Séminaire Cosynus, LIX

Typing Differentiable Programming

Marie Kerjean and Pierre-Marie Pédrot

Inria Rennes, Equipe gallinette

25 février 2020

Differentiable programming

Definition: programming with differential transformations.
" *a theoretical underpinning [of neural networks], even if only conceptual, would greatly accelerate progress* ".

Y. LeCun, abstract of a talk given at the IAS February 22nd 2019.

Differentiable programming

Definition: programming with differential transformations.
" *While a theoretical underpinning [of neural networks], even if only conceptual, would greatly accelerate progress , one must be conscious of the limited practical implications of general theories.* ".

Y. LeCun, abstract of a talk given at the IAS February 22nd 2019.

[Abadi Plotkin POPL20]
[Brunel Mazza Pagani POPL20]
[Elliot ICFP18]
[Wang and al. ICFP 19]

Differentiable programming

Definition: programming with differential transformations.

" *While a theoretical underpinning [of neural networks], even if only conceptual, would greatly accelerate progress , one must be conscious of the limited practical implications of general theories.* ".

Y. LeCun, abstract of a talk given at the IAS February 22nd 2019.

"Au coeur de tout langage de programmation il devrait y avoir un langage fonctionnel pur, de préférence typé , de préférence garantissant la terminaison"

Xavier Leroy, Conclusion du cours 2018/2019 au Collège de France

Differentiable programming

Definition: programming with differential transformations.
" *While a theoretical underpinning [of neural networks], even if only conceptual, would greatly accelerate progress , one must be conscious of the limited practical implications of general theories.* ".

Y. LeCun, abstract of a talk given at the IAS February 22nd 2019.

"Au coeur de tout langage de programmation il devrait y avoir un langage fonctionnel pur, de préférence typé, de préférence garantissant la terminaison"

Xavier Leroy, Conclusion du cours 2018/2019 au Collège de France

"De préférence typé"

Our work centers on finding a good type system for differentiable programming, typing a higher order differential transformation.

Lecun VS Logicians

Gödel Dialectica Transformation is Differentiable Programming.

Lecun VS Linear Logicians

Differential Linear Logic types a language expressing both forward and backward differentiation.

Curry-Howard-Lambek

Programs		**Logic**		**Categories**
Term	$\leftrightsquigarrow$	Proof	$\leftrightsquigarrow$	Morphisms
$\lambda x^A.t^B$		$\dfrac{\vdots}{A \vdash B}$		$f : A \to B$
Type		Formulas		Objects
Execution		*Cut - elimination*		Equality

In a future far far away : type theory allows to reason on basic computer algebra algorithms

Syntactical models (Pédrot)

Programs		**Logic**		**Dialectica**
Term	$\leftrightsquigarrow$	Proof	$\leftrightsquigarrow$	Other Proofs
$\lambda x^{A}.t^{B}$		$\dfrac{\vdots}{A \vdash B}$		$\dfrac{\vdots}{A \vdash B}$
Type		Formulas		Other Formulas
Execution		*Cut - elimination*		Equivalence

In a future far far away : type theory allows to reason on basic computer algebra algorithms

Smooth models (K.)

Programs		**Logic**		**Analysis**
Term	$\leftrightsquigarrow$	Proof	$\leftrightsquigarrow$	Smooth maps
$\lambda x^A.t^B$		$\dfrac{\vdots}{A \vdash B}$		$f : A \to B$
Type		Formulas		Spaces
Execution		*Cut - elimination*		Equality

In a future far far away : type theory allows to reason on basic computer algebra algorithms

Preliminaries

- Automatic Differentiation.
- Linear Logic.
- Differential λ-calculus.

Automatic Differentiation

How does one compute the differentiation of an algebraic expression, computed as a sequence of elementary operations ?

E.g. : $z = y + cos(x)^2$

$$\begin{array}{ll} x_1 = x_0^2 & x_1' = 2x_0x_0' \\ x_2 = cos(x_0) & x_2' = -x_0' sin(x_0) \\ z = y + x_2 & z' = y' + 2x_2x_2' \end{array}$$

The computation of the final results requires the computation of the derivative of all partial computation. But in which order ?

Forward Mode differentiation: $(x_1, x_1') \to (x_2, x_2') \to (z, z')$.
Reverse Mode differentiation: $x_1 \to x_2 \to z \to z' \to x_2' \to x_1'$
while keeping formal the unknown derivative.

AD from a higher-order functional point of view

$$\mathrm{D}_u(f \circ g)(v) = D_{g(u)} f(D_u f(v))$$
$$\mathrm{D}_u(f \circ g) = D_{g(u)} f \circ D_u(f)$$

- **Forward Mode differentiation:**
  $g(u) \to D_u g \to f(g(u)) \to D_{g(u)} f \to D_{g(u)} f \circ D_u(f)$.
- **Reverse Mode differentiation:**
  $g(u) \to f(g(u)) \to D_{g(u)} f \to D_u g \to D_{g(u)} f \circ D_u(f)$

The choice of an algorithm is due to complexity considerations:

- Forward mode for $f : \mathbb{R} \to \mathbb{R}^n$.
- Reverse mode for $f : \mathbb{R}^n \to \mathbb{R}$

Linear logic

Usual Implication

A call-by-name translation

$$A \Rightarrow B = \; !A \multimap B$$

$$\mathcal{C}^{\infty}(A, B) \simeq \mathcal{L}(!A, B)$$

A proof is linear when it uses only once its hypothesis A.

Linear logic

Usual implication

Linear Implication

A call-by-name translation

$$A \Rightarrow B = \,!A \multimap B$$
$$\mathcal{C}^\infty(A,B) \simeq \mathcal{L}(!A,B)$$

A proof is linear when it uses only once its hypothesis A.

Linear logic

Usual implication

Linear implication

A call-by-name translation

$$A \Rightarrow B = !A \multimap B$$

$$\mathcal{C}^\infty(A, B) \simeq \mathcal{L}(!A, B)$$

Exponential

Smooth Semantics

A proof is linear when it uses only once its hypothesis A.

Differential λ-calculus [Ehrhard Regnier. 2004]

Inspired by denotational models of Linear Logic in vector spaces of sequences, it introduces a differentiation of λ-terms.

$D(\lambda x.t)$ is the linearization of $\lambda x.t$, it substitute x linearly, and then it remains a term t' where x is free.

Syntax:

$$\Lambda^d : S, T, U, V ::= 0 \mid s \mid s + T$$
$$\Lambda^s : s, t, u, v ::= x \mid \lambda x.s \mid sT \mid \mathrm{D}s \cdot t$$

Operational Semantics:

$$(\lambda x.s)T \to_\beta s[T/x]$$
$$\mathrm{D}(\lambda x.s) \cdot t \to_{\beta_D} \lambda x.\frac{\partial s}{\partial x} \cdot t$$

where $\frac{\partial s}{\partial x} \cdot t$ is the linear substitution of x by t in s.

The linear substitution ...

... which is not exactly a substitution

$$\frac{\partial y}{\partial x} \cdot T = \begin{cases} T \ if \ x = y \\ 0 \ otherwise \end{cases} \qquad \frac{\partial 0}{\partial x} \cdot T = 0$$

$$\frac{\partial}{\partial x}(\lambda y.s) \cdot T = \lambda y. \frac{\partial s}{\partial x} \cdot T \qquad \frac{\partial}{\partial x}(s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T$$

Differentiating composition:

$$\frac{\partial}{\partial x}(su) \cdot v = (\frac{\partial s}{\partial x} \cdot T)u + ...$$

If x is linear in u, it is not linear in su

The linear substitution ...

... which is not exactly a substitution

$$\frac{\partial y}{\partial x} \cdot T = \begin{cases} T \; if \; x = y \\ 0 \; otherwise \end{cases} \qquad \frac{\partial 0}{\partial x} \cdot T = 0$$

$$\frac{\partial}{\partial x}(\lambda y.s) \cdot T = \lambda y. \frac{\partial s}{\partial x} \cdot T \qquad \frac{\partial}{\partial x}(s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T$$

Differentiating composition:

$$\frac{\partial}{\partial x}(su) \cdot v = (\frac{\partial s}{\partial x} \cdot T)u + ...$$

But x can be free in v. In that case, we do what we would have done in differential geometry :

The linear substitution ...

... which is not exactly a substitution

$$\frac{\partial y}{\partial x} \cdot T = \{ \begin{array}{l} T \; if \; x = y \\ 0 \; otherwise \end{array} \qquad \frac{\partial 0}{\partial x} \cdot T = 0$$

$$\frac{\partial}{\partial x}(\lambda y.s) \cdot T = \lambda y.\frac{\partial s}{\partial x} \cdot T \qquad \frac{\partial}{\partial x}(s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T$$

Differentiating composition:

$$\frac{\partial}{\partial x}(su) \cdot v = (\frac{\partial s}{\partial x} \cdot T)u + (\mathrm{D}s \cdot (\frac{\partial u}{\partial x} \cdot v)u)$$

Remember : We reverse the notations.

$$\frac{\partial f \circ g}{\partial x} v = D_(g(v))f \left(\frac{\partial g}{\partial x}(v) \right)$$

The linear substitution ...

... which is not exactly a substitution

$$\frac{\partial y}{\partial x} \cdot T = \{ \begin{array}{l} T \ if \ x = y \\ 0 \ otherwise \end{array} \qquad \frac{\partial 0}{\partial x} \cdot T = 0$$

$$\frac{\partial}{\partial x}(\lambda y.s) \cdot T = \lambda y.\frac{\partial s}{\partial x} \cdot T \qquad \frac{\partial}{\partial x}(s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T$$

Differentiating composition:

$$\frac{\partial}{\partial x}(su) \cdot v = (\frac{\partial s}{\partial x} \cdot T)u + \overbrace{(\mathrm{D}s\cdot}^{\text{Linear application}} \underbrace{(\frac{\partial u}{\partial x} \cdot v)}_{\text{Linear subsitution}} u)$$

Dialectica : Gödel doing Deep Learning

A Dialectica Transformation

Gödel Dialectica transformation [1958] : a translation from intuitionistic arithmetic to primitive recursive arithmetic.

$$A \rightsquigarrow \exists u : \mathbb{W}(A), \forall x : \mathbb{C}(A), A^D[u, x]$$

DePaiva [1991]: the linearized Dialectica translation operates on Linear Logic (types) and λ-calculus (terms).

A Dialectica Transformation

Gödel Dialectica transformation [1958] : a translation from intuitionistic arithmetic to primitive recursive arithmetic.

$$A \rightsquigarrow \exists u : \mathbb{W}(A), \forall x : \mathbb{C}(A), A^D[u, x]$$

DePaiva [1991]: the linearized Dialectica translation operates on Linear Logic (types) and λ-calculus (terms).

[Pedrot, CLS-LICS2014]

A linearized Dialectica translation preserving β-equivalence, via the introduction of an "abstract multiset constructor" on types on the target.

A Dialectica Transformation

Gödel Dialectica transformation [1958] : a translation from intuitionistic arithmetic to primitive recursive arithmetic.

$$A \rightsquigarrow \exists u : \mathbb{W}(A), \forall x : \mathbb{C}(A), A^D[u, x]$$

DePaiva [1991]: the linearized Dialectica translation operates on Linear Logic (types) and λ-calculus (terms).

[Pedrot, CLS-LICS2014]

A linearized Dialectica translation preserving β-equivalence, via the introduction of an "abstract multiset constructor" on types on the target.

$\rightsquigarrow$ Dialectica as a program translation ... whose abstract multiset is *not smooth enough*

Pédrot Dialectica Transformation

At the source : λ-calculus typed with minimal logic.
At the target : λ-calculus with pairs and an $\mathfrak{M}$ operation.

$$
\begin{array}{lcl}
\mathbb{W}(\alpha) & := & \alpha_{\mathbb{W}} \\
\mathbb{C}(\alpha) & := & \alpha_{\mathbb{C}} \\
\mathbb{W}(A \Rightarrow B) & := & (\mathbb{W}(A) \Rightarrow \mathbb{W}(B)) \times (\mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathfrak{M}\,\mathbb{C}(A)) \\
\mathbb{C}(A \Rightarrow B) & := & \mathbb{W}(A) \times \mathbb{C}(B)
\end{array}
$$

$$
\begin{array}{lcl}
x_x & := & \lambda\pi.\,\{\pi\} \\
x_y & := & \lambda\pi.\,\varnothing \quad \text{if } x \neq y \\
(\lambda x.\,t)_y & := & \lambda\pi.\,(\lambda x.\,t_y)\;\pi.1\;\pi.2
\end{array}
\qquad
\begin{array}{lcl}
x^\bullet & := & x \\
(\lambda x.\,t)^\bullet & := & (\lambda x.\,t^\bullet, \lambda x\pi.\,t_x\;\pi) \\
(t\;u)^\bullet & := & (t^\bullet.1)\;u^\bullet
\end{array}
$$

$$
(t\;u)_y := \lambda\pi.\,(t_y\,(u^\bullet, \pi)) \circledast ((t^\bullet.2)\,u^\bullet\,\pi \gg\!= u_y)
$$

Pédrot Dialectica Transformation

At the source : λ-calculus typed with minimal logic.
At the target : λ-calculus with pairs and an $\mathfrak{M}$ operation.

Soundness [Ped14]

If $\Gamma \vdash t : A$ in the source then we have in the target

- $\mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A)$
- $\mathbb{W}(\Gamma) \vdash t_x : \mathbb{C}(A) \Rightarrow \mathfrak{M}\,\mathbb{C}(X)$ provided $x : X \in \Gamma$.

Tracking differentiation in Dialectica

$$\frac{}{\Gamma \vdash \varnothing : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash m_1 : \mathfrak{M}\, A \qquad \Gamma \vdash m_2 : \mathfrak{M}\, A}{\Gamma \vdash m_1 \circledast m_2 : \mathfrak{M}\, A}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \{t\} : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash m : \mathfrak{M}\, A \qquad \Gamma \vdash f : A \Rightarrow \mathfrak{M}\, B}{\Gamma \vdash m \gg\!= f : \mathfrak{M}\, B}$$

$$\begin{array}{lll} x_x & := & \lambda\pi.\, \{\pi\} \\ x_y & := & \lambda\pi.\, \varnothing \quad \text{if } x \neq y \\ (\lambda x.\, t)_y & := & \lambda\pi.\, (\lambda x.\, t_y)\ \pi.1\ \pi.2 \end{array} \qquad \begin{array}{lll} x^\bullet & := & x \\ (\lambda x.\, t)^\bullet & := & (\lambda x.\, t^\bullet, \lambda x\pi.\, t_x\ \pi) \\ (t\ u)^\bullet & := & (t^\bullet.1)\ u^\bullet \end{array}$$

$$(t\ u)_y := \lambda\pi.\, (t_y\ (u^\bullet, \pi)) \circledast ((t^\bullet.2)\ u^\bullet\ \pi \gg\!= u_y)$$

Tracking differentiation in Dialectica

$$\frac{}{\Gamma \vdash \varnothing : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash m_1 : \mathfrak{M}\, A \qquad \Gamma \vdash m_2 : \mathfrak{M}\, A}{\Gamma \vdash m_1 \circledast m_2 : \mathfrak{M}\, A}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \{t\} : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash m : \mathfrak{M}\, A \qquad \Gamma \vdash f : A \Rightarrow \mathfrak{M}\, B}{\Gamma \vdash m \gg\!= f : \mathfrak{M}\, B}$$

Differential λ-calculus

x_x	$:=$	$\lambda\pi.\,\{\pi\}$	$x^\bullet$	$:=$	x
x_y	$:=$	$\lambda\pi.\,\varnothing$ if $x \neq y$	$(\lambda x.\,t)^\bullet$	$:=$	$(\lambda x.\,t^\bullet, \lambda x \pi.\, t_x\ \pi)$
$(\lambda x.\,t)_y$	$:=$	$\lambda\pi.\,(\lambda x.\,t_y)\ \pi.1\ \pi.2$	$(t\ u)^\bullet$	$:=$	$(t^\bullet.1)\ u^\bullet$

$$(t\ u)_y := \lambda\pi.\,(t_y\ (u^\bullet, \pi)) \circledast ((t^\bullet.2)\ u^\bullet\ \pi \gg\!= u_y)$$

Tracking differentiation in Dialectica

$$\frac{}{\Gamma \vdash \varnothing : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash m_1 : \mathfrak{M}\, A \qquad \Gamma \vdash m_2 : \mathfrak{M}\, A}{\Gamma \vdash m_1 \circledast m_2 : \mathfrak{M}\, A}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \{t\} : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash m : \mathfrak{M}\, A \qquad \Gamma \vdash f : A \Rightarrow \mathfrak{M}\, B}{\Gamma \vdash m \gg\!= f : \mathfrak{M}\, B}$$

Differential λ-calculus

$$\begin{array}{llll} x_x & := & \lambda\pi.\, \frac{\partial x}{\partial x} \cdot \pi & \\ x_y & := & \lambda\pi.\, \frac{\partial x}{\partial y} \cdot \pi & \text{if } x \neq y \\ (\lambda x.\, t)_y & := & \lambda\pi.\, (\lambda x.\, t_y)\ \pi.1\ \pi.2 & \end{array} \qquad \begin{array}{lll} x^\bullet & := & x \\ (\lambda x.\, t)^\bullet & := & (\lambda x.\, t^\bullet, \lambda x\pi.\, t_x\ \pi) \\ (t\ u)^\bullet & := & (t^\bullet.1)\ u^\bullet \end{array}$$

$$(t\ u)_y := \lambda\pi.\, (t_y\ (u^\bullet, \pi)) \circledast ((t^\bullet.2)\ u^\bullet\ \pi \gg\!= u_y)$$

Tracking differentiation in Dialectica

$$\frac{}{\Gamma \vdash \varnothing : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash m_1 : \mathfrak{M}\, A \qquad \Gamma \vdash m_2 : \mathfrak{M}\, A}{\Gamma \vdash m_1 \circledast m_2 : \mathfrak{M}\, A}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \{t\} : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash m : \mathfrak{M}\, A \qquad \Gamma \vdash f : A \Rightarrow \mathfrak{M}\, B}{\Gamma \vdash m \gg= f : \mathfrak{M}\, B}$$

Differential λ-calculus

$$\begin{array}{lll lll}
x_x & := & \lambda \pi.\, \frac{\partial x}{\partial x} \cdot \pi & x^\bullet & := & x \\
x_y & := & \lambda \pi.\, \frac{\partial x}{\partial y} \cdot \pi \quad \text{if } x \neq y & (\lambda x.\, t)^\bullet & := & (\lambda x.\, t^\bullet, \lambda x \pi.\, t_x\ \pi) \\
(\lambda x.\, t)_y & := & \lambda \pi.\, (\lambda x.\, t_y)\ \pi.1\ \pi.2 & (t\ u)^\bullet & \equiv & (\lambda x.\, (t x)^\bullet)\ u^\bullet
\end{array}$$

$$(t\ u)_y := \lambda \pi.\, (t_y\ (u^\bullet, \pi)) \circledast ((t^\bullet.2)\ u^\bullet\ \pi \gg= u_y)$$

Backpropagation

Differential λ-calculus in a hurry

$$\mathrm{D}(\lambda x.s) \cdot t \to_{\beta_D} \lambda x.\frac{\partial s}{\partial x} \cdot t$$

$$\frac{\partial y}{\partial x} \cdot T = \begin{cases} T \ if \ x = y \\ 0 \ otherwise \end{cases} \qquad \frac{\partial}{\partial x}(sU) \cdot T = (\frac{\partial s}{\partial x} \cdot T)U + (\mathrm{D}s \cdot (\frac{\partial U}{\partial x} \cdot T))U$$

$$\frac{\partial}{\partial x}(\lambda y.s) \cdot T = \lambda y.\frac{\partial s}{\partial x} \cdot T \qquad \frac{\partial}{\partial x}(\mathrm{D}s \cdot u) \cdot T = \mathrm{D}(\frac{\partial s}{\partial x} \cdot T) \cdot u + \mathrm{D}s \cdot (\frac{\partial u}{\partial x} \cdot T)$$

$$\frac{\partial 0}{\partial x} \cdot T = 0 \qquad \frac{\partial}{\partial x}(s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T$$

Dialectica is Differentiation

The linearized Dialectica Translation weakens to a transformation from λ-calculus to Differential λ-calculus.

Differential calculus is typed with minimal logic and does not distinguish a specific types on which the formal sum $*$ applies :

$$[\![\mathfrak{M}\, A]\!] = A$$
$$[\![\emptyset]\!] := 0 \qquad [\![t \circledast u]\!] := [\![t]\!] + [\![u]\!] \qquad [\![\{t\}]\!] := [\![t]\!].$$

Proposition

Consider two λ-terms t and u. Then $[\![t_x]\!]u \equiv \frac{\partial t}{\partial x} \cdot u$ and $((\lambda x.t)^{\bullet}.2))u \equiv Dt \cdot u$.

Dialectica enriched with real functions

We now enrich both our source and target λ-calculi with a type of reals $\mathbb{R}$. We assume furthermore that the source contains functions symbols $\varphi, \psi, \ldots : \mathbb{R} \to \mathbb{R}$ with derivative $\varphi', \psi', \ldots$.

$$\mathbb{W}(\mathbb{R}) := \mathbb{R} \qquad \mathbb{C}(\mathbb{R}) := 1$$

$$\varphi^\bullet := (\varphi, \lambda\alpha\,\pi.\,\{() \mapsto \varphi'(\alpha)\}) \qquad \varphi_x := \lambda\pi.\,\varnothing$$

The soundness theorem is then adapted trivially.

Soundness Theorem

The following equation holds in the target.

$$(\varphi_1 \circ \ldots \circ \varphi_n)^\bullet.2\ \alpha\ () \ \equiv \{() \mapsto (\varphi_1 \circ \ldots \circ \varphi_n)'(\alpha)\}$$

Dialectica is Backpropagation

When one distinguishes a *specific types for the codomain of functions*, on which the sums operate, we observe a cut-elimination mimicking the dynamic of backward differentiation.

$$A, B := \alpha \mid A \Rightarrow B \mid A \times B \mid A^{\perp} \mid \mathrm{Tr}(A)$$
$$t, u := x \mid (t)u \mid \lambda x.t \mid (t, u) \mid t \mid u \circledast v \mid \emptyset.$$

Types at the source : Minimal Logic and a type of Traces).

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \{t\} : \mathrm{Tr}(A)} \qquad \frac{\Gamma \vdash t : \mathrm{Tr}(A) \qquad \Gamma \vdash u : \mathrm{Tr}(A)}{\Gamma \vdash t \circledast u : \mathrm{Tr}(A)}$$

$$\frac{}{\Gamma \vdash \emptyset : \mathrm{Tr}(A)} \qquad \frac{\Gamma \vdash t : \mathrm{Tr}(A) \qquad \Gamma \vdash f : A \Rightarrow \mathrm{Tr}(B)}{\Gamma \vdash t \gg= u : \mathrm{Tr}(B)}$$

Dialectica is Backpropagation

When one distinguishes a *specific types for the codomain of functions*, on which the sums operate, we observe a cut-elimination mimicking the dynamic of backward differentiation.

$$A, B := \alpha \mid A \Rightarrow B \mid A \times B \mid A^{\perp} \mid \mathrm{Tr}(A)$$
$$t, u := x \mid (t)u \mid \lambda x.t \mid (t, u) \mid t \mid u \circledast v \mid \emptyset.$$

Two mutually inductively defined translations :

x_x	$:=$	$\lambda\pi.\,\{\pi\}$	$x^{\bullet}$	$:=$	x
x_y	$:=$	$\lambda\pi.\,\varnothing \quad \text{if } x \neq y$	$(\lambda x.\,t)^{\bullet}$	$:=$	$(\lambda x.\,t^{\bullet}, \lambda x\pi.\,t_x\ \pi)$
$(\lambda x.\,t)_y$	$:=$	$\lambda\pi.\,(\lambda x.\,t_y)\ \pi.1\ \pi.2$	$(t\ u)^{\bullet}$	$:=$	$(t^{\bullet}.1)\ u^{\bullet}$

$$(t\ u)_y := \lambda\pi.\,(t_y\,(u^{\bullet}, \pi)) \circledast ((t^{\bullet}.2)\,u^{\bullet}\ \pi \gg= u_y)$$

Dialectica is Backpropagation

When one distinguishes a *specific types for the codomain of functions*, on which the sums operate, we observe a cut-elimination mimicking the dynamic of backward differentiation.

$$A, B := \alpha \mid A \Rightarrow B \mid A \times B \mid A^{\perp} \mid \mathrm{Tr}(A)$$
$$t, u := x \mid (t)u \mid \lambda x.t \mid (t, u) \mid t \mid u \circledast v \mid \emptyset.$$

Two mutually inductively defined translations :

x_x	$:=$	$\lambda \pi.\, \{\pi\}$	$x^{\bullet}$	$:=$	x
x_y	$:=$	$\lambda \pi.\, \varnothing \quad \text{if } x \neq y$	$(\lambda x.\, t)^{\bullet}$	$:=$	$(\lambda x.\, t^{\bullet}, D(\lambda x.t))$
$(\lambda x.\, t)_y$	$:=$	$\lambda \pi.\, (\lambda x.\, t_y)\ \pi.1\ \pi.2$	$(t\ u)^{\bullet}$	$:=$	$(t^{\bullet}.1)\ u^{\bullet}$

$$(t\ u)_y := \lambda \pi.\, (t_y\ (u^{\bullet}, \pi)) \circledast ((Dt)\ u^{\bullet}\ \pi \gg\!= u_y)$$

Dialectica is Backpropagation

When one distinguishes a *specific types for the codomain of functions*, on which the sums operate, we observe a cut-elimination mimicking the dynamic of backward differentiation.

$$A, B := \alpha \mid A \Rightarrow B \mid A \times B \mid A^{\perp} \mid \mathrm{Tr}(A)$$
$$t, u := x \mid (t)u \mid \lambda x.t \mid (t, u) \mid t \mid u \circledast v \mid \emptyset.$$

Two typed differential transformations

When $\Gamma \vdash t : !A \multimap B$ and writing $Dt = (t^{\bullet}.2)$ we have:

$$\Gamma \vdash Dt : A \Rightarrow (B^{\perp} \Rightarrow \mathrm{Tr}(A^{\perp}))$$

$$\Gamma \vdash t_y : A \times B^{\perp} \Rightarrow \mathrm{Tr}(Y^{\perp})$$

Dialectica is Backpropagation

We reuse the arguments of Brunel, Mazza and Pagani: *Backpropagation is encoded through the contravariance of the differential arguments, which is typed by a linear dual* .

Consider $f : \mathbb{R}^n \to \mathbb{R}^m$.

$$\overrightarrow{D}(f) : \begin{cases} \mathbb{R}^n \times \mathbb{R}^m x \to \mathbb{R}^n \times \mathbb{R}^m \\ \qquad (a, x) \mapsto (f(a), D_a f \cdot x) \end{cases}$$

$$\overleftarrow{D}(f) : \begin{cases} \mathbb{R}^n \times \mathbb{R}^{m\perp} \to \in \mathbb{R}^m \times \mathbb{R}^{n\perp} \\ \qquad (a, x) \mapsto (f(a), (v \mapsto v \cdot (\mathrm{D}_a f \cdot x)) \end{cases}$$

- As in differential λ-calculus, the use of two separate differential transformation allows to go higher-order.

Dialectica is Backpropagation

We reuse the arguments of Brunel, Mazza and Pagani:
Backpropagation is encoded through the contravariance of the differential arguments, which is typed by a linear dual .

Consider $f : \mathbb{R}^n \to \mathbb{R}^m$.

$$\overrightarrow{D}(f) : \begin{cases} \mathbb{R}^n \times \mathbb{R}^m x \to \mathbb{R}^n \times \mathbb{R}^m \\ \qquad (a, x) \mapsto (f(a), D_a f \cdot x) \end{cases}$$

Consider $f : E \to F$.

$$\overleftarrow{D}(f) : \begin{cases} E \times F' \to F \times E' \\ \quad (a, \ell) \mapsto (f(a), (v \in F \mapsto (v \cdot (\mathrm{D}_a f \cdot x))) \end{cases}$$

- As in differential λ-calculus, the use of two separate differential transformation allows to go higher-order.

Lessons from Dialectica

- As in differential λ-calculus, the use of two distinct transformations allows to handle the differentiation of higher-order functions.
- As in [BMP20], encoding partial substitutions by *Linear duals* allow the encoding of backpropagation.
- This gives us a differential translation which can be enriched over dependant or positive types.
- Hint: call-by-name agrees with backpropagation.

$\rightsquigarrow$ towards a finer, internal handling of automatic differentiation as a reduction strategy.

Automatic Differentiation as a choice of reduction strategy

Refining λ-calculus with operations from distribution theory.

Just a glimpse at Differential Linear Logic

$$A, B := A \otimes B | 1 | A \parr B | \bot | A \oplus B | 0 | A \times B | \top | !A | !A$$

Exponential rules of DiLL$_0$

$$\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\, c \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\, w \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\, d$$

$$\frac{\vdash \Gamma, !A, \qquad \vdash \Delta, !A}{\vdash \Gamma, \Delta, !A}\, \bar{c} \qquad \frac{\vdash}{\vdash !A}\, \bar{w} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, !A}\, \bar{d}$$

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\, p$$

$\rightsquigarrow$ *A particular point of view on differentiation induced by duality.*

Normal functors, power series and λ-calculus. Girard, APAL(1988)

Differential interaction nets, Ehrhard and Regnier, TCS (2006)

Exponentials are distributions

$$[\![?A]\!] := \mathcal{C}^\infty([\![A]\!]', \mathbb{R})' \qquad [\![!A]\!] := \mathcal{C}^\infty([\![A]\!], \mathbb{R})'$$

functions *distributions*

A typical distribution is the dirac operator:

$$\delta : \begin{cases} E \to \mathcal{C}^\infty(E, \mathbb{R})' \\ x \mapsto (\phi \mapsto \phi(x)) \end{cases}$$

Exponential rules of DiLL$_0$

$$\frac{\vdash \Gamma, f : ?A, g : ?A}{\vdash \Gamma, f.g : ?A}\, c \qquad \frac{\vdash \Gamma}{\vdash \Gamma, cst_0 : ?A}\, w \qquad \frac{\vdash \Gamma, \ell : A}{\vdash \Gamma, \ell : ?A}\, d$$

$$\frac{\vdash \Gamma, \phi : !A, \qquad \vdash \Delta, \psi : !A}{\vdash \Gamma, \Delta, \phi * \psi : !A}\, \bar{c} \qquad \frac{}{\vdash \delta_0 : !A}\, \bar{w} \qquad \frac{\vdash \Gamma, v : A}{\vdash \Gamma, D_0(\_)(v) : !A}\, \bar{d}$$

$$\frac{\vdash ?\Gamma, v : A}{\vdash ?\Gamma, \delta_v : !A}\, p$$

A few operations typed by DiLL

The composition of linear functions:

$$\dfrac{\Gamma \vdash f : A \multimap B \qquad \Delta \vdash g : B \multimap C}{\Gamma, \Delta \vdash g \circ f : A \multimap C}\ \text{cut}$$

The composition of non-linear functions:

$$\dfrac{\dfrac{\Gamma \vdash f : !A \multimap B}{\Delta \vdash (x \mapsto \delta_{g(x)}) : !A \multimap !B}\ \text{p} \qquad \Delta \vdash g : !B \multimap C}{\Gamma, \Delta \vdash g \circ f = (x \mapsto \delta_{f(x)} g) : !A \multimap C}\ \text{cut}$$

The Differentiation of non-linear functions:

$$\dfrac{\Gamma \vdash f : !A \multimap B \qquad \dfrac{\vdash \Delta, v : A}{\vdash \Gamma, D_0(\_)(v) : !A}\ \bar{d}}{\Gamma, \Delta \vdash D_0(f)(v) : B}\ \text{cut}$$

Let's translate this into a term language typed by DiLL.

A few operations typed by DiLL

The chain rule is encoded in the interaction of diracs δ_x with differential arguments $D_u t$.

The composition of non-linear functions:

$$\dfrac{\dfrac{\Gamma \vdash f : !A \multimap B}{\Delta \vdash (x \mapsto \delta_{g(x)}) : !A \multimap !B}\,\mathrm{p} \qquad \Delta \vdash g : !B \multimap C}{\Gamma, \Delta \vdash g \circ f = (x \mapsto \delta_{f(x)} g) : !A \multimap C}\,\mathrm{cut}$$

The Differentiation of non-linear functions:

$$\dfrac{\Gamma \vdash f : !A \multimap B \qquad \dfrac{\vdash \Delta, v : A}{\vdash \Gamma, D_0(\_)(v) : !A}\,\bar{d}}{\Gamma, \Delta \vdash D_0(f)(v) : B}\,\mathrm{cut}$$

Let's translate this into a term language typed by DiLL.

A few operations typed by DiLL

The chain rule is encoded in the interaction of diracs δ_x with differential arguments $D_u t$.

The Chain rule:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash f : !A \multimap B}{\Delta \vdash (x \mapsto \delta_{g(x)}) : !A \multimap !B}\,p \qquad \Delta \vdash g : !B \multimap C}{\Gamma, \Delta \vdash g \circ \delta_f : !A \multimap C}\,\text{cut} \qquad \dfrac{\vdash \Delta', v : A}{\vdash \Gamma, D_0(\_)(v) : !A}\,\bar{d}}{\Gamma, \Delta, \Delta' \vdash D_0(g \circ f)(v) : c}\,\text{cut}$$

Let's translate this into a term language typed by DiLL.

A minimal language allowing to express automatic differentiation

Two class of terms:

$$u, v := x \mid t^{\perp} \mid u * v \mid \emptyset \mid u \otimes v \mid 1 \mid \delta_u \mid D_u(t) \mid \downarrow t$$
$$t, s := u^{\perp} \mid t \cdot s \mid w_1 : N \mid \lambda x.t \mid dx.t \mid \uparrow u$$

A function $\lambda x.t$ can be match to two kind of arguments: diracs δ_u or differential operators $\mathrm{D}_u t$.

$$(\lambda x.t)\delta_u \to t[u/x]$$
$$(\lambda x.t)D_t u \to \cdots$$

The differentiation $\lambda x.t$ of must be inductively defined on t:

$$(\lambda x.(t)u)D_w s \to \uparrow(\downarrow((\lambda x.t)D_w s)u * \downarrow(t((\lambda x.u)D_w s)))$$

Differentiating an application $(t)u$ is symmetric in t and u.

$$(\lambda x.\uparrow\delta_t)D_u s \to (\lambda z.\uparrow(D_z((\lambda x.t)D_u s)))((\lambda x.t)(u)))$$

The abstraction $\lambda x.\uparrow\delta_t$ will be composed with another abstraction and differentiation must take that into account.

Forward / Backward Differentiation as CBV/CBN

Then the differentiation of $(\lambda y.s) \circ (\lambda x.t)$ at a point $u = \delta_w$ according to a vector r computes as follows:

$$
\begin{aligned}
(\lambda x.((\lambda y.s)\delta_t))D_u r &\to {\uparrow}({\downarrow}((\lambda x.(\lambda y.s))D_u r)\delta_t * {\downarrow}((\lambda y.s)((\lambda x.\delta_t)D_u r))) \\
&\to^* {\uparrow}({\downarrow}({\uparrow}\emptyset) * {\downarrow}((\lambda y.s)((\lambda x.\delta_t)D_u r)))\text{as } x \text{ is free in } s \\
&\to^* (\lambda y.s)((\lambda x.\delta_t)D_u r)) \text{ by involutivity of the shifts} \\
&\to (\lambda y.s)(\lambda z.{\uparrow}(D_z((\lambda x.t)D_u r)))((\lambda x.t)(u))) \\
&\to ((\lambda y.s)(\lambda z.{\uparrow}(D_z((\lambda x.t)D_u r))))((t[w/x]))\text{as } u = \delta_w \\
&\to^* (\lambda y.s)D_v((\lambda x.t)D_u r) \text{ if } (t[w/x] \to^* \delta_v)
\end{aligned}
$$

The value of $t[w/x]$ is computed first-hand. Whether we proceed with the computation of the derivative of the first function $((\lambda x.t)D_u r)$ or to the derivative of the second $((\lambda y.s)D_v((\lambda x.t)D_u r))$ depends of the evaluation strategy.

Higher-order addition and Higher-order multiplication

Additions are done on the domain, through convolution (ie higher order addition).

$$\phi * \psi := f \mapsto \phi(x \mapsto \psi(y \mapsto f(x+y))$$
$$\delta_u * \delta_v \to \delta_{u*v}$$

Multiplications are done one the codomain, through contractions (ie higher order multiplication).

$$f \cdot g := x \mapsto f(x) \cdot g(x)$$
$$(\lambda y.t) \cdot (\lambda z.s) \to \lambda x.(t[x/y]) \cdot (s[x/z])$$

Distinguishing Linear and Non-Linear Maps

$$\frac{\vdash \mathbb{N}, t : M, x^{\perp} : (!P)^{\perp} \mid}{\vdash \mathbb{N}, \lambda x.t : (!P)^{\perp} \parr M \mid} (\lambda)$$

$$\frac{\vdash \mathbb{N}, x^{\perp} : P^{\perp}, t : M}{\vdash \mathbb{N}, dx.t : (!P)^{\perp} \parr M} (d)$$

Interpreting Dialectica in DILL

$$[\mathfrak{M}\, A] := !![A] \qquad [x] := x$$

$$[\lambda x.t] := \lambda x.[t] \qquad [(t,u)] := ([t],[u])$$

$$[\emptyset] := \uparrow\emptyset \qquad [\{t\}] := \uparrow(\delta_{\delta_{[t]}})$$

$$[u \circledast v] := \uparrow(\downarrow[u] * \downarrow[v]) \qquad [m \gg\!= f] := (dx.[f]x)[m]$$

A translation on top of Dialectica

If $\Gamma \vdash t : A$ in the target of Dialectica, then $:\mathbb{L}(\Gamma) \vdash [t] : \mathbb{L}(A)$ and if $t \equiv u$ in the target of Dialectica then $[t] \equiv [u]$ in our calculus.

A semantical point of view : if $\chi : \mathcal{C}^\infty(E,F) \simeq \mathcal{L}(!E,F)$ *then* $(\delta_{\delta_e}) \gg\!= f := \chi(f)(\delta_e)$.

Conclusion

What we have:

- Dialectica is a reverse-mode differential transformation.
- Differential Linear Logic gives a type-system for a higher-order functional language, in which forward and reverse mode differentiation identity to reduction strategies.

What we would like to have:

- Higher-Order models.
- A merge between the two : an endo-transformation handling a rich type theory as well as forward or reverse differential transformation.
- A lighter use of shifts.

More on Dialectica

Monadic laws

$$\{t\} \gg= f \equiv f\ t \qquad t \gg= (\lambda x.\, \{x\}) \equiv t$$

$$(t \gg= f) \gg= g \equiv t \gg= (\lambda x.\, f\ x \gg= g)$$

Monoidal laws

$$t \circledast u \equiv u \circledast t \qquad \varnothing \circledast t \equiv t \circledast \varnothing \equiv t$$

$$(t \circledast u) \circledast v \equiv t \circledast (u \circledast v)$$

Distributivity laws

$$\varnothing \gg= f \equiv \varnothing \qquad t \gg= \lambda x.\, \varnothing \equiv \varnothing$$

$$(t \circledast u) \gg= f \equiv (t \gg= f) \circledast (u \gg= f)$$

$$t \gg= \lambda x.\, (f\ x \circledast g\ x) \equiv (t \gg= f) \circledast (t \gg= g)$$