...Machine Learning...

Artur Ezequiel Nelson

Universidade do Minho

26 de Abril

Indice

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)
($\circ$) :: (b'k'c) $\rightarrow$ (a'k'b) $\rightarrow$ (a'k'c)

instance *Category* ($\rightarrow$) where
id = λa $\rightarrow$ a
$g \circ f = \lambda$a $\rightarrow$ g (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)
(∘) :: (b'k'c) → (a'k'b) → (a'k'c)

instance *Category (→)* where
id = λa → a
$g \circ f = \lambda$a → 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) —- $\mathcal{D}^+ id$ = λa $\rightarrow$ (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:

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

Resolver a primeira equação é trivial(definir id da instância como sendo $\mathcal{D}(\lambda$a $\rightarrow$ (id a,id))).
A segunda equação será resolvida resolvendo uma condição mais geral: $\mathcal{D}g \circ \mathcal{D}f$ = $\mathcal{D}(\lambda a \rightarrow let\{(b, f')$ = f a; (c, g') = g b } in $(c, g' \circ f')$), 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
  $\mathcal{D}g \circ \mathcal{D}f = \mathcal{D}(\lambda a \rightarrow let\{(b, f') = \text{f a}; (c, g') = \text{g b}\}\ \text{in}\ (c, g' \circ f'))$

Prova da instância

Antes de continuarmos devemos verificar se esta instância obedece às leis (C.1) e (C.2).
Se considerarmos apenas morfismos $\hat{f}$ :: $\mathcal{D}$ 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

$(\times)$::(a'k'c)$\to$(b'k'd)$\to$((a$\times$b)'k'(c$\times$d))

instance *Monoidal* $(\to)$ where

$f \times g = \lambda$(a,b)$\to$(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) = λ(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 = $\mathcal{D}(\lambda$(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 = $\mathcal{D}$(λ(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 = λ(a,b) $\rightarrow$ a
  exr = λ(a,b) $\rightarrow$ b
  dup = λa $\rightarrow$ (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* $\Rightarrow$ *Cocartesian k* where:
inl :: a'k'(a$\times$b)
inlr:: b'k'(a$\times$b)
jam :: (a$\times$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 inl = inl
- F inr = inr
- F jam = jam

Fork and Join

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

Instance of $\rightarrow^{+}$

$$
\begin{array}{l}
\textbf{newtype}\ a \rightarrow^{+} b = \mathit{AddFun}\ (a \rightarrow b) \\
\textbf{instance}\ \mathit{Category}\ (\rightarrow^{+})\ \textbf{where} \\
\quad \textbf{type}\ \mathit{Obj}\ (\rightarrow^{+}) = \mathit{Additive} \\
\quad \mathit{id} = \mathit{AddFun}\ \mathit{id} \\
\quad \mathit{AddFun}\ g \circ \mathit{AddFun}\ f = \mathit{AddFun}\ (g \circ f) \\
\textbf{instance}\ \mathit{Monoidal}\ (\rightarrow^{+})\ \textbf{where} \\
\quad \mathit{AddFun}\ f \times \mathit{AddFun}\ g = \mathit{AddFun}\ (f \times g) \\
\textbf{instance}\ \mathit{Cartesian}\ (\rightarrow^{+})\ \textbf{where} \\
\quad \mathit{exl} = \mathit{AddFun}\ \mathit{exl} \\
\quad \mathit{exr} = \mathit{AddFun}\ \mathit{exr} \\
\quad \mathit{dup} = \mathit{AddFun}\ \mathit{dup}
\end{array}
$$

Instance of $\rightarrow^{+}$

$$
\begin{aligned}
&\textbf{instance } \mathit{Cocartesian}\ (\rightarrow^{+})\ \textbf{where}\\
&\quad \mathit{inl} = \mathit{AddFun}\ \mathit{inlF}\\
&\quad \mathit{inr} = \mathit{AddFun}\ \mathit{inrF}\\
&\quad \mathit{jam} = \mathit{AddFun}\ \mathit{jamF}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{inlF} :: \mathit{Additive}\ b \Rightarrow a \rightarrow a \times b\\
&\mathit{inrF} :: \mathit{Additive}\ a \Rightarrow b \rightarrow a \times b\\
&\mathit{jamF} :: \mathit{Additive}\ a \Rightarrow a \times a \rightarrow a
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{inlF} = \lambda a \rightarrow (a, 0)\\
&\mathit{inrF} = \lambda b \rightarrow (0, b)\\
&\mathit{jamF} = \lambda (a, b) \rightarrow a + b
\end{aligned}
$$

NumCat definition

$$
\begin{array}{l}
\textbf{class}\ \mathit{NumCat}\ k\ a\ \textbf{where}\\
\quad \mathit{negateC} :: a\ `k`\ a\\
\quad \mathit{addC} :: (a \times a)\ `k`\ a\\
\quad \mathit{mulC} :: (a \times a)\ `k`\ a\\
\quad \ldots\\
\textbf{instance}\ \mathit{Num}\ a \Rightarrow \mathit{NumCat}\ (\rightarrow)\ a\ \textbf{where}\\
\quad \mathit{negateC} = \mathit{negate}\\
\quad \mathit{addC} = \mathit{uncurry}\ (+)\\
\quad \mathit{mulC} = \mathit{uncurry}\ (*)\\
\quad \ldots
\end{array}
$$

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

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

$$
\begin{array}{l}
\textbf{class}\ \mathit{Scalable}\ k\ a\ \textbf{where} \\
\quad \mathit{scale} :: a \rightarrow (a\ \text{`}\ k\ \text{`}\ a) \\
\textbf{instance}\ \mathit{Num}\ a \Rightarrow \mathit{Scalable}\ (\rightarrow^{+})\ a\ \textbf{where} \\
\quad \mathit{scale}\ a = \mathit{AddFun}\ (\lambda da \rightarrow a * da) \\
\textbf{instance}\ \mathit{NumCat}\ D\ \textbf{where} \\
\quad \mathit{negateC} = \mathit{linearD}\ \mathit{negateC} \\
\quad \mathit{addC} = \mathit{linearD}\ \mathit{addC} \\
\quad \mathit{mulC} = D\ (\lambda(a, b) \rightarrow (a * b, \mathit{scale}\ b \triangledown \mathit{scale}\ a)) \\
\textbf{instance}\ \mathit{FloatingCat}\ D\ \textbf{where} \\
\quad \mathit{sinC} = D\ (\lambda a \rightarrow (\mathit{sin}\ a, \mathit{scale}\ (\mathit{cos}\ a))) \\
\quad \mathit{cosC} = D\ (\lambda a \rightarrow (\mathit{cos}\ a, \mathit{scale}\ (-\mathit{sin}\ a))) \\
\quad \mathit{expC} = D\ (\lambda a \rightarrow \textbf{let}\ e = \mathit{exp}\ a\ \textbf{in}\ (e, \mathit{scale}\ e)) \\
\quad \ldots
\end{array}
$$

Examples

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

With a compiler plugin we can obtain

$$sqr = mulC \circ (id \vartriangle id)$$
$$magSqr = addC \circ (mulC \circ (exl \vartriangle exl) \vartriangle mulC \circ (exr \vartriangle exr))$$
$$cosSinProd = (cosC \vartriangle sinC) \circ mulC$$

Generalizing Automatic Differentiation

$$\textbf{newtype}\ D_k\ a\ b = D\ (a \to b \times (a\ `k`\ b))$$

$$linearD :: (a \to b) \to (a\ `k`\ b) \to D_k\ a\ b$$
$$linearD\ f\ f' = D\ (\lambda a \to (f\ a, f'))$$

$$\textbf{instance}\ Category\ k \Rightarrow Category\ D_k\ \textbf{where}$$
$$\quad \textbf{type}\ Obj\ D_k = Additive \wedge Obj\ k\ \ldots$$

$$\textbf{instance}\ Monoidal\ k \Rightarrow Monoidal\ D_k\ \textbf{where}\ \ldots$$

$$\textbf{instance}\ Cartesian\ k \Rightarrow Cartesian\ D_k\ \textbf{where}\ \ldots$$

$$\textbf{instance}\ Cocartesian\ k \Rightarrow Cocartesian\ D_k\ \textbf{where}$$
$$\quad inl = linearD\ inlF\ inl$$
$$\quad inr = linearD\ inrF\ inr$$
$$\quad jam = linearD\ jamF\ jam$$

$$
\begin{aligned}
&\textbf{instance } \mathit{Scalable}\ k\ s \Rightarrow \mathit{NumCat}\ D_k\ s\ \textbf{where}\\
&\quad \mathit{negateC} = \mathit{linearD}\ \mathit{negateC}\ \mathit{negateC}\\
&\quad \mathit{addC} = \mathit{linearD}\ \mathit{addC}\ \mathit{addC}\\
&\quad \mathit{mulC} = D\ (\lambda(a, b) \rightarrow (a * b, \mathit{scale}\ b \triangledown \mathit{scale}\ a))
\end{aligned}
$$

- Practical applications often involves high-dimensional spaces.
- Binary products are a very inefficient and unwieldy way of encoding high-dimensional spaces.
- A practical alternative is to consider n-ary products as representable functors(?)

class *Category k* $\Rightarrow$ *MonoidalI k h* **where**
  crossI :: $h\ (a\ `k`\ b) \rightarrow (h\ a\ `k`\ h\ b)$
instance *Zip h* $\Rightarrow$ *MonoidalI* $(\rightarrow)$ *h* **where**
  crossI $=$ *zipWith id*

$$\begin{array}{l}
\textbf{class } \mathit{MonoidalI}\ k\ h \Rightarrow \mathit{CartesianI}\ k\ h \textbf{ where}\\
\quad \mathit{exl} \quad :: h\ (h\ a \text{ ` } k \text{ ` } a)\\
\quad \mathit{replI} :: a \text{ ` } k \text{ ` } h\ a
\end{array}$$

$$\begin{array}{l}
\textbf{class } (\mathit{Representable}\ h, \mathit{Zip}\ h, \mathit{Pointed}\ h) \Rightarrow\\
\quad \mathit{CartesianI}\ (\rightarrow)\ h \textbf{ where}\\
\quad \mathit{exl} = \mathit{tabulate}\ (\mathit{flip}\ \mathit{index})\\
\quad \mathit{replI} = \mathit{point}
\end{array}$$

- The following is not the class the author was thinking

$$\begin{array}{l}
\textbf{class } \mathit{Representable}\ h \textbf{ where}\\
\quad \textbf{type } \mathit{Rep}\ h :: *\\
\quad \mathit{tabulate} :: (\mathit{Rep}\ h \rightarrow a) \rightarrow h\ a\\
\quad \mathit{index} :: h\ a \rightarrow \mathit{Rep}\ h \rightarrow a
\end{array}$$

$$
\begin{array}{l}
\textbf{class } \mathit{MonoidalI}\ k\ h \Rightarrow \mathit{CocartesianI}\ k\ h \textbf{ where} \\
\quad \mathit{inI} :: h\ (a \text{ `} k \text{` } h\ a) \\
\quad \mathit{jamI} :: h\ a \text{ `} k \text{` } a \\
\textbf{instance } (\mathit{MonoidalI}\ k\ h, \mathit{Zip}\ h) \Rightarrow \mathit{MonoidalI}\ D_k\ h \textbf{ where} \\
\quad \mathit{crossI}\ \mathit{fs} = D\ ((\mathit{id} \times \mathit{crossI}) \circ \mathit{unzip} \circ \mathit{crossI}\ (\mathit{fmap}\ \mathit{unD}\ \mathit{fs})) \\
\textbf{instance } (\mathit{CocartesianI}\ (\rightarrow)\ h, \mathit{CartesianI}\ k\ h, \mathit{Zip}\ h) \Rightarrow \\
\quad \mathit{CartesianI}\ D_k\ h \textbf{ where} \\
\quad \mathit{exI} = \mathit{linearD}\ \mathit{exI}\ \mathit{exI} \\
\quad \mathit{replI} = \mathit{zipWith}\ \mathit{linearD}\ \mathit{replI}\ \mathit{replI} \\
\textbf{instance } (\mathit{CocartesianI}\ k\ h, \mathit{Zip}\ h) \Rightarrow \mathit{CocartesianI}\ D_k\ h \textbf{ where} \\
\quad \mathit{inI} = \mathit{zipWith}\ \mathit{linearD}\ \mathit{inIF}\ \mathit{inI} \\
\quad \mathit{jamI} = \mathit{linearD}\ \mathit{sum}\ \mathit{jamI}
\end{array}
$$

- Suggests that some of the work refered does just a part of this paper.
- This paper was a continuation of the [Elliot 2017]
- Suggests that this implementation is simple, efficient, it can free memory dinamically (RAD) and is naturally parallel.
- Future work are detailed performace analysis; higher-order differentiation and automatic incrementation (continuing previous work [Elliot 2017])