Automatic Differentiation in PCF

Damiano Mazza
CNRS, LIPN, Université Paris 13

joint work with

Aloïs Brunel
Deepomatic

Michele Pagani
IRIF, Université Paris 7

Mathematical Foundations Seminar
Bath, 21 April 2020

(Happy 2772nd birthday Rome! :-))
Automatic Differentiation

Goal: compute the derivative (gradient, Jacobian...) of a function specified by a (first order) program.

Straight-line programs:

\[ G(x_1, x_2) = \]
\[ \text{let } z_1 = x_1 - x_2 \text{ in} \]
\[ \text{let } z_2 = z_1 \times z_1 \text{ in} \]
\[ \text{let } y = \sin(z_2) \text{ in } y \]

Semantics: \( \mathbb{R}^n \rightarrow \mathbb{R}^m \)

\[ [G](x_1, x_2) = \sin((x_1 - x_2)^2) \]
If $x$ contributes to $y_1, \ldots, y_m$ (and no other node), and we have

\[
y_1 := f_1(\ldots, x, \ldots) \\
\vdots \\
y_m := f_m(\ldots, x, \ldots)
\]

then,

\[
x := \sum_{i=1}^{m} \frac{\partial f_i}{\partial x}(\ldots, x, \ldots) \cdot y_i
\]
Interesting because:

- makes re-use and optimizations possible (TensorFlow);
- conceptually, it is easier to generalize a program transformation than an algorithm.
Differentiable Programming

Goal: compute the derivative (gradient, Jacobian…) of a function specified by an arbitrary program.

• How people do it today: let $x_1 : \mathbb{R}, \ldots, x_n : \mathbb{R} \vdash t : \mathbb{R}^m$ with $t$ arbitrary

  \begin{align*}
  t & \xrightarrow{\text{extract straight-line program}} G' \\
  & \xrightarrow{\text{first-order backprop}} \text{bp}(G')
  \end{align*}

  \begin{itemize}
  \item no reuse of work;
  \item \text{bp}(–) is not compositional.
  \end{itemize}

• Would like to do backprop directly on $t$:
  \begin{itemize}
  \item Pearlmutter and Siskind 2008
  \item Elliott [ICFP 2018]
  \item Wang, Zheng, Decker, Wu, Essertel, Rompf [ICFP 2019]
  \item Abadi and Plotkin [POPL 2020]
  \item our work [POPL 2020]; Huot, Staton, Vákár [FOSSACS 2020]; Wand…
  \end{itemize}
Demystifying Differentiable Programming


(A preprint exists since early 2018 at least).

• Interesting for two reasons:
  – a compositional program transformation implementing backprop using references and delimited continuations;
  – introduces the idea that derivatives at higher types are not necessary for computing the gradient of a first order program $M : \mathbb{R}^n \rightarrow \mathbb{R}$, even if $M$ uses higher-order primitives!

• What’s lacking:
  – the logical structure underlying backprop;
  – a soundness proof;
  – a complexity analysis.

• We provide all of that! (With a much simpler program transformation).
Why Do Derivatives at Higher Types Not Matter?

Wang et al. do not explain it this way, but it’s basically a matter of working with a free construction:

\[
\begin{align*}
\text{CG} \quad & \quad \downarrow \eta \\
\downarrow & \quad \text{backprop transformation} \\
\Lambda(\text{CG}) \quad & \quad \exists ! \xrightarrow{} \partial \Lambda
\end{align*}
\]

where

- \(\text{CG} = \) computational graphs (first order);
- \(\partial \Lambda = \) a calculus with first order derivatives (higher order to do backprop);
- \(\Lambda(\text{CG}) = \) the \(\lambda\)-calculus with first order primitives from \(\text{CG}\).

\(\Lambda(\text{CG})\) is the free 2-CCC on \(\text{CG}\) (technically delicate but morally true). Since \(\partial \Lambda\) is a 2-CCC, we get to lift the transformation for free.
Reverse Derivative, Compositionally

- Let $\mathbb{R}^\perp := \mathbb{R} \rightarrow E$ for an arbitrary vector space $E$.
- Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be derivable.
- Let

$$D_r f : \mathbb{R} \times \mathbb{R}^\perp \rightarrow \mathbb{R} \times \mathbb{R}^\perp$$

$$(x, x') \mapsto (f(x), \lambda a. x^* (a \cdot f'(x)))$$

- We have:
  1. $D_r (f \circ g) = D_r f \circ D_r g$;
  2. taking $E = \mathbb{R}$, $(\pi_2 D_r f(x, I)) 1 = f'(x)$. 


First Order Reverse AD, Compositionally

- Let $f : \mathbb{R}^n \to \mathbb{R}$ be derivable.
- Let
  
  $$\nabla^{-1}_D(f) : (\mathbb{R} \times \mathbb{R}^\bot)^n \longrightarrow \mathbb{R} \times \mathbb{R}^\bot$$

  $$((x_1, x_1^*), \ldots, (x_n, x_n^*)) \longmapsto (f(x), \lambda a. \sum_{i=1}^n x_i^*(a \cdot \partial_i f(x)))$$

  with $x = x_1, \ldots, x_n$. The $x_i^*$ are called backpropagators.

- We have:
  1. $\nabla^{-1}_D(f \circ^i g) = \nabla^{-1}_D(f) \circ^i \nabla^{-1}_D(g)$ (morphism of operads = product-preserving functor);
  2. taking $E = \mathbb{R}^n, (\pi_2 \nabla^{-1}_D(f)((x_1, \iota_1), \ldots, (x_n, \iota_n))1 = \nabla f(x_1, \ldots, x_n)$, where $\iota_i : \mathbb{R} \to \mathbb{R}^n$ is the $i$-th injection.

- This gives the reverse AD transformation for computational graphs.
  By Wang et al., it actually gives it for the whole simply-typed $\lambda$-calculus!
A Simply-Typed λ-Calculus with Linear Negation

• Types:

\[ A, B ::= R \mid A \times B \mid A \to B \mid R^\perp \]

(morally, \(R^{\perp} = R \to R^d\) for arbitrary \(d \in \mathbb{N}\))

• Terms:

\[ t, u ::= x \mid \lambda x.t \mid tu \mid \langle t, u \rangle \mid t[\langle x, y \rangle := u] \mid \phi \]

where \(\phi\) ranges over a set of real function symbols (constants, sum, product...).

• Typing: standard, but tracking linearity (for the type \(R^{\perp}\)).

• Operational semantics: standard, plus linear factoring

\[ x^*t + x^*u \to x^*(t + u) \quad \text{provided} \quad x^* : R^{\perp} \]
The Reverse AD Transformation

- We (arbitrarily!) fix a map \((\phi, i) \mapsto \partial_i \phi\), with \(1 \leq i \leq \text{arity}(\phi)\).
- On types:

\[
\begin{align*}
\hat{\mathcal{D}}(R) & := R \times R^\perp \\
\hat{\mathcal{D}}(A \rightarrow B) & := \hat{\mathcal{D}}(A) \rightarrow \hat{\mathcal{D}}(B) \\
\hat{\mathcal{D}}(A \times B) & := \hat{\mathcal{D}}(A) \times \hat{\mathcal{D}}(B)
\end{align*}
\]

- On terms:

\[
\begin{align*}
\hat{\mathcal{D}}(x^A) & := x \hat{\mathcal{D}}(A) \\
\hat{\mathcal{D}}(\langle t, u \rangle) & := \langle \hat{\mathcal{D}}(t), \hat{\mathcal{D}}(u) \rangle \\
\hat{\mathcal{D}}(tu) & := \hat{\mathcal{D}}(t) \hat{\mathcal{D}}(u) \\
\hat{\mathcal{D}}(t[\langle x, y \rangle := u]) & := \hat{\mathcal{D}}(t)[\langle x, y \rangle := \hat{\mathcal{D}}(u)] \\
\hat{\mathcal{D}}(\phi) & := \lambda p. \left( \phi\langle x \rangle, \lambda a. \sum_{i=1}^{k} x_i^* (a \cdot \partial_i \phi\langle x \rangle) \right) \left[ \langle x, x^* \rangle := p \right]
\end{align*}
\]
Properties

Soundness and complexity bound:

\[
\begin{align*}
    \text{fold } F \ Z \ [a_1, \ldots, a_n] & \quad \xrightarrow{\mathcal{D}} \quad \text{fold } \mathcal{D}(F) \mathcal{D}(Z) \ [\mathcal{D}(a_1), \ldots, \mathcal{D}(a_n)] \\
\end{align*}
\]

Conservativity crucially relies on linear factoring.

The transformation preserves the structure of programs and data types:
Why linear negation?

There is an issue of sharing. Let

\[ G := \sin(z_2) [z_2 := z_1 \cdot z_1 [z_1 := x_1 - x_2]]. \]

Then, \( \overleftarrow{D}(G) \) reduces to

\[
\langle \sin(z_2), \lambda a. z_2^*(\cos(z_2) \cdot a) \rangle [\langle z_2, z_2^* \rangle := \langle z_1 \cdot z_1, \lambda b. z_1^*(z_1 \cdot b) + z_1^*(z_1 \cdot b) \rangle] \\
[\langle z_1, z_1^* \rangle := \langle x_1 - x_2, \lambda c. x_1^*(1 \cdot c) + x_2^*((-1) \cdot c) \rangle].
\]

In \( \text{bp}(G) \), \( F \) is not duplicated: it would cause a loss of efficiency.
PCF

• How about arbitrary programs?
  – Fwd mode AD for straight-line programs with if and goto
  – Soundness almost everywhere [Joss 1976] [Speelpenning 1980]
• PCF = STLC + conditionals and fixpoints:

  \[ M ::= [\text{STLC}] \mid \text{if } P \leq 0 \text{ then } M \text{ else } N \mid \text{fix } f^{A \rightarrow B}.M \]

  \[
  \begin{align*}
  \text{if } r \leq 0 \text{ then } M_1 \text{ else } M_2 & \quad \rightarrow \quad M_{r \leq 0}^{1:2} \\
  \text{fix } f.M & \quad \rightarrow \quad M\{\lambda x. (\text{fix } f.M)x/f\}
  \end{align*}
  \]

• Function symbols \( \phi \) may be supposed to be total and analytic.
• Reverse AD transformation straightforward:

  \[
  \begin{align*}
  \overleftarrow{D}(\text{if } P \leq 0 \text{ then } M \text{ else } N) & \quad := \quad \text{if } \pi_1(\overrightarrow{D}(P)) \leq 0 \text{ then } \overleftarrow{D}(M) \text{ else } \overleftarrow{D}(N) \\
  \overleftarrow{D}(\text{fix } f.M) & \quad := \quad \text{fix } f.\overleftarrow{D}(M)
  \end{align*}
  \]
Traces

• On types:

\[ A_1 \sqsubseteq A \quad \ldots \quad A_n \sqsubseteq A \quad B' \sqsubseteq B \]
\[ R \sqsubseteq R \]
\[ A'_1 \times \cdots \times A'_n \rightarrow B' \sqsubseteq A \rightarrow B \]
\[ A'_1 \times \cdots \times A'_n \sqsubseteq A \times \cdots \times A_n \]

• On terms:

\[ \Xi, x_1, \ldots, x_n \sqsubseteq x \vdash x_i \sqsubseteq x \]
\[ \Xi \vdash \phi \sqsubseteq \phi \]
\[ \Xi, x_1, \ldots, x_n \sqsubseteq x \vdash t \sqsubseteq M \]
\[ \Xi \vdash \lambda \langle x_1, \ldots, x_n \rangle.t \sqsubseteq \lambda x.M \]
\[ \Xi \vdash \lambda \langle x_1, \ldots, x_n \rangle.t \sqsubseteq \lambda x.M \]
\[ \Xi \vdash \langle u_1, \ldots, u_n \rangle \sqsubseteq MN \]
\[ \Xi \vdash t_3-i \sqsubseteq P \quad \Xi \vdash t_i \sqsubseteq M_i \]
\[ \Xi \vdash \pi_i(t_1, t_2) \sqsubseteq \text{if } P \leq 0 \text{ then } M_1 \text{ else } M_2 \]
\[ \Xi \vdash t \sqsubseteq \text{fix} f.M \]
\[ \Xi \vdash t \sqsubseteq \text{fix} f.M \]

where \( \text{fix}_0 f.M := \text{fix} f.f \) and \( \text{fix}_{n+1} f.M := (\lambda f.M)(\lambda x.(\text{fix}_n f.M)x) \)

• On reductions: \( (t \rightarrow^* u) \sqsubseteq (M \rightarrow N) \) with \( t \sqsubseteq M \) and \( u \sqsubseteq N \).

• \( t \sqsubseteq M \) if (reduction of \( t \)) \sqsubseteq (reduction of \( M \) to NF).
Soundness on Stable Points

• Let

\[ x_1 : R, \ldots, x_n : R \vdash M : R. \]

• A point \( r \in \mathbb{R}^n \) is \textit{stable} for \( M \) if

\[
\text{there exist } t \sqsubseteq M \text{ and } \varepsilon > 0 \text{ such that } \\
\forall r' \in B_{\varepsilon}(r) \quad t\{r'/x\} \sqsubseteq M\{r'/x\}. 
\]

\textbf{Theorem.} \textit{If } \( r \in \mathbb{R}^n \text{ is stable, } \pi_2 \overrightarrow{D}(M)\{r/x\} \textit{ computes } \nabla \llbracket M \rrbracket(r).\]

• The proof uses:
  - \( t \sqsubseteq M \) implies \( \overrightarrow{D}(t) \sqsubseteq \overrightarrow{D}(M) \)
  - [POPL 2020]
  - \ldots and more.
Bounding Unstable Points

• By the previous theorem,

\[ \pi_2 \delta(M) \{ r/x \} 1 \text{ fails to compute } \nabla[\lfloor M \rfloor](r) \implies r \text{ is unstable} \]

• Important: this is not a problem of lack of differentiability!
We are saying that \( \delta(M) \) fails when \( \nabla[\lfloor M \rfloor] \) is defined! Example:

\[
\text{if } x = 0 \text{ then } 0 \text{ else } x
\]

• Not as silly as it seems! Things like

\[ \text{ReLU}(x) - \text{ReLU}(-x) \]

happen all the time in neural networks.

• **Theorem.** The set of unstable points is of measure zero.
Proof by logical predicates (I’m hiding important details…).
Discussion

• Adding inductive types to [POPL 2020] is immediate (and anything total, probably).

• How meaningful is measure-zero? (Everything computable has measure zero…)

• With conditionals, AD does not preserve the natural semantics:

\[
[M] = [N] \text{ but } \left[ \bar{\mathcal{D}}(M) \right] \neq \left[ \bar{\mathcal{D}}(N) \right].
\]

So an internal \( \bar{\mathcal{D}} \) needs a non-standard semantics (Abadi and Plotkin [POPL 2020]).

• Implementation: linear factoring?

• If \( A \neq \mathbb{R}^n \) and \( t : A \to \mathbb{R} \), \( \bar{\mathcal{D}}(t) \) does not in general compute \( \mathcal{D}t \) in the sense of the differential \( \lambda \)-calculus.

• If \( A \) is first order, Huot, Staton, Vákár [FOSSACS 2020] prove that it’s still meaningful. What about \( A \) higher order? Do we care?