Automatic differentiation

Brooks Paige

Week 4

Those optimization routines required gradients!

Computing gradients by hand is tedious and prone to error. There are a few different ways to compute gradients **automatically**:

- Finite differences (inexact and slow)
- Symbolic differentiation
- Automatic differentiation (**autodiff**):
  - Forward mode (fast when functions have few inputs, many outputs)
  - Reverse mode (fast when functions have many inputs, few outputs)
  - ...

Symbolic differentiation

You might think this is the gold standard: an approach which replicates the "by hand" technique, but done by a computer. However, symbolic expressions for gradients can actually be inefficient.

Example: to compute the gradient of $f(x_1, x_2) = (x_1^2 + x_2^2)^2$, symbolically we find

$$\frac{\partial f}{\partial x_1} = 2(x_1^2 + x_2^2)2x_1 \qquad \frac{\partial f}{\partial x_2} = 2(x_1^2 + x_2^2)2x_2$$

Computationally, evaluating this as-is would be inefficient. It would be better to first evaluate an intermediate value $\kappa = 4(x_1^2 + x_2^2)$, and then compute

$$\frac{\partial f}{\partial x_1} = \kappa x_1 \qquad \frac{\partial f}{\partial x_2} = \kappa x_2.$$

This would require either a magic `simplify()` function, or maybe clever caching.

Example: David Barber

Symbolic differentiation problems

More broadly, this will never work well in cases with control flow (`if` or `loop` statements), particularly if random choices are involved.

```
def f(theta):
    y = theta + dist.Normal(0, 1).sample()
    if y > 2.0:
        return y
    else:
        return theta + f(theta + 0.5*theta**2)

theta = -2.0
L = f(theta)
```

What is $\frac{\partial}{\partial\theta}\mathbb{E}[f]$, evaluated at $\theta = -2$?

Autodiff

- Autodiff takes a function $f(\boldsymbol{\theta})$, $f\ \mathbb{R}^D \to \mathbb{R}$, and returns an **exact** value of the gradient $\mathbf{g} \in \mathbb{R}^D$, with $g_i(\boldsymbol{\theta}) = \frac{\partial}{\partial \theta_i} f\Big|_{\boldsymbol{\theta}}$.

Autodiff

- Autodiff takes a function $f(\boldsymbol{\theta})$, $f\ \mathbb{R}^D \to \mathbb{R}$, and returns an **exact** value of the gradient $\mathbf{g} \in \mathbb{R}^D$, with $g_i(\boldsymbol{\theta}) = \frac{\partial}{\partial \theta_i} f \big|_{\boldsymbol{\theta}}$.
- **Forward mode** calculates a derivative along a single direction (e.g. a single partial derivative or directional derivative).
  - Low memory requirements; easy to implement
  - Computing the full gradient has cost $\mathcal{O}(Df)$
  - Not straightforward to apply to conditional statements or loops

Autodiff

- Autodiff takes a function $f(\boldsymbol{\theta})$, $f\ \mathbb{R}^D \to \mathbb{R}$, and returns an **exact** value of the gradient $\mathbf{g} \in \mathbb{R}^D$, with $g_i(\boldsymbol{\theta}) = \frac{\partial}{\partial \theta_i} f\big|_{\boldsymbol{\theta}}$.
- **Forward mode** calculates a derivative along a single direction (e.g. a single partial derivative or directional derivative).
  - Low memory requirements; easy to implement
  - Computing the full gradient has cost $\mathcal{O}(Df)$
  - Not straightforward to apply to conditional statements or loops
- **Reverse mode** computes a full gradient by running the function forward, then tracing the computation graph "backward" to compute the gradient
  - Requires keeping around pointers from every computed value to its parents
  - Memory intensive (can't free any intermediate computed values...), but runtime is $\mathcal{O}(f)$ — *the same as the original function*
  - The "backprop" algorithm is a special case of reverse-mode autodiff

Forward-mode autodiff

Dual numbers

Forward mode is ingenious in its simplicity. It uses **dual arithmetic**, which resembles complex numbers.

- Define an idempotent infintesimal variable ϵ, such that $\epsilon^2 = 0$
- Define a function $\text{DualPart}(\cdot)$, which returns the "dual" component (i.e. the coefficient of ϵ)
- For any function $f(x)$, we have $f'(x) = \text{DualPart}(f(x))$.

Dual numbers

Forward mode is ingenious in its simplicity. It uses **dual arithmetic**, which resembles complex numbers.

- Define an idempotent infintesimal variable ϵ, such that $\epsilon^2 = 0$
- Define a function $\mathrm{DualPart}(\cdot)$, which returns the "dual" component (i.e. the coefficient of ϵ)
- For any function $f(x)$, we have $f'(x) = \mathrm{DualPart}(f(x))$.

This requires some primitive operations to be overloaded, in order to compute derivatives:

$$
\begin{aligned}
(v_1 + \epsilon) + v_2 &= v_1 + v_2 + \epsilon \\
(v_1 + \epsilon)v_2 &= v_1 v_2 + v_2 \epsilon \\
\sin(v_1 + \epsilon) &= \sin(v_1) + \cos(v_1)\epsilon \\
&\vdots
\end{aligned}
$$

Forward-mode examples (1/2)

Let $f(x) = x^2$, and then consider

$$f(x+\epsilon) = (x+\epsilon)^2 = x^2 + 2x\epsilon.$$

Then $f'(x) = \text{DualPart}(f(x+\epsilon)) = \text{DualPart}(x^2 + 2x\epsilon) = 2x$.

Forward-mode examples (2/2)

For functions like $f(v_1, v_2) = v_1 v_2 - \sin(v_2)$, we'd have to compute the two partial derivatives separately:

$$f(v_1 + \epsilon, v_2) = \underbrace{v_1 v_2 - \sin(v_2)}_{f(v_1, v_2)} + \underbrace{v_2}_{\frac{\partial f}{\partial v_1}} \epsilon$$

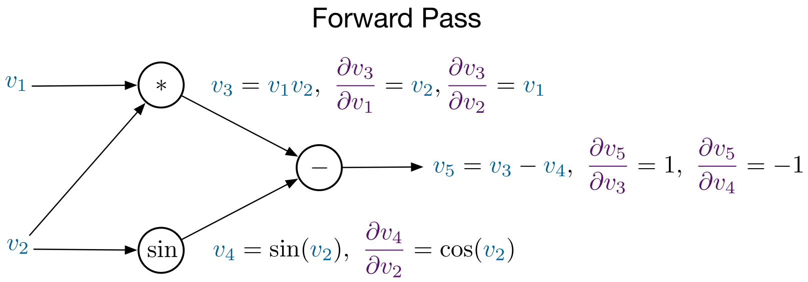
Forward-mode examples (2/2)

For functions like $f(v_1, v_2) = v_1 v_2 - \sin(v_2)$, we'd have to compute the two partial derivatives separately:

$$f(v_1 + \epsilon, v_2) = \underbrace{v_1 v_2 - \sin(v_2)}_{f(v_1,v_2)} + \underbrace{v_2}_{\frac{\partial f}{\partial v_1}} \epsilon$$

$$\begin{aligned} f(v_1, v_2 + \epsilon) &= v_1(v_2 + \epsilon) - \sin(v_2 + \epsilon) \\ &= v_1 v_2 + v_1 \epsilon - \sin(v_2) - \cos(v_2)\epsilon \\ &= \underbrace{v_1 v_2 - \sin(v_2)}_{f(v_1,v_2)} + \underbrace{(v_1 - \cos(v_2))}_{\frac{\partial f}{\partial v_2}} \epsilon \end{aligned}$$

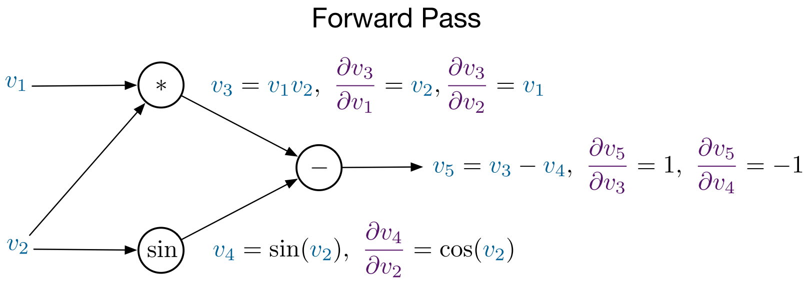
Forward computation graph



$v_1 \qquad v_2$

Figure: Van de Meent et al. (2021)

Forward computation graph

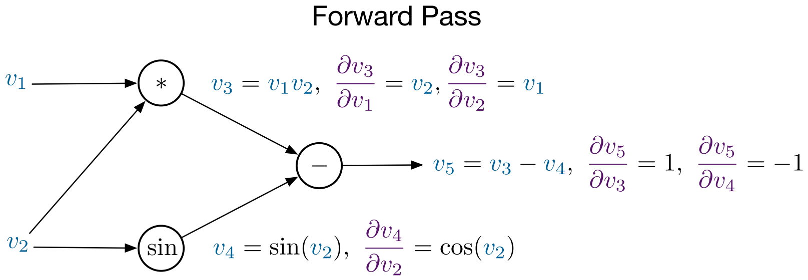


$v_1 \qquad v_2 \qquad v_3 = v_1 v_2$

$$\frac{\partial v_3}{\partial v_2} = v_1$$

Figure: Van de Meent et al. (2021)

Forward computation graph

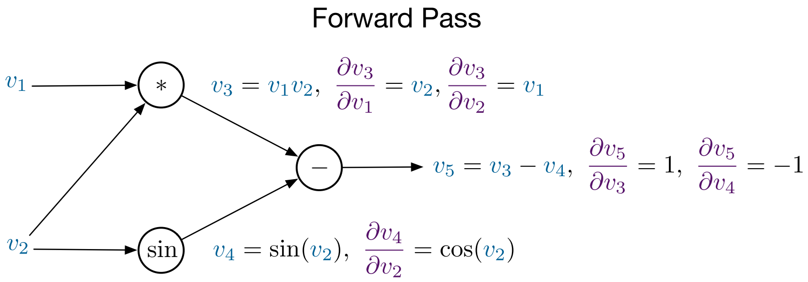


$$v_1 \qquad v_2 \qquad v_3 = v_1 v_2 \qquad v_4 = \sin(v_2)$$

$$\frac{\partial v_3}{\partial v_2} = v_1 \qquad \frac{\partial v_4}{\partial v_2} = \cos(v_2)$$

Figure: Van de Meent et al. (2021)

Forward computation graph



$$v_1 \qquad v_2 \qquad v_3 = v_1 v_2 \qquad v_4 = \sin(v_2) \qquad v_5 = v_3 - v_4$$

$$\frac{\partial v_3}{\partial v_2} = v_1 \qquad \frac{\partial v_4}{\partial v_2} = \cos(v_2) \qquad \frac{\partial v_5}{\partial v_2} = \frac{\partial v_5}{\partial v_3}\frac{\partial v_3}{\partial v_2} + \frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_2}$$

$$= (1)(v_1) + (-1)(\cos(v_2))$$

Figure: Van de Meent et al. (2021)

Reverse-mode autodiff

Reverse computation graph

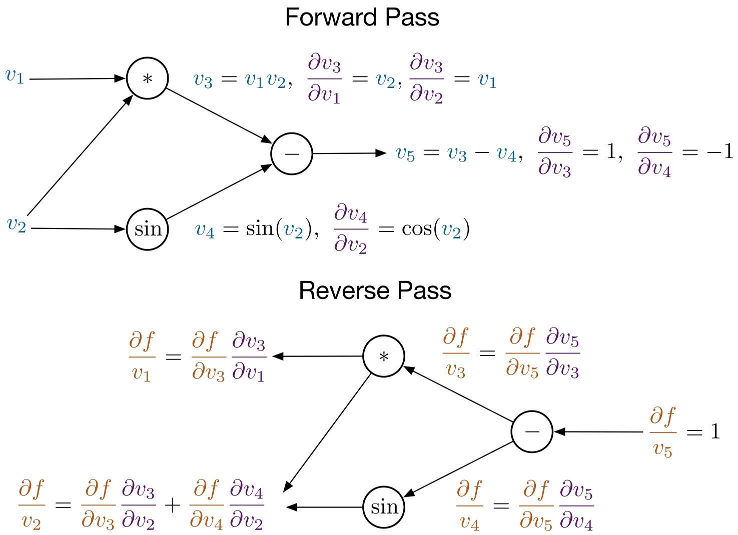
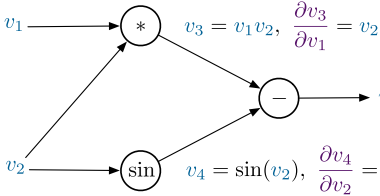


Figure: Van de Meent et al. (2021)

Reverse mode, step-by-step

Forward Pass

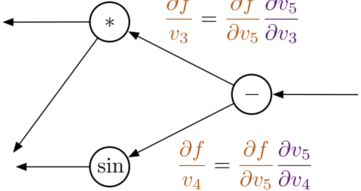


$$v_3 = v_1 v_2,\ \frac{\partial v_3}{\partial v_1} = v_2,\ \frac{\partial v_3}{\partial v_2} = v_1$$

$$v_5 = v_3 - v_4,\ \frac{\partial v_5}{\partial v_3} = 1,\ \frac{\partial v_5}{\partial v_4} = -1$$

$$v_4 = \sin(v_2),\ \frac{\partial v_4}{\partial v_2} = \cos(v_2)$$

Reverse Pass



$$\frac{\partial f}{v_1} = \frac{\partial f}{\partial v_3}\frac{\partial v_3}{\partial v_1}$$

$$\frac{\partial f}{v_3} = \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_3}$$

$$\frac{\partial f}{v_5} = 1$$

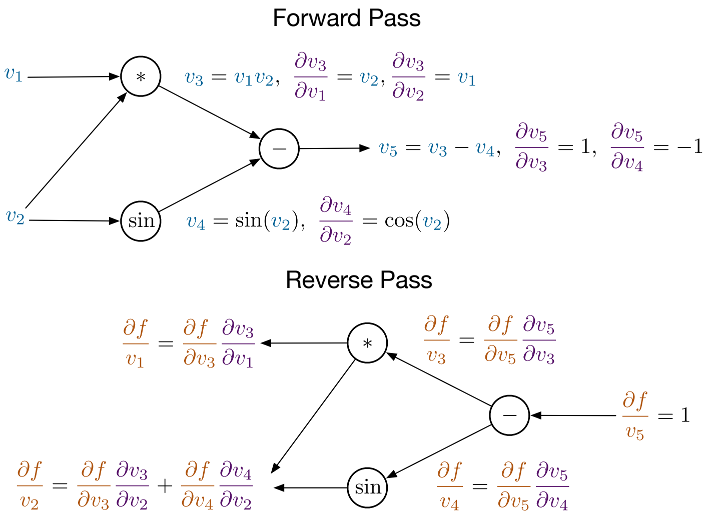
$$\frac{\partial f}{v_2} = \frac{\partial f}{\partial v_3}\frac{\partial v_3}{\partial v_2} + \frac{\partial f}{\partial v_4}\frac{\partial v_4}{\partial v_2}$$

$$\frac{\partial f}{v_4} = \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_4}$$

$$\frac{\partial f}{\partial v_5} = 1$$

Figure: Van de Meent et al. (2021)

Reverse mode, step-by-step



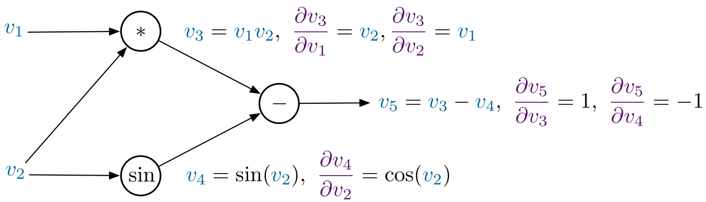
$$\frac{\partial f}{\partial v_5} = 1$$

$$\frac{\partial f}{\partial v_4} = (1)(-1)$$

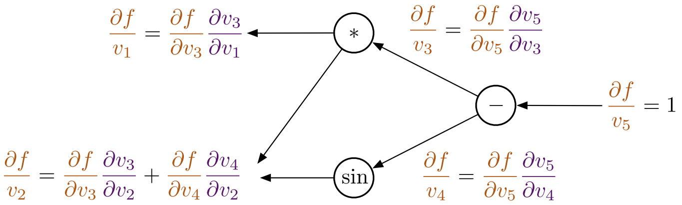
Figure: Van de Meent et al. (2021)

Reverse mode, step-by-step

Forward Pass



Reverse Pass



$$\frac{\partial f}{\partial v_5} = 1$$

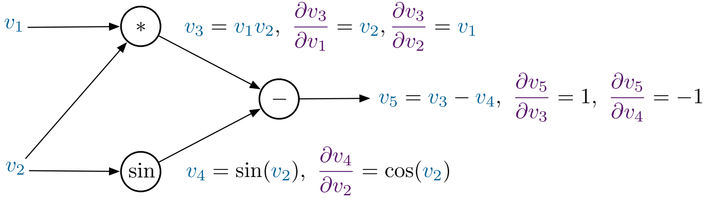
$$\frac{\partial f}{\partial v_4} = (1)(-1)$$

$$\frac{\partial f}{\partial v_3} = (1)(1)$$

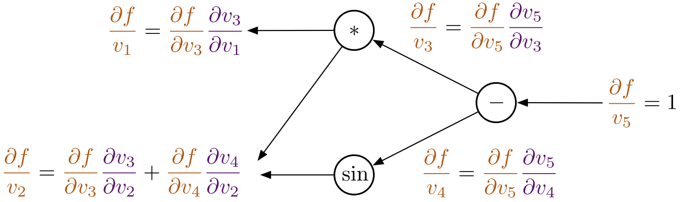
Figure: Van de Meent et al. (2021)

Reverse mode, step-by-step

Forward Pass



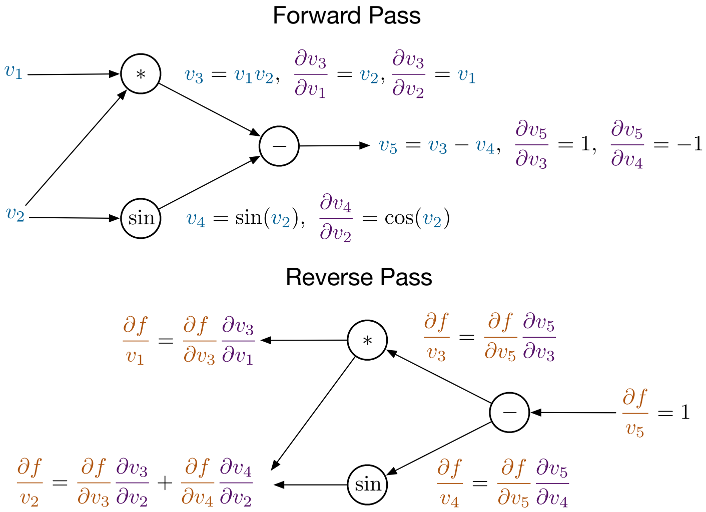
Reverse Pass



$$\frac{\partial f}{\partial v_5} = 1$$
$$\frac{\partial f}{\partial v_4} = (1)(-1)$$
$$\frac{\partial f}{\partial v_3} = (1)(1)$$
$$\frac{\partial f}{\partial v_2} = (1)(v_1) + (-1)(\cos(v_2))$$

Figure: Van de Meent et al. (2021)

Reverse mode, step-by-step



$$\frac{\partial f}{\partial v_5} = 1$$

$$\frac{\partial f}{\partial v_4} = (1)(-1)$$

$$\frac{\partial f}{\partial v_3} = (1)(1)$$

$$\frac{\partial f}{\partial v_2} = (1)(v_1) + (-1)(\cos(v_2))$$

$$\frac{\partial f}{\partial v_1} = (1)(v_2)$$

Figure: Van de Meent et al. (2021)

Reverse mode overview

1. When running the forward computation, construct a "computation graph" where each node
    - stores a pointer to its parent nodes
    - computes the partial derivatives w.r.t. each input

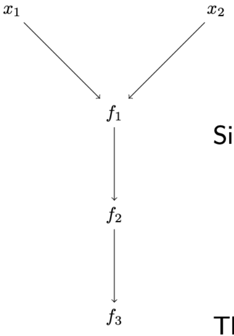
Reverse mode overview

1. When running the forward computation, construct a “computation graph” where each node
   - stores a pointer to its parent nodes
   - computes the partial derivatives w.r.t. each input
2. Eventually, the forward computation produces a scalar output. This is the “root” node of the computation graph.

Reverse mode overview

1. When running the forward computation, construct a "computation graph" where each node
   - stores a pointer to its parent nodes
   - computes the partial derivatives w.r.t. each input
2. Eventually, the forward computation produces a scalar output. This is the "root" node of the computation graph.
3. Run a backward computation, rolling backward through the computation graph in reverse order;
   - at each intermediate node, accumulate the "incoming" partial derivatives from its parents
   - at each leaf node, report the entry of the gradient

Reverse mode prevents duplicate computation



$$\frac{df_3}{dx_1} = \frac{\partial f_3}{\partial f_2}\frac{df_2}{dx_1} = \underbrace{\frac{\partial f_3}{\partial f_2}\frac{df_2}{df_1}}_{\frac{df_3}{df_1}}\frac{df_1}{dx_1}$$

Similarly,

$$\frac{df_3}{dx_2} = \underbrace{\frac{\partial f_3}{\partial f_2}\frac{df_2}{df_1}}_{\frac{df_3}{df_1}}\frac{df_1}{dx_2}$$

The two derivatives share the same computation branch.

Figure: David Barber

Other thoughts

- Reverse mode autodiff computes gradients in a "backwards pass" that has the same order runtime as the forward computation (albeit with increased memory costs)
- In general, differentiating through the graph a second time to compute Hessians is expensive, but there are tricks to compute Hessian-vector products $\mathbf{Hv}$ much more efficiently than instantiating the entire Hessian
- AD packages exist for many programming languages — mostly, they just require overloading operators, and being careful to avoid in-place operations
- We'll be using Pytorch for the coursework, which is essentially an implementation of reverse-mode AD; I'll also upload a Jupyter notebook with some demos