

Ch 1 Auto Differentiation (AD) aka "Algorithmic Differentiation"

Monday, September 16, 2024 10:11 AM

"Reverse mode" is aka "Backpropagation"

contested history, rediscovered many times

See associated Jupyter notebook

"Under quite reasonable assumptions, the evaluation of the gradient requires no more than $5 \times$ the effort of evaluating the underlying function itself"
— Andreas Griewank '88

Software Examples

ADIFOR — Fortran code, won Wilkinson prize in 1995

TensorFlow, PyTorch, Jax — modern Python packages

Adimat — Matlab code (not a lot of good options in Matlab)

... many ... — Julia has many good options

What is AD?

Not finite differences. Not exactly symbolic. It's using the chain rule (like symbolic) but in computationally efficient ways.

Applies automatically to your source code, and some AD packages can handle "for" loops and other complicated constructions

but first, motivation

If you can compute derivative symbolically, should you?

Ex Specifying function $g(x) = \prod_{i=1}^m g_i(x)$, eg take $g_i(x) = x - t_i$, $x \in \mathbb{R}$

so $g(x) = \prod_{i=1}^m (x - t_i)$, i.e., a m-degree polynomial in factored form

A polynomial in coefficient form $a_n x^n + a_{n-1} x^{n-1} + \dots$ can be efficiently evaluated via Horner's method, and also easy to find derivative

but in this factored form, a naive symbolic differentiator might give you the Calc I product rule output:

$$g'(x) = \sum_{i=1}^m \prod_{j \neq i} (x - t_j) \text{ which takes } O(m^2) \text{ to evaluate!}$$

$$\text{or } g'(x) = \sum_{i=1}^m \frac{g(x)}{x - t_i} \text{ now } O(m) \text{ but unstable when } x \approx t_i$$

Reverse-Mode AD approach speeds this up (and not unstable) at the cost of extra memory:

$$\text{let } f^{(0)} = 1, f^{(k)} = \prod_{i=1}^k x - t_i \quad \text{"forward"}$$

$$r^{(k)} = \prod_{i=k}^n x - t_i, r^{(n+1)} = 1 \quad \text{"reverse"}$$

def $g(x)$:

$f = 0$

for i in range(m):

$f = f \cdot (x - t[i])$

return f

computing this is cheap ($O(m)$) if done like

$$f^{(k)} = (x - t_k) f^{(k-1)}$$

$$r^{(k)} = (x - t_k) r^{(k+1)}$$

} ... but adds

$O(m)$ storage ...

$$\text{then } g'(x) = \sum_{i=1}^m f^{(i-1)} \cdot r^{(i+1)}$$

Fundamental speed vs. storage tradeoff in AD

Symbolic vs Automatic Differentiation

Monday, September 16, 2024 10:31 AM

Symbolic Differentiation

Gives a mathematical object

Doesn't give an implementation

(or, in practice, it does give you one implementation.)

Need not be a good one.

You might be able to call Mathematica's "Simplify"
to get another)

i.e. an equivalence class of implementations

$$\text{ex: } f(x) = \sin(x)$$

$$f'(x) = \cos(x) = \cos(x) + (7 - 17 = \sin(\pi/2 - x))$$

$$= \frac{\sin(x)}{\tan(x)} = \dots \text{many implementations!}$$

AD

Computes derivative, but how depends not just on the function, but depends on how you implement it.

$$\text{ex: } f(x) = 0$$

def $f(x)$:

return $x - x$

then AD will (implicitly) define

def $fprime(x)$:

return 1 - 1

Mathematical and computational object

Forward vs Reverse Mode

Thursday, September 4, 2025 5:00 PM

AutoDiff is efficiently applying the chain rule

Write your function as a composition, sum and/or product

of simpler functions... until it's in terms of simple primitives

(like $\sin(\cdot)$, multiplication, e^x , etc). for which we know the derivatives.

(conceptually... in practice, the software does this)

Simplest explanation is via an example:

$$f: \mathbb{R}^n \rightarrow \mathbb{R}, \quad f(\vec{x}) = \text{sum}(B \cdot (A \cdot \vec{x})), \quad A, B \in \mathbb{R}^{n \times n}, \quad \vec{x} \in \mathbb{R}^n$$

$$= \underbrace{\vec{c}^T \cdot B \cdot A \cdot \vec{x}}_{\vec{c}^T}, \quad \vec{c} \in \mathbb{R}^n \text{ is all ones vector.}$$

$$= \vec{c}^T \cdot \vec{x}.$$

Fact: for a linear function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$, $f(\vec{x}) = C\vec{x}$, $C \in \mathbb{R}^{m \times n}$

then $\overline{J_f}(x) = C$

$$\text{so } \mathbf{J}_f(\vec{x}) = \vec{c}^T, \text{ i.e. } \nabla f(\vec{x}) = \vec{c} \quad (= \mathbf{A}^T \mathbf{B}^T \cdot \mathbf{1}). \quad \left. \right\} \text{our derivation by hand}$$

What does AutoDiff do?

does AutoDiff do?
 $f(\vec{x}) = P(g(r(\vec{x})))$, $r(\vec{x}) = A \cdot \vec{x}$, $J_r(\vec{x}) = A$

50

$$g(\vec{y}) = \beta \cdot \vec{y}, \quad \sigma_g(\vec{y}) = \beta$$

$$P(\vec{z}) = \vec{\Pi}^T \cdot \vec{z}, \quad J_P(\vec{z}) = \vec{\Pi}^T$$

$$J_f(x) = J_p(g(r(x))) \cdot J_g(r(x)) \cdot J_r(x) \quad \} \text{Chain rule}$$

to evaluate this, two options:

① FORWARD MODE

$$J_f(\vec{x}) = \left(J_p(g(r(\vec{x}))) \cdot \left(J_q(r(\vec{x})) \cdot \left(J_r(\vec{x}) \right) \right) \right)$$

$$= I^T \cdot (B \cdot A) \quad \text{Cost: } B \cdot A \text{ is } O(n^3) \text{ flops}$$

(2) REVERSE MODE

$$\begin{aligned} J_f(\bar{x}) &= \left(J_p(g(r(\bar{x}))) \right) \cdot J_g(r(\bar{x})) \cdot J_r(\bar{x}) \\ &= (I^T \cdot B) \cdot A \end{aligned}$$

Cost: $I^T B$ is $O(n^2)$ flops

Need to do a forward pass first, save values, then make backward pass. REQUIRES A LOT OF MEMORY

↓
More detail
on next page

AD in more detail

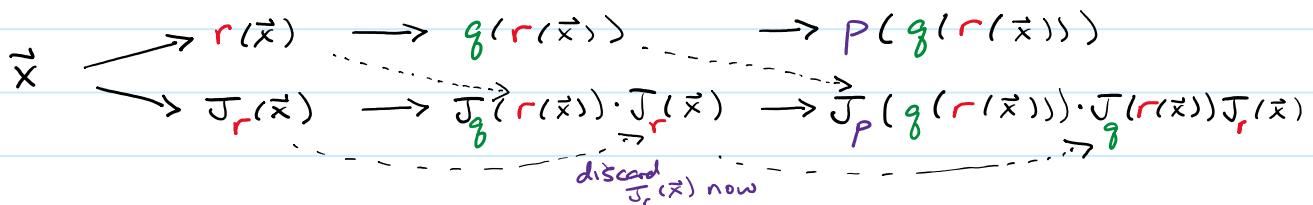
Friday, September 5, 2025 9:32 AM

... in more detail:

FORWARD MODE

$$f(\vec{x}) = p(g(r(\vec{x})))$$

$$J_f(\vec{x}) = \left(J_p(g(r(\vec{x}))) \cdot \left(J_g(r(\vec{x})) \cdot (J_r(\vec{x})) \right) \right)$$

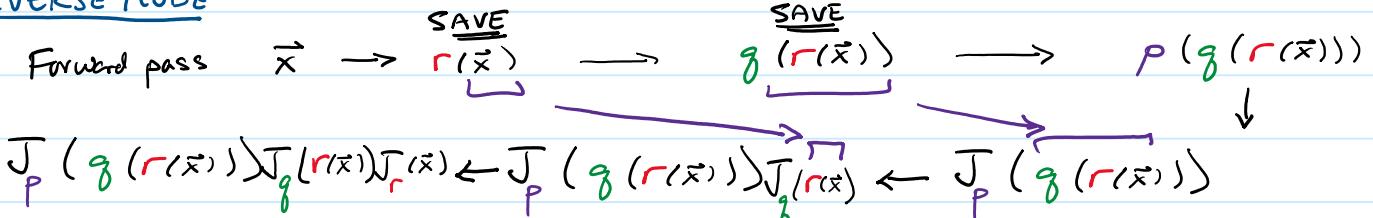


$$B \cdot A$$

$(n \times n) \cdot (n \times n)$ is $O(n^3)$ flops

$J^T(BA)$ is $O(n^2)$ flops ... $O(n^3)$ flops total
 $(1 \times n) \cdot (n \times n)$

REVERSE MODE



more efficient in terms of computation (if $f: \mathbb{R}^n \rightarrow \mathbb{R}$)
but requires more memory

In general, if

$$\begin{array}{c} \nearrow \\ f: \mathbb{R}^n \rightarrow \mathbb{R}^P \\ \text{Common setting} \end{array}$$

forward-mode costs n times the cost of a function evaluation
reverse-mode costs the same as a function eval.

forward-mode costs the same as a function evaluation
reverse-mode costs P times...

$$f: \mathbb{R}^n \rightarrow \mathbb{R}^P$$

forward mode costs n times the cost of a function eval
reverse-mode costs P times...

"Backprop" = Reverse-mode, popular since $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is common
... but memory issues!