~~9/22/22~~

9/20/22

Automatic Differentiation

Simple Motivating Example

input x, y

(1) $a = \sin x$

(2) $b = a/y$

(3) $z = b + x$

return z

<u>Problem</u>: Find $\frac{dz}{dx}$ and $\frac{dz}{dy}$

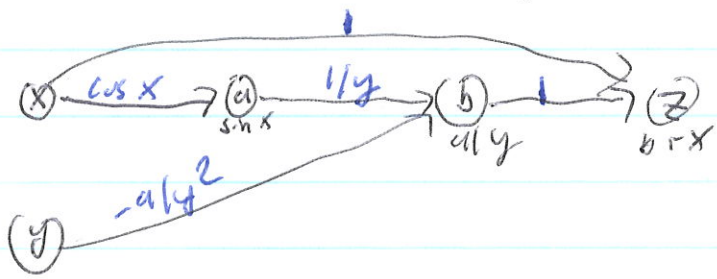
<u>Solution 1</u>: Symbolically like an 18.01 student

$z = b + x = a/y + x = \sin x/y + x$

$dz/dx = \cos x/y + 1$

$dz/dy = -\sin x/y^2$

<u>Computational</u> <u>Graph</u> (DAG)



Notation not standardized but I like to put var names on nodes.

Derivatives on edges

Put "one-step" derivs on each edge

Forward View

Follow Paths input to output & (left) multiply as you go & add multiple paths

path 1: $\underset{(3)}{1} \cdot \underset{(2)}{\frac{1}{y}} \cdot \underset{(1)}{\cos x} + \underset{(3)}{1} \rightarrow \frac{\cos x}{y} + 1$

path 2: $\underset{(2)}{1} \cdot \underset{(1)}{-\frac{a}{y^2}} \rightarrow -\frac{a}{y^2} = \frac{\sin x}{y^2}$

Remark: If you have numerical derivatives on the edges the algorithm <u>works</u>

<u>Recap view</u>

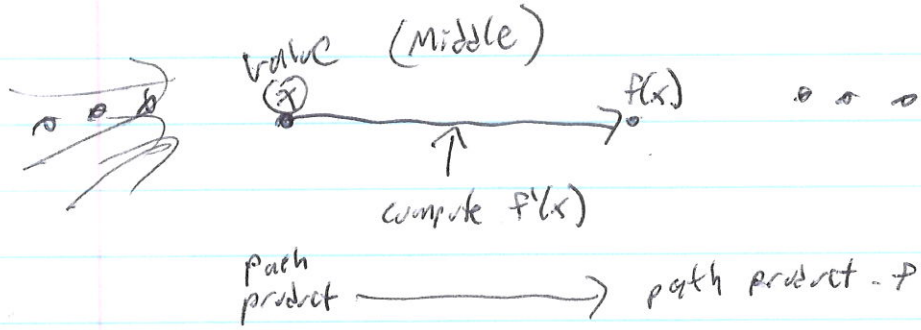
Follow Paths Backwards (write right to left)

$z \to x$: $\underset{(3)}{\cos x} \cdot \underset{(2)}{\frac{1}{y}} \cdot \underset{(1)}{1} + \underset{(3)}{1}$

$z \to y$: $\underset{(2)}{-\frac{a}{y^2}} \cdot \underset{(1)}{1}$

Implementation of Forward Mode

One Arrow Case



(value, path product) is an ordered pair

$$\text{e.g.} \quad (x, p) \xrightarrow{\cos x} (\sin x, p \cos x)$$

Let's overload every function

$$(x, p) \longrightarrow (f(x), p f'(x))$$

Starting with $(x, 1)$

Note $f(x) = 2$ $\quad (x, p) \to (2, 0)$ constant 2

Multiple Arrows Forward Mode

input x

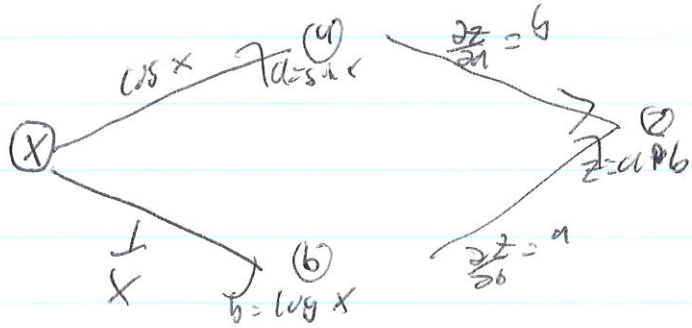
(1) $a = \sin x$

(2) $b = \log x$

(3) $z = a + b$

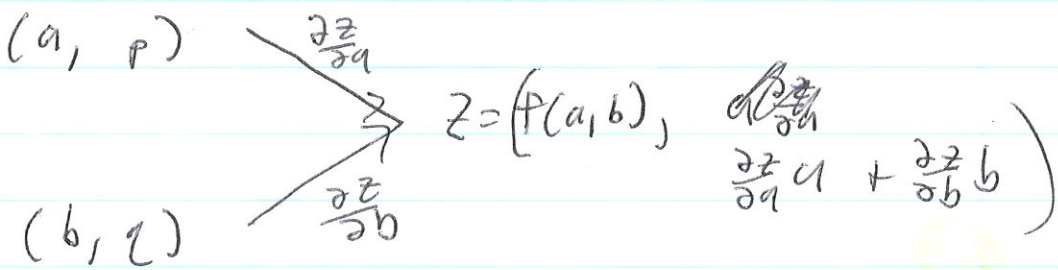
Sym $z = \sin x \log x$

$\frac{dz}{dx} = \cos x \log x + \sin x / x$



Answer: $b \cos x + \frac{a}{x}$

generalize



arrows going in

What do you see in scalar computer programs?

i constants

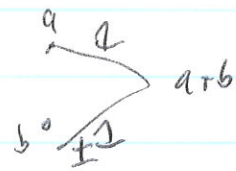
ii $+, -, *, \div$

iii built in functions: ~~+,-,*,÷~~ e.g. sqrt, cos, log, ---

iv control

i Constants: $(a, 0)$

ii $f(a,b) = a+b$



$(a,p) + (b,q) = (a+b, p+q)$

overkill way of saying derivative add

$(a,p) * (b,q) = (ab, aq + bp)$

~~Overkill way of saying~~

$f(a,b) = a/b$ $\quad \frac{\partial f}{\partial a} = \frac{1}{b}$ $\quad \frac{\partial f}{\partial b} = \frac{-a}{b^2}$

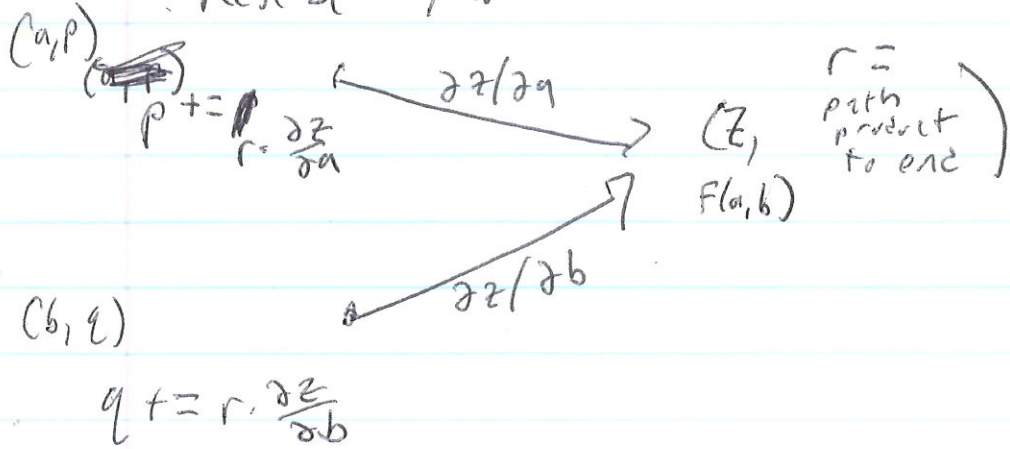
$(a,p) / (b,q) = \left(\frac{a}{b}, \frac{bp - aq}{b^2}\right)$

Recall: $\frac{(\text{den})(\text{num}) - \text{num}(\text{den})'}{(\text{den})^2}$

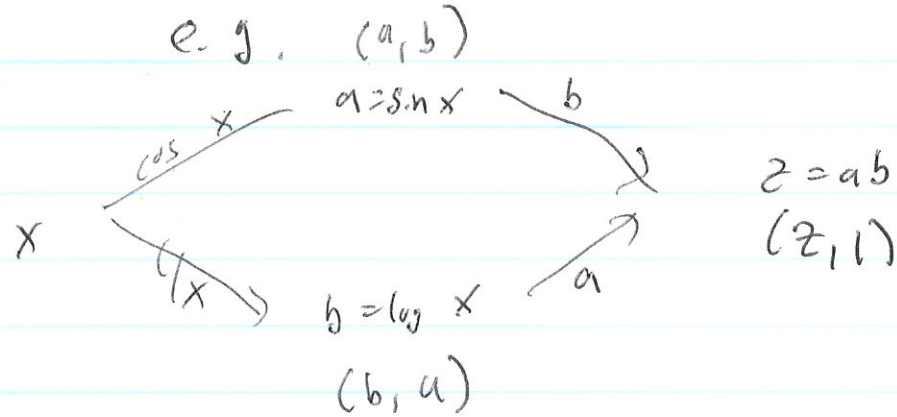
iii $f(a,p) = (f(a), f'(a)p)$

(iv) Control, no change

Multiple Arrows converge
Reverse Mult



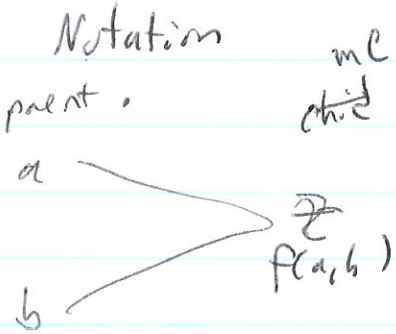
e.g.



$$(x, 0) \to (x, b\cos x) \to \left(x, b\cos x + \frac{a}{x}\right)$$

Chain Rule Notation

Forward:

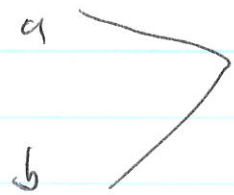


$$\frac{dz}{d(\text{start})} = \frac{\partial z}{\partial a}\frac{da}{d(\text{start})} + \frac{\partial z}{\partial b}\frac{db}{d(\text{start})}$$

$$\frac{d(\text{me})}{d(\text{start})} = \cancel{\sum \frac{\partial(\text{me})}{\partial(\text{child})}}$$

parent

Reverse



$$\sum \frac{\partial \text{me}}{\partial \text{parent}} \frac{d(\text{parent})}{d(\text{start})}$$

$$\cancel{\frac{da}{d(\text{end})} + = \frac{\partial z}{\partial(\text{end})}}$$

A's

$$\frac{d(\text{end})}{\partial a} + = \frac{d(\text{end})}{\partial z} \cdot \frac{\partial z}{\partial a}$$

$$\frac{d(\text{end})}{d(\text{me})} = \sum_{\text{parents}} \cancel{\frac{\partial(\text{end})}{\partial \text{parent}} \cdot \frac{d(\text{parent})}{d(\text{me})}}$$

$$\frac{d(\text{child})}{d(\text{end})} \frac{\partial \text{end}}{\partial(\text{me})}$$

(8)

ε notation, multidual notation

Extensions to multivariate

~~9/20/22~~ 9/22/22

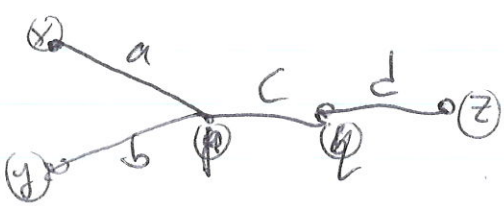
(1)

key Questions

(1) Are ~~the~~ Forward & Reverse Modes Sym?

Yes: From A big picture view
No: From a practical view, Reverse seems harder

(2) Why is reverse more efficient than Forward.



Forward: $\frac{dz}{dx} = dca$ 4 mults
$\frac{dz}{dy} = dcb$

input x, y

$p = p(x,y)$ $a = \frac{\partial p}{\partial x}$ $b = \frac{\partial p}{\partial y}$
$q = q(p)$ $c = \partial q/\partial p$
$z = z(q)$ $d = \frac{\partial z}{\partial q}$

Output z

Reverse

$d \circ c \begin{cases} a = dz/dx \\ b = dz/dy \end{cases}$

3 mults

Many inputs, few or 1 output reverse tends to be more efficient

(3) Can the variables be vectors & matrices

Answer: yes

$A = \frac{\partial p}{\partial x} = \left[\frac{\partial p_i}{\partial x_j}\right]$

x-vector

p-vector

y-vector $B = \frac{\partial p}{\partial y} = \left[\frac{\partial p_i}{\partial y_j}\right]$

$\cancel{ACD}$ DCA
$\cancel{BCD}$ DCB

or

$DC \begin{cases} A \\ B \end{cases}$

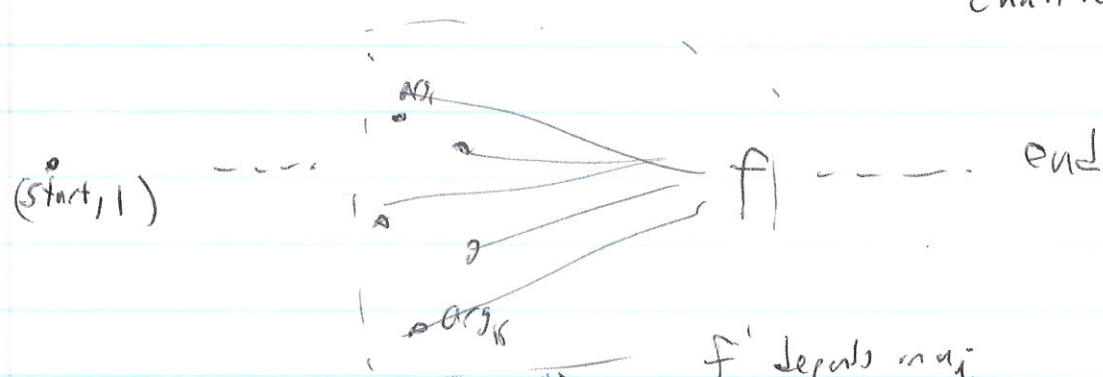
In practice $\begin{cases} A^T \\ B^T \end{cases} C^T D^T$ "adjoint method"

Forward Mode Basic step

$$result = f(arg_1, arg_2, \ldots, arg_K)$$

$$\text{(MCR)} \quad \frac{d\,result}{d\,start} = \frac{\partial f}{\partial arg_1}\frac{d\,arg_1}{d\,start} + \cdots + \frac{\partial f}{\partial arg_K}\frac{d\,arg_K}{d\,start}$$

Multivariate Chain Rule



Think path "prefixes" pushed forward to get the f prefix

f' depends on a_i

Synchronous

Given $(arg_1, d_1) \ldots (arg_K, d_K)$

compute $\frac{d\,result}{d\,start}$ using (MCR)

Remark: Since computer program lines execute at the time all args are ready, one can perform the as you go

(result, dresult)

Asynch

initially ~~(result, dresult~~ dresult = 0

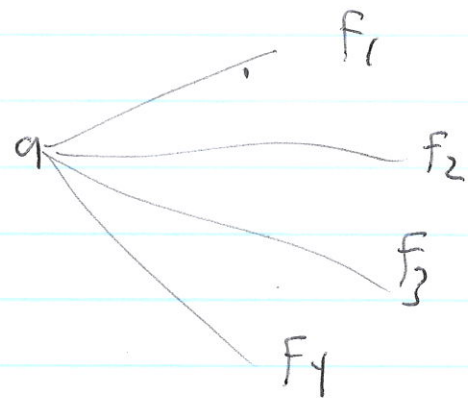
when (arg_i, d_i) is available

$$d\,result \mathrel{+}= \frac{\partial f}{\partial arg_i} d_i$$

end

can complete when everything you depend on is complete

Reverse Mode

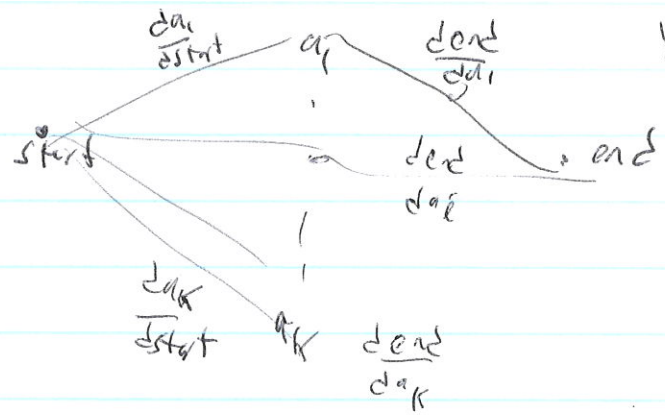


$o(\text{end}, 1)$

f_i depend on a $i=1,\dots,k$

(MCR) $$\frac{d\,\text{end}}{da} = \frac{d\,\text{end}}{df_1}\frac{\partial f_1}{\partial a} + \dots + \frac{d\,\text{end}}{df_k}\frac{\partial f_k}{\partial a}$$

Think path suffices pulling back to get earlier suff.



View: Just nodes

Reverse Mode Asynchronously

For all dependencies

$a \longrightarrow f$

if $(f, \frac{d_{end}}{df})$ is available

$(a, d_a) \longrightarrow (a, d_a + \frac{d_{end}}{df}\frac{\partial f}{\partial a})$

end

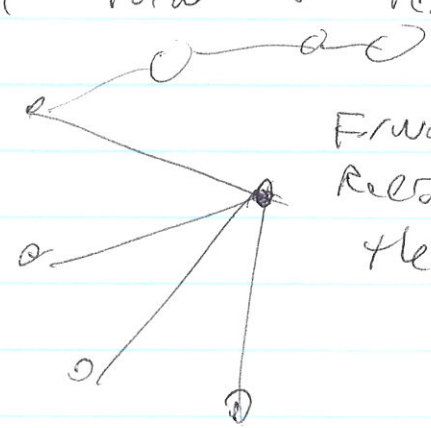
In practice

init $(end, d_{end} = 1)$

Run through nodes in reverse order of dep graph

update all vars that it depends on

Both Forw & reverse work on



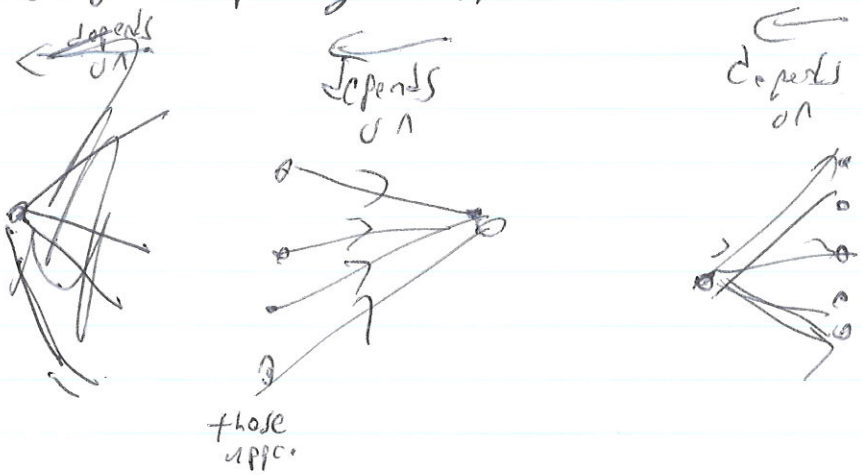
Forward will do the full sweep

Reverse will do just the one update

E

Not just a dag but a program

A program is a dag with a topological sort meaning that the nodes are ordered & ~~for~~ every dependency appears before the node



<u>Implication</u>: Forward can run with the program

Reverse can not