...Machine Learning...

Artur Ezequiel Nelson

Universidade do Minho

26 de Abril

Indice

titulo

Uma curta introdução

- Queremos calcular $\mathcal{D}^+$.

Uma curta introdução

- Queremos calcular $\mathcal{D}^+$.
- Problema: $\mathcal{D}$ não é computável.

Uma curta introdução

- Queremos calcular $\mathcal{D}^{+}$.
- Problema: $\mathcal{D}$ não é computável.
- Solução: observar corolários apresentados e implementar recorrendo a categorias.

Uma curta introdução

Corolário 1.1

NOTA: adicionar definição do corolário 1.1 aqui

Corolário 2.1

NOTA: adicionar definição do corolário 2.1 aqui

Corolário 3.1

NOTA: adicionar definição do corolário 3.1 aqui

Definição de Derivada

Definição

Seja $f : \mathbb{R} \to \mathbb{R}$ uma função. A derivada de f no ponto $x \in \mathbb{R}$ é definido da seguinte forma:

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x + \varepsilon) - f(x)}{\varepsilon}$$

Definição de Derivada

Definição

Seja $f : \mathbb{R} \to \mathbb{R}$ uma função. A derivada de f no ponto $x \in \mathbb{R}$ é definido da seguinte forma:

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x + \varepsilon) - f(x)}{\varepsilon}$$

A definição acima também funcionará para funções de tipos $\mathbb{C} \to \mathbb{C}$ e $\mathbb{R} \to \mathbb{R}^n$.

Definição de Derivada

Para funções F de tipos $\mathbb{R}^m \to \mathbb{R}$ e $\mathbb{R}^m \to \mathbb{R}^n$ (com $n > 1$), precisamos de uma definição diferente.

Definição de Derivada

Para funções F de tipos $\mathbb{R}^m \to \mathbb{R}$ e $\mathbb{R}^m \to \mathbb{R}^n$ (com $n > 1$), precisamos de uma definição diferente.

- Em funções $\mathbb{R}^m \to \mathbb{R}$ é necessário introduzir a noção de derivadas parciais, $\frac{\partial F}{\partial x_j}$, com $j \in \{1, ..., m\}$.

Definição de Derivada

Para funções F de tipos $\mathbb{R}^m \to \mathbb{R}$ e $\mathbb{R}^m \to \mathbb{R}^n$ (com $n > 1$), precisamos de uma definição diferente.

- Em funções $\mathbb{R}^m \to \mathbb{R}$ é necessário introduzir a noção de derivadas parciais, $\frac{\partial F}{\partial x_j}$, com $j \in \{1, ..., m\}$.
- Em funções $\mathbb{R}^m \to \mathbb{R}^n$ (com $n > 1$), para além de derivadas parciais, é necessário utilizar matrizes Jacobianas $\mathbf{J}_{i,j} = \frac{\partial F_i}{\partial x_j}$, onde $i \in \{1, ..., n\}$ e F_i é uma função $\mathbb{R}^m \to \mathbb{R}$.

Generalização e Regra da Cadeia

Sejam $\mathbf{A}$ e $\mathbf{B}$ duas matrizes Jacobianas.

A regra da cadeia em $\mathbb{R}^m \to \mathbb{R}^n$ é:

$$(\mathbf{A} \cdot \mathbf{B})_{i,j} = \sum_{k=1}^{m} \mathbf{A}_{i,k} \cdot \mathbf{B}_{k,j}$$

Generalização e Regra da Cadeia

Assumindo que a noção de derivada que queremos corresponde a uma transformação linear, onde é aceita a regra da cadeia vista anteriormente, vamos definir uma nova generalização:

Generalização e Regra da Cadeia

Assumindo que a noção de derivada que queremos corresponde a uma transformação linear, onde é aceita a regra da cadeia vista anteriormente, vamos definir uma nova generalização:

$$\lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon} - f'(x) = 0 \Leftrightarrow \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - (f(x) + \varepsilon \cdot f'(x))}{\varepsilon} = 0$$

$$\Leftrightarrow \lim_{\varepsilon \to 0} \frac{\| f(x+\varepsilon) - (f(x) + \varepsilon \cdot f'(x)) \|}{\| \varepsilon \|} = 0$$

Derivada como transformação linear

Definição

Seja $f :: a \to b$ uma função, onde a e b são espaços vetoriais com base no mesmo corpo. A primeira definição de derivada é da seguinte forma:

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

Se diferenciamos duas vezes temos:

$$\mathcal{D}^2 = \mathcal{D} \circ \mathcal{D} :: (a \to b) \to (a \to (a \multimap a \multimap b))$$

Regras de diferenciação - Composição

Teorema

Sejam $f :: a \to b$ e $g :: b \to c$ *duas funções. Então a derivada da composta de* f e g *é*

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

Regras de diferenciação - Composição

Como infelizmente o teorema anterior não é uma regra eficiente para a composição, vamos introduzir uma segunda definição de derivada:

$$\mathcal{D}_0^+ :: (a \to b) \to ((a \to b) \times (a \to (a \multimap b)))$$
$$\mathcal{D}_0^+\, f = (f, \mathcal{D} f)$$

Regras de diferenciação - Composição

Como infelizmente o teorema anterior não é uma regra eficiente para a composição, vamos introduzir uma segunda definição de derivada:

$$\mathcal{D}_0^+ :: (a \to b) \to ((a \to b) \times (a \to (a \multimap b)))$$
$$\mathcal{D}_0^+\ f = (f, \mathcal{D} f)$$

Com isto a regra da cadeia ficará da seguinte forma:

$\mathcal{D}_0^+\,(g \circ f) =$

$= (g \circ f, \mathcal{D}\,(g \circ f)\,)$ (definição de $\mathcal{D}_0^+$)

$= (\lambda a \to g(f\ a), \lambda a \to \mathcal{D}\ g\ (f\ a) \circ \mathcal{D}\ f\ a)$ (Teorema e definição de $g \circ f$)

Regras de diferenciação - Composição

Tendo em mente otimizações, vamos introduzir a última definição de derivada:

$$\mathcal{D}^{+} :: (a \to b) \to (a \to (b \times (a \multimap b)$$
$$\mathcal{D}^{+}\ f\ a = (f\,a, \mathcal{D}\,f\,a)$$

Regras de diferenciação - Composição

Tendo em mente otimizações, vamos introduzir a última definição de derivada:

$$\mathcal{D}^{+} :: (a \to b) \to (a \to (b \times (a \multimap b))$$
$$\mathcal{D}^{+}\ f\ a = (f\,a, \mathcal{D}\,f\,a)$$

Como $\times$ tem mais prioridade do que $\to$ e $\multimap$, podemos reescrever $\mathcal{D}^{+}$ da seguinte forma:

$$\mathcal{D}^{+} :: (a \to b) \to (a \to b \times (a \multimap b))$$
$$\mathcal{D}^{+}\ f\ a = (f\,a, \mathcal{D}\,f\,a)$$

Regras de diferenciação - Composição

Corolário

$\mathcal{D}^+$ é eficientemente composicional em relação a $(\circ)$, ou seja, em linguagem Haskell:

$$\mathcal{D}^+\ (g \circ f)\ a = \textbf{\textit{let}}\,\{(b,\ f') = \mathcal{D}^+\ f\ a;\ (c,\ g') = \mathcal{D}^+\ g\ b\}\ \textbf{\textit{in}}\ (c,\ g' \circ f')$$

Regras de diferenciação - Split

Outra forma importante de combinar funçõe é a operação cross, que combina duas funções de forma paralela:

$$
\begin{aligned}
&(\times) :: (a \to c) \to (b \to d) \to (a \times b \to c \times d) \\
&f \times g = \lambda(a, b) \to (f\, a, g\, b)
\end{aligned}
$$

Regras de diferenciação - Split

Outra forma importante de combinar funçõe é a operação cross, que combina duas funções de forma paralela:

$$(\times) :: (a \rightarrow c) \rightarrow (b \rightarrow d) \rightarrow (a \times b \rightarrow c \times d)$$
$$f \times g = \lambda(a, b) \rightarrow (f\, a, g\, b)$$

Teorema

Seja $f :: a \rightarrow c$ e $g :: b \rightarrow d$ duas funções. Então a regra do cross é da seguinte forma:

$$\mathcal{D}\,(f \times g)\,(a, b) = \mathcal{D}\, f\ a \times \mathcal{D}\, g\ b$$

Regras de diferenciação - Split

Corolário

A função $\mathcal{D}^+$ é composicional em relação a $(\times)$

$$\mathcal{D}^+ (f \times g)\ (a, b) = \textbf{\textit{let}}\, \{(c, f') = \mathcal{D}^+\, f\ a;\ (d, g') = \mathcal{D}^+\, g\ b\}\ \textbf{\textit{in}}\, ((c, d),\ f' \times g')$$

Derivada e funções lineares

Definição

Uma função f diz-se linear quando preserva a adição e a multiplicação escalar.

$$f(a + a') = f\,a + f\,a'$$
$$f(s \cdot a) = s \cdot f\,a$$

Derivada e funções lineares

Definição

Uma função f diz-se linear quando preserva a adição e a multiplicação escalar.

$$f(a + a') = f\,a + f\,a'$$
$$f(s \cdot a) = s \cdot f\,a$$

Teorema

Para todas as funções lineares f, $\mathcal{D}\,f\,a = f$.

Derivada e funções lineares

Definição

Uma função f diz-se linear quando preserva a adição e a multiplicação escalar.

$$f\,(a + a') = f\,a + f\,a'$$
$$f\,(s \cdot a) = s \cdot f\,a$$

Teorema

Para todas as funções lineares f, $\mathcal{D}\,f\;a = f$.

Corolário

Para todas as funções lineares f, $\mathcal{D}^{+}\,f = \lambda a \rightarrow (f a,\, f)$.

Categorias clássicas

Uma categoria é um conjunto de objetos(conjuntos e tipos) e de morfismos(operações entre objetos). Uma categoria tem definidas 2 operações básicas, identidade e composição de morfismos, e 2 leis:

- (C.1) —– $id \circ f = id \circ f = f$
- (C.2) —– $f \circ (g \circ h) = (f \circ g) \circ h$

Categorias clássicas

Uma categoria é um conjunto de objetos(conjuntos e tipos) e de morfismos(operações entre objetos). Uma categoria tem definidas 2 operações básicas, identidade e composição de morfismos, e 2 leis:

- (C.1) —— $id \circ f = id \circ f = f$
- (C.2) —— $f \circ (g \circ h) = (f \circ g) \circ h$

Para os efeitos deste papel, objetos são tipos de dados e morfismos são funções.

class *Category k* where
 id :: (a'k'a)
 (∘) :: (b'k'c) → (a'k'b) → (a'k'c)

instance *Category (→)* where
 id = λa → a
 $g \circ f$ = λa → g (f a)

Functores clássicos

Um functor F entre categorias $\mathcal{U}$ e $\mathcal{V}$ é tal que:

- para qualquer objeto t $\in \mathcal{U}$ temos que F t $\in \mathcal{V}$
- para qualquer morfismo m :: a $\rightarrow$ b $\in \mathcal{U}$ temos que F m :: F a $\rightarrow$ F b $\in \mathcal{V}$
- F id $(\in \mathcal{U})$ = id $(\in \mathcal{V})$
- F $(f \circ g)$ = F f $\circ$ F g

Nota

Devido à definição de categoria deste papel(objetos são tipos de dados) os functores mapeiam tipos neles próprios.

Objetivo

Começamos por definir um novo tipo de dados:

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

Depois adaptamos $\mathcal{D}^{+}$ para usar este tipo de dados:

Definição adaptada

$\hat{\mathcal{D}}$ :: (a $\to$ b) $\to$ $\mathcal{D}$ a b
$\hat{\mathcal{D}}$ f $= \mathcal{D}$ ($\mathcal{D}^{+}$ f)

O nosso objetivo é a dedução de uma instância de categoria para $\mathcal{D}$ onde $\hat{\mathcal{D}}$ seja functor.

Dedução da instância

Recordando os corolários 3.1 e 1.1 deduzimos que

- (DP.1) —– $\mathcal{D}^{+}id = \lambda \mathsf{a} \to (\mathsf{id\ a, id})$
- (DP.2) —– $\mathcal{D}^{+}(g \circ f) = \lambda a \to let\{(b,\ f') = \mathcal{D}^{+}\ \mathsf{f\ a};\ (c, g') = \mathcal{D}^{+}\ \mathsf{g\ b}\ \}\ \mathsf{in}\ (c,\ g' \circ f')$

$\hat{\mathcal{D}}$ ser functor é equivalente a dizer que, para todas as funções f e g de tipos apropriados:

- $\mathsf{id} = \hat{\mathcal{D}}\ \mathsf{id} = \mathcal{D}\ (\mathcal{D}^{+}id)$
- $\hat{\mathcal{D}}\ \mathsf{g} \circ \hat{\mathcal{D}}\ \mathsf{f} = \hat{\mathcal{D}}\ (\mathsf{g} \circ \mathsf{f}) = \mathcal{D}\ (\mathcal{D}^{+}(g \circ f))$

Dedução da instância

Com base em (DP.1) e (DP.2) podemos reescrever como sendo:

- $\mathrm{id} = \mathcal{D}(\lambda \mathrm{a} \to (\mathrm{id\ a}, \mathrm{id}))$
- $\hat{\mathcal{D}}\ \mathrm{g} \circ \hat{\mathcal{D}}\ \mathrm{f} = \mathcal{D}\ (\ \lambda a \to let\{(b, f') = \mathcal{D}^{+}\ \mathrm{f\ a};\ (c, g') = \mathcal{D}^{+}\ \mathrm{g\ b}\ \}\ \mathrm{in}\ (c, g' \circ f')\ )$

Resolver a primeira equação é trivial(definir id da instância como sendo $\mathcal{D}(\lambda \mathrm{a} \to (\mathrm{id\ a}, \mathrm{id}))$).

A segunda equação será resolvida resolvendo uma condição mais geral: $\mathcal{D}g \circ \mathcal{D}f = \mathcal{D}\ (\lambda a \to let\{(b, f') = \mathrm{f\ a};\ (c, g') = \mathrm{g\ b}\ \}\ \mathrm{in}\ (c, g' \circ f'))$, cuja solução é igualmente trivial.

Dedução da instância

Definição de $\hat{\mathcal{D}}$ para funções lineares

linearD :: (a $\rightarrow$ b) $\rightarrow$ $\mathcal{D}$ a b
linearD f = $\mathcal{D}$ (λa $\rightarrow$ (f a,f))

Instância da categoria que deduzimos

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

Prova da instância

Antes de continuarmos devemos verificar se esta instância obedece às leis (C.1) e (C.2).

Se considerarmos apenas morfismos $\hat{f}$ :: $\mathcal{D}$ a b tal que $\hat{f} = \mathcal{D}^+$ f para f :: a $\rightarrow$ b (o que podemos garantir se transformarmos $\mathcal{D}$ a b em tipo abstrato) podemos garantir que $\mathcal{D}^+$ é functor.

Prova de (C.1)

id $\circ \hat{\mathcal{D}}$

$= \hat{\mathcal{D}}\ id \circ \hat{\mathcal{D}}$ f -lei functor de id (especificação de $\hat{\mathcal{D}}$)

$= \hat{\mathcal{D}}$ (id $\circ$ f) - lei functor para ($\circ$)

$= \hat{\mathcal{D}}$ f - lei de categoria

Prova da instância

Prova de (C.2)

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

$=$ $\hat{\mathcal{D}}$ h $\circ$ $\hat{\mathcal{D}}$ (g $\circ$ f) - lei functor para ($\circ$)

$=$ $\hat{\mathcal{D}}$ (h $\circ$ (g $\circ$ f)) - lei functor para ($\circ$)

$=$ $\hat{\mathcal{D}}$ ((h $\circ$ g) $\circ$ f) - lei de categoria

$=$ $\hat{\mathcal{D}}$ (h $\circ$ g) $\circ$ $\hat{\mathcal{D}}$ f - lei functor para ($\circ$)

$=$ ($\hat{\mathcal{D}}$ h $\circ$ $\hat{\mathcal{D}}$ g) $\circ$ $\hat{\mathcal{D}}$ f - lei functor para ($\circ$)

Nota

Estas provas não requerem nada de $\mathcal{D}$ e $\hat{\mathcal{D}}$ para além das leis do functor, logo nas próximas instâncias deduzidas de um functor não precisamos de voltar a realizar estas provas.

Categorias e functores monoidais

A versão generalizada da composição paralela será definida através de uma categoria monoidal:

class *Category k* $\Rightarrow$ *Monoidal k* where	instance *Monoidal* $(\rightarrow)$ where
$(\times)$::(a'k'c)$\rightarrow$(b'k'd)$\rightarrow$((a$\times$b)'k'(c$\times$d))	$f \times g = \lambda$(a,b)$\rightarrow$(f a,g b)

Definição de functor monoidal

Um functor F monoidal entre categorias $\mathcal{U}$ e $\mathcal{V}$ é tal que:

- F é functor clássico
- F (f $\times$ g) $=$ F f $\times$ F g

Dedução da instância

A partir do corolário 2.1 deduzimos que:

$\mathcal{D}^{+}$ (f $\times$ g) $= \lambda$(a,b) $\rightarrow$ let{(c,f')= $\mathcal{D}^{+}$ f a; (d,g') $= \mathcal{D}^{+}$ g b }
in ((c,d),f' $\times$ g')

Se definirmos o functor F a partir de $\hat{\mathcal{D}}$ chegamos à seguinte condição:

$\mathcal{D}$ ($\mathcal{D}^{+}$ f) $\times$ $\mathcal{D}$ ($\mathcal{D}^{+}$ g) $=$ $\mathcal{D}$ ($\mathcal{D}^{+}$ (f $\times$ g))

Substituindo e fortalecendo-a obtemos:

$\mathcal{D}$ f $\times$ $\mathcal{D}$ g $=$ $\mathcal{D}$ (λ(a,b) $\rightarrow$ let{(c,f') $=$ f a; (d,g') $=$ g b } in ((c,d),f' $\times$ g'))

e esta condição é suficiente para obtermos a nossa instância.

Dedução da instância

Instância da categoria que deduzimos

instance *Monoidal* $\mathcal{D}$ where

$\mathcal{D}$ f $\times$ $\mathcal{D}$ g $=$ $\mathcal{D}(\lambda$(a,b) $\rightarrow$ let{(c,f') $=$ f a; (d,g') $=$ g b } in ((c,d),f'$\times$g'))

Categorias e funtores cartesianas

class *Monoidal k* $\Rightarrow$ *Cartesean k* where
  exl :: (a$\times$b)'k'a
  exr :: (a$\times$b)'k'b
  dup :: a'k'(a$\times$a)

instance *Cartesean* $(\rightarrow)$ where
  exl $= \lambda$(a,b) $\rightarrow$ a
  exr $= \lambda$(a,b) $\rightarrow$ b
  dup $= \lambda$a $\rightarrow$ (a,a)

Um functor F cartesiano entre categorias $\mathcal{U}$ e $\mathcal{V}$ é tal que:

- F é functor monoidal
- F exl = exl
- F exp = exp
- F dup = dup

Dedução da instância

Pelo corolário 3.1 e pelo facto que exl,exr e dup são linerares deduzimos que:

$\mathcal{D}^+$ exl λp $\rightarrow$ (exp p, exl)

$\mathcal{D}^+$ exr λp $\rightarrow$ (exr p, exr)

$\mathcal{D}^+$ dup λa $\rightarrow$ (dup a, dup)

Após esta dedução podemos continuar a determinar a instância:

exl $= \mathcal{D}$ ($\mathcal{D}^+$ exl)

exr $= \mathcal{D}$ ($\mathcal{D}^+$ exr)

dup $= \mathcal{D}$ ($\mathcal{D}^+$ dup)

Dedução da instância

Substituindo e usando a definição de linearD obtemos:

exl = linearD exl

exr = linearD exr

dup = linearD dup

E podemos converter a dedução acima diretamente em instância:

Instância da categoria que deduzimos

instance *Cartesian* $\mathcal{D}$ where
  exl = linearD exl
  exr = linearD exr
  dup = linearD dup

Categorias cocartesianas

São o dual das categorias cartesianas.

Nota

Neste papel os coprodutos correspondem aos produtos das categorias, i.e., categorias de biprodutos.

class *Category k* ⇒ *Cocartesian k* where:
inl :: a'k'(a×b)
inlr:: b'k'(a×b)
jam :: (a×a)'k'a

Functores cocartesianos

Definição de functor cocartesiano

Um functor F cartesiano entre categorias $\mathcal{U}$ e $\mathcal{V}$ é tal que:

- F é functor
- $F\ \mathsf{inl} = \mathsf{inl}$
- $F\ \mathsf{inr} = \mathsf{inr}$
- $F\ \mathsf{jam} = \mathsf{jam}$

Fork e Join

- (Δ) :: Cartesian k $\Rightarrow$ (a 'k' c) $\rightarrow$ (a 'k' d) $\rightarrow$ (a 'k' (c $\times$ d))
- (∇) :: Cartesian k $\Rightarrow$ (c 'k' a) $\rightarrow$ (d 'k' a) $\rightarrow$ ((c $\times$ d) 'k' a)

instancia de $\rightarrow^{+}$

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

instance Category ($\rightarrow^{+}$) **where**
    type Obj ($\rightarrow^{+}$) = Additive
    id = AddFun id
    AddFun g $\circ$ AddFun f = AddFun (g $\circ$ f)

instance Monoidal ($\rightarrow^{+}$) **where**
    AddFun f $\times$ AddFun g = AddFun (f $\times$ g)

instance Cartesian ($\rightarrow^{+}$) **where**
    exl = AddFun exl
    exr = AddFun exr
    dup = AddFun dup

instancia de $\rightarrow^{+}$

instance Cocartesian $(\rightarrow^{+})$ **where**
inl = AddFun inlF
inr = AddFun inrF
jam = AddFun jamF

inlF :: Additive b $\Rightarrow$ a $\rightarrow$ a $\times$ b
inrF :: Additive a $\Rightarrow$ b $\rightarrow$ a $\times$ b
jamF :: Additive a $\Rightarrow$ a $\times$ a $\rightarrow$ a

inlF = λa $\rightarrow$ (a, 0)
inrF = λb $\rightarrow$ (0, b)
jamF = λ(a, b) $\rightarrow$ a + b

definição de NumCat

class NumCat k a **where**
    negateC :: a 'k' a
    addC :: (a $\times$ a) 'k' a
    mulC :: (a $\times$ a) 'k' a
    ...

instance Num a $\Rightarrow$ NumCat ($\rightarrow$) a **where**
    negateC $=$ negate
    addC $=$ uncurry $(+)$
    mulC $=$ uncurry $(\cdot)$
    ...

$$\mathrm{D}\ (\mathrm{negate}\ \mathrm{u}) = \mathrm{negate}\ (\mathrm{D}\ \mathrm{u})$$
$$\mathrm{D}\ (\mathrm{u} + \mathrm{v}) = \mathrm{D}\ \mathrm{u} + \mathrm{D}\ \mathrm{v}$$
$$\mathrm{D}\ (\mathrm{u} \cdot \mathrm{v}) = \mathrm{u} \cdot \mathrm{D}\ \mathrm{v} + \mathrm{v} \cdot \mathrm{D}\ \mathrm{u}$$

- Impreciso na natureza de u e v.
- Algo mais preciso seria defenir a diferenciação das operações em si.

class Scalable k a **where**
    scale :: a $\rightarrow$ (a 'k' a)

instance Num a $\Rightarrow$ Scalable ($\rightarrow^{+}$) a **where**
    scale a = AddFun (λda $\rightarrow$ a $\cdot$ da)

instance NumCat D **where**
    negateC = linearD negateC
    addC = linearD addC
    mulC = D (λ(a, b) $\rightarrow$ (a $\cdot$ b, scale b ∇ scale a))

Generalizing Automatic Differentiation

newtype D_k a b = D (a → b × (a 'k' b))

linearD :: (a → b) → (a 'k' b) → D_k a b
linearD f f'= D (λa → (f a, f'))

instance Category k → Category D_k **where**
    type Obj D_k = Additive ∧ Obj k ...

instance Monoidal k ⇒ Monoidal D_k **where** ...

instance Cartesian k ⇒ Cartesian D_k **where** ...

instance Cocartesian k ⇒ Cocartesian D_k **where**
    inl = linearD inlF inl
    inr = linearD inrF inr
    jam = linearD jamF jam

instance Scalable k s ⇒ NumCat D_k s **where**
    negateC = linearD negateC negateC
    addC = linearD addC addC
    mulC = D (λ(a, b) → (a · b, scale b ∇ scale a))

Exemplos

Generalizar