# 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 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?

we've seen inaccuracy issue in HW. Also a cost issue in high dimension.

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 Speed penning 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:

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

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

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

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

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

```
def g(x):
    f = 0
    for i in range(m):
        f = f*(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 ...

Fundamental speed vs. storage tradeoff in AD

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

# 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)

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

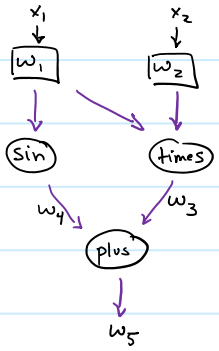
| | Cost (in units of cost of $f(x)$ evaluation) | |
|---|---|---|
| Forward | $O(n)$ ... | so great if $f: \mathbb{R} \to \mathbb{R}^p$ |
| Reverse | $O(p)$ ... | so great if $f: \mathbb{R}^n \to \mathbb{R}$ ← usual optimization and ML setting |

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




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

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

"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 $O(n)$ 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"

← Fixed

$$\frac{\partial f}{\partial x_1} = \frac{\partial w_5}{\partial w_3}\cdot\left[\underbrace{\frac{\partial w_3}{\partial w_1}\cdot\frac{\partial w_1}{\partial x_1} + \frac{\partial w_3}{\partial w_2}\cdot\frac{\partial w_2}{\partial x_1}}_{\frac{dw_3}{dx_1} = \dot{w}_3}\right] + \frac{\partial w_5}{\partial w_4}\cdot\left[\underbrace{\frac{\partial w_4}{\partial w_1}\cdot\frac{\partial w_1}{\partial x_1}}_{\dot{w}_4}\right]$$

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

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

$w_3 = w_1 \cdot w_2$ (... so $\frac{\partial w_3}{\partial w_1} = w_2$ ...)

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

(matches $\dot{w}_3 = \underbrace{\frac{\partial w_3}{\partial w_1}}_{w_2}\cdot\underbrace{\frac{\partial w_1}{\partial x_1}}_{\dot{w}_1} + \underbrace{\frac{\partial w_3}{\partial w_2}}_{w_1}\cdot\underbrace{\frac{\partial w_2}{\partial x_1}}_{\dot{w}_2}$ )

This is the "intuitive" way to do AD

$w_4 = \sin(w_1)$

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

$w_5 = w_3 + w_4$

$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$ . Done ✓

# 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

ex: $f(x) = \sin(x)$

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

$= \frac{\sin(x)}{\tan(x)} = \ldots$ many implementations.

## AD

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

ex: $f(x) = 0$

```
def f(x):
    return x - x
```

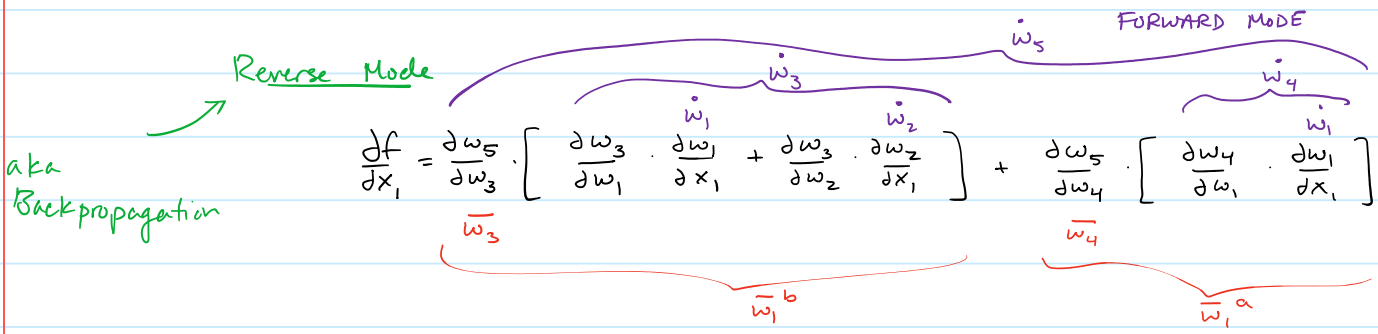
then AD will (implicitly) define

```
def fprime(x):
    return 1 - 1
```

Mathematical and computational object

# AutoDiff modes

Monday, September 16, 2024 10:13 AM




Philosophy: FORWARD: for $w_i$, how does $w$ change w/ respect to input? $\frac{dw}{dx_i}$

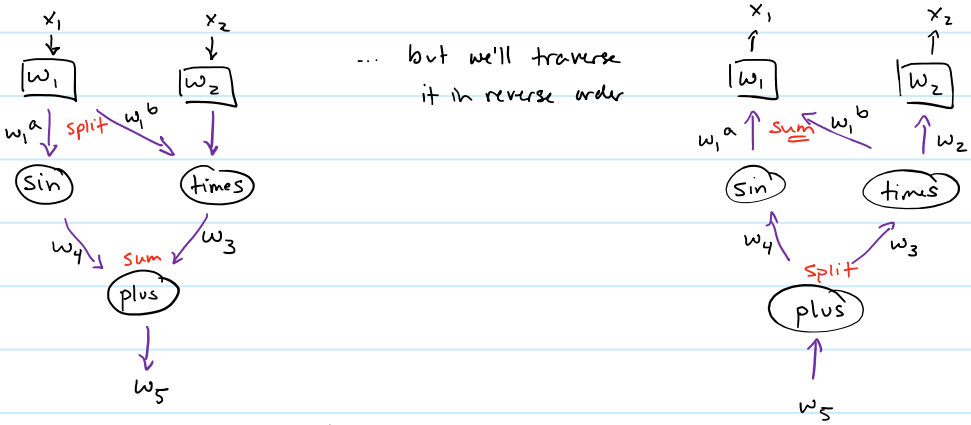
REVERSE: for $w_i$, how does output change w/ respect to $w$? $\frac{dy_i}{dw}$

not same as adjoint of linear operator

Def "adjoint" $\bar{w}_i = \frac{dy_1}{dw_i}$ ← Fixed where output is $f(x) = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix}$ $f: \mathbb{R}^n \to \mathbb{R}^p$

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

Use same function evaluation tree:




goal is to find $\bar{w}_1$ $\left( = \frac{dy_1}{dw_1} = \frac{dy_1}{dx_1} \right)$ and $\bar{w}_2$ $\left( = \frac{dy_1}{dw_2} = \frac{dy_1}{dx_2} \right)$

Start at $w_5$:

$y_1 = w_5$ so $\bar{w}_5 \left( = \frac{dy_1}{dw_5} \right) = 1$ "seed"

# AutoDiff modes

Monday, September 16, 2024 10:14 AM

then $w_5 = w_3 + w_4$, so $\bar{w_4} = \frac{dy_1}{dw_4} = \frac{dy_1}{dw_5} \cdot \frac{dw_5}{dw_4} = \bar{w_5} \cdot 1 = \bar{w_5}$ (chain rule; $\bar{w_5}$; via calculus)

(so $\frac{dw_5}{dw_4} = 1$ calculus)

and similarly $\bar{w_3} = \bar{w_5}$

then $w_4 = \sin(w_1^a)$, so $\bar{w_1^a} = \frac{dy_1}{dw_1^a} = \frac{dy_1}{dw_4} \cdot \frac{dw_4}{dw_1^a} = \bar{w_4} \cdot \cos(w_1)$ (chain rule; $\bar{w_4}$; via calculus; via forward pass)

(so $\frac{dw_4}{dw_1^a} = \cos(w_1)$ calculus)

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

wait a minute... $w_1 = x_1$ is easy, but what if we had $\cos(w_{15})$ or something. What is $w_{15}$?

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 $\bar{w_1^b} = \frac{dy_1}{dw_1^b} = \frac{dy_1}{dw_3} \cdot \frac{dw_3}{dw_1^b} = \bar{w_3} \cdot w_2$ (chain rule; $\bar{w_3}$; via calculus; via forward pass)

(so $\frac{dw_3}{dw_1^b} = w_2$ calculus; prod. rule)

and similarly $\bar{w_2} = \bar{w_3} \cdot w_1^b = \bar{w_3} \cdot w_1 = \frac{dy_1}{dx_2}$ ✓

then $w_1 = w_1^a$

and $w_1 = w_1^b$

so $\bar{w_1} = \bar{w_1^a} + \bar{w_1^b} = \frac{dy_1}{dx_1}$ ✓

(so $\frac{dy_1}{dw_1} = \frac{dy_1}{dw_1^a} + \frac{dy_1}{dw_1^b}$ calculus)

Done!

punchline

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