$ from $s = 0$ with an appropriate step length ν_k. Therefore, computing s_k by continuing the proximal iterations from $\widehat{s}_{k,\text{cp}}$ leads to further decrease in $m(s; x_k, \sigma_k)$.

The next assumption requires the norm of the computed step $\widehat{s}_{k,\text{cp}}$ to be at least a fraction of that of an exact step $s_{k,\text{cp}}$.

Assumption 6.3.5. *There exists* $\kappa_s \in (0, 1]$ *such that, for all* k*,*

$$\|\widehat{s}_{k,\text{cp}}\| \geq \kappa_s \min\{\|s_{k,\text{cp}}\| \mid s_{k,\text{cp}} \in \operatorname*{prox}_{\nu\psi(\cdot;x_k)}(-\nu_k \widehat{\nabla} f(x_k))\}.$$

In the experiments of Section 6.4, $\psi(\cdot; x_k)$ satisfies the assumptions of Proposition 1 and, therefore, the minimum in Assumption 6.3.5 is well defined.

Assumption 6.3.5 holds when $s_{k,\text{cp}}$ is computed exactly, i.e., $\widehat{s}_{k,\text{cp}} = s_{k,\text{cp}}$. Indeed, let $\|s_{k,\min}\|$ be the smallest norm across all possible choices of $s_{k,\text{cp}}$. Several cases can occur. Firstly, if $\|s_{k,\min}\| > 0$, then $\|s_{k,\text{cp}}\| > 0$ necessarily, and Assumption 6.3.5 holds with $\kappa_s := \min(1, \|s_{k,\text{cp}}\|/\|s_{k,\min}\|)$. If, on the other hand, $\|s_{k,\min}\| = 0$, the same holds if we compute $s_{k,\text{cp}} \neq 0$ but, should we compute $s_{k,\text{cp}} = 0$, Proposition 2 would imply that x_k is stationary and the iterations would stop. This case will be clarified in Lemma 14.

Details on how we satisfy Assumption 6.3.5 when $\widehat{s}_{k,\text{cp}} \neq s_{k,\text{cp}}$ in certain situations relevant in practice can be found in Section 6.4. We further comment on Assumption 6.3.5 in Section 6.6.

In the same fashion as [71], we bound evaluation errors in terms of the step. Similar assumptions are made in [36] in a trust-region context.

Assumption 6.3.6. *There exist* $\kappa_f > 0$ *and* $\kappa_\nabla > 0$ *such that, for all* $k \in \mathbb{N}$*,*

$$|f(x_k) - \widehat{f}(x_k)| \leq \kappa_f \|s_k\|^2, \tag{6.18}$$

$$|f(x_k + s_k) - \widehat{f}(x_k + s_k)| \leq \kappa_f \|s_k\|^2, \tag{6.19}$$

$$\|\nabla f(x_k) - \widehat{\nabla} f(x_k)\| \leq \kappa_\nabla \|s_k\|. \tag{6.20}$$

Finally, we assume that the objective is bounded below, which is only required in the complexity analysis.

Assumption 6.3.7. *There exists* $(f + h)_{\text{low}} \in \mathbb{R}$ *such that* $(f + h)(x) \geq (f + h)_{\text{low}}$ *for all* $x \in \mathbb{R}^n$*.*

6.3.2 Convergence Analysis

Our first result relates the decrease predicted by the model to the step size.

Lemma 11. *Let Assumption 6.3.4 hold. Then,*

$$\varphi(0; x_k) + \psi(0; x_k) - (\varphi(s_k; x_k) + \psi(s_k; x_k)) \geq \tfrac{1}{2}(1 - \theta_1)\theta_2^{-2}\nu_k^{-1}\|s_k\|^2.$$

Proof. Assumption 6.3.4, (6.16) and line 10 of Algorithm 14 yield

$$\begin{aligned}\varphi(0;x_k)+\psi(0;x_k)-(\varphi(s_k;x_k)+\psi(s_k;x_k)) &\geq (1-\theta_1)\widehat{\xi}_{k,\mathrm{cp}}\\ &\geq \tfrac{1}{2}(1-\theta_1)\nu_k^{-1}\|\widehat{s}_{k,\mathrm{cp}}\|^2\\ &\geq \tfrac{1}{2}(1-\theta_1)\theta_2^{-2}\nu_k^{-1}\|s_k\|^2.\end{aligned}$$

□

Our next result mirrors [11, Theorem 4.1] and shows that whenever σ_k exceeds a threshold σ_{succ}, iteration k is very successful and σ_{k+1} decreases.

Lemma 12. *Let Assumptions 6.3.1 to 6.3.4 and 6.3.6 be satisfied and define*

$$\sigma_{\mathrm{succ}} \coloneqq \max\left(\frac{\theta_1\theta_2^2(L+\kappa_B+2\kappa_h+4\kappa_f+2\kappa_\nabla)}{(1-\theta_1)(1-\widehat{\eta}_2)},\ \lambda^{-1}\right) > 0.$$

If, at iteration k of Algorithm 14, $s_k \neq 0$ and $\sigma_k \geq \sigma_{\mathrm{succ}}$, then $\widehat{\rho}_k \geq \widehat{\eta}_2$, and iteration k is very successful.

Proof. As in the proof of [11, Theorem 4.1], σ_k increases as long as it is below $\lambda_{x_k}^{-1}$. Thus, we assume that $\sigma_k \geq \lambda^{-1}$. The definitions of $\widehat{\rho}_k$ and φ, Assumption 6.3.4, the triangle inequality and Lemma 11 yield

$$\begin{aligned}&|\widehat{\rho}_k - 1|\\ &= \frac{|\widehat{f}(x_k+s_k)-\widehat{f}(x_k)-\widehat{\nabla}f(x_k)^Ts_k-\frac{1}{2}s_k^TB_ks_k+h(x_k+s_k)-\psi(s_k;x_k)|}{\varphi(0;x)+\psi(0;x)-(\varphi(s_k;x_k)+\psi(s_k;x_k))}\\ &\leq \frac{|\widehat{f}(x_k+s_k)-\widehat{f}(x_k)-\widehat{\nabla}f(x_k)^Ts_k|+|\frac{1}{2}s_k^TB_ks_k|+|h(x_k+s_k)-\psi(s_k;x_k)|}{\frac{1}{2}(1-\theta_1)\theta_2^{-2}\nu_k^{-1}\|s_k\|^2}.\end{aligned}$$

The triangle inequality along with Assumptions 6.3.1 and 6.3.6 bound the first term in the numerator as

$$\begin{aligned}&|\widehat{f}(x_k+s_k)-\widehat{f}(x_k)-\widehat{\nabla}f(x_k)^Ts_k|\\ &\qquad\leq |f(x_k+s_k)-f(x_k)-\nabla f(x_k)^Ts_k|+2\kappa_f\|s_k\|^2+\kappa_\nabla\|s_k\|^2\\ &\qquad\leq (\tfrac{1}{2}L+2\kappa_f+\kappa_\nabla)\|s_k\|^2.\end{aligned}$$

Assumption 6.3.2 bounds the second term in the numerator by $\frac{1}{2}\|B_k\|\|s_k\|^2 \leq \frac{1}{2}\kappa_B\|s_k\|^2$. Assumption 6.3.3 bounds the last term in the numerator by $\kappa_h\|s_k\|^2$. After simplifying by

$\|s_k\|^2$ and using $\nu_k \leq \theta_1/\sigma_k$ by definition in Algorithm 14, those observations give

$$|\widehat{\rho}_k - 1| \leq \frac{\theta_1\theta_2^2(L + \kappa_B + 2\kappa_h + 4\kappa_f + 2\kappa_\nabla)}{(1-\theta_1)\sigma_k}.$$

Therefore, $\sigma_k \geq \sigma_{\text{succ}}$ implies that $\widehat{\rho}_k \geq \widehat{\eta}_2$.

□

In lemma 12, we showed that $\sigma_k \geq \sigma_{\text{succ}} \implies \widehat{\rho}_k \geq \widehat{\eta}_2$, which means that there is a decrease in $\widehat{f} + h$. Next, we show that there exists $\eta_1 > 0$ such that $\widehat{\rho}_k \geq \widehat{\eta}_1 \implies \rho_k \geq \eta_1$, and similarly for $\widehat{\eta}_2$. Therefore, a decrease also occurs in $f + h$ every time a step is accepted.

Lemma 13. *Let Assumptions 6.3.4 and 6.3.6 hold. At iteration k, denote*

$$\rho_k := \frac{f(x_k) + h(x_k) - (f(x_k + s_k) + h(x_k + s_k))}{\varphi(0; x_k) + \psi(0; x_k) - (\varphi(s_k; x_k) + \psi(s_k; x_k))}$$

the measure of agreement between the actual and predicted decrease in $f + h$. Let $\sigma_{\min}$ be as in Algorithm 14 and

$$\eta_1 := \widehat{\eta}_1 - \frac{4\kappa_f\theta_1\theta_2^2}{(1-\theta_1)\sigma_{\min}} > 0, \qquad \eta_2 := \widehat{\eta}_2 - \frac{4\kappa_f\theta_1\theta_2^2}{(1-\theta_1)\sigma_{\min}} > 0.$$

Then, $\widehat{\rho}_k \geq \widehat{\eta}_1 \implies \rho_k \geq \eta_1$ and $\widehat{\rho}_k \geq \widehat{\eta}_2 \implies \rho_k \geq \eta_2$.

Proof. By definition of $\widehat{\rho}_k$ and ρ_k,

$$\widehat{\rho}_k = \rho_k + \frac{(\widehat{f} - f)(x_k) + (f - \widehat{f})(x_k + s_k)}{(\varphi + \psi)(0; x_k) - (\varphi + \psi)(s_k; x_k)}.$$

Because Algorithm 14 enforces $\sigma_k \geq \sigma_{\min} > 0$, we obtain $\nu_k \leq \theta_1/\sigma_k \leq \theta_1/\sigma_{\min}$. Thus, Lemma 11 and Assumption 6.3.6 give

$$|\widehat{\rho}_k - \rho_k| \leq \frac{2\kappa_f\|s_k\|^2}{\frac{1}{2}(1-\theta_1)\theta_2^{-2}\nu_k^{-1}\|s_k\|^2} \leq \frac{4\kappa_f\theta_1\theta_2^2}{(1-\theta_1)\sigma_{\min}}.$$

Now, if $\widehat{\rho}_k \geq \widehat{\eta}_1$,

$$\rho_k \geq \widehat{\eta}_1 - \frac{4\kappa_f\theta_1\theta_2^2}{(1-\theta_1)\sigma_{\min}} = \eta_1.$$

The lower bound on $\sigma_{\min}$ ensures $\eta_1 > 0$. The same holds for η_2 because $\widehat{\eta}_2 \geq \widehat{\eta}_1$. □

Lemmas 12 and 13 together imply that $\widehat{\rho}_k \geq \widehat{\eta}_1$ guarantees a decrease in $f + h$.

The next result is classic and considers the case where only a finite number of successful iterations occur.

Lemma 14. *Let Assumptions 6.3.1 to 6.3.4 and 6.3.6 be satisfied. Suppose Algorithm 14 generates finitely many successful iterations. Then $x_k = x_\star$ for all k sufficiently large and $x_\star$ is first-order stationary.*

Proof. By assumption, there is $k_0 \in \mathbb{N}$ such that $x_k = x_\star$ for all $k \geq k_0$. If $x_\star$ is not stationary, as of iteration k_0, Algorithm 14 repeatedly computes nonzero steps s_k, all of which are rejected, i.e., $\rho_k < \eta_1$. Thus, for all $k \geq k_0$, $\sigma_{k+1} > \sigma_k$. Hence, for sufficiently large k, $\sigma_k > \sigma_{\text{succ}}$, which triggers a successful iteration, and is absurd.

□

Lemma 12 implies that there exists $\sigma_{\max} = \min(\sigma_0, \gamma_2\sigma_{\text{succ}})$ such that $\sigma_k \leq \sigma_{\max}$ for all $k \in \mathbb{N}$. Consequently, Assumption 6.3.2 yields that for all $k \in \mathbb{N}$,

$$\nu_{\min} \leq \nu_k \leq \nu_{\max}, \quad \nu_{\min} := \theta_1/(\kappa_B + \sigma_{\max}), \quad \nu_{\max} := \theta_1/\sigma_{\min}. \tag{6.21}$$

Let $\epsilon > 0$. We seek a bound on $k_\epsilon := \min\{k \in \mathbb{N} \mid \nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\| < \epsilon\} = |\mathcal{S}(\epsilon)| + |\mathcal{U}(\epsilon)| + 1$, where

$$\mathcal{S}(\epsilon) := \{k \in \mathbb{N} \mid \widehat{\rho}_k \geq \widehat{\eta}_1 \text{ and } k < k_\epsilon\}, \qquad \mathcal{U}(\epsilon) := \{k \in \mathbb{N} \mid \widehat{\rho}_k < \widehat{\eta}_1 \text{ and } k < k_\epsilon\}.$$

Lemma 15. *Let Assumptions 6.3.1 to 6.3.4, 6.3.6 and 6.3.7 be satisfied. Assume that Algorithm 14 generates infinitely many successful iterations. Then,*

$$|\mathcal{S}(\epsilon)| \leq \frac{(f+h)(x_0) - (f+h)_{\text{low}}}{\frac{1}{2}\eta_1(1-\theta_1)\nu_{\min}}\ \epsilon^{-2} := \omega_s\epsilon^{-2},$$

where $\nu_{\min}$ is defined in (6.21).

Proof. Let $k \in \mathcal{S}(\epsilon)$. By definition, $\widehat{\rho}_k \geq \widehat{\eta}_1$, which, by Lemma 13, implies that $\rho_k \geq \eta_1$.

Assumption 6.3.4, (6.21), (6.16) and the fact that $k < k_\epsilon$ then imply

$$\begin{aligned}(f+h)(x_k) - (f+h)(x_k+s_k) &\geq \eta_1((\varphi+\psi)(0;x_k) - (\varphi+\psi)(s_k;x_k)) \\ &\geq \eta_1(1-\theta_1)\widehat{\xi}_{k,\text{cp}} \\ &\geq \tfrac{1}{2}\eta_1(1-\theta_1)\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\|^2 \\ &\geq \tfrac{1}{2}\eta_1(1-\theta_1)\nu_k\epsilon^2 \\ &\geq \tfrac{1}{2}\eta_1(1-\theta_1)\nu_{\min}\epsilon^2.\end{aligned}$$

The rest of the proof is classic and identical to, e.g., [11, Lemma 4.3].

□

It is remarkable that the bound in Lemma 15 is identical to that of the standard R2N, which is more apparent when comparing with [11, Lemma 4.3] than with [40, Theorem 6.4]. The extra factor $\frac{1}{2}$ in the denominator of our bound on $|\mathcal{S}(\epsilon)|$ is due to the fact that we use $\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\|$ as stationarity measure instead of $\nu_k^{-1/2}\widehat{\xi}_{k,\text{cp}}^{1/2}$, as in [11].

Finally, we recover a worst-case complexity bound of the same order as in the analysis with exact proximal operator evaluations. The proof is identical to that of, e.g., [11, Theorem 4.5], and is omitted.

Theorem 7. *Let Assumptions 6.3.1 to 6.3.4, 6.3.6 and 6.3.7 be satisfied. Then,*

$$|\mathcal{S}(\epsilon)| + |\mathcal{U}(\epsilon)| = \left(1 + |\log_{\gamma_1}(\gamma_3)|\right)\omega_s\epsilon^{-2} + \log_{\gamma_1}(\sigma_{\max}/\sigma_0) = O(\epsilon^{-2}),$$

where ω_s is defined in Lemma 15.

Theorem 7 shows that iR2N brings the measure $\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\|$ below ϵ in $O(\epsilon^{-2})$ iterations. That measure is not a stationarity measure because it includes the inexactness on $\widehat{s}_{k,\text{cp}}$. By Assumption 6.3.5, there exists an exact Cauchy step $s_{k_\epsilon,\text{cp}}$ such that

$$\nu_k^{-1}\|s_{k_\epsilon,\text{cp}}\| \leq \kappa_s^{-1}\nu_k^{-1}\|\widehat{s}_{k_\epsilon,\text{cp}}\| < \kappa_s^{-1}\epsilon. \tag{6.22}$$

Thus, if $\nu_k^{-1}\|\widehat{s}_{k_\epsilon,\text{cp}}\|$ is small, $\nu_k^{-1}\|s_{k_\epsilon,\text{cp}}\|$ is comparably small. The next result shows that when the latter occurs, we have identified a near stationary point, and marks the impact of κ_s on the analysis.

Theorem 8. *Let Assumptions 6.3.5 and 6.3.6 be satisfied. Let $\epsilon > 0$ and assume $\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\| < \epsilon$. There exists $s_{k,\text{cp}} \in \text{prox}_{\nu\psi(\cdot;x_k)}(-\nu_k\widehat{\nabla}f(x_k))$ that satisfies Assumption 6.3.5 such that*

$\|s_{k,\text{cp}}\| < \kappa_s^{-1}\nu_{\max}\epsilon$, *and* $u_k \in \nabla f(x_k) + \partial\psi(s_{k,\text{cp}}; x_k)$ *such that*

$$\|u_k\| < \left(\kappa_\nabla\theta_2\nu_{\max} + \kappa_s^{-1}\right)\epsilon. \tag{6.23}$$

Proof. By definition, $s_{k,\text{cp}}$ is an exact minimizer of (6.6), thus

$$\begin{aligned} 0 &\in \widehat{\nabla} f(x_k) + \nu_k^{-1}s_{k,\text{cp}} + \partial\psi(s_{k,\text{cp}}; x_k) \\ &= \nabla f(x_k) + g_k + \nu_k^{-1}s_{k,\text{cp}} + \partial\psi(s_{k,\text{cp}}; x_k), \end{aligned} \tag{6.24}$$

where $g_k \coloneqq \widehat{\nabla} f(x_k) - \nabla f(x_k)$ and $\|g_k\| \leq \kappa_\nabla\|s_k\| \leq \kappa_\nabla\theta_2\|\widehat{s}_{k,\text{cp}}\|$ from Assumption 6.3.6 and line 10 of Algorithm 14. By (6.21) and $\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\| < \epsilon$, $\|\widehat{s}_{k,\text{cp}}\| \leq \nu_k\epsilon < \nu_{\max}\epsilon$. Thus, $\|g_k\| < \kappa_\nabla\theta_2\nu_{\max}\epsilon$.

On the other hand, Assumption 6.3.5 gives

$$\|\nu_k^{-1}s_{k,\text{cp}}\| \leq \kappa_s^{-1}\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\| < \kappa_s^{-1}\epsilon.$$

Now, (6.24) implies that

$$u_k \coloneqq -(g_k + \nu_k^{-1}s_{k,\text{cp}}) \in \nabla f(x_k) + \partial\psi(s_{k,\text{cp}}; x_k).$$

Because $\|u_k\| \leq \|g_k\| + \|\nu_k^{-1}s_{k,\text{cp}}\|$, (6.23) holds. Finally, the same reasoning as above shows that $\|s_{k,\text{cp}}\|$ is bounded as announced.

□

The following results directly from Theorem 7 and mirrors [63, Lemma 3].

Lemma 16. *Under the assumptions of Theorem 7 and Assumption 6.3.5, there exists an infinite index set* $N \subseteq \mathbb{N}$ *and* $\{s_{k,\text{cp}}\}$ *where* $s_{k,\text{cp}} \in \text{prox}_{\nu\psi(\cdot;x_k)}(-\nu_k\widehat{\nabla} f(x_k))$ *for all* k *such that*

1. $\{\widehat{s}_{k,\text{cp}}\}_N \to 0$ *and* $\{s_{k,\text{cp}}\}_N \to 0$,
2. $\{s_k\}_N \to 0$
3. *there exists* $u_k \in \nabla f(x_k) + \partial\psi(s_{k,\text{cp}}; x_k)$ *such that* $\{u_k\}_N \to 0$.

Proof. Claim 1 follows directly from Theorem 7, (6.21) and (6.22). Claim 2 follows from Line 10 of Algorithm 14. Claim 3 results from Theorem 8.

□

We close this section with a result stating that every limit point of the sequence $\{x_k\}_N$ generated by Algorithm 14 is stationary, where N is defined in Lemma 16, under an assumption on the subdifferential of the models $\psi(\cdot; x_k)$.

Recall that for a sequence of sets $\{\mathcal{A}_k\}$ with $\mathcal{A}_k \subseteq \mathbb{R}^n$ for all $k \in \mathbb{N}$, the set $\limsup \mathcal{A}_k$ is the set of limits of all possible convergent sequences $\{a_k\}_N$ with $N \subset \mathbb{N}$ infinite and $a_k \in \mathcal{A}_k$ for all $k \in N$.

Theorem 9. *Under the assumptions of Theorem 7, Assumptions 6.2.1 and 6.3.5, let $N \subseteq \mathbb{N}$ be as in Lemma 16. Assume that $\{x_k\}_N \to \bar{x}$ and that*

$$\limsup_{k \in N} \partial\psi(s_{k,\text{cp}}; x_k) \subseteq \partial\psi(0; \bar{x}). \tag{6.25}$$

Then $\bar{x}$ is stationary for (6.1).

Proof. By our assumptions, Lemma 16, continuity of ∇f and Assumption 6.2.1,

$$0 \in \nabla f(\bar{x}) + \limsup_{k \in N} \partial\psi(s_{k,\text{cp}}; x_k) \subseteq \nabla f(\bar{x}) + \partial\psi(0; \bar{x}) \subseteq \nabla f(\bar{x}) + \partial h(\bar{x}).$$

Thus, $\bar{x}$ is stationary for (6.1).

□

As Leconte and Orban [63] explain, (6.25) holds in several relevant cases, e.g.,

1. each $\psi(\cdot; x_k)$ and $\psi(\cdot; \bar{x})$ are proper, lsc and convex with $\psi(\cdot; x_k) \to \psi(\cdot; \bar{x})$ in the epigraphical sense, and $0 \in \text{dom } \psi(\cdot; \bar{x})$;
2. $\psi(s; x) := h(x + s)$ and $h(x_k + s_{k,\text{cp}}) \to h(\bar{x})$ as would occur, in particular but not exclusively, when h is continuous.

6.4 Evaluation of inexact proximal operators

In this section, we discuss the practical implementation of Algorithm 14 with focus on computing an approximate solution of (6.10) that satisfies Assumption 6.3.5. Our approach is simple: assume that an upper bound $M_k > 0$ on $\|s_{k,\text{cp}}\|$ can be determined based on properties of $\psi(\cdot; x_k)$. Assume also that a descent procedure is applied to (6.10) starting from $s = 0$ that generates iterates $\widehat{s}_j$, $j \geq 0$. Then, stopping the procedure as soon as $\|\widehat{s}_j\| \geq \kappa_s M_k$ ensures that Assumption 6.3.5 holds.

We consider three regularizers whose proximal operators (6.2) are not known analytically and must be computed inexactly:

$$h(x) = \ell_p(x) = \|x\|_p \qquad (1 \le p < \infty), \tag{6.26}$$

$$h(x) = \mathrm{TV}_p(x) = (\sum_i |x_i - x_{i-1}|^p)^{1/p} \qquad (1 \le p < \infty), \tag{6.27}$$

$$h(x) = \chi_{p,r}(x) = \begin{cases} 0 & \text{if } \|x\|_p^p \le r \\ \infty & \text{otherwise} \end{cases} \qquad (0 < p < 1), \tag{6.28}$$

where TV_p is the one-dimensional total-variation operator, and $\chi_{p,r}$ is the indicator of the ℓ_p-pseudo norm "ball" of radius $r^{1/p}$ for $r > 0$.

The next lemmas derive bounds on the norm of solutions to the proximal problems associated with those regularizers.

Lemma 17. *Let h be given by (6.26) and $\psi(s; x_k) := h(x_k + s)$ with $s \in \mathbb{R}^n$. The unique solution $s_{k,\mathrm{cp}}$ of (6.10) is such that*

$$\|s_{k,\mathrm{cp}}\| \le \begin{cases} \nu_k(\|\widehat{\nabla} f(x_k)\| + n^{1/p-1/2}) & (1 \le p < 2) \\ \nu_k(\|\widehat{\nabla} f(x_k)\| + 1) & (p \ge 2). \end{cases} \tag{6.29}$$

Proof. Since $\psi(\cdot; x_k)$ is convex, (6.10) is strongly convex and, therefore, has a unique solution $s_{k,\mathrm{cp}}$. The necessary optimality conditions read

$$\widehat{\nabla} f(x_k) + \nu_k^{-1} s_{k,\mathrm{cp}} + u_k = 0, \qquad u_k \in \partial\psi(s_{k,\mathrm{cp}}; x_k).$$

Here, $\partial\psi(s_{k,\mathrm{cp}}; x_k) = \{u \in \mathbb{R}^n \mid \|u\|_q \le 1 \text{ and } u^T(s_{k,\mathrm{cp}} + x_k) = \|s_{k,\mathrm{cp}} + x\|_p\}$, where q is such that $1/p + 1/q = 1$. By equivalence of norms,

$$\|u_k\| \le n^{1/2-1/q} \|u_k\|_q \le n^{1/2-1/q} = n^{1/p-1/2}.$$

When $1 \le p \le 2$, the latter bound is attained for $u_k := (n^{-1/q}, n^{-1/q}, \ldots, n^{-1/q})$ with $\|u_k\|_q = 1$. When $p > 2$, the bound simplifies to $\|u_k\| \le 1$, which is attained for $u_k := (1, 0, \ldots, 0)$. Thus, $\|s_{k,\mathrm{cp}}\| = \nu_k \|\widehat{\nabla} f(x_k) + u_k\| \le \nu_k(\|\widehat{\nabla} f(x_k)\| + \|u_k\|)$, which yields (6.29).

□

The next result helps bound solutions of (6.10) when h is given by (6.27), but is more general, which is why it is stated separately.

Lemma 18. *Let $A \in \mathbb{R}^{m\times n}$, $h(x) \coloneqq \|Ax\|_\bullet$ where $\|\cdot\|_\bullet$ is a norm on $\mathbb{R}^m$, and $\psi(s; x_k) \coloneqq h(x_k + s)$. The unique solution $s_{k,\text{cp}}$ of (6.10) satisfies*

$$\|s_{k,\text{cp}}\| \leq \nu_k \left(\|\widehat{\nabla} f(x_k)\| + \|A\| \, \|u_k\|\right), \tag{6.30}$$

where $u_k \in \partial\|A(x_k + s_{k,\text{cp}})\|_\bullet$.

Proof. Here again, $s_{k,\text{cp}}$ is unique by strong convexity of (6.10). For $\eta(y) \coloneqq \|y\|_\bullet$,

$$\partial\eta(y) = \{u \in \mathbb{R}^m \mid \|u\|_\star \leq 1 \text{ and } u^T y = \|y\|_\bullet\},$$

where $\|\cdot\|_\star$ is the dual norm of $\|\cdot\|_\bullet$. By [82, Theorem 23, 9], $\partial\psi(s; x_k) = A^T\partial\eta(A(x_k + s))$. Thus, the first-order optimality conditions of (6.10) imply

$$0 \in \widehat{\nabla} f(x_k) + \nu_k^{-1} s_{k,\text{cp}} + A^T u_k,$$

where $u_k \in \partial\eta(A(x_k + s_{k,\text{cp}}))$. We extract $s_{k,\text{cp}} = -\nu_k(\widehat{\nabla} f(x_k) + A^T u_k)$, and $\|s_{k,\text{cp}}\| \leq \nu_k(\|\widehat{\nabla} f(x_k)\| + \|A^T\| \, \|u_k\|)$, which is (6.30) since $\|A\| = \|A^T\|$.

□

Lemma 18 does not state a bound on $\|u_k\|$ as one would depend on $\|\cdot\|_\bullet$ and the bound $\|u_k\|_\star \leq 1$. The next corollary applies Lemma 18 to (6.27).

Corollary 1. *Let h be as in (6.27) and $\psi(s; x_k) \coloneqq h(x_k + s)$. The unique solution $s_{k,\text{cp}}$ of (6.10) satisfies*

$$\|s_{k,\text{cp}}\| \leq \begin{cases} \nu_k \left(\|\widehat{\nabla} f(x_k)\| + 2\sin\left(\frac{\pi(n-1)}{2n}\right) n^{1/p-1/2}\right) & (1 \leq p < 2) \\ \nu_k \left(\|\widehat{\nabla} f(x_k)\| + 2\sin\left(\frac{\pi(n-1)}{2n}\right)\right) & (p \geq 2). \end{cases} \tag{6.31}$$

Proof. Apply Lemma 18 with $\|\cdot\|_\bullet = \|\cdot\|_p$ and

$$A \coloneqq \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{(n-1)\times n}.$$

Note that $A^T A$ is the centered finite-difference operator for second derivatives, which is symmetric, tridiagonal and Toeplitz. Its eigenvalues are thus known in closed form, hence the value of $\|A\|$ [89, p. 54]. Finally, $\|u_k\|$ can be bounded as in the proof of Lemma 17. □

The final lemma derives a bound on the solution of the proximal problem associated to the indicator function in (6.28).

Lemma 19. *Let h be as in (6.28) and $\psi(s; x_k) := h(x_k + s)$. Any solution $s_{k,\text{cp}}$ of (6.10) satisfies*

$$\|s_{k,\text{cp}}\| \leq r^{1/p} + \|x_k\|. \tag{6.32}$$

Proof. Because $0 < p < 1$, $t \mapsto t^p$ is concave for $t \geq 0$, and thus subadditive, i.e., $(a+b)^p \leq a^p + b^p$ for any $a, b \geq 0$. Let $u \in \mathbb{R}^n$. By recurrence on n, $\|u\|_p^p = \sum_{i=1}^n |u_i|^p \geq (\sum_{i=1}^n |u_i|)^p$, which states that $\|u\|_p \geq \|u\|_1$. This implies that the unit "ball" in ℓ_p-pseudo-norm is a subset of the unit ℓ_1-norm ball. In turn, the latter is a subset of the unit ℓ_2-norm ball. A scaling argument shows that the same holds with balls of radius $r > 0$. Therefore, because $\|x_k + s_{k,\text{cp}}\|_p \leq r^{1/p}$, we have $\|x_k + s_{k,\text{cp}}\| \leq r^{1/p}$. The triangle inequality yields $\|s_{k,\text{cp}}\| \leq \|x_k + s_{k,\text{cp}}\| + \|x_k\| \leq r^{1/p} + \|x_k\|$.

□

In (6.29), (6.31) and (6.32), the bound on $\|s_{k,\text{cp}}\|$ depends only on known quantities at iteration k. Thus, we can enforce Assumption 6.3.5 by stopping the inexact proximal procedure as soon as $\|\widehat{s}_{k,\text{cp}}^{(j)}\|$ exceeds a fixed fraction of said bound.

6.5 Numerical experiments

In this section, we present numerical experiments indicating that exploiting inexact objective values, gradients and proximal operators can reduce computational cost substantially. We implement Algorithm 14 in the Julia language [23] as a modification of the R2N solver [40] in [14].

The implementation of the proximal operator of (6.26) and (6.27), which are both convex, is available from the Julia interface [3] to the `proxTV` library [16]. Both implement iterative methods. The method for (6.26) computes projected quasi-Newton search directions, and performs a backtracking line search to determine the step size. That for (6.27) alternates between gradient projection into the ℓ_p-norm ball and Frank-Wolfe steps. After each update, the primal solution is reconstructed from the dual variable, and a new gradient is computed.

Our implementation of the proximal operator of (6.28) is based on the Iteratively Reweighted ℓ_p-Ball Projection (IRBP) scheme of [101]. At each iteration, IRBP approximates the ℓ_p-"ball" norm via a weighted linearization of the nonconvex set around the current iterate. This results in a convex subproblem describing a projection into a weighted ℓ_1-norm ball,

which can be solved efficiently [35]. A smoothing vector is maintained and adaptively updated to avoid numerical instability and improve convergence. The nonconvex nature of $\chi_{p,r}$ implies that there may be non-global minima or saddle points [101]. Therefore, the step output by $\chi_{p,r}$ may not even induce $\widehat{\xi}_{k,\text{cp}} \geq 0$. To the best of our knowledge, there is currently no procedure that is guaranteed to determine a global minimum. In order to mitigate the issue, we implement a multi-start strategy to increase the odds that $\widehat{s}_{k,\text{cp}}$ be a global solution. Our strategy is not always successful, but nevertheless often results in acceptable steps. Part of future work is to find a procedure that identifies a global minimizer. Our implementation is available from [2].

In each case, inexactness in the proximal operator evaluations is controlled by $0 < \kappa_s \leq 1$ in Assumption 6.3.5. For $\kappa_s \approx 0$, the expectation on the quality of $\widehat{s}_{k,\text{cp}}$ is at its lowest, i.e., Assumption 6.3.5 is easiest to satisfy, but (6.33) is harder to reach. Thus the solver may spend less time inside each (cheap) proximal operator evaluation at the cost of potentially performing more (costly) outer iterations. On the other hand, when $\kappa_s \approx 1$, the $\widehat{s}_{k,\text{cp}}$ should be close to an exact solution. In this case, the solver may spend more time than necessary inside each proximal operator evaluation, which may adversely affect the total solution time. In our experiments, we vary the value of κ_s to assess the impact of the inexactness on the performance of iR2N.

Step 9 in Algorithm 14 is performed by a special case of Algorithm 14 with $B = 0$ in which the proximal step computation is the only subproblem. In effect, that is a variant of the R2 algorithm [9, Algorithm 6.1] extended to the inexact proximal framework. We refer to this variant as iR2. Although iR2 is also allowed to perform inexact evaluations of its smooth objective and gradient, we evaluate the quadratic model $\varphi(s; x_k)$ exactly in our experiments.

Each procedure to solve (6.26)–(6.28) comes with its original stopping condition. We say that we run iR2N in *exact* mode when we use this original stopping condition, independently of Assumption 6.3.5, and we consider that the resulting proximal operator is then evaluated exactly. By contrast, we run iR2N in *inexact* mode when the iterations of the proximal operator evaluation are terminated as soon as either (i) $\|\widehat{s}_{k,\text{cp}}\| \geq \kappa_s M_k$, where M_k is the upper bound on $\|s_{k,\text{cp}}\|$ given in (6.29), (6.31), or (6.32), or (ii) the original stopping condition of the proximal operator evaluation is met. In proximal operator evaluations, iR2 uses the same value of κ_s as iR2N.

Inequalities (6.22) suggest using $\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\| \leq \kappa_s \epsilon$ as stopping criterion in Algorithm 14, since it guarantees that $\nu_k^{-1}\|s_{k,\text{cp}}\| \leq \epsilon$. However, we will see that small values of κ_s yield the best performance but make that stopping condition overly stringent. In addition, the bound M_k given in Lemmas 17 to 19 need not be tight, and could indeed be quite loose. For those

reasons, all our experiments use the simple stopping condition

$$\nu_k^{-1} \|\widehat{s}_{k,\text{cp}}\| \leq \epsilon. \tag{6.33}$$

In the next sections, we report the performance of iR2N on problems that use the inexact proximal operators described above. In Sections 6.5.1 to 6.5.3, both the objective and gradient are assumed to be evaluated exactly, i.e., only subject to the limits of floating-point operations. In Section 6.5.4, we consider inexact evaluations of the objective and gradient. All our tests are performed in double precision on a 2020 MacBook Air with an M1 chip (8-core CPU, 8 GB unified memory).

Because f in our test problems is based on randomly-generated data, we average the statistics over 10 runs. It is useful to keep in mind that each iR2N and iR2 iteration evaluates a single proximal operator—see Line (8) of Algorithm 14. Tables in the next sections use the following headers: "κ_s" is the value of the inexactness parameter in Assumption 6.3.5, "iR2N" is the average number of outer iterations, "iR2" is the average number of inner iterations per outer iteration, "prox" is the average number of iterations per proximal operator evaluation, and "time (s)" is the average CPU solution time in seconds.

6.5.1 Basis pursuit denoising problem (BPDN)

The BPDN problem is stated as

$$\min_{x \in \mathbb{R}^n} \tfrac{1}{2}\|Ax - b\|_2^2 + \mu \|x\|_p, \tag{6.34}$$

where $\mu = 10^{-1}$, $A \in \mathbb{R}^{200\times 512}$ is random with orthonormal rows, $b = A\bar{x} + \varepsilon$, $\bar{x}$ has 10 nonzeros, and ε is a noise vector from a normal $(0, 1)$ distribution. We use $p = 1.1$ to attempt to recover a sparse solution. In (6.33), we set $\epsilon = 10^{-6}$.

Table 6.1 Statistics on (6.34) for several values of κ_s.

κ_s	iR2N	iR2	prox	time (s)
1.00e−07	1.61e+01	1.21e+02	1.02e+02	5.03e+00
1.00e−05	1.57e+01	1.63e+02	1.90e+02	9.80e+00
1.00e−03	1.49e+01	1.33e+01	4.02e+02	1.55e+01
1.00e−02	1.49e+01	1.78e+01	6.02e+02	1.77e+01
1.00e−01	1.45e+01	1.39e+01	5.81e+02	1.32e+01
5.00e−01	1.45e+01	1.37e+01	5.90e+02	1.28e+01
9.00e−01	1.45e+01	1.39e+01	5.80e+02	1.25e+01
9.90e−01	1.46e+01	1.37e+01	5.90e+02	1.38e+01
exact mode	1.45e+01	1.35e+01	5.68e+02	1.20e+01

Table 6.1 shows that the average number of iR2N/iR2 iterations decreases globally as κ_s increases. The proximal operator iterations increase as κ_s increases, as expected. For small values of κ_s, inexact mode yields a substantial reduction in the number of proximal iterations and solution time compared with exact mode at the expense of a modest increase in outer iterations. For large values of κ_s the behavior of iR2N is close to that of exact mode.

Figure 6.1 shows that the solutions produced in exact and inexact mode are essentially identical, and that both recover the sparse support of $\bar{x}$.

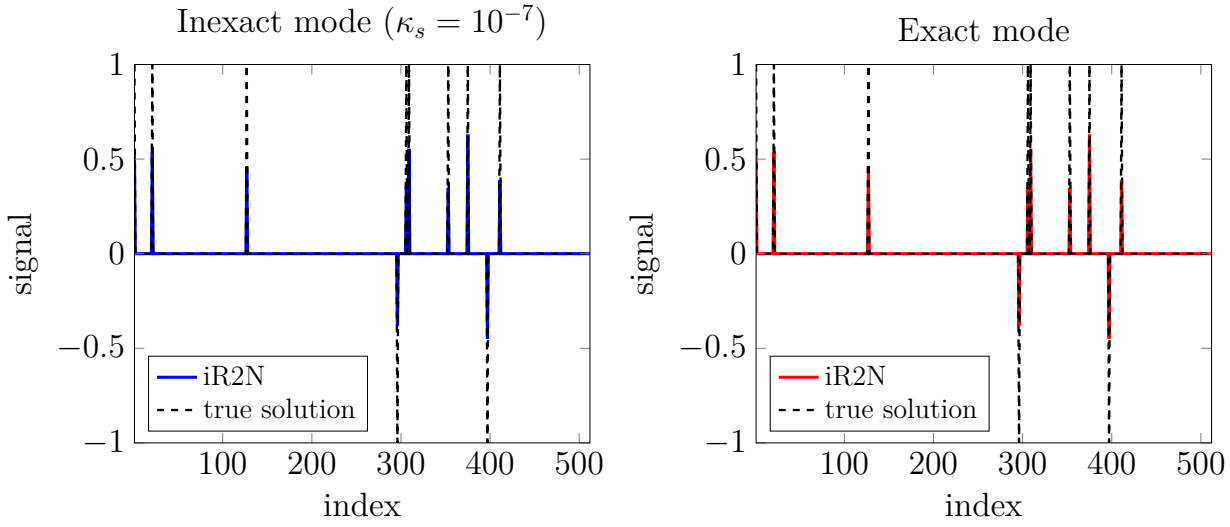

Figure 6.1 Components of the solution of (6.34) found by iR2N and of $\bar{x}$.

6.5.2 Matrix completion problem

The problem is stated as

$$\min_{x \in \mathbb{R}^n} \tfrac{1}{2} \|P(x - A)\|_F^2 + \mu \mathrm{TV}_p(x), \tag{6.35}$$

where $\mu = 10^{-1}$, $p = 1.1$ and $A \in \mathbb{R}^{10 \times 12}$ is a fixed matrix representing an image and the operator P only retains a subset of pixels. In (6.33), $\epsilon = 10^{-3}$.

Table 6.2 gathers our results on (6.35). The benefits of choosing κ_s small are similar to those in Table 6.1. Figure 6.2 shows that the reconstruction error with the solutions of exact and inexact mode are close, as is the discrepancy between the two solutions.

Table 6.2 Statistics on (6.35) for several values of κ_s.

κ_s	iR2N	iR2	prox	time (s)
1.00e−07	3.69e+01	3.41e+02	5.88e+02	9.46e+01
1.00e−05	3.72e+01	3.03e+02	8.71e+02	1.42e+02
1.00e−03	3.69e+01	2.09e+02	3.76e+03	3.54e+02
1.00e−02	3.77e+01	2.12e+02	4.06e+03	3.73e+02
1.00e−01	3.41e+01	1.90e+02	4.37e+03	3.25e+02
5.00e−01	3.56e+01	2.19e+02	4.31e+03	3.54e+02
9.00e−01	3.77e+01	1.81e+02	4.49e+03	3.57e+02
9.90e−01	3.55e+01	2.01e+02	4.27e+03	3.54e+02
exact mode	3.18e+01	1.67e+02	4.49e+03	3.36e+02

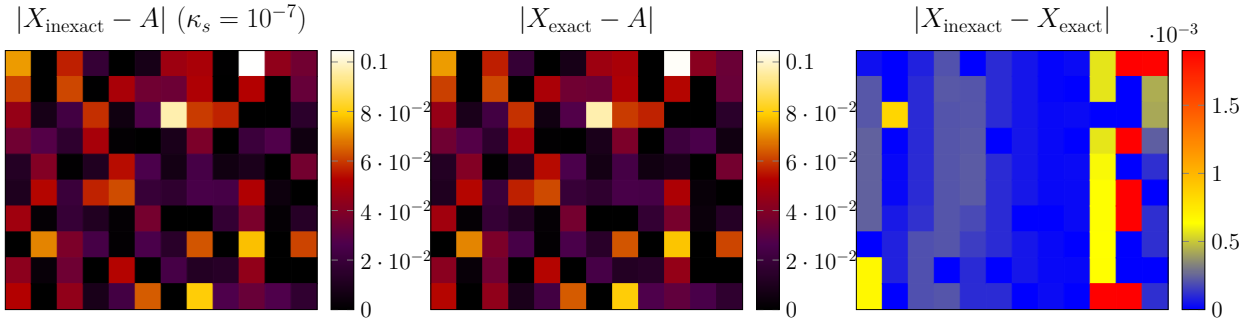

Figure 6.2 Left: Heatmap of the difference between the solution X found by iR2N in inexact and exact mode, and A. Right: Difference between the two solutions. The values masked by P are set to zero and shown in black.

6.5.3 Fitzhugh-Nagumo inverse problem

The FitzHugh–Nagumo system is a simplified representation of a neuron's action potential modeled by the system of differential equations

$$V'(t) = x_2^{-1}(V(t) - \tfrac{1}{3}V(t)^3 - W(t) + x_1), \qquad W'(t) = x_2(x_3 V(t) - x_4 W(t) + x_5). \quad (6.36)$$

We use initial conditions $V(0) = 2$ and $W(0) = 0$, and generate data $\bar{v}(x), \bar{w}(x)$ by solving (6.36) with $\bar{x} = (0, 0.2, 1, 0, 0)$, which corresponds to the Van der Pol oscillator, to which we add random noise. We then aim to recover $\bar{x}$ by minimizing the misfit while encouraging a sparse solution:

$$\min_{x \in \mathbb{R}^5} \tfrac{1}{2} \|F(x)\|_2^2 + \chi_{p,r}(x), \quad (6.37)$$

where $p = 0.5$, $r = 2$, $F : \mathbb{R}^5 \to \mathbb{R}^{2n+2}$, $F(x) := (v(x) - \bar{v}(x), w(x) - \bar{w}(x))$, and $v(x) = (v_1(x), \ldots, v_{n+1}(x))$ and $w(x) = (w_1(x), \ldots, w_{n+1}(x))$ are sampled values of V and W at $n+1$ discretization points. We set $\epsilon = 10^{-5}$ in (6.33). Table 6.3 reports our results.

Table 6.3 Statistics on (6.37) for $p = \frac{1}{2}$ and $r = 2$ with several values of κ_s.

κ_s	iR2N	iR2	prox	time (s)
1.00e−07	5.14e+02	4.90e+02	3.51e−01	5.28e+00
1.00e−05	5.72e+02	4.64e+02	4.62e−01	5.21e+00
1.00e−03	6.31e+02	5.47e+02	5.96e−01	5.56e+00
1.00e−02	5.71e+02	4.81e+02	6.22e−01	5.17e+00
1.00e−01	4.95e+02	4.89e+02	4.11e−01	5.85e+00
5.00e−01	4.90e+02	4.59e+02	1.94e+00	6.42e+00
9.00e−01	5.12e+02	4.98e+02	2.06e+00	6.53e+00
9.90e−01	5.24e+02	5.09e+02	1.91e+00	6.84e+00
exact mode	4.92e+02	5.03e+02	3.92e+01	6.88e+00

The small number of iterations per proximal call arises from the fact that $\chi_{p,r}$ is an indicator; the projection of a point that already belongs to the set requires zero iterations. The value of κ_s has little effect on the number of iR2N/iR2 iterations. As in Sections 6.5.1 and 6.5.2, inexact mode yields a reduction in computational cost, though more modest because the smooth objective and its gradient are costlier in (6.37) than in (6.34) or (6.35). Thus, the reduction in proximal evaluations must outweigh the increase in outer iterations. Table 6.4 gives the approximate solution identified by the exact and inexact variants, and the final value of the smooth objective. Although both exact and inexact mode recover a solution that has one more nonzero than $\bar{x}$, the final smooth objective values are close to that at $\bar{x}$. Figure 6.3 plots the simulation of (6.36) with parameters found by iR2N with $\kappa_s = 1.0\text{e}{-07}$ when solving (6.37). The solutions with exact and inexact mode are indistinguishable.

Table 6.4 Approximate solutions of (6.37) found by the exact and inexact variants with $\kappa_s = 1.0\text{e}{-07}$. The last column shows the smooth objective value at the solution.

	x					$\frac{1}{2}\|F(x)\|^2$
True	0.00e+00	2.00e−01	1.00e+00	0.00e+00	0.00e+00	8.82e−01
Inexact	0.00e+00	2.00e−01	9.98e−01	−1.00e−02	0.00e+00	8.96e−01
Exact	0.00e+00	2.00e−01	9.98e−01	−1.00e−02	0.00e+00	8.96e−01

6.5.4 Inexact objective and gradient evaluations

We now consider inexact evaluations of the smooth objective and its gradient. In (6.37), each evaluation of F involves solving an ODE system numerically, which inherently depends on a stopping tolerance that introduces an approximation error. We use the Verner [97] 9/8 optimal Runge-Kutta method as implemented in [79]. In our implementation of F, the accuracy of the ODE solve can be adjusted via a parameter `prec` > 0 that sets the absolute and relative stopping tolerances. The gradient is computed via automatic differentiation, and

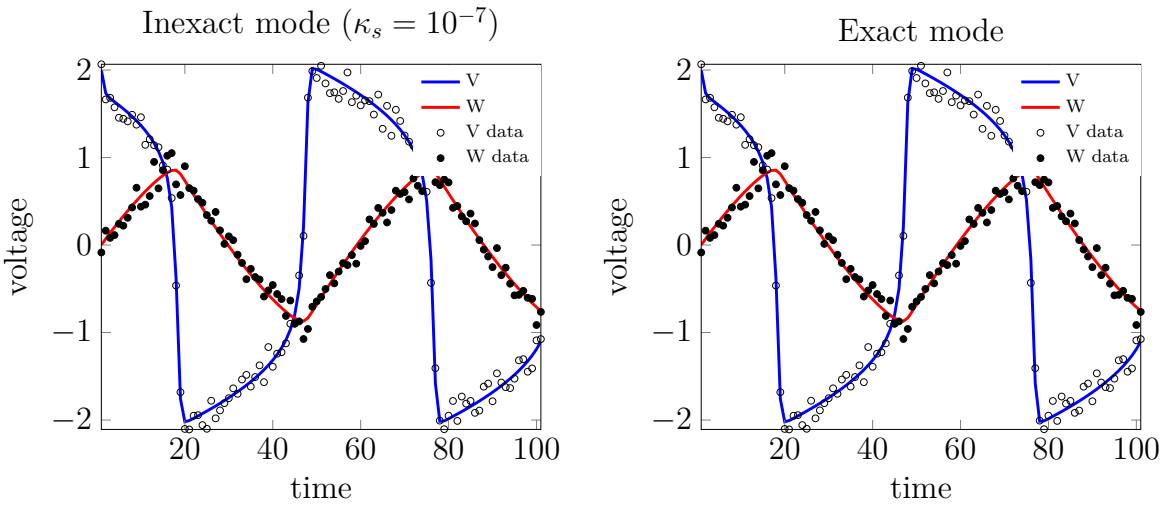

Figure 6.3 Simulation of (6.36) with solutions of (6.37) found by iR2N.

hence, its accuracy also depends on `prec`. Decreasing this tolerance improves the accuracy of the objective and gradient but increases the computational cost. The results of Section 6.5.3 used $\texttt{prec} = 10^{-14}$ as the reference "exact" objective and gradient evaluations.

Because Assumption 6.3.6 may not be easily verifiable in practice, we propose a heuristic inspired from trust-region methods for derivative-free optimization [37, chapter 10], that consists in adapting the accuracy based on the progress of the algorithm. More precisely, we increase the accuracy on unsuccessful iterations, i.e., $\rho_k < \eta_1$ in Algorithm 14. At iteration k, we set `prec` to

$$\texttt{prec}(k) := \max(10^{-3} \exp(\log(10^{-14}/10^{-3})\, n_F/N),\, 10^{-14}), \tag{6.38}$$

where N is a preset maximum number of unsuccessful iterations after which $\texttt{prec} = 10^{-14}$ is always used, and n_F counts the number of unsuccessful iterations. Small values of N lead to a rapid increase in accuracy, whereas larger ones maintain low-accuracy evaluations over more iterations. Though (6.38) may not guarantee Assumption 6.3.6 at every iteration, the objective and gradient accuracy improves as the algorithm progresses, as required by the assumption.

We focus on (6.37) with the setting of Section 6.5.3 and we use (6.38) for inexact objective and gradient. We vary the value of N with fixed $\kappa_s = 10^{-7}$ in Table 6.5.

The first line of Table 6.5 reports the number of iterations and the solution time obtained with "exact" objective and gradient. Lines 2–5 use (6.38) for several values of N. As N increases, iR2N spends a larger fraction of its iterations in a *low*-precision regime, making it

Table 6.5 Iterations and time on (6.37) with inexact objective and gradient evaluations.

N	fail rate	iter iR2N	iter iR2	prox	time (s)
exact F	0%	5.14e+02	4.90e+02	3.51e−01	5.28e+00
20	0%	5.66e+02	5.10e+02	4.55e−01	5.16e+00
50	20%	6.36e+02	5.07e+02	3.77e−01	4.31e+00
100	30%	6.31e+02	5.08e+02	3.46e−01	3.27e+00
200	80%	6.67e+02	5.47e+02	3.69e−01	2.46e+00

increasingly likely that Assumption 6.3.6 is violated. When iR2N operates with insufficient accuracy for too long, the algorithm may eventually stall, cease to make progress, and reach the maximum number of allowed iterations. The second column of Table 6.5 reports the proportion of such failed runs over ten trials. Importantly, the iteration and timing statistics shown in Table 6.5 correspond *only* to the successful runs. The failure rate increases with N, and for $N = 200$ few runs succeed. Moderate values of N yield significant benefits in terms of solution time.

In Table 6.6, we report the performance of Algorithm 14 using inexact objective, gradient and proximal operator evaluations following rule (6.38) on (6.37) with $N = 100$. The number of iR2N, iR2 and proximal iterations is globally unaffected by inexact evaluations, but the latter yield significant savings in terms of solution time.

Table 6.6 Statistics on (6.37) with increasing accuracy given by (6.38) with $N = 100$ and several values of κ_s. Each entry reports the multiplicative gain or loss compared to the reference values in Table 6.3. A value smaller than 1 indicates a gain.

κ_s	iR2N	iR2	prox	time (s)
1.00e−07	1.23e+00	1.04e+00	9.90e−01	6.20e−01
1.00e−05	1.08e+00	1.02e+00	1.38e+00	4.90e−01
1.00e−03	8.40e−01	7.70e−01	5.50e−01	2.70e−01
1.00e−02	1.00e+00	1.00e+00	1.10e+00	3.60e−01
1.00e−01	1.11e+00	9.60e−01	5.20e−01	3.00e−01
5.00e−01	9.90e−01	9.20e−01	1.19e+00	2.50e−01
9.00e−01	1.03e+00	8.80e−01	1.17e+00	3.00e−01
9.90e−01	9.40e−01	8.30e−01	1.36e+00	2.50e−01
average factor	1.03e+00	9.30e−01	1.03e+00	3.60e−01

6.6 Discussion

Method iR2N subsumes R2N [40] by allowing inexact evaluations of the objective, its gradient, and the proximal operator. Under usual global convergence conditions, we showed that inexact evaluations and proximal operators do not deteriorate asymptotic complexity compared to methods that use exact evaluations. Our assumptions on the inexactness of f

and ∇f are standard.

Assumption 6.3.5 on the inexact evaluation of proximal operators differs in nature from Definitions (ii) and (iii) of [86]. Their Definition (i), also used in [83], can be written $\|\widehat{s}_{k,\text{cp}} - s_{k,\text{cp}}\| \leq \epsilon_k$ for at least one $s_{k,\text{cp}}$, where $\{\epsilon_k\}$ is positive and summable. It is equivalent to $\|s_{k,\text{cp}}\| - \epsilon_k \leq \|\widehat{s}_{k,\text{cp}}\| \leq \|s_{k,\text{cp}}\| + \epsilon_k$, which is strictly stronger than Assumption 6.3.5 in that we only require one of the inequalities. Moreover, we use the specific value $\epsilon_k = (1 - \kappa_s)\|s_{k,\text{cp}}\|$, which need not be summable. Indeed, by the same reasoning as in the proof of Lemma 15, for any successful iteration k, there exists a Cauchy step $s_{k,\text{cp}}$ such that

$$\begin{aligned}(f+h)(x_k) - (f+h)(x_k + s_k) &\geq \tfrac{1}{2}\eta_1(1-\theta_1)\nu_k^{-1}\|\widehat{s}_{k,\text{cp}}\|^2 \\ &\geq \tfrac{1}{2}\eta_1(1-\theta_1)\nu_{\max}^{-1}\|\widehat{s}_{k,\text{cp}}\|^2 \\ &\geq \tfrac{1}{2}\eta_1(1-\theta_1)\nu_{\max}^{-1}\kappa_s^2\|s_{k,\text{cp}}\|^2.\end{aligned}$$

Therefore, if we sum those inequalities over the set $\mathcal{S}$ of all successful iterations and use Assumption 6.3.7, we obtain

$$(f+h)(x_0) - (f+h)_{\text{low}} \geq \tfrac{1}{2}\eta_1(1-\theta_1)\nu_{\max}^{-1}\kappa_s^2 \sum_{k\in\mathcal{S}} \|s_{k,\text{cp}}\|^2.$$

A similar inequality holds for $\widehat{s}_{k,\text{cp}}$. Thus, both $\{\widehat{s}_{k,\text{cp}}\}$ and $\{s_{k,\text{cp}}\}$ are square summable. However, showing that they are summable appears to require the stronger Kurdyka-Łojasiewicz assumption [27, Theorem 1], which is not used in our analysis.

iR2N naturally generalizes the special cases R2 [9] with $B(x) = 0$, R2DH [40] with $B(x)$ diagonal, and LM [11] when f is a squared residual norm and $B(x) = J(x)J(x)^T$, where $J(x)$ is the residual Jacobian. It stands to reason that the same mechanisms can be used to extend the trust-region variants (TR [9], TRDH [64], and LMTR [11]) to inexact evaluations and proximal operators with minimal modifications.

Numerical experiments confirm that iR2N provides substantial flexibility in contexts where exact evaluations are expensive or unavailable, and demonstrate that controlled inexactness can be leveraged to reduce computational cost without compromising convergence behavior.

In the context of trust-region methods for (6.1), Aravkin et al. [9, 11] give procedures based on the solution of a nonlinear equation to obtain an element of (6.6) with the additional constraint $\|s\|_\infty \leq \Delta$, where $\Delta > 0$, or, equivalently, with the additional term $\chi(s \mid \Delta\mathbf{B}_\infty)$ in the objective, where $\mathbf{B}_\infty$ is the ℓ_∞-norm unit ball and χ is the indicator of a set. They do so for two choices of ψ. Our results apply directly to both regularizers, and indeed to any regularizer combined with a trust-region constraint. Here, $\mathbf{B}_2 \subset \mathbf{B}_\infty$, and hence,

$\|s_{k,\text{cp}}\|_2 \leq \Delta$. Thus, we may use the stopping condition $\|\widehat{s}_{k,\text{cp}}\| \geq \kappa_s \Delta$.

Future work will focus on allowing inexact evaluations of the quadratic model (6.7), particularly regarding B_k, which itself may be computed inexactly—for instance, when represented in reduced numerical precision or when linear systems involving B_k are solved approximately.

CHAPTER 7 GENERAL DISCUSSION

7.1 Overview of the contributions

The contributions of this thesis can be summarized by the following question: *"Can we design optimization methods that are robust to inexact evaluations and capable of exploiting them to reduce computational costs?"*

The first contribution of this thesis was to demonstrate that it is indeed possible to train language models using zeroth-order methods (Chapter 4). This work, carried out in collaboration with Huawei Technologies' Noah's Ark Lab, introduced me to applied research in artificial intelligence and allowed me to appreciate the vast range of opportunities this field offers.

This work, awarded the Best Industrial Paper at the ICPRAM 2025 conference, led to an invitation to submit an extended version to the journal *Springer Nature Computer Science* (SNCS). Since the numerical implementations belonged to Huawei, we had to rebuild everything from scratch for this extension, which gave us the opportunity to explore new ideas and adopt different approaches. The outcome of this effort (Chapter 5) is KronZO, an original and efficient method for training language models using zeroth-order techniques. We established its theoretical convergence rate, comparable to other zeroth-order methods [31], while showing superior training efficiency and lower memory consumption.

Finally, the last contribution of this thesis is the generalization of the R2N method of Diouane et al. [40] to the case where the objective, its gradient, and the proximal operator are evaluated inexactly: the iR2N method. Theoretical results show that iR2N converges to a stationary point in $\mathcal{O}(\epsilon^{-2})$ iterations, matching the bound obtained in the exact evaluation case. Numerical experiments confirm that using inexact evaluations can significantly reduce computational costs without compromising convergence.

7.2 Limitations of the Proposed Approach

Regarding Chapters 4 and 5, the main limitation lies in *scaling* to very large models. Although KronZO already outperforms the method presented in Chapter 4 and can train models with up to one billion parameters, its efficiency in terms of objective reduction remains below that of first-order methods. Tomorrow's large language models will reach the *trillion*-parameter scale. At that scale, it becomes necessary to rethink zeroth-order methods to make them

more performant while retaining their remarkable memory efficiency.

In Chapter 6, we focused on three cases for which we derive an upper bound on the optimal step. Part of the future work is to extend this analysis to other regularizers whose proximal operator is evaluated inexactly.

7.3 Future Directions and Perspectives

If there is one thing this PhD has taught me, it is that the deeper one digs into a subject, the farther the bottom seems to recede — and that one must learn to admire this infinite depth.

An initial improvement for KronZO would be to implement a variant with *momentum*, as this has proven effective for other zeroth-order methods [31, 70]. Moreover, instead of fixing the parameter ϵ in (5.14), it would be interesting to adapt this parameter dynamically, similar to a trust-region radius. Finally, as discussed in this section, the main challenge remains improving the scalability of KronZO to enhance its performance on very large models while preserving its memory advantage.

For iR2N, two natural research directions emerge: first, to extend the theoretical analysis to trust-region variants; and second, to enrich the library of inexact proximal operators already available and to derive an upper bound on the optimal step for each.

Lastly, recent works [32] combining zeroth-order methods and proximal operators open promising avenues at the intersection of the contributions presented in Chapters 5 and 6.

REFERENCES

[1] IEEE standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[2] N. Allaire and D. Orban. IRBP.jl an efficient numerical algorithm for solving the euclidean projection of a vector onto the nonconvex ℓ_p-"ball". `https://github.com/nathanemac/IRBP.jl`, September 2024.

[3] N. Allaire, D. Orban, and A. Montoison. ProxTV.jl: Wrapper for general total variation regularization. `https://github.com/JuliaSmoothOptimizers/ProxTV.jl`, September 2024.

[4] N. Allaire, M. Ghazvini Nejad, S. Le Digabel, and V. Partovi Nia. Zeroth order optimization for pretraining language models. In *Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM*, pages 113–121, Porto, Portugal, 2025. SciTePress.

[5] N. Allaire, S. Le Digabel, and D. Orban. An inexact modified quasi-Newton method for nonsmooth regularized optimization. Les Cahiers du GERAD G-2024-73, Groupe d'études et de recherche en analyse des décisions, GERAD, Montréal QC H3T 2A7, Canada, 2025.

[6] N. Allaire, S. Le Digabel, D. Orban, and V. Partovi Nia. Zeroth-order Kronecker optimization for pretraining language models. Les Cahiers du GERAD G-2024-44, Groupe d'études et de recherche en analyse des décisions, GERAD, Montréal QC H3T 2A7, Canada, 2025.

[7] S.-i. Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993. ISSN 0925-2312.

[8] A. Aravkin, R. Baraldi, and D. Orban. A proximal quasi-Newton trust-region method for nonsmooth regularized optimization, 2021. Preliminary technical report.

[9] A. Aravkin, R. Baraldi, and D. Orban. A proximal quasi-Newton trust-region method for nonsmooth regularized optimization. *SIAM J. Optim.*, (2):900–929, 2022.

[10] A. Aravkin, R. Baraldi, G. Leconte, and D. Orban. Corrigendum: A proximal quasi-Newton trust-region method for nonsmooth regularized optimization. Cahier G-2021-12-SM, GERAD, Montréal, QC, Canada, 2024.

[11] A. Y. Aravkin, R. Baraldi, and D. Orban. A Levenberg-Marquardt method for nonsmooth regularized least squares. *SIAM J. Sci. Comput.*, 46(4):A2557–A2581, 2024.

[12] C. Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[13] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer Verlag, 2017.

[14] R. Baraldi, G. Leconte, and D. Orban. RegularizedOptimization.jl: Algorithms for regularized optimization. `https://github.com/JuliaSmoothOptimizers/RegularizedOptimization.jl`, September 2024.

[15] R. J. Baraldi and D. P. Kouri. A proximal trust-region method for nonsmooth optimization with inexact function and gradient evaluations. *Math. Program.*, 201(1):559–598, 2023.

[16] A. Barbero and S. Sra. Modular proximal optimization for multidimensional total-variation regularization. *J. Mach. Learn Res.*, 19(56):1–82, 2018.

[17] M. Barré, A. B. Taylor, and F. Bach. Principled analyses and design of first-order methods with inexact proximal operators. *Math. Program.*, 201(1):185–230, 2023.

[18] A. Beck. *First-Order Methods in Optimization*. SIAM, Philadelphia, USA, 2017.

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

[20] S. Becker, J. Fadili, and P. Ochs. On quasi-Newton forward-backward splitting: Proximal calculus and convergence. *SIAM J. Optim.*, 29(4):2445–2481, 2019.

[21] Y. Bello-Cruz, M. L. Gonçalves, and N. Krislock. On inexact accelerated proximal gradient methods with relative error rules, 2020.

[22] L. Ben Allal, A. Lozhkov, G. Penedo, T. Wolf, and L. von Werra. Cosmopedia, 2024.

[23] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Rev.*, 59(1):65–98, 2017.

[24] S. Biderman, L. Gao, J. Lehman, E. Hallahan, C. Zue, S. Gray, K. McDonell, J. Phang, S. Weinbach, and E. Pavlick. Openwebtext corpus. `https://skylion007.github.io/OpenWebTextCorpus/`, 2019.

[25] E. G. Birgin, J. L. Gardenghi, J. M. Martínez, S. A. Santos, and P. L. Toint. Worst-case evaluation complexity for unconstrained nonlinear optimization using high-order regularized models. *Mathematical Programming*, 163(1):359–368, 2017.

[26] J. R. Blum. Multidimensional stochastic approximation methods. *The annals of mathematical statistics*, pages 737–744, 1954.

[27] J. Bolte, S. Sabach, and M. Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Math. Program.*, (146):459–494, 2014.

[28] S. Bonettini, S. Rebegoldi, and V. Ruggiero. Inertial variable metric techniques for the inexact forward–backward algorithm. *SIAM*, 40(5):A3180–A3210, 2018.

[29] O. Burdakov, L. Gong, S. Zikrin, and Y.Yuan. On efficiently combining limited-memory and trust-region techniques. *Math. Program. Comp.*, 9(1):101–134, 2017.

[30] C. Cartis, N. I. M. Gould, and P. L. Toint. *Evaluation Complexity of Algorithms for Nonconvex Optimization: Theory, Computation and Perspectives*. SIAM, Philadelphia, PA, 2022.

[31] Y. Chen, Y. Zhang, L. Cao, K. Yuan, and Z. Wen. Enhancing zeroth-order fine-tuning for language models with low-rank structures, 2024.

[32] Z. Chen, P. Yu, and H. Huang. Zeroth-order methods for stochastic nonconvex nonsmooth composite optimization, 2025.

[33] G. Chierchia, E. Chouzenoux, P. Combettes, and J.-C. Pesquet. *The proximity operator repository*, 2020. `http://proximity-operator.net/download/guide.pdf`. Accessed, September 2024.

[34] L. Condat. A direct algorithm for 1-d total variation denoising. 20(11):1054–1057, 2013.

[35] L. Condat. Fast projection onto the simplex and the ℓ_1-ball. *Math. Program.*, 158(1):575–585, 2016.

[36] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust Region Methods*. SIAM, Philadelphia, PA, 2000. MPS-SIAM Series on Optimization.

[37] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. MPS-SIAM Series on Optimization. SIAM, 2009.

[38] Y. Dai and D. P. Robinson. Inexact proximal-gradient methods with support identification. *arXiv preprint arXiv:2211.02214*, 2022.

[39] A. Dhurandhar, T. Pedapati, A. Balakrishnan, P.-Y. Chen, K. Shanmugam, and R. Puri. Model agnostic contrastive explanations for structured data, 2019.

[40] Y. Diouane, M. L. Habiboullah, and D. Orban. A proximal modified quasi-Newton method for non-smooth regularized optimization. Cahier du GERAD G-2024-64, GERAD, Montréal, Canada, Sept. 2024.

[41] J. C. Duchi, M. I. Jordan, M. J. Wainwright, and A. Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61 (5):2788–2806, 2015.

[42] M. Fukushima and H. Mine. A generalized proximal point algorithm for certain non-convex minimization problems. 12:989–1000, 1981.

[43] P. Gao, E. Trautmann, B. Yu, G. Santhanam, S. Ryu, K. Shenoy, and S. Ganguli. A theory of multineuronal dimensionality, dynamics and measurement. *bioRxiv*, 2017.

[44] T. Gautam, Y. Park, H. Zhou, P. Raman, and W. Ha. Variance-reduced zeroth-order methods for fine-tuning language models. 2024.

[45] S. Ghadimi and G. Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[46] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. In G. K. Thiruvathukal, Y.-H. Lu, J. Kim, Y. Chen, and B. Chen, editors, *Low-Power Computer Vision: Improve the Efficiency of Artificial Intelligence*, pages 291–326. Chapman & Hall/CRC, Taylor & Francis Group, Boca Raton, FL, 2022. Chap. 13.

[47] A. Graham. *Kronecker products and matrix calculus with applications*. Courier Dover Publications, Mineola, NY, 2018.

[48] B. Gu, D. Wang, Z. Huo, and H. Huang. Inexact proximal gradient methods for non-convex and non-smooth optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[49] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, hosted in Seattle, WA, 2020. Association for Computational Linguistics.

[50] M. Hameed, A. Mosleh, M. Tahaei, and V. Partovi Nia. SeKron: A decomposition method supporting many factorization structures, 2022.

[51] O. C. Herfindahl. *Concentration in the Steel Industry*. PhD thesis, Columbia University, 1950. https://archive.org/details/herfindahl-concentration-in-the-steel-industry-1950-publish. PhD thesis; accessible via Internet Archive (published online 2021).

[52] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, USA, second edition, 2002.

[53] G. Hinton. Lecture 6a - overview of mini-batch gradient descent. Coursera: Neural Networks for Machine Learning, 2012. University of Toronto.

[54] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models, 2021.

[55] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin. Black-box adversarial attacks with limited queries and information, 2018.

[56] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models, 2020.

[57] A. Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2023. GitHub repository.

[58] P. D. Khanh, B. S. Mordukhovich, V. T. Phat, and D. B. Tran. Inexact proximal methods for weakly convex functions. *Journal of Global Optimization*, 91(3):611–646, 2025.

[59] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[60] D. Lakhmiri, S. Le Digabel, and C. Tribes. HyperNOMAD: Hyperparameter Optimization of Deep Neural Networks Using Mesh Adaptive Direct Search. *ACM Transactions on Mathematical Software*, 47(3), 2021.

[61] D. Lakhmiri, D. Orban, and A. Lodi. A stochastic proximal method for nonsmooth regularized finite sum optimization. Cahier G-2022-27, GERAD, Montréal, QC, Canada, June 2022.

[62] B. W. Larsen, S. Fort, N. Becker, and S. Ganguli. How many degrees of freedom do we need to train deep networks: a loss landscape perspective, 2022.

[63] G. Leconte and D. Orban. Complexity of trust-region methods with unbounded Hessian approximations for smooth and nonsmooth optimization. Cahier G-2023-65, GERAD, Montréal, QC, Canada, 2023.

[64] G. Leconte and D. Orban. The indefinite proximal gradient method. *Comput. Optim. Appl.*, 91(2): 861–903, 2025.

[65] E. Lehmann and G. Casella. *Theory of Point Estimation*. Springer Texts in Statistics. Springer New York, 1998. ISBN 9780387985022. https://books.google.ca/books?id=9St7DCbu9AUC.

[66] C. Li, H. Farkhoor, R. Liu, and J. Yosinski. Measuring the intrinsic dimension of objective landscapes, 2018.

[67] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1–3):503–528, 1989.

[68] S. Liu, B. Kailkhura, P.-Y. Chen, P. Ting, S. Chang, and L. Amini. Zeroth-order stochastic variance reduction for nonconvex optimization, 2018.

[69] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[70] S. Malladi, T. Gao, E. Nichani, A. Damian, J. Lee, D. Chen, and S. Arora. Fine-tuning language models with just forward passes, 2024.

[71] D. Monnet and D. Orban. A multi-precision quadratic regularization method for unconstrained optimization with rounding error analysis. *Comput. Optim. Appl.*, 91:997–1031, 2025.

[72] J.-J. Moreau. Proximité et dualité dans un espace hilbertien. 93:273–299, 1965.

[73] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[74] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35 (151):773–782, 1980.

[75] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, second edition, 2006.

[76] 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, and Z. DeVito. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[77] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[78] M. Prazeres and A. M. Oberman. Stochastic gradient descent with polyak's learning rate. *Journal of Scientific Computing*, 89(1):25, 2021.

[79] C. Rackauckas and Q. Nie. DifferentialEquations.jl – A performant and feature-rich ecosystem for solving differential equations in Julia. 5(1), 2017.

[80] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 2019. Technical report, OpenAI.

[81] H. Robbins and S. Monro. A stochastic approximation method. 22(3):400–407, 1951.

[82] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[83] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM J. Control Optim.*, 14(5):877–898, 1976.

[84] R. T. Rockafellar and R. Wets. *Variational Analysis*, volume 317 of *Grundlehren der mathematischen Wissenschaften*. Springer Verlag, 2009.

[85] S. M. Ross. *Simulation*. academic press, 2022.

[86] S. Salzo and S. Villa. Inexact and accelerated proximal point algorithms. 19(4):1167–1192, 2012.

[87] M. Schmidt, N. Roux, and F. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[88] A. J. Shepherd. *Second-order methods for neural networks: Fast and reliable training methods for multi-layer perceptrons*. Springer Science & Business Media, 2012.

[89] G. D. Smith. *Numerical solution of partial differential equations: finite difference methods*. Number 1 in Oxford Applied Mathematics and Computing Science Series. Oxford University Press, Oxford, England, third edition, 1985.

[90] J. C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[91] S. Sra. Scalable nonconvex inexact proximal splitting. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[92] L. Tao et al. Transfer learning: A friendly introduction. *Journal of Big Data*, 2022.

[93] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

[94] C. Tribes, S. Benarroch-Lelong, P. Lu, and I. Kobyzev. Hyperparameter Optimization for Large Language Model Instruction-Tuning. Technical Report G-2023-62, Les cahiers du GERAD, 2023.

[95] C.-C. Tu, P. Ting, P.-Y. Chen, S. Liu, H. Zhang, J. Yi, C.-J. Hsieh, and S.-M. Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks, 2020.

[96] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, 2017.

[97] J. H. Verner. Numerically optimal Runge–Kutta pairs with interpolants. *Numer. Algor.*, 53(2–3): 383–396, 2010.

[98] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

[99] P. Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969.

[100] X. Wu, R. Ward, and L. Bottou. Wngrad: Learn the learning rate in gradient descent, 2020.

[101] X. Yang, J. Wang, and H. Wang. Towards an efficient approach for the nonconvex ℓ_p ball projection: algorithm and analysis. *J. Mach. Learn Res.*, 23(101):1–31, 2022.

[102] A. Zeng, X. Liu, Z. Du, Z. Wang, H. Lai, M. Ding, Z. Yang, Y. Xu, W. Zheng, X. Xia, W. L. Tam, Z. Ma, Y. Xue, J. Zhai, W. Chen, P. Zhang, Y. Dong, and J. Tang. GLM-130B: An open bilingual pre-trained model, 2023.

[103] L. Zhang, B. Li, K. Thekumparampil, S. Oh, M. Muehlebach, and N. He. Zeroth-order optimization finds flat minima, 2025.

[104] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

[105] Y. Zhang, P. Li, J. Hong, J. Li, Y. Zhang, W. Zheng, P.-Y. Chen, J. D. Lee, W. Yin, M. Hong, Z. Wang, S. Liu, and T. Chen. Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark, 2024.

[106] P. Zhao, S. Liu, P.-Y. Chen, N. Hoang, K. Xu, B. Kailkhura, and X. Lin. On the design of black-box adversarial examples by leveraging gradient-free optimization and operator splitting method, 2019.