Simple essence of AD

Artur Ezequiel Nelson

Universidade do Minho

24 de Maio

Index

Relembrar

Conceitos base

Objetivo

Introdução breve a categorias e funtores

Adaptação de definições

Dedução de instâncias em categorias

Instâncias deduzidas

Algoritmo AD generalizado

RAD e FAD generalizados

Scaling up

Bibliografia



Relembrar

Documento estudado: "The simple essence of automatic differentiation"

Objetivo: criar uma formalização genérica do conceito de ML para aprendizagem supervisionada

Definição de \mathcal{D} e conversão em \mathcal{D}^+

 \mathcal{D} - aproximação linear de uma função

Definition

Let $f :: a \to b$ be a function, where a and b are vectorial spaces that share a common underlying field. The first derivative definition is the following:

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

 \mathcal{D}^+ - versão de D com composição eficiente

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

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

Corolários associados a \mathcal{D}^+

Corolário 1.1

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

Corolário 2.1

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

Corolário 3.1

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

Objetivo

Criar uma implementação de \mathcal{D}^+ através da transcrição dos seus corolários para teoria de categorias de modo a obter um algoritmo generalizado para AD.

Categorias e funtores

Categoria: conjunto de objetos e morfismos com duas operações base(id e composição) e 2 regras:

- $id \circ f = f \circ id = f$
- $f \circ (g \circ h) = (f \circ g) \circ h$

Funtor: mapeia uma categoria noutra, preservando a estrutura

- Dado um objeto $t \in \mathcal{U}$ existe um objeto correspondente F $t \in \mathcal{V}$
- Dado um morfismo m :: $a \to b \in \mathcal{U}$ existe um morfismo correspondente F m :: F $a \to F$ $b \in \mathcal{V}$
- F id $(\in \mathcal{U})$ = id $(\in \mathcal{V})$
- $F(f \circ g) = Ff \circ Fg$

Adaptação de definições

Definição de tipo \mathcal{D}

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

Definição de $\hat{\mathcal{D}}$

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

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

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

Passos para obter a instância a partir da definição do funtor

- Passo 1 Assumir que $\hat{\mathcal{D}}$ é funtor de uma instância de \mathcal{D} a determinar
- Passo 2 Substituir pelo que determinamos nos corolários
- Passo 3 Generalizar condições se necessário para obtermos instância

Este processo é o mesmo para vários tipos de categorias.

Exemplo para categorias

Passo 1

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

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

Passo 2

$$egin{aligned} id &= \mathcal{D} \ (\lambda a
ightarrow (id \ a, id)) \ \hat{\mathcal{D}} \ g \circ \hat{\mathcal{D}} \ f &= \mathcal{D} \ (\lambda a
ightarrow \mathbf{let} \ \{(b, f') = \mathcal{D}^+ \ f \ a; (c, g') = \mathcal{D}^+ \ g \ b\} \ \mathbf{in} \ (c, g' \circ f')) \end{aligned}$$

Passo 3

Para a nossa instância a primeira equação que determinamos serve como definição da identidade.

Para definir a composição generalizamos a condição:

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

Definição da classe de categoria

class Category k where

id :: (a'k'a)

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

Instância deduzida para a categoria

instance $Category \mathcal{D}$ where

id = linearD id

 $\mathcal{D} g \circ \mathcal{D} f =$

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

Definição da classe de categoria monoidal

class Category
$$k \Rightarrow Monoidal \ k$$
 where $(\times) :: (a' \ k' \ c) \rightarrow (b' \ k' \ d) \rightarrow ((a \times b)' \ k' \ (c \times d))$

Instância deduzida para a categoria monoidal

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

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

Definição da classe de categoria cartesiana

class Monoidal $k \Rightarrow Cartesian \ k$ where exl :: (a, b) ' k' a exr :: (a, b) ' k' b dup :: a ' k' (a, a)

Instância deduzida para a categoria cartesiana

instance Cartesian D where
 exl = linearD exl
 exr = linearD exr
 dup = linearD dup

Definição da classe de categoria cocartesiana

```
class Category k \Rightarrow Cocartesian k where inl :: a' k' (a,b) inr :: b' k' (a,b) jam :: (a,a)' k' a
```

Instâncias que deduzimos - caso (\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
```

Instâncias que deduzimos - caso (\rightarrow^+)

```
instance Cocartesian (\rightarrow^+) where
   inl = AddFun inlF
   inr = AddFun inrF
   iam = AddFun jamF
in F: 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)
iamF = \lambda(a, b) \rightarrow a + b
```

Instância deduzida para AD genérico

```
newtype D_k a b = D (a \rightarrow b \times (a'k'b))
linearD :: (a \rightarrow b) \rightarrow (a'k'b) \rightarrow D_k ab
linearD f f' = D (\lambda a \rightarrow (f a, f'))
instance Category k \Rightarrow Category D_k where
  type Obj D_k = Additive \wedge Obj k ...
instance Monoidal k \Rightarrow Monoidal D_k where ...
instance Cartesian k \Rightarrow Cartesian D_k where ...
instance Cocartesian k \Rightarrow Cocartesian D_k where
  inl = linearD inlF inl
  inr = linearD inrF inr
  jam = linearD jamF iam
```

Generalização de AD

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

Generalizando RAD e FAD

Obter FAD e RAD de algoritmo AD genérico: forçar a direção da composição de morfismos

Conversão da escrita de funções

 $f :: a' k' b \Rightarrow (\circ f) :: (b' k' r) \rightarrow (a' k' r)$ para r objeto de categoria k.

Definição de novo tipo

newtype
$$Cont_k^r$$
 a b = $Cont$ $((b ' k' r) \rightarrow (a ' k' r))$

Funtor derivado dele

cont :: Category
$$k \Rightarrow (a' k' b) \rightarrow Cont_k^r a b$$
 cont $f = Cont(\circ f)$

Instância deduzida para RAD genérico

```
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<sub>k</sub> where
   Conf f \times Cont \ g = Cont \ (join \circ (f \times g) \circ unjoin)
instance Cartesian k \Rightarrow Cartesian Cont_{k}^{r} where
  exl = Cont (join \circ inl); exr = Cont (join \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 (join \circ dup)
instance Scalable k a \Rightarrow Scalable Cont_k^r a where
   scale s = Cont (scale s)
```

...