Confusion of Tagged Perturbations in Forward Automatic Differentiation of Higher-Order Functions

Oleksandr Manzyuk* Barak A. Pearlmutter*
Alexey Andreyevich Radul* David R. Rush* Jeffrey Mark Siskind†

November 22, 2012

Abstract

Forward Automatic Differentiation (AD) is a technique for augmenting programs to both perform their original calculation and also compute its directional derivative. The essence of Forward AD is to attach a derivative value to each number, and propagate these through the computation. When derivatives are nested, the distinct derivative calculations, and their associated attached values, must be distinguished. In dynamic languages this is typically accomplished by creating a unique tag for each application of the derivative operator, tagging the attached values, and overloading the arithmetic operators. We exhibit a subtle bug, present in fielded implementations, in which perturbations are confused *despite* the tagging machinery.

1 Forward AD using Tagged Tangents

Forward AD (Wengert, 1964) computes the derivative of a function $f : \mathbb{R} \to \alpha$ at a point c by evaluating $f(c + \varepsilon)$ under a nonstandard interpretation that associates a conceptually infinitesimal perturbation with each real number, propagates these augmented values according to the rules of calculus (Leibniz, 1664), and extracts the perturbation of the result. When x is a number, we use $x + \bar{x}\varepsilon$ to denote a tangent-vector bundle: the primal value x bundled with the tangent value $\bar{x}$, where $\bar{x}$ has the same type as x. We consider this tangent-vector bundle to also be a number, with arithmetic defined by regarding it as a truncated power series, or equivalently, by taking $\varepsilon^2 = 0$ but $\varepsilon \neq 0$. This implies that $f(x + \bar{x}\varepsilon) = f(x) + \bar{x}f'(x)\varepsilon$ where $f'(x)$ is the first derivative of f at x (Newton, 1704).

*Hamilton Inst & Dept Comp Sci, NUI Maynooth, Co. Kildare, Ireland

†School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907-2035, USA

We can define a first-derivative operator[1] by

$$\mathcal{D}\ f\ x = \mathbf{tg}\ \varepsilon\ (f\ (x+\varepsilon)) \qquad\qquad \text{where } \varepsilon \text{ is fresh} \tag{1}$$

In order for $\mathcal{D}$ to nest correctly we must distinguish between different sets of tangent spaces introduced by different invocations of $\mathcal{D}$ (Lavendhomme, 1996), which can be implemented by tagging (Siskind and Pearlmutter, 2005, 2008). We will indicate different tags by different subscripts on ε. The tangent extraction function **tg** extracts the tangent part of a tangent-vector bundle, with the appropriate tag indicated in the first argument.

The tangent part of a numeric tangent-vector bundle is:

$$\mathbf{tg}\ \varepsilon\ (a + b\varepsilon) \triangleq b \tag{2}$$

When the primal part of the tangent-vector bundle is a function, the tangent-vector bundle is itself a function and **tg** is defined by post-composition:

$$\mathbf{tg}\ \varepsilon\ (\lambda x\ .\ e) \triangleq \lambda x\ .\ \mathbf{tg}\ \varepsilon\ e \tag{3}$$

This is the technique used to implement Forward AD in dynamic languages: arithmetic operators are overloaded to handle the chosen representation of numeric tangent-vector bundles, with tags generated using "`gensym`" or an analogous mechanism.

2 A Bug

If we have properly defined $\mathcal{D}$ and **tg**, then we can reasonably expect to use them to calculate correct derivatives in commonly occurring mathematical situations. In particular, if we define an offset operator:

$$\begin{aligned} s &: \mathbb{R} \to (\mathbb{R} \to \alpha) \to (\mathbb{R} \to \alpha) \\ s\ u\ f\ x &\triangleq f\ (x+u) \end{aligned} \tag{4}$$

the derivative of s at zero should be the same as the derivative operator: if we define

$$\hat{\mathcal{D}} \triangleq \mathcal{D}\ s\ 0 \tag{5}$$

then $\hat{\mathcal{D}} = \mathcal{D}$ should hold, since

$$\mathcal{D}\ f\ y = \mathbf{tg}\ \varepsilon\ (f\ (y+\varepsilon)) = \mathbf{tg}\ \varepsilon\ (f(y) + f'(y)\varepsilon) = f'(y) \tag{6a}$$

$$\begin{aligned} \hat{\mathcal{D}}\ f\ y = \mathcal{D}\ s\ 0\ f\ y &= (d/du) s\ u\ f\ y|_{u=0} \\ &= (d/du) f(y+u)|_{u=0} = f'(y) \end{aligned} \tag{6b}$$

[1] The type signature would be $\mathcal{D} : (\mathbb{R} \to \alpha) \to \mathbb{R} \to \alpha'$ where α' is the tangent space of α. It is natural to equate $\mathbb{R}' = \mathbb{R}$, and because we only consider $\mathbb{R}$ and functions built on $\mathbb{R}$, and we equate $(\alpha \to \beta)' = \alpha \to \beta'$, and it follows from Church encoding that $(\alpha \times \beta)' = \alpha' \times \beta'$, we can in all present examples equate $\alpha' = \alpha$. A full treatment of this topic is beyond our present scope.

Unfortunately, as we shall see, the above can exhibit a subtle bug:

$$\hat{\mathcal{D}}\ (\hat{\mathcal{D}}\ f)\ x = 0 \neq \mathcal{D}\ (\mathcal{D}\ f)\ x = f''(x) \tag{7}$$

This is not an artificial example. It is quite natural to construct an x-axis differential operator and apply it to a two-dimensional function twice, along the x and then y axis directions, by applying the operator, a rotation, and the operator again, thus creating precisely this sort of cascaded use of a defined differential operator.

Note that

$$\begin{aligned}\hat{\mathcal{D}} = \mathcal{D}\ s\ 0 &= \mathbf{tg}\ \varepsilon\ (s\ (0+\varepsilon))\\ &= \mathbf{tg}\ \varepsilon\ (\lambda f\ .\ \lambda x\ .\ f\ (x+\varepsilon))\\ &= \lambda f\ .\ \lambda x\ .\ \mathbf{tg}\ \varepsilon\ (f\ (x+\varepsilon)) \qquad (8)\end{aligned}$$

Assuming that $g : \mathbb{R} \to \mathbb{R}$ we can substitute using (8) and then reduce:

$$\begin{aligned}\hat{\mathcal{D}}\ (\hat{\mathcal{D}}\ g)\ y &= (\lambda f\ .\ \lambda x\ .\ \mathbf{tg}\ \varepsilon\ (f\ (x+\varepsilon)))\\ &\qquad ((\lambda f\ .\ \lambda x\ .\ \mathbf{tg}\ \varepsilon\ (f\ (x+\varepsilon)))\ g)\ y && \text{(9a)}\\ &= (\lambda f\ .\ \lambda x\ .\ \mathbf{tg}\ \varepsilon\ (f\ (x+\varepsilon)))\\ &\qquad (\lambda x\ .\ \mathbf{tg}\ \varepsilon\ (g\ (x+\varepsilon)))\ y && \text{(9b)}\\ &= (\lambda x\ .\ \mathbf{tg}\ \varepsilon\ ((\lambda x\ .\ \mathbf{tg}\ \varepsilon\ (g\ (x+\varepsilon)))\ (x+\varepsilon)))\ y && \text{(9c)}\\ &= \mathbf{tg}\ \varepsilon\ ((\lambda x\ .\ \mathbf{tg}\ \varepsilon\ (g\ (x+\varepsilon)))\ (y+\varepsilon)) && \text{(9d)}\\ &= \mathbf{tg}\ \varepsilon\ (\mathbf{tg}\ \varepsilon\ (g\ ((y+\varepsilon)+\varepsilon))) && \text{(9e)}\\ &= \mathbf{tg}\ \varepsilon\ (\mathbf{tg}\ \varepsilon\ (g\ (y+2\varepsilon))) && \text{(9f)}\\ &= \mathbf{tg}\ \varepsilon\ (\mathbf{tg}\ \varepsilon\ (g(y)+2g'(y)\varepsilon)) && \text{(9g)}\\ &= \mathbf{tg}\ \varepsilon\ (2g'(y)) && \text{(9h)}\\ &= 0 && \text{(9i)}\end{aligned}$$

This went wrong, yielding 0 instead of $g''(y)$, because the tag ε was generated exactly *once*, when the definition of $\hat{\mathcal{D}}$ was reduced to normal form in (8). The instantiation of $\mathcal{D}$ is the point at which a fresh tag is introduced; early instantiation can result in reuse of the same tag in logically distinct derivative calculations. Here, the first derivative and the second derivative become confused at (9f). We have two nested applications of **tg** for ε, but for correctness these should be distinctly tagged: ε_1 vs ε_2. If $\hat{\mathcal{D}}$ were not already reduced to normal form, and we instead substitute its definition, then $\mathcal{D}$ will be instantiated twice, giving two fresh tags and a correct result:

$$\begin{aligned}
\hat{\mathcal{D}}\ (\hat{\mathcal{D}}\ g)\ y &= \mathcal{D}\ s\ 0\ (\mathcal{D}\ s\ 0\ g)\ y && \text{(10a)}\\
&= (\lambda f\ .\ \lambda x\ .\ \mathbf{tg}\ \varepsilon_1\ (f\ (x+\varepsilon_1)))\\
&\qquad ((\lambda f\ .\ \lambda x\ .\ \mathbf{tg}\ \varepsilon_2\ (f\ (x+\varepsilon_2)))\ g)\ y && \text{(10b)}\\
&= (\lambda f\ .\ \lambda x\ .\ \mathbf{tg}\ \varepsilon_1\ (f\ (x+\varepsilon_1)))\\
&\qquad (\lambda x\ .\ \mathbf{tg}\ \varepsilon_2\ (g\ (x+\varepsilon_2)))\ y && \text{(10c)}\\
&= (\lambda x\ .\ \mathbf{tg}\ \varepsilon_1\ ((\lambda x\ .\ \mathbf{tg}\ \varepsilon_2\ (g\ (x+\varepsilon_2)))\ (x+\varepsilon_1)))\ y && \text{(10d)}\\
&= \mathbf{tg}\ \varepsilon_1\ ((\lambda x\ .\ \mathbf{tg}\ \varepsilon_2\ (g\ (x+\varepsilon_2)))\ (y+\varepsilon_1)) && \text{(10e)}\\
&= \mathbf{tg}\ \varepsilon_1\ (\mathbf{tg}\ \varepsilon_2\ (g\ ((y+\varepsilon_1)+\varepsilon_2))) && \text{(10f)}\\
&= \mathbf{tg}\ \varepsilon_1\ (\mathbf{tg}\ \varepsilon_2\ (g(y+\varepsilon_1)+g'(y+\varepsilon_1)\varepsilon_2)) && \text{(10g)}\\
&= \mathbf{tg}\ \varepsilon_1\ g'(y+\varepsilon_1) && \text{(10h)}\\
&= \mathbf{tg}\ \varepsilon_1\ (g'(y)+g''(y)\varepsilon_1) && \text{(10i)}\\
&= g''(y) && \text{(10j)}
\end{aligned}$$

3 Discussion

In a Forward AD system which uses tags to distinguish instances of $\mathcal{D}$, eta reduction is unsound. The definition $\hat{\mathcal{D}}\ f\ y = \mathcal{D}\ s\ 0\ f\ y$ must not be eta reduced to $\hat{\mathcal{D}} = \mathcal{D}\ s\ 0$, and one must not memoize or hoist $\mathcal{D}\ s\ 0$, as it is impure due to the requirement for a fresh tag. Even the above constraint can be insufficient when $\hat{\mathcal{D}}$ is applied to a function that is not $\mathbb{R} \to \mathbb{R}$ but instead $\mathbb{R} \to \alpha$ for some other α. In fact, expanded variants of $\hat{\mathcal{D}}$ are needed for various α. For instance, applying $\hat{\mathcal{D}}$ to a function $\underbrace{\mathbb{R} \to \cdots \to \mathbb{R}}_{n} \to \mathbb{R}$ requires an eta-reduction-protected

$$\hat{\mathcal{D}}_n\ f\ y_1\ \ldots\ y_n \stackrel{\triangle}{=} \mathcal{D}\ s\ 0\ f\ y_1\ \ldots\ y_n \tag{11}$$

In general, $\mathcal{D}$ should only be instantiated in a context that contains all arguments necessary to subsequently allow the post-composition of the **tg** introduced by the instantiation of $\mathcal{D}$ to immediately beta reduce to a non-function-containing value. Note that **tg** distributes over aggregates like tuples and lists, further complicating the determination of when $\mathcal{D}$ can be instantiated.

Another alternative would be to guard the returned function object against tag collision. In a programming language with opaque closures, post-composition must be implemented using a wrapper:

$$\mathbf{tg}\ \varepsilon\ (\lambda x\ .\ e) \stackrel{\triangle}{=} \lambda y\ .\ \mathbf{tg}\ \varepsilon\ ((\lambda x\ .\ e)\ y) \tag{12}$$

This wrapper can be augmented to guard against the problem we have encountered:

$$\begin{aligned}\mathbf{tg}\ \varepsilon_1\ (\lambda x\ .\ e) \stackrel{\triangle}{=} \lambda y\ .\ (\mathbf{swiz}\ \varepsilon_2\ \varepsilon_1\ (\mathbf{tg}\ \varepsilon_1\ ((\lambda x\ .\ e)\ (\mathbf{swiz}\ \varepsilon_1\ \varepsilon_2\ y))))\\ \text{where } \varepsilon_2 \text{ is fresh}\end{aligned} \tag{13}$$

Here "**swiz** ε_1 ε_2 v" substitutes ε_2 for every occurrence of ε_1 in v. In a language with opaque closures, **swiz** must operate on function objects by appropriate pre- and post-composition. This technique was used to address the present issue in the 30-Aug-2011 release of scmutils, a software package that accompanies a textbook on classical mechanics (Sussman et al., 2001), in response to an early version of this manuscript. Unfortunately the computational burden of such "swizzling" violates the complexity guarantees of Forward AD. This leaves us in the awkward position of there being *no known technique* for implementing Forward AD, with its defining complexity guarantee, and generalized to functions with higher-order outputs (including even curried functions), in a dynamic language.

We have used fresh tags to implement a form of dependent typing, where a fresh set of tangent spaces is created each time $\mathcal{D}$ is instantiated. Forward AD implementations in dynamically typed languages which support operator overloading (e.g., Scheme, Python) are susceptible to the problem we have exhibited due to the impurity of "`gensym`." It seems reasonable to speculate that static type systems (particularly those with at least some limited form of dependent typing such as existential types) may prevent this error. However, (a) current type systems prevent first-class automatic differentiation operators themselves from being defined, and (b) an intuition is a far cry from a proof. It is a current topic of research to satisfactorily define a λ-calculus based system which correctly models Forward AD (Ehrhard and Regnier, 2003; Manzyuk, 2012a,b).

Acknowledgments

This work was supported, in part, by Science Foundation Ireland Principal Investigator grant 09/IN.1/I2637 and Army Research Laboratory Cooperative Agreement Number W911NF-10-2-0060. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of SFI, ARL, or the Irish or U.S. Governments. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

References

Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, December 2003.

René Lavendhomme. *Basic Concepts of Synthetic Differential Geometry.* Kluwer Academic, 1996.

Gottfried Wilhelm Leibniz. A new method for maxima and minima as well as tangents, which is impeded neither by fractional nor irrational quantities, and a remarkable type of calculus for this. *Acta Eruditorum*, 1664.

Oleksandr Manzyuk. A simply typed λ-calculus of forward automatic differentiation. In *Mathematical Foundations of Programming Semantics Twenty-*

eighth Annual Conference, pages 259–273, Bath, UK, June 6–9 2012a. URL `http://dauns.math.tulane.edu/~mfps/mfps28proc.pdf`.

Oleksandr Manzyuk. Tangent bundles in differential λ-categories. Technical Report 1202.0411, ArXiV, 2012b. URL `http://arxiv.org/abs/1202.0411`.

Isaac Newton. De quadratura curvarum, 1704. In *Optiks*, 1704 edition. Appendix.

Jeffrey Mark Siskind and Barak A. Pearlmutter. Perturbation confusion and referential transparency: Correct functional implementation of forward-mode AD. In Andrew Butterfield, editor, *Implementation and Application of Functional Languages—17th International Workshop, IFL'05*, pages 1–9, Dublin, Ireland, September 19–21 2005. Trinity College Dublin Computer Science Department Technical Report TCD-CS-2005-60.

Jeffrey Mark Siskind and Barak A. Pearlmutter. Nesting forward-mode AD in a functional framework. *Higher-Order and Symbolic Computation*, 21(4):361–76, 2008. doi: 10.1007/s10990-008-9037-1.

Gerald Jay Sussman, Jack Wisdom, and Meinhard E. Mayer. *Structure and Interpretation of Classical Mechanics.* MIT Press, Cambridge, MA, 2001.

Robert Edwin Wengert. A simple automatic derivative evaluation program. *Comm. of the ACM*, 7(8):463–4, 1964.