

...Machine Learning...

Artur Ezequiel Nelson

Universidade do Minho

26 de Abril

Indice

- 1 Nelson
- 2 Categories
- 3 Fork and Join
- 4 Operacoes Numericas
- 5 Generalizing Automatic Differentiation
- 6 Exemplos
- 7 Generalizar

Nelson

Categories

Fork and Join

Operacoes Numericas

Generalizing Automatic Differentiation

Exemplos

Generalizar

titulo

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

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

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

Categorias clássicas

Uma categoria é um conjunto de objetos(conjuntos e tipos) e de morfismos(operações entre objetos), tendo 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)
  (o) :: (b'k'c) -> (a'k'b) -> (a'k'c)
```

```
instance Category (→) where
  id = λa → a
  a ∘ f = λa → a (f a)
```


Categorias clássicas

Uma categoria é um conjunto de objetos(conjuntos e tipos) e de morfismos(operações entre objetos), tendo 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)
  (o) :: (b'k'c) -> (a'k'b) -> (a'k'c)
```

```
instance Category (→) where
  id = λa → a
  o ∘ f = λa → o (f a)
```

Funtores 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 $F m :: F a \rightarrow F b \in \mathcal{V}$
- $F \text{ id } (\in \mathcal{U}) = \text{id } (\in \mathcal{V})$
- $F (f \circ g) = F f \circ F g$

Nota

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

Objetivo

Começamos por definir um novo tipo de dados:

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

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

Definição adaptada

$\hat{\mathcal{D}} :: (a \rightarrow b) \rightarrow \mathcal{D} \text{ a } b$

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

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) — $\mathcal{D}^+ id = \lambda a \rightarrow (id\ a, id)$
- (DP.2) —

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

$$(c, g' \circ f')$$

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

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

Dedução da instância

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

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

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

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

Dedução da instância

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

$\text{linearD} :: (a \rightarrow b) \rightarrow \mathcal{D} \ a \ b$

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

Instância da categoria que deduzimos

`instance Category \mathcal{D} where`

`id = linearD id`

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

Prova da instância

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

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

Prova de (C.1)

$$\begin{aligned} & \text{id} \circ \hat{\mathcal{D}} \\ &= \hat{\mathcal{D}} \text{id} \circ \hat{\mathcal{D}} f \text{ - lei functor de id (especificação de } \hat{\mathcal{D}}) \\ &= \hat{\mathcal{D}} (\text{id} \circ f) \text{ - lei functor para } (\circ) \\ &= \hat{\mathcal{D}} f \text{ - lei de categoria} \end{aligned}$$

Prova da instância

Prova de (C.2)

$$\begin{aligned} & \hat{D} h \circ (\hat{D} g \circ \hat{D} f) \\ &= \hat{D} h \circ \hat{D} (g \circ f) - \text{lei functor para } (\circ) \\ &= \hat{D} (h \circ (g \circ f)) - \text{lei functor para } (\circ) \\ &= \hat{D} ((h \circ g) \circ f) - \text{lei de categoria} \\ &= \hat{D} (h \circ g) \circ \hat{D} f - \text{lei functor para } (\circ) \\ &= (\hat{D} h \circ \hat{D} g) \circ \hat{D} f - \text{lei functor para } (\circ) \end{aligned}$$

Nota

Estas provas não requerem nada de \mathcal{D} e \hat{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 funtores monoidais

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

```
class Category k ⇒ Monoidal k
  where
    (×)::(a'k'c)→(b'k'd)→((a×b)'k'(c×d))
    instance Monoidal (→)
      where
        f × g = λ(a,b)→(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:

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

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:

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

e esta condição é suficiente para obtermos a nossa instância.

Dedução da instância

Instância da categoria que deduzimos

instance *Monoidal* \mathcal{D} where

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

Categorias e funtores cartesianas

class Monoidal $k \Rightarrow \text{Cartesean}$
k where

exl :: $(a \times b) \rightarrow k' a$

exr :: $(a \times b) \rightarrow k' b$

dup :: $a \rightarrow k' (a \times a)$

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
- $F \text{ exl} = \text{exl}$
- $F \text{ exp} = \text{exp}$
- $F \text{ dup} = \text{dup}$

Dedução da instância

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

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

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

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

Após esta dedução podemos continuar a determinar a instância:

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

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

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

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* *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* $k \Rightarrow \text{Cocartesian } k$ where:

inl :: $a'k'(a \times b)$

inlr :: $b'k'(a \times b)$

jam :: $(a \times a)'k'a$

Funtores cocartesianos

Definição de functor cocartesiano

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

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

Fork and Join

- $\Delta :: \text{Cartesian } k \Rightarrow (a \text{ ' } k' \text{ } c) \rightarrow (a \text{ ' } k' \text{ } d) \rightarrow (a \text{ ' } k' \text{ } (c \times d))$
- $\nabla :: \text{Cartesian } k \Rightarrow (c \text{ ' } k' \text{ } a) \rightarrow (d \text{ ' } k' \text{ } a) \rightarrow ((c \times d) \text{ ' } k' \text{ } a)$

Instance of \rightarrow^+

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

instance *Category* (\rightarrow^+) **where**

type *Obj* $(\rightarrow^+) = \text{Additive}$

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*

Instance of \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 = $\lambda a \rightarrow (a, 0)$

inrF = $\lambda b \rightarrow (0, b)$

jamF = $\lambda(a, b) \rightarrow a + b$

NumCat definition

class *NumCat* *k a* **where**

negateC :: *a* ' *k* ' *a*

addC :: (*a* × *a*) ' *k* ' *a*

mulC :: (*a* × *a*) ' *k* ' *a*

...

instance *Num a* ⇒ *NumCat* (→) *a* **where**

negateC = *negate*

addC = *uncurry* (+)

mulC = *uncurry* (*)

...

$$\mathcal{D} (\textit{negate } u) = \textit{negate } (\mathcal{D} u)$$

$$\mathcal{D} (u + v) = \mathcal{D} u + \mathcal{D} v$$

$$\mathcal{D} (u * v) = u * \mathcal{D} v + v * \mathcal{D} u$$

- Imprecise on the nature of u and v .
- A precise and simpler definition would be to differentiate the operations themselves.

class *Scalable* *k a* **where**

scale :: $a \rightarrow (a' \text{ ' } k \text{ ' } a)$

instance *Num* *a* \Rightarrow *Scalable* (\rightarrow^+) *a* **where**

scale *a* = *AddFun* $(\lambda da \rightarrow a * da)$

instance *NumCat* *D* **where**

negateC = *linearD* *negateC*

addC = *linearD* *addC*

mulC = *D* $(\lambda(a, b) \rightarrow (a * b, \text{scale } b \nabla \text{scale } a))$

Generalizing Automatic Differentiation

newtype $D\ k\ a\ b = D\ (a \rightarrow b \times (a' \ k' b))$

linearD :: $(a \rightarrow b) \rightarrow (a' \ k' b) \rightarrow D\ k\ a\ b$

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

instance *Category* $k \Rightarrow \text{Category}\ (D\ k)$ **where**

type *Obj* $(D\ k) = \text{Additive} \wedge \text{Obj}\ k \dots$

instance *Monoidal* $k \Rightarrow \text{Monoidal}\ (D\ k)$ **where** ...

instance *Cartesian* $k \Rightarrow \text{Cartesian}\ (D\ k)$ **where** ...

instance *Cocartesian* $k \Rightarrow \text{Cocartesian}\ (D\ k)$ **where**

inl = *linearD inlF inl*

inr = *linearD inrF inr*

jam = *linearD jamF jam*

instance *Scalable* $k \Rightarrow \text{NumCat } (D \ k) \ s$ **where**
negateC = *linearD negateC negateC*
addC = *linearD addC addC*
mulC = $D (\lambda(a, b) \rightarrow (a * b, \text{scale } b \nabla \text{scale } a))$

Nelson
Categories
Fork and Join
Operacoes Numericas
Generalizing Automatic Differentiation
Exemplos
Generalizar

Exemplos

Nelson
Categories
Fork and Join
Operacoes Numericas
Generalizing Automatic Differentiation
Exemplos
Generalizar

Generalizar