

# Automatic 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

# AutoDiff modes: forward and reverse

Monday, September 16, 2024 10:13 AM

Modes AD has forward mode and reverse mode (and hybrid schemes, designed to mitigate downsides of both approaches)

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

Cost (in units of cost of  $f(x)$  evaluation)

Forward

$O(n)$  ... so great if  $f: \mathbb{R} \rightarrow \mathbb{R}^P$

Reverse

$O(P)$  ... so great if  $f: \mathbb{R}^n \rightarrow \mathbb{R}$

usual optimization and ML setting

$$f: \mathbb{R}^n \rightarrow \mathbb{R}, \text{ Reverse mode AD costs similar to } f(x)$$

Finite differences costs about  $n$  times  $f(x)$

... the catch?

Reverse mode AD can require huge amounts of memory.

$$\text{ex: } f(x) = h_3(h_2(h_1(x))) = w_4 \quad f: \mathbb{R} \rightarrow \mathbb{R}$$

$$f'(x) = h_3'(w_3) \cdot h_2'(w_2) \cdot h_1'(w_1)$$

$$= \frac{\partial w_4}{\partial w_3} \cdot \frac{\partial w_3}{\partial w_2} \cdot \frac{\partial w_2}{\partial w_1} \cdot \underbrace{\frac{\partial w_1}{\partial x}}_{\dot{w}_1}$$

Forward Mode

$$\dot{w}_i := \frac{\partial w_i}{\partial x} \text{ "total deriv."}$$

Reverse Mode

$$\dot{w}_i := \frac{\partial w_i}{\partial w_i} \text{ "adjoint"}$$

$$\text{ex: } \dot{w}_3 := \frac{\partial w_3}{\partial x} = \frac{\partial w_3}{\partial w_2} \cdot \frac{\partial w_2}{\partial x}$$

$\dot{w}_3$  depends on  $\dot{w}_2$  ...  
"Forward"

$$= \frac{\partial w_3}{\partial w_2} \cdot \dot{w}_2$$

lookup from set of simple operations  
(+, ×, ÷, sine,  $e^x$ , ...)

$$= \frac{\partial w_4}{\partial w_3} \cdot \frac{\partial w_3}{\partial w_2} \cdot \frac{\partial w_2}{\partial w_1} \cdot \frac{\partial w_1}{\partial x}$$

$$\text{ex: } \overline{w}_2 := \frac{\partial w_4}{\partial w_2} = \frac{\partial w_4}{\partial w_3} \cdot \frac{\partial w_3}{\partial w_2}$$

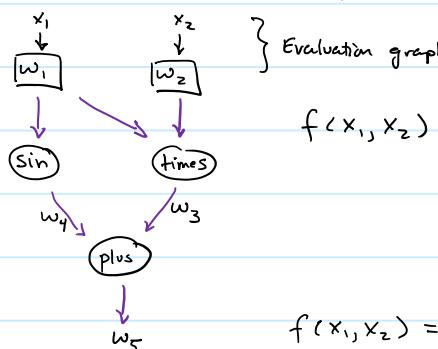
$$= \overline{w}_3 \cdot \frac{\partial w_3}{\partial w_2}$$

$\overline{w}_2$  depends on  $\overline{w}_3$   
"Backward"

# AutoDiff modes: forward and reverse

10:13 AM

**Ex**  $f: \mathbb{R}^2 \rightarrow \mathbb{R}$ ,  $f(x_1, x_2) = x_1 \cdot x_2 + \sin(x_1)$



Evaluation graph (software/compiler does this for you) [depends on your implementation of  $f$ ]

$$\begin{aligned} f(x_1, x_2) &= x_1 \cdot x_2 + \sin(x_1) \\ &= \underbrace{w_1 \cdot w_2}_{w_3} + \underbrace{\sin(w_1)}_{w_4} \\ &= \underbrace{w_3}_{w_5} + w_4 \end{aligned}$$

" $w$ " are helper variables/functions

$$f(x_1, x_2) = w_5(w_3(w_1(\bar{x}), w_2(\bar{x})), w_4(w_1(\bar{x})))$$

Forward Mode Find  $\frac{df}{dx_1}$  (to find  $\frac{df}{dx_2}$  we'll do similar, making new calculations... i.e., loop, hence  $O(m)$  cost)

Define  $\dot{w}_i = \frac{dw_i}{dx_1}$  (total derivative), so  $w_1 = x_1, \dot{w}_1 = 1$  } "Seed"  
 $w_2 = x_2, \dot{w}_2 = 0$  } "Fixed"

$$\frac{df}{dx_1} = \frac{d w_5}{d w_3} \left[ \underbrace{\frac{d w_3}{d w_1} \cdot \frac{d w_1}{d x_1} + \frac{d w_3}{d w_2} \cdot \frac{d w_2}{d x_1}}_{\frac{d w_3}{d x_1} = \dot{w}_3} \right] + \frac{d w_5}{d w_4} \left[ \underbrace{\frac{d w_4}{d w_1} \cdot \frac{d w_1}{d x_1}}_{\dot{w}_4} \right]$$

Make a forward pass of the tree, passing on pair of variables  $(w_i, \dot{w}_i)$

At each node (like "times"), do operation and calculus

aka "dual numbers"

$$w_3 = w_1 \cdot w_2 \quad (\dots \text{so } \frac{d w_3}{d w_1} = w_2 \dots)$$

$$\dot{w}_3 = w_1 \cdot \dot{w}_2 + \dot{w}_1 \cdot w_2 \quad (\text{product rule})$$

$$(\text{matches } \dot{w}_3 = \underbrace{\frac{d w_3}{d w_1} \cdot \frac{d w_1}{d x_1}}_{\dot{w}_1} + \underbrace{\frac{d w_3}{d w_2} \cdot \frac{d w_2}{d x_1}}_{w_1})$$

$$w_4 = \sin(w_1)$$

$$\dot{w}_4 = \cos(w_1) \cdot \dot{w}_1$$

$$w_5 = w_3 + w_4$$

$$\dot{w}_5 = \dot{w}_3 + \dot{w}_4 - \text{Done } \checkmark$$

This is the "intuitive" way to do AD

# AutoDiff modes

Monday, September 16, 2024

10:13 AM

Reverse Mode

aka Backpropagation

$$\frac{\partial f}{\partial x_1} = \frac{\partial w_5}{\partial w_3} \cdot \left[ \frac{\partial w_3}{\partial w_1} \cdot \frac{\dot{w}_1}{\partial x_1} + \frac{\partial w_3}{\partial w_2} \cdot \frac{\dot{w}_2}{\partial x_1} \right] + \frac{\partial w_5}{\partial w_4} \cdot \left[ \frac{\partial w_4}{\partial w_1} \cdot \frac{\dot{w}_1}{\partial x_1} \right]$$

FORWARD MODE

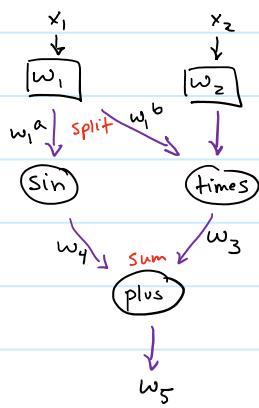
Philosophy: FORWARD : for  $w$ , how does  $w$  change w.r.t input?  $\frac{\partial w}{\partial x_i}$

REVERSE : for  $w$ , how does output change w.r.t  $w$ ?  $\frac{\partial y_i}{\partial w}$

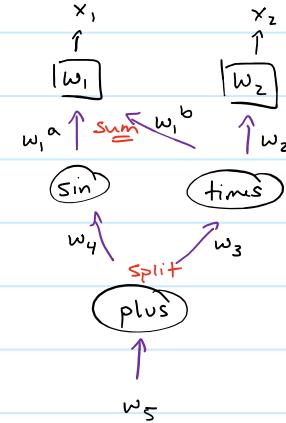
Def "adjoint"  $\bar{w}_i = \frac{\partial y_i}{\partial w_i}$  where output is  $f(x) = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix}$   
 $f: \mathbb{R}^n \rightarrow \mathbb{R}^p$

If  $p > 1$ , loop over all  $p$  outputs (but all  $n$  inputs done simultaneously)

Use same function evaluation tree:



... but we'll traverse it in reverse order



goal is to find  $\bar{w}_1 (= \frac{\partial y_1}{\partial w_1} = \frac{\partial y_1}{\partial x_1})$  and  $\bar{w}_2 (= \frac{\partial y_1}{\partial w_2} = \frac{\partial y_1}{\partial x_2})$

Start at  $w_5$ :

$$y_1 = w_5 \text{ so } \bar{w}_5 (= \frac{\partial y_1}{\partial w_5}) = 1 \quad \text{"seed"}$$

# AutoDiff modes

Monday, September 16, 2024 10:14 AM

then

$$w_5 = w_3 + w_4, \text{ so } \bar{w}_4 = \frac{\partial y_1}{\partial w_4} = \underbrace{\frac{\partial y_1}{\partial w_5}}_{\substack{\text{chain rule} \\ \bar{w}_5}} \cdot \underbrace{\frac{\partial w_5}{\partial w_4}}_{\substack{\text{via calculus}}} = \bar{w}_5 \cdot 1 = \bar{w}_5$$

(so  $\frac{\partial w_5}{\partial w_4} = 1$ )

and similarly  $\bar{w}_3 = \bar{w}_5$

then

$$w_4 = \sin(w_1^a), \text{ so } \bar{w}_1^a = \frac{\partial y_1}{\partial w_1^a} = \underbrace{\frac{\partial y_1}{\partial w_4}}_{\substack{\text{chain rule} \\ \bar{w}_4}} \cdot \underbrace{\frac{\partial w_4}{\partial w_1^a}}_{\substack{\text{via calculus}}} = \bar{w}_4 \cdot \cos(w_1) \stackrel{=}{\uparrow}$$

(so  $\frac{\partial w_4}{\partial w_1^a} = \cos(w_1)$ )

wait a minute...  $w_1 = x$ , is easy, but what if we had  $\cos(w_{15})$  or something. What is  $w_{15}$ ?

(Reverse mode  
is a forward pass  
then a backward pass)

Solution: on the forward pass (needed to just evaluate  $f(x)$ )  
we save all intermediate variables  $w_i$ :

$$\text{then } w_3 = w_1^b \cdot w_2, \text{ so } \bar{w}_1^b = \frac{\partial y_1}{\partial w_1^b} = \underbrace{\frac{\partial y_1}{\partial w_3}}_{\substack{\text{chain rule} \\ \bar{w}_3}} \cdot \underbrace{\frac{\partial w_3}{\partial w_1^b}}_{\substack{\text{via calculus}}} = \bar{w}_3 \cdot w_2 \stackrel{=}{\uparrow}$$

(so  $\frac{\partial w_3}{\partial w_1^b} = w_2$ )

via forward pass

and similarly  $\bar{w}_2 = \bar{w}_3 \cdot w_1^b = \bar{w}_3 \cdot \underline{w_1} = \frac{\partial y_1}{\partial x_2} \checkmark$

then  $w_1 = w_1^a$

and  $w_1 = w_1^b$

$$(\text{so } \frac{\partial y_1}{\partial w_1} = \frac{\partial y_1}{\partial w_1^a} + \frac{\partial y_1}{\partial w_1^b})$$

calculus

$$\text{so } \bar{w}_1 = \bar{w}_1^a + \bar{w}_1^b = \frac{\partial y_1}{\partial x_1} \checkmark$$

Done!

punchline

Reverse-mode AD is cleverly grouping parentheses in the chain rule

## Loops

are possible: just think (conceptually) of unrolling them

even adaptive works, e.g. for  $i=1, 2, \dots \infty$

$$x \leftarrow x/2$$

break if  $|x - x_{\text{old}}| < \epsilon$

Since we don't need a formula for all points,  
just a computation for current point/input