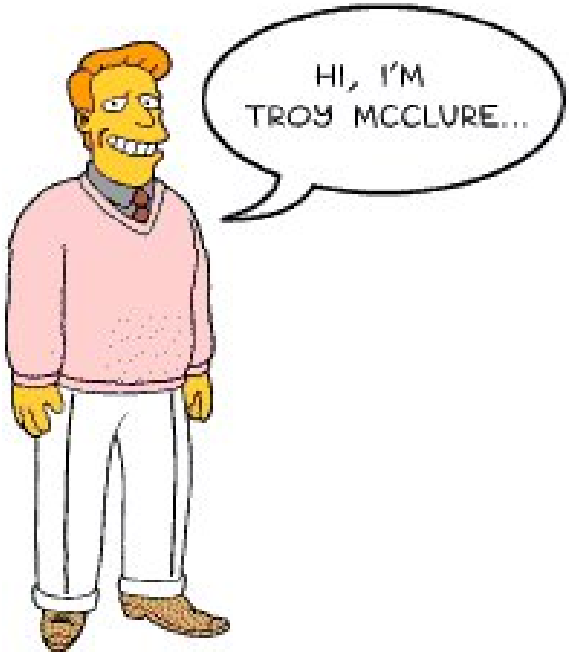

HI, I'M
TROY MCCLURE...

Automatic Differentiation: History and Headroom

Barak A. Pearlmutter

Department of Computer Science, Maynooth University, Co. Kildare, Ireland

Prof Andrei A. Markov

Lev Semenovich Pontryagin P. S. Alexandrov Andrey N. Kolmogorov

The very first computer science PhD dissertation introduced forward accumulation mode automatic differentiation.

The very first computer science PhD dissertation introduced forward accumulation mode automatic differentiation.

Wengert (1964)

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

A procedure for automatic evaluation of total/partial derivatives of arbitrary algebraic functions is presented. The technique permits computation of numerical values of derivatives without developing analytical expressions for the derivatives. **The key to the method is the decomposition of the given function**, by introduction of intermediate variables, **into a series of elementary functional steps.** A library of elementary function subroutines is provided for the automatic evaluation and differentiation of these new variables. The final step in this process produces the desired function's derivative. The main feature of this approach is its simplicity. It can be used as a quick-reaction tool where the derivation of analytical derivatives is laborious and also as a debugging tool for programs which contain derivatives.

WENGERT'S NUMERICAL METHOD FOR PARTIAL DERIVATIVES, ORBIT DETERMINATION, AND QUASILINEARIZATION

R. Bellman, H. Kagiwada and R. Kalaba

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

R. E. Bellman, H. Kagiwada, and R. E. Kalaba (1965) *Wengert's numerical method for partial derivatives, orbit determination and quasilinearization*, Communications of the ACM **8***(4)*:231–2, April 1965, doi:10.1145/363831.364886

In a recent article in the Communications of the ACM, R. Wengert suggested a technique for machine evaluation of the partial derivatives of a function given in analytical form. In solving nonlinear boundary-value problems using quasilinearization many partial derivatives must be formed analytically and then evaluated numerically. **Wengert's method** appears very attractive from the programming viewpoint and **permits the treatment of large systems** of differential equations **which might not otherwise be undertaken.**

Automatic Differentiation: a crash course

Automatic Differentiation (AD) mechanically calculates the derivatives (Leibnitz, 1664; Newton, 1704) of functions expressed as computer programs (Turing, 1936), at machine precision (Konrad Zuse, 1941, Z3; Burks, Goldstine, and von Neumann, 1946, §5.3, p14), and with complexity guarantees.

Automatic Differentiation

- Derivative of $f : \mathbb{R}^n \to \mathbb{R}^m$ is $m \times n$ "Jacobian matrix" J.
- AD, forward accumulation mode: Jv (Wengert, 1964)
- AD, reverse accumulation mode: $J^T v$ (Speelpenning, 1980)
- About a zillion other modes and tricks
- Big Iron FORTRAN-77 valve-age implementations
- Vibrant field with regular workshops, conferences, updated community portal (http://autodiff.org)

What is AD?

Automatic Differentiation
aka *Algorithmic Differentiation*
aka *Computational Differentiation*

AD Type I: A calculus for efficiently calculating derivatives of functions specified by a set of equations.

AD Type II: A way of transforming a computer program implementing a numeric function to also efficiently calculate some derivatives.

AD Type III: A computer program which automatically transforms an input computer program specifying a numeric function into one that also efficiently calculates derivatives.

Forward AD

Symmetric Truncated Taylor (1715) Expansion

$$f(x+\epsilon) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x)}{i!}\,\epsilon^i = f(x) + f'(x)\,\epsilon + \mathcal{O}(\epsilon^2)$$

Symmetric Truncated Taylor (1715) Expansion

$$f(x+\epsilon) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x)}{i!}\,\epsilon^i = f(x) + f'(x)\,\epsilon + \mathcal{O}(\epsilon^2)$$

$$f(x+\overline{x'}\,\epsilon) = f(x) + f'(x)\,\overline{x'}\,\epsilon + \mathcal{O}(\epsilon^2)$$

Symmetric Truncated Taylor (1715) Expansion

$$f(x+\epsilon) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x)}{i!}\,\epsilon^i = f(x) + f'(x)\,\epsilon + \mathcal{O}(\epsilon^2)$$

$$f(x+\overline{x'}\,\epsilon) = f(x) + f'(x)\,\overline{x'}\,\epsilon + \mathcal{O}(\epsilon^2)$$

$$f(x+\overline{x'}\,\epsilon + \mathcal{O}(\epsilon^2)) = f(x) + f'(x)\,\overline{x'}\,\epsilon + \mathcal{O}(\epsilon^2)$$

Symmetric Truncated Taylor (1715) Expansion

$$f(x+\epsilon) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x)}{i!}\,\epsilon^i = f(x) + f'(x)\,\epsilon + \mathcal{O}(\epsilon^2)$$

$$f(x+\overline{x'}\,\epsilon) = f(x) + f'(x)\,\overline{x'}\,\epsilon + \mathcal{O}(\epsilon^2)$$

$$f(x+\overline{x'}\,\epsilon + \mathcal{O}(\epsilon^2)) = f(x) + f'(x)\,\overline{x'}\,\epsilon + \mathcal{O}(\epsilon^2)$$

$$f(x \rhd \overline{x'}) = f(x) \rhd f'(x)\,\overline{x'}$$

$$f(x \triangleright \overrightarrow{x}) = f(x) \triangleright f'(x) \overrightarrow{x}$$

Won't *anyone* think of the ~~children~~ types?

$$f : \mathbb{R} \to \mathbb{R}$$

Won't *anyone* think of the ~~children~~ types?

$$f : \mathbb{R} \to \mathbb{R}$$

$$x,\ \overline{x'},\ f(x) : \mathbb{R}$$

Won't *anyone* think of the ~~children~~ types?

$$f : \mathbb{R} \to \mathbb{R}$$

$$x,\ \overrightarrow{x'},\ f(x) : \mathbb{R}$$

$$(x \rhd \overrightarrow{x'}) : \mathbb{D}\mathbb{R} \qquad \leftarrow \text{dual number (Clifford, 1873)}$$

Won't *anyone* think of the ~~children~~ types?

$$f : \mathbb{R} \to \mathbb{R}$$

$$x,\ \overline{x'},\ f(x) : \mathbb{R}$$

$$(x \rhd \overline{x'}) : \mathbb{DR} \qquad \leftarrow \text{dual number (Clifford, 1873)}$$

$$f(x \rhd \overline{x'}) = f(x) \rhd f'(x)\,\overline{x'}$$

Won't *anyone* think of the ~~children~~ types?

$$f : \mathbb{R} \to \mathbb{R}$$

$$x,\ \overline{x'},\ f(x) : \mathbb{R}$$

$$(x \rhd \overline{x'}) : \mathbb{D}\mathbb{R} \qquad \leftarrow \text{dual number (Clifford, 1873)}$$

$$f(x \rhd \overline{x'}) = f(x) \rhd f'(x)\,\overline{x'} \qquad \leftarrow \textit{type error!}$$

Won't *anyone* think of the ~~children~~ types?

$$f : \mathbb{R} \to \mathbb{R}$$

$$x,\ \overrightarrow{x},\ f(x) : \mathbb{R}$$

$$(x \rhd \overrightarrow{x}) : \mathbb{DR} \qquad \leftarrow \text{dual number (Clifford, 1873)}$$

$$f(x \rhd \overrightarrow{x}) = f(x) \rhd f'(x)\,\overrightarrow{x} \qquad \leftarrow \textit{type error!}$$

$$\overrightarrow{\mathcal{J}} : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{DR} \to \mathbb{DR})$$

$$\overrightarrow{\mathcal{J}} f(x \rhd \overrightarrow{x}) = f(x) \rhd f'(x)\,\overrightarrow{x}$$

$$\overrightarrow{\mathcal{J}} f(x \triangleright \overrightarrow{x}) = f(x) \triangleright f'(x) \overrightarrow{x}$$

Multifaceted Key to Forward AD!

$$\overrightarrow{\mathcal{J}} f(x \triangleright \overline{x'}) = f(x) \triangleright f'(x)\,\overline{x'}$$

Multifaceted Key to Forward AD!

$$\overrightarrow{\mathcal{J}} f\,(x \rhd \overrightarrow{x'}) = f(x) \rhd f'(x)\,\overrightarrow{x'}$$

Generalises beyond dual numbers (Clifford, 1873) and scalars:

$f : \mathbb{R}^n \to \mathbb{R}^m$	(multidimensional)
$x,\ \overrightarrow{x'} : \mathbb{R}^n$	(column vectors)
$x \rhd \overrightarrow{x'} : \mathbb{D}\mathbb{R}^n$	(vector of *dual numbers*)
$f'(x) : \mathbb{R}^{m \times n}$	(Jacobian matrix, **J**)
$\overrightarrow{\mathcal{J}} : (\mathbb{R}^n \to \mathbb{R}^m) \to (\mathbb{D}\mathbb{R}^n \to \mathbb{D}\mathbb{R}^m)$	(Forward AD transform)

Multifaceted Key to Forward AD!

$$\overrightarrow{\mathcal{J}} f(x \rhd \overrightarrow{x'}) = f(x) \rhd f'(x)\, \overrightarrow{x'}$$

Generalises beyond dual numbers (Clifford, 1873) and scalars:

$f : \mathbb{R}^n \to \mathbb{R}^m$	(multidimensional)
$x,\ \overrightarrow{x'} : \mathbb{R}^n$	(column vectors)
$x \rhd \overrightarrow{x'} : \mathbb{DR}^n$	(vector of *dual numbers*)
$f'(x) : \mathbb{R}^{m \times n}$	(Jacobian matrix, **J**)
$\overrightarrow{\mathcal{J}} : (\mathbb{R}^n \to \mathbb{R}^m) \to (\mathbb{DR}^n \to \mathbb{DR}^m)$	(Forward AD transform)

1. Compositional: $\overrightarrow{\mathcal{J}}(f \circ g) = \overrightarrow{\mathcal{J}} f \circ \overrightarrow{\mathcal{J}} g$
2. How to "lift" when f is a primop (elt of numeric basis)
3. What such "lifting" delivers when f is a defined function

Example: Application of $\overrightarrow{\mathcal{J}}$ to a Primop

$$v := \sin u$$

$$\Rightarrow \boxed{\overrightarrow{\mathcal{J}}} \Rightarrow$$

Example: Application of $\overrightarrow{\mathcal{J}}$ to a Primop

$$v := \sin u$$

$$\Rightarrow \boxed{\overrightarrow{\mathcal{J}}} \Rightarrow$$

$$\overrightarrow{v} := \overrightarrow{\mathcal{J}} \sin \overrightarrow{u}$$

Example: Application of $\overrightarrow{\mathcal{J}}$ to a Primop

$$v := \sin u$$

$\Rightarrow \boxed{\overrightarrow{\mathcal{J}}} \Rightarrow$

$$\overrightarrow{v} := \overrightarrow{\mathcal{J}} \sin \overrightarrow{u}$$

$\Rightarrow$ inline & destructure $\Rightarrow$

$$v \rhd \overrightarrow{v}' := \overrightarrow{\mathcal{J}} \sin (u \rhd \overrightarrow{u}')$$

Example: Application of $\overrightarrow{\mathcal{J}}$ to a Primop

$$v := \sin u$$

$\Rightarrow \boxed{\overrightarrow{\mathcal{J}}} \Rightarrow$

$$\overrightarrow{v} := \overrightarrow{\mathcal{J}} \sin \overrightarrow{u}$$

$\Rightarrow$ inline & destructure $\Rightarrow$

$$v \rhd \overrightarrow{v'} := \overrightarrow{\mathcal{J}} \sin (u \rhd \overrightarrow{u'})$$

$$v \rhd \overrightarrow{v'} := \sin u \rhd (\cos u) * \overrightarrow{u'}$$

Example: Application of $\overrightarrow{\mathcal{J}}$ to a Primop

$$v := \sin u$$

$\Rightarrow \boxed{\overrightarrow{\mathcal{J}}} \Rightarrow$

$$\overrightarrow{v} := \overrightarrow{\mathcal{J}} \sin \overrightarrow{u}$$

$\Rightarrow$ inline & destructure $\Rightarrow$

$$v \rhd \overrightarrow{v'} := \overrightarrow{\mathcal{J}} \sin (u \rhd \overrightarrow{u'})$$

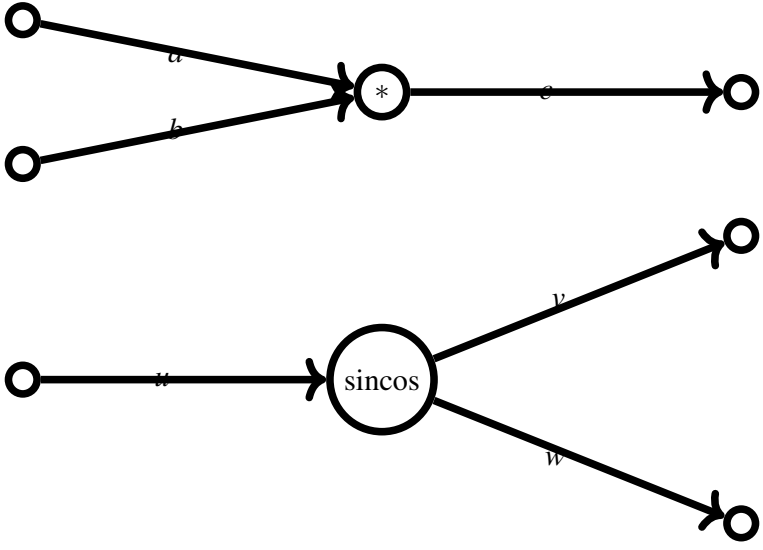
$$v \rhd \overrightarrow{v'} := \sin u \rhd (\cos u) * \overrightarrow{u'}$$

$$v := \sin u$$
$$\overrightarrow{v'} := (\cos u) * \overrightarrow{u'}$$

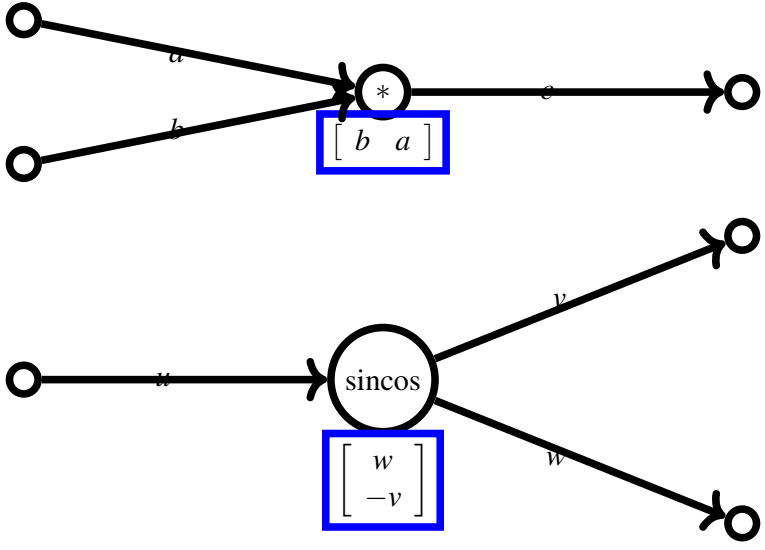
Simple Code

$$c := a * b$$
$$(v, w) := \text{sincos } u$$

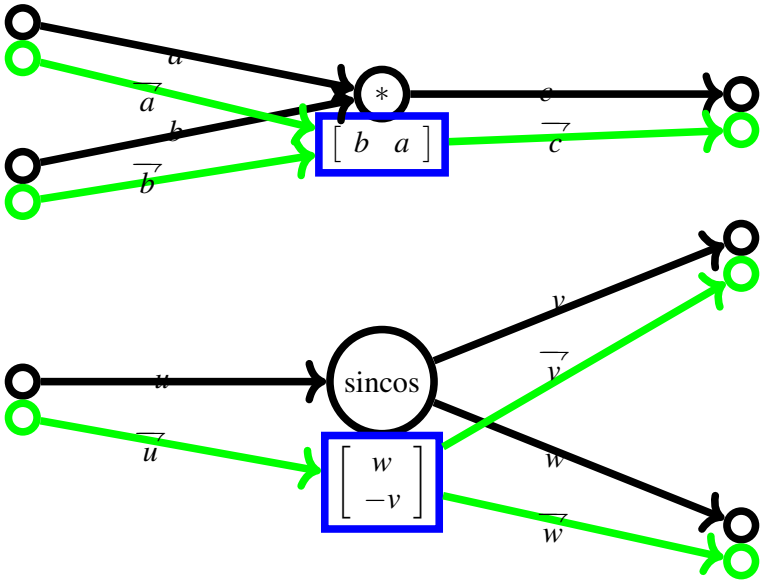
Data Flow Graph



Data Flow Graph



Data Flow Graph



Transform Graph as Netlist, *i.e.*, Code

$$c := a * b \qquad \Rightarrow \boxed{\overrightarrow{\mathcal{J}}} \Rightarrow \qquad c := a * b$$

$$(v, w) := \text{sincos } u \qquad \overline{c}' := a * \overline{b}' + b * \overline{a}'$$

$$(v, w) := \text{sincos } u$$

$$\overline{v}' := w * \overline{u}'$$

$$\overline{w}' := -v * \overline{u}'$$

AKA

- Forward Automatic Differentiation
- Forward Propagation
- Directional Derivative
- Push Forward
- Perturbation Analysis

Reverse AD

(aka backprop)

In the 1970s, tools for automated generation of adjoint codes (aka reverse accumulation mode automatic differentiation, aka backpropagation) were developed.

Type I: Geniuses transforming mathematical systems
(Gauss; Feynman (1939); Rozonoer and Pontryagin (1959))

Type II: Manual transformation of computational processes
(Bryson (1962); Werbos (1974); Le Cun (1985); Rumelhart, Hinton, and Williams (1986))

Type III: Computer programs transform other computer programs
(Speelpenning (1980); LUSH; TAPENADE)

Type IV: First-class AD operators; closure
(STALIN∇; R^{6}RS-AD; AUTOGRAD; DIFFSHARP)

Bert Speelpenning

UIUCDCS-R-80-1002

UILU-ENG 80 1702
COO-2383-0063

COMPILING FAST PARTIAL DERIVATIVES OF FUNCTIONS GIVEN BY ALGORITHMS

by

Bert Speelpenning

January 1980

by arbitrary algorithms.

What is needed is a system as sketched in Figure 1.

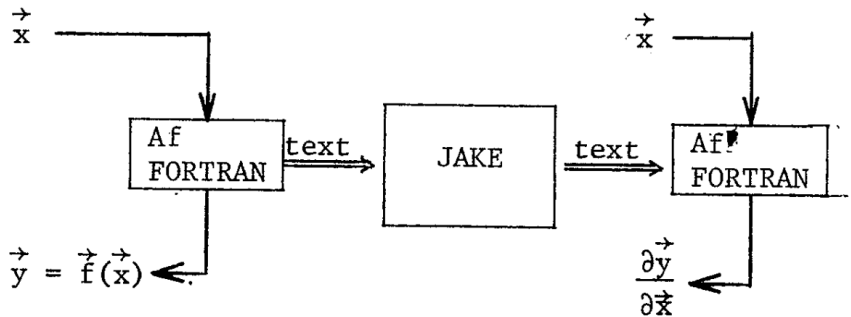


Figure 1. Use of Jake

Such a system will accept the text of an algorithm Af, embodying $y=f(x)$

1.2. Major Results of this Research

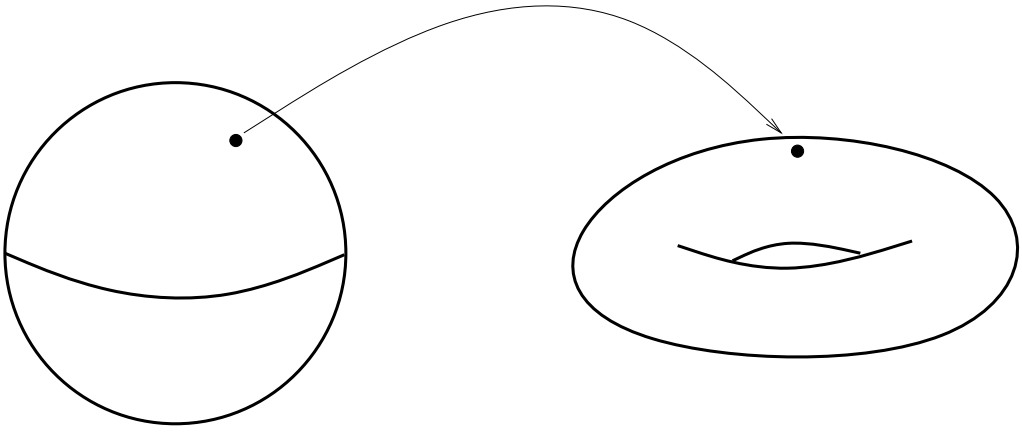
A full solution to the problem of compiling fast gradients has been obtained. For the problem of compiling fast Jacobians of arbitrary shape a partial solution has been found. This thesis describes a method and its implementation capable of producing algorithms Af′ that compute the gradient of a function $f(x_1,\ldots,x_n)$ in an amount of time equivalent to a constant number of function evaluations independent of n. The space requirements of the algorithm Af′ are modest.

specialization occurring within Computer Science. The separation of Numerical Analysis from Software is virtually complete and few people care to bridge the gap between the areas. In Numerical Analysis, the notion of programs that produce programs rather than numbers is largely absent. For most numerical analysts the FORTRAN compiler is completely transparent, as if Created on the same day as the computer. There is little awareness of language processing as a software writing tool in the sense of the products we have come to expect from places like Bell Labs. Notable exceptions include user languages for physical modeling and for statistical computations. Conversely, people involved in writing software tools may have a tendency to write only such software tools that aid in the writing of other software tools, and although this opens fascinating avenues of auto-catalysis, the real usefulness of these tools must ultimately come from application to outside areas. Tools are means to an end, not ends in themselves. What seems required is not merely cooperation between software people and numerical analysts but efforts by people with a certain minimal understanding and interest in both areas. The effort invested in such hetero-catalysis could pay off very handsomely.

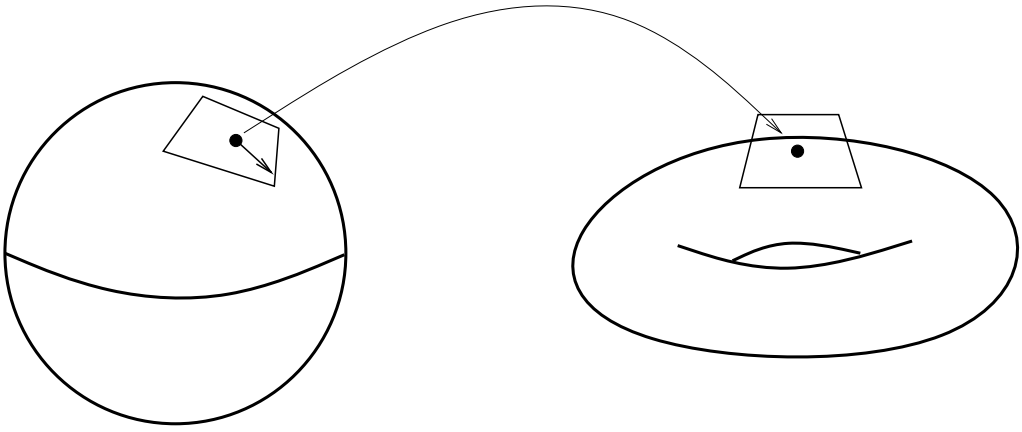
Differential Geometry

(digression)

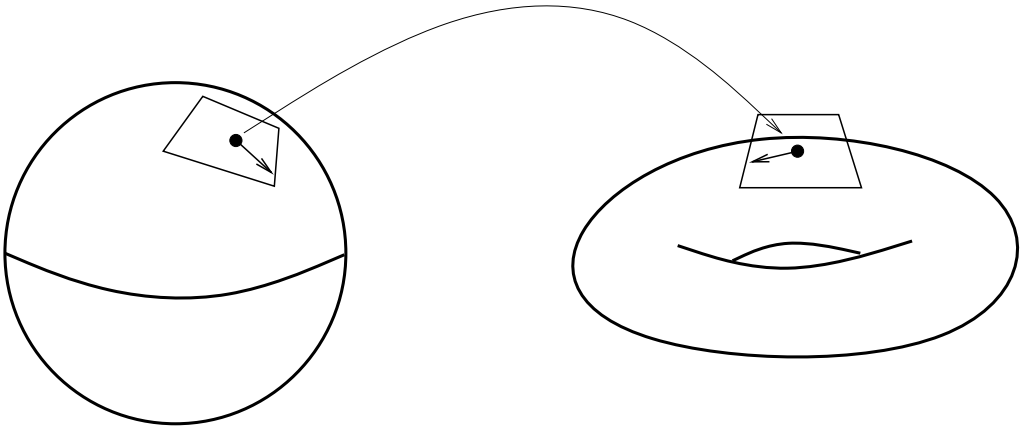
Tangent Space



Tangent Space



Tangent Space



Cotangent Space

$$\overleftarrow{a} : \overleftarrow{\alpha_a}$$

$$\overleftarrow{\alpha_a} = \overrightarrow{\alpha'_a} \xrightarrow{\text{linear}} \mathbb{R}$$

$$(\bullet) : \overleftarrow{\alpha_a} \to \overrightarrow{\alpha'_a} \to \mathbb{R}$$

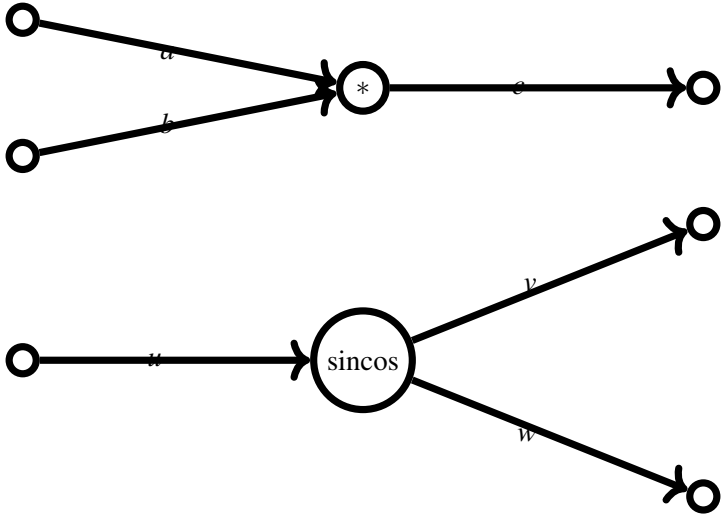
Gradients & Reverse AD are *Dual* to Perturbations & Forward AD

$$\overleftarrow{a} \bullet \overrightarrow{a'} = \overleftarrow{b} \bullet \overrightarrow{b'} \qquad (\bullet) : \overleftarrow{\alpha} \to \overrightarrow{\alpha'} \to \mathbb{R}$$

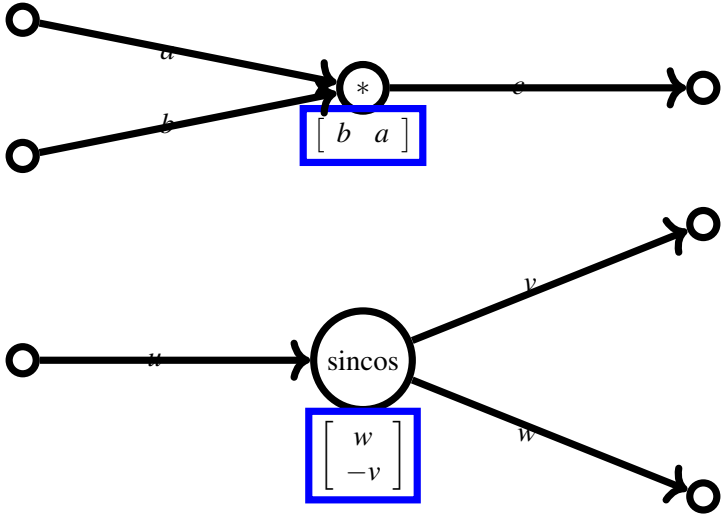
where we let

$$\begin{aligned} b &= f\ a & f &: \alpha \to \beta \\ (b \rhd \overrightarrow{b'}) &= \overrightarrow{\mathcal{J}}\ f\ (a \rhd \overrightarrow{a'}) & \overrightarrow{\mathcal{J}}\ f &: \overrightarrow{\alpha} \to \overrightarrow{\beta} \\ (b, \bar{f}) &= \overleftarrow{\mathcal{J}}\ f\ a & \overleftarrow{\mathcal{J}}\ f &: \alpha \to (\beta \times (\overleftarrow{\beta} \to \overleftarrow{\alpha})) \\ \overleftarrow{a} &= \bar{f}\ \overleftarrow{b} & \bar{f} &: \overleftarrow{\beta} \to \overleftarrow{\alpha} \end{aligned}$$

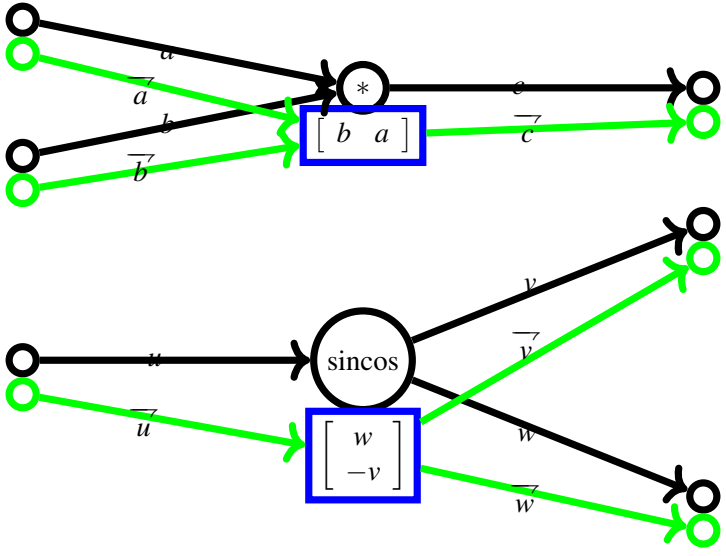
Data Flow Graph



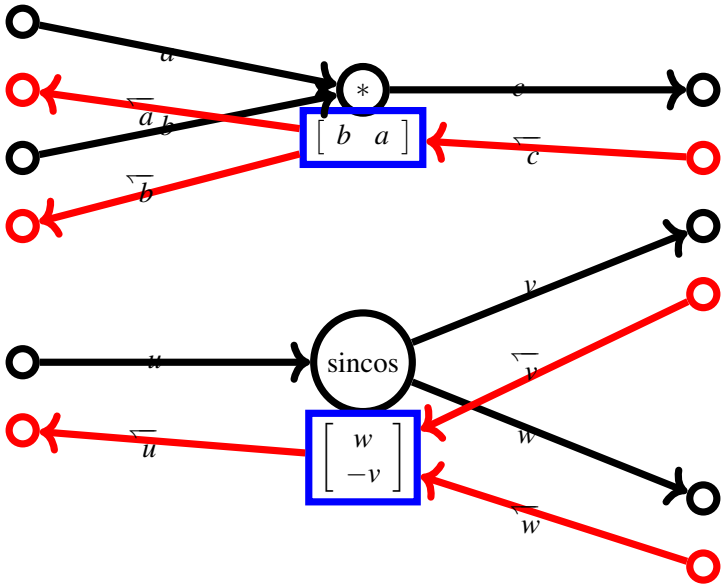
Data Flow Graph



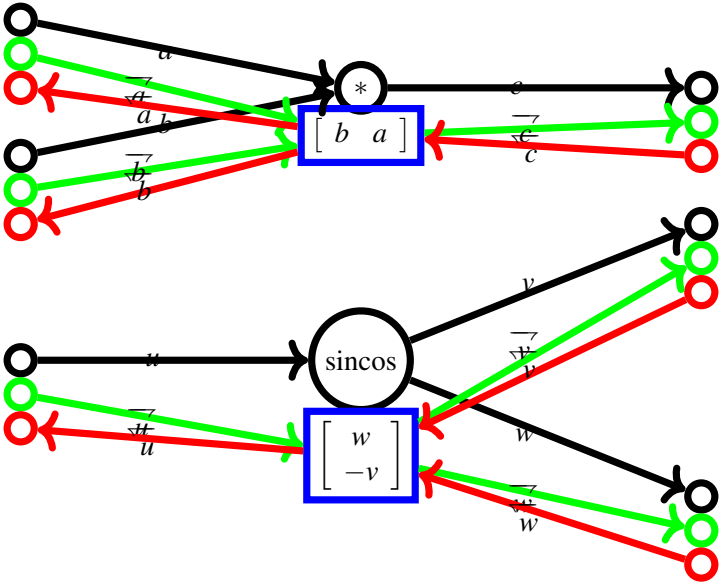
Data Flow Graph



Data Flow Graph



Data Flow Graph



$$
\begin{aligned}
c &:= a * b \\
(v, w) &:= \operatorname{sincos} u
\end{aligned}
\qquad \Rightarrow \boxed{\overrightarrow{\mathcal{F}}} \Rightarrow \qquad
\begin{aligned}
c &:= a * b \\
\overline{c}' &:= a * \overline{b}' + b * \overline{a}' \\
(v, w) &:= \operatorname{sincos} u \\
\overline{v}' &:= w * \overline{u}' \\
\overline{w}' &:= -v * \overline{u}'
\end{aligned}
$$

$$
\begin{aligned}
c &:= a * b \\
(v, w) &:= \operatorname{sincos} u
\end{aligned}
\quad \Rightarrow \boxed{\overrightarrow{\mathcal{J}}} \Rightarrow \quad
\begin{aligned}
c &:= a * b \\
\overrightarrow{c} &:= a * \overrightarrow{b} + b * \overrightarrow{a} \\
(v, w) &:= \operatorname{sincos} u \\
\overrightarrow{v} &:= w * \overrightarrow{u} \\
\overrightarrow{w} &:= -v * \overrightarrow{u}
\end{aligned}
$$

$$
\Rightarrow \boxed{\overleftarrow{\mathcal{J}}} \Rightarrow \quad
\begin{aligned}
c &:= a * b \\
(v, w) &:= \operatorname{sincos} u \\
&\vdots \\
\overleftarrow{u} &:= w * \overleftarrow{v} - v * \overleftarrow{w} \\
\overleftarrow{a} &:= b * \overleftarrow{c} \\
\overleftarrow{b} &:= a * \overleftarrow{c}
\end{aligned}
$$

Generalise: All Types Are Manifolds

- can be disconnected (e.g., union type)
- components can have varying dimensionality (e.g., list $\mathbb{R}$)
- components can be zero dimensional (e.g., **bool**, enum, $\mathbb{Z}$), in which case tangent space is zero dimensional (**void**)

primary $\overleftarrow{\mathcal{I}}$ technical difficulty:

fanout

even today, our tools for high-performance numeric computations do not support automatic differentiation as a first-class citizen.

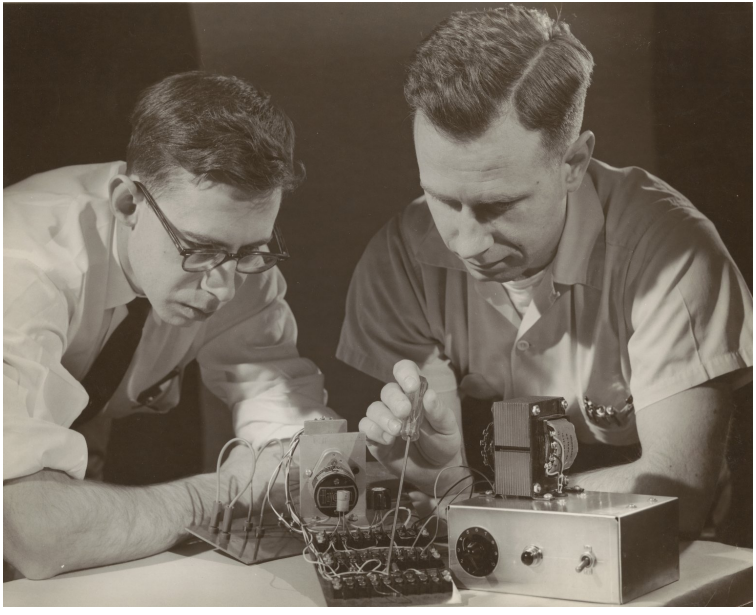
even today, our tools for high-performance numeric computations do not support automatic differentiation as a first-class citizen.

Dominant AD technology for high-performance systems: preprocessors.

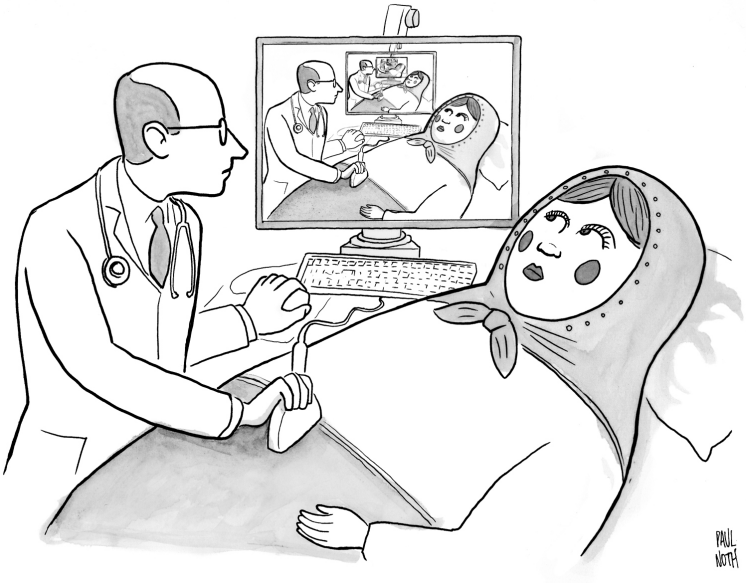
even today, our tools for high-performance numeric computations do not support automatic differentiation as a first-class citizen.

Dominant AD technology for high-performance systems: preprocessors.

- very hard to apply in a nested fashion
- caller-derives API impedes modularity
- brittle and idiosyncratic.



Rosenblatt Wightman



nesting

Uses of Nesting

- Differential objective:

$$\min_w \sum_i \|f(x_i; w) - y_i\|^2 + \| (d/dx) f(x; w)|_{x=x_i} - z_i\|^2$$

- Multilevel optimization (GANs, learn-to-learn, etc. So hot!)
- Optimizing game's rules so rational players exhibit desired behaviour
- Design optimization of "smart" devices, or devices involving PDEs
- Hyperparameter optimization
- Sensitivity/robustness analysis of processes involving AD

Generalise

Generalise $\overrightarrow{\mathcal{J}}$, $\overleftarrow{\mathcal{J}}$ to apply to *all* functions ...

$$\overrightarrow{\mathcal{J}} : (\alpha \to \beta) \to (\overrightarrow{\alpha} \to \overrightarrow{\beta})$$
$$\overleftarrow{\mathcal{J}} : (\alpha \to \beta) \to (\alpha \to (\beta \times (\overleftarrow{\beta} \to \overleftarrow{\alpha})))$$

... to *all* objects ...

$$\overrightarrow{\mathcal{J}} : \alpha \to \overrightarrow{\alpha}$$
$$\overrightarrow{\alpha \to \beta} = \overrightarrow{\alpha} \to \overrightarrow{\beta}$$
$$\overleftarrow{\mathcal{J}} : \alpha \to \overleftarrow{\alpha}$$

Technicalities!

- Tangent space is usually isomorphic to "$\mathbb{R}$ holes" in primal space, since $\mathbb{R}$ is our only non-zero-dimensional primitive type.

  But not always (function types).

- Cotangent space is usually isomorphic to tangent space.

  But not always (function types).

- Due to issues related to this, parts of reverse mode must be "lazy" even if primal & forward AD computations are "eager".

Functions Diff. Geom. Handles

- arithmetic functions
- functions over discrete spaces
- functions over disconnected manifolds of differing dimensionality
- higher-order functions over concrete linear functions
- higher-order functions like `map` and compose (`∘`)
- higher-order functions like `numeric-iterate-to-fixedpoint` (Feynman, 1939; Pineda, 1987; Almeida, 1987)
- higher-order functions like $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$

delicate dance

fielded systems with first-class AD:

slow

rough edges

headroom for acceleration

research prototype compiler

Benchmarks

		particle				saddle				probabilistic-lambda-calculus		probabilistic-prolog		backprop		
		FF	FR	RF	RR	FF	FR	RF	RR	F	R	F	R	F	Fv	R
VLAD	STALIN∇	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	■	1.00
FORTRAN	ADIFOR	2.05	■	■	■	5.44	■	■	■	■	■	■	■	15.51	3.35	■
	TAPENADE	5.51	■	■	■	8.09	■	■	■	■	■	■	■	14.97	5.97	6.86
C	ADIC	■	■	■	■	■	■	■	■	■	■	■	■	22.75	5.61	■
C++	ADOL–C	■	■	■	■	■	■	■	■	■	■	■	■	12.16	5.79	32.77
	CPPAD	■	■	■	■	■	■	■	■	■	■	■	■	54.74	■	29.24
	FADBAD++	93.32	■	■	■	60.67	■	■	■	■	■	■	■	132.31	46.01	60.71
ML	MLTON	78.13	111.27	45.95	32.57	114.07	146.28	12.27	10.58	129.11	114.88	848.45	507.21	95.20	■	39.90
	OCAML	217.03	415.64	352.06	261.38	291.26	407.67	42.39	50.21	249.40	499.43	1260.83	1542.47	202.01	■	156.93
	SML/NJ	153.01	226.84	270.63	192.13	271.84	299.76	25.66	23.89	234.62	258.53	2505.59	1501.17	181.93	■	102.89
HASKELL	GHC	209.44	■	■	■	247.57	■	■	■	■	■	■	■	■	■	■
SCHEME	BIGLOO	627.78	855.70	275.63	187.39	1004.85	1076.73	105.24	89.23	983.12	1016.50	12832.92	7918.21	743.26	■	360.07
	CHICKEN	1453.06	2501.07	821.37	1360.00	2276.69	2964.02	225.73	252.87	2324.54	3040.44	44891.04	24634.44	1626.73	■	1125.24
	GAMBIT	578.94	879.39	356.47	260.98	958.73	1112.70	89.99	89.23	1033.46	1107.26	26077.48	14262.70	671.54	■	379.63
	IKARUS	266.54	386.21	158.63	116.85	424.75	527.57	41.27	42.34	497.48	517.89	8474.57	4845.10	279.59	■	165.16
	LARCENY	964.18	1308.68	360.68	272.96	1565.53	1508.39	126.44	112.82	1658.27	1606.44	25411.62	14386.61	1203.34	■	511.54
	MIT SCHEME	2025.23	3074.30	790.99	609.63	3501.21	3896.88	315.17	295.67	4130.88	3817.57	87772.39	49814.12	2446.33	■	1113.09
	MZC	1243.08	1944.00	740.31	557.45	2135.92	2434.05	194.49	187.53	2294.93	2346.13	57472.76	31784.38	1318.60	■	754.47
	MZSCHEME	1309.82	1926.77	712.97	555.28	2371.35	2690.64	224.61	219.29	2721.35	2625.21	60269.37	33135.06	1364.14	■	772.10
	SCHEME->C	582.20	743.00	270.83	208.38	910.19	913.66	82.93	69.87	811.37	803.22	10605.32	5935.56	597.67	■	280.93
	SCMUTILS	4462.83	■	■	■	7651.69	■	■	■	7699.14	■	83656.17	■	5889.26	■	■
	STALIN	364.08	547.73	399.39	295.00	543.68	690.64	63.96	52.93	956.47	1994.44	15048.42	16939.28	435.82	■	281.27

Comparative benchmark results for the `particle` and `saddle` examples (Siskind and Pearlmutter, 2008a), the `probabilistic-lambda-calculus` and `probabilistic-prolog` examples (Siskind, 2008) and an implementation of backpropagation in neural networks using AD. Column labels are for AD modes and nesting: F for forward, Fv for forward-vector aka stacked tangents, RF for reverse-over-forward, etc. All run times normalized relative to a unit run time for STALIN∇ on the corresponding example except that run times for `backprop-Fv` are normalized relative to a unit run time for STALIN∇ on `backprop-F`. Pre-existing AD tools are named in blue, others are custom implementations. Key: ■ not implemented but could implement, including FORTRAN, C, and C++; ■ not implemented in pre-existing AD tool; ■ problematic to implement. All code available at http://www.bcl.hamilton.ie/~qobi/ad2016-benchmarks/.

Benchmarks

COME TO JEFF SISKIND'S TALK

		particle				probabilistic-lambda-calculus				probabilistic-prolog		backprop		
		FF	FR	RF	RR	FF	FR	RF	RR	F	R	F	Fv	R
VLAD	STALIN∇	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	■	1.00
FORTRAN	ADIFOR	2.05	■	■	■	1.41	■	■	■	■	■	■	■	■
	TAPENADE	5.51	■	■	■	2.09	■	■	■	■	■	■	■	■
C	ADIC	■	■	■	■	■	■	■	■	■	■	■	■	■
C++	ADOL-C	■	■	■	■	■	■	■	■	■	■	■	■	■
	CPPAD	■	■	■	■	■	■	■	■	■	■	■	■	■
	FADBAD++	93.32	■	■	■	60.67	■	■	■	■	■	■	■	■
ML	MLTON	78.13	111.27	45.95	32.57	14.07	146.28	12.68	10.68	29.11	114.88	848.45	507.21	95.20
	OCAML	217.03	415.64	352.06	261.38	11.26	407.67	42.50	50.21	49.40	499.43	1260.83	1542.47	202.01
	SML/NJ	153.01	226.84	270.63	192.13	11.84	295.76	25.23	23.89	54.42	258.53	2505.59	1501.17	181.93
HASKELL	GHC	209.44	■	■	■	77.57	■	■	■	■	■	■	■	■
SCHEME	BIGLOO	627.78	855.70	275.63	187	94.85	876.73	101	89.23	83.12	1016.50	12832.92	7918.21	743.26	■	360.07
	CHICKEN	1453.06	2501.07	821.37	1360.0	276.69	2964.02	225.73	252.87	2324.54	3040.44	44891.04	24634.44	1626.73	■	1125.24
	GAMBIT	578.94	879.39	356.47	260.98	958.73	1112.70	89.99	89.23	1033.46	1107.26	26077.48	14262.70	671.54	■	379.63
	IKARUS	266.54	386.21	158.63	116.85	424.75	527.57	41.27	42.34	497.48	517.89	8474.57	4845.10	279.59	■	165.16
	LARCENY	964.1	130	68	272.9	1	53	508.39	6.44	82	658.	06.44	541	62	1386.61	1203.34	■	511.54
	MIT SCHEME	2025.2	2074.30	99	09.63	3	21	3896.88	5.17	2	7	130.	3	57	8771	49814.12	2446.33	■	1113.09
	MZC	1243.08	00	31	2	2	2434.05	4.49	187	294.	2	13	57472	784.38	1318.60	■	754.47
	MZSCHEME	1309.2	192	97	2.97	555.	2	35	90.64	4.61	219.2	721.	2	6.21	60209.37	35.06	1364.14	■	772.10
	SCHEME->C	582.1	74	00	1.83	208	8	19	66	2.93	69.87	811	803.22	106	32	5935.56	597.67	■	280.93
	SCMUTILS	4462.83	■	■	■	7651.69	■	■	■	7699.14	■	83656.17	■	■	5889.26	■	■
	STALIN	364.08	547.73	399.39	295.00	543.68	690.64	63.96	52.93	956.47	1994.44	15048.42	16939.28	435.82	■	281.27

Comparative benchmark results for the particle and puddle examples (Siskind and Pearlmutter, 2008a), the probabilistic-lambda-calculus and probabilistic-prolog examples (Siskind 2008) and an implementation of backpropagation in neural networks using AD. Column labels are for modes and timing: F forward, Fv forward-vector aka stacked tangents, RF for reverse-over-forward, etc. All run times normalized relative to a unit run time for STALIN∇ the corresponding example except that run times for backprop-Fv are normalized relative to a unit run time for STALIN∇ backprop-F. Re-executing AD tools' names in blue, others are custom implementations. Key: ■ not implemented but could implement, including FORTRAN, C, and C++; ■ not implemented in pre-existing AD tool; ■ problematic to implement. All code available at http://www.bcl.hamilton.ie/~qobi/ad2016-benchmarks/.

Functional AD: A Usable System

DiffSharp is a functional automatic differentiation (AD) library in F# for the multiplatform **.NET** framework.

```
let (y, dydx) = grad' f x
```

https://diffsharp.github.io/DiffSharp/

https://github.com/DiffSharp/DiffSharp

DiffSharp-using library shows how nested AD allows succinct implementations of, e.g., optimization of hyperparameters:
https://hypelib.github.io/Hype/

Atılım Güneş Baydin

history of automatic differentiation and of backpropagation

history of automatic differentiation and of backpropagation ✓

history of automatic differentiation and of backpropagation ✓

embellishments and variants (backpropagation through time, RTRL, etc)

history of automatic differentiation and of backpropagation ✓

embellishments and variants (backpropagation through time, RTRL, etc)

(Pearlmutter, 1994; Williams and Zipser, 1989; Simard et al., 1992)

```
backProp E f w x = ∇ (w↦E(f x)) w

hessianVector f x v = dd (r↦∇ f (x+r*v)) 0

RTRL f w x E =
  map (i↦(dd (w↦E(f w x)) w (e i))) (ι(dim w))

tangentProp E r f x =
  ∇ (w↦E(f x) + sqr(len(dd (θ↦f(r θ x)) 0)))
      w

hyperOpt E R train1 train2 =
  argmin (h↦
   let w0 =
    argmin (w↦R h w + sum(map (t↦E w t) train1))
    in sum(map (t↦E w0 t) train2)
```

Method of Temporal Differences

$$E(w) = \cdots + \lambda \sum_{t=0}^{t_f-2} \|y(t;w) - y(t+1;w)\|^2 + \cdots \qquad \text{TD}(\lambda)$$

Method of Temporal Differences

$$E(w) = \cdots + \lambda \sum_{t=0}^{t_f-2} \|y(t;w) - y(t+1;w)\|^2 + \cdots \qquad \text{TD}(\lambda)$$

$$\nabla \; E \; w \qquad ?$$

Method of Temporal Differences

$$E(w) = \cdots + \lambda \sum_{t=0}^{t_f-2} \|y(t;w) - y(t+1;w)\|^2 + \cdots \qquad \text{TD}(\lambda)$$

$$\nabla\ E\ w \qquad ?$$

$$\nabla\ (w \mapsto \|y(t;w) - y(t+1;w)\|^2)\ w \qquad ?$$

Method of Temporal Differences

$$E(w) = \cdots + \lambda \sum_{t=0}^{t_f-2} \|y(t;w) - y(t+1;w)\|^2 + \cdots \qquad \text{TD}(\lambda)$$

$$\nabla\ E\ w \qquad ?$$

$$\nabla\ (w \mapsto \|y(t;w) - y(t+1;w)\|^2)\ w \qquad ?$$

No!

Method of Temporal Differences

$$E(w) = \cdots + \lambda \sum_{t=0}^{t_f-2} \|y(t;w) - y(t+1;w)\|^2 + \cdots \qquad \text{TD}(\lambda)$$

$$\nabla\ E\ w \qquad ?$$

$$\nabla\ (w \mapsto \|y(t;w) - y(t+1;w)\|^2)\ w \qquad ?$$

No!

$$\text{let } v = w \text{ in } \nabla\ (w \mapsto \|y(t;w) - y(t+1;v)\|^2)\ w$$

Hooks

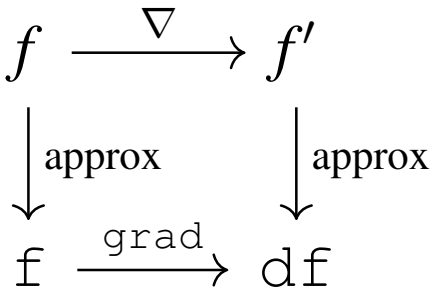
- Do you know what *Checkpoint reverse* is? *Cross-country optimization*?
- Did you know that computing $\partial^n f(x_1, \ldots, x_n)/\partial x_1 \cdots \partial x_n$ is #P-complete?
- Have you heard of Tapenade? FadBad++? ADIFOR/ADIC? Adolc? Stalin∇? ADiMat? DiffSharp? autograd? Haskell ad? http://autodiff.org?

Theoretical Frontier of AD

my idiosyncratic ravings

- Preallocation
- Not-so-simple derivatives (e.g., input vs feature space, natural gradient)
- Storage reduction by clever re-computation
- AD-enabled JIT Compiler
- Nice λ-Calculus Formulation (Correctness Proofs)
- Convergent Loops — Detailed Pragmatics
- Tropical Tangent/Co-Tangent Algebras for HMMs, etc
- Efficient $\nabla(x \mapsto \cdots \sum \cdots)$
- Derivatives and Approximation Do Not Commute

Does Not Commute! Does Not Commute!



Does Not Commute! Does Not Commute!

$$\begin{array}{ccc} f & \xrightarrow{\nabla} & f' \\ \downarrow \text{approx} & & \downarrow \text{approx} \\ \texttt{f} & \xrightarrow{\texttt{grad}} & \texttt{df} \end{array}$$


YOUR COMMUTE OFFICIALLY MAKES YOU MISERABLE

Conclusions

- AD is ancient.
- AD is in its infancy
- “Manual” AD is bug-ridden and scales poorly.
- Existing AD tools are fantastic when they match your needs.
- Better (more general, faster) tools are on the horizon.

Conclusions

- AD is ancient.
- AD is in its infancy
- “Manual” AD is bug-ridden and scales poorly.
- Existing AD tools are fantastic when they match your needs.
- Better (more general, faster) tools are on the horizon.
  If we only had the resources to build them...

References I

Luis B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In Maureen Caudill and Charles Butler, editors, *IEEE First International Conference on Neural Networks*, volume 2, pages 609–18, San Diego, CA, June 21–24 1987.

Atılım Güneş Baydin and Barak A. Pearlmutter. Automatic differentiation of algorithms for machine learning. Technical Report arXiv:1404.7456, April 28 2014. Also in Proceedings of the AutoML Workshop at the International Conference on Machine Learning (ICML), Beijing, China, June 21–26, 2014.

Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. Technical Report arXiv:1502.05767, 2015a.

Atılım Güneş Baydin, Barak A. Pearlmutter, and Jeffrey Mark Siskind. DiffSharp: Automatic differentiation library. Technical Report arXiv:1511.07727, 2015b.

Atılım Güneş Baydin, Barak A. Pearlmutter, and Jeffrey Mark Siskind. DiffSharp: An AD library for .NET languages. Technical Report arXiv:1611.03423, September 2016. Extended abstract presented at the AD 2016 Conference, Oxford UK.

References II

R. E. Bellman, H. Kagiwada, and R. E. Kalaba. Wengert's numerical method for partial derivatives, orbit determination and quasilinearization. *Comm. of the ACM*, 8(4):231–2, April 1965. doi: 10.1145/363831.364886.

Arthur E. Bryson, Jr. A steepest ascent method for solving optimum programming problems. *Journal of Applied Mechanics*, 29(2):247, 1962.

Arthur W. Burks, Herman H. Goldstine, and John von Neumann. Preliminary discussion of the logical design of an electronic computing instrument. Technical report, Report to the U.S. Army Ordnance Department, 1946. URL https://library.ias.edu/files/Prelim_Disc_Logical_Design.pdf.

William Kingdon Clifford. Preliminary sketch of bi-quaternions. *Proceedings of the London Mathematical Society*, 4:381–95, 1873.

Richard Phillips Feynman. Forces in molecules. *Physical Review*, 56(4):340–3, August 1939. doi: 10.1103/PhysRev.56.340.

Yann Le Cun. Une procédure d'apprentissage pour réseau à seuil assymétrique. In *Cognitiva 85: A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, pages 599–604, Paris 1985, 1985. CESTA, Paris.

References III

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

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

Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–60, 1994. doi: 10.1162/neco.1994.6.1.147.

Barak A. Pearlmutter and Jeffrey Mark Siskind. Lazy multivariate higher-order forward-mode AD. In *Proc of the 2007 Symposium on Principles of Programming Languages*, pages 155–60, Nice, France, January 2007. doi: 10.1145/1190215.1190242.

Fernando Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 19(59):2229–32, 1987.

L. I. Rozonoer and Lev Semenovich Pontryagin. Maximum principle in the theory of optimal systems I. *Automation Remote Control*, 20:1288–302, 1959.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–6, 1986.

References IV

Patrice Simard, Bernard Victorri, Yann LeCun, and John Denker. Tangent prop—a formalism for specifying selected invariances in an adaptive network. In *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, 1992.

Jeffrey Mark Siskind. AD for probabilistic programming. NIPS 2008 workshop on Probabilistic Programming: Universal Languages and Inference; systems; and applications, 2008.

Jeffrey Mark Siskind and Barak A. Pearlmutter. First-class nonstandard interpretations by opening closures. In *Proceedings of the 2007 Symposium on Principles of Programming Languages*, pages 71–6, Nice, France, January 2007. doi: 10.1145/1190216.1190230.

Jeffrey Mark Siskind and Barak A. Pearlmutter. Using polyvariant union-free flow analysis to compile a higher-order functional-programming language with a first-class derivative operator to efficient Fortran-like code. Technical Report TR-ECE-08-01, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA, January 2008a. URL http://docs.lib.purdue.edu/ecetr/367.

References V

Jeffrey Mark Siskind and Barak A. Pearlmutter. Putting the automatic back into AD: Part I, What's wrong. Technical Report TR-ECE-08-02, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA, January 2008b. URL http://docs.lib.purdue.edu/ecetr/368.

Jeffrey Mark Siskind and Barak A. Pearlmutter. Binomial checkpointing for arbitrary programs with no user annotation. Technical Report arXiv:1611.03410, September 2016a. Extended abstract presented at the AD 2016 Conference, Oxford UK.

Jeffrey Mark Siskind and Barak A. Pearlmutter. Efficient implementation of a higher-order language with built-in AD. Technical Report arXiv:1611.03146, September 2016b. Extended abstract presented at the AD 2016 Conference, Oxford UK.

Bert Speelpenning. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, January 1980.

Brook Taylor. *Methodus Incrementorum Directa et Inversa*. London, 1715.

References VI

A. M. Turing. On computable numbers with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230–65, December 1936. Correction, ibid, 2(43) 544-546 (jan 1937).

Robert Edwin Wengert. A simple automatic derivative evaluation program. *Comm. of the ACM*, 7 (8):463–4, August 1964. doi: 10.1145/355586.364791.

Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–80, 1989.