

Computational Nonsmooth Analysis

Modern Applications and Recent Developments

Lai Tian, Anthony Man-Cho So

INTRODUCTION

Deep learning has emerged as a cornerstone of modern machine learning, driving remarkable progress in a wide range of applications. The theoretical understanding of deep learning has garnered increasing attention, with optimization theory playing a central role in many recent breakthroughs. Some important results such as lazy training [1, Section 12.4], landscape analysis [1, Section 12.3], and implicit regularization [1, Section 12.1] have been significantly influenced by optimization techniques.

Classical optimization algorithms typically assume that the objective function is smooth or, in the case of nonsmooth objectives, exhibits highly favorable properties, such as convexity or simple decomposable structure, that make analysis more tractable. Deep learning objectives, by contrast, can be nonsmooth in far more intricate ways, due to nonsmooth components (e.g., ReLU and max-pooling), nonsmooth losses (e.g., hinge), and nonsmooth min-max formulations (e.g., GAN training), all compounded by nonconvexity and compositional architectures. These characteristic nonsmooth elements, while improving model performance to some extent, also introduce technical challenges.

Certain specialized analyses that leverage structural properties have yielded remarkable insights; however, a more general and systematic approach to understanding and training nonsmooth deep learning models necessitates the application of techniques from nonsmooth analysis and optimization. Nonsmooth analysis [2], which originated in convex analysis [3] and developed as a subdiscipline of variational analysis [4], provides powerful tools for studying nonsmooth objectives but often poses significant computational challenges. In particular, the computation of fundamental objects such as subdifferentials and generalized gradients is typically far more intricate than in the smooth case.

In contrast to many existing papers and surveys that focus primarily on nonconvexity in optimization for modern systems, this article emphasizes the interplay between nonconvexity and nonsmoothness, with an explicitly algorithmic perspective. Specifically, we aim to spotlight a distinct area of growing importance, which we refer to as “computational nonsmooth analysis,” mimicking its convex counterpart [5]. This

Special Issue Area Editor email: SPM-SI-AREA@LISTSERV.IEEE.ORG.

Manuscript received July 1, 2024.

field has seen rapid development on many fronts recently, driven by the increasing use of nonsmooth models in modern deep learning and signal processing. This survey aims to introduce some of these recent developments, highlighting general-purpose techniques that can offer new perspectives and opportunities for analyzing, designing, and understanding algorithms for both existing and future deep learning models. Our discussion centers on three representative tasks that are fundamental to optimizing modern nonsmooth problems, while noting that space constraints prevent us from covering several additional topics.

(SUB)GRADIENT METHODS AND THREE COMPUTATIONAL PROBLEMS

For large-scale optimization problems, when first-order information about the objective function is readily available or can be computed easily, first-order methods, such as (sub)gradient-type methods, are the *de facto* workhorses, especially for training deep neural networks, which typically involves optimizing nonconvex, high-dimensional problems. When we say (sub)gradient-type methods, we are referring to procedures that determine the next iterate according to a linear combination of past iterates and (sub)gradients evaluated along the way. Notable examples include the vanilla (sub)gradient method, Adam, Nesterov’s acceleration, AdaGrad, RMSProp, etc. While the training of many learning models involves optimizing functions that are simultaneously nonconvex and nonsmooth, these (sub)gradient-type methods were originally proposed and analyzed for convex or smooth objectives. Hence, towards an optimization theory that is generically applicable to cutting-edge deep learning models and prepares for the future, we may need to consider concepts and ideas beyond conventional smooth and convex analysis.

To set the stage, let us begin by recalling the classic analysis of gradient and subgradient methods when either smoothness or convexity is present. In this process, we will isolate and highlight three basic problems, some of whose computations might be taken for granted or even overlooked when favorable properties (e.g., smoothness or convexity) are present. However, as the discussion proceeds, it will become clear that when these properties are lacking, the three computational problems, while being of fundamental importance, can be highly nontrivial or even intractable.

Gradient Method for Smooth Functions

Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a β -smooth function in the sense that f is differentiable everywhere, and the gradient mapping $\nabla f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is β -Lipschitz continuous, namely, $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2 \leq \beta\|\mathbf{x} - \mathbf{y}\|_2$ for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. If the function f is unbounded from below, then minimizing f can be meaningless since $\inf f = -\infty$ and $\emptyset = \operatorname{argmin} f$. Therefore, it is natural to assume that f is bounded from below.

Starting from any initial point $\mathbf{x}_0 \in \mathbb{R}^d$ and adopting the fixed constant step size $1/\beta$, the (vanilla) gradient method generates iterates according to the following rule:

$$\mathbf{x}_{t+1} := \mathbf{x}_t - (\beta^{-1}) \cdot \nabla f(\mathbf{x}_t), \quad \forall t \geq 0. \quad (1)$$

Since the function f is β -smooth, using a basic property of β -smoothness (a.k.a. the descent lemma; see [6, Lemma 1.2.3]), we obtain $f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t) - \frac{1}{2\beta} \|\nabla f(\mathbf{x}_t)\|_2^2$ for any $t \geq 0$, which, by a telescoping summation from $t = 0$ to T , implies that

$$\min_{0 \leq t \leq T} \|\nabla f(\mathbf{x}_t)\|_2 \leq \sqrt{\frac{1}{T+1} \sum_{t=0}^T \|\nabla f(\mathbf{x}_t)\|_2^2} \leq \sqrt{\frac{2\beta(f(\mathbf{x}_0) - \inf f)}{T+1}} < \infty. \quad (2)$$

The result in (2) is usually stated as an $O(\beta/\varepsilon^2)$ convergence rate for computing an ε -stationary point of f ; i.e., a point \mathbf{x} satisfying $\|\nabla f(\mathbf{x})\|_2 \leq \varepsilon$. Now, let us look a bit more closely into the implementation details of the gradient method and highlight three observations.

- 1) A fundamental prerequisite of implementing the updating rule in (1) is the computation of the gradient $\nabla f(\mathbf{x})$ at a given point \mathbf{x} . In the conventional analysis of gradient method (see [6, p. 10]), we usually assume the availability of a first-order oracle that, given a point \mathbf{x} , returns the function value $f(\mathbf{x})$ and the gradient $\nabla f(\mathbf{x})$ in reasonable time. In many applications, the objective function can be expressed as a sum, product, or composition of some elemental functions, whose function value and gradient can be evaluated efficiently. For these problems, the first-order oracle has very efficient implementation, sometimes even in an automatic way by using Algorithmic Differentiation (AD) software such as PyTorch, TensorFlow, and JAX. The rationale behind these software packages lies in the extensive calculus rules for gradients. According to the ‘‘cheap gradient principle’’ of AD [7, Section 3.3], the time needed to evaluate the gradient roughly equals that needed to evaluate the function value.
- 2) Equipped with a gradient oracle, we can generate a sequence $\{\mathbf{x}_t\}_t$ with the gradient method updating rule (1). The result in (2) is key to analyzing the performance of gradient methods and furnishes an estimate of the number of iterations needed for computing an approximate ε -stationary point. In particular, for a given accuracy parameter $\varepsilon > 0$, if $T + 1 \geq \left\lceil \frac{2\beta(f(\mathbf{x}_0) - \inf f)}{\varepsilon^2} \right\rceil$, then there exists an index $t \in \{0, \dots, T\}$ such that the point \mathbf{x}_t is ε -stationary. Notably, such an upper bound on the number of iterations is independent of the dimension d of the function f , making it applicable to very large-scale settings such as the training of overparameterized deep neural networks.
- 3) The third observation concerns the termination of the gradient method. If the collection of iterates $\{\mathbf{x}_t\}_{t=0}^T$ contains an ε -stationary point, then such a point can be identified simply by computing and comparing the gradient norms of the iterates. In practice, we can of course keep track of the

minimal gradient norm seen so far and terminate the algorithm immediately if an ε -stationary point is identified. Consequently, using the gradient method, we can compute an ε -stationary point of a lower-bounded, β -smooth function by calling a first-order oracle and terminating in $O(\beta/\varepsilon^2)$ steps.

These three observations, while being of fundamental importance when minimizing f , can be considered trivial or obvious. However, as we will see shortly, similar tasks can be highly nontrivial or even intractable for functions lacking favorable properties. For the purpose of comparison, the above discussion can be summarized as follows.

Gradient method for nonconvex, smooth functions
<ol style="list-style-type: none"> 1) The gradient $\nabla f(\mathbf{x})$ at any point \mathbf{x} can be computed using extensive gradient calculus rules. 2) The number of steps T required to compute an ε-stationary point is dimension-free and can be determined <i>a priori</i>. 3) An ε-stationary point can be easily identified from the generated iterates.

Subgradient Method for Convex Functions

In parallel with smoothness, another favorable property of functions is convexity. A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is convex if the inequality $tf(\mathbf{x}) + (1-t)f(\mathbf{y}) \geq f(t\mathbf{x} + (1-t)\mathbf{y})$ holds for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ and $t \in (0, 1)$. Similar to the gradient method, a textbook algorithm for optimizing a convex function is the subgradient method. Recall that the set of all subgradients of a convex function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ evaluated at the point \mathbf{x} is called the subdifferential $\partial f(\mathbf{x}) := \{\mathbf{g} \in \mathbb{R}^d : f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^\top(\mathbf{y} - \mathbf{x}), \forall \mathbf{y} \in \mathbb{R}^d\}$; see [3, Section 23]. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a convex, L -Lipschitz, lower-bounded function, and assume that the point $\mathbf{x}^* \in \mathbb{R}^d$ is one of the optimal solutions of the problem of minimizing f ; i.e., $f(\mathbf{x}^*) = \inf f$. Fix an accuracy parameter $\varepsilon > 0$. Using a fixed constant step size ε/L , the subgradient method generates a sequence $\{\mathbf{x}_t\}_t$ according to the following rule:

$$\mathbf{x}_{t+1} := \mathbf{x}_t - (\varepsilon/L) \cdot \mathbf{g}_t / \|\mathbf{g}_t\|_2, \quad \mathbf{g}_t \in \partial f(\mathbf{x}_t), \quad \forall t \geq 0. \quad (3)$$

For any $T \in \mathbb{N}$, one can show that (see [6, Theorem 3.2.2])

$$\left(\min_{0 \leq t \leq T} f(\mathbf{x}_t) \right) - \inf f \leq \frac{L^2 \|\mathbf{x}_0 - \mathbf{x}^*\|_2^2}{2\varepsilon T} + \frac{\varepsilon}{2}. \quad (4)$$

That is to say, to compute an ε -optimal solution (i.e., a point \mathbf{x} satisfying $f(\mathbf{x}) - \inf f \leq \varepsilon$), it suffices to set $T \geq \left\lceil \frac{L^2 \|\mathbf{x}_0 - \mathbf{x}^*\|_2^2}{\varepsilon^2} \right\rceil$. For comparison, we also highlight three observations from convex, nonsmooth optimization as counterparts to the nonconvex, smooth case.

- 1) Though not as strong as those for smooth functions, under very mild assumptions (see [3, Section 23]), rich calculus rules are available for convex functions. Hence, it is fair to say that for many

real-world convex functions, at least one subgradient can be easily computed at any queried point; see also [6, Section 3.1.6]. In fact, the updating rule in (3) of the subgradient method only requires an *arbitrary* subgradient at the concerned point.

- 2) The result in (4), proved using only the Lipschitz continuity of f and the definition of convex subgradients, provides a satisfactory performance estimate for the fixed-step-size subgradient method applied to convex functions. To obtain an ε -optimal solution, we only need at most $O(1/\varepsilon^2)$ steps, which, like the upper bound for the gradient method, is independent of the dimension d and can be determined in an *a priori* manner.
- 3) The third observation is quite straightforward. If the sequence $\{\mathbf{x}_t\}_{t=0}^T$ contains an ε -optimal solution, then one of these can be identified by computing and comparing objective values along the way.

For the purpose of comparison, we again summarize the above discussion as follows.

Subgradient method for convex, nonsmooth functions
<ol style="list-style-type: none"> 1) An element of $\partial f(\mathbf{x})$ at any point \mathbf{x} can be computed using rich convex subdifferential calculus rules. 2) The number of steps T required to compute an ε-optimal point is dimension-free and can be determined <i>a priori</i>. 3) An ε-optimal point can be easily identified from the generated iterates.

Nonconvex, Nonsmooth Functions: Generalized Differentiation

Now, we proceed to the main focus of this paper—the simultaneously nonconvex and nonsmooth case. We have seen that, in both the smooth and convex settings, local approximation of the function under consideration plays a key role. In the smooth case, the gradient serves as the crucial component for local linear approximation in the spirit of Taylor’s theorem, whereas in the convex case, the subdifferential provides a local conic approximation by definition. A particularly interesting and comprehensive class of nonconvex, nonsmooth functions is the class of locally Lipschitz functions. A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is locally Lipschitz continuous if for any $\mathbf{x} \in \mathbb{R}^d$, there exist a scalar $L_{\mathbf{x}}$ and a neighborhood $\mathcal{N}_{\mathbf{x}}$ of \mathbf{x} such that f is $L_{\mathbf{x}}$ -Lipschitz continuous on $\mathcal{N}_{\mathbf{x}}$. In the nonsmooth and variational analysis literature, many different notions of the subdifferential have been proposed, yielding different local approximations. Among them, the Clarke subdifferential is particularly convenient for algorithm design and analysis (at the cost of tightness; see discussions in [8, Section V]). The Clarke subdifferential of a locally Lipschitz function is defined as

$$\partial f(\mathbf{x}) := \text{conv} \{ \mathbf{s} : \exists \mathbf{x}_n \rightarrow \mathbf{x}, \nabla f(\mathbf{x}_n) \text{ exists, } \nabla f(\mathbf{x}_n) \rightarrow \mathbf{s} \},$$

where, for a set S , $\text{conv} S$ denotes its convex hull. We record some useful properties of the Clarke subdifferential $\partial f(\mathbf{x})$ of f at the point \mathbf{x} as follows.

Theorem 1 (cf. [2, Propositions 2.1.2, 2.2.4, 2.2.7, 2.3.2, 2.1.5, and p. 39], [9, Proposition 2.5]). *Suppose that the functions f, g are locally Lipschitz continuous. The following hold.*

- (a) $\partial f(\mathbf{x})$ is convex, compact, and non-empty.
- (b) If f is continuously differentiable at \mathbf{x} , then $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$.
- (c) If f is convex, then $\partial f(\mathbf{x})$ equals the convex subdifferential of f at \mathbf{x} .
- (d) If \mathbf{x} is a local minimizer of f , then $\mathbf{0} \in \partial f(\mathbf{x})$.
- (e) If $\mathbf{x}_n \rightarrow \mathbf{x}, \mathbf{g}_n \rightarrow \mathbf{g}$ with $\mathbf{g}_n \in \partial f(\mathbf{x}_n)$, then $\mathbf{g} \in \partial f(\mathbf{x})$.
- (f) $\partial(\alpha \cdot f) = \alpha \cdot \partial f$ for any $\alpha \in \mathbb{R}$. In particular, $\partial(-f) = -\partial f$.
- (g) $\partial(f + g)(\mathbf{x}) \subseteq \partial f(\mathbf{x}) + \partial g(\mathbf{x})$.
- (h) If f is continuously differentiable at \mathbf{x} , then $\partial(f + g)(\mathbf{x}) = \nabla f(\mathbf{x}) + \partial g(\mathbf{x})$.
- (i) $\partial[(\mathbf{x}, \mathbf{y}) \mapsto f(\mathbf{x}) + g(\mathbf{y})](\mathbf{x}, \mathbf{y}) = \partial f(\mathbf{x}) \times \partial g(\mathbf{y})$.

Here are some quick remarks on the properties listed in Theorem 1 and we refer the reader to more detailed discussions on motivations and extensions in [8]. Item (a) promises that a Clarke subgradient at \mathbf{x} (i.e., an element of the subdifferential $\partial f(\mathbf{x})$) is always well defined (recall that the usual convex subdifferential may not be well defined for a nonconvex function, such as $x \mapsto -x^2$). Items (b) and (c) show that the Clarke subdifferential coincides with well-defined notions (gradient and convex subdifferential) when the function has favorable properties. Besides, continuous differentiability in Item (b) cannot be replaced by differentiability; see [4, p. 304]. Item (d) is a generalization of Fermat's rule and shows that $\mathbf{0} \in \partial f(\mathbf{x})$ is a legitimate optimality condition for a general locally Lipschitz function f . Item (e) shows that the mapping ∂f is so-called outer semicontinuous (see [4, Definition 5.4]), and this property is crucial for the consistency of approximate stationarity. Items (f) to (i) present useful calculus rules; we refer the reader to [2, Chapter 2] and [4, Chapter 10] for more systematic treatments.

Nonconvex, Nonsmooth Functions: Three Tasks and Examples

Having defined the Clarke subdifferential, we now consider a generalization of the three basic problems previously discussed for smooth and convex functions.

- 1) The first question is how to obtain a subgradient of a locally Lipschitz function f at a specified point \mathbf{x} . This subgradient can be used to implement a subgradient-type method for optimizing nonsmooth functions beyond convexity. Recall that the computation of gradients for smooth functions and subgradients for convex functions relies heavily on calculus rules. However, for general locally

Lipschitz functions, the same calculus rules may fail or only hold in a weak sense—in the form of set inclusions rather than equalities (see, e.g., Theorem 1(g)). Therefore, the seemingly simple task of computing gradient/subgradient for smooth/convex functions can be highly nontrivial for general locally Lipschitz functions. Possible solutions may require new theoretical and algorithmic ideas.

- 2) The second question is how to compute an approximate stationary point. We must point this out right at the beginning: It is even nontrivial to define an appropriate concept of approximate stationarity. A proper concept should be computable in reasonable time and have a meaningful connection to the “exact” stationary points, i.e., the set $\{\mathbf{x} : \mathbf{0} \in \partial f(\mathbf{x})\}$. Recall that in the smooth and/or convex case, the efforts required to find an approximate solution can be upper-bounded in a dimension-free and *a priori* manner. Can we achieve a similar result for more general nonconvex, nonsmooth functions? It turns out that the computability may vary across different classes of functions, and this is still an active area of research, both in terms of theoretical analysis and algorithm design.
- 3) The third question is how we can verify the quality of a solution. The motivation for studying solution testing may not seem that strong in the smooth and/or convex cases. However, when optimizing a nonconvex, nonsmooth function, many algorithms are only known to enjoy asymptotic convergence guarantees to stationary points; that is, they generate sequences that converge to stationary points, but without a known convergence rate or an implementable stopping rule. Hence, to obtain a finite-time algorithm, it is crucial to know when we encounter an approximate stationary point and how this point can be identified. As with the first question above, one major hurdle here is the failure of exact subdifferential calculus rules. For example, as illustrated in Theorem 1(g), having $\mathbf{0} \in \partial f(\mathbf{x}) + \partial g(\mathbf{x})$ does not guarantee that $\mathbf{0} \in \partial(f + g)(\mathbf{x})$. More challengingly, when evaluating a subgradient, the algorithm takes an arbitrary element from the subdifferential. However, to give a correct answer of “no” for testing whether $\mathbf{0} \in \partial f(\mathbf{x})$, one needs to certify that no element in $\partial f(\mathbf{x})$ is the vector $\mathbf{0}$, which could cause additional difficulty. Indeed, current research on verifying the quality of solutions is still in its early stages, and there are far more open problems than those that have been resolved.

It is evident that the three tasks mentioned above are of fundamental importance in solving nonsmooth optimization problems for modern applications. Almost all optimization methods, especially subgradient-type methods, involve evaluating subgradients, finding stationary points, and verifying solution quality. These tasks can be grouped under the umbrella term “computational nonsmooth analysis” and can be summarized as follows.

Three computational problems for nonconvex, nonsmooth functions

- 1) Given a function f and a point \mathbf{x} , find a vector \mathbf{g} in $\partial f(\mathbf{x})$.
- 2) Given a function f , find a point \mathbf{x} in $\{\mathbf{x} : \mathbf{0} \in \partial f(\mathbf{x})\}$.
- 3) Given a function f and a point \mathbf{x} , examine the membership of \mathbf{x} in $\{\mathbf{x} : \mathbf{0} \in \partial f(\mathbf{x})\}$.

Below, we provide two examples of nonconvex, nonsmooth problems encountered in training modern neural networks. For the first example, we consider the setting of training a smooth deep neural network using the hinge loss.

Example 1 (smooth network with hinge loss). *Let $(\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \{-1, 1\}$ for $n \in \{1, \dots, N\}$ be the given data. Consider a neural network $f : \mathbb{R}^s \times \mathbb{R}^d \rightarrow \mathbb{R}$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^s$. For input $\mathbf{x} \in \mathbb{R}^d$, the network f with parameter $\boldsymbol{\theta}$ outputs its label estimate $f(\boldsymbol{\theta}, \mathbf{x}) \in \mathbb{R}$. We assume that f is smooth with Lipschitz continuous gradient mapping ∇f . Using the hinge loss (i.e., $(y, \hat{y}) \mapsto \max\{0, 1 - y \cdot \hat{y}\}$), we can write the loss for training the network f on the n th data point as follows:*

$$\ell_n(\boldsymbol{\theta}) := \max\{0, 1 - y_n \cdot f(\boldsymbol{\theta}, \mathbf{x}_n)\}.$$

The training loss for the entire dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ can be written as $\ell(\boldsymbol{\theta}) := \sum_{n=1}^N \ell_n(\boldsymbol{\theta})$, which can be simultaneously nonconvex and nonsmooth for very simple settings. For example, consider $N = s = d = 1 = x_1 = y_1 = 1$, $f(\theta, x) := (\theta \cdot x)^2$, for which $\ell(\theta) = \max\{0, 1 - \theta^2\}$.

Next, we examine the problem of training a two-layer ReLU network with quadratic loss.

Example 2 (two-layer ReLU network). *Let $(\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$ for $n \in \{1, \dots, N\}$ be the given data. We consider a two-layer neural network with ReLU activation function $t \mapsto \max\{t, 0\}$. Let $\bar{\mathbf{x}}_n := (\mathbf{x}_n, 1)$. Using a quadratic loss, we can write the loss for training the network on the n th data point as follows:*

$$\ell_n(\mathbf{W}, \mathbf{u}) := \left(y_n - \sum_{h=1}^H u_h \cdot \max\{0, \mathbf{w}_h^\top \bar{\mathbf{x}}_n\} \right)^2,$$

where $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_H] \in \mathbb{R}^{(d+1) \times H}$, $\mathbf{u} \in \mathbb{R}^H$. The training loss can be written as $\ell(\mathbf{W}, \mathbf{u}) := \sum_{n=1}^N \ell_n(\mathbf{W}, \mathbf{u})$, which can be simultaneously nonconvex and nonsmooth in general.

As a side remark, we note that typical machine learning models involve optimizing the expectation of a random objective $\boldsymbol{\theta} \mapsto \mathbb{E}_{\boldsymbol{\xi}}[\ell(\boldsymbol{\xi}, \boldsymbol{\theta})]$ (the population loss), where $\boldsymbol{\xi}$ is a random variable usually composed of a random data point and its label. Unfortunately, since a rigorous treatment of such random functions requires the exposition of more sophisticated concepts and given the page limitations, we are only able to cover some superficial aspects of stochasticity in this introductory article. We refer the interested reader to [4, Chapter 14], [10], [11, Chapter 10], and the references therein for a more detailed treatment.

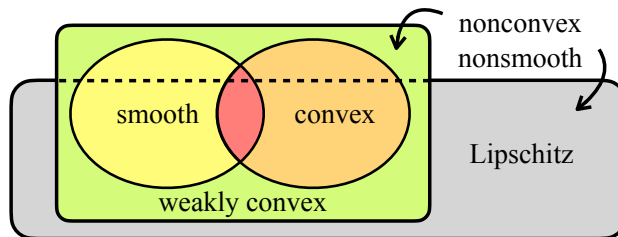


Fig. 1: Illustration of different function classes.

EVALUATION OF SUBDIFFERENTIALS AND SUBGRADIENTS

In practical training of deep neural networks, subgradient-type methods have become workhorses. At the heart of subgradient-type methods lies the computation of subgradients, a task that, unlike its smooth counterpart, is intricate and still a subject of intense investigation. In modern computational environments, subgradients are usually provided by algorithmic differentiation software [7]. However, when applied to functions with nonsmooth components (e.g., ReLU, max-pooling), the correctness of the output from these software packages is often questionable; see, e.g., [12]–[14]. In this section, we survey three different approaches that resolve or bypass these difficulties.

Evaluation of Subdifferentials

The validity of calculus rules is at the very heart of evaluating gradients and convex subdifferentials. However, for nonconvex, nonsmooth functions, these rules do not hold without conditions. Among various regularity conditions, weak convexity is particularly relevant to our discussion.

A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is called ρ -weakly convex with $\rho \in \mathbb{R}$ if $f + \frac{\rho}{2} \|\cdot\|_2^2$ is convex. When $\rho \leq 0$, ρ -weak convexity is equivalent to the conventional $(-\rho)$ -strong convexity. When $\rho > 0$, a ρ -weakly convex function f can be simultaneously nonconvex and nonsmooth. By Theorem 1(h) and (c), the (Clarke) subdifferential of a ρ -weakly convex function f at \mathbf{x} can be written as $\partial[f + \frac{\rho}{2} \|\cdot\|_2^2](\mathbf{x}) - \{\rho\mathbf{x}\}$. We also note that both smoothness and convexity are special subclasses of weak convexity; see Figure 1 for an illustration. For a more detailed discussion of these regularity conditions and their associated calculus rules, we refer the reader to [8] and [11].

Our first example of evaluating subdifferentials concerns the hinge loss in Example 1.

Example 3 (subdifferential for a smooth network with hinge loss). *In the setting of Example 1, we observe that the loss function ℓ_n of the n th data point is represented in a convex-composite form; namely, it is*

the composition of a convex function and a smooth function. Hence, we know that the function ℓ_n is actually weakly convex [15, Lemma 4.2], and its subdifferential can be written as

$$\partial\ell_n(\boldsymbol{\theta}) = \begin{cases} \{\mathbf{0}\} & \text{for } 1 < y_n \cdot f(\boldsymbol{\theta}, \mathbf{x}_n), \\ \{-y_n t \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \mathbf{x}_n) : t \in [0, 1]\} & \text{for } 1 = y_n \cdot f(\boldsymbol{\theta}, \mathbf{x}_n), \\ \{-y_n \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \mathbf{x}_n)\} & \text{for } 1 > y_n \cdot f(\boldsymbol{\theta}, \mathbf{x}_n). \end{cases}$$

Since sums of weakly convex functions are weakly convex and weakly convex functions satisfy strong sum rules (see [15, Theorem 3.1]), it follows that

$$\partial\ell(\boldsymbol{\theta}) = \sum_{n=1}^N \partial\ell_n(\boldsymbol{\theta}).$$

It may seem straightforward to derive the details in Example 3, since what we just did is not that different from the convex case. However, we need to point out that the validity of the calculus rules used in Example 3 is a blessing of weak convexity, a favorable property beyond convexity and smoothness, which is also very fragile. In the next example, we show that a simple “swap” of the order of composition can change the game drastically.

Example 4 (subdifferential for a two-layer ReLU network). *In the setting of Example 2, for simplicity, we fix $\mathbf{u} = \mathbf{1}$ and only consider the function $\mathbf{W} \mapsto \ell(\mathbf{W}, \mathbf{1})$. The argument can be extended to cover the general case of arbitrary \mathbf{u} at the cost of cumbersome notation; see [16, Section 6] for full details. With the help of a partial linearization result [16, Proposition 6.1], the subdifferential of the loss function $\ell(\cdot, \mathbf{1}) = \sum_{n=1}^N \ell_n(\cdot, \mathbf{1})$ can be written as*

$$\partial \left[\sum_{n=1}^N \ell_n(\cdot, \mathbf{1}) \right] (\mathbf{W}) = \partial \left[\sum_{n=1}^N \overline{\ell_{n, \mathbf{W}}}(\cdot, \mathbf{1}) \right] (\mathbf{W}),$$

where the function $\overline{\ell_{n, \mathbf{W}}}(\cdot, \mathbf{1})$ is a partial linearization of the function $\ell_n(\cdot, \mathbf{1})$ at \mathbf{W} and is defined as

$$\overline{\ell_{n, \mathbf{W}}}(\mathbf{W}', \mathbf{1}) := \underbrace{2 \left(y_n - \sum_{h=1}^H \max \{0, \mathbf{w}_h^\top \bar{\mathbf{x}}_n\} \right)}_{=: p_n, \text{ which is a constant w.r.t. } \mathbf{W}'}. \cdot \left(y_n - \sum_{h=1}^H \max \{0, (\mathbf{w}'_h)^\top \bar{\mathbf{x}}_n\} \right).$$

It is easy to see that the function $\overline{\ell_{n, \mathbf{W}}}(\cdot, \mathbf{1})$ is a piecewise affine function, and hence, the function $\sum_{n=1}^N \overline{\ell_{n, \mathbf{W}}}(\cdot, \mathbf{1})$ is also piecewise affine. Therefore, the crux of describing $\partial[\ell(\cdot, \mathbf{1})](\mathbf{W})$ lies in characterizing the subdifferential of a piecewise affine function $\sum_{n=1}^N \overline{\ell_{n, \mathbf{W}}}(\cdot, \mathbf{1})$. Meanwhile, we can write the function $\sum_{n=1}^N \overline{\ell_{n, \mathbf{W}}}(\cdot, \mathbf{1})$ as the difference of two convex piecewise affine functions, say, $h - g$, where $h := \sum_{p_n \leq 0} \overline{\ell_{n, \mathbf{W}}}(\cdot, \mathbf{1})$ and $g := -\sum_{p_n > 0} \overline{\ell_{n, \mathbf{W}}}(\cdot, \mathbf{1})$. Using recent results in [16, Theorem 4.11, Proposition 4.19], we conclude that the equality

$$\partial[\ell(\cdot, \mathbf{1})](\mathbf{W}) = \sum_{n=1}^N \partial[\ell_n(\cdot, \mathbf{1})](\mathbf{W})$$

holds if and only if $\text{par}(\partial h(\mathbf{W})) \cap \text{par}(\partial g(\mathbf{W})) = \{\mathbf{0}\}$, where $\text{par}(S)$ is the subspace parallel to the affine hull of S , and h, g are both convex functions.

In contrast to the derivation in Example 3, the situation in Example 4 is considerably more involved, and the resulting subdifferential representation is not as strong as in the weakly convex case. Therefore, the main message of this subsection is that, while subdifferential calculus rules play a vital role in characterizing convex subdifferentials, they become very delicate and complicated when applied to modern nonsmooth neural networks. Before ending this subsection, let us mention a recently proposed framework [14] that in some sense mitigates the failure of calculus rules by defining a new (and relaxed) notion called conservative set-valued field. Due to page limitations, we refer the reader to [14] for more details on developments in this direction.

Lexicographic Differentiation

Instead of aiming to evaluate the entire subdifferential set at a point, we review a technique termed lexicographic differentiation, which is used to evaluate a subgradient or a subset of the subdifferential. This line of work was initiated by Nesterov in [17]. Informally, the construction of the so-called lexicographic subgradient involves a systematic (or “lexicographic”) exploration of the local behavior of a directional derivative function to select an “active piece” in a reproducible manner. The exploration is carried out by sequentially taking higher-order-like directional derivatives (referred to as *homogenization* in [17]). After several such operations, this process yields a linear function, whose gradient is taken as the lexicographic subgradient of the original function.

Formally, for a locally Lipschitz continuous, possibly vector-valued, and directionally differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$, we say that f is lexicographically smooth [17, Definition 2] if, for any $\mathbf{U} \in \mathbb{R}^{d \times k}$ with $\text{rank}(\mathbf{U}) = d$, the following higher-order-like directional derivative functions are well defined:

$$f_{\mathbf{x}, \mathbf{U}}^{(0)} := \mathbf{d} \mapsto f'(\mathbf{x}; \mathbf{d}), \quad f_{\mathbf{x}, \mathbf{U}}^{(1)} := \mathbf{d} \mapsto [f_{\mathbf{x}, \mathbf{U}}^{(0)}]'(\mathbf{u}_1; \mathbf{d}), \quad \dots, \quad f_{\mathbf{x}, \mathbf{U}}^{(k)} := \mathbf{d} \mapsto [f_{\mathbf{x}, \mathbf{U}}^{(k-1)}]'(\mathbf{u}_k; \mathbf{d}),$$

where $f_{\mathbf{x}, \mathbf{U}}^{(j)} : \mathbb{R}^d \rightarrow \mathbb{R}^m$ is recursively defined as the directional derivative (in a coordinate-wise sense) $[f_{\mathbf{x}, \mathbf{U}}^{(j-1)}]'(\mathbf{u}_j; \cdot)$ of the function $f_{\mathbf{x}, \mathbf{U}}^{(j-1)}$ at the point \mathbf{u}_j . The subscripts \mathbf{x}, \mathbf{U} emphasize the dependence on the base point \mathbf{x} , which defines $f_{\mathbf{x}, \mathbf{U}}^{(0)}$, and on the directions $\{\mathbf{u}_j\}_{j=1}^k$. This construction makes use of the fact that the directional derivative is positively homogeneous in its direction. The functions $\{f_{\mathbf{x}, \mathbf{U}}^{(j)}\}_{j=0}^k$ probe the behavior of $f'(\mathbf{x}; \cdot)$ along the directions $\{\mathbf{u}_j\}_{j=1}^k$ sequentially, which explains part of the rationale behind the term lexicographic differentiation.

The class of lexicographically smooth functions is quite general and contains many interesting subclasses, for example, continuously differentiable functions, convex functions, and piecewise smooth

functions (see [18, Proposition 3.1]). Most importantly, the composition of lexicographically smooth functions is still lexicographically smooth (see [17, Theorem 1]).

According to [17, Theorem 2], for a lexicographically smooth f with its associated full-row-rank matrix \mathbf{U} , the function $f_{\mathbf{x},\mathbf{U}}^{(k)}$ is linear. We call its gradient $\nabla f_{\mathbf{x},\mathbf{U}}^{(k)}(\mathbf{0}) \in \mathbb{R}^{d \times m}$ the lexicographic subgradient of f ; see [17, Definition 5]. By considering all full-row-rank matrices \mathbf{U} , we can also define the lexicographic subdifferential of f at the point \mathbf{x} as follows [17, Definition 6]:

$$\partial^{\text{lex}} f(\mathbf{x}) := \left\{ \nabla f_{\mathbf{x},\mathbf{U}}^{(k)}(\mathbf{0}) : \mathbf{U} \in \mathbb{R}^{d \times k}, \text{rank}(\mathbf{U}) = d \right\} \subseteq \mathbb{R}^{d \times m}.$$

All of the above concepts naturally apply to the more familiar real-valued case (i.e., $m = 1$). Analogous to the properties in Theorem 1, the following result holds for the lexicographic subdifferential.

Theorem 2 (cf. [17, Theorems 10, 11]). *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a given lexicographically smooth function.*

- *If \mathbf{x} is a local minimizer of f , then $\mathbf{0} \in \text{cl conv } \partial^{\text{lex}} f(\mathbf{x}) \subseteq \mathbb{R}^d$.¹*
- *$\partial^{\text{lex}} f(\mathbf{x}) \subseteq \partial f(\mathbf{x}) \subseteq \mathbb{R}^d$.*
- *If f is continuously differentiable, then $\partial^{\text{lex}} f(\mathbf{x}) = \partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$.*

Hence, to find a Clarke subgradient of f , it suffices to compute a lexicographic subgradient. A remarkable property of the lexicographic subgradient is that it enjoys a strong chain rule, which facilitates an efficient computation pipeline. This strong chain rule can be naturally stated using the notion of *lexicographic directional derivative* of $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ defined as follows (see [18, Definition 2.1]):

$$f'_{\text{lex}}(\mathbf{x}; \mathbf{U}) := \left[f_{\mathbf{x},\mathbf{U}}^{(0)}(\mathbf{u}_1) \mid \cdots \mid f_{\mathbf{x},\mathbf{U}}^{(k-1)}(\mathbf{u}_k) \right] \in \mathbb{R}^{m \times k}.$$

For a square full-rank matrix $\mathbf{U} \in \mathbb{R}^{d \times d}$ (hence $k = d$) and functions $g : \mathbb{R}^m \rightarrow \mathbb{R}$, $h : \mathbb{R}^d \rightarrow \mathbb{R}^m$, $f := g \circ h$, the lexicographic directional derivative admits a chain rule similar to that of the vanilla directional derivative, namely, $f'_{\text{lex}}(\mathbf{x}; \mathbf{U}) = [g \circ h]'_{\text{lex}}(\mathbf{x}; \mathbf{U}) = g'_{\text{lex}}(h(\mathbf{x}); h'_{\text{lex}}(\mathbf{x}; \mathbf{U})) \in \mathbb{R}^{1 \times d}$; see [18, Proposition 2.2] and [17, Theorem 5]. It is shown in [18, p. 6] that $f'_{\text{lex}}(\mathbf{x}; \mathbf{U}) = \nabla f_{\mathbf{x},\mathbf{U}}^{(d)}(\mathbf{0})^\top \mathbf{U} \in \mathbb{R}^{1 \times d}$. Therefore, the lexicographic subgradient $\nabla f_{\mathbf{x},\mathbf{U}}^{(d)}(\mathbf{0}) \in \partial^{\text{lex}} f(\mathbf{x})$ can be computed efficiently by solving a system of linear equations. We note a recent use of lexicographic differentiation in ReLU neural ODEs [19].

Piecewise Differentiation

We have seen in the last subsection that lexicographic differentiation is a powerful tool for evaluating subgradients. In this subsection, we consider another local approximation technique, initiated by Griewank [20], to directly construct a piecewise affine approximation of certain subclass of piecewise differentiable

¹For a set S , $\text{cl } S$ denotes the closure of the set S .

functions. This subclass includes functions defined by evaluation programs involving smooth elementals, absolute values, and the max and min operators. By the identities $\max\{a, b\} = (a + b)/2 + |a - b|/2$ and $\min\{a, b\} = (a + b)/2 - |a - b|/2$, this class of functions is closed under composition.

Let $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}$ be a function in such subclass. We can define a piecewise affine approximation $\varphi(\mathbf{x}) + \Delta\varphi(\mathbf{x}; \mathbf{d})$ of φ at the point \mathbf{x} in the direction \mathbf{d} simply by replacing all smooth elements by their first-order approximation and keeping the absolute value function unchanged. The idea is quite simple and intuitive, as illustrated by the following example.

Example 5. Consider the first Chebyshev–Rosenbrock function of Nesterov $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$:

$$\varphi(\mathbf{x}) := \frac{1}{4}(x_1 - 1)^2 + \sum_{i=1}^{n-1} |1 - 2x_i^2 + x_{i+1}|.$$

For each $\mathbf{x} \in \mathbb{R}^n$, the piecewise linearized model $\mathbf{d} \mapsto \varphi(\mathbf{x}) + \Delta\varphi(\mathbf{x}; \mathbf{d})$ of φ at \mathbf{x} is

$$\varphi(\mathbf{x}) + \Delta\varphi(\mathbf{x}; \mathbf{d}) = \frac{1}{4}(x_1 - 1)^2 + \frac{d_1}{2}(x_1 - 1) + \sum_{i=1}^{n-1} |1 - 2x_i^2 + x_{i+1} - 4x_i d_i + d_{i+1}|.$$

It is notable that the vanilla directional derivative $\varphi'(\mathbf{x}; \cdot)$ can also be seen as a piecewise approximation of φ at the point \mathbf{x} . Such an approximation is positively homogeneous, which allows it to be used to define a corresponding subdifferential (known as the Fréchet subdifferential; see [8]). However, the vanilla directional derivative only considers the pieces that are currently being activated and cannot take nearby inactivated pieces into consideration. Hence, while losing the positive homogeneity, $\Delta\varphi(\mathbf{x}; \cdot)$ can be a better local approximation of a piecewise smooth function in some sense. We conclude this section by noting that the aforementioned piecewise differentiation technique has been applied in the training of nonsmooth neural networks; see [21] and the survey [22] for more applications.

FINDING (APPROXIMATE) STATIONARY POINTS

Although nonconvex optimization models are pervasive in applications, computing a global solution to such models is generally intractable. It is therefore natural to focus on more relaxed solution concepts. A stationary point is one that satisfies the first-order optimality condition defined by the subdifferential; thus, obtaining a stationary point provides an optimality guarantee for the model under consideration. In this section, we discuss algorithms for finding approximate stationary points and the inherent difficulties involved in doing so for certain representative classes of nonconvex and nonsmooth functions.

Oracles and Approximate Stationarity Concepts

A conventional approach to evaluating algorithm efficiency and problem complexity is using the information-theoretic framework in [23]. Simply put, we measure the effort of an algorithm by counting

the number of calls it makes to certain oracles before termination. Usually, when queried at a point \mathbf{x} for a function f , an oracle provides certain local information about f around \mathbf{x} . In this section, we consider the following oracles with different strengths:

- 1) **Subgradient oracle:** For given $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^d$, the oracle $\mathcal{O}_f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ satisfies $\mathcal{O}_f(\mathbf{x}) \in \partial f(\mathbf{x})$. In other words, when queried at \mathbf{x} , a subgradient oracle returns an *arbitrary* subgradient from the subdifferential set $\partial f(\mathbf{x})$. Among many possibilities, the lexicographic differentiation technique discussed in the previous section can be used to implement a subgradient oracle.
- 2) **Subdifferential oracle:** For given $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^d$, the oracle $\mathcal{O}_f : \mathbb{R}^d \rightrightarrows \mathbb{R}^d$ satisfies $\mathcal{O}_f(\mathbf{x}) = \partial f(\mathbf{x})$, where \rightrightarrows indicates that \mathcal{O}_f is a set-valued mapping. Compared with the notion of subgradient oracle, a subdifferential oracle provides more information about the local geometry and also allows the algorithm to select a *particular* subgradient from $\partial f(\mathbf{x})$. A subdifferential oracle can be implemented using subdifferential calculus rules, as long as they hold in a strong sense.
- 3) **Local oracle:** For given $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^d$, a local oracle $\mathcal{O}_f : \mathbb{R}^d \rightarrow (\mathbb{R}^d \rightarrow \mathbb{R})$ returns a function $f_x : \mathbb{R}^d \rightarrow \mathbb{R}$ such that $f(\mathbf{y}) = f_x(\mathbf{y})$ for any $\mathbf{y} \in \mathbb{B}_r(\mathbf{x}) := \{\mathbf{z} : \|\mathbf{z} - \mathbf{x}\|_2 \leq r\}$ and some (unknown) $r > 0$. The concept of local oracle, being mainly a theoretical tool, is very difficult to implement in practice but is capable of providing remarkably extensive local information about the function. For example, for a smooth function f , an algorithm interacting with f via a local oracle \mathcal{O}_f can obtain information of $f(\mathbf{x})$, $\nabla f(\mathbf{x})$, $\nabla^2 f(\mathbf{x})$ with a single call of \mathcal{O}_f at the point \mathbf{x} .

In some modern large-scale applications, instead of computing an exact subgradient, we may only be able to obtain a stochastic estimate of it. This idea is formalized through the following notion of a stochastic subgradient oracle:

- 1') **Stochastic subgradient oracle:** For given $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^d$, the oracle $\mathcal{O}_f : \Xi \times \mathbb{R}^d \rightarrow \mathbb{R}^d$ satisfies $\mathbb{E}_\xi[\mathcal{O}_f(\xi, \mathbf{x})] \in \partial f(\mathbf{x})$ and $\mathbb{E}_\xi[\|\mathcal{O}_f(\xi, \mathbf{x})\|_2] \leq \sigma$ for some $\sigma < \infty$.

For a smooth and Lipschitz continuous function $f : \Xi \times \mathbb{R}^d \rightarrow \mathbb{R}$, a typical mini-batch implementation of the stochastic (sub)gradient oracle for the objective function $\mathbf{x} \mapsto \mathbb{E}_\xi[f(\xi, \mathbf{x})]$ involves evaluating and averaging $\nabla_{\mathbf{x}} f(\xi, \mathbf{x})$ over independent and identically distributed (iid) samples of ξ , since we have $\nabla_{\mathbf{x}} \mathbb{E}_\xi[f(\xi, \mathbf{x})] = \mathbb{E}_\xi[\nabla_{\mathbf{x}} f(\xi, \mathbf{x})]$ under mild assumptions; see [10, Theorem 9.56]. However, when the function is nonsmooth, according to [2, Theorem 2.7.2], we may only have the set inclusion $\partial \mathbb{E}_\xi[f(\xi, \cdot)](\mathbf{x}) \subseteq \mathbb{E}_\xi[\partial f(\xi, \cdot)](\mathbf{x})$, where the expectation on the right-hand side is defined as the set of all integrals of measurable selections (see [2, p. 76] for the definition). Hence, for a random Lipschitz function, mini-batch subgradient estimates may not be unbiased in some sense.

Next, we turn to discuss the notion of approximate stationarity that we aim to compute. Since nonconvex

functions are the main focus of this section, computing an ε -optimal solution (i.e., a point \mathbf{x} satisfying $f(\mathbf{x}) \leq \inf f + \varepsilon$) is generally intractable. Another relevant solution concept is stationary point. To define “ ε -stationarity” in the nonsmooth case, a natural approach is to mimic the definition $\|\nabla f(\mathbf{x})\|_2 \leq \varepsilon$ in the smooth case and proceed as follows: We call \mathbf{x} an ε -stationary point of f if $\text{dist}(\mathbf{0}, \partial f(\mathbf{x})) \leq \varepsilon$, or equivalently, $\mathbf{0} \in \partial f(\mathbf{x}) + \varepsilon \mathbb{B}$, where $\mathbb{B} := \mathbb{B}_1(\mathbf{0})$ and, for a set C , $\text{dist}(\mathbf{x}, C) := \inf_{\mathbf{y} \in C} \|\mathbf{x} - \mathbf{y}\|_2$. However, this naïve approximation is not computable in a very strong sense.

Theorem 3 (cf. [24, Theorem 5], [25, Proposition 11]). *For any $d, T \in \mathbb{N}$, $\varepsilon, 1 - \gamma \in [0, 1)$, and any possibly randomized algorithm \mathcal{A} interacting with a local oracle, there exists a 2-Lipschitz continuous, lower-bounded function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ such that the sequence $\{\mathbf{x}_t\}_t$ generated by the algorithm \mathcal{A} satisfies*

$$\text{dist}(\mathbf{0}, \partial f(\mathbf{x}_t)) > \varepsilon, \quad \forall t \in \{0, \dots, T\} \quad (5)$$

with probability at least $1 - \gamma$ (or deterministically if \mathcal{A} is deterministic).

Naturally, the hardness result in Theorem 3 still holds when the local oracle is replaced by the more practical but weaker subgradient or subdifferential oracle, together with access to function values. This is in sharp contrast to the smooth case, where an ε -stationary point can be obtained by calling a gradient oracle at most $O(1/\varepsilon^2)$ times. This leads to an immediate and intriguing question:

What type of approximate stationary point should we shoot for?

Before proposing any possible answer to this question, let us list some desirable properties that we wish a good concept of approximate stationarity to possess. First, for a wide class of functions, an approximate stationary point should be computable by some algorithm in reasonable time. We have seen before that ε -stationarity is too stringent a notion to be computable. Hence, further relaxation on the approximation is expected for nonsmooth functions. Second, the introduced approximation should be consistent, namely, when the parameters describing the degree of approximation (e.g., the parameter ε in ε -stationarity) go to zero, the set of stationary points $\{\mathbf{x} : \mathbf{0} \in \partial f(\mathbf{x})\}$ should be recovered. Requiring only computability without consistency can lead to trivial and meaningless solution concepts (e.g., one can define the whole space \mathbb{R}^d as the “solution set”). Besides, consistency naturally guarantees that if $\varepsilon_t \downarrow 0$ and \mathbf{x}_t is ε_t -stationary, then every limit point of $\{\mathbf{x}_t\}_t$ is a stationary point. Third, while consistency pertains to the asymptotic behavior of the approximation, from a non-asymptotic perspective, we also expect that the approximation should be as tight as possible, so as to exclude points that have nothing to do with the stationarity of the function.

In summary, we outline the following desiderata for an ideal concept of approximate stationarity.

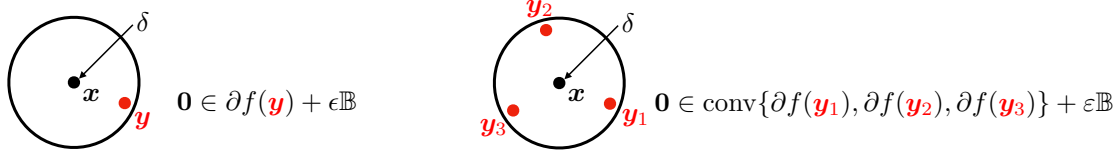


Fig. 2: (Left) Illustration of NAS in Definition 1: the point x is an (ε, δ) -NAS of f . (Right) Illustration of GAS in Definition 2: the point x is an (ε, δ) -GAS of f .

Desiderata of approximate stationarity concepts

- **Consistency:** As the approximation improves, the set of approximate solutions should converge to the exact solution set.
- **Computability:** For a broad class of functions, an approximate solution should be computable by some algorithm within reasonable time.
- **Tightness:** The approximation should be as precise as possible, without including too many irrelevant points.

In the remainder of this section, we discuss two possible solution concepts that satisfy these desiderata to varying degrees and study the convergence behavior of different algorithms for computing these concepts when they interact with the oracles mentioned earlier.

Near-Approximate Stationarity

The hardness result in Theorem 3 suggests that we need to relax the notion of ε -stationarity. One natural idea is to introduce an additional approximation to capture the neighborhood of stationary points. This motivates the following definition.

Definition 1 (NAS). We call a point x an (ε, δ) -near-approximate stationary (NAS) point, if there exists a point $y \in \mathbb{B}_\delta(x)$ such that y is ε -stationary. In other words, the point x is (ε, δ) -NAS if

$$\mathbf{0} \in \bigcup_{y \in \mathbb{B}_\delta(x)} \partial f(y) + \varepsilon \mathbb{B},$$

or equivalently, $\text{dist}(\mathbf{0}, \partial f(x + \delta \mathbb{B})) \leq \varepsilon$.²

Using Theorem 1(e), it can be shown that the set of (ε, δ) -NAS points $\{x : \mathbf{0} \in \partial f(x + \delta \mathbb{B}) + \varepsilon \mathbb{B}\}$ converges to $\{x : \mathbf{0} \in \partial f(x)\}$ when $\varepsilon, \delta \downarrow 0$. Therefore, the notion of NAS points is a consistent

²For a set S , we write $\partial f(S)$ for the set $\bigcup_{y \in S} \partial f(y)$.

approximation of stationary points. Such a simple relaxation already furnishes favorable algorithmic consequences for a specific class of nonconvex, nonsmooth functions, encompassing Example 1.

Theorem 4 (cf. [26, Theorem 3.1]). *Given any L -Lipschitz, ρ -weakly convex, Δ -lower-bounded³ function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, a stochastic subgradient-type method interacting with a stochastic subgradient oracle generates a sequence $\{\mathbf{x}_t\}_t$ satisfying*

$$\min_{0 \leq t \leq T} \mathbb{E} \left[\text{dist}(\mathbf{0}, \partial f(\mathbf{x}_t + \delta \mathbb{B})) \right] \leq \varepsilon \quad (6)$$

whenever $T \geq \text{poly}(\rho, L, \Delta, \varepsilon^{-1}, \delta^{-1})$.⁴

Consider $f(\mathbf{x}) = \mathbb{E}_{\xi}[\ell(\xi, \mathbf{x})]$, where $\ell(\xi, \cdot)$ is L -Lipschitz and ρ -weakly convex for almost every ξ , then f is also ρ -weakly convex. According to [2, Theorem 2.7.2], $\partial f(\mathbf{x}) = \mathbb{E}_{\xi}[\partial \ell(\xi, \cdot)(\mathbf{x})]$. A mini-batch subgradient estimate therefore provides a valid implementation of a stochastic subgradient oracle.

A key quantity in the analysis of [26] is the gradient norm of the Moreau envelope (see [4, Section 1.G]) of a ρ -weakly convex function. The work [26] shows that this gradient norm can be made smaller than ε in $\text{poly}(\rho, L, \Delta, \varepsilon^{-1})$ steps. Then, using a property of the Moreau envelope, one can obtain a polynomial complexity bound for computing (ε, δ) -NAS points by translating the gradient norm to upper bounds in terms of ε and δ in NAS; see [26, Section 2.2]. Since the main focus of this section is on computability in general, we hide these fine-grained polynomial terms and refer the reader to [26] for more precise details on complexity.

Weakly convex functions have interesting and useful applications (see Example 1), but this is not the end of our story, since the training of a simple shallow ReLU network in Example 2 is not a weakly convex problem. Indeed, even the function $t \mapsto -\max\{t, 0\}$ cannot be ρ -weakly convex for any $\rho \in \mathbb{R}$. To proceed, let us consider the problem of computing an (ε, δ) -NAS point for Lipschitz, lower-bounded functions, dropping the weak convexity assumption. It turns out that a strong hardness result holds for fairly general algorithms.

Theorem 5 (cf. [25, Theorem 1]). *Let $c > 0$ be a small numerical constant. For any $d \geq 2$, $T \leq c^d$, any $\varepsilon, \delta \in [0, c)$, and any possibly randomized algorithm \mathcal{A} interacting with a local oracle, there exists a $(1/c)$ -lower-bounded $(1/c)$ -Lipschitz function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ such that, with high probability, the sequence $\{\mathbf{x}_t\}_t$ generated by the algorithm \mathcal{A} satisfies*

$$\text{dist}(\mathbf{0}, \partial f(\mathbf{x}_t + \delta \mathbb{B})) > \varepsilon, \quad \forall t \in \{0, \dots, T\}.$$

³A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is Δ -lower-bounded if $f(\mathbf{0}) - \inf f \leq \Delta$.

⁴We use $\text{poly}(n_1, \dots, n_m)$ to denote a polynomial in terms of n_1, \dots, n_m .

For comparison, with the help of a classic tool in variational analysis called Ekeland’s variational principle (see [4, Proposition 10.44]) and a covering argument, one can show that a grid search procedure can find an (ε, δ) -NAS point in $O((1 + 1/(\varepsilon\delta))^d)$ steps, which is polynomial in ε^{-1} and δ^{-1} when the dimension d is fixed. In sum, we have seen that NAS, as a consistent approximation, can be computed efficiently for weakly convex Lipschitz functions but not for general Lipschitz functions. However, as shown in Example 2, even the training of two-layer ReLU neural networks is not a weakly convex problem. To obtain a computable approximate stationarity concept that is applicable to the training of popular deep learning models, we should consider further relaxations.

Goldstein Approximate Stationarity

While the Clarke subdifferential $\partial f(\mathbf{x})$ is a non-empty convex set for any $\mathbf{x} \in \mathbb{R}^d$, the union of sets $\partial f(\mathbf{x} + \delta\mathbb{B})$ in the definition of NAS is usually nonconvex for $\delta > 0$. Hence, an immediate relaxation of NAS is to take the convex hull over the set $\partial f(\mathbf{x} + \delta\mathbb{B})$, which yields the following definition.

Definition 2 (GAS). *We call a point \mathbf{x} an (ε, δ) -Goldstein approximate stationary (GAS) point if*

$$\mathbf{0} \in \text{conv } \partial f(\mathbf{x} + \delta\mathbb{B}) + \varepsilon\mathbb{B} = \text{conv} \left\{ \bigcup_{\mathbf{y} \in \mathbb{B}_\delta(\mathbf{x})} \partial f(\mathbf{y}) \right\} + \varepsilon\mathbb{B},$$

or equivalently, $\text{dist}(\mathbf{0}, \text{conv } \partial f(\mathbf{x} + \delta\mathbb{B})) \leq \varepsilon$.

The convex set $\text{conv } \partial f(\mathbf{x} + \delta\mathbb{B})$ that appears in Definition 2 can be traced back to Goldstein’s seminal work [27] on Lipschitz optimization and is usually called Goldstein’s δ -subdifferential, denoted $\partial_\delta f(\mathbf{x}) := \text{conv } \partial f(\mathbf{x} + \delta\mathbb{B})$ in the literature. The only difference between the definitions of GAS and NAS is the additional convex hull operator. It is easy to see that an (ε, δ) -NAS point is also an (ε, δ) -GAS point but not vice versa. This simple relaxation leads to a notion of approximate stationarity for general Lipschitz continuous functions that is much more amenable to computation.

Theorem 6 (cf. [24, Theorem 8], [28, Theorem 3.4], [29, Theorem 2.6]). *There exists a randomized algorithm that, given any L -Lipschitz, Δ -lower-bounded function, computes an (ε, δ) -GAS point with probability at least $1 - \gamma$ using at most*

$$\text{poly}(L, \Delta, \varepsilon^{-1}, \delta^{-1}, \log(\gamma^{-1})) \quad \text{function values and subgradient oracle calls.}$$

The algorithm mentioned in Theorem 6 is due to a recent randomized implementation of Goldstein’s conceptual scheme in [24]. The algorithm proposed in [24] can be implemented by a subdifferential oracle but not by a more practical subgradient oracle. In subsequent works [28], [29], by introducing extra randomization and making use of Rademacher’s Theorem, the subdifferential oracle can be replaced

by a subgradient oracle (or even a gradient oracle; see [28] for details). It is still unknown what the optimal oracle complexity is for computing an (ε, δ) -GAS point using a subgradient oracle. When only a stochastic estimate of subgradient is available, the work [30] provides an optimal algorithm with a rate of $O(\varepsilon^{-3}\delta^{-1})$, where the lower bound crucially relies on the presence of noise in the oracle.

A notable feature of these algorithms is their exploitation of randomization in algorithm design. Recall, however, that there is nothing uncertain about the computational problem itself: the objective function f is provided by a deterministic oracle, and the (ε, δ) -GAS points are defined deterministically. An intriguing question, which is posed in [24], is whether the algorithm mentioned in Theorem 6 can be derandomized. In other words, is there a deterministic algorithm that computes an (ε, δ) -GAS point using only $\text{poly}(L, \Delta, \varepsilon^{-1}, \delta^{-1})$ function values and subgradient oracle calls?

The answer is, perhaps surprisingly, negative.

Theorem 7 (cf. [31, Theorem 1], [32, Theorem 3.1]). *Let $c > 0$ be a small numerical constant. For any $T \in \mathbb{N}$, $d \geq T + 1$, any $\varepsilon, \delta \in [0, c)$, and any deterministic algorithm \mathcal{A} interacting with a local oracle, there exists a $(1/c)$ -Lipschitz continuous, piecewise affine, $(1/c)$ -lower-bounded function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ such that the sequence $\{\mathbf{x}_t\}_t$ generated by the algorithm \mathcal{A} satisfies*

$$\text{dist}(\mathbf{0}, \text{conv } \partial f(\mathbf{x}_t + \delta \mathbb{B})) > \varepsilon, \quad \forall t \in \{0, \dots, T\}. \quad (7)$$

In other words, no deterministic algorithm can achieve the result in Theorem 6 using even a local oracle.

The work [32] establishes the conclusion of the above theorem for deterministic algorithms that interact with a function value and subdifferential oracle, but the proof does not hold under a local oracle model. When only function value information is available, zero-order methods based on randomized smoothing can also be used to find an (ε, δ) -GAS point, albeit with dimension-dependent guarantees; see, e.g., [33].

TESTING OF SOLUTION CONCEPTS

Since computational resources are almost always limited, we are typically forced to stop the training algorithm at some point. In some cases, we may hold a model provided by others without access to detailed information about the parameters and/or algorithms used to train the deep network. Hence, some natural questions arise: How can we certify the quality of a given solution of an optimization problem? Is it a locally optimal or approximate stationary point of the objective function? For smooth functions, testing stationarity is a relatively easy task, as it is usually computationally cheap to evaluate the gradient norm and verify, for instance, that $\|\nabla f(\mathbf{x})\|_2 \leq \varepsilon$. For convex but possibly nonsmooth functions, similar concepts can also be verified efficiently due to the rich collection of convex subdifferential calculus

rules. However, testing of approximate stationarity for simple nonconvex, nonsmooth functions can be significantly more challenging.

In this section, we illustrate some recent developments in testing different solution concepts by examining two important classes of functions. The first class consists of polynomials, which are smooth (even analytic). A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is a degree- p polynomial if it can be written as

$$f(\mathbf{x}) = \sum_{\alpha \in \mathbb{N}^d: \|\alpha\|_1 \leq p} c_{\alpha} \cdot x_1^{\alpha_1} \cdots x_d^{\alpha_d},$$

where $c_{\alpha} \in \mathbb{R}$ is the coefficient of the monomial $x_1^{\alpha_1} \cdots x_d^{\alpha_d}$. The second class comprises piecewise affine functions, which can be nonconvex and nonsmooth. As we have seen in Example 4, the subdifferential of a two-layer ReLU network crucially depends on characterizing the subdifferential of the difference of two convex piecewise affine functions. As it turns out, any (possibly nonconvex) piecewise affine function can be written in such a form.

Theorem 8 (cf. [34, Proposition 4]). *Any piecewise affine function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ can be written as the difference of two convex piecewise affine functions $h, g : \mathbb{R}^d \rightarrow \mathbb{R}$; i.e., $f = h - g$.*

We use the standard representation for degree- p polynomials and express any convex piecewise affine function as a finite composition of summation and pointwise maximum operations, which provides a natural representation for the functions in Example 4.

Local Optimality

Among the various solution concepts in optimization, global and local minimizers are usually considered the most desirable types. However, even verifying the local optimality of a given point, when done by definition, requires examining all function values around the concerned point, which is computationally demanding. In fact, for certain smooth and nonsmooth functions, determining whether a given point is a local minimizer is already computationally hard.

Theorem 9 (cf. [35, Section 4], [36, Table 1], [16, Theorem 3.2(a)]). *The problems of testing whether a given point is a local minimizer of*

- 1) *degree- p polynomials with $p \geq 4$, and*
 - 2) *difference of two convex piecewise affine functions*
- are both strongly co-NP-complete.*

The two problems in Theorem 9 both have very simple structures. Remarkably, the degree-4 assumption for polynomials is actually tight, since [36, Theorem 3.3] shows that local optimality can be verified in

polynomial time for degree- p polynomials when $p \in \{0, 1, 2, 3\}$. Moreover, when allowing the presence of linear constraints, verifying local optimality and the second-order necessary condition are both co-NP-hard even for quadratic polynomials; see [35, Theorem 2] and [37, Theorem 4.2].

First-Order Stationarity

Since testing local optimality can be computationally intractable even for smooth functions, we turn to the task of testing first-order stationarity. In contrast to Theorem 9, it is easy to see that testing ε -stationarity for degree- p polynomials can be performed in polynomial time for any $p \in \mathbb{N}$. However, for piecewise affine functions, stationarity testing remains computationally intractable unless $P = NP$.

Theorem 10 (cf. [16, Theorem 3.2(b)]). *Checking whether a given point is Clarke stationary of the difference of two convex piecewise affine functions is strongly NP-complete.*

The NP-hardness part in Theorem 10 holds even when we only want to test stationarity approximately. A corollary of Theorem 10 pertains to Convolutional Neural Networks (CNNs), one of the most popular architectures for image classification, and shows that determining whether a point is ε -stationary for the loss function of training a CNN is NP-hard; see [16, Corollary 3.8] for details.

Complementing these general hardness results, efficient testing becomes possible when certain regularity conditions are satisfied. Considering the loss of ReLU networks in Example 4, one such condition is the intersection-of-parallel-subspace condition discussed therein, which precisely characterizes when a strong sum rule holds (cf. Theorem 1(g)). The work [38] provides another sufficient condition for this strong sum rule, requiring the data points $\{\mathbf{x}_n\}_{n=1}^N$ to be in general position in a specific sense.

Definition 3 (general position). *The points $\{\mathbf{x}_n\}_{n=1}^N \subseteq \mathbb{R}^d$ are in general position if no $d + 1$ points lie on the same affine hyperplane.*

If the distribution of the data is absolutely continuous with respect to the Lebesgue measure, then the generated iid samples $\{\mathbf{x}_n\}_{n=1}^N$ are almost surely in general position.

Theorem 11 (cf. [38, Section 3.1]). *Assuming that the data points are in general position, testing ε -stationarity for Example 2 can be done by solving polynomially many quadratic programs.*

DC-Criticality

Given the above hardness results, one possible way to proceed is to consider other surrogate solution concepts that are computationally easier to verify. Since the hardness result in Theorem 10 is for

difference-of-convex (DC) functions, which is pervasive in optimization and statistics [39], it is natural to look at a classic solution concept in the DC-programming literature, called DC-criticality.

Definition 4. A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is called a *difference-of-convex (DC) function* if it can be written as the difference of two convex functions $h, g : \mathbb{R}^d \rightarrow \mathbb{R}$; i.e., $f = h - g$. We call a point $\mathbf{x} \in \mathbb{R}^d$ a *DC-critical point* of f if $\mathbf{0} \in \partial h(\mathbf{x}) - \partial g(\mathbf{x})$, or equivalently, $\emptyset \neq \partial h(\mathbf{x}) \cap \partial g(\mathbf{x})$.

For convex functions h, g and a point \mathbf{x} , verifying DC-criticality (i.e., $\mathbf{0} \in \partial h(\mathbf{x}) - \partial g(\mathbf{x})$) is considerably easier than verifying stationarity (i.e., $\mathbf{0} \in \partial(h - g)(\mathbf{x})$), since the convex subdifferentials $\partial h(\mathbf{x})$ and $\partial g(\mathbf{x})$ enjoy more complete and satisfactory calculus rules than the Clarke subdifferential $\partial(h - g)(\mathbf{x})$. Therefore, DC-criticality is a more favorable choice at least from the solution testing perspective. Indeed, the algorithm in [38, Section 3.1] can be viewed as verifying DC-criticality, which is equivalent to stationarity when the data points are in general position. However, we should note that DC-criticality is, in general, a weaker solution concept than (Clarke) stationarity, and distinguishing between them can still be NP-hard; see [16, Corollary 3.6]. Moreover, DC-criticality depends on the specific DC decomposition of $f := h - g$, which is not unique (e.g., one can write $f = (h + \|\cdot\|_2) - (g + \|\cdot\|_2)$). For any point $\mathbf{x}_0 \in \mathbb{R}^d$ and any DC function f with convex components h, g , there always exists an unfavorable DC decomposition of f such that \mathbf{x}_0 becomes DC-critical with respect to this new decomposition. This can be easily seen by examining the following new DC decomposition of $f = \hat{h} - \hat{g}$, where \hat{h}, \hat{g} are convex and defined as follows:

$$\hat{h} := \max\{h, g + f(\mathbf{x}_0)\} + h, \quad \hat{g} := \max\{h, g + f(\mathbf{x}_0)\} + g.$$

For any $\mathbf{g}_1 \in \partial h(\mathbf{x}_0), \mathbf{g}_2 \in \partial g(\mathbf{x}_0)$, one has $\mathbf{g}_1 + \mathbf{g}_2 \in \partial \hat{h}(\mathbf{x}_0) \cap \partial \hat{g}(\mathbf{x}_0)$, hence \mathbf{x}_0 is DC-critical. Hence, in general, DC-criticality is not a representation-independent solution concept, and any point can be a DC-critical point of a DC function.

Towards Stopping Criteria

In the previous subsections, we have discussed exact testing of solutions, that is, the input point is exactly the point for solution verification. However, for a numerical algorithm, reaching the exact optimal or stationary point is virtually impossible. For instance, consider running the subgradient method on the convex function $x \mapsto |x|$. For $\varepsilon \in [0, 1)$, it is clear that the only ε -stationary point of the function is $x = 0$, which cannot be reached for almost every initial point and step size. Therefore, there is a need to develop a *robust* solution testing approach that, given a query point \mathbf{x} , certifies or refutes the existence of a solution *near* \mathbf{x} .

For Lipschitz functions, when seeking GAS points, the randomized algorithm in Theorem 6 is of the Las Vegas type,⁵ and can return a GAS point together with an explicit certificate. Thus, stopping and verification are straightforward when a certificate is available. Without a certificate, however, testing whether a given point is GAS remains highly nontrivial. For the more desirable NAS points, Theorem 4 provides, under the important weak convexity assumption, a polynomial upper bound on the number of iterations T required to ensure that $\{\mathbf{x}_t\}_{t=0}^T$ contains at least one NAS point. In this case, one may simply return the point in $\{\mathbf{x}_t\}_{t=0}^T$ with the smallest objective value, which is guaranteed to be no worse (in objective value) than an NAS point. Beyond weak convexity, the situation is more challenging. Asymptotic results (e.g., [40]) imply that, under mild assumptions, any cluster point of subgradient iterates is stationary, and hence there exists a finite T such that \mathbf{x}_T is a NAS point. However, obtaining a usable finite bound on T can be difficult (see [31, Theorem 2]), so a principled stopping rule is unclear. For ReLU networks, robust solution testing is also an important direction; see [38, Section 5].

For piecewise affine functions, the following result takes a step toward robust testing for NAS that can be used for the ReLU networks studied in [38]. It is intriguing to ask whether similar results hold for other important function classes beyond weak convexity.

Theorem 12 (cf. [16, Corollary 5.10]). *Let $h, g : \mathbb{R}^d \rightarrow \mathbb{R}$ be two given convex piecewise affine functions. Suppose that we have an exact testing oracle for $h - g$, and that the sequence $\{\mathbf{x}_t\}_t \subseteq \mathbb{B}_r(\mathbf{0})$ is bounded. There exist a constant $\delta^* > 0$, depending only on h, g , and r , and a polynomial-time algorithm \mathcal{A} that, for any $t \in \mathbb{N}$ and $\delta > 0$, calls the exact testing oracle at most polynomially many times and either*

- 1) *finds a point $\hat{\mathbf{x}}$ such that $\|\mathbf{x}_t - \hat{\mathbf{x}}\|_2 \leq \delta$ and $\hat{\mathbf{x}}$ is a solution, or*
- 2) *asserts that every solution $\hat{\mathbf{x}}$ satisfies $\|\mathbf{x}_t - \hat{\mathbf{x}}\|_2 > \min\{\delta, \delta^*\}$.*

CONCLUSION AND OPEN DIRECTIONS

In this article, we have examined three key computational problems in large-scale nonconvex, non-smooth optimization, with particular emphasis on training modern deep neural networks. We have highlighted recent advances in these areas and illustrated challenges that are largely absent in their convex or smooth counterparts. Throughout the paper, we have discussed several open questions; here, we mention three to motivate further developments. First, more efficient computation of subgradients is desirable, since lexicographic subgradients are computationally demanding. Second, the optimal convergence rate for computing GAS using a deterministic subgradient oracle remains unknown. Third, for more general function classes, can we establish finite-time termination criteria for subgradient methods and related

⁵A Las Vegas algorithm is a randomized algorithm that always gives correct results but has a random runtime.

algorithms? We hope this introductory discussion serves as a starting point for readers interested in exploring and advancing the algorithmic aspects of nonsmooth analysis and optimization.

REFERENCES

- [1] F. Bach, *Learning Theory from First Principles*, MIT Press, 2024.
- [2] F. H. Clarke, *Optimization and Nonsmooth Analysis*, SIAM, 1990.
- [3] R. T. Rockafellar, *Convex Analysis*, vol. 18, Princeton University Press, 1970.
- [4] R. T. Rockafellar and R. J-B Wets, *Variational Analysis*, vol. 317, Springer Science & Business Media, 2009.
- [5] Y. Lucet, “What shape is your conjugate? A survey of computational convex analysis and its applications,” *SIAM Review*, vol. 52, no. 3, pp. 505–542, 2010.
- [6] Yu. Nesterov, *Lectures on Convex Optimization*, vol. 137, Springer, 2018.
- [7] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, 2008.
- [8] J. Li, A. M.-C. So, and W.-K. Ma, “Understanding notions of stationarity in nonsmooth optimization: A guided tour of various constructions of subdifferential for nonsmooth functions,” *IEEE Signal Processing Magazine*, vol. 37, no. 5, pp. 18–31, 2020.
- [9] R. T. Rockafellar, “Extensions of subgradient calculus with applications to optimization,” *Nonlinear Analysis: Theory, Methods & Applications*, vol. 9, no. 7, pp. 665–698, 1985.
- [10] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, SIAM, 2021.
- [11] Y. Cui and J.-S. Pang, *Modern Nonconvex Nondifferentiable Optimization*, SIAM, 2021.
- [12] S. M. Kakade and J. D. Lee, “Provably correct automatic sub-differentiation for qualified programs,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [13] W. Lee, H. Yu, X. Rival, and H. Yang, “On correctness of automatic differentiation for non-differentiable functions,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 6719–6730, 2020.
- [14] J. Bolte and E. Pauwels, “Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning,” *Mathematical Programming*, vol. 188, no. 1, pp. 19–51, 2021.
- [15] D. Drusvyatskiy and C. Paquette, “Efficiency of minimizing compositions of convex functions and smooth maps,” *Mathematical Programming*, vol. 178, pp. 503–558, 2019.
- [16] L. Tian and A. M.-C. So, “Testing approximate stationarity concepts for piecewise affine functions,” in *ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2025.
- [17] Yu. Nesterov, “Lexicographic differentiation of nonsmooth functions,” *Mathematical Programming*, vol. 104, pp. 669–700, 2005.
- [18] K. A. Khan and P. I. Barton, “A vector forward mode of automatic differentiation for generalized derivative evaluation,” *Optimization Methods and Software*, vol. 30, no. 6, pp. 1185–1212, 2015.
- [19] C. Ji, H. Zhang, and S. Mitra, “Reachability for nonsmooth systems with lexicographic Jacobians,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2025, pp. 364–384.
- [20] A. Griewank, “On stable piecewise linearization and generalized algorithmic differentiation,” *Optimization Methods and Software*, vol. 28, no. 6, pp. 1139–1178, 2013.
- [21] A. Griewank and Á. Rojas, “Generalized abs-linear learning,” in *Program Transformations for ML Workshop at NeurIPS*, 2019.

- [22] A. Griewank and A. Walther, “Beyond the oracle: Opportunities of piecewise differentiation,” in *Numerical Nonsmooth Optimization: State of the Art Algorithms*, pp. 331–361. Springer, 2020.
- [23] A. Nemirovski and D. Yudin, *Problem Complexity and Method Efficiency in Optimization*, Wiley-Interscience, 1983.
- [24] J. Zhang, H. Lin, S. Jegelka, A. Jadbabaie, and S. Sra, “Complexity of finding stationary points of nonsmooth nonconvex functions,” in *International Conference on Machine Learning*, 2020, pp. 11173–11182.
- [25] G. Kornowski and O. Shamir, “Oracle complexity in nonsmooth nonconvex optimization,” *Journal of Machine Learning Research*, vol. 23, no. 314, pp. 1–44, 2022.
- [26] D. Davis and D. Drusvyatskiy, “Stochastic model-based minimization of weakly convex functions,” *SIAM Journal on Optimization*, vol. 29, no. 1, pp. 207–239, 2019.
- [27] A. A. Goldstein, “Optimization of Lipschitz continuous functions,” *Mathematical Programming*, vol. 13, no. 1, pp. 14–22, 1977.
- [28] L. Tian, K. Zhou, and A. M.-C. So, “On the finite-time complexity and practical computation of approximate stationarity concepts of Lipschitz functions,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 21360–21379.
- [29] D. Davis, D. Drusvyatskiy, Y. T. Lee, S. Padmanabhan, and G. Ye, “A gradient sampling method with complexity guarantees for Lipschitz functions in high and low dimensions,” in *Advances in Neural Information Processing Systems*, 2022, pp. 6692–6703.
- [30] A. Cutkosky, H. Mehta, and F. Orabona, “Optimal stochastic non-smooth non-convex optimization through online-to-non-convex conversion,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 6643–6670.
- [31] L. Tian and A. M.-C. So, “No dimension-free deterministic algorithm computes approximate stationarities of Lipschitzians,” *Mathematical Programming*, vol. 208, pp. 51–74, 2024.
- [32] M. Jordan, G. Kornowski, T. Lin, O. Shamir, and M. Zampetakis, “Deterministic nonsmooth nonconvex optimization,” in *Conference on Learning Theory*. PMLR, 2023, pp. 4570–4597.
- [33] T. Lin, Z. Zheng, and M. I. Jordan, “Gradient-free methods for deterministic and stochastic nonsmooth nonconvex optimization,” in *Advances in Neural Information Processing Systems*, 2022, pp. 26160–26175.
- [34] D. Melzer, “On the expressibility of piecewise-linear continuous functions as the difference of two piecewise-linear convex functions,” *Mathematical Programming Studies*, vol. 29, pp. 118–134, 1986.
- [35] K. G. Murty and S. N. Kabadi, “Some NP-complete problems in quadratic and nonlinear programming,” *Mathematical Programming*, vol. 39, pp. 117–129, 1987.
- [36] A. A. Ahmadi and J. Zhang, “Complexity aspects of local minima and related notions,” *Advances in Mathematics*, vol. 397, pp. 108119, 2022.
- [37] M. Nouiehed, J. D. Lee, and M. Razaviyayn, “Convergence to second-order stationarity for constrained non-convex optimization,” *arXiv preprint arXiv:1810.02024*, 2018.
- [38] C. Yun, S. Sra, and A. Jadbabaie, “Efficiently testing local optimality and escaping saddles for ReLU networks,” in *International Conference on Learning Representations*, 2019.
- [39] M. Nouiehed, J.-S. Pang, and M. Razaviyayn, “On the pervasiveness of difference-convexity in optimization and statistics,” *Mathematical Programming*, vol. 174, no. 1, pp. 195–222, 2019.
- [40] D. Davis, D. Drusvyatskiy, S. Kakade, and J. D. Lee, “Stochastic subgradient method converges on tame functions,” *Foundations of Computational Mathematics*, vol. 20, no. 1, pp. 119–154, 2020.

BIOGRAPHIES

Lai Tian received the Ph.D. degree from the Department of Systems Engineering and Engineering Management of The Chinese University of Hong Kong (CUHK). His research interests lie at the interface of mathematical optimization and computer science, with an emphasis on modern variational problems in data-driven decision making, machine learning, and data science.

Anthony Man-Cho So (*Fellow, IEEE*) is currently Dean of the Graduate School, Acting Master of Morningside College, and Professor in the Department of Systems Engineering and Engineering Management of The Chinese University of Hong Kong (CUHK). His research focuses on optimization theory and its applications in various areas of science and engineering, including computational geometry, machine learning, signal processing, and statistics. Dr. So currently serves on the editorial boards of Journal of Global Optimization, Mathematical Programming, Mathematics of Operations Research, Open Journal of Mathematical Optimization, and Optimization Methods and Software. He has also served as the Lead Guest Editor of IEEE Signal Processing Magazine Special Issue on Non-Convex Optimization for Signal Processing and Machine Learning.