

# Automatic Differentiation

Friday, April 9, 2021 6:37 AM

contested history, rediscovered many times

aka AD

aka Algorithmic Differentiation

See associated Jupyter notebook

"Under quite reasonable assumptions, the evaluation of the gradient requires no more than 5x 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 Speelpenning function  $f(x) = \prod_{i=1}^m g_i(x)$ , 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:

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

$$r^{(0)} = \prod_{i=k}^n x - t_i, r^{(n+1)} = 1 \quad \text{"reverse"}$$

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 ...

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

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 setting

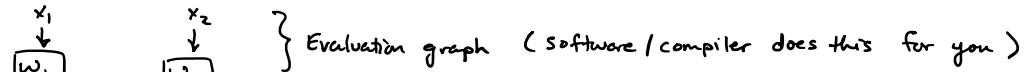
$f: \mathbb{R}^n \rightarrow \mathbb{R}$ , Reverse mode AD costs similar to  $f(x)$

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

... the catch?

Reverse mode AD can require huge amounts of memory.

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



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

"w" are helper variables / functions

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

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  $w_1 = x_1, \dot{w}_1 = 1$ ,  $w_2 = x_2, \dot{w}_2 = 0$  } "Seed"

$$\begin{aligned} \frac{df}{dx_1} &= \frac{\partial f}{\partial w_3} \left[ \underbrace{\frac{\partial w_3}{\partial w_1} \frac{\partial w_1}{\partial x_1} + \frac{\partial w_3}{\partial w_2} \frac{\partial w_2}{\partial x_1}}_{\frac{dw_3}{dx_1} = \dot{w}_3} \right] + \frac{\partial f}{\partial w_4} \left[ \underbrace{\frac{\partial w_4}{\partial w_1} \frac{\partial w_1}{\partial x_1}}_{\dot{w}_4} \right] \end{aligned}$$

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

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

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

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

$$w_4 = \sin(w_1)$$

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

$$w_5 = w_3 + w_4$$

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

aka  
Backpropagation

$$\begin{aligned} \text{Reverse Mode} & \quad \text{FORWARD MODE} \\ \frac{\partial f}{\partial x_i} = \frac{\partial w_5}{\partial w_3} \cdot \left[ \frac{\partial w_3}{\partial w_1} \cdot \frac{\partial w_1}{\partial x_i} + \frac{\partial w_3}{\partial w_2} \cdot \frac{\partial w_2}{\partial x_i} \right] + \frac{\partial w_5}{\partial w_4} \cdot \left[ \frac{\partial w_4}{\partial w_1} \cdot \frac{\partial w_1}{\partial x_i} \right] \\ & \quad \bar{w}_3 \qquad \qquad \qquad \bar{w}_4 \\ & \quad \bar{w}_1^b \qquad \qquad \qquad \bar{w}_1^a \end{aligned}$$

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

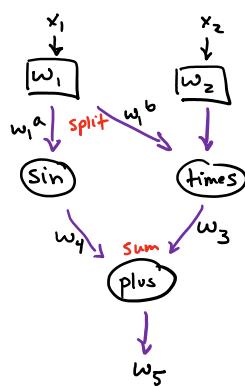
REVERSE : for  $w$ , how does output change w.r.t  $w$ ?  $\frac{\partial y_i}{\partial w}$

*not same as  
adjoint of  
linear operator*

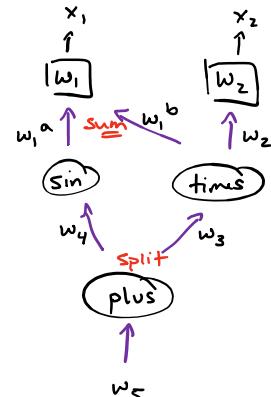
Def "adjoint"  $\bar{w}_i = \frac{\partial y_i}{\partial w_i}$  *Fixed* where output is  $f(x) = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix}$   
 $f: \mathbb{R}^n \rightarrow \mathbb{R}^p$

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  $w_5$ :

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

then

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

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

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

then

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

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

↑  
via forward pass

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

wait a minute...  $w_1 = x_1$  is easy, but what if we had  
 $\cos(w_1)$  or something. What is  $w_1$ ?

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

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

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

↑  
via forward pass

and similarly  $\bar{w}_2 = \bar{w}_3 \cdot w_1^b = \bar{w}_3 \cdot \underline{w_1} = \frac{\partial y_1}{\partial x_1}$  ✓

then  $w_1 = w_1^a$

and  $w_1 = w_1^b$

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

Done!

punchline

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