Roadmap

- Machine learning is about solving an optimization problem whose variables are the parameters of a given model.
- Solving optimization problems requires gradient information.
- Central to this chapter is the concept of the function, which we often write

$$
\begin{aligned}
f : \mathbb{R}^D &\mapsto \mathbb{R} \\
\mathbf{x} &\mapsto f(\mathbf{x})
\end{aligned}
$$

Roadmap

Why Should I Care About Derivatives?

- Training AI models: Every time a neural network learns, it computes derivatives to figure out how to improve
- Gradient descent: The derivative tells the model which direction to "walk" to reduce its errors
- Physics simulations: Derivatives describe velocity, acceleration, and change in games and simulations
- Economics: Marginal cost, marginal revenue — all derivatives!

In one sentence: Derivatives tell you how things change — and AI is all about learning from change.

- Difference Quotient. The average slope of f between x and $x + \partial x$

$$\frac{\partial y}{\partial x} \stackrel{\text{def}}{=} \frac{f(x + \partial x) - f(x)}{\partial x}$$

- Derivative. Pointing in the direction of steepest ascent of f.

$$\frac{df}{dx} \stackrel{\text{def}}{=} \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

- Unless confusion arises, we often use $f' = \frac{df}{dx}$.

Taylor Series

- Representation of a function as an infinite sum of terms, using derivatives of f evaluated at x_0.
- Taylor polynomial. The Taylor polynomial of degree n of $f : \mathbb{R} \mapsto \mathbb{R}$ at x_0 is:

$$T_n(x) \stackrel{\text{def}}{=} \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k, \text{ where } f^{(k)}(x_0) \text{ is the } k\text{th derivative of } f \text{ at } x_0.$$

- Taylor Series. For a smooth function $f \in \{C\}^{\infty}$, the Taylor series of f at x_0 is:

$$T_\infty(x) \stackrel{\text{def}}{=} \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

- If $f(x) = T_\infty(x)$, f is called analytic.

Differentiation Rules

- Product rule. $(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$
- Quotient rule. $\left(\dfrac{f(x)}{g(x)}\right)' = \dfrac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2}$
- Sum rule. $(f(x) + g(x))' = f'(x) + g'(x)$
- Chain rule. $(g(f(x)))' = g'(f(x))f'(x)$

Quick Recap: Derivatives

What we just learned:

- Derivative = slope of a function at a point (rate of change)
- Taylor series = approximate any smooth function using its derivatives
- Key rules: product, quotient, sum, chain — these are your tools for computing derivatives
- **Next up:** What happens when functions have *multiple* inputs?

Roadmap

Why Should I Care About Gradients?

- Multi-dimensional optimization: Real models have thousands to billions of parameters — you need partial derivatives for each one
- The gradient points in the direction of steepest ascent (negate it $\rightarrow$ steepest descent $=$ learning!)
- Image recognition: The gradient tells you which pixels matter most for the model's prediction
- ChatGPT & LLMs: Every word prediction involves gradient-based updates across millions of parameters

In one sentence: The gradient is the compass that guides AI models toward better predictions.

Gradient

- Now, $f : \mathbb{R}^n \mapsto \mathbb{R}$.
- Gradient of f w.r.t. $\mathbf{x}$, denoted $\nabla_{\mathbf{x}} f$: Vary one variable at a time, keeping the others constant.

Partial Derivative. For $f : \mathbb{R}^n \mapsto \mathbb{R}$,

$$\frac{\partial f}{\partial x_1} = \lim_{h \to 0} \frac{f(x_1 + h, x_2, \ldots, x_n) - f(\mathbf{x})}{h}$$

$$\vdots$$

$$\frac{\partial f}{\partial x_n} = \lim_{h \to 0} \frac{f(x_1, x_2, \ldots, x_n + h) - f(\mathbf{x})}{h}$$

Gradient. Get the partial derivatives and collect them in the row vector.

$$\nabla_{\mathbf{x}} f = \frac{df}{d\mathbf{x}} = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{1 \times n}$$

Example

- **Example.** $f(x,y) = (x+2y^3)^2$

$$\frac{\partial f(x,y)}{\partial x} = 2(x+2y^3)\frac{\partial x + 2y^3}{\partial x} = 2(x+2y^3)$$
$$\frac{\partial f(x,y)}{\partial y} = 2(x+2y^3)\frac{\partial x + 2y^3}{\partial y} = 12(x+2y^3)y^2$$

- **Example.** $f(x_1,x_2) = x_1^2 x_2 + x_1 x_2^3$

$$\nabla_{(x_1,x_2)} f = \frac{df}{dx} = \begin{pmatrix} \frac{\partial f(x_1,x_2)}{\partial x_1} & \frac{\partial f(x_1,x_2)}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2x_1x_2 + x_2^3 & x_1^2 + 3x_1x_2^2 \end{pmatrix}$$

Rules for Partial Differentiation

- Product rule

$$\frac{\partial}{\partial \mathbf{x}}\big(f(\mathbf{x})g(\mathbf{x})\big) = \frac{\partial f}{\partial \mathbf{x}} g(\mathbf{x}) + f(\mathbf{x}) \frac{\partial g}{\partial \mathbf{x}}$$

- Sum rule

$$\frac{\partial}{\partial \mathbf{x}}\big(f(\mathbf{x}) + g(\mathbf{x})\big) = \frac{\partial f}{\partial \mathbf{x}} + \frac{\partial g}{\partial \mathbf{x}}$$

- Chain rule

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

More about Chain Rule

- $f : \mathbb{R}^2 \mapsto \mathbb{R}$ of two variables x_1 and x_2. $x_1(t)$ and $x_2(t)$ are functions of t.

$$\frac{df}{dt} = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{pmatrix} \begin{pmatrix} \frac{\partial x_1(t)}{\partial t} \\ \frac{\partial x_2(t)}{\partial t} \end{pmatrix} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}$$

- **Example.** $f(x_1, x_2) = x_1^2 + 2x_2$, where $x_1(t) = \sin(t), \; x_2(t) = \cos(t)$

$$\frac{df}{dt} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t} = 2\sin(t)\cos(t) - 2\sin(t) = 2\sin(t)(\cos(t) - 1)$$

- $f : \mathbb{R}^2 \mapsto \mathbb{R}$ of two variables x_1 and x_2. $x_1(s,t)$ and $x_2(s,t)$ are functions of s, t.

$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial s} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial s}$$
$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}$$

$$\frac{df}{d(s,t)} = \frac{\partial f}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial(s,t)} = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{pmatrix} \begin{pmatrix} \frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t} \end{pmatrix}$$

Quick Recap: Partial Derivatives & Gradients

What we just learned:

- Partial derivative: change one variable, keep the rest fixed
- Gradient: collect all partial derivatives into a row vector
- Chain rule extends naturally to multiple variables
- **Next up:** What if the output is also a vector? $\rightarrow$ Jacobian matrices

Roadmap

(1) Differentiation of Univariate Functions
(2) Partial Differentiation and Gradients
(3) Gradients of Vector-Valued Functions
(4) Gradients of Matrices
(5) Useful Identities for Computing Gradients
(6) Backpropagation and Automatic Differentiation
(7) Higher-Order Derivatives
(8) Linearization and Multivariate Taylor Series

- For a function $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ and vector $\mathbf{x} = \begin{pmatrix} x_1 & \dots & x_n \end{pmatrix}^\top \in \mathbb{R}^n$, the vector-valued function is:

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{pmatrix}$$

- Partial derivative w.r.t. x_i is a column vector: $\frac{\partial \mathbf{f}}{\partial x_i} = \begin{pmatrix} \frac{\partial f_1}{\partial x_i} \\ \vdots \\ \frac{\partial f_m}{\partial x_i} \end{pmatrix}$
- Gradient (or Jacobian): $\frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n} \end{pmatrix}$

Jacobian

$$\mathbf{J} = \nabla_{\mathbf{x}} \mathbf{f} = \frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

- For a $\mathbb{R}^n \to \mathbb{R}^m$ function, its Jacobian is an $m \times n$ matrix.

Example: Gradient of Vector-Valued Function

- $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$, $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$
- Partial derivatives: $f_i(\mathbf{x}) = \sum_{j=1}^{n} A_{ij} x_j \implies \dfrac{\partial f_i}{\partial x_j} = A_{ij}$
- Gradient

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & & \vdots \\ A_{m1} & \cdots & A_{mn} \end{pmatrix} = \mathbf{A}$$

Example: Chain Rule

- $h : \mathbb{R} \mapsto \mathbb{R}$, $h(t) = (f \circ g)(t)$ with

$$f : \mathbb{R}^2 \mapsto \mathbb{R},\ f(\mathbf{x}) = \exp(x_1 x_2^2), \quad g : \mathbb{R} \mapsto \mathbb{R}^2,\ \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = g(t) = \begin{pmatrix} t\cos(t) \\ t\sin(t) \end{pmatrix}$$

- (Note) $\frac{\partial f}{\partial \mathbf{x}} \in \mathbb{R}^{1\times 2}$ and $\frac{\partial g}{\partial t} \in \mathbb{R}^{2\times 1}$
- Using the chain rule,

$$\begin{aligned} \frac{dh}{dt} = \frac{\partial f}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial t} &= \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{pmatrix} \begin{pmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{pmatrix} \\ &= \begin{pmatrix} \exp(x_1 x_2^2) x_2^2 & 2\exp(x_1 x_2^2) x_1 x_2 \end{pmatrix} \begin{pmatrix} \cos(t) - t\sin(t) \\ \sin(t) + t\cos(t) \end{pmatrix} \end{aligned}$$

Example: Least-Square Loss (1)

- A linear model: $\mathbf{y} = \boldsymbol{\Phi}\boldsymbol{\theta}$
- $\boldsymbol{\theta} \in \mathbb{R}^D$: parameter vector
- $\boldsymbol{\Phi} \in \mathbb{R}^{N \times D}$: input features
- $\mathbf{y} \in \mathbb{R}^N$: observations
- Goal: Find a good parameter vector that provides the best-fit, formulated by minimizing the following loss $L : \mathbb{R}^D \mapsto \mathbb{R}$ over the parameter vector $\boldsymbol{\theta}$.

$$L(\mathbf{e}) \stackrel{\text{def}}{=} \|\mathbf{e}\|^2, \quad \text{where } \mathbf{e}(\boldsymbol{\theta}) = \mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}$$

Example: Least-Square Loss (2)

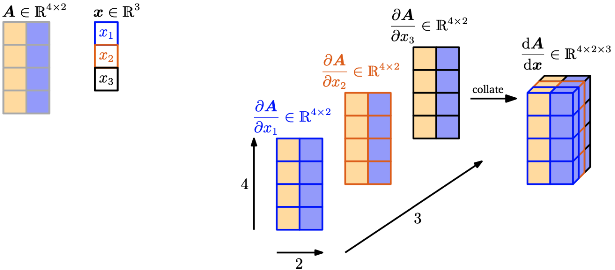
- $\frac{\partial L}{\partial \boldsymbol{\theta}} = \frac{\partial L}{\partial \mathbf{e}} \frac{\partial \mathbf{e}}{\partial \boldsymbol{\theta}}$
- Note. $\frac{\partial L}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{1 \times D}$, $\frac{\partial L}{\partial \mathbf{e}} \in \mathbb{R}^{1 \times N}$, $\frac{\partial \mathbf{e}}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{N \times D}$
- Using that $\|\mathbf{e}\|^2 = \mathbf{e}^\top \mathbf{e}$, $\frac{\partial L}{\partial \mathbf{e}} = 2\mathbf{e}^\top \in \mathbb{R}^{1 \times N}$ and $\frac{\partial \mathbf{e}}{\partial \boldsymbol{\theta}} = -\boldsymbol{\Phi} \in \mathbb{R}^{N \times D}$

Finally, we get: $$\frac{\partial L}{\partial \boldsymbol{\theta}} = 2\mathbf{e}^\top(-\boldsymbol{\Phi}) = -\underbrace{2(\mathbf{y}^\top - \boldsymbol{\theta}^\top \boldsymbol{\Phi}^\top)}_{1 \times N} \underbrace{\boldsymbol{\Phi}}_{N \times D}$$

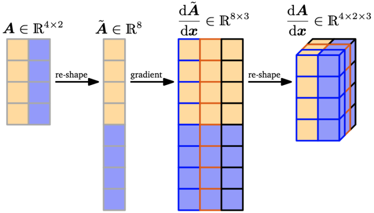
Roadmap

Gradients of Matrices

- Gradient of matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ w.r.t. matrix $\mathbf{B} \in \mathbb{R}^{p\times q}$
- Jacobian: A four-dimensional tensor[1] $\mathbf{J} = \frac{d\mathbf{A}}{d\mathbf{B}} \in \mathbb{R}^{(m\times n)\times(p\times q)}$



(a) Approach 1: We compute the partial derivative $\frac{\partial \boldsymbol{A}}{\partial x_1}, \frac{\partial \boldsymbol{A}}{\partial x_2}, \frac{\partial \boldsymbol{A}}{\partial x_3}$, each of which is a 4×2 matrix, and collate them in a $4 \times 2 \times 3$ tensor.



(b) Approach 2: We re-shape (flatten) $\boldsymbol{A} \in \mathbb{R}^{4\times 2}$ into a vector $\tilde{\boldsymbol{A}} \in \mathbb{R}^{8}$. Then, we compute the gradient $\frac{\mathrm{d}\tilde{\boldsymbol{A}}}{\mathrm{d}\boldsymbol{x}} \in \mathbb{R}^{8\times 3}$. We obtain the gradient tensor by re-shaping this gradient as illustrated above.

[1] A multidimensional array

Example: Gradient of Vectors for Matrices (1)

- $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$, $\mathbf{f} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$. What is $\frac{d\mathbf{f}}{d\mathbf{A}}$?
- Dimension: If we consider $\mathbf{f} : \mathbb{R}^{m \times n} \mapsto \mathbb{R}^m$, $\frac{d\mathbf{f}}{d\mathbf{A}} \in \mathbb{R}^{m \times (m \times n)}$
- Partial derivatives: $\frac{\partial f_i}{\partial \mathbf{A}} \in \mathbb{R}^{1 \times (m \times n)}$, $\quad \frac{d\mathbf{f}}{d\mathbf{A}} = \begin{pmatrix} \frac{\partial f_1}{\partial \mathbf{A}} \\ \vdots \\ \frac{\partial f_m}{\partial \mathbf{A}} \end{pmatrix}$

$$f_i = \sum_{j=1}^{n} A_{ij} x_j, \; i = 1, \ldots, m \implies \frac{\partial f_i}{\partial A_{iq}} = x_q,$$

$$\frac{\partial f_i}{\partial A_{i\cdot}} = \mathbf{x}^\top \in \mathbb{R}^{1 \times 1 \times n} \text{ (for } i\text{th row vector)}$$

$$\frac{\partial f_i}{\partial A_{k \neq i \cdot}} = \mathbf{0}^\top \in \mathbb{R}^{1 \times 1 \times n} \text{ (for } k\text{th row vector, } k \neq i)$$

$$\frac{\partial f_i}{\partial \mathbf{A}} = \begin{pmatrix} \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \\ \mathbf{x}^\top \\ \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \end{pmatrix} \in \mathbb{R}^{1 \times (m \times n)}$$

Example: Gradient of Matrices for Matrices (2)

- $\mathbf{R} \in \mathbb{R}^{m\times n}$ and $\mathbf{f} : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{n\times n}$ with $\mathbf{f}(\mathbf{R}) = \mathbf{K} \stackrel{\text{def}}{=} \mathbf{R}^\top \mathbf{R} \in \mathbb{R}^{n\times n}$. What is $\frac{d\mathbf{K}}{d\mathbf{R}} \in \mathbb{R}^{(n\times n)\times(m\times n)}$?
- $\frac{dK_{pq}}{d\mathbf{R}} \in \mathbb{R}^{1\times m\times n}$. Let $\mathbf{r}_i$ be the ith column of $\mathbf{R}$. Then $K_{pq} = \mathbf{r}_p^\top \mathbf{r}_q = \sum_{k=1}^m R_{kp}R_{kq}$.
- Partial derivative $\frac{\partial K_{pq}}{\partial R_{ij}}$

$$\frac{\partial K_{pq}}{\partial R_{ij}} = \sum_{k=1}^{m} \frac{\partial}{\partial R_{ij}} R_{kp}R_{kq} = \partial_{pqij}, \;\; \partial_{pqij} = \begin{cases} R_{iq} & \text{if } j = p, p \neq q \\ R_{ip} & \text{if } j = q, p \neq q \\ 2R_{iq} & \text{if } j = p, p = q \\ 0 & \text{otherwise} \end{cases}$$

Roadmap

Useful Identities

$$\frac{\partial}{\partial \boldsymbol{X}} \boldsymbol{f}(\boldsymbol{X})^\top = \left(\frac{\partial \boldsymbol{f}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right)^\top \tag{5.99}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \mathrm{tr}(\boldsymbol{f}(\boldsymbol{X})) = \mathrm{tr}\left(\frac{\partial \boldsymbol{f}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right) \tag{5.100}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \det(\boldsymbol{f}(\boldsymbol{X})) = \det(\boldsymbol{f}(\boldsymbol{X}))\mathrm{tr}\left(\boldsymbol{f}(\boldsymbol{X})^{-1}\frac{\partial \boldsymbol{f}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right) \tag{5.101}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \boldsymbol{f}(\boldsymbol{X})^{-1} = -\boldsymbol{f}(\boldsymbol{X})^{-1}\frac{\partial \boldsymbol{f}(\boldsymbol{X})}{\partial \boldsymbol{X}}\boldsymbol{f}(\boldsymbol{X})^{-1} \tag{5.102}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{X}^{-1}\boldsymbol{b}}{\partial \boldsymbol{X}} = -(\boldsymbol{X}^{-1})^\top \boldsymbol{a}\boldsymbol{b}^\top (\boldsymbol{X}^{-1})^\top \tag{5.103}$$

$$\frac{\partial \boldsymbol{x}^\top \boldsymbol{a}}{\partial \boldsymbol{x}} = \boldsymbol{a}^\top \tag{5.104}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{a}^\top \tag{5.105}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{X}\boldsymbol{b}}{\partial \boldsymbol{X}} = \boldsymbol{a}\boldsymbol{b}^\top \tag{5.106}$$

$$\frac{\partial \boldsymbol{x}^\top \boldsymbol{B}\boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{x}^\top(\boldsymbol{B}+\boldsymbol{B}^\top) \tag{5.107}$$

$$\frac{\partial}{\partial \boldsymbol{s}}(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s})^\top \boldsymbol{W}(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s}) = -2(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s})^\top \boldsymbol{W}\boldsymbol{A} \quad \text{for symmetric } \boldsymbol{W} \tag{5.108}$$

Roadmap

Why Should I Care About Backpropagation?

- The engine of deep learning: Without backprop, training neural networks would be computationally impossible
- Efficiency: Backprop computes gradients for millions of parameters in roughly the same time as one forward pass
- Every AI framework uses it: PyTorch, TensorFlow, JAX — they all implement automatic differentiation via backprop
- Understanding backprop = understanding how AI learns

In one sentence: Backpropagation is *the* algorithm that makes training deep networks practical.

Motivation: Neural Networks with Many Layers (1)

- In a neural network with many layers, the function $\mathbf{y}$ is a many-level function composition

$$\mathbf{y} = (f_K \circ f_{K-1} \circ \cdots \circ f_1)(\mathbf{x}),$$

  where, for example,
  - $\mathbf{x}$: images as inputs, $\mathbf{y}$: class labels (e.g., cat or dog) as outputs
  - each f_i has its own parameters
- In neural networks, with the model parameters $\boldsymbol{\theta} = \{\mathbf{A}_0, \mathbf{b}_0, \ldots, \mathbf{A}_{K-1}, \mathbf{b}_{K-1}\}$

$$\begin{cases} \mathbf{f}_0 & \stackrel{\text{def}}{=} \mathbf{x} \\ \mathbf{f}_1 & \stackrel{\text{def}}{=} \sigma_1(\mathbf{A}_0\mathbf{f}_0 + \mathbf{b}_0) \\ \vdots & \\ \mathbf{f}_K & \stackrel{\text{def}}{=} \sigma_K(\mathbf{A}_{K-1}\mathbf{f}_{K-1} + \mathbf{b}_{K-1}) \end{cases}$$

◦ Minimizing the loss function over $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}),$$

where $L(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{f}_K(\boldsymbol{\theta}, \mathbf{x})\|^2$

◦ σ_i is called the activation function at i-th layer

Motivation: Neural Networks with Many Layers (2)

- In neural networks, with the model parameters $\boldsymbol{\theta} = \{\mathbf{A}_0, \mathbf{b}_0, \ldots, \mathbf{A}_{K-1}, \mathbf{b}_{K-1}\}$

$$
\begin{cases}
\mathbf{f}_0 & \stackrel{\text{def}}{=} \mathbf{x} \\
\mathbf{f}_1 & \stackrel{\text{def}}{=} \sigma_1(\mathbf{A}_0 \mathbf{f}_0 + \mathbf{b}_0) \\
\vdots & \\
\mathbf{f}_K & \stackrel{\text{def}}{=} \sigma_K(\mathbf{A}_{K-1} \mathbf{f}_{K-1} + \mathbf{b}_{K-1})
\end{cases}
$$

◦ Minimizing the loss function over $\boldsymbol{\theta}$:

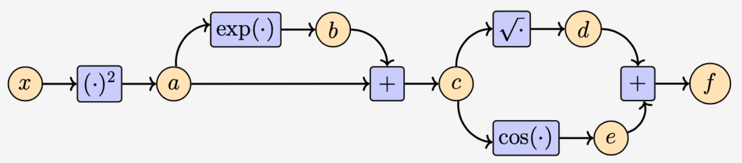
$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}),$$

where $L(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{f}_K(\boldsymbol{\theta}, \mathbf{x})\|^2$

◦ σ_i is called the activation function at i-th layer

- **Question.** How can we efficiently compute $\dfrac{dL}{d\boldsymbol{\theta}}$ in computers?

- $f(x) = \sqrt{x^2 + \exp(x^2)} + \cos\left(x^2 + \exp(x^2)\right)$
- Computation graph: Connect via "elementary" operations



$$a = x^2,\ b = \exp(a),\ c = a + b,\ d = \sqrt{c},\ e = \cos(c),\ f = d + e$$

- Automatic Differentiation
  - A set of techniques to numerically (not symbolically) evaluate the gradient of a function by working with intermediate variables and applying the chain rule.

Backpropagation: Example (2)

- $a = x^2,\ b = \exp(a),\ c = a + b,\ d = \sqrt{c},\ e = \cos(c),\ f = d + e$
- Derivatives of the intermediate variables with their inputs

$$\frac{\partial a}{\partial x} = 2x,\ \frac{\partial b}{\partial a} = \exp(a),\ \frac{\partial c}{\partial a} = 1 = \frac{\partial c}{\partial b},\ \frac{\partial d}{\partial c} = \frac{1}{2\sqrt{c}},\ \frac{\partial e}{\partial c} = -\sin(c),\ \frac{\partial f}{\partial d} = 1 = \frac{\partial}{\partial}$$

- Compute $\frac{\partial f}{\partial x}$ by working backward from the output

$$\frac{\partial f}{\partial c} = \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c},\quad \frac{\partial f}{\partial b} = \frac{\partial f}{\partial c}\frac{\partial c}{\partial b}$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a},\quad \boxed{\frac{\partial f}{\partial x}} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial x}$$

$$\frac{\partial f}{\partial c} = 1 \cdot \frac{1}{2\sqrt{c}} + 1 \cdot (-\sin(c))$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \cdot 1,\quad \frac{\partial f}{\partial a} = \frac{\partial f}{\partial b}\exp(a) + \frac{\partial f}{\partial c} \cdot 1$$

$$\boxed{\frac{\partial f}{\partial x}} = \frac{\partial f}{\partial a} \cdot 2x$$

Quick Recap: Backpropagation

What we just learned:

- Build a computation graph from elementary operations
- Compute derivatives of each elementary step
- Work backward from output to input, chaining derivatives
- Result: efficient gradient of the entire function!
- **Next up:** Higher-order derivatives and Taylor approximations

Backpropagation

- Implementation of gradients can be very expensive, unless we are careful.
- Using the idea of automatic differentiation, the whole gradient computation is decomposed into a set of gradients of elementary functions and application of the chain rule.
- Why backward?
  - In neural networks, the input dimensionality is often much higher than the dimensionality of labels.
  - In this case, backward computation is much cheaper than forward computation.
- Works if the target is expressed as a computation graph whose elementary functions are differentiable. If not, some care needs to be taken.

Roadmap

Higher-Order Derivatives

- Some optimization algorithms (e.g., Newton's method) require second-order derivatives, if they exist.
- (Truncated) Taylor series is often used as an approximation of a function.
- For $f : \mathbb{R}^n \mapsto \mathbb{R}$ of variable $\mathbf{x} \in \mathbb{R}^n$, $\nabla_{\mathbf{x}} f = \frac{df}{d\mathbf{x}} = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{1 \times n}$
  - If f is twice-differentiable, the order doesn't matter.

$$\nabla^2_{\mathbf{x}} f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \, \partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial^2 f}{\partial x_1 \, \partial x_n} & \frac{\partial^2 f}{\partial x_2 \, \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n \, \partial x_n} \end{pmatrix}$$

- For $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, $\nabla_{\mathbf{x}} f \in \mathbb{R}^{m \times n}$
  - Thus, $\nabla^2_{\mathbf{x}} f \in \mathbb{R}^{m \times n \times n}$ (a tensor)

Roadmap

Function Approximation: Linearization and More

- First-order approximation of $f(\mathbf{x})$ (i.e., linearization by taking the first two terms of Taylor Series)
$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + (\nabla_{\mathbf{x}} f)(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$
- Multivariate Taylor Series for $f : \mathbb{R}^D \mapsto \mathbb{R}$ at $\mathbf{x}_0$
$$f(\mathbf{x}) = \sum_{k=0}^{\infty} \frac{D_{\mathbf{x}}^k f(\mathbf{x}_0)}{k!} \delta^k,$$
where $D_{\mathbf{x}}^k f(\mathbf{x}_0)$ is the kth derivative of f w.r.t. $\mathbf{x}$, evaluated at $\mathbf{x}_0$, and $\delta \stackrel{\text{def}}{=} \mathbf{x} - \mathbf{x}_0$.
  - Partial sum up to, say n, can be an approximation of $f(\mathbf{x})$.
  - $D_{\mathbf{x}}^k f(\mathbf{x}_0)$ and δ^k are kth order tensors, i.e., k-dimensional arrays.
  - δ^k is a k-fold outer product $\otimes$. For example, $\delta^2 = \delta \otimes \delta = \delta\delta^\top$. $\delta^3 = \delta \otimes \delta \otimes \delta$.

Calculus Concepts at a Glance

Concept	Input → Output	Result
Derivative	$f : \mathbb{R} \to \mathbb{R}$	scalar $f'(x)$
Partial Derivative	$f : \mathbb{R}^n \to \mathbb{R}$	scalar $\frac{\partial f}{\partial x_i}$
Gradient	$f : \mathbb{R}^n \to \mathbb{R}$	row vector $\in \mathbb{R}^{1\times n}$
Jacobian	$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$	matrix $\in \mathbb{R}^{m\times n}$
Hessian	$f : \mathbb{R}^n \to \mathbb{R}$	matrix $\in \mathbb{R}^{n\times n}$

Pattern: As inputs/outputs grow in dimension, derivatives grow in dimension too!

Common Mistakes to Avoid

(1) Forgetting the chain rule in multi-layer functions
Each layer contributes a factor — multiply them all together.

(2) Confusing gradient shape (row vs. column)
Gradient of $f : \mathbb{R}^n \to \mathbb{R}$ is a row vector ($1 \times n$), not a column vector.

(3) Treating partial derivatives as total derivatives
$\frac{\partial f}{\partial x_1}$ holds $x_2, \ldots, x_n$ fixed. The total derivative does not.

(4) Wrong matrix dimensions in the chain rule
Always check: inner dimensions must match when multiplying Jacobians.

(5) Forgetting that the Hessian is symmetric
For twice-differentiable f: $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$.

Key Takeaways

(1) Derivative = rate of change; gradient = vector of all partial derivatives

(2) Chain rule is the foundation of backpropagation: compose simple derivatives to get complex ones

(3) Jacobian ($m \times n$ matrix) generalizes the gradient for vector-valued functions $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$

(4) Backpropagation = efficient gradient computation via chain rule applied backward through a computation graph

(5) Hessian captures curvature (second-order information) — used in advanced optimizers like Newton's method

(6) Concept Chain:

$$\text{Derivative} \to \text{Gradient} \to \text{Jacobian} \to \text{Backprop} \to \text{Hessian} \to \text{Taylor}$$

Questions?

Review Question 1

- In machine learning, optimization minimizes a loss function. Explain how the derivative definition

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

relates to updating model parameters in gradient descent.

Review Question 2

- The chain rule is central to deep learning. Explain why

  $$(g(f(x)))' = g'(f(x))f'(x)$$

  is required to train neural networks with multiple layers.

Review Question 3

- For $f : \mathbb{R}^n \mapsto \mathbb{R}$, the gradient gives the direction of steepest ascent. Explain how gradients are used to update model parameters during training.

Review Question 4

- Consider a neural network layer written as

  $$\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}.$$

  Show that its Jacobian is $\mathbf{A}$. What does this represent in terms of feature transformation in deep learning?

Review Question 5

- In computer vision, images are represented as high-dimensional vectors. Explain how partial derivatives measure how sensitive the loss is to small changes in pixel values.

Review Question 6

- Using the chain rule,

$$\frac{df}{dt} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t},$$

explain how backpropagation computes gradients through multiple layers.

Review Question 7

- For the least-squares loss

$$L(\theta) = \|\mathbf{y} - \mathbf{\Phi}\theta\|^2,$$

explain what the matrix $\mathbf{\Phi}$ represents in machine learning and how minimizing this loss corresponds to fitting a model.

Review Question 8

- Explain why backpropagation is computationally efficient when training deep neural networks with many parameters.

Review Question 9

- The Hessian matrix contains second-order derivatives. Explain how curvature information can influence optimization in machine learning (e.g., faster convergence or instability).

Review Question 10

- Taylor expansion gives a local approximation:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + (\nabla f)(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0).$$

Explain how this idea relates to linearization in machine learning models and local behavior of neural networks.

Theory (1): Backprop as Vector–Jacobian Products (VJP)

- Consider a composition

$$\mathbf{f}_K = (f_K \circ f_{K-1} \circ \cdots \circ f_1)(\mathbf{x}), \qquad L = \ell(\mathbf{f}_K).$$

- Jacobians:

$$\mathbf{J}_i \overset{\text{def}}{=} \frac{df_i}{d\mathbf{f}_{i-1}} \quad \Rightarrow \quad \frac{d\mathbf{f}_K}{d\mathbf{x}} = \mathbf{J}_K \mathbf{J}_{K-1} \cdots \mathbf{J}_1.$$

- Naively storing $\mathbf{J}_i$ is infeasible. Backprop computes VJP:

$$\underbrace{\frac{dL}{d\mathbf{f}_{i-1}}}_{1\times d_{i-1}} = \underbrace{\frac{dL}{d\mathbf{f}_i}}_{1\times d_i} \underbrace{\mathbf{J}_i}_{d_i \times d_{i-1}} .$$

- Practical takeaway:
  - Backward mode is efficient when output is scalar (typical loss).
  - Complexity is comparable to forward pass (up to constant factors).

Theory (2): Hessian Structure and Curvature-Aware Updates

- For parameters $\boldsymbol{\theta} \in \mathbb{R}^P$, gradient and Hessian:

$$\nabla L(\boldsymbol{\theta}) \in \mathbb{R}^{1\times P}, \qquad \nabla^2 L(\boldsymbol{\theta}) \in \mathbb{R}^{P\times P}.$$

- Second-order Taylor expansion:

$$L(\boldsymbol{\theta} + \Delta) \approx L(\boldsymbol{\theta}) + \nabla L(\boldsymbol{\theta})\Delta + \frac{1}{2}\Delta^\top \nabla^2 L(\boldsymbol{\theta})\Delta.$$

- Newton step (ideal):

$$\Delta^* = -\nabla^2 L(\boldsymbol{\theta})^{-1} \nabla L(\boldsymbol{\theta})^\top.$$

- Why we care in deep nets:
  - Sharp directions (large eigenvalues of ∇^2) constrain step size.
  - Flat directions (small eigenvalues) dominate generalization behavior.
- Common approximation idea (diagonal / low-rank):

$$\nabla^2 \approx \mathbf{U}_k \Lambda_k \mathbf{U}_k^\top + \lambda \mathbf{I}.$$

Theory (3): Gauss–Newton / Fisher from Jacobians

- For squared loss with residual $\mathbf{r}(\boldsymbol{\theta}) \in \mathbb{R}^N$:

$$L(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{r}(\boldsymbol{\theta})\|^2, \qquad \mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}(\boldsymbol{\theta}).$$

- Jacobian of residuals:

$$\mathbf{J}_r \stackrel{\text{def}}{=} \frac{d\mathbf{r}}{d\boldsymbol{\theta}} \in \mathbb{R}^{N\times P}.$$

- Gradient:

$$\nabla L(\boldsymbol{\theta}) = \mathbf{r}^\top \mathbf{J}_r.$$

- Exact Hessian:

$$\nabla^2 L(\boldsymbol{\theta}) = \mathbf{J}_r^\top \mathbf{J}_r + \sum_{i=1}^{N} r_i \nabla^2 r_i.$$

- Gauss–Newton approximation (drop the second term):

$$\nabla^2 L(\boldsymbol{\theta}) \approx \mathbf{J}_r^\top \mathbf{J}_r,$$

  which is PSD and captures curvature near good fits (small r_i).
- Link to Fisher (probabilistic view): curvature $\approx$ covariance of score.

Theory (4): Linearization and Adversarial Sensitivity

- First-order change in loss under input perturbation δ:

$$L(\mathbf{x} + \delta) \approx L(\mathbf{x}) + \nabla_{\mathbf{x}} L(\mathbf{x})\,\delta.$$

- Worst-case (under ℓ_2-budget $\|\delta\|_2 \leq \epsilon$):

$$\max_{\|\delta\|_2 \leq \epsilon} \nabla_{\mathbf{x}} L\,\delta = \epsilon\,\|\nabla_{\mathbf{x}} L\|_2\,.$$

- For ℓ_∞-budget $\|\delta\|_\infty \leq \epsilon$:

$$\delta^* = \epsilon\,\mathrm{sign}\big(\nabla_{\mathbf{x}} L(\mathbf{x})\big) \quad \text{(FGSM-type direction)}.$$

- Vision connection:
  - Robustness is controlled by Jacobian norms of the network.
  - Regularizers such as $\|\mathbf{J}_{\mathbf{x}}\|$ penalize local sensitivity.

Theory (5): Jacobian Rank and Feature Collapse

- Let $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^d$ be a learned representation (penultimate layer). Jacobian:

$$\mathbf{J}_{\mathbf{x}} \stackrel{\text{def}}{=} \frac{d\mathbf{f}}{d\mathbf{x}} \in \mathbb{R}^{d \times D}.$$

- Local dimension is governed by $\operatorname{rank}(\mathbf{J}_{\mathbf{x}})$:
  - If $\operatorname{rank}(\mathbf{J}_{\mathbf{x}}) \ll \min(d, D)$, features vary only in a low-dim subspace.
  - This is a differential view of representation collapse.
- A useful summary:

$$\text{effective dim}(\mathbf{x}) \;\approx\; \operatorname{rank}(\mathbf{J}_{\mathbf{x}}) \quad \text{or} \quad \operatorname{tr}(\mathbf{J}_{\mathbf{x}}\mathbf{J}_{\mathbf{x}}^{\top}).$$

- Practical link:
  - Self-supervised methods control collapse by constraints on covariance / spectrum.
  - In CNNs, collapse is often layerwise and spatially structured.

PhD Level Content Only

The next slides are research-level vector calculus for deep learning theory:

- Implicit bias via gradient flow
- NTK / kernel regression dynamics
- Fisher geometry and natural gradients
- Spectral structure of Hessians in vision models
- Differential view of invariances (Lie groups)

Not intended for undergraduate coursework

PhD (6): Gradient Flow and Implicit Regularization

- Continuous-time limit of gradient descent:

$$\frac{d\boldsymbol{\theta}(t)}{dt} = -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}(t))^\top.$$

- For linear regression with separable data and certain losses, gradient flow selects a minimum-norm solution (implicit bias).
- Prototype statement (informal but useful):

$$\boldsymbol{\theta}(t) \text{ converges to } \arg\min_{\boldsymbol{\theta}} \ \|\boldsymbol{\theta}\|_2 \quad \text{s.t.} \quad \text{zero training error.}$$

- In deep linear networks, the induced implicit bias often corresponds to low-rank / nuclear-norm-like structure on end-to-end maps.
- Conceptual bridge to your Lecture 4:

$$\text{Training} \Rightarrow \text{spectral shrinkage of singular values.}$$

PhD (7): NTK as a Spectral Learning Operator

- Linearization at initialization $\boldsymbol{\theta}_0$:

$$f_{\boldsymbol{\theta}}(x) \approx f_{\boldsymbol{\theta}_0}(x) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_0}(x)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0).$$

- Define NTK kernel on dataset $\{x_i\}$:

$$\mathbf{K}_{ij} = \langle \nabla_{\boldsymbol{\theta}} f(x_i), \nabla_{\boldsymbol{\theta}} f(x_j) \rangle.$$

- Gradient descent on squared loss yields (in function space):

$$\hat{\mathbf{y}}(t) = \mathbf{y} - e^{-\eta \mathbf{K} t} \big(\mathbf{y} - \hat{\mathbf{y}}(0)\big).$$

- Spectral decomposition $\mathbf{K} = \mathbf{U} \boldsymbol{\Lambda} \mathbf{U}^\top$ gives modewise rates:

mode k converges at rate $e^{-\eta \lambda_k t}$.

- Vision implication: spectrum of $\mathbf{K}$ determines which patterns are learned first.

PhD (8): Fisher Information and Natural Gradient

- Probabilistic model $p(y|x, \boldsymbol{\theta})$ with negative log-likelihood loss $L(\boldsymbol{\theta})$.
- Fisher information:

$$\mathbf{F}(\boldsymbol{\theta}) = \mathbb{E}\Big[\nabla_{\boldsymbol{\theta}} \log p(y|x, \boldsymbol{\theta})\ \nabla_{\boldsymbol{\theta}} \log p(y|x, \boldsymbol{\theta})^\top\Big].$$

- Natural gradient step (Riemannian metric induced by $\mathbf{F}$):

$$\boldsymbol{\theta}^+ = \boldsymbol{\theta} - \eta\, \mathbf{F}(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top.$$

- Spectral view:

$$\mathbf{F} = \mathbf{U}\Lambda\mathbf{U}^\top \quad \Rightarrow \quad \mathbf{F}^{-1} = \mathbf{U}\Lambda^{-1}\mathbf{U}^\top.$$

- Interpretation:
  - learning is slowed along high-curvature directions
  - preconditioning equalizes progress across parameter manifolds

PhD (9): Hessian Spectrum, Sharpness, and Generalization

- Local quadratic model:

$$L(\boldsymbol{\theta} + \Delta) \approx L(\boldsymbol{\theta}) + \nabla L\, \Delta + \frac{1}{2}\Delta^\top \nabla^2 L\, \Delta.$$

- Let $\nabla^2 L = \mathbf{U}\Lambda\mathbf{U}^\top$ with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_P$.
- Sharpness proxy (spectral):

$$\text{sharpness} \sim \lambda_1 \quad \text{or} \quad \mathrm{tr}(\nabla^2 L) = \sum_i \lambda_i.$$

- In large vision nets, empirical Hessians are often:
  - low effective rank (few large eigenvalues)
  - heavy-tailed spectra (many small but nontrivial modes)
- Practical bridge:

$$\text{SVD / low-rank approx} \;\Rightarrow\; \text{efficient curvature modeling.}$$

PhD (10): Differential Invariances via Lie Groups

- Suppose a transformation group $g(\alpha)$ acts on inputs: $x \mapsto g(\alpha) \cdot x$.
- Invariance objective:

$$f(g(\alpha) \cdot x) \approx f(x) \quad \forall \alpha \text{ small.}$$

- Differentiate at identity ($\alpha = 0$):

$$\frac{d}{d\alpha} f(g(\alpha) \cdot x)\Big|_{\alpha=0} = \nabla_x f(x) \, \frac{d}{d\alpha}(g(\alpha) \cdot x)\Big|_{\alpha=0} \approx 0.$$

- This yields a constraint on the gradient:

$$\nabla_x f(x) \, \mathbf{T}(x) \approx 0,$$

  where $\mathbf{T}(x)$ spans the tangent space of the orbit (translations, rotations, etc.).
- Vision takeaway:
  - CNN inductive bias approximates invariance by architectural constraints.
  - Gradient-based regularizers can enforce invariances at training time.