An Introduction to Differentiable Programming

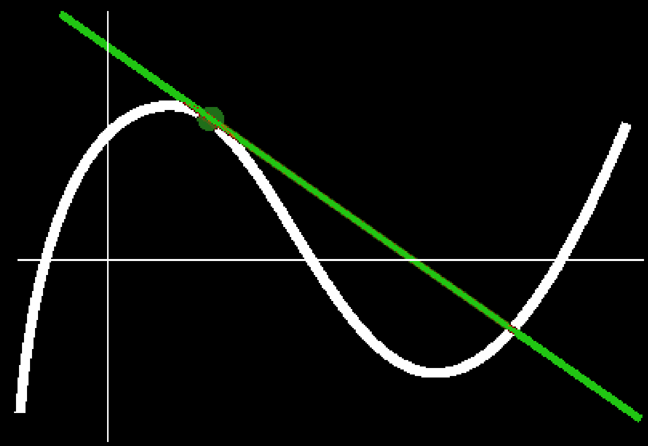
Damiano Mazza
CNRS, LIPN, Université Sorbonne Paris Nord

FoPSS, Bertinoro, 13–14 February 2023

Remember derivatives?

$$f : \mathbb{R} \to \mathbb{R}$$

$$f'(x) := \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



Thought of as an operator $(-)' : (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R}$

$$(\alpha f + \beta g)'(x) = \alpha f'(x) + \beta g'(x)$$

$$(fg)'(x) = f(x)g'(x) + g(x)f'(x)$$

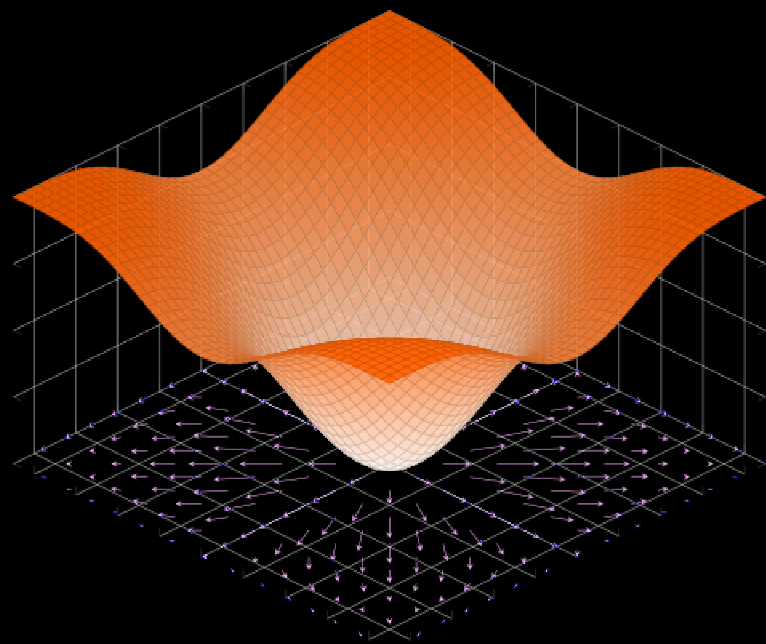
$$(f \circ g)'(x) = f'(g(x))g'(x)$$

Remember gradients?

$$f : \mathbb{R}^n \to \mathbb{R}$$

Gradient:

$$\nabla f : \mathbb{R}^n \to \mathbb{R}^n$$



$$\partial_i f(x_1, \ldots, x_n) := \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_n) - f(x_1, \ldots, x_n)}{h}$$

$$\nabla f = (\partial_1 f, \ldots, \partial_n f)$$

Why gradients?

Gradient descent!

Target function $f : \mathbb{R}^m \to \mathbb{R}^p$, training set $X \subseteq_{\text{fin}} \mathbb{R}^m$.

Parametric approximation $g : \mathbb{R}^{n+m} \to \mathbb{R}^p$.

Want to find $\vec{w}_{\text{opt}} \in \mathbb{R}^n$ such that $g(\vec{w}_{\text{opt}}) \approx f$.

Define

$$e(\vec{w}) := \frac{1}{|X|} \sum_{\vec{x} \in X} \|g(\vec{w}, \vec{x}) - f(\vec{x})\|^2 : \mathbb{R}^n \to \mathbb{R}$$

For initial $\vec{w}_0 \in \mathbb{R}^n$, step $\eta < 0$, and $i \in \mathbb{N}$, set

$$\vec{w}_{i+1} := \vec{w}_i + \eta \nabla e(\vec{w}_i)$$

When $e(\vec{w}_i)$ is close to zero, set $\vec{w}_{\text{opt}} := \vec{w}_i$.

Guaranteed to happen under convexity assumptions on e.

Automatic Differentiation (AD)

- In machine learning (ML), g is computed by a **neural network** (NN).
- In their simplest form, these are layers of **neurons**

$$\theta(x_1, \ldots, x_k) := \sigma\left(\sum_{i=1}^{k} w_i x_i\right) : \mathbb{R}^k \to \mathbb{R}$$

  where σ is some **activation function**.

- $n =$ number of weights in the net. (These days it can easily be 10^8 or 10^9).

AD = methods for automatically computing gradients of functions specified by a computer program (e.g. the loss function of a neural network).

Differentiable Programming

- In recent years (from 2015-2016), NN architectures used in ML started becoming more and more complex.

- I.e., g is computed by more and more sophisticated programs.

- Need programming languages with a built-in engine for efficiently computing derivatives/gradients/Jacobians.

differentiable programming = programming languages + AD

Two wrong ideas

- Approximate the definition:

$$f'_{\sim}(x) := \frac{f(x+h) - f(x)}{h}$$

  with h very small.

  - How do we choose h?
  - Contains two “deadly sins” of numerical computation.

- Symbolic computation (like in Mathematica).

  - Good idea, but needs to be extended to programs.
  - Inefficient, needs sharing.

Dual numbers

- The commutative ring of dual numbers is defined as

$$\widehat{\mathbb{R}} := \mathbb{R}[\varepsilon]/\langle \varepsilon^2 \rangle$$

- Its elements are pairs $(a, a') \in \mathbb{R}^2$, with

$$\begin{aligned} 0 &= (0, 0) \\ (a, a') + (b, b') &= (a + b, a' + b') \\ 1 &= (1, 0) \\ (a, a')(b, b') &= (ab, a'b + ab') \end{aligned}$$

Dual numbers and derivatives

- If $f : \mathbb{R} \to \mathbb{R}$ is differentiable, we extend it to $\widehat{f} : \widehat{\mathbb{R}} \to \widehat{\mathbb{R}}$ as follows:

$$\widehat{f}(a, a') := (f(a), f'(a)a')$$

- Using the chain rule, we have

$$\begin{aligned}\widehat{f \circ g}(a, a') &= (f(g(a)), a'(f \circ g)'(a)) = (f(g(a)), f'(g(a))g'(a)a') \\ &= \widehat{f}(g(a), g'(a)a') = \widehat{f}(\widehat{g}(a, a')) \\ \widehat{\mathrm{id}}(a, a') &= (a, \mathrm{id}'(a)a') = (a, 1 \cdot a') = (a, a')\end{aligned}$$

- Furthermore, notice that

$$\widehat{f}(a, 1) = (f(a), f'(a))$$

Dual numbers and AD for unary straight-line programs

- The above suggests the following program transformation:

```
def f(x):
    z1 = f1(x)
    z2 = f2(z1)
    ...
    return g(zn)
```

$\rightsquigarrow$

```
def df(x):
    dx  = 1
    z1  = f1(x)
    dz1 = dx * df1(x)
    z2  = f2(z1)
    dz2 = dz1 * df2(z1)
    ...
    return dzn * dg(zn)
```

- Exact computation.
- Preserves the complexity (modulo a factor of 3).

Dual numbers and partial derivatives

- If $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, define $\widehat{f} : \widehat{\mathbb{R}}^n \to \widehat{\mathbb{R}}$ as

$$\widehat{f}(a_1, a'_1, \ldots, a_n, a'_n) := (f(\vec{a}), \nabla f(\vec{a}) \cdot \vec{a}')$$
$$= \left(f(a_1, \ldots, a_n), \sum_{i=1}^{n} a'_i \, \partial_i f(a_1, \ldots, a_n) \right)$$

- We still have

$$\widehat{f \circ (g_1, \ldots, g_n)} = \widehat{f} \circ (\widehat{g_1}, \ldots, \widehat{g_n})$$

- Furthermore, we have

$$\widehat{f}(a_1, 0, \ldots, a_i, 1, \ldots, a_n, 0) = (f(\vec{a}), \partial_i f(\vec{a}))$$

Dual numbers and AD for straight-line programs

The above transformation generalizes to

```
def f(x1,...,xn):
    ...
    z = g(w1,...,wk)
    ...
    return y
```

$\rightsquigarrow$

```
def df(i,x1,...,xn):
    dx1,...,dxi,...,dxn = 0,...,1,...0
    ...
    z = g(w1,...,wk)
    dz = dw1 * dg(1,w1,...,wk) + ... + dwk * dg(k,w1,...,wk)
    ...
    return dy
```

- Still exact computation, still complexity-preserving.
- Covers the case of loss functions of NNs.
- However, inefficient for gradients: requires n passes.

Dual numbers and AD for arbitrary programs

```
{z = g(w1,...,wk)}    :=    z = g(w1,...,wk)
                            dz = dw1 * dg(1,w1,...,wk) + ...\
                               + dwk * dg(k,w1,...,wk)
```

```
{ if x <= 0:          }        if x <= 0:
{     <code1>         }   :=       {<code1>}
{ else:               }        else:
{     <code2>         }            {<code2>}
```

```
{ while x <= 0: }   :=   while x <= 0:
{     <code>    }            {<code>}
```

```
                                def f(x1,...,xn):
                                    <code>
                                    return y
def f(x1,...,xn):
    <code>              ~>      def df(i,x1,...,xn):
    return y                        dx1,...,dxi,...,dxn = 0,...,1,...,0
                                    {<code>}
                                    return dy
```

Theorem (Joss 1976). *Taking a set of basic arithmetic functions as primitive,* $[\![\mathtt{df}(i)]\!] = \partial_i [\![\mathtt{f}]\!]$ *almost everywhere.*

PCF with real numbers

$$
\begin{array}{rrcl}
\text{Types:} & A, B & ::= & \mathsf{R} \mid A \times B \mid A \to B \\
\text{Programs:} & M, N, P & ::= & x \mid \lambda f(x).M \mid MN \mid \langle M, N\rangle \mid \pi_i M \\
 & & \mid & \mathsf{if}(P \leq 0; M, N) \mid \underline{r} : \mathsf{R} \mid \mathsf{f} : \mathsf{R}^k \to \mathsf{R} \\
\text{Evaluation:} & (\lambda f(x).M)N & \to & M[N/x][\lambda f(x).M/f] \\
 & \pi_i \langle M_1, M_2\rangle & \to & M_i \\
 & \mathsf{if}(\underline{r} \leq 0; M, N) & \to & \begin{cases} M & \text{if } r \leq 0 \\ N & \text{if } r > 0 \end{cases} \\
 & \mathsf{f}\langle \underline{r_1}, \ldots, \underline{r_k}\rangle & \to & \underline{[\![\mathsf{f}]\!](r_1, \ldots, r_n)}
\end{array}
$$

Every program $x_1 : \mathsf{R}, \ldots, x_n : \mathsf{R} \vdash M : \mathsf{R}$ has a natural semantics

$$[\![M]\!] : \mathbb{R}^n \rightharpoonup \mathbb{R}.$$

We write $\mathrm{d}(M)$ for the open subset of $\mathbb{R}^n$ where $[\![M]\!]$ is differentiable.

AD in PCF

$$\overrightarrow{\mathbf{D}}(\mathsf{R}) := \mathsf{R} \times \mathsf{R} \qquad \overrightarrow{\mathbf{D}}(A \times B) := \overrightarrow{\mathbf{D}}(A) \times \overrightarrow{\mathbf{D}}(B) \qquad \overrightarrow{\mathbf{D}}(A \to B) := \overrightarrow{\mathbf{D}}(A) \to \overrightarrow{\mathbf{D}}(B)$$

$$\overrightarrow{\mathbf{D}}(x : A) := x : \overrightarrow{\mathbf{D}}(A) \qquad \overrightarrow{\mathbf{D}}(\underline{r}) := \langle \underline{r}, 0 \rangle$$

$$\overrightarrow{\mathbf{D}}(\lambda f(x).M) := \lambda f(x).\overrightarrow{\mathbf{D}}(M) \qquad \overrightarrow{\mathbf{D}}(MN) := \overrightarrow{\mathbf{D}}(M)\overrightarrow{\mathbf{D}}(N)$$

$$\overrightarrow{\mathbf{D}}(\langle M, N \rangle) := \left\langle \overrightarrow{\mathbf{D}}(M), \overrightarrow{\mathbf{D}}(N) \right\rangle \qquad \overrightarrow{\mathbf{D}}(\pi_i M) := \pi_i \overrightarrow{\mathbf{D}}(M)$$

$$\overrightarrow{\mathbf{D}}(\mathsf{if}(P \leq 0; M, N)) := \mathsf{if}(\pi_1 \overrightarrow{\mathbf{D}}(P) \leq 0; \overrightarrow{\mathbf{D}}(M), \overrightarrow{\mathbf{D}}(N))$$

$$\overrightarrow{\mathbf{D}}(\mathsf{f}) := \lambda \mathbf{z}. \left\langle \mathsf{f}\langle \pi_1 \mathbf{z} \rangle, \sum_{i=1}^{k} (\pi_2 z_i) \cdot \partial_i \mathsf{f} \langle \pi_1 \mathbf{z} \rangle \right\rangle$$

Lemma. $M \to N$ *implies* $\overrightarrow{\mathbf{D}}(M) \to^* \overrightarrow{\mathbf{D}}(N)$

Soundness for simple programs

Basic assumption: *for every primitive* $\mathsf{f} : \mathsf{R}^k \to \mathsf{R}$ *and for all* $1 \le i \le k$, *we have* $[\![\partial_i \mathsf{f}]\!] = \partial_i [\![\mathsf{f}]\!]$ *on* $\mathrm{d}(\mathsf{f})$.

Let $x_1 : \mathsf{R}, \ldots, x_n : \mathsf{R} \vdash M : \mathsf{R}$ and let

$$\overrightarrow{\mathbf{D}}_i(M) := \pi_2 \overrightarrow{\mathbf{D}}(M)[\langle x_1, 0\rangle / x_1] \cdots [\langle x_i, 1\rangle / x_i] \cdots [\langle x_n, 0\rangle / x_n]$$

We still have $x_1 : \mathsf{R}, \ldots, x_n : \mathsf{R} \vdash \overrightarrow{\mathbf{D}}_i(M) : \mathsf{R}$.

Definition. $\overrightarrow{\mathbf{D}}$ is *sound* on $S \subseteq \mathrm{d}(M)$ if $[\![\overrightarrow{\mathbf{D}}_i(M)]\!] = \partial_i [\![M]\!]$ in S.

Ideally, we would like $\overrightarrow{\mathbf{D}}$ to be sound on $\mathrm{d}(M)$ for every M!

Definition. A PCF program is *simple* if it contains no if and no recursive def.

Theorem. *For every simple program* t, $\overrightarrow{\mathbf{D}}$ *is sound on* $\mathrm{d}(t)$.

Soundness for simple programs: proof idea

Two possibilities:

- Reduce to correctness for straight-line progs (direct):

$$\begin{tikzcd}
\text{simple progs} \arrow[r, "\overrightarrow{\mathbf{D}}"] \arrow[d, "\mathsf{nf}"'] & \text{simple progs} \arrow[d, "\mathsf{nf}"] \\
\text{straight-line progs} \arrow[r, "\{-\}"'] & \text{straight-line progs}
\end{tikzcd}$$

- Logical relations.

Unsoundness

Let

$$\mathrm{sillyId} := \mathsf{if}(x \leq 0; \mathsf{if}(-x \leq 0; 0, x), x).$$

We obviously have $[\![\mathrm{sillyId}]\!] = \mathrm{id}$. However

$$\begin{aligned}\overrightarrow{\mathbf{D}}_1(\mathrm{sillyId}) &= \pi_2(\mathsf{if}(\pi_1 \langle x, 1\rangle \leq 0; \mathsf{if}(\pi_1 \langle -x, -1\rangle \leq 0; \langle 0, 0\rangle , \langle x, 1\rangle), \langle x, 1\rangle)) \\ &\sim \mathsf{if}(x \leq 0; \mathsf{if}(-x \leq 0; 0, 1), 1)\end{aligned}$$

hence

$$[\![\overrightarrow{\mathbf{D}}_1(\mathrm{sillyId})]\!](0) \neq \partial_1[\![\mathrm{sillyId}]\!](0)$$

NB: may happen in practice! If $\mathrm{ReLU}(x) := \mathsf{if}(x \leq 0; 0, x)$, then

$$[\![\mathrm{ReLU}(x) - \mathrm{ReLU}(-x))]\!] = \mathrm{id}$$

has the same behavior as above.

Approximations and traces

On types:

$$\frac{}{\mathsf{R} \sqsubset \mathsf{R}} \qquad \frac{A' \sqsubset A \quad B' \sqsubset B}{A' \times B' \sqsubset A \times B} \qquad \frac{A'_1 \sqsubset A \quad \ldots \quad A'_n \sqsubset A \quad B' \sqsubset B}{A'_1 \times \cdots \times A'_n \to B' \sqsubset A \to B}$$

On terms:

$$\frac{A_1 \sqsubset A, \ldots, A_n \sqsubset A \quad p : A_1 \times \cdots \times A_n \quad x : A}{\Xi, p \sqsubset x \vdash \pi_i p \sqsubset x} \qquad \frac{}{\Xi \vdash \underline{r} \sqsubset \underline{r}} \qquad \frac{}{\Xi \vdash \mathsf{f} \sqsubset \mathsf{f}}$$

$$\frac{\Xi, p \sqsubset x \vdash t \sqsubset M}{\Xi \vdash \lambda p.t \sqsubset \lambda x.M} \qquad \frac{\Xi \vdash t \sqsubset M \quad \Xi \vdash u_1 \sqsubset N \quad \ldots \quad \Xi \vdash u_n \sqsubset N}{\Xi \vdash t\langle u_1, \ldots, u_n\rangle \sqsubset M N}$$

$$\frac{\Xi \vdash t_i \sqsubset M_i}{\Xi \vdash \pi_i\langle t_i, t_i\rangle \sqsubset \mathsf{if}(P \leq 0; M_1, M_2)}\ i \in \{1,2\} \qquad \frac{\Xi \vdash t \sqsubset \lambda_n f(x).M}{\Xi \vdash t \sqsubset \lambda f(x).M}$$

where $\lambda_0 f(x).M := \lambda f(x).M$ and $\lambda_{n+1} f(x).M := (\lambda f.\lambda x.M)(\lambda_n f(x).M)$

On reductions: $(t \to^* u) \sqsubset (M \to N)$ if $t \sqsubset M$, $u \sqsubset N$ and t "simulates" M.

$t \underset{\sim}{\sqsubset} M$ if (reduction of t) $\sqsubset$ (reduction of M to normal form).

Soundness on stable points

Let $x_1 : \mathsf{R}, \ldots, x_n : \mathsf{R} \vdash M : \mathsf{R}$.

Definition. A point $\mathbf{r} \in \mathbb{R}^n$ is *stable* for M if
there exist $t \sqsubset M$ and $\varepsilon > 0$ such that
$\forall\, \mathbf{r}' \in \mathbb{R}^n$, $\|\mathbf{r}' - \mathbf{r}\| < \varepsilon$ implies $t[\mathbf{r}'/\mathbf{x}] \sqsubseteq_{\sim} M[\mathbf{r}'/\mathbf{x}]$.

Theorem. *For every M, $\overrightarrow{\mathbf{D}}$ is sound on the stable points of $\mathrm{d}(M)$.*

The proof is based on

Lemma. *$\mathbf{r}$ stable for M implies that there exists $t \sqsubset M$ such that $[\![M]\!] = [\![t]\!]$ on a neighborhood of $\mathbf{r}$.*

Lemma. *If $t[\mathbf{r}/\mathbf{x}] \sqsubseteq_{\sim} M[\mathbf{r}/\mathbf{x}]$ and u is the normal form of $\overrightarrow{\mathbf{D}}_i(t)[\mathbf{r}/\mathbf{x}]$, then $\overrightarrow{\mathbf{D}}_i(M)[\mathbf{r}/\mathbf{x}]$ has a normal form N and $u \sqsubset N$.*

Quasivarieties and unsoundness

$h : \mathbb{R}^k \rightharpoonup \mathbb{R}$ is *basic* if it is in the clone generated by $\{[\![\mathsf{f}]\!]\}_{\mathsf{f}\text{ primitive}}$.

Additional assumption: *for every basic function h:*

1. *h is continuous on its domain;*
2. *if $h \neq 0$, then $h^{-1}(0)$ is of Lebesgue measure zero in $\mathbb{R}^k$.*

For example, me may restrict to f's such that $[\![\mathsf{f}]\!]$ is analytic on its domain.

Definition. *Quasivariety* $Z \subseteq \mathbb{R}^k$ if $\exists\{h_i : \mathbb{R}^k \to \mathbb{R}\}_{i<\omega}$ basic non-zero

$$Z \subseteq \bigcup_{i<\omega} h_i^{-1}(0)$$

Quasivarieties are negligible: they are of measure zero and are stable under subsets and countable unions.

Theorem. *The unstable points of a program form a quasivariety.*

Corollary. *For every M, the set $\{\mathbf{r} \in \mathrm{d}(M) \mid [\![\overrightarrow{\mathbf{D}}_i(M)]\!](\mathbf{r}) \neq \partial_i[\![M]\!](\mathbf{r})\}$ is a quasivariety. In particular, $\overrightarrow{\mathbf{D}}$ is sound on almost all of $\mathrm{d}(M)$.*

Proof: logical predicates

$\mathrm{U}(M)$:= unstable converging points of M. $\Gamma := x_1 :$ R$, \ldots, x_n :$ R.

$$\mathrm{P}_\Gamma(\mathsf{R}) := \{\Gamma \vdash M : \mathsf{R} \mid \mathrm{U}(M) \text{ is a quasivariety}\}$$

$$\mathrm{P}_\Gamma(A \to B) := \{\Gamma \vdash M : A \to B \mid \forall N \in \mathrm{P}_\Gamma(A),\ MN \in \mathrm{P}_\Gamma(B)\}$$

$$\mathrm{P}_\Gamma(A_1 \times A_2) := \{\Gamma \vdash M : A \times B \mid \forall i \in \{1, 2\},\ \pi_i M \in \mathrm{P}_\Gamma(A_i)\}$$

Proof: logical predicates with quasicontinuity

$\mathrm{U}(M)$:= unstable converging points of M. $\Gamma := x_1 : \mathsf{R}, \ldots, x_n : \mathsf{R}$.

$$\mathrm{P}_\Gamma(\mathsf{R}) := \{\Gamma \vdash M : \mathsf{R} \mid \mathrm{U}(M) \text{ is a quasivariety and } [\![M]\!] \text{ is cqc}\}$$

$$\mathrm{P}_\Gamma(A \to B) := \{\Gamma \vdash M : A \to B \mid \forall N \in \mathrm{P}_\Gamma(A),\ MN \in \mathrm{P}_\Gamma(B)\}$$

$$\mathrm{P}_\Gamma(A_1 \times A_2) := \{\Gamma \vdash M : A \times B \mid \forall i \in \{1,2\},\ \pi_i M \in \mathrm{P}_\Gamma(A_i)\}$$

Definition. *Quasiopen set* of $\mathbb{R}^n$ (U open and h basic):

$$Q, Q' ::= U \mid h^{-1}(0) \mid \bigcup_{i<\omega} Q_i \mid Q \cap Q'$$

Definition. $f : \mathbb{R}^n \rightharpoonup \mathbb{R}^m$ *quasicontinuous* if $Q \subseteq \mathbb{R}^m$ quasiopen implies $f^{-1}(Q)$ quasiopen. It is *completely quasicontinuous* (cqc) if $\mathrm{id}_{\mathbb{R}^k} \times f$ is quasicontinuous for all $k \in \mathbb{N}$.

Lemma. *$\Gamma, y_1 : A_1, \ldots, y_m : A_m \vdash M : A$ and $N_i \in \mathrm{P}_\Gamma(A_i)$ for all $1 \leq i \leq m$ implies $M[N_1/y_1]\cdots[N_m/y_m] \in \mathrm{P}_\Gamma(A)$.*

Back to derivatives

If $f : A \to B$, its derivative (if it exists), is a function

$$\mathrm{D}f : A \to (A \multimap B)$$

where $A \multimap B$ is the space of linear functions from A to B.

Given $x \in A$, one often writes $\mathrm{D}_x f$ for the function $\mathrm{D}f(x) : A \multimap B$.
With this notation, the *chain rule* becomes

$$\mathrm{D}_x(f \circ g) = \mathrm{D}_{g(x)} f \circ \mathrm{D}_x g$$

If $A = \mathbb{R}^n$, $B = \mathbb{R}^m$ and $\mathbf{x} \in \mathbb{R}^n$, $J_{\mathbf{x}} f := \mathrm{D}_{\mathbf{x}} f$ is the *Jacobian matrix* ($m \times n$), or gradient $\nabla_{\mathbf{x}} f$ if $m = 1$. Composition is matrix product:

$$\mathbb{R}^{n_0} \xrightarrow{f_1} \mathbb{R}^{n_2} \xrightarrow{f_2} \cdots \xrightarrow{f_{p-1}} \mathbb{R}^{n_{p-1}} \xrightarrow{f_p} \mathbb{R}^{n_p}$$

$$J_{\mathbf{x}}(f_p \circ \cdots \circ f_1) = J_{f_{p-1}(\ldots f_1(\mathbf{x}))} f_p \cdots J_{\mathbf{x}} f_1$$

NB: when $A = B = \mathbb{R}$, the Jacobian matrix is just a scalar, hence the high school definition.

Computing gradients: from forward to reverse mode

Consider a straight-line programs P with p lines. The i-th line

```
zi = gi(y1,...,yk)
```

induces a function $f_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$, with $n_{i-1} \geq k$ and n_i equal to the number of variables (including `zi`) used by the lines $> i$.

Hence, $[\![P]\!] = f_p \circ \cdots \circ f_1$ as above, and computing $\nabla[\![P]\!]$ means:

- computing each matrix $J_{f_{i-1}(\ldots f_1(\mathbf{x}))} f_i$
- multiply them together.

$\overrightarrow{\mathbf{D}}$ may be adapted to compute $\nabla[\![P]\!]$, starting from the right.
But matrix product is associative, so we may also start from the left!

- Say that $n_0 \approx n_1 \approx \cdots \approx n_{p-1} \approx n$, whereas $n_p = 1$.
- $J_{\mathbf{x}} f_1, J_{f_1(\mathbf{x})} f_2, \ldots, J_{f_{p-2}(\ldots f_1(\mathbf{x}))} f_{p-1}$ are $n \times n$.
- $J_{f_{p-1}(\ldots f_1(\mathbf{x}))} f_p$ is a row vector of size n!

We go from $O(n^2)$ scalar products to $O(n)$!

Reverse mode AD as transposition

Remember: if $f : \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbf{x} \in \mathbb{R}^n$

$$J_{\mathbf{x}} f : \mathbb{R}^n \multimap \mathbb{R}^m$$

Linear maps may be **transposed**:

$$J^t_{\mathbf{x}} f : \mathbb{R}^m \multimap \mathbb{R}^n$$

Technically, this uses $(-)^{\perp}$, but $\mathbb{R}^{\perp} \cong \mathbb{R}$.

The chain rule becomes

$$\mathbb{R}^{n_0} \xrightarrow{f_1} \mathbb{R}^{n_2} \xrightarrow{f_2} \dots \xrightarrow{f_{p-1}} \mathbb{R}^{n_{p-1}} \xrightarrow{f_p} \mathbb{R}^{n_p}$$

$$J^t_{\mathbf{x}}(f_p \circ \dots \circ f_1) = J^t_{\mathbf{x}} f_1 \circ \ldots \circ J^t_{f_{p-1}(\ldots f_1(\mathbf{x}))} f_p$$

Reverse mode AD for straight-line programs

```
def f(x1,...,xn):
    z1 = g1(v1,...,vk)
    ...
    zp = gp(w1,...,wh)
    return zp
```

⇝

```
def grad_f(x1,...,xn):
    z1 = g1(v1,...,vk)
    ...
    zp = gp(w1,...,wh)
    # reverse pass starts here
    dx1,...,dxn,dz1,...,dz{p-1},dzp = 0,...,0,0,...,0,1
    dw1 += dgp(1,w1,...,wh) * dzp
    ...
    dwh += dgp(h,w1,...,wh) * dzp
    ...
    dv1 += dg1(1,v1,...,vk) * dz1
    ...
    dvk += dg1(k,v1,...,vk) * dz1
    return dx1,...,dxn
```

Backpropagators and derivatives

- For an arbitrary space E, let $\mathbb{R}^\perp := \mathbb{R} \multimap E$ and $\mathbb{R}^\bullet := \mathbb{R} \times \mathbb{R}^\perp$.
- An element of $\mathbb{R}^\perp$ is called backpropagator.
- If $f : \mathbb{R} \to \mathbb{R}$ is differentiable, we define $f^\bullet : \mathbb{R}^\bullet \to \mathbb{R}^\bullet$ as follows:

$$f^\bullet(x, x^*) := (f(x), \lambda a.x^*(a\, f'(x)))$$

- We have

$$(f \circ g)^\bullet = f^\bullet \circ g^\bullet \qquad\qquad \mathrm{id}^\bullet = \mathrm{id}$$

- Furthermore, notice that, taking $E = \mathbb{R}$,

$$f^\bullet(x, \lambda a.a) = (f(x), \lambda a.a\, f'(x))$$

Backpropagators and gradients

- If $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, we define $f^\bullet : \mathbb{R}^{\bullet n} \to \mathbb{R}^\bullet$ as follows:

$$f^\bullet(x_1, x_1^*, \ldots, x_n, x_n^*) := \left(f(\vec{x}), \lambda a. \sum_{i=1}^{n} x_i^*(a\, \partial_i f(\vec{x})) \right)$$

- We still have

$$(f \circ (g_1, \ldots, g_n))^\bullet = f^\bullet \circ (g_1^\bullet, \ldots, g_n^\bullet)$$

- Furthermore, notice that, taking $E = \mathbb{R}^n$,

$$f^\bullet(x_1, \iota_1, \ldots, x_n, \iota_n) = (f(\vec{x}), \lambda a. a \nabla f(\vec{x}))$$

  where $\iota_i : \mathbb{R} \multimap \mathbb{R}^n$ is the i-th injection.

Reverse mode AD in PCF with linear negation

- Types: $A, B ::= \mathsf{R} \mid A \times B \mid A \to B \mid \mathsf{R} \multimap A$
- Same programs. Typing judgments $\Gamma_{\text{non-lin}}; \Delta_{\text{lin}} \vdash M : A$ to track linearity.
- Linear factoring rule: if $x^* : \mathsf{R} \multimap A$,

$$x^* M + x^* N \;\to\; x^*(M + N)$$

- Reverse mode AD has source PCF and target PCF with linear negation.
- For any type E, we let $\overleftarrow{\mathbf{D}}_E(\mathsf{R}) := \mathsf{R} \times (\mathsf{R} \multimap E)$. Homomorphic on the rest.
- On programs, homomorphic everywhere except

$$\overleftarrow{\mathbf{D}}_E(\underline{r}) := \langle \underline{r}, \lambda a.0 \rangle \qquad\qquad \overleftarrow{\mathbf{D}}_E(\mathsf{f}) := \lambda \mathbf{z}. \left\langle \mathsf{f}\langle \pi_1 \mathbf{z} \rangle, \lambda a. \sum_{i=1}^{k} (\pi_2 z_i)(a\, \partial_i \mathsf{f} \langle \pi_1 \mathbf{z} \rangle) \right\rangle$$

Lemma. *$M \to N$ implies $\overleftarrow{\mathrm{D}}(M) \to^* \overleftarrow{\mathrm{D}}(N)$*

Soundness for reverse mode AD

We work under the same assumptions about the f's as above.

Let $x_1 : \mathsf{R}, \ldots, x_n : \mathsf{R} \vdash M : \mathsf{R}$ and let (using $\mathsf{R}^\perp = \mathsf{R} \multimap \mathsf{R}^n$)

$$\mathbf{grad}(M) := (\pi_2 \overrightarrow{\mathbf{D}}(M)[\langle x_1, \iota_1 \rangle / x_1] \cdots [\langle x_n, \iota_n \rangle / x_n])1$$

We have $x_1 : \mathsf{R}, \ldots, x_n : \mathsf{R} \vdash \mathbf{grad}(M) : \mathsf{R}^n$.

Definition. $\overleftarrow{\mathbf{D}}$ is *sound* on $S \subseteq \mathrm{d}(M)$ if $[\![\mathbf{grad}(M)]\!] = \nabla[\![M]\!]$ in S.

The soundness proof may be adapted to reverse mode:

Theorem. *For every M, $\overleftarrow{\mathbf{D}}$ is sound on the stable points of $\mathrm{d}(M)$.*

Corollary. *For all M, the set $\{\mathbf{r} \in \mathrm{d}(M) \mid [\![\mathbf{grad}(M)]\!](\mathbf{r}) \neq \nabla[\![M]\!](\mathbf{r})\}$ is a quasivariety. In particular, $\overleftarrow{\mathbf{D}}$ is sound on almost all of $\mathrm{d}(M)$.*

Soundness and efficiency for simple programs

$$\begin{tikzcd}
\text{simple progs} \arrow[rrr, "\overleftarrow{\mathbf{D}}"] \arrow[dd, "m"'] & & & \text{simple progs} \arrow[d, "O(m)\text{ (by 2-functoriality)}"] \\
& & & \text{simple progs} \arrow[d, "\text{linear factoring}"] \\
\text{straight-line progs} \arrow[r, "(-)^{\bullet}"] & \text{straight-line progs} & \sim & \text{straight-line progs}
\end{tikzcd}$$

Without linear factoring, execution is inefficient. Consider

$$M := (\lambda z.z \sin z)N$$

$$\overleftarrow{\mathbf{D}}(M) \to^* (\lambda\langle z, z^*\rangle.\langle z \sin z, \lambda a.z^*(a \sin z) + z^*(az \cos z)\rangle)\, \langle \underline{r}, \lambda b.B\rangle$$

Duplicating $\lambda b.B$ is inefficient, need to apply factoring before.
This brings up the question of how to implementat all this.

A personal, partial bibliography

1964 Wengert: reverse mode AD

1976 Joss (PhD Thesis): forward mode AD as a transformation on (Turing-complete) straight-line programs

1980 Speelpenning (PhD Thesis): backprop on straight-line programs

2008 Pearlmutter and Siskind: backprop is higher order! Differentiable programming *ante litteram*

2016 Abadi et al.: TensorFlow

2017 Paszke et al.: PyTorch

2018 Elliott (ICFP): AD is functorial!

2019 Wang, Zheng, Decker, Wu, Essertel, Rompf (ICFP): backprop as typed transformation, fully general, HO

2020 Abadi and Plotkin (POPL): first-order, "internal" AD
Barthe, Crubillé, Dal Lago, Gavazzo (ESOP): correctness by logical relations
Brunel, Mazza, Pagani (POPL): reverse mode AD with linear negation, simply-typed λ-calculus
Huot, Staton, Vákár (FoSSaCS): correctness proofs by logical relations with diffeologies
Mak, Ong (Arxiv): reverse mode AD base on differential forms

2021 Kerjean and Pédrot (unpublished): AD and Dialectica (related to Pearlmutter and Siskind?)
Mazza and Pagani (POPL): (un)soundness of AD in PCF
Sherman, Michel, Carbin (POPL): semantics for AD
Vákár (ESOP): homomorphic AD

2022 Krawiec, Jones, Krishnaswami, Ellis, Eisenberg, Fitzgibbon (POPL): reverse mode AD in Haskell
Vákár, Smeding (ToPLAS): categorically-grounded AD (related to Pearlmutter and Siskind?)

2023 Alvarez-Picallo, Ghica, Sprunger, Zanasi (CSL): reverse mode AD in string diagrams
Lew, Huot, Mansinghka, ??? (unpublished): semantic proof of our POPL 2021 results, via ωPAP functions
Radul, Paszke, Frostig, Johnson, Maclaurin (POPL): how JAX works
Smeding, Vákár (POPL): implementation of our POPL 2020 paper

Challenge: “internal” AD

Differentiation as a programming primitive, not a transformation
(like [Pearlmutter and Siskind 2008], [Abadi and Plotkin 2020], or the differential λ-calculus):

$$M, N ::= x \mid \lambda x.M \mid MN \mid \ldots \mid \overleftarrow{\mathbf{D}}_\Gamma\, M \qquad \frac{x_1 : C_1, \ldots, x_n : C_n \vdash M : A}{x_1 : \overleftarrow{\mathbf{D}}_\Gamma(C_1), \ldots, x_n : \overleftarrow{\mathbf{D}}_\Gamma(C_n) \vdash \overleftarrow{\mathbf{D}}_\Gamma\, M : \overleftarrow{\mathbf{D}}_\Gamma(A)}$$

- “True” differentiable programming (with higher-order derivatives).
- Naive idea: turn the transformation defn into rewriting rules.
- But the target language must be the same as the source…
- NB: with if-then-else, internal AD breaks the std semantics:

$$[\![\lambda x.x]\!] = [\![\lambda x.\mathsf{ReLU}(x) - \mathsf{ReLU}(-x)]\!]$$

$$[\![\overleftarrow{\mathbf{D}}_\Gamma(\lambda x.x)]\!] \neq [\![\overleftarrow{\mathbf{D}}_\Gamma(\lambda x.\mathsf{ReLU}(x) - \mathsf{ReLU}(-x))]\!]$$

Question: the benefit of compositionality?

Remember the two routes:

$$
\begin{tikzcd}
\underset{\text{arbitrary progs}}{[\mathsf{R}^n \to \mathsf{R}]} \arrow[rrr, "\text{compositional backprop}"] \arrow[d, "\text{exec}"'] & & & {[\mathbf{D}(\mathsf{R})^n \to \mathbf{D}(\mathsf{R})]} \arrow[d, "\text{exec}"] \\
\underset{\text{straight-line progs}}{[\mathsf{R}^n \to \mathsf{R}]} \arrow[r, "\text{backprop}"'] & {[\mathsf{R}^{n+1} \to \mathsf{R}^{1+n}]} & \rightsquigarrow & {[\mathbf{D}(\mathsf{R})^n \to \mathbf{D}(\mathsf{R})]}
\end{tikzcd}
$$

Question:

are there examples (NN architectures. . .) where the HO route is *substantially better* (faster, more convenient. . .) than the FO route?

Current implementations do not seem to provide an answer.

Challenge: almost-everywhere correctness?

- The set of inputs on which AD is incorrect has measure zero.
- The set of representable reals has measure zero (it's actually finite).
- Smartass. Ok, look, in $\mathrm{PCF}_{+,\times}$ it's actually of this form:

$$\mathrm{Fail} \subseteq \bigcup_{i<\omega} P_i^{-1}(0)$$

  where the P_i are polynomials (not identically zero, not necessarily distinct).
- In fact, the P_i come from "cusps" of if-then-else statements.
- Is it possible to automatically infer an upper bound on Fail?

Question: AD in the differential λ-calculus?

The diff λ-calculus computes derivatives with respect to numbers which are *not* the ones that programs have direct access to.

- In the differential λ-calculus:
  - type $=$ topological $\mathbb{R}$-vector space
  - program $A \to B$ $=$ smooth function $A \to B$
  - derivative $=$ smooth function of type $A \to (A \multimap B)$
  - unit type $= \mathbb{R}$, Booleans $= \mathbb{R}^2$, reals $= \mathbb{R}\langle$uncountable basis$\rangle$.
  - $0.5 \cdot 2 + 0.5 \cdot 4 = 3 \neq 0.5 \cdot 2 + 0.5 \cdot 4$.
- Different behavior at higher types. Below, $f : \mathsf{R} \to \mathsf{R}$:

$$D(\lambda x^{\mathsf{R}}.f(fx)) = \lambda x^{\mathsf{R}}.\alpha(fx) + f'(fx) \cdot (\alpha x) \quad \text{with } \alpha : \mathsf{R} \to \mathsf{R}$$

$$\overrightarrow{\mathbf{D}}(\lambda x^{\mathsf{R}}.f(fx)) = \lambda X^{\mathsf{R}^2}.F(FX) \quad \text{with } F : \mathsf{R}^2 \to \mathsf{R}^2$$

- There is no differential PCF! (Recently fixed by Ehrhard's *coherent differentiation*).