# Simple essence of AD

Artur Ezequiel Nelson

Universidade do Minho

24 de Maio

# Index

# 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:

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

$\mathcal{D}^+$ - 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

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

## Corolário 2.1

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

## Corolário 3.1

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

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

## Definição de $\hat{\mathcal{D}}$

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

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

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

# 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

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

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

## Passo 3

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

Para definir a composição generalziamos a condição:

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

## Definição da classe de categoria

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

## Instancia deduzida para a categoria

```
instance Category 𝒟 where
  id = linearD id
  𝒟 g ∘ 𝒟 f =
  𝒟 (λa → let {(b, f') = f a; (c, g') = g b} in (c, g' ∘ f'))
```

## Definição da classe de categoria monoidal

$$\textbf{class}\ \mathit{Category}\ k \Rightarrow \mathit{Monoidal}\ k\ \textbf{where}$$
$$(\times) :: (a\ \texttt{`k`}\ c) \to (b\ \texttt{`k`}\ d) \to ((a \times b)\ \texttt{`k`}\ (c \times d))$$

## Instancia deduzida para a categoria monoidal

$$\textbf{instance}\ \mathit{Monoidal}\ \mathcal{D}\ \textbf{where}$$
$$\mathcal{D}\ f \times \mathcal{D}\ g = \mathcal{D}\ (\lambda(a, b) \to \textbf{let}\ \{(c, f') = f\ a; (d, g') = g\ b\}$$
$$\textbf{in}\ ((c, d), f' \times g'))$$

## Definição da classe de categoria cartesiana

```
class Monoidal k ⇒ Cartesian k where
  exl :: (a, b) `k` a
  exr :: (a, b) `k` b
  dup :: a `k` (a, a)
```

## Instancia 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** $a \rightarrow^+ b = \mathit{AddFun}\ (a \rightarrow b)$

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

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

  $\mathit{id} = \mathit{AddFun}\ \mathit{id}$

  $\mathit{AddFun}\ g \circ \mathit{AddFun}\ f = \mathit{AddFun}\ (g \circ f)$

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

  $\mathit{AddFun}\ f \times \mathit{AddFun}\ g = \mathit{AddFun}\ (f \times g)$

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

  $\mathit{exl} = \mathit{AddFun}\ \mathit{exl}$

  $\mathit{exr} = \mathit{AddFun}\ \mathit{exr}$

  $\mathit{dup} = \mathit{AddFun}\ \mathit{dup}$

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

**instance** *Cocartesian* $(\rightarrow^{+})$ **where**

$inl = AddFun\ inlF$

$inr = AddFun\ inrF$

$jam = AddFun\ jamF$

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

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

$jamF :: Additive\ a \Rightarrow a \times a \rightarrow a$

$inlF = \lambda a \rightarrow (a, 0)$

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

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

# Instancia deduzida para AD genérico

**newtype** $D_k\ a\ b = D\ (a \to b \times (a\ `k`\ b))$

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

**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**
  $inl = linearD\ inlF\ inl$
  $inr = linearD\ inrF\ inr$
  $jam = linearD\ jamF\ jam$

# 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

$f :: a\ \texttt{'k'}\ b \Rightarrow (\circ f) :: (b\ \texttt{'k'}\ r) \rightarrow (a\ \texttt{'k'}\ r)$ where r is any object of k.

## Definição de novo tipo

$$\textbf{newtype}\ \mathit{Cont}_k^r\ a\ b = \mathit{Cont}\ ((b\ \texttt{'k'}\ r) \rightarrow (a\ \texttt{'k'}\ r))$$

## Funtor derivado dele

$$\mathit{cont} :: \mathit{Category}\ k \Rightarrow (a\ \texttt{'k'}\ b) \rightarrow \mathit{Cont}_k^r\ a\ b$$
$$\mathit{cont}\ f = \mathit{Cont}\ (\circ f)$$

# Instancia 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** $Monoidal\ k \Rightarrow Monoidal\ Cont_k^r$ **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**
$scale\ s = Cont\ (scale\ s)$

...

...