Microsoft Research

Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available.

The simple essence of automatic differentiation

Conal Elliott

Target

January/June 2018

What's a derivative?

- Number
- Vector
- Covector
- Matrix
- Higher derivatives

Chain rule for each.

What's a derivative?

$$\mathcal{D} :: (a \to b) \to (a \to (a \multimap b))$$

where

$$\lim_{\varepsilon \to 0} \frac{\| f\ (a + \varepsilon) - (f\ a + \mathcal{D}\ f\ a\ \varepsilon) \|}{\| \varepsilon \|} = 0$$

See *Calculus on Manifolds* by Michael Spivak.

Composition

Sequential:

$$(\circ) :: (b \to c) \to (a \to b) \to (a \to c)$$
$$(g \circ f)\ a = g\ (f\ a)$$

$$\mathcal{D}\ (g \circ f)\ a = \mathcal{D}\ g\ (f\ a) \circ \mathcal{D}\ f\ a \quad \text{-- ``chain rule''}$$

What's a derivative?

$$\mathcal{D} :: (a \to b) \to (a \to (a \multimap b))$$

where

$$\lim_{\varepsilon \to 0} \frac{\|f\ (a + \varepsilon) - (f\ a + \mathcal{D}\ f\ a\ \varepsilon)\|}{\|\varepsilon\|} = 0$$

See *Calculus on Manifolds* by Michael Spivak.

Composition

Sequential:

$$(\circ) :: (b \to c) \to (a \to b) \to (a \to c)$$
$$(g \circ f)\ a = g\ (f\ a)$$

$$\mathcal{D}\ (g \circ f)\ a = \mathcal{D}\ g\ (f\ a) \circ \mathcal{D}\ f\ a \quad \text{-- ``chain rule''}$$

Parallel:

$$(\vartriangle) :: (a \to c) \to (a \to d) \to (a \to c \times d)$$
$$(f \vartriangle g)\ a = (f\ a, g\ a)$$

$$\mathcal{D}\ (f \vartriangle g)\ a = \mathcal{D}\ f\ a \vartriangle \mathcal{D}\ g\ a$$

Compositionality

Chain rule:

$$\mathcal{D}\ (g \circ f)\ a = \mathcal{D}\ g\ (f\ a) \circ \mathcal{D}\ f\ a \quad \text{-- non-compositional}$$

To fix, combine regular result with derivative:

$$\hat{\mathcal{D}} :: (a \to b) \to (a \to (b \times (a \multimap b)))$$
$$\hat{\mathcal{D}}\ f = f \vartriangle \mathcal{D}\ f \quad \text{-- specification}$$

Often much work in common to f and $\mathcal{D}\ f$.

Linear functions

Linear functions are their own perfect linear approximations.

$$
\begin{aligned}
\mathcal{D}\ \mathit{id}\quad a &= \mathit{id} \\
\mathcal{D}\ \mathit{fst}\ \ a &= \mathit{fst} \\
\mathcal{D}\ \mathit{snd}\ a &= \mathit{snd} \\
&\ldots
\end{aligned}
$$

For linear functions f,

$$\hat{\mathcal{D}}\ f\ a = (f\ a, f)$$

Abstract algebra for functions

$$
\begin{array}{l}
\textbf{class } \mathit{Category}\ (\leadsto)\ \textbf{where} \\
\quad \mathit{id}\ \ :: a \leadsto a \\
\quad (\circ) :: (b \leadsto c) \to (a \leadsto b) \to (a \leadsto c)
\end{array}
$$

$$
\begin{array}{l}
\textbf{class } \mathit{Category}\ (\leadsto) \Rightarrow \mathit{Cartesian}\ (\leadsto)\ \textbf{where} \\
\quad \mathit{exl} :: (a \times b) \leadsto a \\
\quad \mathit{exr} :: (a \times b) \leadsto b \\
\quad (\vartriangle) :: (a \leadsto c) \to (a \leadsto d) \to (a \leadsto (c \times d))
\end{array}
$$

Plus laws and classes for arithmetic etc.

Linear functions

Linear functions are their own perfect linear approximations.

$$
\begin{aligned}
\mathcal{D}\ id\ \ \ a &= id \\
\mathcal{D}\ fst\ \ a &= fst \\
\mathcal{D}\ snd\ a &= snd \\
&\ldots
\end{aligned}
$$

For linear functions f,

$$\hat{\mathcal{D}}\ f\ a = (f\ a, f)$$

Abstract algebra for functions

$$
\begin{array}{l}
\textbf{class } \mathit{Category}\ (\leadsto)\ \textbf{where} \\
\quad \mathit{id}\ \ :: a \leadsto a \\
\quad (\circ) :: (b \leadsto c) \to (a \leadsto b) \to (a \leadsto c)
\end{array}
$$

$$
\begin{array}{l}
\textbf{class } \mathit{Category}\ (\leadsto) \Rightarrow \mathit{Cartesian}\ (\leadsto)\ \textbf{where} \\
\quad \mathit{exl}\ :: (a \times b) \leadsto a \\
\quad \mathit{exr} :: (a \times b) \leadsto b \\
\quad (\vartriangle) :: (a \leadsto c) \to (a \leadsto d) \to (a \leadsto (c \times d))
\end{array}
$$

Plus laws and classes for arithmetic etc.

Automatic differentiation

$$\textbf{newtype}\ D\ a\ b = D\ (a \to b \times (a \multimap b))$$

$$\hat{\mathcal{D}} :: (a \to b) \to D\ a\ b$$
$$\hat{\mathcal{D}}\ f = D\ (f \vartriangle \mathcal{D}\ f) \quad \text{-- not computable}$$

Require $\hat{\mathcal{D}}$ to preserve *Category* and *Cartesian* structure:

$$\hat{\mathcal{D}}\ id = id$$
$$\hat{\mathcal{D}}\ (g \circ f) = \hat{\mathcal{D}}\ g \circ \hat{\mathcal{D}}\ f$$

$$\hat{\mathcal{D}}\ exl = exl$$
$$\hat{\mathcal{D}}\ exr = exr$$
$$\hat{\mathcal{D}}\ (f \vartriangle g) = \hat{\mathcal{D}}\ f \vartriangle \hat{\mathcal{D}}\ g$$

The game: solve these equations for the RHS operations.

Solution: simple automatic differentiation

$$\textbf{newtype}\ D\ a\ b = D\ (a \to b \times (a \multimap b))$$

$$\mathit{linearD}\ f = D\ (\lambda a \to (f\ a, a))$$

$\textbf{instance}\ \mathit{Category}\ D\ \textbf{where}$

$\quad \mathit{id} = \mathit{linearD}\ \mathit{id}$

$\quad D\ g \circ D\ f = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))$

$\textbf{instance}\ \mathit{Cartesian}\ D\ \textbf{where}$

$\quad \mathit{exl} = \mathit{linearD}\ \mathit{exl}$

$\quad \mathit{exr} = \mathit{linearD}\ \mathit{exr}$

$\quad D\ f \vartriangle D\ g = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))$

$\textbf{instance}\ \mathit{NumCat}\ D\ \textbf{where}$

$\quad \mathit{negate} = \mathit{linearD}\ \mathit{negate}$

$\quad \mathit{add} = \mathit{linearD}\ \mathit{add}$

$\quad \mathit{mul} = D\ (\mathit{mul} \vartriangle (\lambda(a, b) \to \lambda(da, db) \to b * da + a * db))$

Automatic differentiation

$$\textbf{newtype}\ D\ a\ b = D\ (a \to b \times (a \multimap b))$$

$$\hat{\mathcal{D}} :: (a \to b) \to D\ a\ b$$
$$\hat{\mathcal{D}}\ f = D\ (f \vartriangle \mathcal{D}\ f) \quad \text{-- not computable}$$

Require $\hat{\mathcal{D}}$ to preserve *Category* and *Cartesian* structure:

$$\hat{\mathcal{D}}\ id = id$$
$$\hat{\mathcal{D}}\ (g \circ f) = \hat{\mathcal{D}}\ g \circ \hat{\mathcal{D}}\ f$$

$$\hat{\mathcal{D}}\ exl = exl$$
$$\hat{\mathcal{D}}\ exr = exr$$
$$\hat{\mathcal{D}}\ (f \vartriangle g) = \hat{\mathcal{D}}\ f \vartriangle \hat{\mathcal{D}}\ g$$

The game: solve these equations for the RHS operations.

Solution: simple automatic differentiation

$\mathbf{newtype}\ D\ a\ b = D\ (a \rightarrow b \times (a \multimap b))$

$\mathit{linearD}\ f = D\ (\lambda a \rightarrow (f\ a, a))$

$\mathbf{instance}\ \mathit{Category}\ D\ \mathbf{where}$
$\quad \mathit{id} = \mathit{linearD}\ \mathit{id}$
$\quad D\ g \circ D\ f = D\ (\lambda a \rightarrow \mathbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \mathbf{in}\ (c, g' \circ f'))$

$\mathbf{instance}\ \mathit{Cartesian}\ D\ \mathbf{where}$
$\quad \mathit{exl} = \mathit{linearD}\ \mathit{exl}$
$\quad \mathit{exr} = \mathit{linearD}\ \mathit{exr}$
$\quad D\ f \vartriangle D\ g = D\ (\lambda a \rightarrow \mathbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \mathbf{in}\ ((b, c), f' \vartriangle g'))$

$\mathbf{instance}\ \mathit{NumCat}\ D\ \mathbf{where}$
$\quad \mathit{negate} = \mathit{linearD}\ \mathit{negate}$
$\quad \mathit{add} = \mathit{linearD}\ \mathit{add}$
$\quad \mathit{mul} = D\ (\mathit{mul} \vartriangle (\lambda(a, b) \rightarrow \lambda(da, db) \rightarrow b * da + a * db))$

Running examples

$$sqr :: Num\ a \Rightarrow a \rightarrow a$$
$$sqr\ a = a * a$$

$$magSqr :: Num\ a \Rightarrow a \times a \rightarrow a$$
$$magSqr\ (a, b) = sqr\ a + sqr\ b$$

$$cosSinProd :: Floating\ a \Rightarrow a \times a \rightarrow a \times a$$
$$cosSinProd\ (x, y) = (cos\ z, sin\ z)\ \mathbf{where}\ z = x * y$$

In categorical vocabulary:

$$sqr = mul \circ (id \vartriangle id)$$

$$magSqr = add \circ (mul \circ (exl \vartriangle exl) \vartriangle mul \circ (exr \vartriangle exr))$$

$$cosSinProd = (cos \vartriangle sin) \circ mul$$

Solution: simple automatic differentiation

$$\textbf{newtype}\ D\ a\ b = D\ (a \to b \times (a \multimap b))$$

$$linearD\ f = D\ (\lambda a \to (f\ a, a))$$

$\textbf{instance}\ Category\ D\ \textbf{where}$

$\quad id = linearD\ id$

$\quad D\ g \circ D\ f = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))$

$\textbf{instance}\ Cartesian\ D\ \textbf{where}$

$\quad exl = linearD\ exl$

$\quad exr = linearD\ exr$

$\quad D\ f \vartriangle D\ g = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))$

$\textbf{instance}\ NumCat\ D\ \textbf{where}$

$\quad negate = linearD\ negate$

$\quad add = linearD\ add$

$\quad mul = D\ (mul \vartriangle (\lambda(a, b) \to \lambda(da, db) \to b * da + a * db))$

Running examples

$$sqr :: Num\ a \Rightarrow a \rightarrow a$$
$$sqr\ a = a * a$$

$$magSqr :: Num\ a \Rightarrow a \times a \rightarrow a$$
$$magSqr\ (a, b) = sqr\ a + sqr\ b$$

$$cosSinProd :: Floating\ a \Rightarrow a \times a \rightarrow a \times a$$
$$cosSinProd\ (x, y) = (cos\ z, sin\ z)\ \mathbf{where}\ z = x * y$$

In categorical vocabulary:

$$sqr = mul \circ (id \vartriangle id)$$

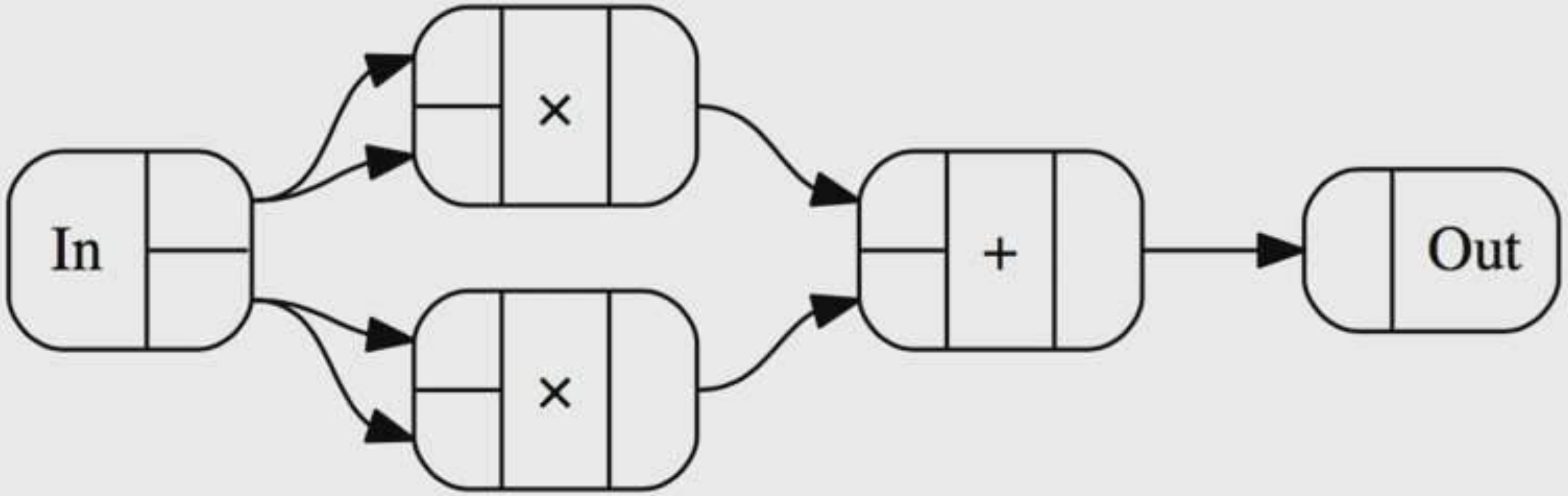
$$magSqr = add \circ (mul \circ (exl \vartriangle exl) \vartriangle mul \circ (exr \vartriangle exr))$$

$$cosSinProd = (cos \vartriangle sin) \circ mul$$

Visualizing computations

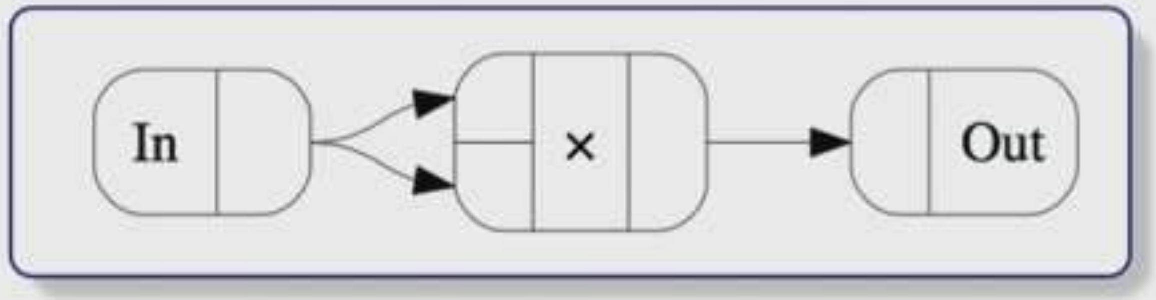
$$magSqr\ (a, b) = sqr\ a + sqr\ b$$

$$magSqr = add \circ (mul \circ (exl \vartriangle exl) \vartriangle mul \circ (exr \vartriangle exr))$$



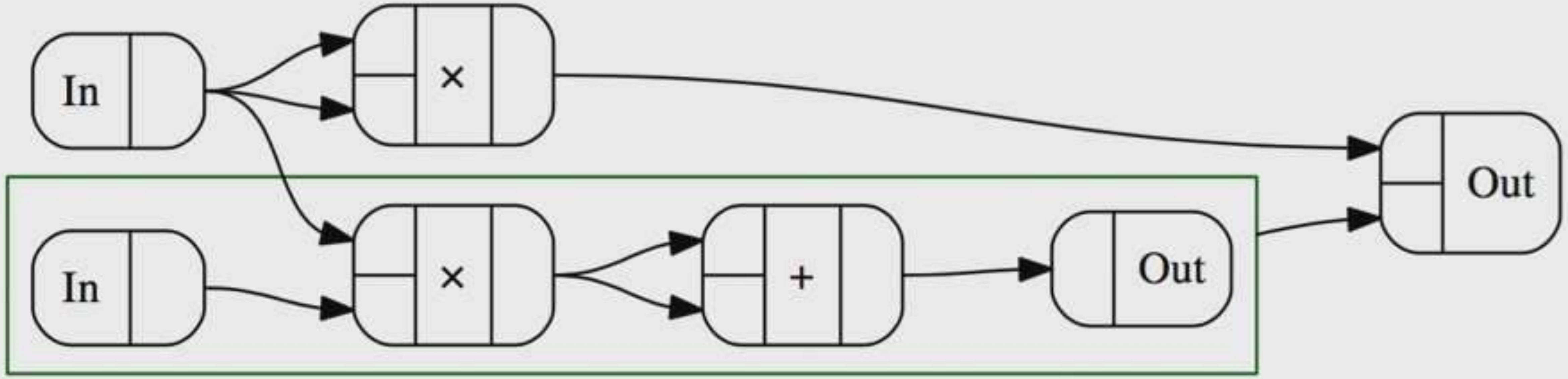
Auto-generated from Haskell code. See *Compiling to categories*.

AD example

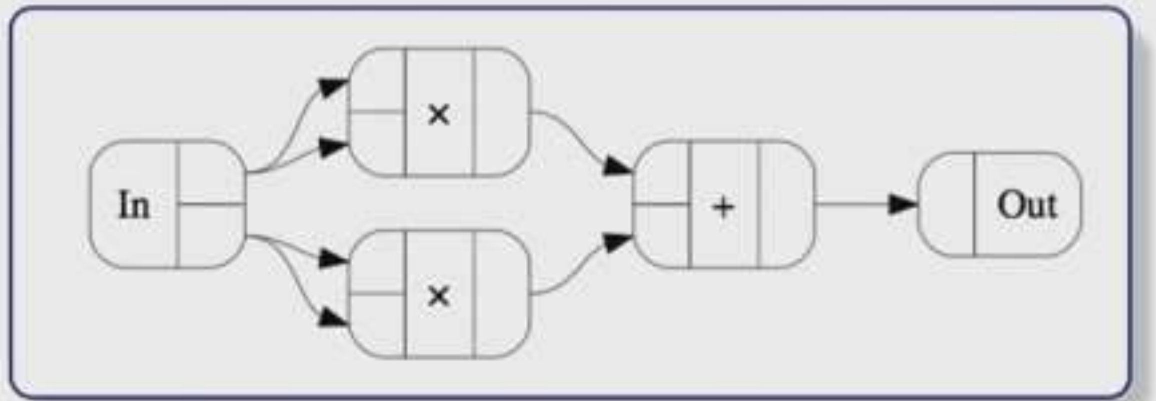


$$sqr\ a = a * a$$

$$sqr = mul \circ (id \vartriangle id)$$

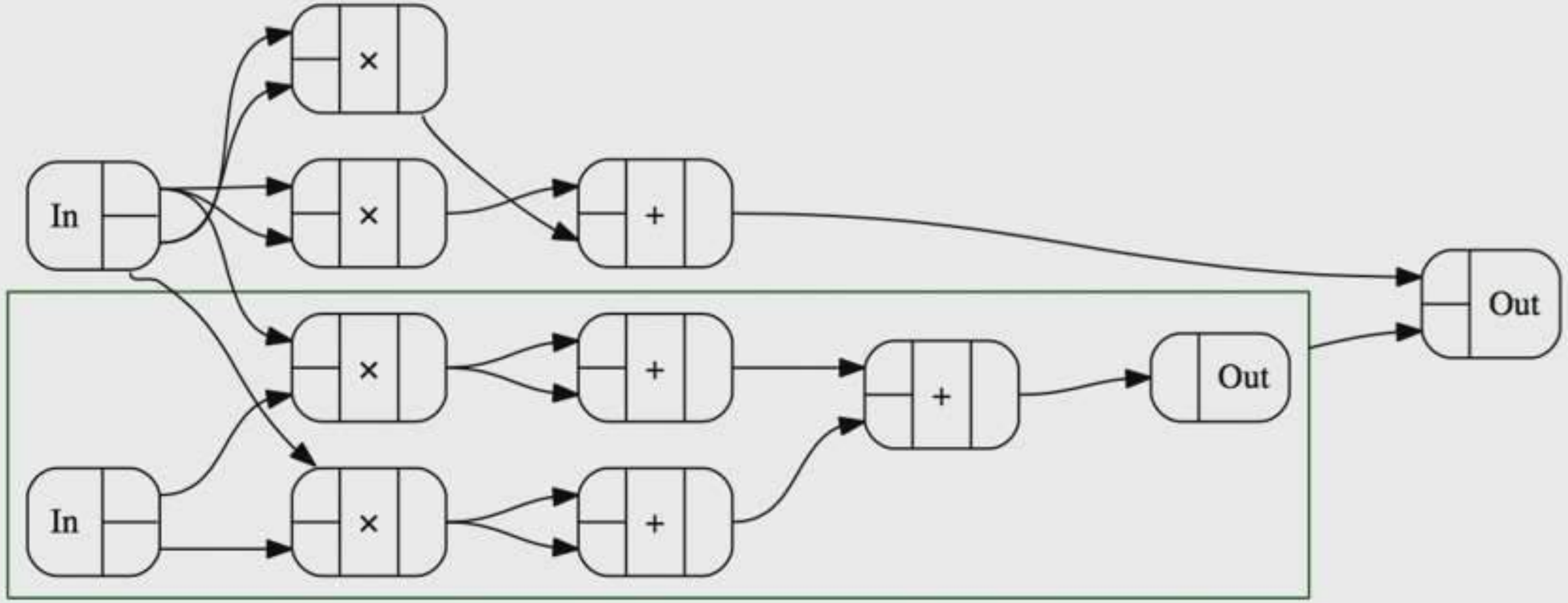


AD example

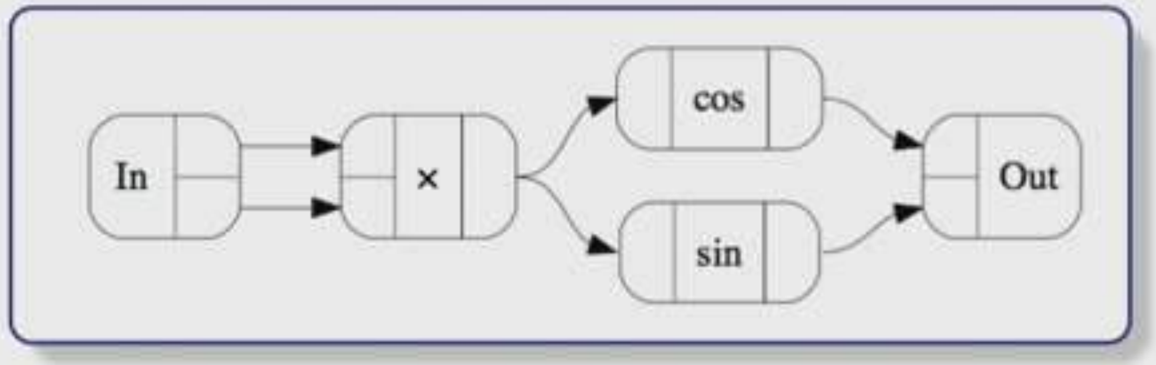


$$magSqr\ (a, b) = sqr\ a + sqr\ b$$

$$magSqr = add \circ (mul \circ (exl \vartriangle exl) \vartriangle mul \circ (exr \vartriangle exr))$$

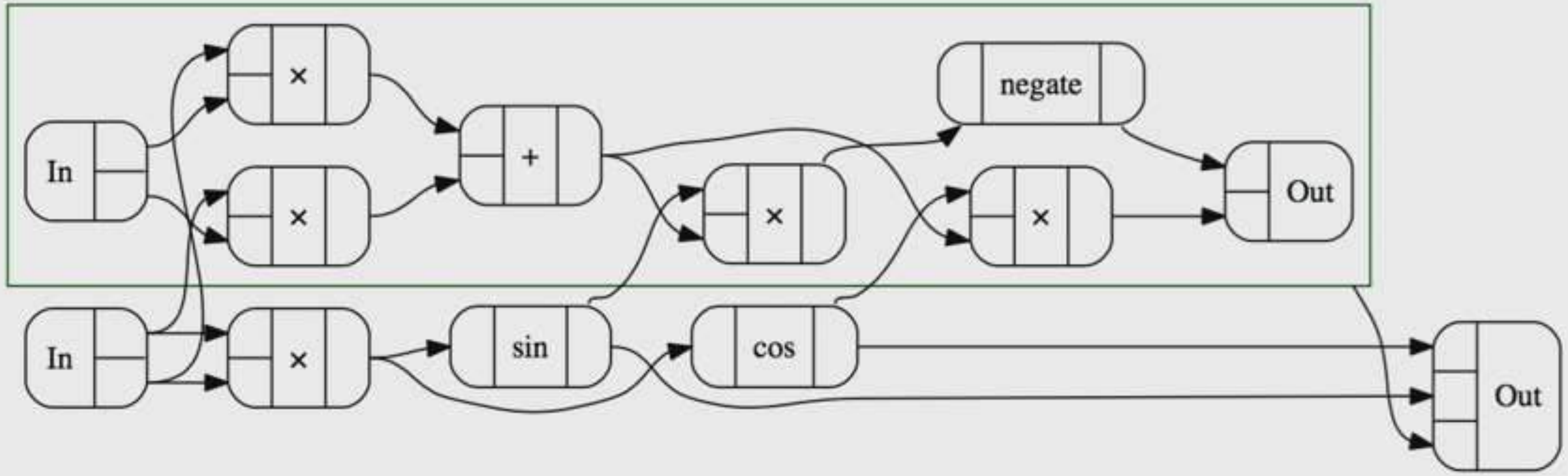


AD example



$$cosSinProd\ (x, y) = (cos\ z, sin\ z)\ \textbf{where}\ z = x * y$$

$$cosSinProd = (cos \vartriangle sin) \circ mul$$



Generalizing AD

$$\textbf{newtype}\ D\ a\ b = D\ (a \to b \times (a \multimap b))$$

$$linearD\ f = D\ (\lambda a \to (f\ a, f))$$

$$\begin{array}{l}\textbf{instance}\ Category\ D\ \textbf{where} \\ \quad id = linearD\ id \\ \quad D\ g \circ D\ f = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))\end{array}$$

$$\begin{array}{l}\textbf{instance}\ Cartesian\ D\ \textbf{where} \\ \quad exl = linearD\ exl \\ \quad exr = linearD\ exr \\ \quad D\ f \vartriangle D\ g = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))\end{array}$$

Each D operation just uses corresponding $(\multimap)$ operation.

Generalize from $(\multimap)$ to other cartesian categories.

Generalized AD

$\textbf{newtype}\ D_{(\rightsquigarrow)}\ a\ b = D\ (a \rightarrow b \times (a \rightsquigarrow b))$

$linearD\ f\ f' = D\ (\lambda a \rightarrow (f\ a, f'))$

$\textbf{instance}\ Category\ (\rightsquigarrow) \Rightarrow Category\ D_{(\rightsquigarrow)}\ \textbf{where}$
$\quad id = linearD\ id\ id$
$\quad D\ g \circ D\ f = D\ (\lambda a \rightarrow \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))$

$\textbf{instance}\ Cartesian\ (\rightsquigarrow) \Rightarrow Cartesian\ D_{(\rightsquigarrow)}\ \textbf{where}$
$\quad exl = linearD\ exl\ exl$
$\quad exr = linearD\ exr\ exr$
$\quad D\ f \vartriangle D\ g = D\ (\lambda a \rightarrow \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))$

$\textbf{instance} \ldots \Rightarrow NumCat\ D\ \textbf{where}$
$\quad negate = linearD\ negate\ negate$
$\quad add = linearD\ add\ add$
$\quad mul = ??$

Generalizing AD

$$\textbf{newtype}\ D\ a\ b = D\ (a \to b \times (a \multimap b))$$

$$\mathit{linearD}\ f = D\ (\lambda a \to (f\ a, f))$$

$$\begin{array}{l}\textbf{instance}\ \mathit{Category}\ D\ \textbf{where} \\ \quad \mathit{id} = \mathit{linearD}\ \mathit{id} \\ \quad D\ g \circ D\ f = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))\end{array}$$

$$\begin{array}{l}\textbf{instance}\ \mathit{Cartesian}\ D\ \textbf{where} \\ \quad \mathit{exl} = \mathit{linearD}\ \mathit{exl} \\ \quad \mathit{exr} = \mathit{linearD}\ \mathit{exr} \\ \quad D\ f \vartriangle D\ g = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))\end{array}$$

Each D operation just uses corresponding $(\multimap)$ operation.

Generalize from $(\multimap)$ to other cartesian categories.

Generalized AD

$\textbf{newtype}\ D_{(\leadsto)}\ a\ b = D\ (a \to b \times (a \leadsto b))$

$linearD\ f\ f' = D\ (\lambda a \to (f\ a, f'))$

$\textbf{instance}\ Category\ (\leadsto) \Rightarrow Category\ D_{(\leadsto)}\ \textbf{where}$

$\quad id = linearD\ id\ id$

$\quad D\ g \circ D\ f = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))$

$\textbf{instance}\ Cartesian\ (\leadsto) \Rightarrow Cartesian\ D_{(\leadsto)}\ \textbf{where}$

$\quad exl = linearD\ exl\ exl$

$\quad exr = linearD\ exr\ exr$

$\quad D\ f \vartriangle D\ g = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))$

$\textbf{instance}... \Rightarrow NumCat\ D\ \textbf{where}$

$\quad negate = linearD\ negate\ negate$

$\quad add = linearD\ add\ add$

$\quad mul = ??$

Generalizing AD

$$\textbf{newtype}\ D\ a\ b = D\ (a \to b \times (a \multimap b))$$

$$linearD\ f = D\ (\lambda a \to (f\ a, f))$$

$\textbf{instance}\ Category\ D\ \textbf{where}$
$\quad id = linearD\ id$
$\quad D\ g \circ D\ f = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))$

$\textbf{instance}\ Cartesian\ D\ \textbf{where}$
$\quad exl = linearD\ exl$
$\quad exr = linearD\ exr$
$\quad D\ f \vartriangle D\ g = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))$

Each D operation just uses corresponding $(\multimap)$ operation.

Generalize from $(\multimap)$ to other cartesian categories.

Generalized AD

$\textbf{newtype}\ D_{(\leadsto)}\ a\ b = D\ (a \to b \times (a \leadsto b))$

$linearD\ f\ f' = D\ (\lambda a \to (f\ \ a, f'))$

$\textbf{instance}\ Category\ (\leadsto) \Rightarrow Category\ D_{(\leadsto)}\ \textbf{where}$
$\quad id = linearD\ id\ id$
$\quad D\ g \circ D\ f = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ \ a; (c, g') = g\ \ b\}\ \textbf{in}\ (c, g' \circ f'))$

$\textbf{instance}\ Cartesian\ (\leadsto) \Rightarrow Cartesian\ D_{(\leadsto)}\ \textbf{where}$
$\quad exl\ = linearD\ exl\ exl$
$\quad exr = linearD\ exr\ exr$
$\quad D\ f \vartriangle D\ g = D\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ \ a; (c, g') = g\ \ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))$

$\textbf{instance}... \Rightarrow NumCat\ D\ \textbf{where}$
$\quad negate = linearD\ negate\ negate$
$\quad add\ = linearD\ add\ add$
$\quad mul = ??$

Numeric operations

Specific to (linear) *functions*:

$$mul = D\ (mul \vartriangle (\lambda(a, b) \rightarrow \lambda(da, db) \rightarrow b * da + a * db))$$

Rephrase:

$$\begin{aligned}&scale :: Multiplicative\ a \Rightarrow a \rightarrow (a \multimap a)\\&scale\ u = \lambda v \rightarrow u * v\end{aligned}$$

$$\begin{aligned}&(\triangledown) :: (a \multimap c) \rightarrow (b \multimap c) \rightarrow ((a \times b) \multimap c)\\&f \triangledown g = \lambda(a, b) \rightarrow f\ a + g\ b\end{aligned}$$

Now

$$mul = D\ (mul \vartriangle (\lambda(a, b) \rightarrow scale\ b \triangledown scale\ a))$$

Linear arrow vocabulary

$$
\begin{array}{l}
\textbf{class } \mathit{Category}\ (\leadsto) \textbf{ where} \\
\quad \mathit{id}\ \ :: a \leadsto a \\
\quad (\circ) :: (b \leadsto c) \to (a \leadsto b) \to (a \leadsto c)
\end{array}
$$

$$
\begin{array}{l}
\textbf{class } \mathit{Category}\ (\leadsto) \Rightarrow \mathit{Cartesian}\ (\leadsto) \textbf{ where} \\
\quad \mathit{exl}\ :: (a \times b) \leadsto a \\
\quad \mathit{exr} :: (a \times b) \leadsto b \\
\quad (\vartriangle) :: (a \leadsto c) \to (a \leadsto d) \to (a \leadsto (c \times d))
\end{array}
$$

$$
\begin{array}{l}
\textbf{class } \mathit{Category}\ (\leadsto) \Rightarrow \mathit{Cocartesian}\ (\leadsto) \textbf{ where} \\
\quad \mathit{inl}\ :: a \leadsto (a \times b) \\
\quad \mathit{inr} :: b \leadsto (a \times b) \\
\quad (\triangledown) :: (a \leadsto c) \to (b \leadsto c) \to ((a \times b) \leadsto c)
\end{array}
$$

$$
\begin{array}{l}
\textbf{class } \mathit{ScalarCat}\ (\leadsto)\ a \textbf{ where} \\
\quad \mathit{scale} :: a \to (a \leadsto a)
\end{array}
$$

Linear transformations as functions

$$\mathbf{newtype}\ a \rightarrow^{+} b = \mathit{AddFun}\ (a \rightarrow b)$$

$$\begin{array}{l}
\mathbf{instance}\ \mathit{Category}\ (\rightarrow^{+})\ \mathbf{where} \\
\quad \mathit{id}\ \ = \mathit{AddFun}\ \mathit{id} \\
\quad (\circ) = \mathit{inNew}_2\ (\circ)
\end{array}$$

$$\begin{array}{l}
\mathbf{instance}\ \mathit{Cartesian}\ (\rightarrow^{+})\ \mathbf{where} \\
\quad \mathit{exl}\ = \mathit{AddFun}\ \mathit{exl} \\
\quad \mathit{exr} = \mathit{AddFun}\ \mathit{exr} \\
\quad (\vartriangle) = \mathit{inNew}_2\ (\vartriangle)
\end{array}$$

$$\begin{array}{l}
\mathbf{instance}\ \mathit{Cocartesian}\ (\rightarrow^{+})\ \mathbf{where} \\
\quad \mathit{inl}\ = \mathit{AddFun}\ (,0) \\
\quad \mathit{inr} = \mathit{AddFun}\ (0,) \\
\quad (\triangledown) = \mathit{inNew}_2\ (\lambda f\ g\ (x, y) \rightarrow f\ x + g\ y)
\end{array}$$

$$\begin{array}{l}
\mathbf{instance}\ \mathit{Multiplicative}\ s \Rightarrow \mathit{ScalarCat}\ (\rightarrow^{+})\ s\ \mathbf{where} \\
\quad \mathit{scale}\ s = \mathit{AddFun}\ (s\ *)
\end{array}$$

Extracting a data representation

- How to extract a matrix or gradient vector?
- Sample over a domain *basis* (rows of identity matrix).
- For n-dimensional *domain*,
  - Make n passes.
  - Each pass works on n-D sparse ("one-hot") input.
  - Very inefficient.
- For gradient-based optimization,
  - High-dimensional domain.
  - Very low-dimensional (1-D) codomain.

Generalized matrices

$$\textbf{newtype}\ M_s\ a\ b = L\,(V_s\ b\,(V_s\ a\ s))$$

$$applyL :: M_s\ a\ b \to (a \to b)$$

Require *applyL* to preserve structure. Solve for methods.

Core vocabulary

Sufficient to build arbitrary "matrices":

$$
\begin{array}{lll}
\mathit{scale} :: a \to (a \leadsto a) & & \text{-- } 1 \times 1 \\
(\triangledown) & :: (a \leadsto c) \to (b \leadsto c) \to ((a \times b) \leadsto c) & \text{-- horizontal juxt} \\
(\vartriangle) & :: (a \leadsto c) \to (a \leadsto d) \to (a \leadsto (c \times d)) & \text{-- vertical juxt}
\end{array}
$$

Types guarantee rectangularity.

Efficiency of composition

- Arrow composition is associative.
- Some associations are more efficient than others, so
  - Associate optimally.
  - Equivalent to *matrix chain multiplication* — $O(n \log n)$.
  - Choice determined by *types*, i.e., compile-time information.
- All-right: "forward mode AD" (FAD).
- All-left: "reverse mode AD" (RAD).
- RAD is much better for gradient-based optimization.

Left-associating composition (RAD)

- CPS-like category:
  - Represent $a \leadsto b$ by $(b \leadsto r) \to (a \leadsto r)$.
  - Meaning: $f \mapsto (\circ\, f)$.
  - Results in left-composition.
  - Initialize with $id :: r \leadsto r$.
  - Construct $h \circ \mathcal{D}\, f\ a$ directly, without $\mathcal{D}\, f\ a$.

Continuation category

$$\textbf{newtype}\ Cont^{r}_{(\leadsto)}\ a\ b = Cont\ ((b \leadsto r) \to (a \leadsto r))$$

$$\begin{aligned} &cont :: Category\ (\leadsto) \Rightarrow (a \leadsto b) \to Cont^{r}_{(\leadsto)}\ a\ b \\ &cont\ f = Cont\ ({\circ}\ f) \end{aligned}$$

Require *cont* to preserve structure. Solve for methods.

We'll use an isomorphism:

$$\begin{aligned} &join\ \ \ \, :: Cocartesian\ (\leadsto) \Rightarrow (c \leadsto a) \times (d \leadsto a) \to ((c \times d) \leadsto a) \\ &unjoin :: Cocartesian\ (\leadsto) \Rightarrow ((c \times d) \leadsto a) \to (c \leadsto a) \times (d \leadsto a) \end{aligned}$$

$$\begin{aligned} &join\ (f, g) = f \triangledown g \\ &unjoin\ h\ \ \ = (h \circ inl, h \circ inr) \end{aligned}$$

Continuation category (solution)

$$
\begin{aligned}
&\textbf{instance } \mathit{Category}\ (\leadsto) \Rightarrow \mathit{Category}\ \mathit{Cont}^{r}_{(\leadsto)} \textbf{ where}\\
&\quad \mathit{id} = \mathit{Cont}\ \mathit{id}\\
&\quad \mathit{Cont}\ g \circ \mathit{Cont}\ f = \mathit{Cont}\ (f \circ g)
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance } \mathit{Cartesian}\ (\leadsto) \Rightarrow \mathit{Cartesian}\ \mathit{Cont}^{r}_{(\leadsto)} \textbf{ where}\\
&\quad \mathit{exl}\ = \mathit{Cont}\ (\mathit{join} \circ \mathit{inl})\\
&\quad \mathit{exr} = \mathit{Cont}\ (\mathit{join} \circ \mathit{inr})\\
&\quad (\vartriangle) = \mathit{inNew}_2\ (\lambda f\ g \rightarrow (f \triangledown g) \circ \mathit{unjoin})
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance } \mathit{Cocartesian}\ (\leadsto) \Rightarrow \mathit{Cocartesian}\ \mathit{Cont}^{r}_{(\leadsto)} \textbf{ where}\\
&\quad \mathit{inl}\ = \mathit{Cont}\ (\mathit{exl} \circ \mathit{unjoin})\\
&\quad \mathit{inr} = \mathit{Cont}\ (\mathit{exr} \circ \mathit{unjoin})\\
&\quad (\triangledown) = \mathit{inNew}_2\ (\lambda f\ g \rightarrow \mathit{join} \circ (f \vartriangle g))
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance } \mathit{ScalarCat}\ (\leadsto)\ a \Rightarrow \mathit{ScalarCat}\ \mathit{Cont}^{r}_{(\leadsto)}\ a \textbf{ where}\\
&\quad \mathit{scale}\ s = \mathit{Cont}\ (\mathit{scale}\ s)
\end{aligned}
$$

Reverse-mode AD without tears

$$D_{Cont^{r}_{M_s}}$$

Duality

- Vector space dual: $u \multimap s$, with u a vector space over s.
- If u has finite dimension, then $u \multimap s \cong u$.
- For $f :: u \multimap s$, $f = dot\ v$ for some $v :: u$.
- Gradients are derivatives of functions with scalar codomain.
- Represent $a \multimap b$ by $(b \multimap s) \to (a \multimap s)$ by $b \to a$.
- *Ideal* for extracting gradient vector. Just apply to 1 (id).

Duality

$$\textbf{newtype}\ Dual_{(\leadsto)}\ a\ b = Dual\ (b \leadsto a)$$

$$asDual :: Cont^{s}_{(\leadsto)}\ a\ b \rightarrow Dual_{(\leadsto)}\ a\ b$$
$$asDual\ (Cont\ f) = Dual\ (dot^{-1} \circ f \circ dot)$$

where

$$dot \quad :: u \rightarrow (u \multimap s)$$
$$dot^{-1} :: (u \multimap s) \rightarrow u$$

Require *asDual* to preserve structure. Solve for methods.

Duality (solution)

$$\textbf{newtype}\ \mathit{Dual}_{(\leadsto)}\ a\ b = \mathit{Dual}\ (b \leadsto a)$$

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Category}\ (\leadsto) \Rightarrow \mathit{Category}\ \mathit{Dual}_{(\leadsto)}\ \textbf{where} \\
\quad \mathit{id}\ \ = \mathit{Dual}\ \mathit{id} \\
\quad (\circ) = \mathit{inNew}_2\ (\mathit{flip}\ (\circ))
\end{array}
$$

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Cocartesian}\ (\leadsto) \Rightarrow \mathit{Cartesian}\ \mathit{Dual}_{(\leadsto)}\ \textbf{where} \\
\quad \mathit{exl}\ = \mathit{Dual}\ \mathit{inl} \\
\quad \mathit{exr} = \mathit{Dual}\ \mathit{inr} \\
\quad (\vartriangle)\ = \mathit{inNew}_2\ (\triangledown)
\end{array}
$$

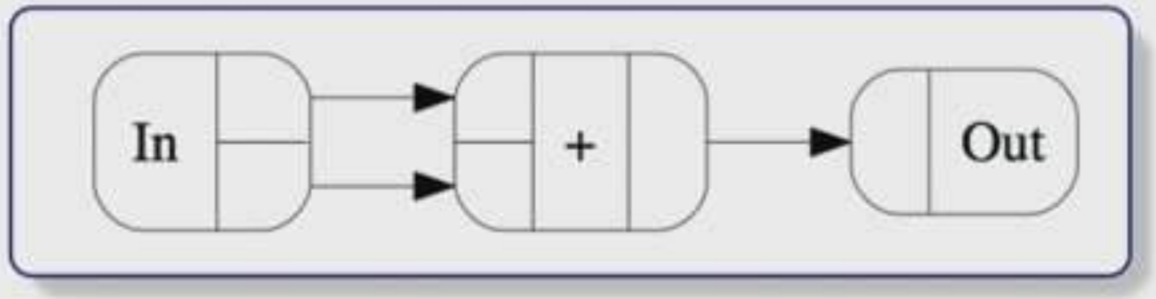
$$
\begin{array}{l}
\textbf{instance}\ \mathit{Cartesian}\ (\leadsto) \Rightarrow \mathit{Cocartesian}\ \mathit{Dual}_{(\leadsto)}\ \textbf{where} \\
\quad \mathit{inl}\ = \mathit{Dual}\ \mathit{exl} \\
\quad \mathit{inr} = \mathit{Dual}\ \mathit{exr} \\
\quad (\triangledown) = \mathit{inNew}_2\ (\vartriangle)
\end{array}
$$

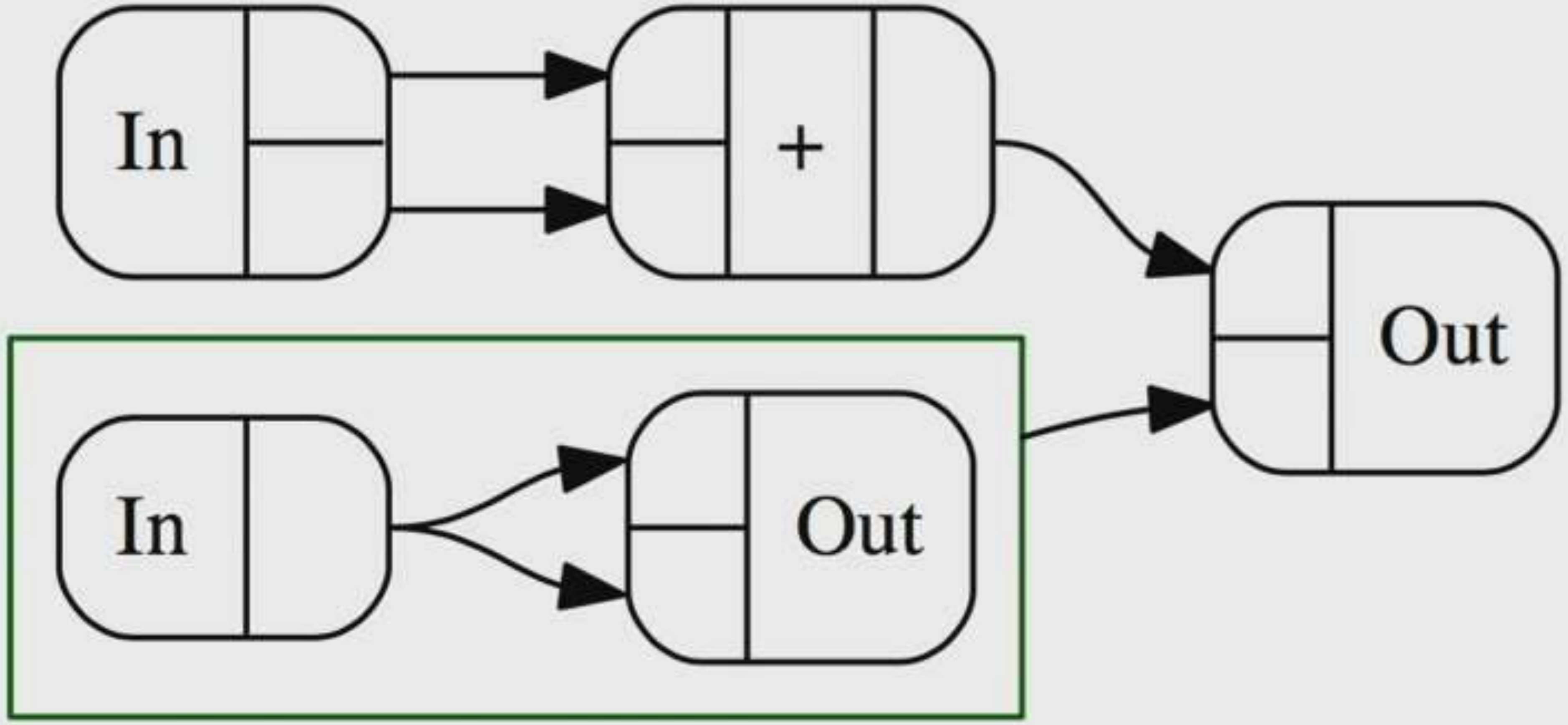
$$
\begin{array}{l}
\textbf{instance}\ \mathit{ScalarCat}\ (\leadsto)\ s \Rightarrow \mathit{ScalarCat}\ \mathit{Dual}_{(\leadsto)}\ s\ \textbf{where} \\
\quad \mathit{scale}\ s = \mathit{Dual}\ (\mathit{scale}\ s)
\end{array}
$$

Backpropagation

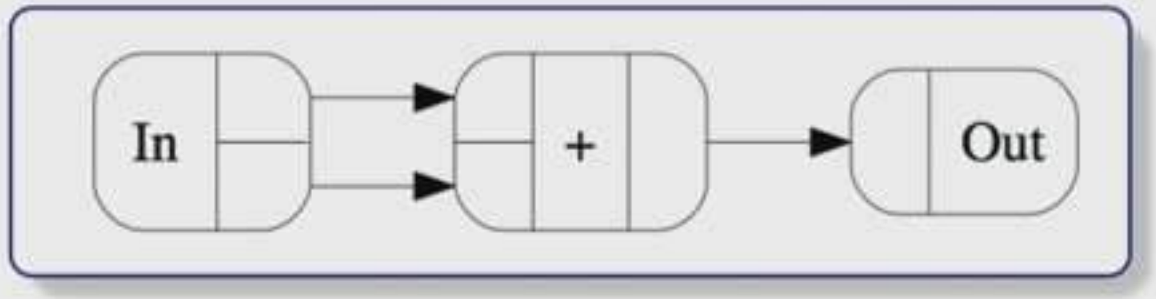
$$D_{Dual_{\multimap^+}}$$

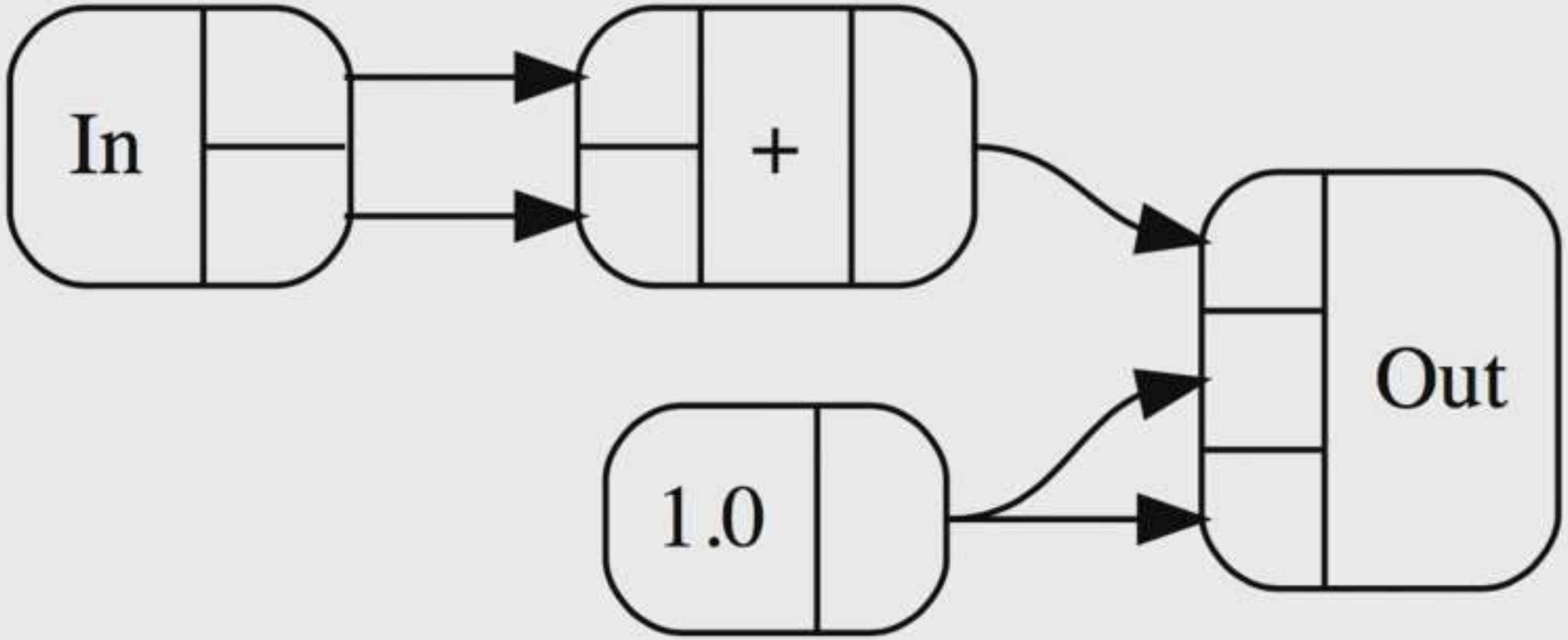
RAD example (dual function)





RAD example (dual vector)





Solution: simple automatic differentiation

$$\mathbf{newtype}\ D\ a\ b = D\ (a \rightarrow b \times (a \multimap b))$$

$$linearD\ f = D\ (\lambda a \rightarrow (f\ a, a))$$

$$\begin{aligned}
&\mathbf{instance}\ Category\ D\ \mathbf{where}\\
&\quad id = linearD\ id\\
&\quad D\ g \circ D\ f = D\ (\lambda a \rightarrow \mathbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \mathbf{in}\ (c, g' \circ f'))
\end{aligned}$$

$$\begin{aligned}
&\mathbf{instance}\ Cartesian\ D\ \mathbf{where}\\
&\quad exl = linearD\ exl\\
&\quad exr = linearD\ exr\\
&\quad D\ f \vartriangle D\ g = D\ (\lambda a \rightarrow \mathbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \mathbf{in}\ ((b, c), f' \vartriangle g'))
\end{aligned}$$

$$\begin{aligned}
&\mathbf{instance}\ NumCat\ D\ \mathbf{where}\\
&\quad negate = linearD\ negate\\
&\quad add = linearD\ add\\
&\quad mul = D\ (mul \vartriangle (\lambda(a, b) \rightarrow \lambda(da, db) \rightarrow b * da + a * db))
\end{aligned}$$

Generalizing AD

$\textbf{newtype}\ D\ a\ b = D\ (a \rightarrow b \times (a \multimap b))$

$linearD\ f = D\ (\lambda a \rightarrow (f\ a, f))$

$\textbf{instance}\ Category\ D\ \textbf{where}$
$\quad id = linearD\ id$
$\quad D\ g \circ D\ f = D\ (\lambda a \rightarrow \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))$

$\textbf{instance}\ Cartesian\ D\ \textbf{where}$
$\quad exl = linearD\ exl$
$\quad exr = linearD\ exr$
$\quad D\ f \vartriangle D\ g = D\ (\lambda a \rightarrow \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ a\}\ \textbf{in}\ ((b, c), f' \vartriangle g'))$

Each D operation just uses corresponding $(\multimap)$ operation.

Generalize from $(\multimap)$ to other cartesian categories.