There and Back Again: A Tale of Expectations and Slopes. A Neurips 2020 Tutorial by Marc Deisenroth...
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 $f(x)$ we want its gradient $\nabla f(x)$
Motivation

How do we efficiently calculate a gradient?

For example, the gradient of

\[
\frac{\exp(x)}{x^2}
\]

looks a lot more complex

\[
\frac{\exp(x)(x - 2)}{x^3}
\]

(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

\[ (g(f(x)))' = (g \circ f)'(x) = g'(f(x))f'(x) \]

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

Vector chain rule

\[ \frac{\partial}{\partial x} (g \circ f)(x) = \frac{\partial}{\partial x} (g(f(x))) = \frac{\partial g}{\partial f} \frac{\partial f}{\partial x} \]

where we define the gradient as a row vector

\[ \nabla_x f = \text{grad} f = \frac{df}{dx} = [\frac{\partial f(x)}{\partial x_1} \ \frac{\partial f(x)}{\partial x_2} \ \ldots \ \frac{\partial f(x)}{\partial x_n}] \in \mathbb{R}^{1 \times n} \]
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:

\[
\frac{df}{d(s, t)} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial (s, t)} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \\ \frac{\partial x_1}{\partial s} & \frac{\partial x_2}{\partial s} \\ \frac{\partial x_1}{\partial t} & \frac{\partial x_2}{\partial t} \end{bmatrix} \cdot 
\]

\[
= \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial x}{\partial (s, t)} \end{bmatrix}.
\]
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

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

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

\[
\frac{dz}{dw} = \frac{dz}{dy} \left( \frac{dy}{dx} \frac{dx}{dw} \right) \quad \text{(forward mode)}
\]

\[
\frac{dz}{dw} = \left( \frac{dz}{dy} \frac{dy}{dx} \right) \frac{dx}{dw} \quad \text{(reverse mode)}
\]
Origin story...

Who invented backprop?
http://people.idsia.ch/~juergen/who-invented-backpropagation.html

Paul Werbos, PhD thesis 1974:

\[
\frac{\partial F_j}{\partial x_1} = \sum_{k=j+1}^{n} \frac{\partial^+ x_n}{\partial x_k} \cdot \frac{\partial f_k}{\partial x_1} \quad \text{for} \quad 1 \leq j < n \\
1 \leq i \leq j
\]
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 $\overset{\leftarrow}{a}$ to form an adjoint pair $(a, \overset{\leftarrow}{a})$.
- The adjoint $\overset{\leftarrow}{a}$ of $a$ is the partial differential operator $\frac{\partial}{\partial a}$.
- The pair $(a, \overset{\leftarrow}{a})$ 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 $(a, \overrightarrow{a})$ can be represented as a matrix $\begin{bmatrix} a & \overrightarrow{a} \\ 0 & a \end{bmatrix}$
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 \((a, \overrightarrow{a})\) can be represented as a matrix \[
\begin{bmatrix}
a & \overrightarrow{a} \\
0 & a
\end{bmatrix}
\]

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

Consists of two passes, going from inputs to outputs and outputs to inputs.
Forward mode: Simple examples

The gradient of a sum is a sum of the gradients

\[(x, \frac{\partial}{\partial x}) + (y, \frac{\partial}{\partial y}) = \begin{bmatrix} x & \frac{\partial}{\partial x} \\ 0 & x \end{bmatrix} + \begin{bmatrix} y & \frac{\partial}{\partial y} \\ 0 & y \end{bmatrix} = \begin{bmatrix} x + y & \frac{\partial}{\partial x} + \frac{\partial}{\partial y} \\ 0 & x + y \end{bmatrix} = (x + y, \frac{\partial}{\partial x} + \frac{\partial}{\partial y})\]
Forward mode: Simple examples

The gradient is given by the product rule of calculus

\[
(x, \overset{\leftarrow}{x}) \times (y, \overset{\leftarrow}{y}) = \begin{bmatrix} x & \overset{\leftarrow}{x} \\ 0 & x \end{bmatrix} \times \begin{bmatrix} y & \overset{\leftarrow}{y} \\ 0 & y \end{bmatrix} \\
= \begin{bmatrix} xy & x \overset{\leftarrow}{y} + y \overset{\leftarrow}{x} \\ 0 & xy \end{bmatrix} \\
= (xy, x \overset{\leftarrow}{y} + y \overset{\leftarrow}{x})
\]
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 $z$.

$$\frac{dz}{dw} = \left( \frac{dz}{dy} \right) \left( \frac{dy}{dx} \right) \frac{dx}{dw}$$

Input adjoint = output adjoint $\times$ forward partials

(reverse mode)
Consider the case of a sum

\[ z = x + y , \]

has derivative

\[ \frac{dz}{dx} = 1 \quad \text{and} \quad \frac{dz}{dy} = 1 . \]

\[ \frac{\delta x}{\delta z} = \frac{dz}{dx} \quad \text{and} \quad \frac{\delta y}{\delta z} = \frac{dz}{dy} , \]
Reverse mode: Simple examples

**sum**

An add gate is a gradient distributor

Gradient w.r.t. $x$, 

\[
\frac{\partial z}{\partial x} + \frac{\partial z}{\partial y}
\]

Gradient w.r.t. $y$, 

\[
\frac{\partial z}{\partial y} + \frac{\partial z}{\partial x}
\]
A multiplication gate is a gradient switcher

Gradient w.r.t. $x$,

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

Gradient w.r.t. $y$

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


