On the complexity of nonsmooth automatic differentiation

Jérôme BOLTE, Ryan BOUSTANY, Edouard PAUWELS, Béatrice PESQUET-POPESCU

Toulouse School of Economics, IRIT, Université de Toulouse, ANITI, THALES, France

Motivation

Baur and Strassen's result, 1983

The arithmetic complexity of evaluating a rational function's derivative is at most 5 times the complexity of function evaluation.

It has now been fifteen years of extensive and global empirical DNN training with nonsmooth components. It was founded on two assumptions:

1. backpropagation outputs a gradient almost.
2. the process is fast.

Motivation: extends the Baur-Strassen's result to the nonsmooth case.

Automatic differentiation in Machine learning

Given a training set $\{(x_i, y_i)\}_{i=1\ldots N}$, the supervised training of a neural network f consists in minimizing the empirical risk:

$$\min_{\theta\in\mathbb{R}^P} \quad J(\theta) := \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i,\theta), y_i) \qquad (1)$$

where $\theta \in \mathbb{R}^P$ are the network's weight parameters and ℓ is a loss function. In general, f is a composition of nonsmooth functions.

- **Automatic differentiation (AD):** A program that evaluates derivatives of numeric functions expressed as computer programs in an efficient and accurate way.

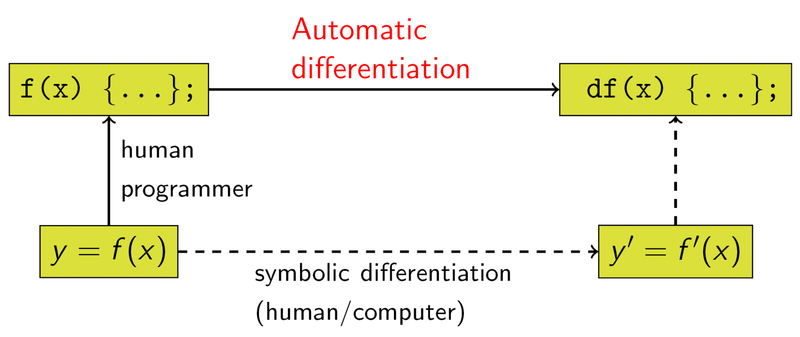


Figure: How automatic differentiation relates to symbolic differentiation

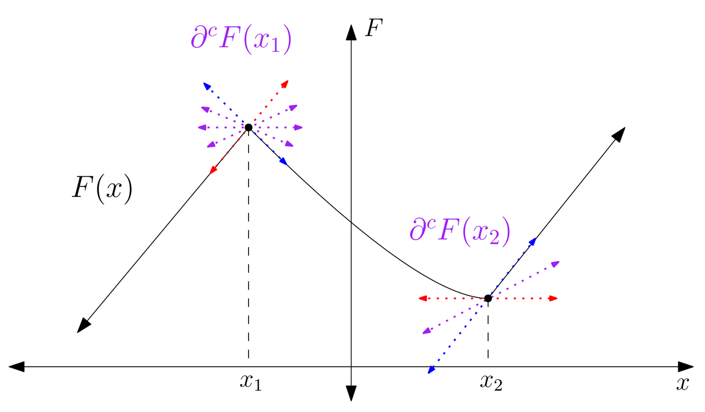
To solve (1), we should use **AD** to compute gradients (in the smooth case) or surrogate gradients (in the nonsmooth case).

Clarke gradients : a nonsmooth oracle

Given a locally Lipschitz continuous function $F : \mathbb{R}^p \to \mathbb{R}$, the ***Clarke subdifferential*** of F is

$$\partial^c F(x) = \mathrm{conv}\left\{\lim_{k\to+\infty} \nabla F(x_k) : x_k \in \mathrm{diff}_F, x_k \underset{k\to+\infty}{\to} x\right\} \qquad (2)$$

where diff_F is the full measure set where F is differentiable and ∇F is the standard gradient.



Notations

Let $F : \mathbb{R}^p \mapsto \mathbb{R}$ be a locally Lipschitz function.

- $\mathcal{D}$: collection of elementary operations used to compute F.
- $\mathcal{D}'$: collection of elementary operations used to compute F and a surrogate gradient of F.
- P : program which computes F using operations on $\mathcal{D}$.
- backprop(P): program that computes F and its backpropagation.
- cost(P): number of $\mathcal{D}$ operations required by the program P.
- cost(backprop(P)): number of $\mathcal{D}'$ operations required by the program backprop(P).

Nonsmooth AD with conservative gradients

How does backprop works ?

Consider a locally Lipschitz function $F : \mathbb{R}^p \to \mathbb{R}$ with a m-compositional representation implemented by a program P

$$F = g_1 \circ \ldots \circ g_m.$$

- For each i, x, choose $d_i(x) \in \partial^c g_i(x)$.
- Ex : g_i = ReLU and take $d_i(0) = 0$ (Tensorflow, Pytorch)

Chain-rule the d_i's:

$$d_F(x) := d_1\left(g_2\left(\ldots(g_m(x))\ldots\right)\right) \times d_2\left(g_3\left(\ldots(g_m(x))\ldots\right)\right)\ldots\times d_m(x)$$

⇒ backprop(P) **computes** $d_F(x)$.

This is how PyTorch and TensorFlow work.

The chain-rule, which is required for AD, usually fails for Clarke subgradients.
⇒ **introduce the conservative gradients.**

Conservative gradients

Let $F : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous.
We say that $D_F : \mathbb{R}^n \rightrightarrows \mathbb{R}^n$ is a *conservative gradient* for F if D_F has a closed graph, is locally bounded, and is nonempty with

$$\frac{\mathrm{d}}{\mathrm{d}t}(F\circ\gamma)(t) = D_F(\gamma(t))\dot\gamma(t) \text{ a.e.}$$

whenever γ is an absolutely continuous curve in $\mathbb{R}^n$. F is called *path differentiable.*

Some class of path differentiable functions:

1. convex functions,
2. semialgebraic functions (for instance piecewise polynomial functions),
3. "definable" functions: most of the functions implemented in practice.

⇒ Backpropagation is modeled by conservative gradients!
⇒ Sharp calculus rules used in ML are extended to nonsmooth functions!

Cheap conservative gradient

Let $F : \mathbb{R}^p \mapsto \mathbb{R}$ be a locally Lipschitz function and P a program who compute F using a dictionary $\mathcal{D}$ composed by path differentiable operations. If $F = g_1 \circ \ldots \circ g_m$ and each g_i are operations on $\mathcal{D}$, then :

1. F is path differentiable,
2. $\mathrm{cost}(\mathrm{backprop}(P)) \le \omega_b \times \mathrm{cost}(P)$ where ω_b is a constant.

Computational properties of conservative gradients vs others nonsmooth AD frameworks

Others alternative AD approaches

Computational overhead ratio

Minimum value of the quotient of the cost required to evaluate a program and "its" derived program by the cost to evaluate merely the program.

Let $F : \mathbb{R}^p \to \mathbb{R}$ be a locally Lipschitz function.

Others alternative implementable AD approaches:

- Try to evaluate elements of $\partial^c F$, based on directional derivatives - Khan and Barton (2012;2013;2015)
- Successive local approximations of F, based on lexicographic derivatives
- Computing Clarke subgradients using forward AD

Problem: All these procedures either require to evaluate p directional derivatives.

ReLU networks

Given a set of matrices $M_1 \in \mathbb{R}^{p_1\times p}$, $M_2 \in \mathbb{R}^{p_2\times p_1}$, $\ldots M_{L-1} \in \mathbb{R}^{p_{L-1}\times p_{L-2}}$, $M_L \in \mathbb{R}^{1\times p_{L-1}}$ we consider the associated ReLU network $F : \mathbb{R}^p \to \mathbb{R}$:

$$F : x \mapsto M_L\mathrm{ReLU}(M_{L-1}\mathrm{ReLU}(\ldots M_1 x)).$$

Link between p directional derivatives and matrix multiplication

Let $F : \mathbb{R}^p \to \mathbb{R}$ be a ReLU network function, computational cost defined over $\mathbb{R}$ by circuit complexity ('number of operations'):

1. $c(p) := \mathrm{cost}(p \times p$ matrix multiplication$)$
2. $c(p) \ge p^2$
3. cost(p directional derivatives of F) $\ge c(p)$

⇒ suffers from computational overhead scaling linearly in p

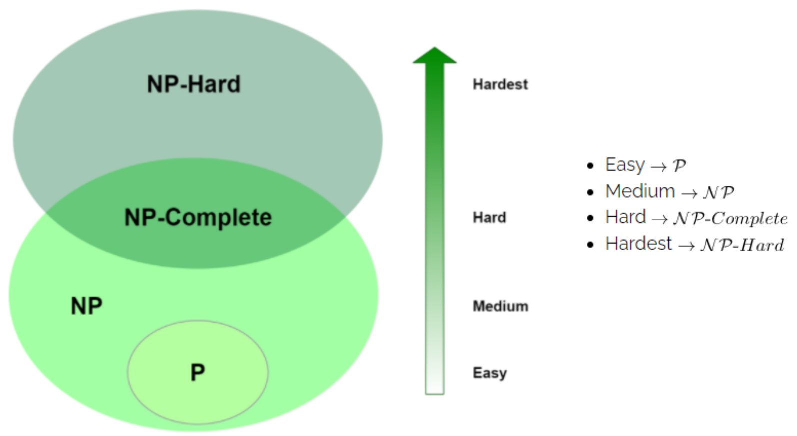
Computational hardness of subgradient enumeration

Goal: show the computational difficulty of dealing with the Clarke subgradient compared to conservative gradient.

Problem: conservative gradient enumeration

Consider $F : \mathbb{R}^p \to \mathbb{R}$ a ReLU network, $x \in \mathbb{R}^p$ and $D_F : \mathbb{R}^p \rightrightarrows \mathbb{R}^p$ a conservative gradient for F.

Compute two distinct elements in $D_F(x)$ or one element if it is a singleton.



Clarke subgradients and NP-Hardness

Let F be a ReLU network with matrix and vector entries in $\{-1, 0, 1\}$ and $x \in \mathbb{R}^p$:

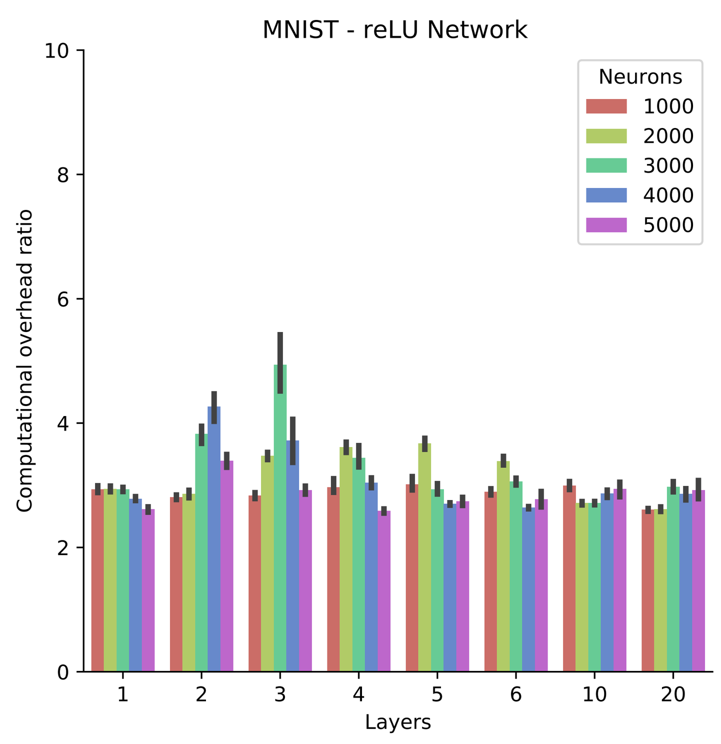
1. The enumeration problem with $D_F = \partial^c F$ is **NP-hard**.
2. Deciding if F is not differentiable at x is **NP-hard**.

Enumeration problem with backprop **and conservative gradients can be solved in polynomial time (easy).**

⇒ Conservative gradients etablish a "nonsmooth cheap gradient" with favorable computational properties compared to others nonsmooth oracles.

Applications with ReLU networks

Computational overhead ratio of MLP Cross Entropy with MNIST.



Computational overhead ratio of MLP Cross Entropy with MNIST according to the number of layers/neurons and the batch size.