

# Overview of Automatic Differentiation

James Yang

27th January 2022

Introduction

Forward-Mode Automatic Differentiation

Reverse-Mode Automatic Differentiation

Tips

Applications

Benchmarks

# Introduction

## Forward-Mode Automatic Differentiation

## Reverse-Mode Automatic Differentiation

## Tips

## Applications

## Benchmarks

# What is Automatic Differentiation (AD)?

- ▶ Addresses the question: *How do we write a framework (e.g. Tensorflow, PyTorch) where users can differentiate a wide class of functions automatically?*
- ▶ Users **should not** manually write derivatives.
- ▶ Solves issues with other forms of differentiation methods.
- ▶ Seppo Linnainmaa (1970) master thesis.

# Why should we care about computing derivatives?

- ▶ Many optimization/statistical methods require the use of gradients.
- ▶ Optimizers: gradient descent and all its variants.
- ▶ Neural network + back propagation: differentiate loss w.r.t. all parameters.
- ▶ Bayesian Hamiltonian Monte Carlo Samplers (HMC, Langevin, NUTS, etc.): differentiate joint-log-pdf of hierarchical model.
- ▶ ODE/PDE solvers: differentiate a known function  $H$ , which is used to specify the dynamics of another variable.
- ▶ MLE computation: gradient descent on the (negative) log-likelihood.

# Why should we care about computing derivatives?

- ▶ Many optimization/statistical methods require the use of gradients.
- ▶ Optimizers: gradient descent and all its variants.
- ▶ Neural network + back propagation: differentiate loss w.r.t. all parameters.
- ▶ Bayesian Hamiltonian Monte Carlo Samplers (HMC, Langevin, NUTS, etc.): differentiate joint-log-pdf of hierarchical model.
- ▶ ODE/PDE solvers: differentiate a known function  $H$ , which is used to specify the dynamics of another variable.
- ▶ MLE computation: gradient descent on the (negative) log-likelihood.
- ▶ Your homework: I know you used Wolfram at some point in your life O\_\_O!

# Finite Difference (FD)

- ▶ Given  $f$  and  $x$ , compute

$$\frac{f(x+h) - f(x)}{h}$$

for small  $h$  and declare it to be  $f'(x)$ .

# Finite Difference (FD)

- ▶ Pros:
  - ▶ Very easy to implement.
  - ▶ Works for *any* programmable functions.
- ▶ Cons:
  - ▶ Suffers from numerical precision issues (dividing two small numbers leads to wild results) (example code: `fd_prec`).
  - ▶ Cannot take advantage of analytical forms even if they exist.
  - ▶ Must run  $p$  times if there are  $p$  input variables.



# Symbolic Differentiation (SD)

- ▶ E.g. Wolfram, Mathematica.
- ▶ Given a formula or mathematical expression of the function of interest  $f$ , generate a new expression for the derivative.

# Symbolic Differentiation (SD)

- ▶ Pros:
  - ▶ if  $f$  is composed of elementary functions, SD produces the analytical form for  $f'$  (no approximations).
  - ▶ Convenient output for mathematicians who need the functional form.
- ▶ Cons:
  - ▶ Difficult to represent programmatic expressions.
    - ▶ How do we differentiate an if-else statement?
    - ▶ How do we differentiate a for-loop?

# Automatic Differentiation (AD)

- ▶ Combines (most of) the Pros and solves the Cons of SD and FD.
- ▶ Computes analytical derivatives for *any* (including programmatic) expressions.
- ▶ Does not suffer from numerical precision issues as in FD.
- ▶ Does not output an expression like SD, but rather the derivative at a given  $x$  like FD.
  - ▶ Allows for optimization in implementation if it doesn't need to return the full expression for gradient.

Introduction

Forward-Mode Automatic Differentiation

Reverse-Mode Automatic Differentiation

Tips

Applications

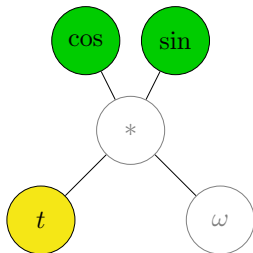
Benchmarks

# Example Function and Expression Graph

$$(x(t), y(t)) = (\cos(\omega t), \sin(\omega t))$$

# Example Function and Expression Graph

$$(x(t), y(t)) = (\cos(\omega t), \sin(\omega t))$$



# Forward AD

- ▶ Each node is represented by a *dual number*,  $(w, \frac{dw}{dx})$ .
- ▶ Extend elementary functions to dual numbers.
- ▶ Unary  $f$ :

$$f\left(\left(w, \frac{dw}{dx}\right)\right) := \left(f(w), \frac{df}{dw} \frac{dw}{dx}\right)$$

- ▶ Binary  $f$ :

$$f\left(\left(w_1, \frac{dw_1}{dx}\right), \left(w_2, \frac{dw_2}{dx}\right)\right) := \left(f(w_1, w_2), \frac{df}{dw_1} \frac{dw_1}{dx} + \frac{df}{dw_2} \frac{dw_2}{dx}\right)$$

# Forward AD

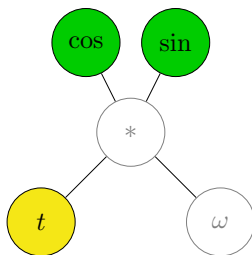
► Example:

$$\sin\left(w, \frac{dw}{dx}\right) = \left(\sin(w), \cos(w) \frac{dw}{dx}\right)$$

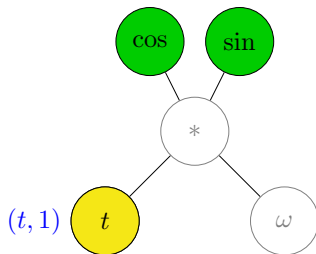
$$\left(w_1, \frac{dw_1}{dx}\right) \cdot \left(w_2, \frac{dw_2}{dx}\right) = \left(w_1 w_2, \frac{dw_1}{dx} w_2 + w_1 \frac{dw_2}{dx}\right)$$



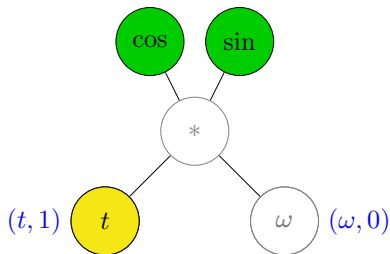
# Forward AD Evaluation



# Forward AD Evaluation

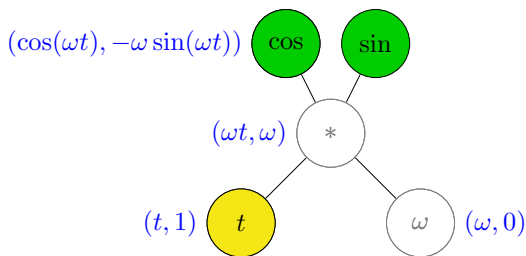


# Forward AD Evaluation

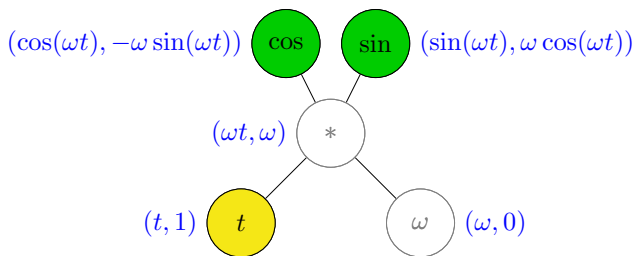




# Forward AD Evaluation



# Forward AD Evaluation



# Forward AD

- ▶ Easy to implement.
- ▶ Fast for  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  where  $m \gg n$  ( $O(n)$  sweeps of computation graph).
- ▶ Useful in physics applications when differentiating w.r.t. time.
- ▶ Example code (`fwd_ad`).

Introduction

Forward-Mode Automatic Differentiation

Reverse-Mode Automatic Differentiation

Tips

Applications

Benchmarks

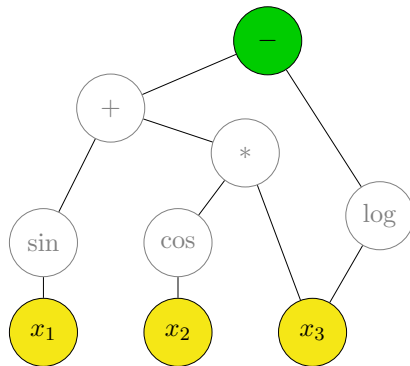


# Example Function and Expression Graph

$$f(x_1, x_2, x_3) = \sin(x_1) + \cos(x_2) \cdot x_3 - \log(x_3)$$

# Example Function and Expression Graph

$$f(x_1, x_2, x_3) = \sin(x_1) + \cos(x_2) \cdot x_3 - \log(x_3)$$



# Expression Tree Conversion

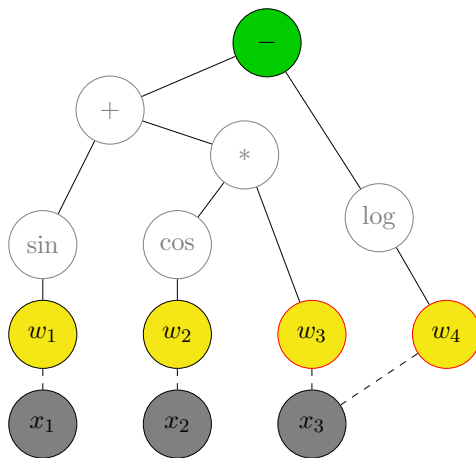
- ▶  $x_i$  can be referenced by multiple nodes.
  - ▶ e.g.  $x_3$  is referenced by the  $*$  and  $\log$  nodes.
- ▶ Convert expression graph into an expression tree.
  - ▶ Replace all nodes with multiple parents as separate nodes that reference back to the actual variables.
- ▶ Mathematically,

$$f(x_1, x_2, x_3) = \tilde{f}(g(x_1, x_2, x_3)) \quad (1)$$

$$\tilde{f}(w_1, w_2, w_3, w_4) = \sin(w_1) + \cos(w_2) \cdot w_3 - \log(w_4)$$

$$g(x_1, x_2, x_3) = (x_1, x_2, x_3, x_3)$$

# Expression Tree Conversion



# Expression Tree Conversion

- ▶ Why do we need this conversion?
  - ▶ All nodes except  $x_i$  have exactly one parent.
  - ▶ Leads to cleaner implementation.
  - ▶ Better to treat  $x_i$  as *containers* for initial values and their **adjoints**,  $\frac{\partial f}{\partial x_i}$ , instead of nodes of the graph.

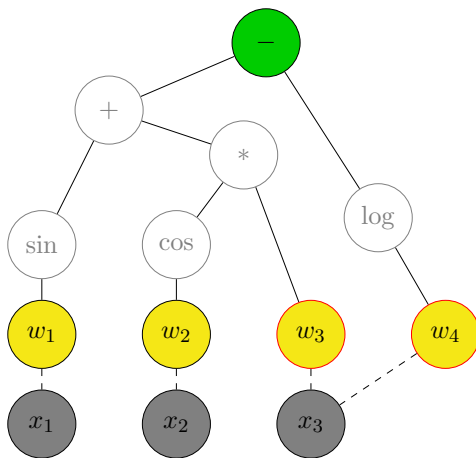
# Reverse-Mode AD Algorithm

- ▶ Assume for the moment that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ .
- ▶ Reverse-mode algorithm consists of two passes of the expression tree:
  - ▶ *forward*-evaluation (not to be confused with forward-mode AD)
  - ▶ *backward*-evaluation

# Forward-Evaluation

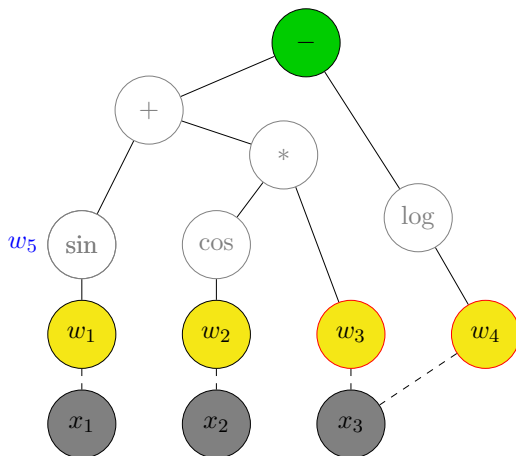
- ▶ Compute expression in the usual fashion.
  - ▶ Start at the root.
  - ▶ Recursively forward-evaluate left to right all its children.
  - ▶ Compute current node operation using children results.
    - ▶ e.g. for `sin` node,  $x_1 \rightarrow w_1 \rightarrow \sin(w_1)$

# Forward-Evaluation

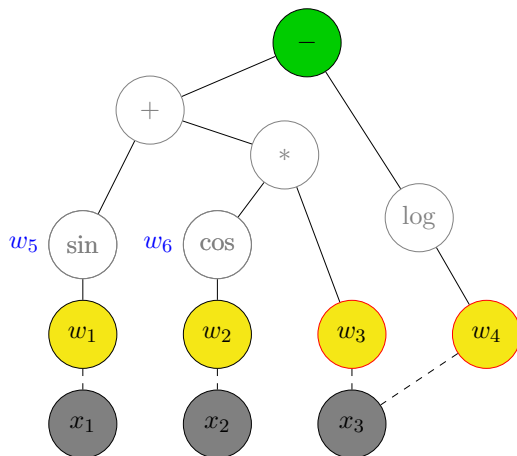




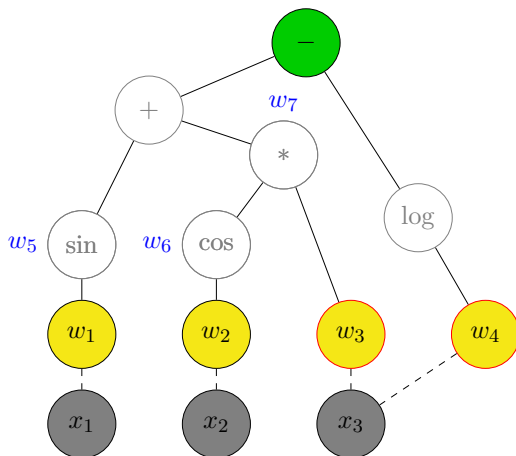
# Forward-Evaluation



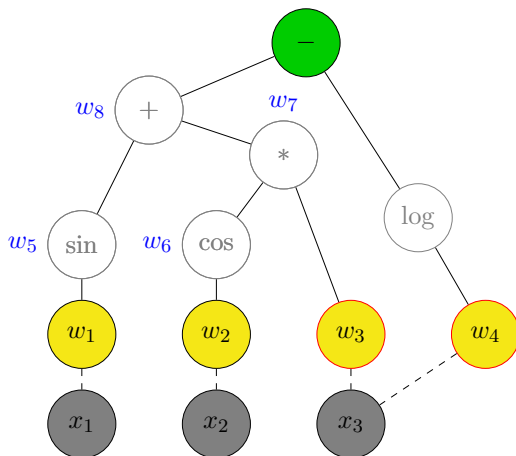
# Forward-Evaluation



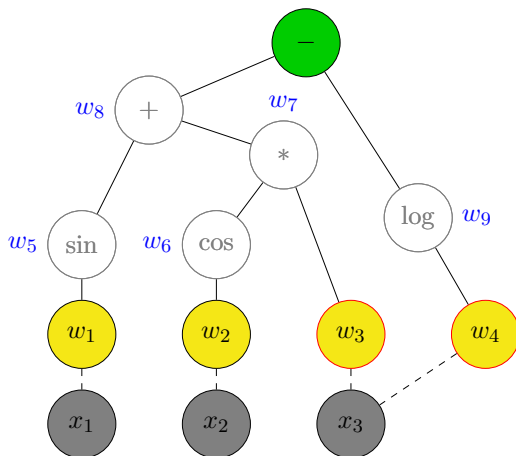
# Forward-Evaluation



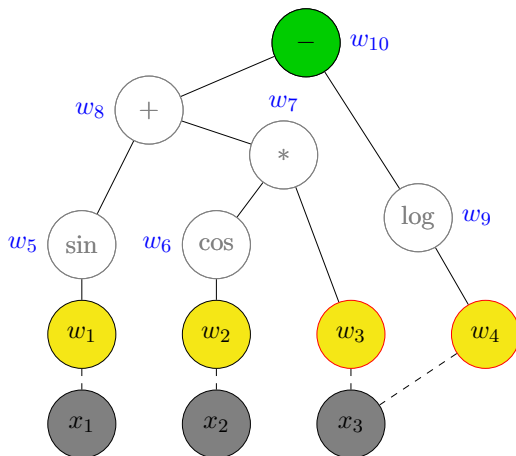
# Forward-Evaluation



# Forward-Evaluation



# Forward-Evaluation



# Backward-Evaluation

- ▶ Current node receives its adjoint from its parent.
- ▶ This adjoint is also referred to as *seed*.
- ▶ Hence, root will receive  $\text{seed} = 1$  from the caller.
- ▶ Current node computes seeds for all its children and recursively backward-evaluates from *right-to-left*.

# Backward-Evaluation: Computing Next Seed

- ▶ Next seed is computed by a simple chain-rule.
- ▶ Let the current node be  $w \in \mathbb{R}^{p \times q}$  and  $v \in \mathbb{R}^{m \times n}$  one of its children.
- ▶ The seed for  $v$  is given by

$$\frac{\partial f}{\partial v_{ij}} = \sum_{k=1}^p \sum_{l=1}^q \frac{\partial f}{\partial w_{kl}} \frac{\partial w_{kl}}{\partial v_{ij}} \quad (2)$$

- ▶ Since we are working with an expression tree,  $f$  only depends on  $v$  through  $w$ , hence this is the full adjoint.



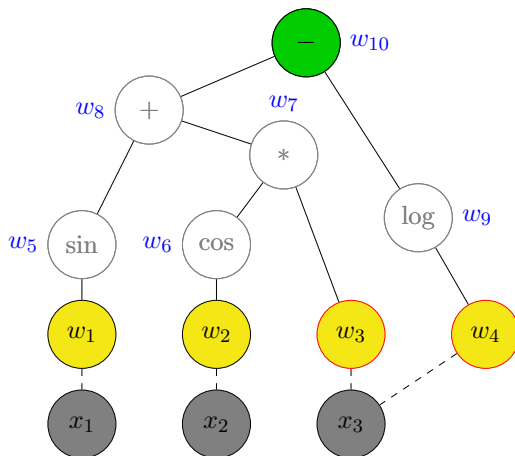
# Backward-Evaluation: Computing Next Seed

- ▶ Nodes with reference to containers must increment the adjoints in the containers with their seed.
  - ▶ e.g.  $w_3$  and  $w_4$  increments the adjoint in  $x_3$  with their seeds.
- ▶ Why? Chain-rule, once again.
- ▶ Let  $w_1, \dots, w_k$  denote all variables with a reference to  $x$ . For simplicity assume they are all scalars (easily generalizable). Then,

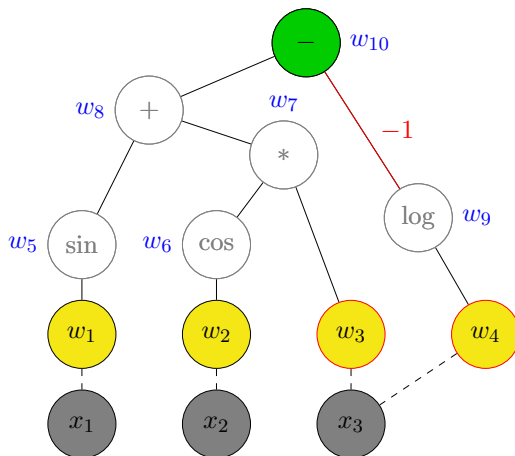
$$\frac{\partial f}{\partial x} = \sum_{i=1}^k \frac{\partial f}{\partial w_i} \frac{\partial w_i}{\partial x} = \sum_{i=1}^k \frac{\partial f}{\partial w_i}$$

- ▶ Accumulated adjoints for  $x_1, x_2, x_3$  is the gradient of  $f$ .

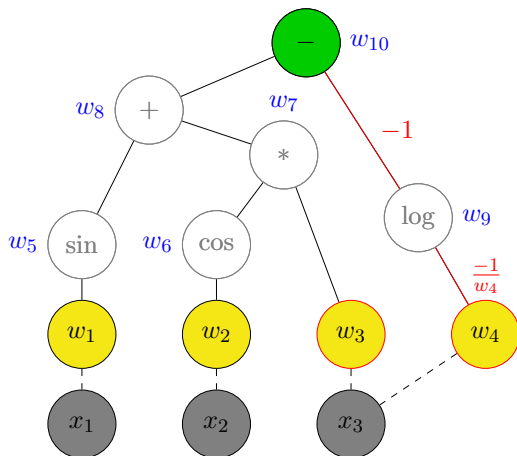
# Backward-Evaluation



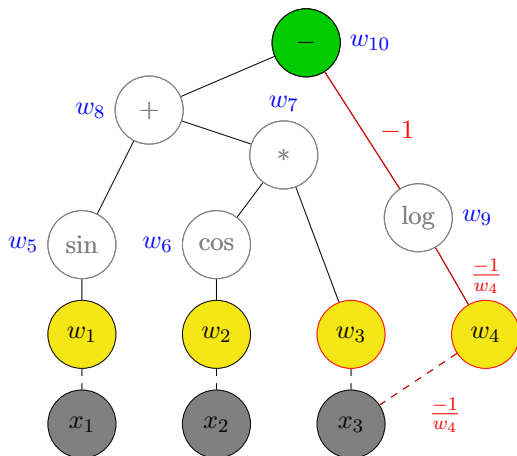
# Backward-Evaluation



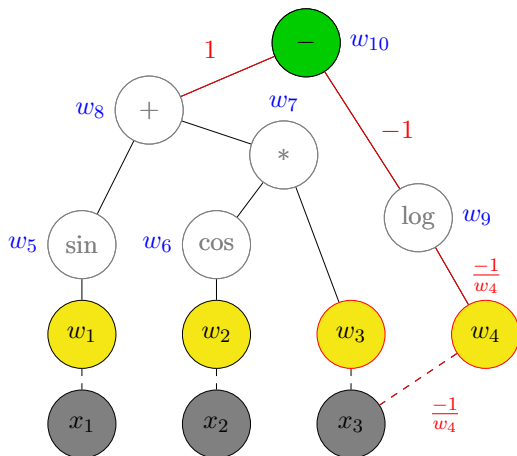
# Backward-Evaluation



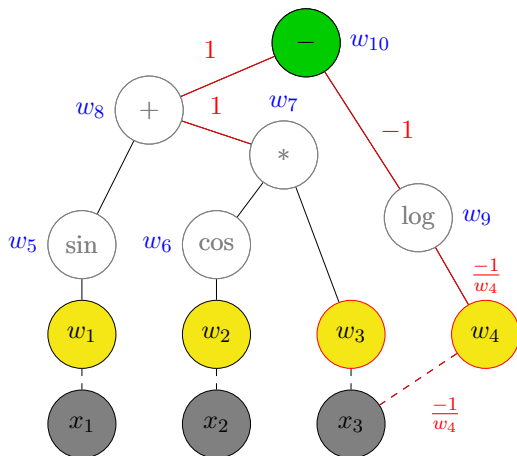
# Backward-Evaluation



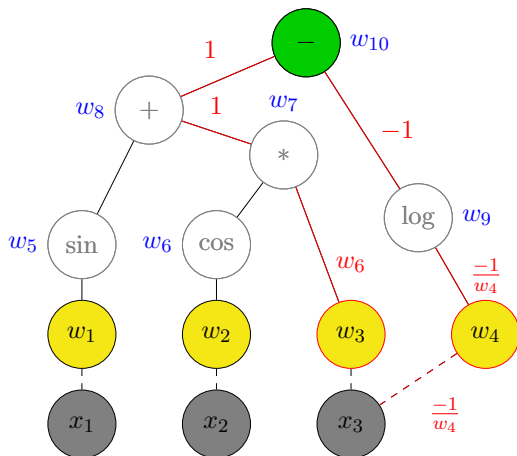
# Backward-Evaluation



# Backward-Evaluation

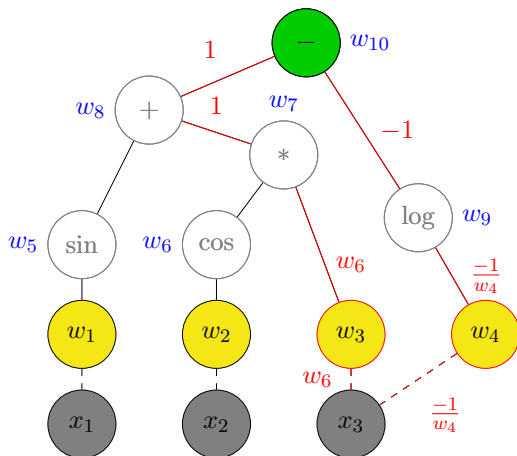


# Backward-Evaluation

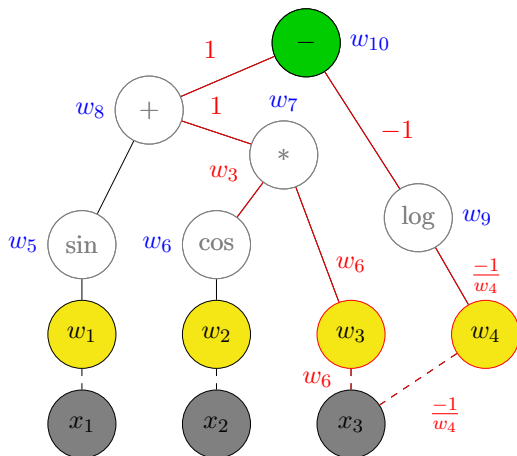




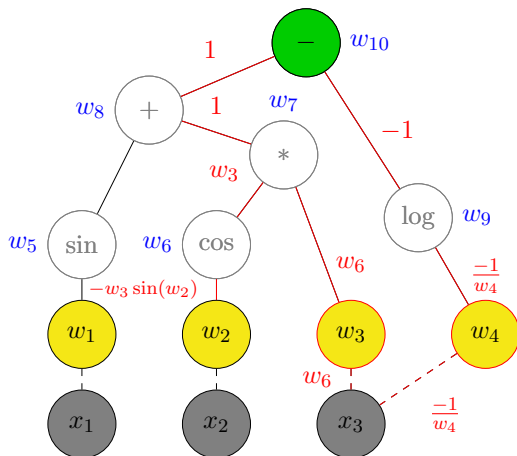
# Backward-Evaluation



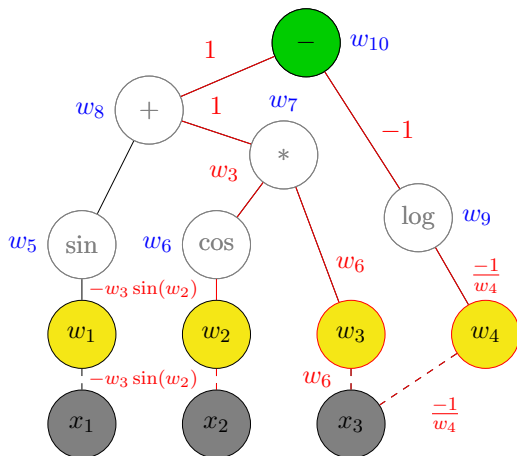
# Backward-Evaluation



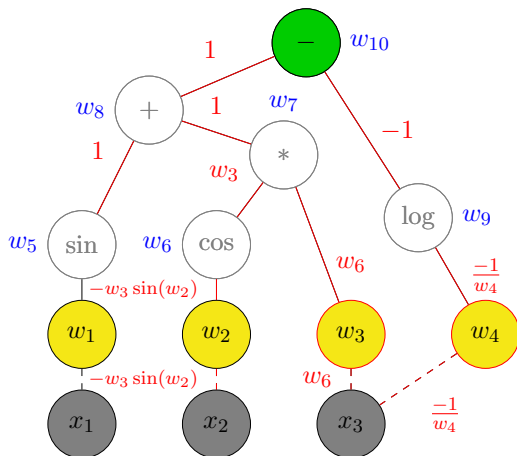
# Backward-Evaluation



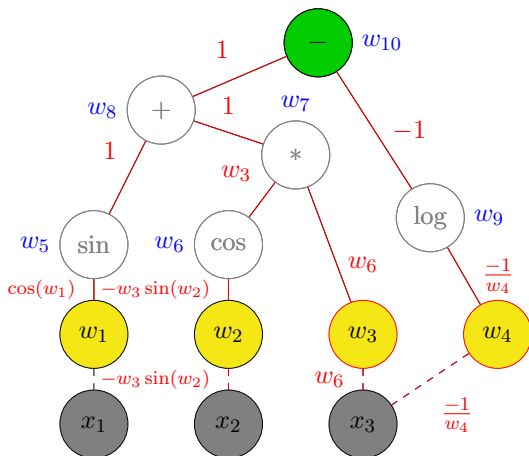
# Backward-Evaluation



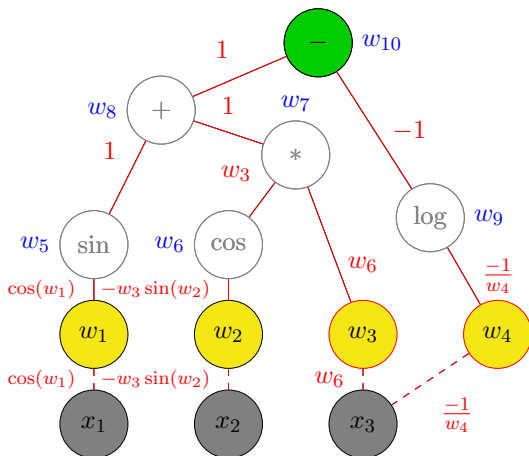
# Backward-Evaluation



# Backward-Evaluation



# Backward-Evaluation



# Backward-Evaluation

► Sanity check:

$$f(x_1, x_2, x_3) = \sin(x_1) + \cos(x_2) \cdot x_3 - \log(x_3)$$

$$\nabla f = (\cos(x_1), -x_3 \sin(x_2), \cos(x_2) - x_3^{-1})$$



# Remarks

- ▶ Much harder to implement (memory management is tricky).
- ▶ Fast for  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  where  $n \gg m$  ( $O(m)$  sweeps of computation graph).
- ▶ Useful when we need to compute gradient (of a scalar function).
- ▶ Example code:
  - ▶ `rv_ad`
  - ▶ `for_each`
  - ▶ `if_else`

Introduction

Forward-Mode Automatic Differentiation

Reverse-Mode Automatic Differentiation

**Tips**

Applications

Benchmarks

# Reverse-Mode

- ▶ Extract common sub-expression into a placeholder variable, to avoid recomputation, e.g.

$$t = x + y$$

$$f(t) = \sin(t) \cos(t)$$

- ▶ Use package-provided functions mostly to take advantage of vectorization, e.g.

for-loop sum over  $x$  ✗

sum( $x$ ) ✓

# Reverse-Mode

- ▶ Minimize number of nodes in computation graph, e.g. if  $x \in \mathbb{R}^{10}$ ,

$$x[1] + \dots + x[10] \implies 9 \text{ adjoints } \times$$

$$\text{sum}(x) \implies 1 \text{ adjoint } \checkmark$$

- ▶ Minimize size of each node of computation graph, e.g. if  $x, y \in \mathbb{R}^n$ ,

$$\text{sum}(x + y) \implies O(n) \text{ memory } \times$$

$$\text{sum}(x) + \text{sum}(y) \implies O(1) \text{ memory } \checkmark$$

Introduction

Forward-Mode Automatic Differentiation

Reverse-Mode Automatic Differentiation

Tips

**Applications**

Benchmarks

# Taste of using AD

- ▶ MLE: Gaussian model.
- ▶ MLE: Gaussian mixture model.
- ▶ MAP: Ridge.

# MLE: Gaussian model

- ▶  $X_i \stackrel{iid}{\sim} N(\mu, 1)$
- ▶  $\mu$  unknown.
- ▶ Negative log-pdf is given by

$$-\sum_{i=1}^n \log(p(x_i)) = \frac{1}{2} \sum_{i=1}^n (X_i - \mu)^2$$

- ▶ Example code (`mle`).

# MLE: Gaussian mixture model



# MLE: Gaussian mixture model

- ▶ This is for you Kevin Senpai.

# MLE: Gaussian mixture model

- ▶ This is for you Kevin Senpai.
- ▶  $X_i \stackrel{iid}{\sim} \pi N(\mu_1, \sigma_1^2) + (1 - \pi)N(\mu_2, \sigma_2^2)$
- ▶  $(\pi, \mu_1, \mu_2, \sigma_1, \sigma_2)$  unknown.
- ▶ Negative log-pdf is given by

$$-\sum_{i=1}^n \log(p(x_i)) = -\sum_{i=1}^n \log(\pi p_{\mu_1, \sigma_1}(X_i) + (1 - \pi)p_{\mu_2, \sigma_2}(X_i))$$

- ▶ Example code (`kevin_senpai_mle`).

# MAP: Ridge

- ▶ MLE is really just MAP of likelihood + non-informative prior.
- ▶ In general, can apply any (differentiable) prior.
- ▶ Ridge regression (fixed  $X$ ):

$$y|\beta \sim \mathcal{N}(X\beta, I)$$

$$\beta \sim \mathcal{N}(0, \lambda^{-1})$$

- ▶ MAP is posterior mean:

$$\mathbb{E}(\beta|y) = (X^\top X + \lambda I)^{-1} X^\top y$$

- ▶ Example code (`map`).

Introduction

Forward-Mode Automatic Differentiation

Reverse-Mode Automatic Differentiation

Tips

Applications

Benchmarks

# Comparison with Finite Difference and Manual

- ▶ Example function:  $x \in \mathbb{R}^p$ ,  $M \in \mathbb{R}^{n \times p}$

$$f(x) = \text{sum}(Mx) - 2 \log\left(\sum_{i=1}^p \exp(x_i)\right)$$

- ▶ Code (`compare_fd`)

Config	FD	AD	Manual (df only)	Manual (f + df)
n=10, p=100	2.2e-3	5.78e-5	3.66e-5	4.31e-5
n=10, p=10000	NA	4.3e-3	3.4e-3	4.1e-3
n=100, p=10	1.1e-4	4.2-5	9.09e-6	1.61e-5
n=10000, p=10	8.56e-3	2.64e-3	3e-4	1e-3