
Karl Gustav Jacob Jacobi
Friedrich Julius Richelot
Carl Gottfried Neumann
William Edward Story
Solomon Lefschetz
Albert William Tucker
Marvin Lee Minsky
Patrick Henry Winston
Robert C. Berwick
Jeffrey Mark Siskind
Scott Fahlman
David Touretzky
Barak Avrum Pearlmutter

Jonathan Eden Siskind

Algorithmic Differentiation of Functional Programs

Jeffrey Mark Siskind

School of Electrical and Computer Engineering
Purdue University
Electrical Engineering Building, Room 313E
465 Northwestern Avenue
West Lafayette IN 47907–2035 USA
voice: 765/496–3197
fax: 765/494–6440
qobi@purdue.edu
http://www.ece.purdue.edu/~qobi

January 24, 2005

Joint work with Barak A. Pearlmutter.

Lambda: the Ultimate Calculus

Jeffrey Mark Siskind

School of Electrical and Computer Engineering
Purdue University
Electrical Engineering Building, Room 313E
465 Northwestern Avenue
West Lafayette IN 47907–2035 USA
voice: 765/496–3197
fax: 765/494–6440
qobi@purdue.edu
http://www.ece.purdue.edu/~qobi

January 24, 2005

Joint work with Barak A. Pearlmutter.

Backpropagation through Functional Programs

Jeffrey Mark Siskind

School of Electrical and Computer Engineering
Purdue University
Electrical Engineering Building, Room 313E
465 Northwestern Avenue
West Lafayette IN 47907–2035 USA
voice: 765/496–3197
fax: 765/494–6440
qobi@purdue.edu
http://www.ece.purdue.edu/~qobi

January 24, 2005

Joint work with Barak A. Pearlmutter.

Lambda: the Ultimate Neural Network

Jeffrey Mark Siskind

School of Electrical and Computer Engineering
Purdue University
Electrical Engineering Building, Room 313E
465 Northwestern Avenue
West Lafayette IN 47907–2035 USA
voice: 765/496–3197
fax: 765/494–6440
qobi@purdue.edu
http://www.ece.purdue.edu/~qobi

January 24, 2005

Joint work with Barak A. Pearlmutter.

Symbolicism: the Ultimate Connectionism

Jeffrey Mark Siskind

School of Electrical and Computer Engineering
Purdue University
Electrical Engineering Building, Room 313E
465 Northwestern Avenue
West Lafayette IN 47907–2035 USA
voice: 765/496–3197
fax: 765/494–6440
qobi@purdue.edu
http://www.ece.purdue.edu/~qobi

January 24, 2005

Joint work with Barak A. Pearlmutter.

Maybe the Brain Really Does Run Lisp After All

Jeffrey Mark Siskind

School of Electrical and Computer Engineering
Purdue University
Electrical Engineering Building, Room 313E
465 Northwestern Avenue
West Lafayette IN 47907–2035 USA
voice: 765/496–3197
fax: 765/494–6440
qobi@purdue.edu
http://www.ece.purdue.edu/~qobi

January 24, 2005

Joint work with Barak A. Pearlmutter.

Leibnitz (1664) + Church (1941) = Pearlmutter & Siskind (2005)

Differential Calculus for Dummies

(in 6 slides)

Notation

- x, y, $\mathbf{x}$, $\mathbf{X}$, x', x_1, $\mathbf{x}[1]$, e, f, g, u, b, p
- comma, right associates
- juxtaposition, left associates
  - function application
  - function composition
  - matrix-vector multiplication
  - matrix-matrix multiplication
  - scalar-scalar multiplication
- $e_1 + e_2$ denotes $+\,(e_1, e_2)$, left associates
- $e_1 \oplus e_2$ denotes $\oplus\,(e_1, e_2)$, left associates
- $\sum_{i=1}^{n} e_i$ denotes $e_1 + \cdots + e_n$
- $\mathbf{X}^\top$: $(\mathbf{X}_1 \ \cdots \ \mathbf{X}_n)^\top = \mathbf{X}_n^\top \ \cdots \ \mathbf{X}_1^\top$

Derivatives

$$\frac{d}{dx} \qquad \underbrace{f}_{\mathbb{R}\to\mathbb{R}} = \underbrace{f'}_{\mathbb{R}\to\mathbb{R}}$$

$$\frac{d}{dx} \;:\; (\mathbb{R}\to\mathbb{R}) \to (\mathbb{R}\to\mathbb{R})$$

$$\mathcal{D} \;:\; (\mathbb{R}\to\mathbb{R}) \to (\mathbb{R}\to\mathbb{R})$$

Partial Derivatives

$$\frac{\partial}{\partial x} \qquad \underbrace{f}_{\mathbb{R}^n \to \mathbb{R}} = \underbrace{f'}_{\mathbb{R}^n \to \mathbb{R}}$$

$$\frac{\partial}{\partial x} \quad : \quad (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R})$$

$$\mathcal{D}_i \quad : \quad (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R})$$

Gradients

$$\nabla\ f\ \mathbf{x} \triangleq (\mathcal{D}_1\ f\ \mathbf{x}), \ldots, (\mathcal{D}_n\ f\ \mathbf{x})$$

$$\nabla : (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}^n)$$

Jacobians

$$f \quad : \quad \mathbb{R}^m \to \mathbb{R}^n$$

$$\mathbf{f} \quad : \quad (\mathbb{R}^m \to \mathbb{R})^n$$

$$(\mathcal{J}\ f\ \mathbf{x})[i,j] \quad \triangleq \quad (\nabla\ \mathbf{f}[i])[j]$$

$$\mathcal{J} \quad : \quad (\mathbb{R}^m \to \mathbb{R}^n) \to (\mathbb{R}^m \to \mathbb{R}^{m \times n})$$

Operators

$\mathcal{D}$, ∇, and $\mathcal{J}$ are traditionally called *operators*.
A more modern term is *higher-order functions*.
Higher-order functions are common in mathematics, physics, and engineering:

> summations, comprehensions, quantifications, optimizations, integrals, convolutions, filters, edge detectors, Fourier transforms, differential equations, Hamiltonians, functionals, ...

The Chain Rule

$$(f \circ g)\ x = (g\ f)\ x = g\ (f\ x)$$

$$\mathcal{D}\ (g\ f)\ x = (\mathcal{D}\ g\ f\ x)\ (\mathcal{D}\ f\ x)$$

$$\mathcal{J}\ (g\ f)\ \mathbf{x} = (\mathcal{J}\ g\ f\ \mathbf{x})\ (\mathcal{J}\ f\ \mathbf{x})$$

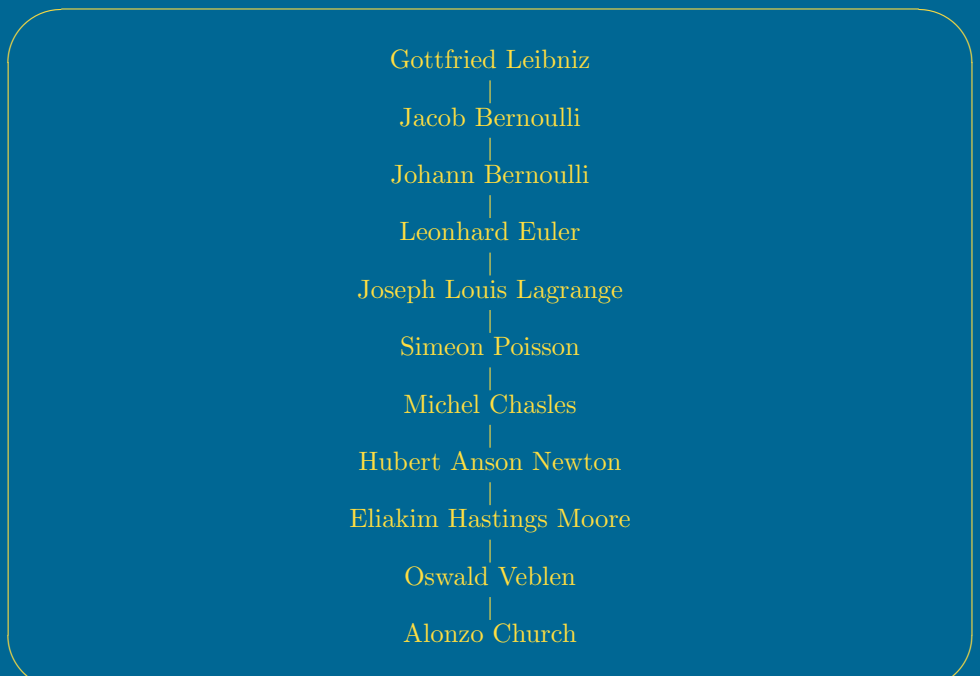
Everything You Always Wanted to Know About the Lambda Calculus*

(in 7 slides)

*But Were Afraid To Ask

Church (1941)

It is, of course, not excluded that the range of arguments or range of values of a function should consist wholly or partly of functions. The derivative, as this notion appears in the elementary differential calculus, is a familiar mathematical example of a function for which both ranges consist of functions.


Gottfried Leibniz
Jacob Bernoulli
Johann Bernoulli
Leonhard Euler
Joseph Louis Lagrange
Simeon Poisson
Michel Chasles
Hubert Anson Newton
Eliakim Hastings Moore
Oswald Veblen
Alonzo Church

Functional Programming

```
int f(int n)
{ int i, p = 1;
  for (i = 1; i<n; i++)
  { p = p*i;}
  return p;}
```

$$f\ n \triangleq \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times (f\ (n-1)) \textbf{ fi}$$

Higher-Order Functions

$$\sum_{i=1}^{n} \exp\ i$$

$$\prod_{i=1}^{n} \sin\ i$$

$$\textsc{fold}\ i, a, f, g \triangleq \textbf{if}\ i = 0\ \textbf{then}\ a\ \textbf{else}\ \textsc{fold}\ (i-1), (g\ a, (f\ i)), f, g\ \textbf{fi}$$

$$\textsc{fold}\ n, 0, \exp, +$$

$$\textsc{fold}\ n, 1, \sin, \times$$

$$\sum_{i=1}^{n} 2i + 1$$

$$f\ i \triangleq 2i + 1$$

$$\textsc{fold}\ n, 0, f, +$$

$$\textsc{fold}\ n, 0, (\lambda i\ 2i + 1), +$$

Closures

$(\lambda x\ 2x)\ 3 = 6$

$(\lambda x\ \lambda y\ x + y)\ 3\ 4 = 7$

$(\lambda x\ \lambda y\ x + y)\ 3 = ?$

$(\lambda x\ \lambda y\ x + y)\ 3 = \langle \{x \mapsto 3\}, \lambda y\ x + y \rangle$

$\lambda x\ \lambda y\ x + y$

$\lambda x, y\ x + y$

Tail Recursion (Steele 1976)

$$
\begin{array}{l}
f\ n \triangleq \textbf{if } n = 0 \\
\qquad \textbf{then } 1 \\
\qquad \textbf{else } n \times (f\ (n-1))\ \textbf{fi}
\end{array}
$$

$$
\begin{array}{rcl}
g\ i, p & \triangleq & \textbf{if } i = 0 \\
 & & \textbf{then } p \\
 & & \textbf{else } g\ (i-1), (p \times i)\ \textbf{fi} \\
f\ n & \triangleq & g\ n, 1
\end{array}
$$

Marvin Lee Minsky

|

Gerald Jay Sussman

|

Guy Lewis Steele, Jr.

Continuations (Landin 1965, Reynolds 1972)

$$
\begin{array}{lcl}
f\ x & \triangleq & e_1 \\
g\ x & \triangleq & e_2 \\
h\ x & \triangleq & e_3 \\
p\ x & \triangleq & h\ (g\ (f\ x))
\end{array}
\qquad\qquad
\begin{array}{lcl}
f'\ c, x & \triangleq & c\ e_1 \\
g'\ c, x & \triangleq & c\ e_2 \\
h'\ c, x & \triangleq & c\ e_3 \\
p'\ c, x & \triangleq & f'\ (\lambda x_1\ g'\ (\lambda x_2\ h'\ c, x_2), x_1), x
\end{array}
$$

The Lambda Calculus

if e_1 **then** e_2 **else** e_3 **fi** $\quad\leadsto\quad$ IF e_1 $(\lambda x\ e_2)$ $(\lambda x\ e_3)$ [ ]

$$e ::= x \mid e_1\ e_2 \mid \lambda x\ e$$

Compositional Derivative Operators—I

$$f_n \cdots f_1$$

$$\mathcal{J}\,(f_n \cdots f_1)$$

$$\mathcal{J}\,(f_n \cdots f_1)\,\mathbf{x} = \prod_{i=n}^{1}\left(\left(\mathcal{J}\,f_i\,\left(\prod_{j=i-1}^{1} f_j\right)\right)\,\mathbf{x}\right)$$

$\mathcal{J}\,(f_n \cdots f_1)$ is not compositional in $(\mathcal{J}\,f_1), \ldots, (\mathcal{J}\,f_n)$.

Compositional Derivative Operators—II

$$\overrightarrow{\nabla}\, f\, \mathbf{x}, \acute{\mathbf{x}} \triangleq \mathcal{J}\, f\, \mathbf{x}\, \acute{\mathbf{x}}$$
$$\overleftarrow{\nabla}\, f\, \mathbf{x}, \grave{\mathbf{y}} \triangleq (\mathcal{J}\, f\, \mathbf{x})^{\top}\, \grave{\mathbf{y}}$$

- $\mathbf{x}$ is a *primal* variable
- $\acute{\mathbf{x}}$ is a *forward adjoint* variable
- $\grave{\mathbf{x}}$ is a *reverse adjoint* variable

The rows and columns of $\mathcal{J}\, f\, \mathbf{x}$ can be computed as $\overleftarrow{\nabla}\, f\, \mathbf{x}, \mathbf{e}$ and $\overrightarrow{\nabla}\, f\, \mathbf{x}, \mathbf{e}$ for basis vectors $\mathbf{e}$ respectively.

Compositional Derivative Operators—III

$$\begin{aligned}\overrightarrow{\nabla}\,(g\,f)\,\mathbf{x},\acute{\mathbf{x}} &= \overrightarrow{\nabla}\,g\,(f\,\mathbf{x}),(\overrightarrow{\nabla}\,f\,\mathbf{x},\acute{\mathbf{x}})\\ \overleftarrow{\nabla}\,(g\,f)\,\mathbf{x},\grave{\mathbf{y}} &= \overleftarrow{\nabla}\,f\,\mathbf{x},(\overleftarrow{\nabla}\,g\,(f\,\mathbf{x}),\grave{\mathbf{y}})\end{aligned}$$

One cannot compose $\overrightarrow{\nabla}\,f$ with $\overrightarrow{\nabla}\,g$ because the input and output of $\overrightarrow{\nabla}\,f$ are not of the same type. Similarly for $\overleftarrow{\nabla}\,f$.

Compositional Derivative Operators—IV

$$\overrightarrow{\mathcal{J}}\ f\ \mathbf{x}, \acute{\mathbf{x}} \triangleq (f\ \mathbf{x}), (\overrightarrow{\nabla}\ f\ \mathbf{x}, \acute{\mathbf{x}})$$
$$\overleftarrow{\mathcal{J}}\ f\ \mathbf{x}, \tilde{\mathbf{x}} \triangleq (f\ \mathbf{x}), (\tilde{\mathbf{x}}\ \lambda \grave{\mathbf{y}}\ \overleftarrow{\nabla}\ f\ \mathbf{x}, \grave{\mathbf{y}})$$

- $\lambda \grave{\mathbf{y}}\ \overleftarrow{\nabla}\ f\ \mathbf{x}, \grave{\mathbf{y}}$ is a *local backpropagator*
- $\tilde{\mathbf{x}}$ is an *input backpropagator*
- their composition is an *output backpropagator*

Compositional Derivative Operators—V

$$
\begin{aligned}
\overrightarrow{\nabla}\, f\, \mathbf{x}, \acute{\mathbf{x}} &= \text{CDR}\, (\overrightarrow{\mathcal{J}}\, f\, \mathbf{x}, \acute{\mathbf{x}}) \\
\overleftarrow{\nabla}\, f\, \mathbf{x}, \grave{\mathbf{y}} &= \text{CDR}\, (\overleftarrow{\mathcal{J}}\, f\, \mathbf{x}, I)\, \grave{\mathbf{y}} \\
\overrightarrow{\mathcal{J}}\, (g\, f) &= (\overrightarrow{\mathcal{J}}\, g)\, (\overrightarrow{\mathcal{J}}\, f) \\
\overleftarrow{\mathcal{J}}\, (g\, f) &= (\overleftarrow{\mathcal{J}}\, g)\, (\overleftarrow{\mathcal{J}}\, f) \\
\overrightarrow{\mathcal{J}}\, (f_n \cdots f_1) &= (\overrightarrow{\mathcal{J}}\, f_n) \cdots (\overrightarrow{\mathcal{J}}\, f_1) \\
\overleftarrow{\mathcal{J}}\, (f_n \cdots f_1) &= (\overleftarrow{\mathcal{J}}\, f_n) \cdots (\overleftarrow{\mathcal{J}}\, f_1)
\end{aligned}
$$

Traditional Forward-Mode AD—I

$$
\begin{array}{rclcrcl}
\mathbf{x}_1 & = & f_1\,\mathbf{x}_0 & & \mathbf{x}_1, \acute{\mathbf{x}}_1 & = & \overrightarrow{\mathcal{T}}\, f_1\,\mathbf{x}_0, \acute{\mathbf{x}}_0 \\
\mathbf{x}_2 & = & f_2\,\mathbf{x}_1 & & \mathbf{x}_2, \acute{\mathbf{x}}_2 & = & \overrightarrow{\mathcal{T}}\, f_2\,\mathbf{x}_1, \acute{\mathbf{x}}_1 \\
 & \vdots & & & & \vdots & \\
\mathbf{x}_n & = & f_n\,\mathbf{x}_{n-1} & & \mathbf{x}_n, \acute{\mathbf{x}}_n & = & \overrightarrow{\mathcal{T}}\, f_n\,\mathbf{x}_{n-1}, \acute{\mathbf{x}}_{n-1}
\end{array}
$$

Traditional Forward-Mode AD—II

$$
\begin{array}{lcllcl}
x_j & = & u\ x_i & x_j, \acute{x}_j & = & (u\ x_i), (\mathcal{D}\ u\ x_i\ \acute{x}_i) \\
x_k & = & b\ x_i, x_j & x_k, \acute{x}_k & = & (b\ x_i, x_j), (\mathcal{D}_1\ b\ x_i, x_j\ \acute{x}_i + \\
 & & & & & \quad \mathcal{D}_2\ b\ x_i, x_j\ \acute{x}_j)
\end{array}
$$

Traditional Reverse-Mode AD—I

$$
\begin{array}{rcl@{\qquad}rcl}
\mathbf{x}_1 & = & f_1\ \mathbf{x}_0 & \mathbf{x}_1, \tilde{\mathbf{x}}_1 & = & \overleftarrow{\mathcal{J}}\ f_1\ \mathbf{x}_0, I \\
\mathbf{x}_2 & = & f_2\ \mathbf{x}_1 & \mathbf{x}_2, \tilde{\mathbf{x}}_2 & = & \overleftarrow{\mathcal{J}}\ f_2\ \mathbf{x}_1, \tilde{\mathbf{x}}_1 \\
 & \vdots & & & \vdots & \\
\mathbf{x}_n & = & f_n\ \mathbf{x}_{n-1} & \mathbf{x}_n, \tilde{\mathbf{x}}_n & = & \overleftarrow{\mathcal{J}}\ f_n\ \mathbf{x}_{n-1}, \tilde{\mathbf{x}}_{n-1} \\
 & & & \tilde{\mathbf{x}}_n\ \dot{\mathbf{x}}_n & &
\end{array}
$$

Traditional Reverse-Mode AD—II

$$
\begin{array}{rcl@{\qquad}rcl}
\mathbf{x}_1 & = & f_1\,\mathbf{x}_0 & \mathbf{x}_1, \tilde{\mathbf{x}}_1 & = & \overleftarrow{\mathcal{J}}\; f_1\,\mathbf{x}_0, I \\
\mathbf{x}_2 & = & f_2\,\mathbf{x}_1 & \mathbf{x}_2, \tilde{\mathbf{x}}_2 & = & \overleftarrow{\mathcal{J}}\; f_2\,\mathbf{x}_1, \tilde{\mathbf{x}}_1 \\
 & \vdots & & & \vdots & \\
\mathbf{x}_n & = & f_n\,\mathbf{x}_{n-1} & \mathbf{x}_n, \tilde{\mathbf{x}}_n & = & \overleftarrow{\mathcal{J}}\; f_n\,\mathbf{x}_{n-1}, \tilde{\mathbf{x}}_{n-1} \\
 & & & \grave{\mathbf{x}}_{n-1} & = & \overleftarrow{\nabla}\; f_n\,\mathbf{x}_n, \grave{\mathbf{x}}_n \\
 & & & \grave{\mathbf{x}}_{n-2} & = & \overleftarrow{\nabla}\; f_{n-1}\,\mathbf{x}_{n-1}, \grave{\mathbf{x}}_{n-1} \\
 & & & & \vdots & \\
 & & & \grave{\mathbf{x}}_0 & = & \overleftarrow{\nabla}\; f_1\,\mathbf{x}_1, \grave{\mathbf{x}}_1
\end{array}
$$

Traditional Reverse-Mode AD—III

$$
\begin{array}{lcl}
x_j & = & u\, x_i \\
x_k & = & b\, x_i, x_j
\end{array}
\qquad
\begin{array}{lcl}
x_j & = & u\, x_i \\
\grave{x}_i & = & \grave{x}_i + \mathcal{D}\, u\, x_i\, \grave{x}_j \\
x_k & = & b\, x_i, x_j \\
\grave{x}_i & = & \grave{x}_i + \mathcal{D}_1\, b\, x_i, x_j\, \grave{x}_k \\
\grave{x}_j & = & \grave{x}_j + \mathcal{D}_2\, b\, x_i, x_j\, \grave{x}_k
\end{array}
$$

Jacobians of CAR, CDR, and CONS—I

- pairs are of type $\mathbb{R}^2$
- CAR : $\mathbb{R}^2 \to \mathbb{R}$
- CDR : $\mathbb{R}^2 \to \mathbb{R}$
- CONS : $((\mathbb{R}^n \to \mathbb{R}) \times (\mathbb{R}^n \to \mathbb{R})) \to (\mathbb{R}^n \to \mathbb{R}^2)$

Jacobians of CAR, CDR, and CONS—II

$$
\begin{aligned}
\mathcal{J}\ \text{CAR}\ \mathbf{x} &= (1\ 0) \\
\mathcal{J}\ \text{CDR}\ \mathbf{x} &= (0\ 1) \\
\mathcal{J}\ \text{CAR}\ \mathbf{x}\ \acute{\mathbf{x}} &= \acute{\mathbf{x}}[1] \\
\mathcal{J}\ \text{CDR}\ \mathbf{x}\ \acute{\mathbf{x}} &= \acute{\mathbf{x}}[2] \\
(\mathcal{J}\ \text{CAR}\ \mathbf{x})^{\top}\ \grave{y} &= \begin{pmatrix} \grave{y} \\ 0 \end{pmatrix} \\
(\mathcal{J}\ \text{CDR}\ \mathbf{x})^{\top}\ \grave{y} &= \begin{pmatrix} 0 \\ \grave{y} \end{pmatrix}
\end{aligned}
$$

Jacobians of CAR, CDR, and CONS—III

$$
\begin{aligned}
(\mathcal{J}\ f, g\ \mathbf{x})[i,1] &= (\nabla\ f\ \mathbf{x})[i] \\
(\mathcal{J}\ f, g\ \mathbf{x})[i,2] &= (\nabla\ g\ \mathbf{x})[i] \\
\mathcal{J}\ f, g\ \mathbf{x} &= \begin{pmatrix} \mathcal{D}_1\ f\ \mathbf{x} & \cdots & \mathcal{D}_n\ f\ \mathbf{x} \\ \mathcal{D}_1\ g\ \mathbf{x} & \cdots & \mathcal{D}_n\ g\ \mathbf{x} \end{pmatrix} \\
\mathcal{J}\ f, g\ \mathbf{x}\ \acute{\mathbf{x}} &= \begin{pmatrix} \sum_{i=1}^{n} \mathcal{D}_i\ f\ \mathbf{x}\ \acute{\mathbf{x}}[i] \\ \sum_{i=1}^{n} \mathcal{D}_i\ g\ \mathbf{x}\ \acute{\mathbf{x}}[i] \end{pmatrix} \\
&= \begin{pmatrix} \mathcal{J}\ f\ \mathbf{x}\ \acute{\mathbf{x}} \\ \mathcal{J}\ g\ \mathbf{x}\ \acute{\mathbf{x}} \end{pmatrix} \\
(\mathcal{J}\ f, g\ \mathbf{x})^{\top}\ \grave{\mathbf{y}} &= \begin{pmatrix} \mathcal{D}_1\ f\ \mathbf{x}\ \grave{\mathbf{y}}[1] + \mathcal{D}_1\ g\ \mathbf{x}\ \grave{\mathbf{y}}[2] \\ \vdots \\ \mathcal{D}_n\ f\ \mathbf{x}\ \grave{\mathbf{y}}[1] + \mathcal{D}_n\ g\ \mathbf{x}\ \grave{\mathbf{y}}[2] \end{pmatrix} \\
&= \mathcal{J}\ f\ \mathbf{x}\ \grave{\mathbf{y}}[1] \oplus \mathcal{J}\ g\ \mathbf{x}\ \grave{\mathbf{y}}[2]
\end{aligned}
$$

VLAD: Functional Language for AD—I

- Similar to SCHEME.
- Only functional (side-effect free) constructs are supported.
- The only data types supported are the empty list, Booleans, real numbers, pairs, and procedures that take one argument and return one result. Thus VLAD objects are all of the following type:

$$\tau ::= \textbf{null} \mid \textbf{boolean} \mid \mathbb{R} \mid \tau_1 \times \tau_2 \mid \tau_1 \to \tau_2$$

- Primitive procedures that take two arguments take them as a pair.
- Except that `cons` is curried.

VLAD: Functional Language for AD—II

procedures $u : \mathbb{R} \to \mathbb{R}$: `sqrt`, `exp`, `log`, `sin`, and `cos`.
procedures $b : (\mathbb{R} \times \mathbb{R}) \to \mathbb{R}$: `+`, `-`, `*`, `/`, and `atan`.
procedures $p : \tau \to$ **boolean** : `=`, `<`, `>`, `<=`, `>=`, `zero?`, `positive?`, `negative?`, `null?`, `boolean?`, `real?`, `pair?`, and `procedure?`.
other : `car`, `cdr`, and `cons`.

VLAD: Functional Language for AD—III

- We use [ ] to denote the empty list and $[e_1; \ldots; e_n]$ as shorthand for $e_1, \ldots, e_n, [\,]$.
- We use e_1, e_2 as shorthand for CONS e_1 e_2.
- We allow lambda expressions to have tuples as parameters as shorthand for the appropriate destructuring. For example:

$$\lambda x_1, (x_2, x_3) \ \ldots x_2 \ldots \ \rightsquigarrow \ \lambda x \ \ldots \text{CAR}\ (\text{CDR}\ x) \ldots$$

- $\overrightarrow{\mathcal{J}}$, $\overleftarrow{\mathcal{J}}$

Adjoint Types

$$
\begin{array}{rcl}
\overline{\mathbf{null}} & \triangleq & \mathbf{null} \\
\overline{\mathbf{boolean}} & \triangleq & \mathbf{null} \\
\overline{\mathbb{R}} & \triangleq & \mathbb{R} \\
\overline{\tau_1 \times \tau_2} & \triangleq & \overline{\tau_1} \times \overline{\tau_2} \\
\overline{\tau_1 \to \tau_2} & \triangleq & \mathbf{null}
\end{array}
$$

The Type of $\overrightarrow{\mathcal{J}}$

$\overrightarrow{\mathcal{J}} : (\tau_1 \to \tau_2) \to ((\overrightarrow{\tau_1} \times \overrightarrow{\tau_1}) \to (\overrightarrow{\tau_2} \times \overrightarrow{\tau_2}))$
$\overrightarrow{\mathcal{J}} : \tau \to \overrightarrow{\tau}$

$$\begin{array}{rcl}
\overrightarrow{\textbf{null}} & \triangleq & \textbf{null} \\
\overrightarrow{\textbf{boolean}} & \triangleq & \textbf{boolean} \\
\overrightarrow{\mathbb{R}} & \triangleq & \mathbb{R} \\
\overrightarrow{\tau_1 \times \tau_2} & \triangleq & \overrightarrow{\tau_1} \times \overrightarrow{\tau_2} \\
\overrightarrow{\tau_1 \to \tau_2} & \triangleq & (\overrightarrow{\tau_1} \times \overrightarrow{\tau_1}) \to (\overrightarrow{\tau_2} \times \overrightarrow{\tau_2})
\end{array}$$

The Definition of $\overrightarrow{\mathcal{J}}$ on Non-Procedures

$$\begin{array}{rcll} \overrightarrow{\mathcal{J}}\ x & \triangleq & x & x \text{ is not a pair} \\ \overrightarrow{\mathcal{J}}\ x_1, x_2 & \triangleq & (\overrightarrow{\mathcal{J}}\ x_1), (\overrightarrow{\mathcal{J}}\ x_2) & \end{array}$$

The Definition of $\overrightarrow{\mathcal{J}}$ on Primitive Procedures

$$
\begin{array}{rcl}
\overrightarrow{\mathcal{J}}\ u & \triangleq & \lambda x, \acute{x}\ (u\ x), (\mathcal{D}\ u\ x\ \acute{x}) \\
\overrightarrow{\mathcal{J}}\ b & \triangleq & \lambda(x_1, x_2), (\acute{x}_1, \acute{x}_2) \\
 & & (b\ x_1, x_2), (\mathcal{D}_1\ b\ x_1, x_2\ \acute{x}_1 + \mathcal{D}_2\ b\ x_1, x_2\ \acute{x}_2) \\
\overrightarrow{\mathcal{J}}\ p & \triangleq & \lambda x, \acute{x}\ (p\ x), [\,] \\
\overrightarrow{\mathcal{J}}\ \textsc{car} & \triangleq & \lambda(x_1, x_2), (\acute{x}_1, \acute{x}_2)\ x_1, \acute{x}_1 \\
\overrightarrow{\mathcal{J}}\ \textsc{cdr} & \triangleq & \lambda(x_1, x_2), (\acute{x}_1, \acute{x}_2)\ x_2, \acute{x}_2 \\
\overrightarrow{\mathcal{J}}\ \textsc{cons} & \triangleq & \lambda x_1, \acute{x}_1\ (\lambda x_2, \acute{x}_2\ (x_1, x_2), (\acute{x}_1, \acute{x}_2)), [\,] \\
\overrightarrow{\mathcal{J}}\ \textsc{if} & \triangleq & \lambda x_1, \acute{x}_1 \\
 & & (\lambda x_2, \acute{x}_2\ (\lambda x_3, \acute{x}_3\ \textbf{if}\ x_1\ \textbf{then}\ x_2, \acute{x}_2\ \textbf{else}\ x_3, \acute{x}_3\ \textbf{fi}), [\,]), [\,]
\end{array}
$$

The Definition of $\overrightarrow{\mathcal{J}}$ on Closures

$$\overrightarrow{\mathcal{J}}\ \langle \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}, e\rangle \triangleq \langle \{x_1 \mapsto \overrightarrow{\mathcal{J}}\ v_1, \ldots, x_n \mapsto \overrightarrow{\mathcal{J}}\ v_n\},\ \overrightarrow{e}\ \rangle$$

$$\begin{array}{rcll}
\overrightarrow{x} & \leadsto & x, \acute{x} & x \text{ is bound in } e \\
\overrightarrow{x} & \leadsto & x, (\underline{0}\ x) & x \text{ is free in } e \\
\overrightarrow{e_1\ e_2} & \leadsto & \text{CAR}\ \overrightarrow{e_1}\ \overrightarrow{e_2} & \\
\overrightarrow{\lambda x\ e'} & \leadsto & (\lambda x, \acute{x}\ \overrightarrow{e'}), [\,] &
\end{array}$$

$$\begin{array}{l}
\underline{0}\ x \triangleq \textbf{if}\ \text{REAL?}\ x\ \textbf{then}\ 0 \\
\quad \textbf{elif}\ \text{PAIR?}\ x\ \textbf{then}\ (\underline{0}\ (\text{CAR}\ x)), (\underline{0}\ (\text{CDR}\ x)) \\
\quad \textbf{else}\ [\,]\ \textbf{fi}
\end{array}$$

The Type of $\overleftarrow{\mathcal{J}}$

$\overleftarrow{\mathcal{J}} : (\tau_1 \to \tau_2) \to ((\overleftarrow{\tau_1} \times (\overline{\tau_1} \to \overline{\tau_3})) \to (\overleftarrow{\tau_2} \times (\overline{\tau_2} \to \overline{\tau_3})))$
$\overleftarrow{\mathcal{J}} : \tau \to \overleftarrow{\tau}$

$$
\begin{aligned}
\overleftarrow{\mathbf{null}} &\triangleq \mathbf{null} \\
\overleftarrow{\mathbf{boolean}} &\triangleq \mathbf{boolean} \\
\overleftarrow{\mathbb{R}} &\triangleq \mathbb{R} \\
\overleftarrow{\tau_1 \times \tau_2} &\triangleq \overleftarrow{\tau_1} \times \overleftarrow{\tau_2} \\
\overleftarrow{\tau_1 \to \tau_2} &\triangleq (\overleftarrow{\tau_1} \times (\overline{\tau_1} \to \overline{\tau_3})) \to (\overleftarrow{\tau_2} \times (\overline{\tau_2} \to \overline{\tau_3}))
\end{aligned}
$$

The Definition of $\overleftarrow{\mathcal{J}}$ on Non-Procedures

$$\begin{array}{rcll} \overleftarrow{\mathcal{J}}\ x & \triangleq & x & x \text{ is not a pair} \\ \overleftarrow{\mathcal{J}}\ x_1, x_2 & \triangleq & (\overleftarrow{\mathcal{J}}\ x_1), (\overleftarrow{\mathcal{J}}\ x_2) & \end{array}$$

The Definition of $\overleftarrow{\mathcal{J}}$ on Primitive Procedures

$$
\begin{array}{rcl}
\overleftarrow{\mathcal{J}}\ u & \triangleq & \lambda x, \tilde{x}\ (u\ x), (\tilde{x}\ (\mathcal{D}\ u\ x)) \\
\overleftarrow{\mathcal{J}}\ b & \triangleq & \lambda(x_1, x_2), \tilde{x} \\
 & & (b\ x_1, x_2), (\tilde{x}\ (\mathcal{D}_1\ b\ x_1, x_2), (\mathcal{D}_2\ b\ x_1, x_2)) \\
\overleftarrow{\mathcal{J}}\ p & \triangleq & \lambda x, \tilde{x}\ (p\ x), \lambda \grave{y}\ \underline{0}\ x \\
\overleftarrow{\mathcal{J}}\ \textsc{car} & \triangleq & \lambda(x_1, x_2), \tilde{x}\ x_1, \lambda \grave{y}\ \tilde{x}\ \grave{y}, (\underline{0}\ x_2) \\
\overleftarrow{\mathcal{J}}\ \textsc{cdr} & \triangleq & \lambda(x_1, x_2), \tilde{x}\ x_2, \lambda \grave{y}\ \tilde{x}\ (\underline{0}\ x_1), \grave{y} \\
\overleftarrow{\mathcal{J}}\ \textsc{cons} & \triangleq & \lambda x_1, \tilde{x}_1\ (\lambda x_2, \tilde{x}_2\ (x_1, x_2), \\
 & & \qquad \lambda \grave{y}\ \tilde{x}_1\ (\textsc{car}\ \grave{y}) \oplus \tilde{x}_2\ (\textsc{cdr}\ \grave{y})), \\
 & & \quad \lambda \grave{y}\ \tilde{x}_1\ (\underline{0}\ x_1) \\
\overleftarrow{\mathcal{J}}\ \textsc{if} & \triangleq & \lambda x_1, \tilde{x}_1 \\
 & & (\lambda x_2, \tilde{x}_2\ (\lambda x_3, \tilde{x}_3\ \textbf{if}\ x_1\ \textbf{then}\ x_2, \tilde{x}_2\ \textbf{else}\ x_3, \tilde{x}_3\ \textbf{fi}), \\
 & & \quad \lambda \grave{y}\ \tilde{x}_2\ (\underline{0}\ x_2)), \\
 & & \lambda \grave{y}\ \tilde{x}_1\ (\underline{0}\ x_1)
\end{array}
$$

The Definition of $\overleftarrow{\mathcal{J}}$ on Closures

$$\overleftarrow{\mathcal{J}}\ \langle\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}, e\rangle \triangleq \langle\{x_1 \mapsto \overleftarrow{\mathcal{J}}\ v_1, \ldots, x_n \mapsto \overleftarrow{\mathcal{J}}\ v_n\}, \overleftarrow{e}\ \rangle$$

where $e = \lambda x_0\ e'$.

$$\begin{array}{rcll}
\overleftarrow{x} & \rightsquigarrow & x, \tilde{x} & x \text{ is bound in } e \\
\overleftarrow{x} & \rightsquigarrow & x, \lambda\grave{y}\ \tilde{x}_0\ (\underline{0}\ x_0) & x \text{ is free in } e \\
\overleftarrow{e_1\ e_2} & \rightsquigarrow & \text{CAR}\ \overleftarrow{e_1}\ \overleftarrow{e_2} & \\
\overleftarrow{\lambda x\ e'} & \rightsquigarrow & (\lambda x, \tilde{x}\ \overleftarrow{e'}), \lambda\grave{y}\ \tilde{x}_0\ (\underline{0}\ x_0) &
\end{array}$$

$$\begin{array}{rl}
x_1 \oplus x_2 \triangleq & \textbf{if } \text{NULL?}\ x_1 \textbf{ then } [\,] \\
& \textbf{elif } \text{REAL?}\ x_1 \textbf{ then } x_1 + x_2 \\
& \textbf{else } (\text{CAR}\ x_1 \oplus \text{CAR}\ x_2), (\text{CDR}\ x_1 \oplus \text{CDR}\ x_2)\ \textbf{fi}
\end{array}$$

Fanout—The Problem

$$
\begin{array}{rrcl}
\lambda x_0 \ \textbf{let} & x_1 & \triangleq & x_0 + x_0; \\
& x_2 & \triangleq & x_1 + x_1; \\
& & \vdots & \\
& x_n & \triangleq & x_{n-1} + x_{n-1} \\
\textbf{in } x_n \ \textbf{end} & & &
\end{array}
$$

Fanout—One Solution

$$\text{FAN}\ f\ x \triangleq f\ x, x$$

$$\lambda x\ x + x + x \ \leadsto\ \lambda x\ \text{FAN}\ (\lambda x_1, x\ \text{FAN}\ (\lambda x_2, x_3\ x_1 + x_2 + x_3)\ x)\ x$$

$$\begin{aligned}
\overleftarrow{\mathcal{T}}\ \text{FAN} \triangleq \lambda f, \tilde{f}\ (\lambda x, \tilde{x}\ &\textbf{let}\ \hat{y} \triangleq f\ (x, x), I;\\
&y \triangleq \text{CAR}\ \hat{y};\\
&\tilde{y} \triangleq \text{CDR}\ \hat{y}\\
&\textbf{in}\ y, \lambda \grave{y}\ \textbf{let}\ \grave{x} \triangleq \tilde{y}\ \grave{y}\\
&\qquad\textbf{in}\ \tilde{x}\ (\text{CAR}\ \grave{x} \oplus \text{CDR}\ \grave{x})\ \textbf{end end}),\\
\lambda \grave{y}\ \tilde{f}\ (\underline{0}\ f)&
\end{aligned}$$

Derivatives

$$\mathcal{D}\ f\ x \triangleq \text{CDR}\ (\overrightarrow{\mathcal{J}}\ f\ x, 1)$$
$$\mathcal{D}\ f\ x \triangleq \text{CDR}\ (\overleftarrow{\mathcal{J}}\ f\ x, I)\ 1$$

Roots using Newton-Raphson

$$\begin{array}{l}\textsc{Root}\ f, x, \epsilon \triangleq \textbf{let}\ x' \triangleq x - \frac{f\ x}{\mathcal{D}\ f\ x} \\ \quad\quad \textbf{in if}\ |x - x'| \leq \epsilon \\ \quad\quad\quad \textbf{then}\ x \\ \quad\quad\quad \textbf{else}\ \textsc{Root}\ f, x', \epsilon\ \textbf{fi end}\end{array}$$

Univariate Optimizer (Line Search)

$$\textsc{Argmin}\ f, x, \epsilon \triangleq \textsc{Root}\ (\mathcal{D}\ f), x, \epsilon$$

Gradients

$$\nabla\, f\, x \triangleq \begin{array}{l} \textbf{let } n \triangleq \textsc{Length}\ x \\ \textbf{in } \textsc{Map}\ (\lambda i\ \textsc{cdr}\ (\overrightarrow{\mathcal{J}}\ f\ x, (e\ 1, i, n))), \\ \qquad (\iota\ n)\ \textbf{end} \end{array}$$

$$\nabla\, f\, x \triangleq \textsc{cdr}\ (\overleftarrow{\mathcal{J}}\ f\ x, I)\ 1$$

Gradient Descent

$$
\begin{array}{l}
\textsc{GradientDescent}\ f, x, \epsilon \triangleq \\
\quad \textbf{let}\ g \triangleq \nabla\ f\ x \\
\quad \textbf{in if}\ ||g|| \leq \epsilon \\
\qquad \textbf{then}\ x \\
\qquad \textbf{else}\ \textsc{GradientDescent} \\
\qquad\quad f, (x + \textsc{Argmin}\ (\lambda k\ f\ (x + kg)), 0, \epsilon\ g), \epsilon\ \textbf{fi end}
\end{array}
$$

Function Inversion

$$f^{-1}\ y \triangleq \textsc{Root}\ (\lambda x\ |(f\ x) - y|), x_0, \epsilon$$

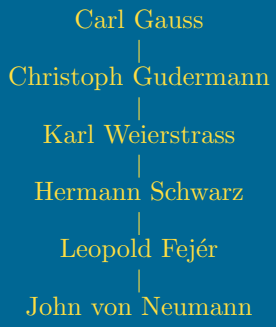
A Rational Agent

- The world is $w : (\textbf{state} \times \textbf{action}) \to \textbf{state}$
- Agent perception is $p_B : \textbf{state} \to \textbf{observation}$
- Agent reward is $r_B : \textbf{observation} \to \mathbb{R}$
- Goal is to maximize $r_B\ (p_B\ (w\ s, a))$
- But agent doesn't have s, w, p_B, and r_B
- Observation $o = p_B\ s$
- Models w_B, p_{BB}, and r_{BB} of w, p_B, and r_B respectively

$$\textsc{Agent}\ w_B, p_{BB}, r_{BB}, o \stackrel{\triangle}{=} \textsc{Argmax}\ (\lambda a\ r_{BB}\ (p_{BB}\ (w_B\ (p_{BB}^{-1}\ o), a))), a_0, \epsilon$$

A Pair of Interacting Rational Agents (von Neumann & Morgenstern 1944)

$$
\begin{array}{l}
\textsc{DoubleAgent}\ w_A, w_{AB}, p_{AA}, p_{AB}, p_{ABB}, r_{AA}, r_{ABB}, o \triangleq \\
\quad \textsc{Argmax}\ (\lambda a\ r_{AA} \\
\qquad\qquad (p_{AA} \\
\qquad\qquad\ (w_A\ (w_A\ (p_{AA}^{-1}\ o), a), \\
\qquad\qquad\qquad (\textsc{Argmax}\ (\lambda a'\ r_{ABB} \\
\qquad\qquad\qquad\qquad\qquad\qquad (p_{ABB} \\
\qquad\qquad\qquad\qquad\qquad\qquad\ (w_{AB}\ (p_{ABB}^{-1} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (p_{AB} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ (w_A\ (p_{AA}^{-1}\ o), a))), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad a'))), \\
\qquad\qquad\qquad\qquad\qquad\qquad a_0, \epsilon)))), \\
\qquad\qquad a_0, \epsilon
\end{array}
$$


Carl Gauss
Christoph Gudermann
Karl Weierstrass
Hermann Schwarz
Leopold Fejér
John von Neumann

Neural Nets
(Rumelhart, Hinton, & Williams 1986)

$$\begin{aligned}
\textsc{Neuron}\ w, x &\triangleq \textsc{Sigmoid}\ (w \cdot x) \\
\textsc{NeuralNet}\ w, x &\triangleq \textsc{Neuron}\ w'', \ldots (\textsc{Neuron}\ w', x') \ldots \\
\textsc{Error}\ w &\triangleq ||[y_1; \ldots; y_n] - \\
&\quad\ [\textsc{NeuralNet}\ w, x_1; \ldots; \textsc{NeuralNet}\ w, x_n]||
\end{aligned}$$

$\textsc{GradientDescent}\ \textsc{Error}, w_0, \epsilon$

Supervised Machine Learning (Function Approximation)

$\textsc{Error}\ w \triangleq |||[y_1; \ldots; y_n] - [f\ w, x_1; \ldots; f\ w, x_n]|||$

$\textsc{GradientDescent}\ \textsc{Error}, w_0, \epsilon$

Maximum Likelihood Estimation (Fisher 1921)

$$L_X(\theta) \stackrel{\triangle}{=} \prod_{x \in X} P(x|\theta)$$

GradientAscent L_X, θ_0, ϵ

Engineering Design

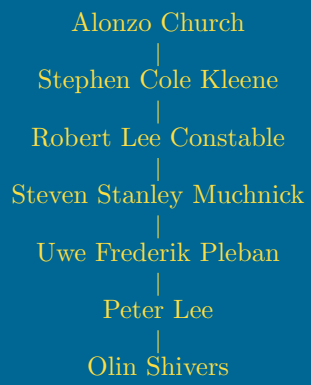
$$
\begin{array}{l}
\textsc{performance SplineControlPoints} \triangleq \\
\quad \textbf{let } \textsc{wing} \triangleq \textsc{SplineToSurface SplineControlPoints}; \\
\qquad \textsc{airflow} \triangleq \textsc{PDEsolver wing, NavierStokes}; \\
\qquad \textsc{lift, drag} \triangleq \textsc{SurfaceIntegral wing, airflow, force} \\
\quad \textbf{in } \textsc{DesignMetric lift, drag, (weight wing)} \textbf{ end}
\end{array}
$$

$$
\textsc{GradientAscent performance, SplineControlPoints}_0, \epsilon
$$

An Optimizing Compiler for VLAD

STALIN∇:

- polyvariant flow analysis (Shivers 1988)
- flow-directed lightweight closure conversion (Wand & Steckler 1994)
- flow-directed inlining
- compiling with continuations (Steele 1979, Appel 1992)
- unboxing
- partial evaluation


Alonzo Church
Stephen Cole Kleene
Robert Lee Constable
Steven Stanley Muchnick
Uwe Frederik Pleban
Peter Lee
Olin Shivers

Advantages—I

Functional programs represent the underlying mathematical notions more closely than imperative programs.

Advantages—II

Greater compositionality:

- root finders built on a derivative-taker
- line search built on root finders
- multivariate optimizers built on line search
- other multivariate optimizers (with identical APIs) build on Hessian-vector multipliers

  ⋮

Advantages—III

Greater modularity: by allowing the callee to specify the necessary AD, rather than insisting that the caller provide appropriately transformed functions, internals can be hidden and changed.

Advantages—IV

It is straightforward to generate higher-order derivatives, i.e. derivatives of derivatives.

Advantages—V

Differential forms become first-class higher-order functions that can be passed to optimizers or PDE solvers as part of an API. This allow one to easily express programming patterns, i.e. algorithm templates, that can be instantiated with different components as fillers. For example, one can construct an algorithm that needs an optimizer and leave the choice of optimizer unspecified, to be filled in later by passing the particular optimizer as a function parameter.

Advantages—VI

Gradients can even be taken through processes that themselves involve AD-based optimization or PDE solution.

Advantages—VII

In traditional AD formulations, the output of a reverse-mode transformation is a 'tape' that is a different kind of entity than user-written functions. It must be interpreted or run-time compiled. In contrast, in our approach, user-written functions, and the input and output of AD operators, are all the same kind of entity. Standard compilation techniques for functional programs can eliminate the need for interpretation or run-time compilation of derivatives and generate, at compile-time, code for derivatives that is as efficient as code for the primal calculation.