mathematics

MDPI

Article

Efficient Automatic Subdifferentiation for Programs with Linear Branches

Sejun Park

Department of Artificial Intelligence, Korea University, Seoul 02841, Republic of Korea; sejunpark@korea.ac.kr

Abstract: Computing an element of the Clarke subdifferential of a function represented by a program is an important problem in modern non-smooth optimization. Existing algorithms either are computationally inefficient in the sense that the computational cost depends on the input dimension or can only cover simple programs such as polynomial functions with branches. In this work, we show that a generalization of the latter algorithm can efficiently compute an element of the Clarke subdifferential for programs consisting of analytic functions and linear branches, which can represent various non-smooth functions such as max, absolute values, and piecewise analytic functions with linear boundaries, as well as any program consisting of these functions such as neural networks with non-smooth activation functions. Our algorithm first finds a sequence of branches used for computing the function value at a random perturbation of the input; then, it returns an element of the Clarke subdifferential by running the backward pass of the reverse-mode automatic differentiation following those branches. The computational cost of our algorithm is at most that of the function evaluation multiplied by some constant independent of the input dimension n, if a program consists of piecewise analytic functions defined by linear branches, whose arities and maximum depths of branches are independent of n.

Keywords: automatic differentiation; Clarke subdifferential; algorithm

MSC: 68Q25; 68W40

Citation: Park, S. Efficient Automatic Subdifferentiation for Programs with Linear Branches. *Mathematics* **2023**, *11*, 4858. https://doi.org/10.3390/math11234858

Academic Editor: Ivan Lirkov

Received: 2 November 2023
Revised: 25 November 2023
Accepted: 29 November 2023
Published: 3 December 2023

Copyright: © 2023 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Automatic differentiation refers to various techniques to compute the derivatives of a function represented by a program, based on the well-known chain rule of calculus. It has been widely used across various domains, and diverse practical automatic differentiation systems have been developed [1–3]. In particular, reverse-mode automatic differentiation [4] has been a driving force of the rapid advances in numerical optimization [5–7].

There are two important properties of reverse-mode automatic differentiation: correctness and efficiency. For programs consisting of smooth functions, it is well known that reverse-mode automatic differentiation always computes the correct derivatives [8,9]. Furthermore, for programs returning a scalar value, it efficiently computes their derivatives in the sense that its computational cost is at most proportional to that of the function evaluation, where the additional multiplicative factor is bounded by five for rational functions [10,11] and by some constant that depends on the underlying implementation of smooth functions; if the arities of the functions are independent of the input dimension n, then this constant is also independent of n [12]. Such correctness and efficiency of reverse-mode automatic differentiation, referred to as the *Cheap Gradient Principle*, have been of central importance for modern nonlinear optimization algorithms [13].

In practical problems, a program often involves branches (e.g., max and an absolute value), and the corresponding target function can be non-smooth. In other words, the derivative of the program may not exist at some inputs. In this work, we investigated a *Cheap Subgradient Principle*: an efficient algorithm that correctly computes an element of

the Clarke subdifferential, a generalized notion of the derivative, for scalar programs. One naïve approach is to directly apply reverse-mode automatic differentiation to the Clarke subdifferential. Such a method is computationally cheap as in the smooth case; however, due to the absence of the sharp chain rule for the Clarke subdifferential, it is incorrect in general even if the target function is differentiable [8,14,15].

There have been extensive research efforts to correctly compute an element of the Clarke subdifferential. A notable line of work is based on the lexicographic subdifferential [16], which is a subset of the Clarke subdifferential, but a sharp chain rule holds under some structural assumptions. Based on this, a series of works [17–20] has shown that an element of the lexicographic subdifferential can be computed by evaluating n directional derivatives, where n denotes the input dimension. Nevertheless, since this approach requires computing n directional derivatives, it incurs a multiplicative factor n in its computational cost compared to that of function evaluation in the worst case.

To avoid such an input-dimension-dependent factor, a two-step randomized algorithm for programs with branches has been proposed [14]. In the first step, the algorithm chooses a random direction δ and finds a sequence of branch selections based on the directional derivative with respect to δ under some qualification condition [14,21]. Then, the second step computes the derivative corresponding to the branches returned in the first step, which is shown to be an element of the Clarke subdifferential. Here, the second step can also be efficiently implemented via reverse-mode automatic differentiation. As a result, this two-step algorithm correctly computes an element of the Clarke subdifferential under the qualification condition, with the computational cost independent of the input dimension. However, this result is only for piecewise polynomial programs, defined by branches and finite compositions of monomials and affine functions.

In this work, we propose an efficient automatic subdifferentiation algorithm by generalizing the algorithm in [14] described above. Our algorithm correctly computes an element of the Clarke subdifferential for programs consisting of any analytic functions (including polynomials) and linear branches. As in the prior efficient automatic (sub)differentiation works, the computational cost of our algorithm is that of the function evaluation multiplied by some constant independent of the input dimension n, if a program consists of piecewise analytic functions (defined by linear branches) whose arities and maximum depths of branches are independent of n (e.g., max and the absolute value).

Related Works

Non-smooth optimization: Although smooth functions are easy to formulate and optimize, they have limited applicability as non-smoothness appears in various science and engineering problems. For example, real-world problems in thermodynamics often involve discrete switching between thermodynamic phases, which can be modeled by non-smooth functions. Dynamic simulation and optimization under these models often require the treatment of these models [22,23]. In machine learning applications, hinge loss, $\mathrm{ReLU}(x) = \max\{x, 0\}$, and maxpool operations are often used, which makes the optimization objective non-smooth [6,24,25]. For optimizing convex, but non-smooth functions, subderivative methods are widely used for approximating a local minimum [26,27]. However, for non-convex functions, the subderivative does not exist in general, and researchers have investigated the generalized notion of derivatives (e.g., the Clarke subdifferential).

Optimization algorithms using generalized derivatives: Recently, the convergence properties of optimization algorithms based on generalized derivatives for non-convex and non-smooth functions have received much attention. Ref. [28] proved that, for locally Lipschitz functions, the stochastic gradient method, where the gradient is chosen from the Clarke subdifferential, converges to a stationary point. However, as we introduced in the previous section, computing an element of the Clarke subdifferential is computationally expensive or can be efficient only for a specific class of programs. Ref. [29] proposed a new notion of gradient called the conservative gradient, which can be efficiently com-

puted; nevertheless, its convergence property is not well understood, especially under practical setups.

2. Problem Setup

2.1. Notations

For $n \in \mathbb{N} \cup \{0\}$, we denote $[n] \triangleq \{1, \ldots, n\}$, where $[0] = \emptyset$. For any set $\mathcal{S}$, we use $\mathcal{S}^0 \triangleq \{()\}$, where $()$ denotes the zero-dimensional vector. For any vector $x \in \mathcal{S}^n$ and $i \in [n]$, we denote x_i for the i-th coordinate of x and $x_{:i} \triangleq (x_1, \ldots, x_{i-1})$, where $x_{:1} = ()$; similarly, for any $x \in \mathcal{S}^n$ and index set $\mathcal{I} = \{i_1, \ldots, i_k\} \subseteq [n]$ with $i_1 < \cdots < i_k$, we use $x_{\mathcal{I}} \triangleq (x_{i_1}, \ldots, x_{i_k})$. For any $u = (u_1, \ldots, u_n) \in \mathbb{R}^n$ and $v = (v_1, \ldots, v_m) \in \mathbb{R}^m$, we use $u \oplus v \triangleq (u_1, \ldots, u_n, v_1, \ldots, v_m)$; when $n = m$, we write $\langle u, v \rangle$ to denote the standard inner product between u and v. For any $x \in \mathbb{R}$, $\operatorname{sign}(x) = 1$ if $x > 0$ and $\operatorname{sign}(x) = -1$ otherwise. For any real-valued vector x, $\operatorname{len}(x)$ denotes the length of x, i.e., $\operatorname{len}(x) = n$ if $x \in \mathbb{R}^n$. For any set $\mathcal{S} \subseteq \mathbb{R}^n$, we use $\operatorname{cl}(\mathcal{S})$, $\operatorname{int}(\mathcal{S})$, and $\operatorname{conv}(\mathcal{S})$ to denote the closure, interior, and convex hull of $\mathcal{S}$, respectively. We lastly define the Clarke subdifferential. Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and the set $\mathcal{D} \subseteq \mathbb{R}^n$ of all points at which f is differentiable, the Clarke subdifferential of f at $x \in \mathbb{R}^n$ is a set defined as

$$\partial^c f(x) \triangleq \operatorname{conv}\Big\{ s \in \mathbb{R}^n : \exists \{u_t\}_{t \in \mathbb{N}} \subseteq \mathcal{D} \text{ such that } u_t \to x \text{ and } \nabla f(u_t) \to s \Big\}.$$

2.2. Programs with Branches

We considered a program P defined in Figure 1 (left). P applies a series of *primitive functions* $F_{n+1}, \ldots, F_{n+m}$ to compute intermediate variables $v_{n+1}, \ldots, v_{n+m}$ and, then, returns the last result v_{n+m}. Each primitive function $F_i : \mathbb{R}^{d_i} \to \mathbb{R}$ is continuous and defined in an inductive way as in Figure 1 (right): F_i either applies a function $f_{i,()}$ or branches via (possibly nested) if–else statements. Namely, F_i is a continuous, piecewise function. If F_i branches with an input $(x_1, \ldots, x_{d_i})$, then it first evaluates $y = \phi_{i,()}(x_1, \ldots, x_{d_i})$ for some $\phi_{i,()} : \mathbb{R}^{d_i} \to \mathbb{R}$ and checks whether $y > c_{i,()}$ or not for some threshold value $c_{i,()} \in \mathbb{R}$. If $y > c_{i,()}$, it executes a code $E_{i,(1)}$ that either applies a function $f_{i,(1)}$ or executes another code $E_{i,(1,1)}$ or $E_{i,(1,-1)}$ depending on whether $\phi_{i,(1)}(x_1, \ldots, x_{d_i}) > c_{i,(1)}$ or not. The case $y \le c_{i,()}$ is handled in a similar way. Here, we assumed that each F_i has finitely many branches. In short, each primitive function F_i first finds a proper piece labeled by $p \in \bigcup_{j=0}^{\infty} \{-1, 1\}^j$ and, then, returns $f_{i,p}(x_1, \ldots, x_{d_i})$. We illustrate a flow chart for a primitive function F_i in Figure 2.

We present an example code for a primitive function F_i that returns $\max\{x_1, x_2, x_3\}$ in Figure 3. In this example, F_i branches at most twice: its first branch is determined by $\phi_{i,()}(x_1, x_2, x_3) = x_1 - x_2$, which is stored in y in the second line in Figure 3. If $y > 0$ (the third line with $c_{i,()} = 0$), then it executes $E_{1,(1)}$, which corresponds to lines 4–6. Otherwise, it moves to $E_{1,(-1)}$, which corresponds to the lines 8–10. Suppose $E_{1,(1)}$ is executed (i.e., $y > 0$ in line 3). Then, it computes $\phi_{i,(1)}(x_1, x_2, x_3) = x_1 - x_3$ and stores it in y as in line 4. If $y > 0$, then $E_{i,(1,1)}$ is executed, which returns the value of x_1 (i.e., $f_{i,(1,1)}(x_1, x_2, x_3) = x_1$). Otherwise, $E_{i,(1,-1)}$ is executed, which returns the value of x_3 (i.e., $f_{i,(1,-1)}(x_1, x_2, x_3) = x_3$).

$$\begin{array}{l} \texttt{prog}\ P(w_1, \ldots, w_n)\{ \\ \quad (v_1, \ldots, v_n) = (w_1, \ldots, w_n); \\ \quad v_{n+1} = F_{n+1}(v_{pa(n+1)}); \\ \qquad \vdots \\ \quad v_{n+m} = F_{n+m}(v_{pa(n+m)}); \\ \quad \texttt{return}\ v_{n+m} \\ \} \end{array}$$

$$\begin{array}{l} \texttt{func}\ F_i(x_1, \ldots, x_{d_i})\{ \\ \quad E_{i,()} \\ \} \end{array}$$

$$E_{i,p} \triangleq \begin{cases} \texttt{return}\ f_{i,p}(x_1, \ldots, x_{d_i}), \ \text{ or} \\ \begin{pmatrix} y = \phi_{i,p}(x_1, \ldots, x_{d_i}); \\ \texttt{if}\ (y > c_{i,p})\{\ E_{i,p\oplus(1)}\ \} \\ \texttt{else}\ \{\ E_{i,p\oplus(-1)}\ \} \end{pmatrix} \end{cases}$$

Figure 1. Definitions of a program P (**left**) and primitive functions $F_{n+1}, \ldots, F_{n+m}$ (**right**).

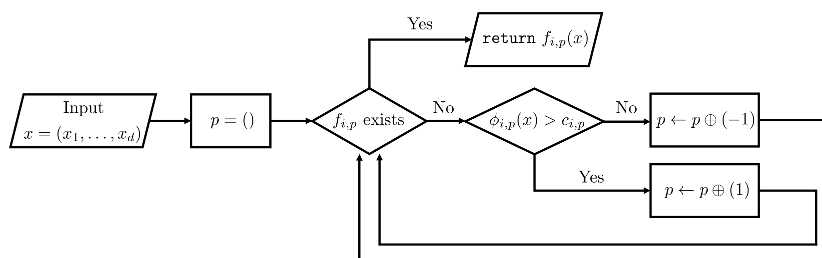

Figure 2. A flow chart illustrating a primitive function F_i.

```
func Fi(x1, x2, x3) {
   y = x1 - x2;   // φ_{i,()}(x) = x1 − x2, c_{i,()} = 0
   if (y > 0) {
      y = x1 - x3;                 // φ_{i,(1)}(x) = x1 − x3, c_{i,(1)} = 0
      if (y > 0) { return x1 }     // f_{i,(1,1)}(x) = x1
      else       { return x3 }     // f_{i,(1,−1)}(x) = x3
   } else {
      y = x2 - x3;                 // φ_{i,(−1)}(x) = x2 − x3, c_{i,(−1)} = 0
      if (y > 0) { return x2 }     // f_{i,(−1,1)}(x) = x2
      else       { return x3 }     // f_{i,(−1,−1)}(x) = x3
   }
}
```

Figure 3. Example code for the max function $F_i : (x_1, x_2, x_3) \mapsto \max\{x_1, x_2, x_3\}$.

We considered each v_i and $v_{pa(i)}$ as a function of an input $w \in \mathbb{R}^n$. Specifically, for all $i \in [n]$, we use $v_i(w) \triangleq w_i$; and for all $i \in [n+m] \setminus [n]$, we use $v_i(w) \triangleq F_i(v_{pa(i)}(w))$ and $v_{pa(i)}(w) \triangleq (v_{j_1}(w), \ldots, v_{j_{d_i}}(w))$, where $pa(i) = \{j_1, \ldots, j_{d_i}\}$ with $0 < j_1 < \cdots < j_{d_i} < i$. Under this notation, $w \mapsto v_{n+m}(w)$ denotes the *target function* represented by the program P. We often omit w and write v_i and $v_{pa(i)}$ if it is clear from the context. We denote the gradient (or Jacobian) of functions with respect to an input w by the operator D: e.g., $Dv_i(w) \triangleq \partial v_i(w)/\partial w$ and $D\phi_{i,p}(v_{pa(i)}(w)) \triangleq \partial \phi_{i,p}(v_{pa(i)}(w))/\partial w$.

Throughout this paper, we focus on programs with linear branches; that is, each $\phi_{i,p}(x_1, \ldots, x_d)$ is linear in $x_1, \ldots, x_d$. Primitive functions with linear branches can express fundamental non-smooth functions such as max, the absolute value, bilinear interpolation, and any piecewise analytic functions with finite linear boundaries. They have been widely used in various fields including machine learning, electrical engineering, and non-smooth analysis. For example, max–min representation (or the abs–normal form) has been extensively studied in non-smooth analysis [30,31]. Furthermore, neural networks using the $\mathrm{ReLU}(x) = \max\{x, 0\}$ activation function and maxpool operations are widely used in machine learning, computer vision, load forecasting, etc. [6,24,25]. The assumption on linear branches will be formally introduced in Assumption 1 in Section 3.1.

2.3. Pieces of Programs

We introduce useful notations here. For each $i \in [n+m] \setminus [n]$, we define the set of the pieces of F_i as

$$\Gamma_i \triangleq \left\{ p \in \bigcup_{j=0}^{\infty} \{-1, 1\}^j : E_{i,p} = \texttt{return}\ f_{i,p}(x_1, \ldots, x_{d_i}) \right\}.$$

We also define $\Gamma \triangleq \{()\}^n \times \prod_{i=n+1}^{m} \Gamma_i$ for the pieces of the overall program. Here, we include an auxiliary piece $()$ for the first n indices so that $\gamma_i \in \Gamma_i$ for any $\gamma = (\gamma_1, \ldots, \gamma_{n+m}) \in \Gamma$ and $i \in [n+m] \setminus [n]$. For each $i \in [n+m] \setminus [n]$, we define the set of the inputs to F_i that corresponds to the piece $p \in \Gamma_i$, as

$$\mathcal{X}_{i,p} \triangleq \{x \in \mathbb{R}^{d_i} : F_i \text{ selects } f_{i,p} \text{ at the input } x\}.$$

Then, $\{\mathcal{X}_{i,p} : p \in \Gamma_i\}$ forms a partition of $\mathbb{R}^{d_i}$. Likewise, we also define the set of the inputs to the overall program P that corresponds to $\gamma \in \Gamma$, as

$$\mathcal{W}_\gamma \triangleq \{w \in \mathbb{R}^n : v_{pa(i)}(w) \in \mathcal{X}_{i,\gamma_i} \text{ for all } i \in [n+m] \setminus [n]\}.$$

Then, $\{\mathcal{W}_\gamma : \gamma \in \Gamma\}$ forms a partition of $\mathbb{R}^n$. Lastly, for each $\gamma \in \Gamma$, we inductively define the function $v_{i,\gamma} : \mathbb{R}^n \to \mathbb{R}$ that corresponds to $v_i(\cdot)$, but is obtained by using the γ_j piece of each F_j, as

$$v_{i,\gamma}(w) \triangleq \begin{cases} w_i & \text{if } i \in [n] \\ f_{i,\gamma_i}(v_{pa(i),\gamma}(w)) & \text{if } i \in [n+m] \setminus [n], \end{cases}$$

where $v_{pa(i),\gamma}(\cdot)$ is defined as in $v_{pa(i)}(\cdot)$. Then, $v_{i,\gamma}(\cdot)$ coincides with $v_i(\cdot)$ on $\mathcal{W}_\gamma$ for all $i \in [n+m]$.

2.4. Reverse-Mode Automatic Differentiation

Reverse-mode automatic differentiation is an algorithm for computing the gradient $Dv_{n+m}(w)$ of the target function (if it exists) by sequentially running one *forward pass* (Algorithm 1) and one *backward pass* (Algorithm 2). Given $w \in \mathbb{R}^n$, its forward pass computes $v_{n+1}(w), \dots, v_{n+m}(w)$ and corresponding pieces $\gamma_i \in \Gamma_i$ such that $w \in \mathcal{W}_\gamma$ for $\gamma = ((), \dots, (), \gamma_{n+1}, \dots, \gamma_{n+m})$. Namely, we have $v_i(w) = v_{i,\gamma}(w)$ for all $i \in [n+m]$.

Algorithm 1 Forward pass of reverse-mode automatic differentiation

1: **Input:** P, $(w_1, \dots, w_n)$
2: **Initialize:** $(v_1, \dots, v_n) = (w_1, \dots, w_n)$
3: **for** $i = n+1, \dots, n+m$ **do**
4: Let $p = ()$
5: **while** $p \notin \Gamma_i$ **do**
6: $p = p \oplus (\text{sign}(\phi_{i,p}(v_{pa(i)}) - c_{i,p}))$
7: **end while**
8: Set $\gamma_i = p$ and $v_i = f_{i,\gamma_i}(v_{pa(i)})$
9: **end for**
10: **return** $(v_1, \dots, v_{n+m})$, $(\gamma_{n+1}, \dots, \gamma_{n+m})$

Algorithm 2 Backward pass of reverse-mode automatic differentiation

1: **Input:** P, $(v_1, \dots, v_{n+m})$, $(\gamma_{n+1}, \dots, \gamma_{n+m})$
2: **Initialize:** $(gv_1, \dots, gv_{n+m}) = (0, \dots, 0, 1)$
3: **for** $i = n+m, \dots, n+1$ **do**
4: **for** $j \in pa(i)$ **do**
5: $gv_j = gv_j + gv_i \cdot \dfrac{\partial f_{i,\gamma_i}}{\partial v_j}(v_{pa(i)})$
6: **end for**
7: **end for**
8: **return** $(gv_1, \dots, gv_n)$

Given $v_1(w), \dots, v_{n+m}(w)$ and γ, the backward pass computes $Dv_{n+m,\gamma}(w)$ by applying the chain rule to the composition of differentiable functions $f_{n+1,\gamma_{n+1}}, \dots, f_{n+m,\gamma_{n+m}}$. In particular, it iteratively updates gv_i and returns $(gv_1, \dots, gv_n) = Dv_{n+m,\gamma}(w)$. It is well known that reverse-mode automatic differentiation computes the correct gradient, i.e., gv_i coincides with $\partial v_{n+m}(w)/\partial w_i$ for all $i \in [n]$, if primitive functions $F_{n+1}, \dots, F_{n+m}$ do not have any branches [8,9]. However, if some F_i uses branches, it may return arbitrary values

even if the target function $v_{n+m}(\cdot)$ is differentiable at w [14,32,33]. In the rest of the paper, we use AD to denote reverse-mode automatic differentiation.

Our algorithm computing an element of the Clarke subdifferential is similar to AD: it first finds some pieces $\gamma_i^* \in \Gamma_i$ and applies the backward pass of AD (Algorithm 2) to compute its output. Here, we chose the pieces γ_i^* that are used for computing the forward pass with some perturbed input, not the original one. Hence, our pieces and that of AD are different in general, which enables our algorithm to correctly compute an element of the Clarke subdifferential. We provide more details including the intuition behind our algorithm in Sections 3.2 and 3.3.

3. Efficient Automatic Subdifferentiation

In this section, we present our algorithm for efficiently computing an element of the Clarke subdifferential. To this end, we first introduce a class of primitive functions, which we consider in the rest of this paper. Then, we describe our algorithm after illustrating its underlying intuition via an example. Lastly, we analyze the computational complexity of our algorithm.

3.1. Assumptions on Primitive Functions

We considered primitive functions that satisfy the following assumptions.

Assumption 1. *For any $i \in [n+m] \setminus [n]$, $p \in \Gamma_i$, and $j \in [\mathrm{len}(p)]$, the following hold:*

- $\mathcal{X}_{i,p} \neq \emptyset$.
- $\phi_{i,p_{:j}}$ *is linear, i.e., there exists $z \in \mathbb{R}^{d_i}$ such that $\phi_{i,p_{:j}}(x) = \langle z, x \rangle$.*
- $f_{i,p}$ *is analytic on $\mathbb{R}^{d_i}$.*

The first assumption states that, for any $i \in [n+m] \setminus [n]$ and $p \in \Gamma_i$, there exists $x \in \mathbb{R}^{d_i}$ such that F_i selects $f_{i,p}$ at x. In other words, there is no non-reachable piece $p \in \Gamma_i$, i.e., all pieces of F_i are necessary to express F_i. The second assumption requires that all if–else statements of F_i have linear $\phi_{i,p}$ in their conditions. Lastly, we considered $f_{i,p}$, which is analytic on its domain (e.g., polynomials, exp, log, and sin), as stated in the third assumption. From this, $v_{i,\gamma}(\cdot)$ is well-defined and analytic on some open set containing $\mathrm{cl}(\mathcal{W}_\gamma)$ for all $i \in [n+m]$ and $\gamma \in \Gamma$.

Assumption 1 admits any primitive function that is analytic or piecewise analytic with linear boundaries such as max and bilinear interpolation. Hence, it allows many interesting programs such as nearly all neural networks considered in modern deep learning, e.g., [6,34].

3.2. Intuition Behind Efficient Automatic Subdifferentiation

As in AD, our algorithm first performs one forward pass (Algorithm 3) to compute the intermediate values $v_{n+1}(w), \ldots, v_{n+m}(w)$ and to find proper pieces $\gamma_{n+1}, \ldots, \gamma_{n+m}$ for the given input w. Then, it runs the original backward pass of AD (Algorithm 2) to compute an element of the Clarke subdifferential at w using the intermediate values and the pieces generated by the forward pass. Here, the key component of our algorithm is about *how to choose proper pieces* in the forward pass so that the backward pass can correctly compute an element of the Clarke subdifferential.

Before describing our algorithm, we explain its underlying intuition. Let $\delta = (\delta_1, \ldots, \delta_n) \in \mathbb{R}^n$ be a random vector drawn from a Gaussian distribution (see the initialization of Algorithm 3). Then, there exists unique $\gamma^* \in \Gamma$ and some $s^* > 0$ almost surely such that

$$w + t \cdot \delta \in \mathrm{int}(\mathcal{W}_{\gamma^*}) \text{ for all } t \in (0, s^*), \tag{1}$$

i.e., a given program takes the same piece γ^* for all inputs close to w along the direction of δ; see Lemma 7 in Section 4 for the details. Since $v_{n+m}(\cdot) = v_{n+m,\gamma^*}(\cdot)$ on $\mathcal{W}_{\gamma^*}$ and $v_{n+m,\gamma^*}(\cdot)$ is differentiable, Equation (1) implies that $v_{n+m}(\cdot)$ is differentiable at $w + t \cdot \delta$ for all $t \in (0, s^*)$. Therefore, the quantity:

$$Dv_{n+m,\gamma^*}(w) = \lim_{t\downarrow 0} Dv_{n+m,\gamma^*}(w + t \cdot \delta) = \lim_{t\downarrow 0} Dv_{n+m}(w + t \cdot \delta) \tag{2}$$

is an element of the Clarke subdifferential $\partial^c v_{n+m}(w)$, and our algorithm computes this very quantity via the backward pass of AD.

Algorithm 3 Forward pass of our algorithm

1: **Input:** P, $(w_1, \ldots, w_n)$
2: **Initialize:** Sample $\delta_i \sim \text{Normal}(0,1)$, and set $(v_i, dv_i) = (w_i, \delta_i)$ for all $i \in [n]$
3: **for** $i = n+1, \ldots, n+m$ **do**
4: Let $p = ()$
5: **while** $p \notin \Gamma_i$ **do**
6: **if** $(\phi_{i,p}(v_{pa(i)}) > c_{i,p}) \vee (\phi_{i,p}(v_{pa(i)}) = c_{i,p} \wedge \phi_{i,p}(dv_{pa(i)}) > 0)$ **then**
7: $p = p \oplus (1)$
8: **else if** $(\phi_{i,p}(v_{pa(i)}) < c_{i,p}) \vee (\phi_{i,p}(v_{pa(i)}) = c_{i,p} \wedge \phi_{i,p}(dv_{pa(i)}) < 0)$ **then**
9: $p = p \oplus (-1)$
10: **else if** $\phi_{i,p}(v_{pa(i)}) = c_{i,p} \wedge \phi_{i,p}(dv_{pa(i)}) = 0$ **then**
11: $p = p \oplus (s)$ for any $s \in \{-1, 1\}$
12: **end if**
13: **end while**
14: Set $\gamma_i = p$, $v_i = f_{i,p}(v_{pa(i)})$, and $dv_i = \langle \nabla f_{i,p}(v_{pa(i)}), dv_{pa(i)} \rangle$
15: **end for**
16: **return** $(v_1, \ldots, v_{n+m})$, $(\gamma_{n+1}, \ldots, \gamma_{n+m})$

We now illustrate the main idea behind our forward pass, which enables the backward pass to compute $Dv_{n+m,\gamma^*}(w)$ in Equation (2). As an example, consider a program with the following primitive functions: $F_{n+1}, \ldots, F_{n+m-1}$ are all analytic, and F_{n+m} branches only once with $\phi_{n+m,()}(\cdot) = \phi(\cdot)$ and $c_{n+m,()} = 0$. For notational simplicity, we use $u(w) = v_{pa(n+m)}(w)$.

If $\phi(u(w)) > 0$, then it is easy to observe that $\gamma^*_{n+m} = 1$ from the continuity of ϕ and u. Likewise, if $\phi(u(w)) < 0$, then $\gamma^*_{n+m} = -1$. In the case that $\phi(u(w)) = 0$, we use the following directional derivatives to determine γ^*_{n+m}:

$$dv_i(w;\delta) \triangleq \lim_{t\downarrow 0} \frac{v_i(w + t \cdot \delta) - v_i(w)}{t} \tag{3}$$

for $i \in [n+m-1]$, which can be easily computed using the chain rule. From the definition of dv_i, the linearity of ϕ, and the chain rule, it holds that

$$\lim_{t\downarrow 0} \frac{\phi(u(w + t \cdot \delta)) - \phi(u(w))}{t} = \sum_{j \in pa(n+m)} \frac{\partial \phi(u(w))}{\partial v_j(w)} \cdot dv_j(w;\delta) = \phi(du(w;\delta)),$$

where $du(w;\delta)$ denotes the vector of all $dv_j(w;\delta)$ with $j \in pa(n+m)$. Then, by Taylor's theorem, $\phi(du(w;\delta)) > 0$ (or $\phi(du(w;\delta)) < 0$) implies $\gamma^*_{n+m} = 1$ (or $\gamma^*_{n+m} = -1$). In summary, if $\phi(u(w)) \neq 0$ or $\phi(du(w;\delta)) \neq 0$, then the exact γ^*_{n+m} can be found, and hence, the backward pass (Algorithm 2) can correctly compute $Dv_{n+m,\gamma^*}(w)$ using $\gamma^* = ((), \ldots, (), \gamma^*_{n+m})$.

Now, we considered the only remaining case: $\phi(u(w)) = 0$ and $\phi(du(w;\delta)) = 0$. Unlike the previous cases, it is non-trivial here to find the correct γ^*_{n+m} because the first-order Taylor series approximation does not provide any information about whether a small perturbation of w toward δ increases $\phi(u(w))$ or not. An important point, however, is that we do not need the exact γ^*_{n+m} to compute an element of the Clarke subdifferential; instead, it suffices to compute $Dv_{n+m,\gamma^*}(w)$. Surprisingly, this can be performed by choosing an *arbitrary piece* of F_{n+m}, as shown below.

For simplicity, suppose that $\phi(x) = x_1$, i.e., $\phi(u(w)) = v_{i^*}(w)$ for some $i^* \in pa(n+m)$; the below argument can be easily extended to an arbitrary linear ϕ. Let $\gamma^\alpha = ((), \dots, (), \alpha)$ for $\alpha \in \{(-1), (1)\}$, i.e., $\gamma^\alpha_{n+m} = \alpha$. Then, for any $\alpha \in \{(-1), (1)\}$, we have

$$Dv_{n+m,\gamma^\alpha}(w) = \sum_{j \in pa(n+m) \setminus \{i^*\}} \frac{\partial f_{n+m,\alpha}(u(w))}{\partial v_j(w)} \cdot Dv_j(w) \tag{4}$$

almost surely, by the chain rule and the following result: $dv_{i^*}(w;\delta) = \phi(du(w;\delta)) = 0$ implies $Dv_{i^*}(w) = 0$ almost surely (Lemma 5 in Section 4). Here, from the continuity and the definition of F_{n+m}, we must have $f_{n+m,(1)} = f_{n+m,(-1)}$ on the hyperplane $\{x \in \mathbb{R}^{d_{n+m}} : x_1 = 0\}$, and thus, $\partial f_{n+m,(1)}(x)/\partial x_j = \partial f_{n+m,(-1)}(x)/\partial x_j$ for any $x \in \{x \in \mathbb{R}^{d_{n+m}} : x_1 = 0\}$ and $j \in [d_{n+m}] \setminus \{1\}$. From this and $v_{i^*}(w) = \phi(u(w)) = 0$, we then obtain

$$\frac{\partial f_{n+m,(1)}(u(w))}{\partial v_j(w)} = \frac{\partial f_{n+m,(-1)}(u(w))}{\partial v_j(w)} \tag{5}$$

for all $j \in pa(n+m) \setminus \{i^*\}$. By combining Equations (4) and (5), we can finally conclude that

$$Dv_{n+m,\gamma^{(1)}}(w) = Dv_{n+m,\gamma^{(-1)}}(w) = Dv_{n+m,\gamma^*}(w)$$

almost surely, where the last equality is from the fact that γ^*_{n+m} is either (1) or (-1). To summarize, if $\phi(u(w)) = 0$ and $\phi(du(w;\delta)) = 0$, we can compute the target element of the Clarke subdifferential (i.e., $Dv_{n+m,\gamma^*}(w)$) by choosing an arbitrary piece of F_{n+m}.

3.3. Forward Pass for Efficient Automatic Subdifferentiation

Our algorithm for computing an element of the Clarke subdifferential is based on the observation made in the previous section: it runs one forward pass (Algorithm 3) for computing $v_{n+1}(w), \dots, v_{n+m}(w)$ and some $\gamma \in \Gamma$ such that $Dv_{n+m,\gamma}(w) = Dv_{n+m,\gamma^*}(w)$ and one backward pass of AD (Algorithm 2) for computing $Dv_{n+m,\gamma}(w)$.

We now describe our forward pass procedure (Algorithm 3). First, it randomly samples a vector $\delta \in \mathbb{R}^n$ from a Gaussian distribution and initializes $dv_i = \delta_i$ for all $i \in [n]$ (line 2). Then, it iterates for $i = n+1, \dots, n+m$ as follows. Given $v_1(w), \dots, v_{i-1}(w)$ and their directional derivatives $dv_1(w;\delta), \dots, dv_{i-1}(w;\delta)$ with respect to δ, lines 5–13 in Algorithm 3 find a proper piece $\gamma_i \in \Gamma_i$ of F_i by exploring its branches. If the condition in line 6 is satisfied, then it moves to the branch corresponding to $\phi_{i,p}(v_{pa(i)}(w)) > c_{i,p}$ (line 7). It moves in a similar way if the condition in line 8 is satisfied. As in our example in Section 3.2, if $\phi_{i,p}(v_{pa(i)}(w)) = c_{i,p}$ and $\phi_{i,p}(dv_{pa(i)}(w;\delta)) = 0$ (line 10), then our algorithm moves to an arbitrary branch (line 11). Once Algorithm 3 finds a proper piece γ_i of F_i, it updates $v_i(w)$ and $dv_i(w;\delta)$ via the chain rule (line 14). Here, $v_i(w)$ can be correctly computed due to the continuity of F_i, while $dv_i(w;\delta)$ can also be correctly computed almost surely; see Lemma 8 in Section 4 for details. We remark that our algorithm is a generalization of the algorithm in [14]. The difference occurs in lines 10–11, where the existing algorithm deterministically chooses s based on some qualification condition [14].

As illustrated in Section 3.2, the piece $\gamma \in \Gamma$ computed by our forward pass satisfies $Dv_{n+m,\gamma}(w) = Dv_{n+m,\gamma^*}(w)$ almost surely, and hence, the backward pass using this γ correctly computes $Dv_{n+m,\gamma^*}(w)$ almost surely, which is an element of the Clarke subdifferential. We formally state the correctness of our algorithm in the following theorem; its proof is given in Section 4.

Theorem 1. *Suppose that Assumption 1 holds. Then, for any $w \in \mathbb{R}^n$, running Algorithm 3 and then Algorithm 2 returns an element of $\partial^c v_{n+m}(w)$ almost surely.*

3.4. Computational Cost

In this section, we analyze the computational cost of our algorithm (both forward and backward passes) on a program P, compared to the cost of running P. Here, we

only counted the cost of arithmetic operations and function evaluations and ignore the cost of memory read and write. We assumed that elementary operations ($+, \times, \wedge, \vee$), the comparison between two scalar values ($>$,$<$, $=$), and sampling a value from the standard normal distribution have a unit cost (e.g., $\text{cost}(+) = 1$), while the cost for evaluating an analytic function f is represented by $\text{cost}(f)$. To denote the cost of evaluating a program P with an input w, we use $\text{cost}(P(w))$. Likewise, for the cost of running our algorithm (i.e., Algorithms 2 and 3) on P and w, we use $\text{cost}(\text{ours}(P, w))$. We also assumed that memory read/write costs are included in our cost function. Under this setup, we bound the computational cost of our algorithm in Theorem 2.

Theorem 2. *Suppose that* $\text{cost}(P(w)) \geq n$ *for all* $w \in \mathbb{R}^n$*. Then, for any program P and its input* $w \in \mathbb{R}^n$, $\text{cost}(\text{ours}(P, w)) \leq \kappa \cdot \text{cost}(P(w))$ *where*

$$\kappa \triangleq 1 + \max_{i \in [n+m] \setminus [n]} \kappa_i, \tag{6}$$

$$\kappa_i \triangleq \frac{\max_{p \in \Gamma_i} 2\text{cost}(\nabla f_{i,p}) + \text{cost}(f_{i,p}) + 4d_i + 4\,\text{len}(p) + 2\sum_{j=1}^{\text{len}(p)} \text{cost}(\phi_{i,p_{:j}})}{\min_{q \in \Gamma_i} \text{cost}(f_{i,q}) + \text{len}(q) + \sum_{j=1}^{\text{len}(q)} \text{cost}(\phi_{i,q_{:j}})}.$$

The assumption in Theorem 2 is mild since it is satisfied if at least one distinct operation is applied to each input for evaluating P. The proof of Theorem 2 is presented in Section 5, where we use program representations of Algorithms 2 and 3 (see Figures 4 and 5 in Section 3.4 for the details).

Suppose that, for each $i \in [n+m] \setminus [n]$, d_i and $\max_{p \in \Gamma_i} \text{len}(p)$ (i.e., the arity and the maximum branch depth of f_i) are independent of n. This condition holds in many practical cases: e.g., the absolute value function has $d_i = 1$ and $\max_{p \in \Gamma_i} \text{len}(p) = 1$; $\max\{\cdot, \cdot\}$ has $d_i = 2$ and $\max_{p \in \Gamma_i} \text{len}(p) = 1$. Under this mild condition, $\text{cost}(f_{i,p})$, $\text{cost}(\nabla f_{i,p})$, and $\text{cost}(\phi_{i,p_{:j}})$ are independent of n, and thus, κ_i does so because the numerator in the definition of κ_i is independent of n and the denominator is at least one (as $\text{cost}(f_{i,q}) \geq 1$). This implies that κ is independent of the input dimension n under the above condition.

In practical setups with large n, the computational cost of our algorithm can be much smaller than that of existing algorithms based on the lexicographic subdifferential [17–20]. For example, modern neural networks have more than a million parameters (i.e., n), where the cost for computing the gradient of each piece in the activation functions (i.e., $\text{cost}(\nabla f_{i,p})$) is typically bounded by $O(\text{cost}(f_{i,p}))$. Further, the depth of branches in these activation functions is often bounded by a constant (e.g., the depth is one for ReLU). Hence, for those networks, $\kappa = O(1)$, and our algorithm does not incur much computational overhead. On the other hand, lexicographic-subdifferential-based approaches require at least n computations of $P(w)$ [17–20], which may not be practical when n is large.

$$
\begin{array}{l}
\texttt{prog}\ P^{\texttt{ADB}}(v_1, \ldots, v_{n+1}, \gamma_{n+1}, \ldots, \gamma_{n+m})\{ \\
\quad (gv_1, \ldots, gv_{n+m-1}, gv_{n+m}) = (0, \ldots, 0, 1); \\
\quad gv_{pa(n+m)} \mathrel{+}= \nabla f_{n+m, \gamma_{n+m}}(v_{pa(n+m)}) \times gv_{n+m}; \\
\qquad \vdots \\
\quad gv_{pa(n+1)} \mathrel{+}= \nabla f_{n+1, \gamma_{n+1}}(v_{pa(n+1)}) \times gv_{n+1}; \\
\quad \texttt{return}\ (gv_1, \ldots, gv_n) \\
\}
\end{array}
$$

Figure 4. A program $P^{\texttt{ADB}}$ implementing the backward pass of AD (Algorithm 2).

```
prog P^ours(w_1, ..., w_{n+1}, γ_{n+1}, ..., γ_{n+m}){
  (v_1, ..., v_n) = (w_1, ..., w_n);
  (v_{n+1}, dv_{n+1}, γ_{n+1}) = F^ours_{n+1}(v_{pa(n+1)}, dv_{pa(n+1)});
    ⋮
  (v_{n+m}, dv_{n+m}, γ_{n+m}) = F^ours_{n+m}(v_{pa(n+m)}, dv_{pa(n+m)});
  return (v_{n+1}, ..., v_{n+m}), (γ_{n+1}, ..., γ_{n+m})
}

func F_i^ours(x_1, ..., x_{d_i}, dx_1, ..., dx_{d_i}){
  E^ours_{i,()}
}
```

$$E^{\mathtt{ours}}_{i,p} \triangleq \begin{cases} \mathtt{return}\ f_{i,p}(x_{1:d_i}), \langle \nabla f_{i,p}(x_{1:d_i}), dx_{1:d_i}\rangle, p & \text{if } E_{i,p} = \mathtt{return}\ f_{i,p}(x_{1:d_i}), \\ \left(\begin{array}{l} y = \phi_{i,p}(x_1, \ldots, x_{d_i}); \\ dy = \phi_{i,p}(dx_1, \ldots, dx_{d_i}); \\ \mathtt{if}\ \ (y > c_{i,p})\{\ E^{\mathtt{ours}}_{i,p\oplus(1)}\ \} \\ \mathtt{elif}\ (y < c_{i,p})\{\ E^{\mathtt{ours}}_{i,p\oplus(-1)}\ \} \\ \mathtt{elif}\ (dy > 0)\{\ E^{\mathtt{ours}}_{i,p\oplus(1)}\ \} \\ \mathtt{elif}\ (dy < 0)\{\ E^{\mathtt{ours}}_{i,p\oplus(-1)}\ \} \\ \mathtt{else}\{\ E^{\mathtt{ours}}_{i,p\oplus(-1)}\ //\ \text{or}\ E^{\mathtt{ours}}_{i,p\oplus(-1)}\ \} \end{array} \right) & \text{otherwise.} \end{cases}$$

Figure 5. A program $P^{\mathtt{ours}}$ implementing Algorithm 3. Here, $x_{1:d_i} = (x_1, \ldots, x_{d_i})$.

4. Proof of Theorem 1

In this section, we prove Theorem 3 under the setup that δ in Algorithm 3 is given instead of randomly sampled. This theorem directly implies Theorem 1 since the statement of Theorem 3 holds for almost every δ and the proof of Theorem 1 requires showing the same statement almost surely, where the randomness comes from δ following an Isotropic Gaussian distribution. Namely, proving Theorem 3 suffices for proving Theorem 1. We note that all results in this section are under Assumption 1.

Theorem 3. *Given $w \in \mathbb{R}^n$, Algorithms 3 and 2 compute an element of $\partial^c v_{n+m}(w)$ for almost every $\delta \in \mathbb{R}^n$.*

4.1. Additional Notations

We frequently use the following shorthand notations: the set of indices of branches:

$$\mathcal{I}_{br} \triangleq \{i \in [n+m] : |\Gamma_i| > 1\}$$

and an auxiliary index set:

$$\mathsf{Idx}_i \triangleq \{(j, p) : p \in \Gamma_i, j \in [\mathrm{len}(p)]\}.$$

For $\gamma \in \Gamma$, $i \in [n+m] \setminus [n]$, and $(j, p) \in \mathsf{Idx}_i$, we use

$$\phi^{\gamma}_{i,p_{:j}}(w) \triangleq \phi_{i,p_{:j}}(v_{pa(i),\gamma}(w)).$$

Note that $v_{i,\gamma}$ and $\phi^{\gamma}_{i,p_{:j}}$ are analytic (and, therefore, differentiable) for all $\gamma \in \Gamma$, $i \in [n+m] \setminus [n]$, and $(j, p) \in \mathsf{Idx}_i$. We next define the set of pieces reachable by our algorithm (Algorithm 3) with inputs $w = (w_1, \ldots, w_n), \delta = (\delta_1, \ldots, \delta_n) \in \mathbb{R}^n$ as $\Gamma(w, \delta)$:

$$\Gamma(w, \delta) \triangleq \{\gamma \in \Gamma : \gamma_{i,j} \in \mathcal{C}^{\gamma}_{i,j}(w; \delta)\ \forall i \in \mathcal{I}_{br}, \forall j \in [\mathrm{len}(\gamma_i)]\}, \quad \text{where}$$

$$\mathcal{C}_{i,j}^{\gamma}(w;\delta) \triangleq \begin{cases} \{1\} & \text{if } \left(\phi_{i,\gamma_{i,:j}}(v_{pa(i),\gamma}(w)) = c_{i,\gamma_{i,:j}} \wedge \phi_{i,\gamma_{i,:j}}(dv_{pa(i),\gamma}(w;\delta)) > 0\right) \\ & \qquad \vee \left(\phi_{i,\gamma_{i,:j}}(v_{pa(i)}^{\gamma}) > c_{i,\gamma_{i,:j}}\right) \\ \{-1\} & \text{if } \left(\phi_{i,\gamma_{i,:j}}(v_{pa(i),\gamma}(w)) = c_{i,\gamma_{i,:j}} \wedge \phi_{i,\gamma_{i,:j}}(dv_{pa(i),\gamma}(w;\delta)) < 0\right) \\ & \qquad \vee \left(\phi_{i,\gamma_{i,:j}}(v_{pa(i)}^{\gamma}) < c_{i,\gamma_{i,:j}}\right) \\ \{-1,1\} & \text{if } \left(\phi_{i,\gamma_{i,:j}}(v_{pa(i),\gamma}(w)) = c_{i,\gamma_{i,:j}} \wedge \phi_{i,\gamma_{i,:j}}(dv_{pa(i),\gamma}(w;\delta)) = 0\right) \end{cases}.$$

4.2. Technical Claims

Lemma 1. *For any open $\mathcal{O} \subset \mathbb{R}$, for any analytic, but non-constant $f : \mathcal{O} \to \mathbb{R}$, and for any $x \in \mathcal{O}$, there exists $\varepsilon > 0$ such that*

$$f(x) \notin f([x-\varepsilon, x+\varepsilon] \setminus \{x\}).$$

Furthermore, f is strictly monotone on $[x, x+\varepsilon]$ and strictly monotone on $[x-\varepsilon, x]$.

Proof of Lemma 1. Without loss of generality, suppose that $f(x) = 0$. Since f is analytic, f is infinitely differentiable and can be represented by the Taylor series on $(x-\delta, x+\delta)$ for some $\delta > 0$:

$$f(z) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x)}{i!}(z-x)^i$$

where $f^{(i)}$ denotes the i-th derivative of f. Since f is non-constant, there exists $i \in \mathbb{N}$ such that $f^{(i)}(x) \neq 0$. Let i^* be the minimum such i. Then, by Taylor's theorem, $f(z) = \frac{f^{(i^*)}(x)}{i^*!}(z-x)^{i^*} + o(|z-x|^{i^*})$.

Consider the case that $f^{(i)}(x) > 0$ and i^* is odd. Then, we can choose $\varepsilon \in (0, \delta)$ so that

$$f^{(1)}(z) < 0 \ \text{ on } \ [x-\varepsilon, x) \quad \text{and} \quad f^{(1)}(z) > 0 \ \text{ on } \ (x, x+\varepsilon]$$

i.e., f is strictly increasing on $[x-\varepsilon, x+\varepsilon]$ (e.g., by the mean value theorem), and hence, $f(x) \notin f([x-\varepsilon, x+\varepsilon] \setminus \{x\})$. One can apply a similar argument to the cases that $f^{(i)}(x) < 0$ and i^* is odd, $f^{(i)}(x) > 0$ and i^* is even, and $f^{(i)}(x) < 0$ and i^* is even. This completes the proof of Lemma 1. □

Lemma 2 (Proposition 0 in [35])**.** *For any $n \in \mathbb{N}$, for any open connected $\mathcal{O} \subset \mathbb{R}^n$, and for any real analytic $f : \mathcal{O} \to \mathbb{R}$, if $\mu_n(zero(f)) > 0$, then $f(x) = 0$ for all $x \in \mathcal{O}$.*

Lemma 3. *For any $n \in \mathbb{N}$, for any open connected $\mathcal{O} \subset \mathbb{R}^n$, and for any real analytic $f, g : \mathcal{O} \to \mathbb{R}$, if $\mu_n(zero(f-g)) > 0$, then $f(x) = g(x)$ for all $x \in \mathcal{O}$.*

Proof of Lemma 3. The proof directly follows from Lemma 2. □

4.3. Technical Assumptions

Assumption 2. *Given $w \in \mathbb{R}^n$, $\delta \in \mathbb{R}^n$ satisfies the following: for any $\gamma \in \Gamma$, $i \in \mathcal{I}_{br}$, and $(j,p) \in \mathsf{Idx}_i$, if $\phi_{i,p_{:j}}^{\gamma}$ is not a constant function, then*

$$\phi_{i,p_{:j}}^{\gamma}(w + t \cdot \delta) \text{ is not a constant function in } t \in \mathbb{R}.$$

Assumption 3. *Given $w \in \mathbb{R}^n$, $\delta \in \mathbb{R}^n$ satisfies the following: for any $\gamma \in \Gamma$, $i \in \mathcal{I}_{br}$, and $(j,p) = \mathsf{Idx}_i$,*

$$\langle \delta, D\phi_{i,p_{:j}}^{\gamma}(w) \rangle = 0 \quad \text{if and only if} \quad D\phi_{i,p_{:j}}^{\gamma}(w) = \vec{0}.$$

4.4. Technical Lemmas

Lemma 4. *Given $w \in \mathbb{R}^n$, almost every $\delta \in \mathbb{R}^n$ satisfies Assumption 2.*

Proof of Lemma 4. Since $|\Gamma| < \infty$, if the set of δ that does not satisfy Assumption 2 has a non-zero measure, then there exist $\gamma \in \Gamma$, $i \in \mathcal{I}_{br}$, and $(j, p) \in \mathsf{Idx}_i$ such that $\phi^{\gamma}_{i,p_{:j}}$ is not a constant function and

$$\mu_n\Big(\mathcal{S} \triangleq \{\delta \in \mathbb{R}^n : \phi^{\gamma}_{i,p_{:j}}(w + t \cdot \delta) \text{ is a constant function in } t \in \mathbb{R}\}\Big) > 0.$$

Without loss of generality, suppose that $\phi^{\gamma}_{i,p_{:j}}(w) = 0$. Then, from the definition of $\mathcal{S}$, $\mathcal{S}$ is contained in the zero set:

$$zero(\tilde{\phi}) \triangleq \{u \in \mathbb{R} : \tilde{\phi}(u) = 0\}$$

of an analytic function $\tilde{\phi} : \mathbb{R}^W \to \mathbb{R}$ defined as

$$\tilde{\phi}(u) \triangleq \phi^{\gamma}_{i,p_{:j}}(w + u).$$

Namely, $\mu_n(zero(\tilde{\phi})) \geq \mu_n(\mathcal{S}) > 0$. However, from Lemma 2, $\tilde{\phi}$ must be a constant function, which contradicts our assumption that $\phi^{\gamma}_{i,p_{:j}}$ is not a constant function. This completes the proof of Lemma 4. □

Lemma 5. *Given $w \in \mathbb{R}^n$, almost every $\delta \in \mathbb{R}^n$ satisfies Assumption 3.*

Proof of Lemma 5. Since $D\phi^{\gamma}_{i,p_{:j}}(w) = \vec{0}$ implies $\langle \delta, \phi^{\gamma}_{i,p_{:j}}(w)\rangle = 0$ for all $\delta \in \mathbb{R}^n$, we prove the converse. Suppose that $D\phi^{\gamma}_{i,p_{:j}}(w) \neq \vec{0}$. Since the set $\{\delta \in \mathbb{R}^n : \langle \delta, D\phi^{\gamma}_{i,p_{:j}}(w)\rangle = 0\}$ has zero measure under $D\phi^{\gamma}_{i,p_{:j}}(w) \neq \vec{0}$,

$$\bigcup_{\gamma \in \Gamma, i \in \mathcal{I}_{br}, (j,p) \in \mathsf{Idx}_i : D\phi^{\gamma}_{i,p_{:j}}(w) \neq \vec{0}} \{\delta \in \mathbb{R}^n : \langle \delta, D\phi^{\gamma}_{i,p_{:j}}(w)\rangle = 0\}$$

also has zero measure. This completes the proof of Lemma 5. □

Lemma 6. *For $i \in \mathcal{I}_{br}$ and $p \in \Gamma_i$, suppose that $x \in \mathbb{R}^{d_i}$ satisfies one of the following for all $j \in [\mathrm{len}(p)]$:*

- $\mathrm{sign}(\phi_{i,p_{:j}}(x) - c_{i,p_{:j}}) = p_j$;
- $\phi_{i,p_{:j}}(x) = c_{i,p_{:j}}$.

Then, $x \in \mathrm{cl}(\mathcal{X}_{i,p})$.

Proof of Lemma 6. Without loss of generality, assume that $x \notin \mathcal{X}_{i,p}$. Since we assumed $\mathcal{X}_{i,p} \neq \emptyset$ by Assumption 1, there exists $y \in \mathcal{X}_{i,p}$, i.e., $\mathrm{sign}(\phi_{i,p_{:j}}(y) - c_{i,p_{:j}}) = p_j$ for all $j \in [\mathrm{len}(p)]$. Define

$$\mathcal{I} \triangleq \{j \in [\mathrm{len}(p)] : \phi_{i,p_{:j}}(x) = c_{i,p_{:j}}\}.$$

Since $\phi_{i,p_{:j}}$ is linear, for $z = y - x$ and for any $j \in \mathcal{I}$, we have

$$\mathrm{sign}(\phi_{i,p_{:j}}(z)) = p_j.$$

This implies that, for any $t > 0$ and $j \in \mathcal{I}$, it holds that

$$\mathrm{sign}(\phi_{i,p_{:j}}(x + t \cdot z)) = p_j. \quad (7)$$

Since $|\phi_{i,p_{:j}}(x) - c_{i,p_{:j}}| > 0$ and $\mathrm{sign}(\phi_{i,p_{:j}}(x)) = p_j$ for all $j \in [\mathrm{len}(p)] \setminus \mathcal{I}$ by the definition of $\mathcal{I}$, there exists $s > 0$ such that

$$|\phi_{i,p_{:j}}(x + t \cdot z) - c_{i,p_{:j}}| > 0 \quad \text{and} \quad \mathrm{sign}(\phi_{i,p_{:j}}(x + t \cdot z)) = p_j \tag{8}$$

for all $t \in [0, s]$ and $j \in [\mathrm{len}(p)] \setminus \mathcal{I}$. Combining Equations (7) and (8) implies that $x + t \cdot z \in \mathcal{X}_{i,p}$ for all $t \in (0, s]$, i.e., x is a limit point of $\mathcal{X}_{i,p}$. This completes the proof of Lemma 6 □

4.5. Key Lemmas

Lemma 7. *For any $w \in \mathbb{R}^n$ and for any $\delta \in \mathbb{R}^n$ satisfying Assumption 2, there exist $s^* > 0$ and $\gamma^* \in \Gamma(w, \delta)$ such that*

$$w + t \cdot \delta \in \mathrm{int}(\mathcal{W}_{\gamma^*}) \quad \textit{for all} \quad t \in (0, s^*).$$

Proof of Lemma 7. We first define some notations: for an analytic function $f : \mathbb{R} \to \mathbb{R}$ and $t \in \mathbb{R}$,

$$\phi^{\gamma,w,\delta}_{i,p_{:j}}(t) \triangleq \phi^{\gamma}_{i,p_{:j}}(w + t \cdot \delta),$$

$$\mathrm{dir}(f) \triangleq \begin{cases} s' & \text{if } s' = \sup_{s \geq 0} \{f(0) \text{ is strictly increasing on } [0, s]\} > 0 \\ s'' & \text{if } s'' = \sup_{s \geq 0} \{f(0) \text{ is strictly decreasing on } [0, s]\} > 0 \\ \perp & \text{otherwise} \end{cases}.$$

Under Assumption 2 and by Lemma 1, one can observe that, for any $i \in \mathcal{I}_{br}$, $p = \gamma_i$, and $j \in [\mathrm{len}(p)]$, $\mathrm{dir}(\phi^{\gamma,w,\delta}_{i,p_{:j}}) = \perp$ if and only if $\phi^{\gamma}_{i,p_{:j}}$ is a constant function. In addition, from Lemma 1, if $\phi^{\gamma}_{i,p_{:j}}$ is not a constant function, then $\mathrm{dir}(\phi^{\gamma,w,\delta}_{i,p_{:j}}) > 0$. Using Algorithm 4, we iteratively construct $\gamma^* \in \Gamma(w, \delta)$ and update $s^* > 0$ for each $i \in \mathcal{I}_{br}$ so that

$$w + t \cdot \delta \in \mathrm{int}(\mathcal{W}_{\gamma^*}) \quad \text{for all} \quad t \in (0, s^*).$$

Under our construction of γ^*, one can observe that $\gamma^* \in \Gamma(w, \delta)$. From our choice of s^*, for any $i \in \mathcal{I}_{br}$, $j \in [\mathrm{len}(p)]$, and for $s_{i,j} = \mathrm{dir}(\phi^{\gamma^*,w,\delta}_{i,\gamma^*_{i,:j}})$, the following statements hold:

- If $s_{i,j} \neq \perp$, then $\phi^{\gamma^*,w,\delta}_{i,\gamma^*_{i,:j}}((0, s^*))$ is open since $\phi^{\gamma^*,w,\delta}_{i,\gamma^*_{i,:j}}$ is strictly monotone on $(0, s^*)$;
- If $s_{i,j} = \perp$, then $\phi^{\gamma^*,w,\delta}_{i,\gamma^*_{i,:j}}, \phi^{\gamma^*}_{i,\gamma^*_{i,:j}}$ are constant functions (i.e., $\phi^{\gamma^*,w,\delta}_{i,\gamma^*_{i,:j}}((0, s^*))$ is a constant) due to Assumption 2.

For any $t \in (0, s^*)$, we have $w + t \cdot \delta \in \mathcal{O} \subset \mathcal{W}_{\gamma^*}$, where

$$\mathcal{O} \triangleq \left(\bigcap_{i,j: z_{i,j} = \perp} (\phi^{\gamma^*}_{i,\gamma^*_{i,:j}})^{-1} \left(\phi^{\gamma^*,w,\delta}_{i,\gamma^*_{i,:j}}((0, s^*)) \right) \right) \cap \left(\bigcap_{i,j: z_{i,j} \neq \perp} (\phi^{\gamma^*}_{i,\gamma^*_{i,:j}})^{-1} \left(\phi^{\gamma^*,w,\delta}_{i,\gamma^*_{i,:j}}((0, s^*)) \right) \right).$$

Here, $\mathcal{O}$ is open since each term for the intersection in the above equation is open; it is $\mathbb{R}^n$ if $s_{i,j} = \perp$, and it is an inverse image of a continuous function of an open set otherwise. This completes the proof of Lemma 7. □

Algorithm 4 Construction of γ^* and s^*

```
Input: P, (w_1, ..., w_n), (δ_1, ..., δ_n)
Initialize: (v_1, ..., v_n) = (w_1, ..., w_n), dv_i = δ_i for all i ∈ [n], s* = ∞, γ* = ((), ..., ())
for i = n+1, ..., n+m do
  Let x = v_{pa(i)}, dx = dv_{pa(i)}, and p = ()
  while p ∉ Γ_i do
    Set y = φ_{i,p}(x) and s = dir(φ_{i,p}^{γ*,w,δ})
    if s = ⊥ ∧ y > c_{i,p} then
      p = p ⊕ 1
    else if s = ⊥ ∧ y ≤ c_{i,p} then
      p = p ⊕ (−1)
    else if s ≠ ⊥ ∧ y > c_{i,p} then
      p = p ⊕ 1 and ε = min{|φ_{i,p}^{γ*,w,δ}(s) − y|, y − c_{i,p}}
      {s'} = (φ_{i,p}^{γ*,w,δ})^{−1}(y + z·ε) ∩ [0, s] and s* = min{s*, s'}
    else if s ≠ ⊥ ∧ y < c_{i,p} then
      p = p ⊕ (−1) and ε = min{|φ_{i,p}^{γ*,w,δ}(s) − y|, c_{i,p} − y}
      {s'} = (φ_{i,p}^{γ*,w,δ})^{−1}(y + z·ε) ∩ [0, s] and s* = min{s*, s'}
    else if s ≠ ⊥ ∧ y = c_{i,p} then
      p = p ⊕ sign(z) and s* = min{s*, s}
    end if
    Set γ_i* = p and v_i = f_{i,γ_i*}(x)
  end while
end for
return γ*, s*
```

Corollary 1. *For any $w \in \mathbb{R}^n$ and for any $\delta \in \mathbb{R}^n$ satisfying Assumption 2, there exist $s^* > 0$ and $\gamma^* \in \Gamma(w, \delta)$ such that*

$$Dv_{n+m}(w + t \cdot \delta) = Dv_{n+m,\gamma^*}(w + t \cdot \delta) \quad \textit{for all} \quad t \in (0, s^*).$$

Proof of Corollary 1. This corollary directly follows from Lemma 7. □

Lemma 8. *For any $w \in \mathbb{R}^n$ and for any δ satisfying Assumption 3, it holds that*

$$Dv_{n+m,\gamma'}(w) = Dv_{n+m,\gamma''}(w) \quad \textit{for all} \quad \gamma', \gamma'' \in \Gamma(w, \delta).$$

Proof of Lemma 8. We use the mathematical induction on i to show that $v_{i,\gamma'}(w) = v_{i,\gamma''}(w)$ and $Dv_{i,\gamma'}(w) = Dv_{i,\gamma''}(w)$ for all $i \in [n+m]$. The base case is trivial: $v_{i,\gamma'}(w) = v_{i,\gamma''}(w)$ and $Dv_{i,\gamma'}(w) = Dv_{i,\gamma''}(w)$ for all $i \in [n]$. Hence, suppose that $i \in \mathcal{I}_{br}$ since the case that $i \in [n+m] \setminus ([n] \cup \mathcal{I}_{br})$ is also trivial. Then, by the induction hypothesis, we have $v_{j,\gamma'}(w) = v_{j,\gamma''}(w)$ and $Dv_{j,\gamma'}(w) = Dv_{j,\gamma''}(w)$ for all $j \in [i-1]$. For notational simplicity, we denote $x_j \triangleq v_{j,\gamma'}(w) = v_{j,\gamma''}(w)$ and $dx_j \triangleq dv_{j,\gamma'}(w;\delta) = dv_{j,\gamma''}(w;\delta)$ for all $j \in [i-1]$.

Let $p' = \gamma_i'$ and $p'' = \gamma_i''$. First, by Lemma 6, the definition of $\Gamma(w, \delta)$, and the induction hypothesis, we have

$$x_{pa(i)} \in \mathrm{cl}(\mathcal{X}_{i,p'}) \cap \mathrm{cl}(\mathcal{X}_{i,p''}).$$

Due to the continuity of F_i, this implies that

$$v_{i,\gamma'}(w) = f_{i,p'}(x_{pa(i)}) = f_{i,p''}(x_{pa(i)}) = v_{i,\gamma''}(w).$$

Now, it remains to show $Dv_{i,\gamma'}(w) = Dv_{i,\gamma''}(w)$. To this end, we define the following:

$$\mathcal{I} \triangleq \Big\{(j,p) \in \mathsf{Idx}_i : \phi_{i,p_{:j}}(dx_{pa(i)}) = 0\Big\},$$
$$\mathcal{S} \triangleq \{x \in \mathbb{R}^{d_i} : \phi_{i,p_{:j}}(x) = 0 \ \text{ for all } \ (j,p) \in \mathcal{I}\}.$$

From the definition of $\mathcal{S}$ and $\mathcal{I}$, $dx_{pa(i)} \in \mathcal{S}$. Furthermore, by Assumption 3, for any $(j,p) \in \mathsf{Idx}_i$ and $\gamma \in \{\gamma', \gamma''\}$, we have $\phi_{i,p_{:j}}(dv_{pa(i),\gamma}(w;\delta)) = 0$ if and only if $D\phi_{i,p_{:j}}(v_{pa(i),\gamma}(w)) = \vec{0}$, i.e., $\partial\phi_{i,p_{:j}}(v_{pa(i),\gamma}(w))/\partial w_\ell = \phi_{i,p_{:j}}(\partial v_{pa(i),\gamma}(w)/\partial w_\ell) = 0$ for all $\ell \in [n]$. Therefore, since $dx_{pa(i)} \in \mathcal{S}$, it holds that

$$\frac{\partial v_{pa(i),\gamma'}(w)}{\partial w_\ell}, \frac{\partial v_{pa(i),\gamma''}(w)}{\partial w_\ell} \in \mathcal{S} \quad \text{for all } \ell \in [n].$$

In addition, due to the identities

$$Dv_{i,\gamma'}(w) = Df_{i,p'}(v_{pa(i),\gamma'}(w)), \ \ Dv_{i,\gamma''}(w) = Df_{i,p''}(v_{pa(i),\gamma''}(w)),$$
$$\frac{\partial f_{i,p'}(v_{pa(i),\gamma'}(w))}{\partial w_\ell} = \left\langle \nabla f_{i,p'}(x_{pa(i)}), \frac{\partial v_{pa(i),\gamma'}(w)}{\partial w_\ell} \right\rangle,$$
$$\frac{\partial f_{i,p''}(v_{pa(i),\gamma''}(w))}{\partial w_\ell} = \left\langle \nabla f_{i,p''}(x_{pa(i)}), \frac{\partial v_{pa(i),\gamma''}}{\partial w_\ell} \right\rangle,$$

showing the following stronger statement suffices for proving Lemma 8:

$$\left\langle \nabla f_{i,p'}(x_{pa(i)}), z \right\rangle = \left\langle \nabla f_{i,p''}(x_{pa(i)}), z \right\rangle \quad \text{for all } \ z \in \mathcal{S}. \tag{9}$$

By Lemma 1 and the induction hypothesis ($dv_{pa(i),\gamma'} = dv_{pa(i),\gamma''}$), there exists $s > 0$ such that, for any $t \in (0,s)$ and $(j,p) \in \mathsf{Idx}_i$ and for $z_t = x_{pa(i)} + t \cdot dx_{pa(i)}$,

$$\phi_{i,p_{:j}}(z_t) > c_{i,p_{:j}} \text{ if } (\phi_{i,p_{:j}}(x_{pa(i)}) = c_{i,p_{:j}} \wedge \phi_{i,p_{:j}}(dx_{pa(i)}) > 0) \vee (\phi_{i,p_{:j}}(x_{pa(i)}) > c_{i,p_{:j}}),$$
$$\phi_{i,p_{:j}}(z_t) < c_{i,p_{:j}} \text{ if } (\phi_{i,p_{:j}}(x_{pa(i)}) = c_{i,p_{:j}} \wedge \phi_{i,p_{:j}}(dx_{pa(i)}) < 0) \vee (\phi_{i,p_{:j}}(x_{pa(i)}) < c_{i,p_{:j}}),$$
$$\phi_{i,p_{:j}}(z_t) = c_{i,p_{:j}} \text{ if } (\phi_{i,p_{:j}}(x_{pa(i)}) = c_{i,p_{:j}} \wedge \phi_{i,p_{:j}}(dx_{pa(i)}) = 0).$$

Since each $\phi_{i,p_{:j}}$ is linear (i.e., continuous) and by the definition of $\mathcal{S}$, for each $t \in (0,s)$, there exists an open neighborhood $\mathcal{O}_t \subset \mathcal{S}$ (open in $\mathcal{S}$) of z_t such that, for any $z \in \mathcal{O}_t$ and $(j,p) \in \mathsf{Idx}_i$,

$$\phi_{i,p_{:j}}(z) > c_{i,p_{:j}} \text{ if } (\phi_{i,p_{:j}}(x_{pa(i)}) = c_{i,p_{:j}} \wedge \phi_{i,p_{:j}}(dx_{pa(i)}) > 0) \vee (\phi_{i,p_{:j}}(x_{pa(i)}) > c_{i,p_{:j}}),$$
$$\phi_{i,p_{:j}}(z) < c_{i,p_{:j}} \text{ if } (\phi_{i,p_{:j}}(x_{pa(i)}) = c_{i,p_{:j}} \wedge \phi_{i,p_{:j}}(dx_{pa(i)}) < 0) \vee (\phi_{i,p_{:j}}(x_{pa(i)}) < c_{i,p_{:j}}),$$
$$\phi_{i,p_{:j}}(z) = c_{i,p_{:j}} \text{ if } (\phi_{i,p_{:j}}(x_{pa(i)}) = c_{i,p_{:j}} \wedge \phi_{i,p_{:j}}(dx_{pa(i)}) = 0). \tag{10}$$

Here, we claim that, for any $t \in (0,s)$,

$$\mathcal{O}_t \subset \mathrm{cl}(\mathcal{X}_{i,p'}) \cap \mathrm{cl}(\mathcal{X}_{i,p''}) \cap \mathcal{S} \triangleq \mathcal{B} \ \text{ and } \ x_{pa(i)} \in \mathcal{B}. \tag{11}$$

First, from the definition of $\mathcal{O}_t$, we have $\mathcal{O}_t \subset \mathcal{S}$. In addition, by Equation (10) and the definition of $\Gamma(w,\delta)$, either $\mathrm{sign}(\phi_{i,p'_{:j}}(z) - c_{i,p'_{:j}}) = p'_j$ or $\phi_{i,p'_{:j}}(z) = c_{i,p'_{:j}}$ for all $j \in [\mathrm{len}(p')]$ and $z \in \mathcal{O}_t$; the same argument also holds for p''. Hence, by Lemma 6, the LHS of Equation (11) holds. Likewise, we have $x_{pa(i)} \in \mathcal{B}$.

Due to the continuity of F_i, Equation (11) implies that, for any $t \in (0,s)$,

$$f_{i,p'} = f_{i,p''} \text{ on } \mathcal{O}_t, \tag{12}$$

i.e., $\langle\nabla f_{i,p'}(z_t), z\rangle = \langle\nabla f_{i,p''}(z_t), z\rangle$ for all $z \in \mathcal{S}$ and $t \in (0,s)$. Here, $f_{i,p'}$ and $f_{i,p''}$ are differentiable at z_t by Equation (11) and Assumption 1. Due to the analyticity of $f_{i,p'}$ and $f_{i,p''}$, this implies that, for any $z \in \mathcal{S}$, we have

$$\langle\nabla f_{i,p'}(x_{pa(i)}), z\rangle = \lim_{t\downarrow 0}\langle\nabla f_{i,p'}(z_t), z\rangle = \lim_{t\downarrow 0}\langle\nabla f_{i,p''}(z_t), z\rangle = \langle\nabla f_{i,p''}(x_{pa(i)}), z\rangle$$

where $f_{i,p'}$ and $f_{i,p''}$ are differentiable at $x_{pa(i)}$ by Equation (11) and Assumption 1. This proves Equation (9) and completes the proof of Lemma 8. □

4.6. Proof of Theorem 3

Under Assumptions 2 and 3, combining Lemmas 4 and 5, Corollary 1, and Lemma 8 completes the proof of Theorem 3.

5. Proof of Theorem 2

Here, we analyze the computational costs based on the program representations in Figures 4 and 5. For simplicity, we use A2 and A3 for the shorthand notations for Algorithm 2 and Algorithm 3, respectively. Under these setups, our cost analysis for $P(w)$ and $\mathrm{A2}(P, \mathrm{A3}(P,w)) = \mathrm{ours}(P,w)$ is as follows: for $\gamma, \gamma' \in \Gamma$ such that $\gamma = ((), \dots, (), \gamma_{n+1}, \dots, \gamma_{n+m})$, where $(\gamma_{n+1}, \dots, \gamma_{n+m})$ and $v(w) = (v_1(w), \dots, v_{n+m}(w))$ are the outputs of $\mathrm{A3}(P,w)$ and $w \in \mathcal{W}_{\gamma'}$:

$$\mathrm{cost}(P(w)) = \sum_{i=n+1}^{n+m}\left(\mathrm{cost}(f_{i,\gamma'_i}) + \mathrm{len}(\gamma'_i) + \sum_{j\in[\mathrm{len}(\gamma'_i)]}\mathrm{cost}(\phi_{i,\gamma'_{i,:j}})\right),$$

$$\mathrm{cost}(\mathrm{A2}(P, v(w), \gamma)) = \sum_{i=n+1}^{n+m} d_i\cdot(\mathrm{cost}(+) + \mathrm{cost}(\times)) + \mathrm{cost}(\nabla f_{i,\gamma_i})$$
$$\leq \sum_{i=n+1}^{n+m} 2d_i + \mathrm{cost}(\nabla f_{i,\gamma_i}),$$

$$\mathrm{cost}(\mathrm{A3}(P,w)) \leq \sum_{i=n+1}^{n+m}\Bigg(\max\{d_i - 1, 0\}\cdot\mathrm{cost}(+) + d_i\cdot\mathrm{cost}(\times)$$
$$+ \mathrm{cost}(f_{i,\gamma_i}) + \mathrm{cost}(\nabla f_{i,\gamma_i}) + \sum_{j\in[\mathrm{len}(\gamma_i)]}(2\mathrm{cost}(\phi_{i,\gamma_{i,:j}}) + 4\mathrm{cost}(>))\Bigg)$$
$$\leq \sum_{i=n+1}^{n+m}\left(2d_i + 4\,\mathrm{len}(\gamma_i) + \mathrm{cost}(f_{i,\gamma_i}) + \mathrm{cost}(\nabla f_{i,\gamma_i}) + 2\sum_{j\in[\mathrm{len}(\gamma_i)]}\mathrm{cost}(\phi_{i,\gamma_{i,:j}})\right),$$

$$\mathrm{cost}(\mathrm{ours}(P,w)) \leq \sum_{i=1}^{n}\mathrm{cost}(\text{sample }\delta_i) + \mathrm{A2}(P, v(w), \gamma)) + \mathrm{cost}(\mathrm{A3}(P,w))$$
$$\leq n + \sum_{i=n+1}^{n+m}\left(4d_i + 4\,\mathrm{len}(\gamma_i) + \mathrm{cost}(f_{i,\gamma_i}) + 2\,\mathrm{cost}(\nabla f_{i,\gamma_i}) + 2\sum_{j\in[\mathrm{len}(\gamma_i)]}\mathrm{cost}(\phi_{i,\gamma_{i,:j}})\right).$$

This implies the following:

$$\frac{\mathrm{cost}(\mathrm{ours}(P,w))}{\mathrm{cost}(P(w))} \leq \frac{n}{\mathrm{cost}(P(w))}$$
$$+\frac{\sum_{i=n+1}^{n+m}\left(4d_i + 4\mathrm{len}(\gamma_i) + \mathrm{cost}(f_{i,\gamma_i}) + 2\mathrm{cost}(\nabla f_{i,\gamma_i}) + 2\sum_{j\in[\mathrm{len}(\gamma_i)]}\mathrm{cost}(\phi_{i,\gamma_{i,:j}})\right)}{\mathrm{cost}(P(w))}$$
$$\leq 1 + \max_{i\in[n+m]\setminus[n]}\kappa_i = \kappa$$

where the first inequality is from the above bound and the second inequality is from the definition of κ_i and the assumption $\mathrm{cost}(P(w)) \geq n$. This completes the proof.

6. Conclusions

In this work, we proposed an efficient subdifferentiation algorithm for computing an element of the Clarke subdifferential of programs with linear branches. In particular, we generalized the existing algorithm in [14] and extended its application from polynomials to analytic functions. The computational cost of our algorithm is at most that of the function evaluation multiplied by an input-dimension-independent factor, for primitive functions whose arities and maximum depths of branches are independent of the input dimension. We believe that extending our algorithm to general functions (e.g., continuously differentiable functions), general branches (e.g., nonlinear branches), and general programs (e.g., programs with loops) will be an important future research direction.

Funding: This research was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00079, Artificial Intelligence Graduate School Program, Korea University) and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2022R1F1A1076180).

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A System for Large-Scale Machine Learning. In Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
2. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in PyTorch. In Proceedings of the NIPS Autodiff Workshop. 2017. Available online: https://openreview.net/forum?id=BJJsrmfCZ (accessed on 1 November 2023)
3. Frostig, R.; Johnson, M.; Leary, C. Compiling machine learning programs via high-level tracing. In Proceedings of the SysML Conference, Stanford, CA, USA, 15–16 February 2018; Volume 4.
4. Speelpenning, B. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*; University of Illinois at Urbana-Champaign: Champaign, IL, USA, 1980.
5. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Lake Tahoe, NV, USA, 3–6 December 2012; pp. 84–90.
6. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
7. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA, 4–9 December 2017; pp. 6000–6010.
8. Griewank, A.; Walther, A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed.; SIAM: Philadelphia, PA, USA, 2008.
9. Pearlmutter, B.A.; Siskind, J.M. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Trans. Program. Lang. Syst.* **2008**, *30*, 7:1–7:36. [CrossRef]
10. Baur, W.; Strassen, V. The complexity of partial derivatives. *Theor. Comput. Sci.* **1983**, *22*, 317–330. [CrossRef]
11. Griewank, A. On automatic differentiation. *Math. Program. Recent Dev. Appl.* **1989**, *6*, 83–107.
12. Bolte, J.; Boustany, R.; Pauwels, E.; Pesquet-Popescu, B. Nonsmooth automatic differentiation: A cheap gradient principle and other complexity results. *arXiv* **2022**, arXiv:2206.01730.
13. Griewank, A. Who invented the reverse mode of differentiation. *Doc. Math. Extra Vol. ISMP* **2012**, *389400* , 389–400.
14. Kakade, S.M.; Lee, J.D. Provably Correct Automatic Sub-Differentiation for Qualified Programs. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Montréal, QC, Canada, 3–8 December 2018; pp. 7125–7135.
15. Lee, W.; Yu, H.; Rival, X.; Yang, H. On Correctness of Automatic Differentiation for Non-Differentiable Functions. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Virtual, 6–12 December 2020; pp. 6719–6730.
16. Nesterov, Y. Lexicographic differentiation of nonsmooth functions. *Math. Program.* **2005**, *104*, 669–700. [CrossRef]
17. Khan, K.A.; Barton, P.I. Evaluating an element of the Clarke generalized Jacobian of a composite piecewise differentiable function. *ACM Trans. Math. Softw.* **2013**, *39*, 1–28.
18. Khan, K.A.; Barton, P.I. A vector forward mode of automatic differentiation for generalized derivative evaluation. *Optim. Methods Softw.* **2015**, *30*, 1185–1212. [CrossRef]

19. Barton, P.I.; Khan, K.A.; Stechlinski, P.; Watson, H.A.J. Computationally relevant generalized derivatives: Theory, evaluation and applications. *Optim. Methods Softw.* **2018**, *33*, 1030–1072. [CrossRef]
20. Khan, K.A. Branch-locking AD techniques for nonsmooth composite functions and nonsmooth implicit functions. *Optim. Methods Softw.* **2018**, *33*, 1127–1155. [CrossRef]
21. Griewank, A. Automatic directional differentiation of nonsmooth composite functions. In Proceedings of the Recent Developments in Optimization: Seventh French-German Conference on Optimization, Dijon, France, 27 June–2 July 1995; Springer: Berlin/Heidelberg, Germany, 1995; pp. 155–169.
22. Sahlodin, A.M.; Barton, P.I. Optimal campaign continuous manufacturing. *Ind. Eng. Chem. Res.* **2015**, *54*, 11344–11359. [CrossRef]
23. Sahlodin, A.M.; Watson, H.A.; Barton, P.I. Nonsmooth model for dynamic simulation of phase changes. *AIChE J.* **2016**, *62*, 3334–3351. [CrossRef]
24. Hanin, B. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics* **2019**, *7*, 992. [CrossRef]
25. Alghamdi, H.; Hafeez, G.; Ali, S.; Ullah, S.; Khan, M.I.; Murawwat, S.; Hua, L.G. An Integrated Model of Deep Learning and Heuristic Algorithm for Load Forecasting in Smart Grid. *Mathematics* **2023**, *11*, 4561. [CrossRef]
26. Boyd, S.P.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004.
27. Rehman, H.U.; Kumam, P.; Argyros, I.K.; Shutaywi, M.; Shah, Z. Optimization based methods for solving the equilibrium problems with applications in variational inequality problems and solution of Nash equilibrium models. *Mathematics* **2020**, *8*, 822. [CrossRef]
28. Davis, D.; Drusvyatskiy, D.; Kakade, S.; Lee, J.D. Stochastic subgradient method converges on tame functions. *Found. Comput. Math.* **2020**, *20*, 119–154. [CrossRef]
29. Bolte, J.; Pauwels, E. Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning. *Math. Program.* **2021**, *188*, 19–51. [CrossRef]
30. Scholtes, S. *Introduction to Piecewise Differentiable Equations*; Springer: Berlin/Heidelberg, Germany, 2012.
31. Griewank, A.; Bernt, J.U.; Radons, M.; Streubel, T. Solving piecewise linear systems in abs-normal form. *Linear Algebra Its Appl.* **2015**, *471*, 500–530. [CrossRef]
32. Bolte, J.; Pauwels, E. A mathematical model for automatic differentiation in machine learning. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Online, 6–12 December 2020; pp. 10809–10819.
33. Lee, W.; Park, S.; Aiken, A. On the Correctness of Automatic Differentiation for Neural Networks with Machine-Representable Parameters. *arXiv* **2023**, arXiv:2301.13370.
34. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Proceedings of the International Conference on Learning Representations (ICLR), Banff, AB, Canada, 14–16 April 2014.
35. Mityagin, B. The zero set of a real analytic function. *arXiv* **2015**, arXiv:1512.07276.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.