A short introdution

- In chapter 4 we've derived an AD algorithm that was generalized in figure 6 of the document
- With fully right-associated compositions this algorithm becomes a foward-mode AD and with fully left-associated becomes a reverse-mode AD
- We want to obtain generalized FAD and RAD algorithms
- How do we describe this in Categorical notation?

A short introdution

- In chapter 4 we've derived an AD algorithm that was generalized in figure 6 of the document
- With fully right-associated compositions this algorithm becomes a foward-mode AD and with fully left-associated becomes a reverse-mode AD
- We want to obtain generalized FAD and RAD algorithms
- How do we describe this in Categorical notation?

A short introdution

- In chapter 4 we've derived an AD algorithm that was generalized in figure 6 of the document
- With fully right-associated compositions this algorithm becomes a foward-mode AD and with fully left-associated becomes a reverse-mode AD
- We want to obtain generalized FAD and RAD algorithms
- How do we describe this in Categorical notation?

A short introdution

- In chapter 4 we've derived an AD algorithm that was generalized in figure 6 of the document
- With fully right-associated compositions this algorithm becomes a foward-mode AD and with fully left-associated becomes a reverse-mode AD
- We want to obtain generalized FAD and RAD algorithms
- How do we describe this in Categorical notation?

Converting morfisms

Given a category k we can represent its morfisms the following way:

Left-Compose functions

f :: a'k'b $\Rightarrow$ ($\circ$ f) :: (b'k'r) $\rightarrow$ (a'k'r) where r is any object of k.

If h is the morfism we'll compose with f then h is the continuation of f.

Instance deduction

Defining new type

$$\textbf{newtype}\ \mathit{Cont}\ (k, r)\ a\ b = \mathit{Cont}\ ((b\,'k\,'r) \to (a\,'k\,'r))$$

Functor derived from type

$$\mathit{cont} :: \mathit{Category}\ k \Rightarrow (a\ '\mathtt{k}'\ b) \to \mathit{Cont}\ (k, r)\ a\ b$$
$$\mathit{cont}\ f = \mathit{Cont}\ (\circ f)$$

Instance deduction

instance *Category k* $\Rightarrow$ *Category Cont* (k, r) **where**
$id = Cont\ id$
$Cont\ g \circ Cont\ f = Cont\ (f \circ g)$

instance *Monoidal k* $\Rightarrow$ *Monoidal Cont* (k, r) **where**
$Conf\ f\ x\ Cont\ g = Cont\ (\bigtriangleup \circ (f\ x\ g) \circ unjoin)$

instance *Cartesian k* $\Rightarrow$ *Cartesian Cont* (k, r) **where**
$exl = Cont\ (\bigtriangleup \circ inl); exr = Cont\ (\bigtriangleup \circ inr)$
$dup = Cont\ (jam \circ unjoin)$

instance *Cocartesian k* $\Rightarrow$ *Cocartesian Cont* (k, r) **where**
$inl = Cont\ (exl \circ unjoin); inr = Cont\ (exr \circ unjoin)$
$jam = Cont\ (\bigtriangleup \circ dup)$

instance *Scalable k a* $\Rightarrow$ *Scalable Cont* (k, r) *a* **where**
$scale\ s = Cont\ (scale\ s)$

A short introdution

Due to it's widespread use in ML we'll talk about a specific case of RAD: computing gradients(derivatives of functions with scalar codomains)
A vector space A over a scalar field has A $\multimap$ s as its dual.
Each linear map in A $\multimap$ s can be represented in the form of dot u for some u :: A where

Definition and instanciation

$$\textbf{class } \mathit{HasDot}\ (S)\ u \textbf{ where } \mathit{dot} :: u \to (u - o\ s)$$

$$\textbf{instance } \mathit{HasDot}\ (\mathit{IR})\ \mathit{IR} \textbf{ where } \mathit{dot} = \mathit{scale}$$

$$\textbf{instance } (\mathit{HasDot}\ (S)\ a, \mathit{HasDot}\ (S)\ b) \Rightarrow \mathit{HasDot}\ (S)\ (a\ x\ b) \textbf{ where } \mathit{dot}\ (u, v) = \mathit{dot}\ u\ \bigtriangledown\ \mathit{dot}\ v$$

Instance deduction

The internal representation of $Cont^{s}_{\multimap}$ a b is (b $\multimap$ s) -> (a $\multimap$ s) which is isomorfic to (a -> b).

Type definition for duality

newtype $Dual\ (K)\ a\ b = Dual\ (b' k' a)$

Instance deduction

All we need to do to create dual representations of linear maps is to convert from $Cont_k^S$ to $Dual_k$ using a functor:

Functor definition

$$
\begin{aligned}
&asDual :: (HasDot\ (S)\ a, HasDot\ (S)\ b) \Rightarrow \\
&\quad ((b - o\ s) \rightarrow (a - o\ s)) \rightarrow (b - o\ a) \\
&asDual\ (Cont\ f) = Dual\ (onDot\ f)
\end{aligned}
$$

where

$$
\begin{aligned}
&onDot :: (HasDot\ (S)\ a, HasDot\ (S)\ b) \Rightarrow \\
&\quad ((b - o\ s) \rightarrow (a - o\ s)) \rightarrow (b - o\ a) \\
&onDot\ f = dot\ \hat{}\ -1 \circ f \circ dot
\end{aligned}
$$

Instance deduction

instance *Category k* $\Rightarrow$ *Category Dual* (k) **where**
$id = Dual\ id$
$Dual\ g \circ Dual\ f = Dual\ (f \circ g)$

instance *Monoidal k* $\Rightarrow$ *Monoidal Dual* (k) **where**
$Dual\ f\ x\ Dual\ g = Dual\ (f\ x\ g)$

instance *Cartesian k* $\Rightarrow$ *Cartesian Dual* (k) **where**
$exl = Dual\ inl; exr = Dual\ inr; dup = Dual\ jam$

instance *Cocartesian k* $\Rightarrow$ *CocartesianDual* (k) **where**
$inl = Dual\ exl; inr = Dual\ exr; jam = Dual\ dup$

instance *Scalable k* $\Rightarrow$ *Scalable Dual* (k) **where**
$scale\ s = Dual\ (scale\ s)$

Final notes

- $\triangle$ *and* ∇ mutually dualize
  $(Dual\ f\ \triangle\ Dual\ g =$
  $Dual\ (f\ \nabla\ g)\ and\ Dual\ f\ \nabla\ Dual\ g = Dual\ (f\ \triangle\ g))$
- Using the definition from chapter 8 we can determine that the duality of a matrix corresponds to it's transposition

Fowards-mode Automatic Differentiation(FAD)

We can use the same deductions we've done in Cont and Dual to derive a category with full right-side association, thus creating a generized FAD algorithm.
This algorithm is far more apropriated for low dimention domains.

Type definition and functor from type

$$\textbf{newtype}\ \mathit{Begin}\ (k, r)\ a\ b = \mathit{Begin}\ ((r' k' a) \to (r' k' b))$$

$$\mathit{begin} :: \mathit{Category}\ k \Rightarrow (a\ \texttt{`k`}\ b) \to \mathit{Begin}\ (k, r)\ a\ b$$

$$\mathit{begin}\ f = \mathit{Begin}\ (f \circ)$$

We can derive categorical instances from the functor above and we can choose r to be the scalar field s, noting that s $\multimap$ a is isomorfic to a.