
2020
Swamps of Numerical Integration
Monte Carlo Heights
Normalizing Flow
Unrolling Hills of Time


2020
Swamps of Numerical Integration
Monte Carlo Heights
Normalizing Flow
Unrolling Hills of Time
Backprop Bay

Backpropagation and automatic differentiation

Cheng Soon Ong
Marc Peter Deisenroth

December 2020

Gradients in machine learning

- In machine learning, we use gradients to train
- Training $=$ optimize objective function w.r.t. model parameters
- Examples: curve fitting, neural networks, mixture models
- For functions $f(\boldsymbol{x})$ we want its gradient $\nabla f(\boldsymbol{x})$

Motivation

How do we efficiently calculate a gradient?
For example, the gradient of

$$\frac{\mathsf{exp}(x)}{x^2}$$

looks a lot more complex

$$\frac{\mathsf{exp}(x)(x-2)}{x^3}$$

(example from Wikipedia)

Motivation

Cache intermediate results

However, it turns out that we can reduce computational cost of computing the gradient if we cache intermediate results.

Trade off computational complexity for space complexity.

Why learn about backpropagation?

- Composition of functions = multiplication of gradients
- Automatic differentiation is implemented in modern machine learning tools
- Learn concepts of calculation of gradients
- Goal: show links to results of calculus and optimization

Why learn about backpropagation?

- Composition of functions = multiplication of gradients
- Automatic differentiation is implemented in modern machine learning tools
- Learn concepts of calculation of gradients
- Goal: show links to results of calculus and optimization

Backpropagation is just ...

the chain rule of differentiation

Recall: Chain rule

Scalar chain rule

$$\big(g(f(x))\big)' = (g \circ f)'(x) = g'(f(x))f'(x)$$

where $g \circ f$ is a function composition $x \mapsto f(x) \mapsto g(f(x))$.

Vector chain rule

$$\frac{\partial}{\partial \boldsymbol{x}}(g \circ f)(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}\big(g(f(\boldsymbol{x}))\big) = \frac{\partial g}{\partial f}\frac{\partial f}{\partial \boldsymbol{x}}$$

where we define the gradient as a row vector

$$\nabla_{\boldsymbol{x}} f = \mathrm{grad} f = \frac{df}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} & \frac{\partial f(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{1\times n}.$$

Why row vector for gradients?

Representing a gradient as a row vector allows us to think of the chain rule as matrix multiplication.

If $f(x_1, x_2)$ is a function of x_1 and x_2, where $x_1(s,t)$ and $x_2(s,t)$ are themselves functions of two variables s and t, the chain rule shows that the gradient is obtained by the matrix multiplication

$$\frac{df}{d(s,t)} = \frac{\partial f}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial (s,t)} = \underbrace{\begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix}}_{=\frac{\partial f}{\partial \boldsymbol{x}}} \underbrace{\begin{bmatrix} \frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t} \end{bmatrix}}_{=\frac{\partial \boldsymbol{x}}{\partial (s,t)}} .$$

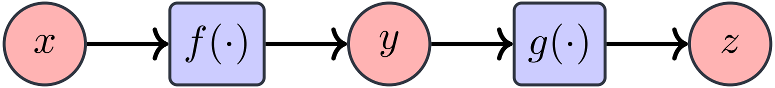
Careful with notation

Define the variable y as the output of $f(x)$ and the variable z as the output of $g(y)$, then we can write in Leibniz notation,

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}\,.$$

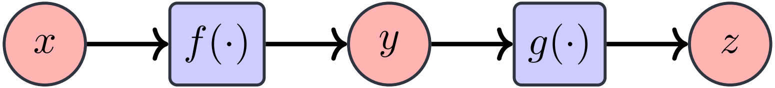
More precisely

$$\left.\frac{dz}{dx}\right|_{x} = \left.\frac{dz}{dy}\right|_{y(x)} \left.\frac{dy}{dx}\right|_{x}\,.$$



Automatic differentiation as cached chain rule

Consider a function $G(x) := g(f(x))$, with intermediate variable y and final output z.



- $y = f(x)$ and $z = g(y)$
- Think of y as a "cache" of the results of computing $f(x)$
- When computing the gradient (going right to left), there is another "cache".

Automatic differentiation as cached chain rule

Given a function $G(w) := g(f(e(w)))$, with intermediate variables x, y and final output z.

$$w \longrightarrow \boxed{e(\cdot)} \longrightarrow x \longrightarrow \boxed{f(\cdot)} \longrightarrow y \longrightarrow \boxed{g(\cdot)} \longrightarrow z$$

We can cache the chain rule from the left (forward) or the right (reverse).

$$\frac{dz}{dw} = \frac{dz}{dy}\left(\frac{dy}{dx}\frac{dx}{dw}\right) \qquad \text{(forward mode)}$$

$$\frac{dz}{dw} = \left(\frac{dz}{dy}\frac{dy}{dx}\right)\frac{dx}{dw} \qquad \text{(reverse mode)}$$

Origin story...

Who invented backprop?

`http://people.idsia.ch/~juergen/who-invented-backpropagation.html`

Paul Werbos, PhD thesis 1974:

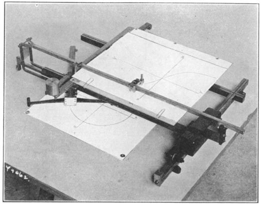
Theorem:

$$\frac{\partial F_j}{\partial x_i} = \sum_{k=j+1}^{n} \frac{\partial^+ x_n}{\partial x_k} \cdot \frac{\partial f_k}{\partial x_i} \quad \text{for} \quad \begin{array}{l} i_0 \leq j < n \\ 1 \leq i \leq j \end{array}$$

Pointers to literature

- History (Wengert, 1964; Werbos, 1975)
- Computer implementation of automatic differentiation (Speelpenning, 1980)
- Automatic differentiation survey (Baydin et al., 2018)
- Introduction to automatic differentiation (Griewank and Walther, 2003, 2008)
- Mathematical introduction for forward mode (Hoffmann, 2016)
- Hessian vector products (Pearlmutter, 1994; Schraudolph, 2002)
- `https://autodiff-workshop.github.io/`

Armin Elmendorf, 1918



Think of a pair of numbers

Intuition

Associate every variable a with its derivative with respect to an output value.
Think of the derivative as a "function".

- Automatic differentiation augments each variable (for example a) with an **adjoint** variable $\overleftarrow{a}$ to form an adjoint pair $(a, \overleftarrow{a})$.
- The adjoint $\overleftarrow{a}$ of a is the partial differential operator $\frac{\partial}{\partial a}$.
- The pair $(a, \overleftarrow{a})$ is called a **dual number**.

Automatic differentiation

Two modes of automatic differentiation, for a function $f : \mathbb{R}^D \to \mathbb{R}^M$

forward mode (efficient when $D \ll M$)

Dual number $(a, \overleftarrow{a})$ can be represented as a matrix $\begin{bmatrix} a & \overleftarrow{a} \\ 0 & a \end{bmatrix}$

Automatic differentiation

Two modes of automatic differentiation, for a function $f : \mathbb{R}^D \to \mathbb{R}^M$

forward mode (efficient when $D \ll M$)

Dual number $(a, \overleftarrow{a})$ can be represented as a matrix $\begin{bmatrix} a & \overleftarrow{a} \\ 0 & a \end{bmatrix}$

reverse mode (efficient when $D \gg M$)

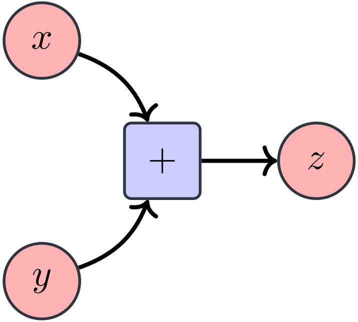
Consists of two passes, going from inputs to outputs and outputs to inputs.

Forward mode: Simple examples

sum

The gradient of a sum is a sum of the gradients

$$(x, \overleftarrow{x}) + (y, \overleftarrow{y}) = \begin{bmatrix} x & \overleftarrow{x} \\ 0 & x \end{bmatrix} + \begin{bmatrix} y & \overleftarrow{y} \\ 0 & y \end{bmatrix}$$
$$= \begin{bmatrix} x+y & \overleftarrow{x} + \overleftarrow{y} \\ 0 & x+y \end{bmatrix}$$
$$= (x+y, \overleftarrow{x} + \overleftarrow{y})$$

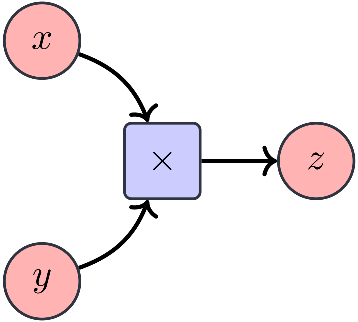


Forward mode: Simple examples

product

The gradient is given by the product rule of calculus

$$
\begin{aligned}
(x, \overleftarrow{x}) \times (y, \overleftarrow{y}) &= \begin{bmatrix} x & \overleftarrow{x} \\ 0 & x \end{bmatrix} \times \begin{bmatrix} y & \overleftarrow{y} \\ 0 & y \end{bmatrix} \\
&= \begin{bmatrix} xy & x\overleftarrow{y} + y\overleftarrow{x} \\ 0 & xy \end{bmatrix} \\
&= (xy, x\overleftarrow{y} + y\overleftarrow{x})
\end{aligned}
$$



Reverse mode autodiff

Also known as backpropagation, has two phases:

Forward pass

- Calculate the forward pass for evaluating the function
- At the same time cache all the partial differentials

Reverse pass

- Set the adjoint (gradient) of the output node to 1
- Increase the input adjoint by the product of the output adjoint with the forward partials

Key Challenge

Need the computations to be done in topological order

Reverse mode automatic differentiation

Given a function $G(w) := g(f(e(w)))$, with intermediate variables x, y and final output z.

$$w \longrightarrow \boxed{e(\cdot)} \longrightarrow x \longrightarrow \boxed{f(\cdot)} \longrightarrow y \longrightarrow \boxed{g(\cdot)} \longrightarrow z$$

Input adjoint $=$ output adjoint $\times$ forward partials

$$\underbrace{\frac{dz}{dw}}_{\overleftarrow{w}} = \underbrace{\left(\frac{dz}{dy}\frac{dy}{dx}\right)}_{\overleftarrow{x}} \frac{dx}{dw} \qquad \text{(reverse mode)}$$

What to do with branches

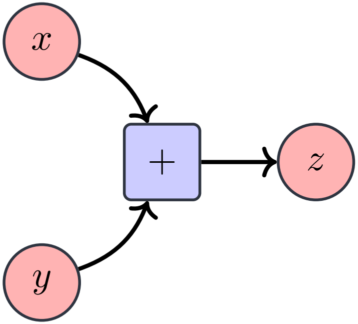
Consider the case of a sum

$$z = x + y\,,$$

has derivative

$$\frac{dz}{dx} = 1 \quad \text{and} \quad \frac{dz}{dy} = 1\,.$$

$$\overleftarrow{x} = \overleftarrow{z} \times \frac{dz}{dx} \quad \text{and} \quad \overleftarrow{y} = \overleftarrow{z} \times \frac{dz}{dy}$$



Reverse mode: Simple examples

sum

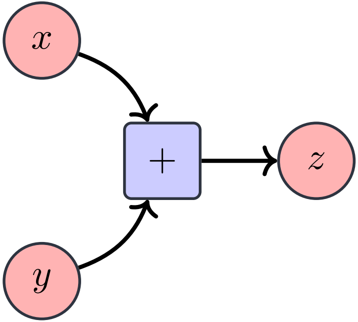
An add gate is a gradient distributor

Gradient w.r.t. x,

$$\overleftarrow{x} \mathrel{+}= \overleftarrow{z}$$

Gradient w.r.t. y

$$\overleftarrow{y} \mathrel{+}= \overleftarrow{z}$$



Reverse mode: Simple examples

product

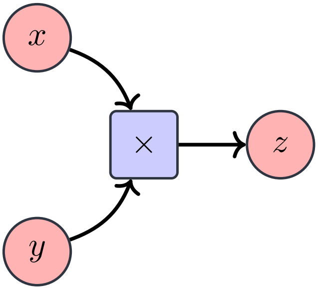
A multiplication gate is a gradient switcher

Gradient w.r.t. x,

$$\overleftarrow{x} \mathrel{+}= \overleftarrow{z}\, y$$

Gradient w.r.t. y

$$\overleftarrow{y} \mathrel{+}= x\, \overleftarrow{z}$$



Summary

- Modern machine learning powered by automatic differentiation libraries
- Forward mode autodiff: matrix multiplication
- Reverse mode autodiff: input adjoint $=$ output adjoint $\times$ partial derivative
- How to efficiently combine forward and reverse mode autodiff is still an open question
- Think of the pair of variable and its adjoint $(x, \overleftarrow{x})$

Backpropagation is just ...

chain rule of differentiation with caching

References

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). Automatic differentiationin machine learning: a survey. *Journal of Machine Learning Research*, 18:1–43.

Deisenroth, M. P., Faisal, A. A., and Ong, C. S. (2020). *Mathematics for Machine Learning*. Cambridge University Press.

Griewank, A. and Walther, A. (2003). Introduction to automatic differentiation. *Proceedings in Applied Mathematics and Mechanics*, 2(1):45–49.

Griewank, A. and Walther, A. (2008). *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation, second edition*. SIAM, Philadelphia.

Hoffmann, P. H. (2016). A hitchhikers guide to automatic differentiation. *Numerical Algorithms*, 72(3):775–811.

Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160.

References (cont.)

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738.

Speelpenning, B. (1980). Compiling fast partial derivatives of functions given by algorithms.

Wengert, R. E. (1964). A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8).

Werbos, P. J. (1975). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis.