Mathematics for Machine Learning

— Vector Calculus: Backpropagation & Automatic Differentiation

Joseph Chuang-Chieh Lin

Department of Computer Science & Engineering,
National Taiwan Ocean University

Spring 2025

Credits for the resource

- The slides are based on the textbooks:
  - *Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong: Mathematics for Machine Learning. Cambridge University Press. 2020.*
  - *Howard Anton, Chris Rorres, Anton Kaul: Elementary Linear Algebra. Wiley. 2019.*
- We could partially refer to the monograph:
  Francesco Orabona: A Modern Introduction to Online Learning. https://arxiv.org/abs/1912.13213

Outline

Outline

Motivation

Consider the function

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2)).$$

Motivation

Consider the function

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2)).$$

Apply the chain rule, we can compute its gradient:

Motivation

Consider the function

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2)).$$

Apply the chain rule, we can compute its gradient:

$$\begin{aligned}
\frac{\mathrm{d}f}{\mathrm{d}x} &= \frac{2x + 2x\exp(x^2)}{2\sqrt{x^2 + \exp(x^2)}} - \sin(x^2 + \exp(x^2))(2x + 2x\exp(x^2)) \\
&= 2x\left(\frac{1}{2\sqrt{x^2 + \exp(x^2)}} - \sin(x^2 + \exp(x^2))\right)(1 + \exp(x^2)).
\end{aligned}$$

Motivation

Consider the function

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2)).$$

Apply the chain rule, we can compute its gradient:

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}x} &= \frac{2x + 2x\exp(x^2)}{2\sqrt{x^2 + \exp(x^2)}} - \sin(x^2 + \exp(x^2))(2x + 2x\exp(x^2)) \\ &= 2x\left(\frac{1}{2\sqrt{x^2 + \exp(x^2)}} - \sin(x^2 + \exp(x^2))\right)(1 + \exp(x^2)).\end{aligned}$$

- Impractical to write it explicitly.
- The implementation of the gradient could be expensive.

Gradients in a Deep Network

$$\mathbf{y} = (f_k \circ f_{k-1} \circ \cdots \circ f_1)(\mathbf{x}) = f_k(f_{k-1}(\cdots(f_1(\mathbf{x}))\cdots)).$$

- $\mathbf{x}$: inputs (e.g., images).
- $\mathbf{y}$: observations (e.g., class labels).
- f_i, $i = 1, \ldots, K$: functions with their own parameters.

Gradients in a Deep Network

$$\mathbf{y} = (f_k \circ f_{k-1} \circ \cdots \circ f_1)(\mathbf{x}) = f_k(f_{k-1}(\cdots(f_1(\mathbf{x}))\cdots)).$$

- $\mathbf{x}$: inputs (e.g., images).
- $\mathbf{y}$: observations (e.g., class labels).
- f_i, $i = 1, \ldots, K$: functions with their own parameters.
  - $f_i(\mathbf{x}_{i-1}) = \sigma(\boldsymbol{A}_{i-1}\mathbf{x}_{i-1} + \mathbf{b}_{i-1})$, in the ith layer.
    - $\mathbf{x}_{i-1}$: the output of layer $i-1$.
    - σ: activation function (e.g., $1/(1+e^{-x})$, $\tanh(\mathbf{x})$, rectified linear unit (ReLU), etc.).
  - $\mathbf{f}_0 := \mathbf{x}$;
    $\mathbf{f}_i := \sigma_i(\boldsymbol{A}_{i-1}\mathbf{f}_{i-1} + \mathbf{b}_{i-1})$, $i = 1, \ldots, K$.

Get the Gradients

- To obtain the gradients w.r.t. the parameter set $\boldsymbol{\theta}$:
  - $\boldsymbol{\theta} = \{\boldsymbol{A}_0, \mathbf{b}_0, \ldots, \boldsymbol{A}_{k-1}, \mathbf{b}_{K-1}\}$.
  - The squared loss: $L(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{f}_K(\boldsymbol{\theta}, \mathbf{x})\|^2$.
  - $\boldsymbol{\theta}_j = \{\boldsymbol{A}_j, \mathbf{b}_j\}$, for $j = 0, 1, \ldots, K-1$.

Get the Gradients

- To obtain the gradients w.r.t. the parameter set $\boldsymbol{\theta}$:
  - $\boldsymbol{\theta} = \{\boldsymbol{A}_0, \mathbf{b}_0, \ldots, \boldsymbol{A}_{k-1}, \mathbf{b}_{K-1}\}$.
  - The squared loss: $L(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{f}_K(\boldsymbol{\theta}, \mathbf{x})\|^2$.
  - $\boldsymbol{\theta}_j = \{\boldsymbol{A}_j, \mathbf{b}_j\}$, for $j = 0, 1, \ldots, K-1$.
- By the chain rule:

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \boldsymbol{\theta}_{\boldsymbol{K-1}}}$$

Get the Gradients

- To obtain the gradients w.r.t. the parameter set $\boldsymbol{\theta}$:
  - $\boldsymbol{\theta} = \{\boldsymbol{A}_0, \mathbf{b}_0, \ldots, \boldsymbol{A}_{k-1}, \mathbf{b}_{K-1}\}$.
  - The squared loss: $L(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{f}_K(\boldsymbol{\theta}, \mathbf{x})\|^2$.
  - $\boldsymbol{\theta}_j = \{\boldsymbol{A}_j, \mathbf{b}_j\}$, for $j = 0, 1, \ldots, K-1$.
- By the chain rule:

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \boldsymbol{\theta}_{\boldsymbol{K-1}}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-2}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \boldsymbol{\theta}_{\boldsymbol{K-2}}}$$

Get the Gradients

- To obtain the gradients w.r.t. the parameter set $\boldsymbol{\theta}$:
  - $\boldsymbol{\theta} = \{\boldsymbol{A}_0, \mathbf{b}_0, \ldots, \boldsymbol{A}_{k-1}, \mathbf{b}_{K-1}\}$.
  - The squared loss: $L(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{f}_K(\boldsymbol{\theta}, \mathbf{x})\|^2$.
  - $\boldsymbol{\theta}_j = \{\boldsymbol{A}_j, \mathbf{b}_j\}$, for $j = 0, 1, \ldots, K-1$.
- By the chain rule:

$$
\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} &= \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \boldsymbol{\theta}_{\boldsymbol{K-1}}} \\
\frac{\partial L}{\partial \boldsymbol{\theta}_{K-2}} &= \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \boldsymbol{\theta}_{\boldsymbol{K-2}}} \\
\frac{\partial L}{\partial \boldsymbol{\theta}_{K-3}} &= \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \mathbf{f}_{K-2}} \frac{\partial \mathbf{f}_{K-2}}{\partial \boldsymbol{\theta}_{\boldsymbol{K-3}}} \\
&\vdots
\end{aligned}
$$

Get the Gradients

- To obtain the gradients w.r.t. the parameter set $\boldsymbol{\theta}$:
  - $\boldsymbol{\theta} = \{\boldsymbol{A}_0, \mathbf{b}_0, \ldots, \boldsymbol{A}_{k-1}, \mathbf{b}_{K-1}\}$.
  - The squared loss: $L(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{f}_K(\boldsymbol{\theta}, \mathbf{x})\|^2$.
  - $\boldsymbol{\theta}_j = \{\boldsymbol{A}_j, \mathbf{b}_j\}$, for $j = 0, 1, \ldots, K-1$.
- By the chain rule:

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} &= \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \boldsymbol{\theta}_{\boldsymbol{K-1}}} \\
\frac{\partial L}{\partial \boldsymbol{\theta}_{K-2}} &= \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \boldsymbol{\theta}_{\boldsymbol{K-2}}} \\
\frac{\partial L}{\partial \boldsymbol{\theta}_{K-3}} &= \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \mathbf{f}_{K-2}} \frac{\partial \mathbf{f}_{K-2}}{\partial \boldsymbol{\theta}_{\boldsymbol{K-3}}} \\
&\vdots
\end{aligned}$$

- Partial derivatives of the output of a layer w.r.t. (1) its inputs or (2) its parameters.

What have we learnt?

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{i+1}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \cdots \frac{\partial \mathbf{f}_{i+3}}{\partial \mathbf{f}_{i+2}} \frac{\partial \mathbf{f}_{i+2}}{\partial \boldsymbol{\theta}_{i+1}}$$

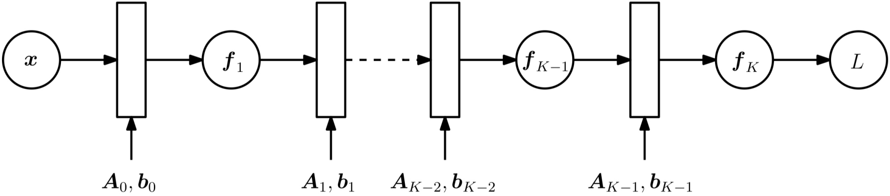
$$\frac{\partial L}{\partial \boldsymbol{\theta}_{i}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \cdots \frac{\partial \mathbf{f}_{i+3}}{\partial \mathbf{f}_{i+2}} \underbrace{\frac{\partial \mathbf{f}_{i+2}}{\partial \mathbf{f}_{i+1}} \frac{\partial \mathbf{f}_{i+1}}{\partial \boldsymbol{\theta}_{i}}}_{\text{not reused yet}}$$

What have we learnt?

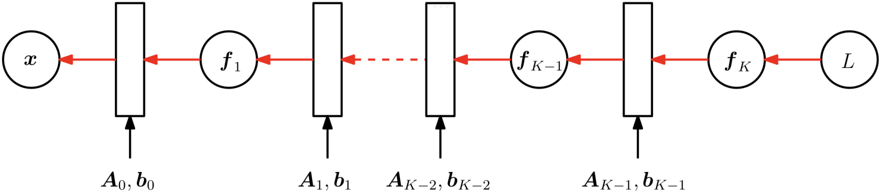
$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\theta}_{i+1}} &= \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \cdots \frac{\partial \mathbf{f}_{i+3}}{\partial \mathbf{f}_{i+2}} \frac{\partial \mathbf{f}_{i+2}}{\partial \boldsymbol{\theta}_{\boldsymbol{i+1}}} \\
\frac{\partial L}{\partial \boldsymbol{\theta}_{i}} &= \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \cdots \frac{\partial \mathbf{f}_{i+3}}{\partial \mathbf{f}_{i+2}} \underbrace{\frac{\partial \mathbf{f}_{i+2}}{\partial \mathbf{f}_{i+1}} \frac{\partial \mathbf{f}_{i+1}}{\partial \boldsymbol{\theta}_{\boldsymbol{i}}}}_{\text{not reused yet}}
\end{aligned}$$

Hence the name *backpropagation*.

Forward Pass



Backward Pass



Outline

Automatic Differentiation

- A set of techniques to *numerically* evaluate the "exact" (up to machine precision) gradient of a function.
  - By intermediate variables & chain rule.
- Complicated functions can be computed automatically(?).

Forward Mode & Reverse Mode

- Input: x; Output: y; Intermediate variables a, b.

$$x \to a \to b \to y$$

Reverse Mode:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \left(\frac{\mathrm{d}y}{\mathrm{d}b}\frac{\mathrm{d}b}{\mathrm{d}a}\right)\frac{\mathrm{d}a}{\mathrm{d}x}$$

Forward Mode:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}b}\left(\frac{\mathrm{d}b}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}x}\right)$$

Example

Example (Reverse Mode)

Consider the function

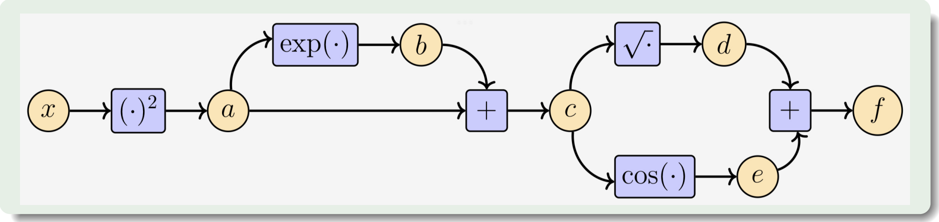
$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2)).$$

Introducing intermediate variables:

$$\begin{aligned} a &= x^2 \\ b &= \exp(a) \\ c &= a + b \\ d &= \sqrt{c} \\ e &= \cos(c) \\ f &= d + e. \end{aligned}$$

Example

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2)).$$



$$
\begin{aligned}
a &= x^2 \\
b &= \exp(a) \\
c &= a + b \\
d &= \sqrt{c} \\
e &= \cos(c) \\
f &= d + e.
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial a}{\partial x} &= 2x \\
\frac{\partial b}{\partial a} &= \exp(a) \\
\frac{\partial c}{\partial a} &= 1 = \frac{\partial c}{\partial b} \\
\frac{\partial d}{\partial c} &= \frac{1}{\sqrt{c}} \\
\frac{\partial e}{\partial c} &= -\sin(c) \\
\frac{\partial f}{\partial d} &= 1 = \frac{\partial f}{\partial e}.
\end{aligned}
$$

Compute $\frac{\partial f}{\partial x}$

$$
\begin{aligned}
\frac{\partial f}{\partial c} &= \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c} \\
\frac{\partial f}{\partial b} &= \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} \\
\frac{\partial f}{\partial a} &= \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} \\
\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial a}\frac{\partial a}{\partial x}
\end{aligned}
$$

Compute $\frac{\partial f}{\partial x}$

$$
\begin{aligned}
\frac{\partial f}{\partial c} &= \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c} = 1 \cdot \frac{1}{2\sqrt{v}} + 1 \cdot (-\sin(c)) \\
\frac{\partial f}{\partial b} &= \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} = \frac{\partial f}{\partial c} \cdot 1 \\
\frac{\partial f}{\partial a} &= \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} = \frac{\partial f}{\partial b}\exp(a) + \frac{\partial f}{\partial c} \cdot 1 \\
\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} = \frac{\partial f}{\partial a} \cdot 2x.
\end{aligned}
$$

Automatic Differentiation in General

Automatic Differentiation in General

- $x_1, \ldots, x_d$: input variables
- $x_{d+1}, \ldots, x_{D-1}$: intermediate variables
- x_D: the output variable

The computation graph can be expressed as

$$\text{For } i = d+1, \ldots, D: \quad x_i = g_i(x_{\mathrm{Pa}(x_i)}),$$

where $g_i(\cdot)$ are (elementary) functions, $x_{\mathrm{Pa}(x_i)}$ are parent nodes of x_i.

Automatic Differentiation in General

Automatic Differentiation in General

- $x_1, \dots, x_d$: input variables
- $x_{d+1}, \dots, x_{D-1}$: intermediate variables
- x_D: the output variable

The computation graph can be expressed as

$$\text{For } i = d+1, \dots, D: \quad x_i = g_i(x_{\mathrm{Pa}(x_i)}), \quad \text{(forward propagation)}$$

where $g_i(\cdot)$ are (elementary) functions, $x_{\mathrm{Pa}(x_i)}$ are parent nodes of x_i.

$f = x_D \implies \frac{\partial f}{\partial x_D} = 1.$

Automatic Differentiation in General

Automatic Differentiation in General

- $x_1, \ldots, x_d$: input variables
- $x_{d+1}, \ldots, x_{D-1}$: intermediate variables
- x_D: the output variable

The computation graph can be expressed as

$$\text{For } i = d+1, \ldots, D: \quad x_i = g_i(x_{\mathrm{Pa}(x_i)}), \quad \text{(forward propagation)}$$

where $g_i(\cdot)$ are (elementary) functions, $x_{\mathrm{Pa}(x_i)}$ are parent nodes of x_i.

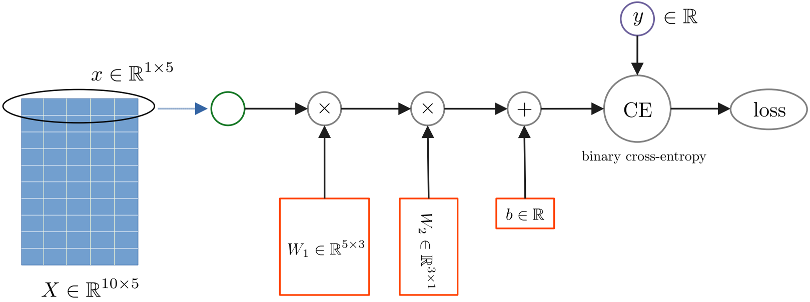
$f = x_D \implies \frac{\partial f}{\partial x_D} = 1$.

$$\frac{\partial f}{\partial x_i} = \sum_{x_j : x_i \in \mathrm{Pa}(x_j)} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i} = \sum_{x_j : x_i \in \mathrm{Pa}(x_j)} \frac{\partial f}{\partial x_j} \frac{\partial g_j}{\partial x_i}. \text{ (backpropagation)}$$

Automatic Differentiation

- A set of techniques to *numerically* evaluate the “exact” (up to machine precision) gradient of a function.
    - By intermediate variables & chain rule.
- Complicated functions can be computed automatically, whenever it can be expressed as a computation graph and the elementary functions are differentiable.
- Programming structures, such as for loops and if statements, require more care as well.

Example



$$\mathrm{CE} \approx -y \cdot \log \sigma(z) + (1-y) \cdot \log(1-\sigma(z))$$
$$z = \mathbf{x}\mathbf{W}_1\mathbf{W}_2 + b$$
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Example

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Nov  1 09:20:33 2023

@author: jcclin
"""

import torch

x = torch.randn(10, 5)  # input tensor
y = torch.zeros(10, 1)  # expected output
w1 = torch.randn(5, 3, requires_grad=True)
w2 = torch.randn(3, 1, requires_grad=True)
b = torch.randn(1, requires_grad=True)
z = torch.matmul(x, w1)
r = torch.matmul(z, w2) + b

loss = torch.nn.functional.binary_cross_entropy_with_logits(r, y)

print(f"Gradient function for z = {z.grad_fn}")
print(f"Gradient function for loss = {loss.grad_fn}")

loss.backward()
print(w1.grad)
print(w2.grad)
print(b.grad)
```

```
In [33]: runfile('E:/Research/ElectionGame/untitled0.py', wdir='E:/Research/ElectionGame')
Gradient function for z = <MmBackward0 object at 0x000001ECA98E5370>
Gradient function for loss = <BinaryCrossEntropyWithLogitsBackward0 object at 0x000001ECA98E5280>
tensor([[ 0.3135,  0.2991,  0.0587],
        [ 0.3177,  0.3031,  0.0595],
        [ 0.3489,  0.3328,  0.0653],
        [-0.1449, -0.1382, -0.0271],
        [-0.3144, -0.2999, -0.0588]])
tensor([[-0.3156],
        [ 1.8223],
        [ 0.5931]])
tensor([0.8936])

In [34]:
```

Discussions