Automatic differentiation From Functional Analysis to Functional Programming

Fritz Henglein
DIKU, University of Copenhagen

Reps at Sixty, Edinburgh

September 11th, 2016

Ongoing joint work with Martin Elsman (DIKU), Gabriele Keller (UNSW), Ken Friis Larsen (DIKU), Dimitrios Vytionitis (MSR Cambridge)

Automatic differentation: What?

- One-shot AD:
  - Input:
    - Procedure p implementing function $f : \mathbb{R}^m \to \mathbb{R}^n$
    - Vector $x \in \mathbb{R}^m$ (point)
    - Input vector $\Delta x \in \mathbb{R}^m$ (offset)
  - Output:
  - $f'(x)(\Delta x)$ where $f'(x) : B(\mathbb{R}^m, \mathbb{R}^n)$ is derivative of f at x.

  $B(\mathbb{R}^m, \mathbb{R}^n)$ = bounded linear functions from $\mathbb{R}^m$ to $\mathbb{R}^n$.
- Staged AD:
  Compute (code for) f' and a neighborhood of x where $f'(x')$ is derivative of (the function implemented by) p at x'.

Automatic differentiation: What for?

- Machine learning (backpropagation of constraints, ...)
- Quantitative finance (sensitivities, "Greeks")
- Atmospheric chemistry
- Breast cancer biostatistical analysis
- Computational fluid dynamics
- Chemical kinetics
- Climate and weather modeling
- Semiconductor device simulation
- Water reservoir simulation
- Mechanical engineering (design optimization)
- ...

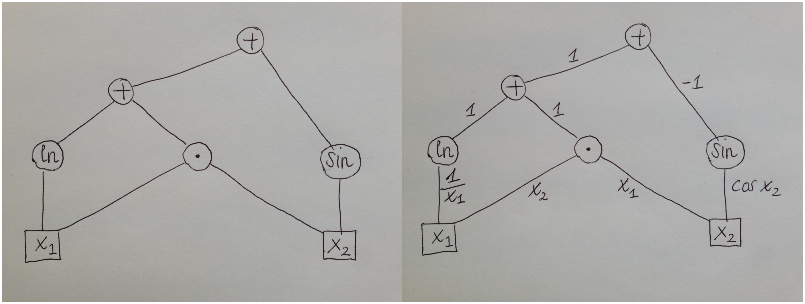
Automatic differentiation: How?

Conceptually:

1. Run p on x with uninterpreted $\mathbb{R}$-primitives, building a computation graph ($= f$ represented as data dependency dag).
2. Annotate edges with derivatives of primitives in nodes above ($= f'$).
3. Compute $y = f(x)$ by evaluating the nodes.
4. Compute $\frac{\partial y}{\partial x}$ as the sum of edge products of all paths from x to y.

Automatic differentiation: Example

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



$$\begin{aligned} \frac{\partial y}{\partial x_1} &= 1 \cdot 1 \cdot \frac{1}{x_1} + 1 \cdot 1 \cdot x_2 &= \frac{1}{x_1} + x_2 \\ \frac{\partial y}{\partial x_2} &= 1 \cdot 1 \cdot x_1 + (-1) \cdot \cos x_2 &= x_1 - \cos x_2 \end{aligned}$$

Automatic differentiation: Basic methods

- Forward-mode AD (1964):
  - Evaluation of $f'(x)(\Delta x)$ by forward (bottom-up) traversal of computation graph.
  - computation graph need not be materialized.
- Reverse-mode AD (1970):
  - Evaluation of $f(x)$ by forward traversal and $f(x)(\Delta x)$ by backward traversal;
  - computation graph ("tape") is materialized.
- Mixed-mode AD: A bit forward, a bit backward.

Jakobian

Definition (Jakobian at x)

$f'(x)$ as $n \times m$-matrix.

Compute Jakobian at x. Basic strategy:

- If $n > m$, use forward mode: For each source (input), compute reachable nodes/traverse whole graph.
- If $m >> n$, use reverse mode: For each sink (output), compute reverse reachable nodes/traverse whole graph.

Theorem (Naumann 2006)

Minimal number of elementary operations required to compute Jakobian from computation graph is NP-complete.

Observations

- AD usually focuses on *scalar* computations:

$$
\begin{array}{l}
\textbf{let } \Delta\bar{v}_4 = \Delta y \cdot 1 \textbf{ in} \\
\textbf{let } \Delta\bar{v}_3 = \Delta y \cdot 1 \textbf{ in} \\
\textbf{let } \Delta\bar{v}_2 = \Delta\bar{v}_4 \cdot 1 \textbf{ in} \\
\textbf{let } \Delta\bar{v}_1 = \Delta\bar{v}_4 \cdot 1 \textbf{ in} \\
\textbf{let } \Delta\bar{x}_2 = \Delta\bar{v}_3 \cdot (-\cos x_2) \textbf{ in} \\
\textbf{let } \Delta v_2 = x_2 \cdot \Delta x_1 + x_1 \cdot \Delta x_2 \textbf{ in} \\
\textbf{let } \Delta\bar{x}_2 = \Delta\bar{v}_2 \cdot x_1 + \Delta\bar{x}_2 \textbf{ in} \\
\textbf{let } \Delta\bar{x}_1 = \Delta\bar{v}_1 \cdot \frac{1}{x_1} + \Delta\bar{v}_2 \cdot x_2 \textbf{ in} \\
\Delta\bar{x}_1 \cdot \Delta x_1 + \Delta\bar{x}_2 \cdot \Delta x_2
\end{array}
$$

  - Obscures computation graph
  - Conflates computation graph with evaluation order
- Derivative represented as Jakobian matrix:
  - For $f : \mathbb{R}^{10000000} \to \mathbb{R}^{10000}$, 10000000×10000 matrix; entries $\mathbb{R}$-expressions with 10000000 free scalar variables
  - What if $f : \mathbb{R}^{\infty} \to \mathbb{R}$?

Fréchet derivative

Definition (Fréchet derivative)

Let V, W be Banach spaces, let $U \subseteq V$ be open, and $f : |U| \to |W|$ a function. $A \in B(V, W)$ is the Fréchet derivative of f at $x \in |V|$, written $f'(x)$, if

$$\lim_{h \to 0} \frac{\|f(x+h) - f(x) - |A|(h)\|}{\|h\|} = 0.$$

f is differentiable at x if it has a Fréchet derivative at x.

(Banach space = vector space + norm + limits)

Chain rule

Theorem (Chain rule)

If $f : U \to V$ *and* $g : V \to W$ *sufficiently differentiable then*

$$(g \circ f)' \quad = \quad (g' \circ f) \mathbin{\hat{\bullet}} f' \tag{1}$$

where

$\circ$ = *function composition;*

$\bullet$ = *linear function composition;*

$\hat{\bullet}$ *lifted linear function composition:* $(g \mathbin{\hat{\bullet}} f)(x) = g(x) \bullet f(x)$.

Bilinear functions

Definition (Bilinear function, tensor product)

$\diamond : U \times V \to W$ is bilinear if it is linear in each argument: for all x, y both $(x\diamond)$ and $(\diamond y)$ are linear maps.

Bilinear functions behave like products: They distribute over addition. Examples:

- Multiplication,
- tensor product,
- linear function composition;
- If $\diamond$ is bilinear, so is $\hat{\diamond}$.

Derivative calculus

Theorem (Linear function derivatives)

If f is linear, then $f'(x) = f$. Equivalently, $f' = K(f)$ where $K(f)(x) = f$.

Theorem (Generalized product rule)

If $\diamond$ is bilinear, then

$$(f \mathbin{\hat{\diamond}} g)'(x)(u) = (f'(x)(u) \diamond g(x)) + (f(x) \diamond g'(x)(u))$$

Equivalently,

$$\begin{aligned}(f \mathbin{\hat{\diamond}} g)'(x) &= (f'(x) \mathbin{\hat{\diamond}} K(g(x))) + (K(f(x)) \mathbin{\hat{\diamond}} g'(x)) \\ (f \mathbin{\hat{\diamond}} g)' &= (f' \mathbin{\hat{\hat{\diamond}}} (K \circ g)) + ((K \circ f) \mathbin{\hat{\hat{\diamond}}} g')\end{aligned}$$

Generalizes product rule of differentiation.

Higher-order derivatives

Using derivatives for primitive functions, the chain rule, rule for linear functions and the generalized product rule, higher-order derivatives can be derived combinatorially.

Corollary

$$
\begin{aligned}
(g \circ f)'' &= ((g' \circ f) \,\hat{\bullet}\, f')' \\
&= (g' \circ f)' \,\hat{\hat{\bullet}}\, (K \circ f') + (K \circ g' \circ f) \,\hat{\hat{\bullet}}\, f'' \\
&= ((g'' \circ f) \,\hat{\bullet}\, f') \,\hat{\hat{\bullet}}\, (K \circ f') + (K \circ g' \circ f) \,\hat{\hat{\bullet}}\, f''
\end{aligned}
$$

Observations

- Linear function representations with explicit composition:
  - Can be much more compact than (normalized) matrix representation. With sharing, linear-sized in function expression differentiated.
  - Embody opportunity for data-parallel computation
  - Optimization of f by linear algebra
  - Optimization of f-generation from p by slicing.
- Extends to towers of derivatives

Benchmarks

In progress. Goal: functions such as

The Gaussian mixture model with Wishart prior has log-posterior function $\log\left(p(\boldsymbol{x};\boldsymbol{w},\boldsymbol{\mu},\boldsymbol{\Sigma})\right) =$

$$\log\left(\prod_{i=1}^{N}\sum_{k=1}^{K} w_k \det\left(2\pi\Sigma_k\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)\right)\prod_{k=1}^{K} C(D,m)|\Sigma_k|^m \exp\left(-\frac{1}{2}\operatorname{trace}(\Sigma_k)\right)\right)$$
$$\text{s.t.} \sum_{k=1}^{K} w_k = 1 \text{ and } \Sigma_k \text{ is positive-semidefinite } \forall k \in \{1,\dots,K\} \tag{1}$$

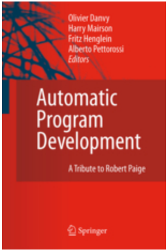
and generating code for derivatives that is sequentially competitive with hand-written (C++/C) code for derivatives and superior on GPUs.

What does this have to do with Tom?

- Tensor product
- Slicing
- ...

What does this have to do with Tom?

- ...
- Computational divided differencing: Generalization of automatic differentiation



- Tom Reps, Computational divided differencing, US Patent App. 10/161,461, 2002
- Tom Reps, Louis Rall, Computational Divided Differencing and Divided-Difference Arithmetics, in Automatic Program Development, A Tribute to Robert Paige, 2008

Happy Birthday!

From Neil Jones, Jakob Rehof and the Programming Languages Group at DIKU!