Karl Gustav Jacob Jacobi

Friedrich Julius Richelot

Carl Gottfried Neumann

William Edward Story

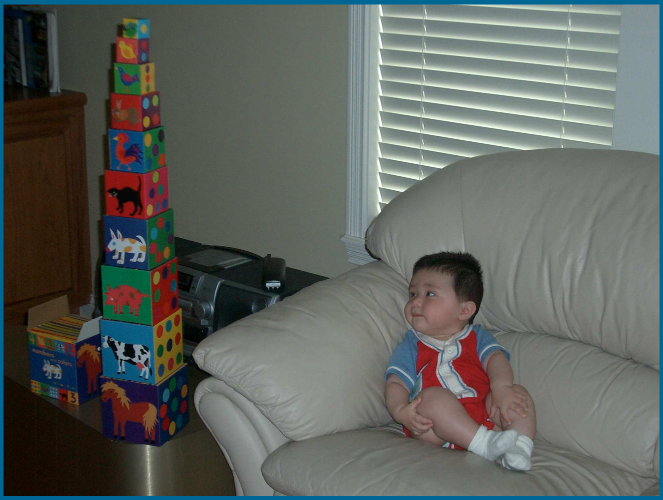
Solomon Lefschetz

Albert William Tucker

Marvin Lee Minsky

Patrick Henry Winston	Scott Fahlman
Robert C. Berwick	David Touretzky
Jeffrey Mark Siskind	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`

August 31, 2004

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`

August 31, 2004

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`

August 31, 2004

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`

August 31, 2004

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`

August 31, 2004

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`

August 31, 2004

Joint work with Barak A. Pearlmutter.

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

Differential Calculus for Dummies

(in 8 slides)

Notation

- x, y, $\mathbf{x}$, $\mathbf{X}$, x', x_1, $\mathbf{x}[1]$, $\mathbf{X}[1,1]$, e, f, g, u, b, p
- e_1, e_2 denotes pairing
- $e_1\ e_2$ (juxtaposition) denotes:
  - function application
  - function composition
  - matrix-vector multiplication
  - matrix-matrix multiplication
  - scalar-scalar multiplication
- $e_1 + e_2$ denotes $+\,(e_1, e_2)$
- $e_1 \oplus e_2$ denotes $\oplus\,(e_1, e_2)$, vector or tree addition
- $\displaystyle\sum_{i=1}^{n} e_i$ denotes $e_1 + \cdots + e_n$
- $\mathbf{X}^\top$ denotes matrix transpotion, $(\mathbf{X}\ \mathbf{Y})^\top = \mathbf{Y}^\top\ \mathbf{X}^\top$
- $f\ x_1\ \cdots\ x_n \triangleq e$ denotes $f \triangleq \lambda x_1\ \cdots\ \lambda x_n\ e$

Notation: Associativity and Priority

1. $\mathbf{x}[i]$ and $\mathbf{X}^\top$
2. e_1, e_2 right associates
3. $e_1\ e_2$ left associates
4. $e_1 + e_2$ and $e_1 \oplus e_2$ left associate
5. $\lambda x\ e$ and $\displaystyle\sum_{i=1}^{n} e_i$

Derivatives

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

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

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

Partial Derivatives

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

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

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

Gradients

$$\nabla \; f \; \mathbf{x} \;\; \stackrel{\triangle}{=} \;\; (\mathcal{D}_1 \; f \; \mathbf{x}), \ldots, (\mathcal{D}_n \; f \; \mathbf{x})$$

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

Jacobians

$$
\begin{aligned}
f &: \mathbb{R}^m \to \mathbb{R}^n \\
\mathbf{f} &: (\mathbb{R}^m \to \mathbb{R})^n \\
(\mathcal{J}\ f\ \mathbf{x})[i,j] &\triangleq (\nabla\ \mathbf{f}[i])[j] \\
\mathcal{J} &: (\mathbb{R}^m \to \mathbb{R}^n) \to (\mathbb{R}^m \to \mathbb{R}^{m\times n})
\end{aligned}
$$

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

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.

Church (1941, fourth paragraph)

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;}
```

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

Higher-Order Functions

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

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

$$\begin{aligned}\textsc{fold}\ i, a, f, g \stackrel{\triangle}{=}\ & \textbf{if}\ i = 0 \\ & \textbf{then}\ a \\ & \textbf{else}\ \textsc{fold}\ (i-1), (g\ a, (f\ i)), f, g\ \textbf{fi}\end{aligned}$$

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

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

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

$f\ i \stackrel{\triangle}{=} 2i + 1$

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

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

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)

$$
\begin{aligned}
f\ n \triangleq\ & \mathbf{if}\ n = 0 \\
& \mathbf{then}\ 1 \\
& \mathbf{else}\ n \times (f\ (n-1))\ \mathbf{fi}
\end{aligned}
$$

$$
\begin{aligned}
g\ i, p \quad &\triangleq \quad \mathbf{if}\ i = 0 \\
& \qquad \mathbf{then}\ p \\
& \qquad \mathbf{else}\ g\ (i-1), (p \times i)\ \mathbf{fi} \\
f\ n \quad &\triangleq \quad g\ n, 1
\end{aligned}
$$

Marvin Lee Minsky
|
Gerald Jay Sussman
|
Guy Lewis Steele, Jr.

Continuations (Landin 1965, Reynolds 1972)

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

The Lambda Calculus

$$\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \textbf{ fi} \quad \leadsto \quad \text{IF } e_1\ (\lambda x\ e_2)\ (\lambda x\ e_3)\ [\,]$$

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

Compositionality

An operator $\bigcirc$ is *compositional* if

$$\begin{aligned} \bigcirc\,(g\ f) &= (\bigcirc\, g)\ (\bigcirc\, f) \\ \bigcirc\,(f_n\ \cdots\ f_1) &= (\bigcirc\, f_n)\ \cdots\ (\bigcirc\, f_1) \end{aligned}$$

Compositional Derivative Operators—I

$$
\begin{array}{rcl}
f_n \ \cdots \ f_1 & & \\
\mathcal{J}\ (f_n \ \cdots \ f_1) & & \\
\mathcal{J}\ (g\ f)\ \mathbf{x} & = & (\mathcal{J}\ g\ f\ \mathbf{x})\ (\mathcal{J}\ f\ \mathbf{x}) \\
\mathcal{J}\ (f_n \ \cdots \ f_1)\ \mathbf{x} & = & (\mathcal{J}\ f_n\ f_{n-1}\ \cdots\ f_2\ f_1\ \mathbf{x}) \\
& & (\mathcal{J}\ f_{n-1}\ \cdots\ f_2\ f_1\ \mathbf{x}) \\
& & \vdots \\
& & (\mathcal{J}\ f_2\ f_1\ \mathbf{x}) \\
& & (\mathcal{J}\ f_1\ \mathbf{x})
\end{array}
$$

problem: $\mathcal{J}$ is not compositional.

Compositional Derivative Operators—II

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

- $\mathbf{x}$ is a *primal* variable.
- $\acute{\mathbf{x}}$ is a *forward adjoint* variable.
  If $\mathbf{y} = f\;\mathbf{x}$, then $\acute{\mathbf{y}} = \mathcal{J}\; f\;\mathbf{x}\;\acute{\mathbf{x}}$.
- $\grave{\mathbf{x}}$ is a *reverse adjoint* variable.
  If $\mathbf{y} = f\;\mathbf{x}$, then $\grave{\mathbf{x}} = (\mathcal{J}\; f\;\mathbf{x})^\top\;\grave{\mathbf{y}}$.

$$\begin{aligned}
\mathcal{J}\; f\;\mathbf{x} &= (\overrightarrow{\nabla}\; f\;\mathbf{x}, \mathbf{e}_1 \cdots \overrightarrow{\nabla}\; f\;\mathbf{x}, \mathbf{e}_m) \\
&= \begin{pmatrix} \overleftarrow{\nabla}\; f\;\mathbf{x}, \mathbf{e}_1 \\ \vdots \\ \overleftarrow{\nabla}\; f\;\mathbf{x}, \mathbf{e}_n \end{pmatrix}
\end{aligned}$$

Compositional Derivative Operators—III

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

problem: $\overrightarrow{\nabla}$ and $\overleftarrow{\nabla}$ are not compositional.

Compositional Derivative Operators—IV

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

- $\tilde{\mathbf{x}}$ is a *backpropagator* variable.
  If $\mathbf{y} = f\ \mathbf{x}$, then $\tilde{\mathbf{y}}\ \grave{\mathbf{y}} = \grave{\mathbf{x}} = (\mathcal{J}\ f\ \mathbf{x})^{\top}\ \grave{\mathbf{y}}$.

Compositional Derivative Operators—V

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

Compositional Derivative Operators—VI

$$\begin{aligned}
\overrightarrow{\mathcal{J}}\,(g\;f) &= (\overrightarrow{\mathcal{J}}\;g)\;(\overrightarrow{\mathcal{J}}\;f)\\
\overleftarrow{\mathcal{J}}\,(g\;f) &= (\overleftarrow{\mathcal{J}}\;g)\;(\overleftarrow{\mathcal{J}}\;f)\\
\overrightarrow{\mathcal{J}}\,(f_n\;\cdots\;f_1) &= (\overrightarrow{\mathcal{J}}\;f_n)\;\cdots\;(\overrightarrow{\mathcal{J}}\;f_1)\\
\overleftarrow{\mathcal{J}}\,(f_n\;\cdots\;f_1) &= (\overleftarrow{\mathcal{J}}\;f_n)\;\cdots\;(\overleftarrow{\mathcal{J}}\;f_1)
\end{aligned}$$

$$\begin{aligned}
\overrightarrow{\nabla}\;f\;\mathbf{x},\acute{\mathbf{x}} &= \textsc{cdr}\;(\overrightarrow{\mathcal{J}}\;f\;\mathbf{x},\acute{\mathbf{x}})\\
\overleftarrow{\nabla}\;f\;\mathbf{x},\grave{\mathbf{y}} &= \textsc{cdr}\;(\overleftarrow{\mathcal{J}}\;f\;\mathbf{x},I)\;\grave{\mathbf{y}}
\end{aligned}$$

$$\begin{aligned}
\mathcal{J}\;f\;\mathbf{x} &= (\overrightarrow{\nabla}\;f\;\mathbf{x},\mathbf{e}_1\cdots\overrightarrow{\nabla}\;f\;\mathbf{x},\mathbf{e}_m)\\
&= \begin{pmatrix} \overleftarrow{\nabla}\;f\;\mathbf{x},\mathbf{e}_1\\ \vdots \\ \overleftarrow{\nabla}\;f\;\mathbf{x},\mathbf{e}_n \end{pmatrix}
\end{aligned}$$

Traditional Forward-Mode AD

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

Traditional Forward-Mode AD
Sparse Unary Functions—I

$$f_i : \begin{array}{ccccc} \mathbf{x}[1] & & \mathbf{x}[2] & \cdots & \mathbf{x}[n] \\ & \swarrow & \downarrow & \cdots & \downarrow \\ u\,\mathbf{x}[2] & & \mathbf{x}[2] & \cdots & \mathbf{x}[n] \end{array}$$

Traditional Forward-Mode AD
Sparse Unary Functions—II

$$\mathcal{J}\ f_i\ \mathbf{x} = \begin{pmatrix} 0 & \mathcal{D}\ u\ \mathbf{x}[2] & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Traditional Forward-Mode AD
Sparse Unary Functions—III

$$\mathcal{J}\ f_i\ \mathbf{x}\ \acute{\mathbf{x}} = \begin{pmatrix} \mathcal{D}\ u\ \mathbf{x}[2]\ \acute{\mathbf{x}}[2] \\ \acute{\mathbf{x}}[2] \\ \vdots \\ \acute{\mathbf{x}}[n] \end{pmatrix}$$

Traditional Forward-Mode AD
Sparse Unary Functions—IV

$$x_l := u\, x_j \qquad \begin{aligned} x_l &:= u\, x_j \\ \acute{x}_l &:= \mathcal{D}\, u\, x_j\, \acute{x}_j \end{aligned}$$

Traditional Forward-Mode AD
Sparse Binary Functions—I

$$f_i : \begin{array}{cccccc} \mathbf{x}[1] & & \mathbf{x}[2] & \mathbf{x}[3] & \cdots & \mathbf{x}[n] \\ & \swarrow & \downarrow & \downarrow & \cdots & \downarrow \\ b\,\mathbf{x}[2], \mathbf{x}[3] & & \mathbf{x}[2] & \mathbf{x}[3] & \cdots & \mathbf{x}[n] \end{array}$$

Traditional Forward-Mode AD
Sparse Binary Functions—II

$$\mathcal{J}\ f_i\ \mathbf{x} = \begin{pmatrix} 0 & \mathcal{D}_1\ b\ \mathbf{x}[2], \mathbf{x}[3] & \mathcal{D}_2\ b\ \mathbf{x}[2], \mathbf{x}[3] & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Traditional Forward-Mode AD
Sparse Binary Functions—III

$$\mathcal{J}\ f_i\ \mathbf{x}\ \acute{\mathbf{x}} = \left(\begin{array}{c} \mathcal{D}_1\ b\ \mathbf{x}[2], \mathbf{x}[3]\ \acute{\mathbf{x}}[2] + \mathcal{D}_2\ b\ \mathbf{x}[2], \mathbf{x}[3]\ \acute{\mathbf{x}}[3] \\ \acute{\mathbf{x}}[2] \\ \vdots \\ \acute{\mathbf{x}}[n] \end{array}\right)$$

Traditional Forward-Mode AD
Sparse Binary Functions—IV

$$x_l \;:=\; b\,x_j, x_k$$

$$\begin{aligned} x_l &:= b\,x_j, x_k \\ \acute{x}_l &:= \mathcal{D}_1\,b\,x_j, x_k\,\acute{x}_j + \mathcal{D}_2\,b\,x_j, x_k\,\acute{x}_k \end{aligned}$$

Traditional Reverse-Mode AD—I

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

Traditional Reverse-Mode AD—II

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

Traditional Reverse-Mode AD
Sparse Unary Functions—I

$$
f_i : \begin{array}{ccccc}
\mathbf{x}[1] & & \mathbf{x}[2] & \cdots & \mathbf{x}[n] \\
& \swarrow & \downarrow & \cdots & \downarrow \\
u\,\mathbf{x}[2] & & \mathbf{x}[2] & \cdots & \mathbf{x}[n]
\end{array}
$$

Traditional Reverse-Mode AD
Sparse Unary Functions—II

$$\mathcal{J}\; f_i\; \mathbf{x} = \begin{pmatrix} 0 & \mathcal{D}\; u\; \mathbf{x}[2] & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Traditional Reverse-Mode AD
Sparse Unary Functions—III

$$(\mathcal{J}\ f_i\ \mathbf{x})^\top\ \grave{\mathbf{y}} = \begin{pmatrix} 0 \\ \grave{\mathbf{y}}[2] + \mathcal{D}\ u\ \mathbf{x}[2]\ \grave{\mathbf{y}}[1] \\ \grave{\mathbf{y}}[3] \\ \vdots \\ \grave{\mathbf{y}}[n] \end{pmatrix}$$

Traditional Reverse-Mode AD
Sparse Unary Functions—IV

$$
x_l \quad := \quad u\, x_j
\qquad\qquad
\begin{array}{rcl}
x_l & := & u\, x_j \\
 & \vdots & \\
\grave{x}_l & := & 0 \\
\grave{x}_j & := & \grave{x}_j + \mathcal{D}\, u\, x_j\, \grave{x}_l
\end{array}
$$

Traditional Reverse-Mode AD
Sparse Binary Functions—I

$$
f_i : \quad \begin{array}{cccccc}
\mathbf{x}[1] & & \mathbf{x}[2] & \mathbf{x}[3] & \cdots & \mathbf{x}[n] \\
 & \swarrow & \downarrow & \downarrow & \cdots & \downarrow \\
b\,\mathbf{x}[2], \mathbf{x}[3] & & \mathbf{x}[2] & \mathbf{x}[3] & \cdots & \mathbf{x}[n]
\end{array}
$$

Traditional Reverse-Mode AD
Sparse Binary Functions—II

$$\mathcal{J}\ f_i\ \mathbf{x} = \begin{pmatrix} 0 & \mathcal{D}_1\ b\ \mathbf{x}[2], \mathbf{x}[3] & \mathcal{D}_2\ b\ \mathbf{x}[2], \mathbf{x}[3] & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Traditional Reverse-Mode AD
Sparse Binary Functions—III

$$(\mathcal{J}\ f_i\ \mathbf{x})^{\top}\ \grave{\mathbf{y}} = \begin{pmatrix} 0 \\ \grave{\mathbf{y}}[2] + \mathcal{D}_1\ b\ \mathbf{x}[2], \mathbf{x}[3]\ \grave{\mathbf{y}}[1] \\ \grave{\mathbf{y}}[3] + \mathcal{D}_2\ b\ \mathbf{x}[2], \mathbf{x}[3]\ \grave{\mathbf{y}}[1] \\ \grave{\mathbf{x}}[4] \\ \vdots \\ \grave{\mathbf{x}}[n] \end{pmatrix}$$

Traditional Reverse-Mode AD
Sparse Binary Functions—IV

$$
\begin{array}{rcl@{\qquad}rcl}
x_l & := & b\, x_j, x_k & x_l & := & b\, x_j, x_k \\
 & & & & \vdots & \\
 & & & \grave{x}_l & := & 0 \\
 & & & \grave{x}_j & := & \grave{x}_j + \mathcal{D}_1\, b\, x_j, x_k\, \grave{x}_l \\
 & & & \grave{x}_k & := & \grave{x}_k + \mathcal{D}_2\, b\, x_j, x_k\, \grave{x}_l
\end{array}
$$

Jacobians of CAR, CDR, and CONS—I

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

Jacobians of CAR, CDR, and CONS—II

$$
\begin{array}{rcl}
\mathcal{J}\ \textsc{car}\ \mathbf{x} & = & (1\ 0)\\
\mathcal{J}\ \textsc{cdr}\ \mathbf{x} & = & (0\ 1)\\
\mathcal{J}\ \textsc{car}\ \mathbf{x}\ \acute{\mathbf{x}} & = & \acute{\mathbf{x}}[1]\\
\mathcal{J}\ \textsc{cdr}\ \mathbf{x}\ \acute{\mathbf{x}} & = & \acute{\mathbf{x}}[2]\\
(\mathcal{J}\ \textsc{car}\ \mathbf{x})^\top\ \grave{y} & = & \begin{pmatrix} \grave{y} \\ 0 \end{pmatrix}\\
(\mathcal{J}\ \textsc{cdr}\ \mathbf{x})^\top\ \grave{y} & = & \begin{pmatrix} 0 \\ \grave{y} \end{pmatrix}
\end{array}
$$

Jacobians of CAR, CDR, and CONS—III

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

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:

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

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

VLAD: Functional Language for AD—II

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

VLAD: <u>F</u>unctional <u>L</u>anguage for <u>AD</u>—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 $\textsc{cons}\ e_1\ e_2$.
- We allow lambda expressions to have tuples as parameters as shorthand for the appropriate destructuring. For example:

$$\lambda x_1, x_2\ e \ \rightsquigarrow\ \lambda x\ \mathbf{let}\ x_1 \triangleq \textsc{car}\ x; x_2 \triangleq \textsc{cdr}\ x\ \mathbf{in}\ e\ \mathbf{end}$$

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

Adjoint Types

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

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

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

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

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

$$
\begin{aligned}
\overrightarrow{\mathcal{J}}\ u &\triangleq \lambda x, \acute{x}\ (u\ x), (\mathcal{D}\ u\ x\ \acute{x})\\
\overrightarrow{\mathcal{J}}\ b &\triangleq \lambda(x_1, x_2), (\acute{x}_1, \acute{x}_2)\\
&\quad\ (b\ x_1, x_2), (\mathcal{D}_1\ b\ x_1, x_2\ \acute{x}_1 + \mathcal{D}_2\ b\ x_1, x_2\ \acute{x}_2)\\
\overrightarrow{\mathcal{J}}\ p &\triangleq \lambda x, \acute{x}\ (p\ x), [\,]\\
\overrightarrow{\mathcal{J}}\ \text{CAR} &\triangleq \lambda(x_1, x_2), (\acute{x}_1, \acute{x}_2)\ x_1, \acute{x}_1\\
\overrightarrow{\mathcal{J}}\ \text{CDR} &\triangleq \lambda(x_1, x_2), (\acute{x}_1, \acute{x}_2)\ x_2, \acute{x}_2\\
\overrightarrow{\mathcal{J}}\ \text{CONS} &\triangleq \lambda x_1, \acute{x}_1\ (\lambda x_2, \acute{x}_2\ (x_1, x_2), (\acute{x}_1, \acute{x}_2)), [\,]\\
\overrightarrow{\mathcal{J}}\ \text{IF} &\triangleq \lambda x_1, \acute{x}_1\\
&\quad\ (\lambda x_2, \acute{x}_2\ (\lambda x_3, \acute{x}_3\ \textbf{if}\ x_1\ \textbf{then}\ x_2, \acute{x}_2\ \textbf{else}\ x_3, \acute{x}_3\ \textbf{fi}), [\,]), [\,]
\end{aligned}
$$

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

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

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

$$\begin{array}{ll}
\underline{0}\ x \stackrel{\triangle}{=} & \textbf{if } \textsc{real}?\ x \textbf{ then } 0\\
& \textbf{elif } \textsc{pair}?\ x \textbf{ then } (\underline{0}\ (\textsc{car}\ x)), (\underline{0}\ (\textsc{cdr}\ x))\\
& \textbf{else } [\,] \textbf{ fi}
\end{array}$$

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

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

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

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

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

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

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

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

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

$$\begin{array}{rl}
x_1 \oplus x_2 \stackrel{\triangle}{=} & \textbf{if } \textsc{null}?\ x_1 \textbf{ then } [\,] \\
& \textbf{elif } \textsc{real}?\ x_1 \textbf{ then } x_1 + x_2 \\
& \textbf{else } (\textsc{car}\ x_1 \oplus \textsc{car}\ x_2), (\textsc{cdr}\ x_1 \oplus \textsc{cdr}\ x_2) \textbf{ fi}
\end{array}$$

Fanout—The Problem

$$
\begin{array}{lrcl}
\lambda x_0 \;\textbf{let} & x_1 & \stackrel{\triangle}{=} & x_0 + x_0; \\
 & x_2 & \stackrel{\triangle}{=} & x_1 + x_1; \\
 & & \vdots & \\
 & x_n & \stackrel{\triangle}{=} & x_{n-1} + x_{n-1} \\
 & \multicolumn{3}{l}{\textbf{in}\; x_n \;\textbf{end}}
\end{array}
$$

Fanout—One Solution

$$\textsc{fan}\ f\ x \stackrel{\triangle}{=} f\ x, x$$

$$\lambda x\ x + x + x \quad \rightsquigarrow \quad \lambda x\ \textsc{fan}\ (\lambda x_1, x\ \textsc{fan}\ (\lambda x_2, x_3\ x_1 + x_2 + x_3)\ x)\ x$$

$$\begin{aligned}\overleftarrow{\mathcal{J}}\ \textsc{fan} \stackrel{\triangle}{=} \lambda f, \tilde{f}\ (\lambda x, \tilde{x}\ &\mathbf{let}\ y, \tilde{y} \stackrel{\triangle}{=} f\ (x, x), I\\ &\mathbf{in}\ y, \lambda \grave{y}\ \mathbf{let}\ \grave{x} \stackrel{\triangle}{=} \tilde{y}\ \grave{y}\\ &\qquad\qquad \mathbf{in}\ \tilde{x}\ (\textsc{car}\ \grave{x} \oplus \textsc{cdr}\ \grave{x})\ \mathbf{end}\ \mathbf{end}),\\ \lambda \grave{y}\ \tilde{f}\ (\underline{0}\ f)\end{aligned}$$

Derivatives

$$\mathcal{D}\ f\ x \quad \stackrel{\triangle}{=} \quad \textsc{cdr}\ (\overrightarrow{\mathcal{J}}\ f\ x, 1)$$
$$\mathcal{D}\ f\ x \quad \stackrel{\triangle}{=} \quad \textsc{cdr}\ (\overleftarrow{\mathcal{J}}\ f\ x, I)\ 1$$

Roots using Newton-Raphson

$$
\begin{aligned}
\textsc{Root}\ f, x, \epsilon \triangleq\ & \mathbf{let}\ x' \triangleq x - \tfrac{f\ x}{\mathcal{D}\ f\ x} \\
& \mathbf{in\ if}\ |x - x'| \leq \epsilon \\
& \quad \mathbf{then}\ x \\
& \quad \mathbf{else}\ \textsc{Root}\ f, x', \epsilon\ \mathbf{fi\ end}
\end{aligned}
$$

Univariate Optimizer (Line Search)

$$\textsc{Argmin}\ f, x, \epsilon \stackrel{\triangle}{=} \textsc{Root}\ (\mathcal{D}\ f), x, \epsilon$$

Gradients

$$
\begin{aligned}
\nabla\ f\ x &\stackrel{\triangle}{=} \textbf{let}\ n \stackrel{\triangle}{=} \textsc{Length}\ x\\
&\quad\ \textbf{in}\ \textsc{Map}\ (\lambda i\ \textsc{cdr}\ (\overrightarrow{\mathcal{J}}\ f\ x, (e\ 1, i, n))),\\
&\qquad\qquad (\iota\ n)\ \textbf{end}\\
\nabla\ f\ x &\stackrel{\triangle}{=} \textsc{cdr}\ (\overleftarrow{\mathcal{J}}\ f\ x, I)\ 1
\end{aligned}
$$

Gradient Descent

$\textsc{GradientDescent}\ f, x, \epsilon \stackrel{\triangle}{=}$
  let $g \stackrel{\triangle}{=} \nabla\ f\ x$
  in if $||g|| \leq \epsilon$
    then x
    else $\textsc{GradientDescent}$
      $f, (x + \textsc{Argmin}\ (\lambda k\ f\ (x + kg)), 0, \epsilon\ g), \epsilon$ **fi end**

Function Inversion

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

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

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

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

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

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)

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

$\textsc{GradientAscent}\ L_X, \theta_0, \epsilon$

Engineering Design

$$
\begin{aligned}
&\textsc{performance}\ \textsc{SplineControlPoints} \triangleq \\
&\quad \textbf{let}\ \textsc{wing} \triangleq \textsc{SplineToSurface}\ \textsc{SplineControlPoints}; \\
&\qquad \textsc{airflow} \triangleq \text{PDE}\textsc{solver}\ \textsc{wing}, \textsc{NavierStokes}; \\
&\qquad \textsc{lift}, \textsc{drag} \triangleq \textsc{SurfaceIntegral}\ \textsc{wing}, \textsc{airflow}, \textsc{force} \\
&\quad \textbf{in}\ \textsc{DesignMetric}\ \textsc{lift}, \textsc{drag}, (\textsc{weight}\ \textsc{wing})\ \textbf{end}
\end{aligned}
$$

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

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

Contributions—I

AD operators are first-class function objects.

Contributions—II

The arguments and results of AD operators are first-class function objects.

Contributions—III

AD operators generalized to arbitrary types.

$$
\begin{aligned}
\overrightarrow{\mathcal{J}}: &\quad (\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow ((\mathbb{R}^n \times \mathbb{R}^n) \rightarrow (\mathbb{R}^m \times \mathbb{R}^m))\\
\overleftarrow{\mathcal{J}}: &\quad (\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow\\
&\quad ((\mathbb{R}^n \times (\mathbb{R}^n \rightarrow \mathbb{R}^l)) \rightarrow (\mathbb{R}^m \times (\mathbb{R}^m \rightarrow \mathbb{R}^l)))
\end{aligned}
$$

$$
\begin{aligned}
\overrightarrow{\mathcal{J}}: &\quad (\tau_1 \rightarrow \tau_2) \rightarrow ((\overrightarrow{\tau_1} \times \overline{\tau_1}) \rightarrow (\overrightarrow{\tau_2} \times \overline{\tau_2}))\\
\overleftarrow{\mathcal{J}}: &\quad (\tau_1 \rightarrow \tau_2) \rightarrow\\
&\quad ((\overleftarrow{\tau_1} \times (\overline{\tau_1} \rightarrow \overline{\tau_3})) \rightarrow (\overleftarrow{\tau_2} \times (\overline{\tau_2} \rightarrow \overline{\tau_3})))
\end{aligned}
$$

1. trees
2. discrete values
3. functions

Contributions—IV

AD operators apply to trees (that contain functions).

$$\overrightarrow{\mathcal{J}} : (\tau_1 \rightarrow \tau_2) \rightarrow \overrightarrow{(\tau_1 \rightarrow \tau_2)}$$
$$\overleftarrow{\mathcal{J}} : (\tau_1 \rightarrow \tau_2) \rightarrow \overleftarrow{(\tau_1 \rightarrow \tau_2)}$$

$$\overrightarrow{\mathcal{J}} : \tau \rightarrow \overrightarrow{\tau}$$
$$\overleftarrow{\mathcal{J}} : \tau \rightarrow \overleftarrow{\tau}$$

Contributions—V

The ‘tape’ of traditional reverse-mode AD is represented as closures and tail calls.

Contributions—VI

Closure property: Can take the derivative of any lambda expression over a basis with known derivatives. That derivative is, in turn, a lambda expression over the same basis.

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.