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 ".

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."


[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.”.


”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."


"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**

<table>
<thead>
<tr>
<th>Programs</th>
<th>Logic</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>Proof</td>
<td>Morphisms</td>
</tr>
<tr>
<td>$\lambda x^A.t^B$</td>
<td>$A \vdash B$</td>
<td>$f : A \to B$</td>
</tr>
<tr>
<td>Type</td>
<td>Formulas</td>
<td>Objects</td>
</tr>
<tr>
<td>Execution</td>
<td>Cut - elimination</td>
<td>Equality</td>
</tr>
</tbody>
</table>

*In a future far far away: type theory allows to reason on basic computer algebra algorithms*
Syntactical models (Pédrot)

Programs

Term \( \lambda x^A.t^B \)

Type

Logic

Proof

\[ : \]

\[ A \vdash B \]

Formulas

Cut - elimination

Equivalence

Dialectica

Other Proofs

\[ : \]

\[ A \vdash B \]

Other Formulas

In a future far far away: type theory allows to reason on basic computer algebra algorithms
Smooth models (K.)

Programs

Term

Logic

Proof

Type

Execution

Analysis

Smooth maps

Spaces

Formulas

Cut - elimination

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

- Automatic Differentiation.
- Linear Logic.
- Differential $\lambda$-calculus.
Automatic Differentiation

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

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

$$
\begin{align*}
x_1 &= x_0^2 \\
x_2 &= \cos(x_0) \\
z &= y + x_2
\end{align*}
$$

$$
\begin{align*}
x_1' &= 2x_0x_0' \\
x_2' &= -x_0's\sin(x_0) \\
z' &= y' + 2x_2x_2'
\end{align*}
$$

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

**Forward Mode differentiation:** $(x_1, x_1') \rightarrow (x_2, x_2') \rightarrow (z, z')$.

**Reverse Mode differentiation:** $x_1 \rightarrow x_2 \rightarrow z \rightarrow z' \rightarrow x_2' \rightarrow x_1'$

while keeping formal the unknown derivative.
AD from a higher-order functional point of view

\[ D_u(f \circ g)(v) = D_{g(u)}f(D_u f(v)) \]
\[ D_u(f \circ g) = D_{g(u)}f \circ D_u(f) \]

- **Forward Mode differentiation:**
  \[ g(u) \rightarrow D_u g \rightarrow f(g(u)) \rightarrow D_{g(u)}f \rightarrow D_{g(u)}f \circ D_u(f) \]

- **Reverse Mode differentiation:**
  \[ g(u) \rightarrow f(g(u)) \rightarrow D_{g(u)}f \rightarrow D_u g \rightarrow D_{g(u)}f \circ D_u(f) \]

The choice of an algorithm is due to complexity considerations:

- Forward mode for \( f : \mathbb{R} \rightarrow \mathbb{R}^n \).
- Reverse mode for \( f : \mathbb{R}^n \rightarrow \mathbb{R} \)
Linear logic

Usual Implication

A call-by-name translation

\[ A \Rightarrow B = \exists ! A \land B \]
\[ C^\infty(A, B) \sim \mathcal{L}(!A, B) \]

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

A call-by-name translation

Usual implication

\[ A \Rightarrow B = ! A \multimap B \]

Linear Implication

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

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

A call-by-name translation

\[ A \Rightarrow B = ! A \multimap B \]
\[ C^\infty(A, B) \simeq L(!A, B) \]

Usual implication

Linear implication

Exponential

Smooth Semantics

A proof is linear when it uses only once its hypothesis A.
Differential $\lambda$-calculus [Ehrhard Regnier. 2004]

Inspired by denotational models of Linear Logic in vector spaces of sequences, it introduces a differentiation of $\lambda$-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:

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

Operational Semantics:

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

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

\[
\frac{\partial y}{\partial x} \cdot T = \begin{cases} T & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \quad \frac{\partial 0}{\partial x} \cdot T = 0
\]

\[
\frac{\partial}{\partial x}(\lambda y.s) \cdot T = \lambda y.\frac{\partial s}{\partial x} \cdot T \quad \frac{\partial}{\partial x}(s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T
\]

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

\[
\frac{\partial y}{\partial x} \cdot T = \begin{cases} 
T & \text{if } x = y \\
0 & \text{otherwise}
\end{cases} \quad \frac{\partial 0}{\partial x} \cdot T = 0
\]

\[
\frac{\partial}{\partial x} (\lambda y.s) \cdot T = \lambda y. \frac{\partial s}{\partial x} \cdot T \quad \frac{\partial}{\partial x} (s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T
\]

Differentiating composition:

\[
\frac{\partial}{\partial x} (sv) \cdot v = (\frac{\partial s}{\partial x} \cdot T)u + ...
\]

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

\[ \frac{\partial y}{\partial x} \cdot T = \begin{cases} T & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \quad \frac{\partial 0}{\partial x} \cdot T = 0 \]

\[ \frac{\partial}{\partial x} (\lambda y.s) \cdot T = \lambda y.\frac{\partial s}{\partial x} \cdot T \quad \frac{\partial}{\partial x} (s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T \]

Differentiating composition:

\[ \frac{\partial}{\partial x} (su) \cdot v = (\frac{\partial s}{\partial x} \cdot T)u + (Ds \cdot (\frac{\partial u}{\partial x} \cdot v)u) \]

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

\[
\frac{\partial y}{\partial x} \cdot T = \begin{cases} T & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \quad \frac{\partial 0}{\partial x} \cdot T = 0
\]

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

Differentiating composition:

\[
\frac{\partial}{\partial x} (su) \cdot v = (\frac{\partial s}{\partial x} \cdot T)u + \underbrace{(Ds \cdot \left(\frac{\partial u}{\partial x} \cdot v\right))}_{\text{Linear substitution}} \quad u)
\]
Dialectica : Gödel doing Deep Learning
A Dialectica Transformation

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

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

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

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

\[ A \leadsto \exists u : \forall x : C(A), A^D[u, x] \]

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.

\[ A \leadsto \exists u : W(A), \forall x : C(A), A^D[u, x] \]

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

[Pedrot, CLS-LICS2014]

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

\( \leadsto \) Dialectica as a program translation ... whose abstract multiset is not smooth enough
Pédrot Dialectica Transformation

At the source: $\lambda$-calculus typed with minimal logic.
At the target: $\lambda$-calculus with pairs and an $M$ operation.

$W(\alpha) := \alpha_W$
$C(\alpha) := \alpha_C$

$W(A \Rightarrow B) := (W(A) \Rightarrow W(B)) \times (W(A) \Rightarrow C(B) \Rightarrow M C(A))$
$C(A \Rightarrow B) := W(A) \times C(B)$

$x_x := \lambda \pi. \{\pi\} \quad x^*: := x$

$xy := \lambda \pi. \emptyset$ if $x \neq y \quad (\lambda x. t)^* := (\lambda x. t^*, \lambda x. \pi. t_x \pi)$

$(\lambda x. t)_y := \lambda \pi. (\lambda x. t_y) \pi.1 \pi.2 \quad (t \ u)^* := (t^*.1) \ u^*$

$(t \ u)_y := \lambda \pi. (t_y (u^*, \pi)) \succeq ((t^*.2) u^* \pi \gg u_y)$
Pédrot Dialectica Transformation

At the source: $\lambda$-calculus typed with minimal logic.
At the target: $\lambda$-calculus with pairs and an $M$ operation.

Soundness [Ped14]

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

- $W(\Gamma) \vdash t^\bullet : W(A)$
- $W(\Gamma) \vdash t_x : C(A) \Rightarrow M\ C(X)$ provided $x : X \in \Gamma$. 
Tracking differentiation in Dialectica

\[
\begin{align*}
\Gamma &\vdash \emptyset : \mathcal{M} A \\
\Gamma &\vdash m_1 : \mathcal{M} A \\
\Gamma &\vdash m_2 : \mathcal{M} A \\
\Gamma &\vdash m_1 \star m_2 : \mathcal{M} A \\
\Gamma &\vdash t : A \\
\Gamma &\vdash \{t\} : \mathcal{M} A \\
\Gamma &\vdash m : \mathcal{M} A \\
\Gamma &\vdash f : A \Rightarrow \mathcal{M} B \\
\Gamma &\vdash m \gg= f : \mathcal{M} B
\end{align*}
\]

\[
\begin{align*}
x_x &:= \lambda \pi. \{\pi\} \\
x_\bullet &:= x \\
\lambda x.y &:= \lambda \pi. \emptyset \quad \text{if } x \neq y \\
(\lambda x.t)_y &:= \lambda \pi. (\lambda x.t_y) \pi.1 \pi.2 \\
(t u)_\bullet &:= (t\cdot1) u\cdot \\
(t u)_y &:= \lambda \pi. (t_y (u\cdot, \pi)) \star ((t\cdot2) u\cdot \pi \gg= u_y)
\end{align*}
\]
Tracking differentiation in Dialectica

\[
\frac{\Gamma \vdash \emptyset : \mathcal{M}A}{\Gamma \vdash t : A} \quad \frac{\Gamma \vdash m : \mathcal{M}A \quad \Gamma \vdash f : A \Rightarrow \mathcal{M}B}{\Gamma \vdash m \gg f : \mathcal{M}B}
\]

Differential \(\lambda\)-calculus

\[
x_x \quad := \quad \lambda \pi. \{\pi\}
\]

\[
x^* \quad := \quad x
\]

\[
x_y \quad := \quad \lambda \pi. \emptyset \quad \text{if } x \neq y
\]

\[
(\lambda x. t)_y \quad := \quad \lambda \pi. (\lambda x. t) \pi.1 \pi.2
\]

\[
(t u)^* \quad := \quad (t^*.1) u^*
\]

\[
(t u)_y := \lambda \pi. (t_y (u^*, \pi)) \odot ((t^*.2) u^* \pi \gg u_y)
\]
Tracking differentiation in Dialectica

\[
\begin{align*}
\Gamma \vdash m_1 : \mathcal{M} A & \quad \Gamma \vdash m_2 : \mathcal{M} A \\
\hline
\Gamma \vdash \emptyset : \mathcal{M} A & \quad \Gamma \vdash m_1 \otimes m_2 : \mathcal{M} A \\
\hline
\Gamma \vdash t : A & \quad \Gamma \vdash f : A \Rightarrow \mathcal{M} B \\
\hline
\Gamma \vdash \{t\} : \mathcal{M} A & \quad \Gamma \vdash m \gg= f : \mathcal{M} B
\end{align*}
\]

Differential \(\lambda\)-calculus

\[
x x := \lambda \pi. \frac{\partial x}{\partial x} \cdot \pi \\
x y := \lambda \pi. \frac{\partial x}{\partial y} \cdot \pi \quad \text{if } x \neq y \\
(\lambda x. t)_y := \lambda \pi. (\lambda x. t y) \pi.1 \pi.2 \\
(t u)^\bullet := (t^\bullet.1) u^\bullet
\]

\[
(t u)_y := \lambda \pi. (t y (u^\bullet, \pi)) \otimes ((t^\bullet.2) u^\bullet \pi \gg= u_y)
\]
Tracking differentiation in Dialectica

\[
\begin{align*}
\Gamma & \vdash \emptyset : \mathcal{M} A \\
\Gamma & \vdash m_1 : \mathcal{M} A \\
\Gamma & \vdash m_2 : \mathcal{M} A \\
\Gamma & \vdash m_1 \odot m_2 : \mathcal{M} A \\
\Gamma & \vdash t : A \\
\Gamma & \vdash m : \mathcal{M} A \\
\Gamma & \vdash f : A \Rightarrow \mathcal{M} B \\
\Gamma & \vdash m \ggg f : \mathcal{M} B
\end{align*}
\]

\text{Differential } \lambda\text{-calculus}

\[
\begin{align*}
x_{x} & := \lambda \pi. \frac{\partial x}{\partial x} \cdot \pi \\
x_{y} & := \lambda \pi. \frac{\partial x}{\partial y} \cdot \pi \quad \text{if } x \neq y \\
(\lambda x.t)_{y} & := \lambda \pi. (\lambda x.t)_{y} \pi.1 \pi.2 \\
(t \ u)_{y} & := \lambda \pi. (t \ y \ (u^{*}, \pi)) \odot ((t^{*}.2) \ u^{*} \pi \ggg u_{y})
\end{align*}
\]

Backpropagation
Differential λ-calculus in a hurry

\[ \text{D}(\lambda x.s) \cdot t \rightarrow_{\beta_D} \lambda x. \frac{\partial s}{\partial x} \cdot t \]

\[
\frac{\partial y}{\partial x} \cdot T = \begin{cases} 
T & \text{if } x = y \\
0 & \text{otherwise}
\end{cases} \\
\frac{\partial}{\partial x} (sU) \cdot T = (\frac{\partial s}{\partial x} \cdot T)U + (Ds \cdot (\frac{\partial U}{\partial x} \cdot T))U
\]

\[
\frac{\partial}{\partial x} (\lambda y.s) \cdot T = \lambda y. \frac{\partial s}{\partial x} \cdot T \\
\frac{\partial}{\partial x} (Ds \cdot u) \cdot T = D(\frac{\partial s}{\partial x} \cdot T) \cdot u + Ds \cdot (\frac{\partial u}{\partial x} \cdot T)
\]

\[
\frac{\partial 0}{\partial x} \cdot T = 0 \\
\frac{\partial}{\partial x} (s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T
\]
Dialectica is Differentiation

The linearized Dialectica Translation weakens to a transformation from $\lambda$-calculus to Differential $\lambda$-calculus.

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

$$\begin{align*}
[\mathcal{M} A] &= A \\
[\emptyset] &= 0 \\
[t \ast u] &= [t] + [u] \\
\{\{t\}\} &= [t].
\end{align*}$$

**Proposition**

Consider two $\lambda$-terms $t$ and $u$. Then $[t_x]u \equiv \frac{\partial t}{\partial x} \cdot u$ and $((\lambda x.t)^* . 2))u \equiv Dt \cdot u$. 
Dialectica enriched with real functions

We now enrich both our source and target $\lambda$-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 $\varphi', \psi', \ldots$.

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

$$\varphi^* := (\varphi, \lambda \alpha \pi. \{() \mapsto \varphi'(\alpha)\}) \quad \varphi_x := \lambda \pi. \emptyset$$

The soundness theorem is then adapted trivially.

**Soundness Theorem**

The following equation holds in the target.

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

\[ A, B := \alpha \mid A \Rightarrow B \mid A \times B \mid A^\perp \mid \text{Tr}(A) \]
\[ t, u := x \mid (t)u \mid \lambda x.t \mid (t, u) \mid t \mid u \otimes v \mid \emptyset. \]

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

\[
\frac{\Gamma \vdash t : A}{\Gamma \vdash \{t\} : \text{Tr}(A)} \quad \frac{\Gamma \vdash t : \text{Tr}(A) \quad \Gamma \vdash u : \text{Tr}(A)}{\Gamma \vdash t \otimes u : \text{Tr}(A)}
\]
\[
\frac{\Gamma \vdash \emptyset : \text{Tr}(A)}{\Gamma \vdash t : \text{Tr}(A) \quad \Gamma \vdash f : A \Rightarrow \text{Tr}(B)}{\Gamma \vdash t \gg= u : \text{Tr}(B)}
\]
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.

\[ A, B := \alpha \mid A \Rightarrow B \mid A \times B \mid A^\perp \mid \text{Tr}(A) \]

\[ t, u := x \mid (t)u \mid \lambda x.t \mid (t, u) \mid t \mid u \otimes v \mid \emptyset. \]

Two mutually inductively defined translations:

\[
\begin{align*}
xx & := \lambda \pi. \{\pi\} & \quad x^\bullet & := x \\
x_y & := \lambda \pi. \emptyset \text{ if } x \neq y & (\lambda x.t)^\bullet & := (\lambda x.t^\bullet, \lambda x\pi.t_x \pi) \\
(\lambda x.t)_y & := \lambda \pi. (\lambda x.t_y) \pi.1 \pi.2 & (t,u)^\bullet & := (t^\bullet.1) u^\bullet \\
(t,u)_y & := \lambda \pi. (t_y(u^\bullet,\pi)) \otimes ((t^\bullet.2)u^\bullet \pi \ggg u_y)
\end{align*}
\]
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.

\[
A, B := \alpha | A \Rightarrow B | A \times B | A^\perp | \text{Tr}(A)
\]
\[t, u := x | (t)u | \lambda x.t | (t, u) | t | u \otimes v | \emptyset.\]

Two mutually inductively defined translations:

\[
\begin{align*}
x_x & := \lambda \pi. \{ \pi \} & x^\bullet & := x \\
x_y & := \lambda \pi. \emptyset \text{ if } x \neq y & (\lambda x. t)^\bullet & := (\lambda x. t^\bullet, D(\lambda x.t)) \\
(\lambda x. t)_y & := \lambda \pi. (\lambda x. t_y) \pi.1 \pi.2 & (t u)^\bullet & := (t^\bullet.1) u^\bullet \\
\end{align*}
\]

\[
(t u)_y := \lambda \pi. (t_y (u^\bullet, \pi)) \otimes ((Dt) u^\bullet \pi \gg u_y)
\]
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.

\[
\begin{align*}
A, B &:= \alpha \mid A \Rightarrow B \mid A \times B \mid A^\perp \mid \text{Tr}(A) \\
t, u &:= x \mid (t)u \mid \lambda x. t \mid (t, u) \mid t \mid u \odot v \mid \emptyset.
\end{align*}
\]

Two typed differential transformations

When \( \Gamma \vdash t : !A \rightarrow B \) and writing \( Dt = (t \cdot 2) \) we have:

\[
\begin{align*}
\Gamma &\vdash Dt : A \Rightarrow (B^\perp \Rightarrow \text{Tr}(A^\perp)) \\
\Gamma &\vdash t_y : A \times B^\perp \Rightarrow \text{Tr}(Y^\perp)
\end{align*}
\]
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 \( f : \mathbb{R}^n \to \mathbb{R}^m. \)

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

\[
\overleftarrow{D}(f) : \begin{cases} \mathbb{R}^n \times \mathbb{R}^m \perp \to \mathbb{R}^m \times \mathbb{R}^n \perp \\ (a, x) \mapsto (f(a), (v \mapsto v \cdot (D_a f \cdot x))) \end{cases}
\]

- As in differential \( \lambda \)-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 $f : \mathbb{R}^n \to \mathbb{R}^m$. 

$$\overrightarrow{D}(f) : \left\{ \mathbb{R}^n \times \mathbb{R}^m x \to \mathbb{R}^n \times \mathbb{R}^m \right\} (a, x) \mapsto (f(a), D_a f \cdot x)$$

Consider $f : E \to F$. 

$$\overleftarrow{D}(f) : \left\{ E \times F' \to F \times E' \right\} (a, \ell) \mapsto (f(a), (v \in F \mapsto (v \cdot (D_a f \cdot x))))$$

▶ As in differential $\lambda$-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.

⇝ towards a finer, internal handling of automatic differentiation as a reduction strategy.
Automatic Differentiation
as a choice of reduction strategy

Refining $\lambda$-calculus with operations from distribution theory.
Just a glimpse at Differential Linear Logic

\[
A, B := A \otimes B \mid A \otimes B \mid A \oplus B \mid 0 \mid A \times B \mid \top \mid !A \mid !A
\]

Exponential rules of DiLL\(_0\)

\[
\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A} \quad c
\quad \frac{\vdash \Gamma}{\vdash \Gamma, ?A} \quad w
\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A} \quad d
\]

\[
\frac{\vdash \Gamma, !A, \vdash \Delta, !A}{\vdash \Gamma, \Delta, !A} \quad \bar{c}
\quad \frac{\vdash \vdash !A}{\vdash !A} \quad \bar{w}
\frac{\vdash \Gamma, A}{\vdash \Gamma, !A} \quad \bar{d}
\]

\[\vdash ?\Gamma, A \quad p\]

\[\vdash ?\Gamma, !A\]

\[\Rightarrow\quad \text{A particular point of view on differentiation induced by duality.}\]

- Differential interaction nets, Ehrhard and Regnier, TCS (2006)
Exponentials are distributions

\[ (?A) := C^\infty([A]', \mathbb{R})' \quad \text{functions} \quad \] 
\[ (!A) := C^\infty([A], \mathbb{R})' \quad \text{distributions} \]

A typical distribution is the dirac operator:

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

Exponential rules of DiLL\(_0\)

\[ \frac{\vdash \Gamma, f : ?A, g : ?A}{\vdash \Gamma, f.g : ?A} \quad c \quad \frac{\vdash \Gamma}{\vdash \Gamma, \text{cst}_0 : ?A} \quad w \quad \frac{\vdash \Gamma, \ell : A}{\vdash \Gamma, \ell : ?A} \quad d \]
\[ \frac{\vdash \Gamma, \phi : !A, \quad \vdash \Delta, \psi : !A}{\vdash \Gamma, \Delta, \phi \ast \psi : !A} \quad \bar{c} \quad \frac{\vdash \delta_0 : !A}{\vdash \bar{\delta}_0} \quad \bar{w} \quad \frac{\vdash \Gamma, v : A}{\vdash \Gamma, D_0(-)(v) : !A} \quad \bar{d} \]
\[ \frac{\vdash ?\Gamma, v : A}{\vdash ?\Gamma, \delta_v : !A} \quad p \]
A few operations typed by DíLL

The composition of linear functions:

\[
\Gamma \vdash f : A \to B \quad \Delta \vdash g : B \to C \\
\Gamma, \Delta \vdash g \circ f : A \to C \\
\text{cut}
\]

The composition of non-linear functions:

\[
\Gamma \vdash f : !A \to B \\
\Delta \vdash (x \mapsto \delta_{g(x)}) : !A \to !B \\
\Gamma, \Delta \vdash g : !B \to C \\
\Gamma, \Delta \vdash g \circ f = (x \mapsto \delta_{f(x)}g) : !A \to C \\
\text{cut}
\]

The Differentiation of non-linear functions:

\[
\Gamma \vdash f : !A \to B \\
\vdash \Delta, v : A \\
\vdash \Delta, \Gamma, D_0(\_)(v) : !A \\
\Gamma, \Delta \vdash D_0(f)(v) : B \\
\text{cut}
\]

Let’s translate this into a term language typed by DíLL.
A few operations typed by DiLL

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

The composition of non-linear functions:

$$
\Gamma \vdash f : !A \to B
$$
$$
\Delta \vdash (x \mapsto \delta g(x)) : !A \to !B
$$
$$
\Delta \vdash g : !B \to C
$$

$$
\Gamma, \Delta \vdash g \circ f = (x \mapsto \delta f(x)g) : !A \to C
$$

The Differentiation of non-linear functions:

$$
\vdash \Delta, v : A
$$
$$
\vdash \Gamma, D_0(\_)(v) : !A
$$

$$
\Gamma, \Delta \vdash D_0(f)(v) : B
$$

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 $\delta_x$ with differential arguments $D_u t$.

The Chain rule:

$$\Gamma \vdash f : !A \to B$$
$$\Delta \vdash (x \mapsto \delta_{g(x)}) : !A \to !B$$
$$\Delta \vdash g : !B \to C$$
$$\Gamma, \Delta \vdash g \circ \delta_f : !A \to C$$
$$\Gamma, \Delta, \Delta' \vdash D_0(g \circ f)(v) : c$$

Let’s translate this into a term language typed by DiLL.
A minimal language allowing to express automatic differentiation

Two class of terms:

\[ u, v := x \mid t \mid u \ast v \mid \emptyset \mid u \otimes v \mid 1 \mid \delta u \mid D_u(t) \mid \downarrow t \]
\[ t, s := u \perp \mid t \cdot s \mid w_1 : N \mid \lambda x.t \mid dx.t \mid \uparrow u \]

A function \( \lambda x.t \) can be matched to two kinds of arguments: diracs \( \delta u \) or differential operators \( D_u t \).

\[
(\lambda x.t)\delta u \rightarrow t[u/x] \\
(\lambda x.t)D_u t \rightarrow \cdots
\]

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

\[
(\lambda x.(t)u)D_w s \rightarrow \uparrow(\downarrow((\lambda x.t)D_w s)u \ast \downarrow(t((\lambda x.u)D_w s)))
\]

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

\[
(\lambda x.\uparrow\delta t)D_u s \rightarrow (\lambda z.\uparrow(D_z((\lambda x.t)D_u s))((\lambda x.t)(u)))
\]

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:

\[
(\lambda x.((\lambda y.s)\delta_t))D_u r \rightarrow \uparrow(\downarrow((\lambda x.(\lambda y.s))D_u r)\delta_t \ast \downarrow((\lambda y.s)((\lambda x.\delta_t)D_u r)))
\]

\[
\rightarrow^* (\lambda y.s)((\lambda x.\delta_t)D_u r) \text{ by involutivity of the shifts}
\]

\[
\rightarrow (\lambda y.s)(\lambda z.\uparrow(D_z((\lambda x.t)D_u r)))((\lambda x.t)(u)))
\]

\[
\rightarrow ((\lambda y.s)(\lambda z.\uparrow(D_z((\lambda x.t)D_u r))))((t[w/x]))\text{ as } u = \delta_w
\]

\[
\rightarrow^* (\lambda y.s)D_v((\lambda x.t)D_u r) \text{ if } (t[w/x] \rightarrow^* \delta_v)
\]

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).

\[ \phi \ast \psi := f \mapsto \phi(x \mapsto \psi(y \mapsto f(x + y))) \]
\[ \delta_u \ast \delta_v \rightarrow \delta_{u \ast v} \]

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

\[ f \cdot g := x \mapsto f(x) \cdot g(x) \]
\[ (\lambda y.t) \cdot (\lambda z.s) \rightarrow \lambda x.(t[x/y]) \cdot (s[x/z]) \]
Distinguishing Linear and Non-Linear Maps

\[
\begin{align*}
\vdash N, t : M, x \perp : (!P) \perp | \\
\vdash N, \lambda x.t : (!P) \perp \not\in M | \\
\vdash N, x \perp : P \perp, t : M \\
\vdash N, dx.t : (!P) \perp \not\in M 
\end{align*}
\]
Interpreting Dialectica in DiLL

\[ [\mathcal{M} A] := !![A] \]
\[ [\lambda x. t] := \lambda x.[t] \]
\[ [\emptyset] := \uparrow \emptyset \]
\[ [u \otimes v] := \uparrow(\downarrow[u] \star \downarrow[v]) \]

\[ [x] := x \]
\[ [(t, u)] := ([t], [u]) \]
\[ [{t}] := \uparrow(\delta_{\delta_{[t]}}) \]
\[ [m \gg f] := (dx.[f]x)[m] \]

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

\[ \{t\} \triangleright\!\!\!\!\!\!\!\!\!\triangleright f \equiv f \ t \quad t \triangleright\!\!\!\!\!\!\!\!\!\triangleright (\lambda x. \{x\}) \equiv t \]

\[ (t \triangleright\!\!\!\!\!\!\!\!\!\triangleright f) \triangleright\!\!\!\!\!\!\!\!\!\triangleright g \equiv t \triangleright\!\!\!\!\!\!\!\!\!\triangleright (\lambda x. f \ x \triangleright\!\!\!\!\!\!\!\!\!\triangleright g) \]

Monoidal laws

\[ t \odot u \equiv u \odot t \quad \emptyset \odot t \equiv t \odot \emptyset \equiv t \]

\[ (t \odot u) \odot v \equiv t \odot (u \odot v) \]

Distributivity laws

\[ \emptyset \triangleright\!\!\!\!\!\!\!\!\!\triangleright f \equiv \emptyset \quad t \triangleright\!\!\!\!\!\!\!\!\!\triangleright (\lambda x. \emptyset) \equiv \emptyset \]

\[ (t \odot u) \triangleright\!\!\!\!\!\!\!\!\!\triangleright f \equiv (t \triangleright\!\!\!\!\!\!\!\!\!\triangleright f) \odot (u \triangleright\!\!\!\!\!\!\!\!\!\triangleright f) \]

\[ t \triangleright\!\!\!\!\!\!\!\!\!\triangleright \lambda x. (f \ x \odot g \ x) \equiv (t \triangleright\!\!\!\!\!\!\!\!\!\triangleright f) \odot (t \triangleright\!\!\!\!\!\!\!\!\!\triangleright g) \]