A Language for Evaluating Derivatives of Functionals Using Automatic Differentiation

Pietro Di Gianantonio [1]   Abbas Edalat [2]   Ran Gutin [2]

Introduction

Sparkled by ML, a recent flourishing of functional languages with derivative operators has occurred

Here is a short list of works most related to our present study:

- Abadi, Plotkin. A simple differentiable programming language, 2021
- Ehrhard, Regnier, The differential lambda-calculus, 2003
- Huot, Staton, Vakar. Higher order automatic differentiation of higher order functions, 2022
- Manzyuk, A simply typed λ-calculus of forward automatic differentiation, 2012
- Mazza, Pagani, Automatic differentiation in PCF, 2021
- Sherman, Michel, Carbin. Computable semantics for differentiable programming with higher-order functions and datatypes, 2021

Commonalities among these studies:

- A simple functional (imperative) programming language
- A derivative operator implemented using forward (or reverse) mode automatic differentiation
- Definition of an operational (denotational) semantics
- Discussion of the correctness of the derivative operator
- Result expressivity

All these are also present in our work.

Differences in our approach

We start by studing the derivative operator in Domain theory and exact real number computation

We have developed a language with a derivative operator for its own interest, with:

- Di Gianantonio, P. and A. Edalat, A language for differentiable functions, (2013)
- The operational semantics of derivative given in terms of symbolic computation
- The idea from automatic differentiation used only at the level of denotational semantics

Since we approach from a different perspective, we consider different problems

Key aspects of our solution

In our study, we use automatic differentiation in a language implementing:

- Exact real number computation where real numbers are obtained as limit of their approximations (finite elements) and rational intervals
- Results that are always correct, including handling the if-then-else constructor
- Fixed point operator
- Derivative of functionals and higher order functions in general
- Semantics given using domain theory
- Definability of (domain) computable elements

The language

Our language is a simply typed lambda calculus:

$$e ::= c \mid x^\tau \mid e_1 e_2 \mid \lambda x^\tau . e$$

With a basic type for real π and dual number δ:

$$\tau \;::=\; o \mid \nu \mid \pi \mid \delta \mid \tau \to \tau$$

- Constants include
  - arithmetic operations, min, max,
  - integration on the interval [0,1], and
  - a directional derivative operator for second-order function types, $\tau = \vec{\tau} \to \delta$, $\mathrm{L}_\tau : ((\tau \to \vec{\tau_\pi} \to \vec{\tau_\pi} \to \pi)$,
  - a recursive operator Y,
  - and more...

Dual numbers

Automatic differentiation (forward mode), for simplicity, uses:

- Dual numbers: numbers in the form $x + \epsilon y$ where ϵ is an infinitesimal value, such that $\epsilon * \epsilon = 0$.
  - Using this equality, arithmetic operations, and in general a function f on reals, can naturally extend to dual numbers with the property:

$$f(x + \varepsilon y) = f(x) + \varepsilon y \cdot f'(x)$$

where f' is the derivative of f

This provides a neat reformulation of forward mode automatic differentiation

Exact computation

- Exact computation is rarely considered in automatic differentiation
- We use a sort interval analysis, and rational intervals to approximate reals.
- An advantage is that intervals naturally represent the (directional) derivative even for non-smooth functions like the absolute value.

Derivative of functionals

Our functional language contains some second-order primitives, such as integration and the maximum of a function on the unit interval $[0, 1]$.

How should these primitives act on the infinitesimal part?

How the infinitesimal part be used to evaluate the derivatives of functionals?

We also deal with partial elements and non-smooth functions

Directional derivative

We exploit the fact that a function space on reals is also a topological vector space

Definition

Given a vector space X and a function on the real line $f : X \to \mathbb{R}$, the domain-theoretic directional derivative $L\,f : X \times X \to \mathbb{IR}$ evaluated at $x \in X$ in the direction $x' \in X$ is defined as the interval:

$$L\,f(x, x') := \left[\liminf_{\substack{y \to x, z \to x' \\ r \to 0^+}} \frac{f(y + rz) - f(y)}{r}, \limsup_{\substack{y \to x, z \to x' \\ r \to 0^+}} \frac{f(y + rz) - f(y)}{r} \right]$$

Derivative of non-smooth functions

For example, the absolute value:

$$L\,(\lambda x.|x|)(0,1) = [-1,1]$$

Clarke gradient

On Banach spaces (normed vector spaces), the domain-theoretic directional derivative coincides with the Clarke gradient, which is a mathematical generalization of the derivative to non-smooth functions.

However, the domain-theoretic directional derivative can be applied to topological vector spaces, to function spaces with the compact-open topology

- Most other approaches deal only with smooth functions
  - differential λ-calculus,
- or use partial functions as derivatives
- Clarke gradient are used in
  - Sherman, Michel, Carbin. Computable semantics for differentiable programming with higher-order functions and datatypes, 2021

If-then-else

`if-then-else` creates discontinuous functions which are not differentiable even with generalized derivatives.

In our approach, differentiable functions are restricted in their use of `if-then-else`

- there are no functions from dual to boolean

```
if b then f else g
```

A dual value cannot appear in the guard `b`.

We prove that `if-then-else` can be replaced by `min,max`.

Proposition

Every differentiable, Scott-computable function can be expressed using `min`, `max` in place of `if-then-else`.

Recursive operator

Surprisingly, few formal functional languages introduce fixed point operators.

Here's a fancy recursive definition of the identity without a base case:

$$g = Y(\lambda f x\,.\,(x + (fx))/2)$$

Unfolding it:

$$g = \lambda x\,.\,(x + (x + (gx))/2)/2$$

Example

The 3rd unfolding of g gives a function equivalent to:

$$\lambda x\,.\,((15x + (gx))/16$$

- If g(x) and g'(x) are known to be bounded by the interval $[-1, 1]$,
- the 3-th unfolding of g on the values $1/2 + \epsilon$ returns the values $[7/16, 1/2] + \epsilon[7/8, 1]$.

By repeated unfolding, we get a better approximation of the value and derivative of the identity function on $1/2$.

Bounds on derivative

Bounds on the return values and derivative of *g* are necessary.

- It is possible to bound the output of a function to a given interval by composing it with a projection function
- The same trick does not work with derivatives

Solution: allow only non-expansive primitives inside recursive definitions.

The fixed point will be a non-expansive function and its derivative should be contained in the interval [-1,1].

Results: adequacy

- Operational semantics
- Denotational semantics
- Proof of adequacy

Result: Automatic Differentiation evaluates directional derivatives

- We introduce a family of derivative operators L_τ with operational semantics:

$$\mathsf{L}_\tau\, Ffg \to \mathsf{ln}\, F(f +_\tau \epsilon_\tau g)$$

Theorem

The language's derivative operators L_τ are sound w.r.t. the domain-theoretic directional derivative L. That is, for any second-order function F, and first-order functions f, g, if F, f, g define total functionals and functions on reals, then:

$$\mathcal{E}[\![\mathsf{L}_\tau\, F\, f\, g]\!] \sqsubseteq L(\mathcal{E}[\![F]\!])^{\mathbf{s}}((\mathcal{E}[\![f]\!])^{\mathbf{s}}, (\mathcal{E}[\![g]\!])^{\mathbf{s}})$$

Consistency of infinitesimal information

We need to establish, even for high order functionals or partially defined functionals, when the infinitesimal is consistent with appreciable parts.

The right notion is the one of the directional derivative.

We prove that consistency holds for any definable function of the language.

This problem is not obvious due to higher order primitives in the language, such as integration, and fixed points of functionals. The solution lies in logical relations.

Definability

Is the language complete?

Computable functions can be defined through Scott-domains

In any computable function (and functional) definable in the language?

There's a positive answer for functions and linear functionals

However, it remains an open problem for general functionals

Examples

The directional derivative of

- $G = \lambda g.\, \lambda x.\, x + g(x)^2$
- at $f = \lambda u.\, u^2$
- in the direction k

$$
\begin{aligned}
& L_{\delta \to \delta,\, \delta} G(\lambda u.\, u^2)\, y\, k\, 0 \to \mathrm{In}(G(\lambda u.\, u^2 + \varepsilon k(u))(y + \varepsilon 0)) \\
& \to \mathrm{In}(\lambda x.\, x + (x^2 + \varepsilon k(x)) * (x^2 + \varepsilon k(x)))(y + \varepsilon 0)) \\
& \to \mathrm{In}(y + (y^4 + \varepsilon 2 y^2 k(y))) \to 2y^2 k(y)
\end{aligned}
$$

- Birkisson, A. and T. A. Driscoll, Automatic Frechet differentiation for the numerical solution of boundary-value problems,

Examples

Initial value problem,

$$\dot{y}'(x) = v(y(x)), \qquad y(0) = 0$$

- $v : O \to \mathbb{R}$ defined on an open neighbourhood $O \subset \mathbb{R}$
- let $e_v : \pi \to \pi$, be an expression defining the function v.
- the solution of IVP is given by the expression:

$$Y(\lambda f \,.\, \lambda x \,.\, \mathrm{int}\, \lambda t \,.\, x * \mathrm{pr}_M(e_v(f(t * x))))$$