Simple essence of AD

Artur Ezequiel Nelson

Universidade do Minho

29 de Maio

Index

Relembrar

Documento estudado: "The simple essence of automatic differentiation"([Elliott 2018][2])
Objetivo: Estudo e implementação de um algoritmo AD genérico

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

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

Definição

Seja $f :: a \to b$ uma função onde a e b são espaços vetoriais sobre um corpo comum. A primeira definição de derivada é:

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

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

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

Teoremas

Teorema 1

Sejam $f :: a \to b$ e $g :: b \to c$ duas funções. A derivada da composição f e g é:

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

Teorema 2

Sejam $f :: a \to b$ e $g :: b \to c$ duas funções. A regra de "cross" é a seguinte:

$$\mathcal{D}\,(f \times g)\,(a, b) = \mathcal{D}\,f\ a \times \mathcal{D}\,g\ b$$

Teorema 3

Para todas as funções lineares f, $\mathcal{D}\,f\ a = f$.

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

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

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

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

Passo 3

Para a nossa instância a primeira equação que determinamos serve como definição da identidade.
Para definir a composição generalizamos a condição:

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

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

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

Passo 3

Para a nossa instância a primeira equação que determinamos serve como definição da identidade.
Para definir a composição generalizamos a condição:

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

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

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

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

Passo 3

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

Para definir a composição generalizamos a condição:

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

Definição da classe de categoria

class *Category k* **where**
$id :: (a\ k\ a)$
$(\circ) :: (b\ k\ c) \to (a\ k\ b) \to (a\ k\ c)$

Instância deduzida para a categoria

instance *Category* $\mathcal{D}$ **where**
$id = linearD\ id$
$\mathcal{D}\ g \circ \mathcal{D}\ f =$
$\mathcal{D}\ (\lambda a \to \textbf{let}\ \{(b, f') = f\ a; (c, g') = g\ b\}\ \textbf{in}\ (c, g' \circ f'))$

Definição da classe de categoria monoidal

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

Instância 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)
```

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*

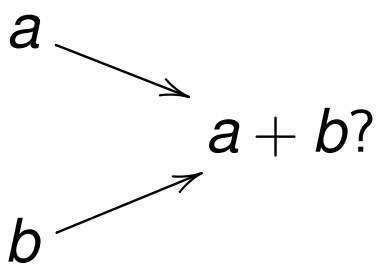
Dedução de$(\rightarrow^{+})$

- Queremos uma categoria que os seus objetos tenham adição e seja capaz de multiplicar por constantes (scaling).

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*

Dedução de$(\rightarrow^{+})$



- Queremos uma categoria que os seus objetos tenham adição e seja capaz de multiplicar por constantes (scaling).

Instância deduzida para AD genérico

Dedução de D_k

- Queremos ter um D_k tal que seja o $\mathcal{D}$ mas com uma categoria de genérica, pois nunca fizemos nada de específico com a $\multimap$.
- Queremos definir também as operações usuais nesta categoria, como a soma, multiplicação, sin, cos, etc.

Generalizando RAD e FAD

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

Conversão da escrita de morfismos

$f :: a\ \mathtt{`k`}\ b \Rightarrow (\circ f) :: (b\ \mathtt{`k`}\ r) \to (a\ \mathtt{`k`}\ r)$ para r objeto de categoria k.

Definição de novo tipo

$$\textbf{newtype}\ \mathit{Cont}^r_k\ a\ b = \mathit{Cont}\ ((b\ \mathtt{`k`}\ r) \to (a\ \mathtt{`k`}\ r))$$

Funtor derivado dele

$$\mathit{cont} :: \mathit{Category}\ k \Rightarrow (a\ \mathtt{`k`}\ b) \to \mathit{Cont}^r_k\ a\ b$$
$$\mathit{cont}\ f = \mathit{Cont}\ (\circ f)$$

Generalizando RAD e FAD

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

Conversão da escrita de morfismos

$f :: a\ \mathtt{`k`}\ b \Rightarrow (\circ f) :: (b\ \mathtt{`k`}\ r) \to (a\ \mathtt{`k`}\ r)$ para r objeto de categoria k.

Definição de novo tipo

$$\textbf{newtype}\ Cont^r_k\ a\ b = Cont\ ((b\ \mathtt{`k`}\ r) \to (a\ \mathtt{`k`}\ r))$$

Funtor derivado dele

$$cont :: Category\ k \Rightarrow (a\ \mathtt{`k`}\ b) \to Cont^r_k\ a\ b$$
$$cont\ f = Cont\ (\circ f)$$

Generalizando RAD e FAD

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

Conversão da escrita de morfismos

$f :: a\ \texttt{`k`}\ b \Rightarrow (\circ f) :: (b\ \texttt{`k`}\ r) \to (a\ \texttt{`k`}\ r)$ para r objeto de categoria k.

Definição de novo tipo

$$\textbf{newtype}\ \mathit{Cont}^r_k\ a\ b = \mathit{Cont}\ ((b\ \texttt{`k`}\ r) \to (a\ \texttt{`k`}\ r))$$

Funtor derivado dele

$$\mathit{cont} :: \mathit{Category}\ k \Rightarrow (a\ \texttt{`k`}\ b) \to \mathit{Cont}^r_k\ a\ b$$
$$\mathit{cont}\ f = \mathit{Cont}\ (\circ f)$$

Instância deduzida para RAD genérico

instance $Category\ k \Rightarrow Category\ Cont^r_k$ **where**
$id = Cont\ id$
$Cont\ g \circ Cont\ f = Cont\ (f \circ g)$

instance $Monoidal\ k \Rightarrow Monoidal\ Cont^r_k$ **where**
$Conf\ f \times Cont\ g = Cont\ (join \circ (f \times g) \circ unjoin)$

instance $Cartesian\ k \Rightarrow Cartesian\ Cont^r_k$ **where**
$exl = Cont\ (join \circ inl); exr = Cont\ (join \circ inr)$
$dup = Cont\ (jam \circ unjoin)$

instance $Cocartesian\ k \Rightarrow Cocartesian\ Cont^r_k$ **where**
$inl = Cont\ (exl \circ unjoin); inr = Cont\ (exr \circ unjoin)$
$jam = Cont\ (join \circ dup)$

instance $Scalable\ k\ a \Rightarrow Scalable\ Cont^r_k\ a$ **where**
$scale\ s = Cont\ (scale\ s)$

Demonstração prática

Demonstração prática
Link do projeto git: [LEI 2019] [1]

Projeto LEI.
https://github.com/Ezequiel-Moreira/LEI_2019_Uminho.
Accessed: 2019-05-28.

ELLIOTT, C.
The simple essence of automatic differentiation.
Proc. ACM Program. Lang. 2, ICFP (July 2018), 70:1–70:29.