

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.



# Relembrar

Documento estudado: "The simple essence of automatic differentiation"

Objetivo: Criar uma formalização de uma ML para aprendizagem supervisionada genérica

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

$\mathcal{D}$  - aproximação linear de uma função

## Definition

Let  $f :: a \rightarrow b$  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  $\mathcal{D}$  com composição eficiente

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

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

# Corolários associados a $\mathcal{D}^+$

## Corolário 1.1

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

## Corolário 2.1

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

## 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  $m :: a \rightarrow b \in \mathcal{U}$  existe um morfismo correspondente  $F m :: F a \rightarrow F b \in \mathcal{V}$
- $F id (\in \mathcal{U}) = id (\in \mathcal{V})$
- $F (f \circ g) = F f \circ F g$





## Adaptação de definições

### Definição de tipo $\mathcal{D}$

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

### Definição de $\hat{\mathcal{D}}$

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

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

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

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

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

## 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

## 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

$$id = \mathcal{D} (\lambda a \rightarrow (id\ a, id))$$

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

### Passo 3

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

Para definir a composição generalizamos a condição:

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

## 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

$$id = \mathcal{D} (\lambda a \rightarrow (id\ a, id))$$

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

### Passo 3

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

Para definir a composição generalizamos a condição:

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

## 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

$$id = \mathcal{D} (\lambda a \rightarrow (id\ a, id))$$

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

### Passo 3

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

Para definir a composição generalizamos a condição:

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

## 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

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

## Definição da classe de categoria monoidal

```
class Category  $k \Rightarrow \text{Monoidal } k$  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**  
 $\mathcal{D} \ f \times \mathcal{D} \ g = \mathcal{D} \ (\lambda(a, b) \rightarrow \text{let } \{(c, f') = f \ a; (d, g') = g \ b\}$   
**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*  $k \Rightarrow \text{Cocartesian } k$  **where**

*inl* ::  $a \text{ ' } k \text{ ' } (a, b)$

*inr* ::  $b \text{ ' } k \text{ ' } (a, b)$

*jam* ::  $(a, a) \text{ ' } k \text{ ' } a$

## Instâncias que deduzimos - caso $(\rightarrow^+)$

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

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

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

$$id = AddFun\ id$$
$$\text{AddFun } g \circ \text{AddFun } f = \text{AddFun } (g \circ f)$$

**instance** *Monoidal*  $(\rightarrow^+)$  **where**

$$AddFun\ f \times AddFun\ g = AddFun\ (f \times g)$$

**instance** *Cartesian* ( $\rightarrow^+$ ) **where**

$$exl = AddFun\ exl$$
$$exr = AddFun\ exr$$
$$dup = AddFun\ dup$$



## Instância deduzida para AD genérico

**newtype**  $D_k$   $a \rightarrow b = D (a \rightarrow b \times (a \text{ ' } k \text{ ' } b))$

*linearD* ::  $(a \rightarrow b) \rightarrow (a \text{ ' } k \text{ ' } b) \rightarrow D_k a \rightarrow b$

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

**instance**  $Category \ k \Rightarrow Category \ D_k$  **where**

**type**  $Obj \ D_k = Additive \wedge Obj \ k \dots$

**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 jam*

## Generalização de AD

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

## Generalizando RAD e FAD

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

### Conversão da escrita de funções

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

### Definição de novo tipo

**newtype**  $Cont_k^f \text{ } a \text{ } b = Cont \ ((b' \text{ ' } k' \text{ ' } r) \rightarrow (a' \text{ ' } k' \text{ ' } r))$

### Functor derivado dele

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



## Generalizando RAD e FAD

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

## Conversão da escrita de funções

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

## Definição de novo tipo

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

Funtor derivado dele

$$\begin{aligned} cont &:: \text{Category } k \Rightarrow (a' \text{ k}' b) \rightarrow \text{Cont}_k^f a b \\ cont \, f &= \text{Cont} \, (\circ f) \end{aligned}$$





