Derivatives for Machine Learning

From Scalars to Tensors with Visual Intuition

Saurabh

June 14, 2025

Contents

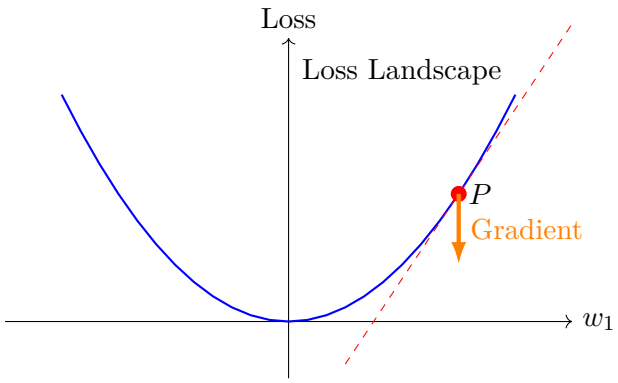
1 Your Journey into the Mathematics of AI

Welcome to the world of machine learning, where we teach computers to learn from data. At the heart of this process is a simple goal: **minimizing error**. We need a way to measure how wrong our model is and a method to systematically make it better.

Imagine you are in a vast, foggy valley, and your goal is to find the absolute lowest point. This valley is the "loss function"—a mathematical landscape where height represents error. To navigate, you can't see far, but you can feel the slope of the ground beneath your feet. The smartest move is to take a step in the direction where the ground descends most steeply.

That "slope" is the **derivative**.

This guide is your map and compass. We will walk you through everything you need to know about derivatives, starting from the basics and building up, step-by-step, to the powerful concepts that drive modern AI. Each section is packed with detailed explanations, visual diagrams, and practical examples to build a deep and lasting intuition.



2 Part 1: Scalar-Valued Functions (Output is one number)

These functions calculate a single "score" or "cost". They are the foundation of optimization in machine learning.

2.1 Case 1: Scalar input, Scalar output

The Classic Derivative

A function where a single input number maps to a single output number.

$$y = f(x)$$

The derivative, $\frac{dy}{dx}$, is also a scalar.

The Slope of a Curve

The derivative tells you the slope of the function's tangent line at a specific point. It answers the question: "If I nudge the input x a tiny bit, how fast and in what direction does the output y change?" A positive derivative means y increases as x increases; a negative derivative means y decreases.
Think of it as the speedometer for your function. But instead of just telling you your speed, it also tells you if you're going forward or backward.

Calculation 1

Question: Find the derivative of $y = 4x^3 - 5x + 7$.
Solution: We use the power rule on each term.

$$\frac{dy}{dx} = \frac{d}{dx}(4x^3) - \frac{d}{dx}(5x) + \frac{d}{dx}(7) = 4(3x^2) - 5(1) + 0 = \mathbf{12x^2 - 5}$$

Calculation 2

Question: Find the derivative of $y = e^{2x}\sin(x)$ using the product rule $(uv)' = u'v + uv'$.
Solution: Let $u = e^{2x}$ and $v = \sin(x)$. Then $u' = 2e^{2x}$ and $v' = \cos(x)$.

$$\frac{dy}{dx} = (2e^{2x})(\sin(x)) + (e^{2x})(\cos(x)) = \mathbf{e^{2x}(2\sin(x) + \cos(x))}$$

2.2 Case 2: Vector input, Scalar output

The Gradient

A function where a vector of inputs maps to a single output number.

$$y = f(\vec{w}) \quad \text{where} \quad \vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

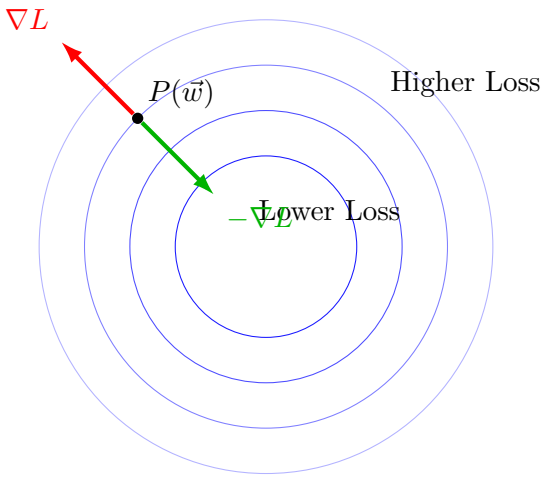
The derivative is a **vector** of the same dimension as the input, called the **gradient**, denoted ∇f.

$$\nabla f(\vec{w}) = \frac{dy}{d\vec{w}} = \begin{pmatrix} \partial y/\partial w_1 \\ \vdots \\ \partial y/\partial w_n \end{pmatrix}$$

The Compass and The Recipe

The Compass Analogy: Think of the input vector $\vec{w}$ as your coordinates on a map, and the output y as your altitude. The gradient vector is a compass that always points in the direction of the **steepest uphill slope**. Its length (magnitude) tells you how steep that slope is.
The Recipe Analogy: Suppose your function y is the "tastiness score" of a cake, and $\vec{w}$ is the vector of ingredient amounts (flour, sugar, etc.). The gradient is a recipe for

improvement. It tells you the exact ratio of ingredients to add or subtract to make the cake tastier, faster. For example, if $\nabla y = \begin{pmatrix} -0.1 \\ 2.5 \end{pmatrix}$, it means reducing flour a bit and increasing sugar a lot is the fastest way to improve the taste from this point.



The gradient ∇L points towards higher loss.
We move in the opposite direction, $-\nabla L$, to minimize the loss.

The Heart of Optimization

This is the single most important derivative form in all of ML. Our loss function is a scalar value that depends on a vector of model weights, $L(\vec{w})$. To train the model, we compute the gradient ∇L and update the weights in the opposite direction: $\vec{w}_{new} = \vec{w}_{old} - \eta \nabla L$. This is **Gradient Descent**. More advanced optimizers like **Adam** or **RMSprop** still use the gradient as their core input but adapt the step size η intelligently.

Calculation 1

Question: Find the gradient of $L(w_1, w_2) = (3w_1 - 4)^2 + 5w_2^2$.
Solution: Find the partial derivative w.r.t. w_1 and w_2.

$$\frac{\partial L}{\partial w_1} = 2(3w_1 - 4) \cdot 3 = 18w_1 - 24 \quad \text{and} \quad \frac{\partial L}{\partial w_2} = 2 \cdot 5w_2 = 10w_2$$

The gradient vector is: $\nabla L = \begin{pmatrix} 18w_1 - 24 \\ 10w_2 \end{pmatrix}$.

Calculation 2 (Cross-Entropy Loss)

Question: Find the gradient of the binary cross-entropy loss $L = -(y \log(p) + (1 - y) \log(1 - p))$ with respect to p. (y is a constant 0 or 1).
Solution:

$$\frac{\partial L}{\partial p} = -\left(y \cdot \frac{1}{p} + (1 - y) \cdot \frac{1}{1 - p} \cdot (-1)\right) = -\left(\frac{y}{p} - \frac{1 - y}{1 - p}\right) = \frac{1 - y}{1 - p} - \frac{y}{p} = \frac{p - y}{p(1 - p)}$$

This form is crucial for classification problems.

2.3 Case 3: Matrix input, Scalar output

The Gradient Matrix

A function where a matrix of inputs maps to a single output number.

$$y = f(X)$$

The derivative, $\frac{dy}{dX}$, is a **matrix** of the same dimensions as the input matrix X.

$$\left(\frac{dy}{dX}\right)_{ij} = \frac{\partial y}{\partial X_{ij}}$$

A Sensitivity Heatmap

Imagine the input matrix X is a grid of control knobs, and the output y is a single master alarm light. The derivative matrix is like a heatmap overlaid on your grid of knobs. The "hottest" (largest value) elements in the derivative matrix are the knobs that have the most powerful effect on the alarm light. An entry of zero means that knob is disconnected from the alarm.

Derivatives for Image Kernels

This is used when a single loss value depends on a matrix of parameters, like the weights in a **convolutional neural network (CNN) kernel** or a covariance matrix in a statistical model. We need to know how to adjust each weight in the kernel to reduce the overall loss. The derivative matrix gives us the precise update for every parameter in the matrix.

Calculation 1

Question: Let $X = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix}$ and let y be the sum of its squared elements, $y = \sum_{i,j} x_{ij}^2$. Find $\frac{dy}{dX}$.
Solution: $y = x_{11}^2 + x_{12}^2 + x_{21}^2 + x_{22}^2$. We compute the partial derivative for each position.

$$\frac{dy}{dX} = \begin{pmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} \\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} \end{pmatrix} = \begin{pmatrix} 2x_{11} & 2x_{12} \\ 2x_{21} & 2x_{22} \end{pmatrix} = 2X$$

Calculation 2

Question: For a 2×2 matrix X, find the derivative of its determinant, $y = \det(X)$, with respect to X.
Solution: $y = x_{11}x_{22} - x_{12}x_{21}$.

$$\frac{dy}{dX} = \begin{pmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} \\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} \end{pmatrix} = \begin{pmatrix} x_{22} & -x_{21} \\ -x_{12} & x_{11} \end{pmatrix}$$

This matrix is known as the adjugate matrix of X.

3 Part 2: Vector-Valued Functions (Output is a vector)

These functions transform a set of inputs into a set of outputs. They represent a transformation from one space to another, like a layer in a neural network.

3.1 Case 4: Vector input, Vector output

The Jacobian Matrix

A function where an n-dimensional vector of inputs maps to an m-dimensional vector of outputs.

$$\vec{y} = F(\vec{x}) \quad \text{where} \quad \vec{x} \in \mathbb{R}^n, \vec{y} \in \mathbb{R}^m$$

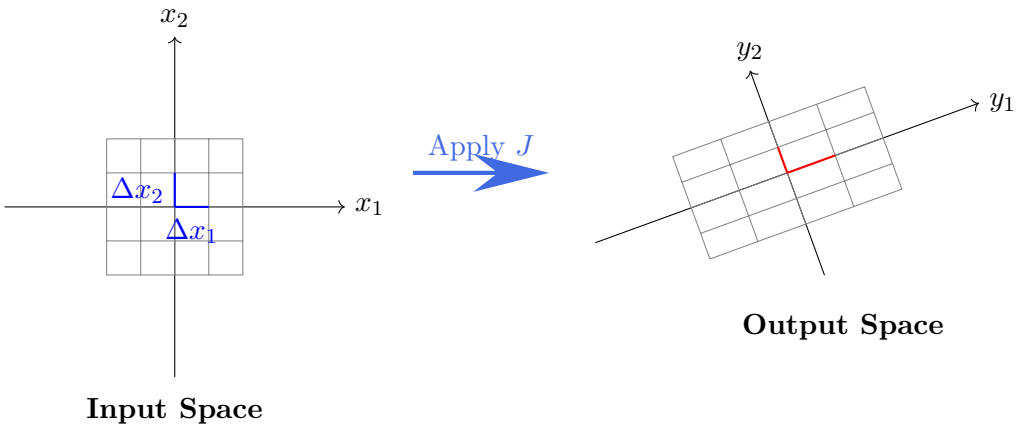
The derivative is an $m \times n$ **matrix** called the **Jacobian**, denoted J.

$$J_{ij} = \frac{\partial y_i}{\partial x_j} \quad \text{(How output } i \text{ changes with input } j\text{)}$$

The Local Rulebook for Transformation

Linear Approximation: Functions can be complex and curvy. The Jacobian gives us a simpler, linear approximation of that function at a specific point. It tells us how a tiny input vector change will be stretched, squeezed, rotated, and sheared into an output vector change.

Video Game Analogy: Imagine a character in a video game whose complex, flowing movement is controlled by a joystick. The joystick's position is the input vector $\vec{x}$. The character's velocity (speed and direction) is the output vector $\vec{y}$. The function F is the game's physics engine. The Jacobian is the rulebook at one specific instant: if you nudge the joystick slightly forward, the Jacobian tells you the exact change in the character's velocity vector (e.g., "+5 speed forward, +1 speed right").



The Engine of Backpropagation

The Jacobian is the heart of the chain rule for vectors. In a neural network, each layer is a vector-to-vector function. Backpropagation uses the Jacobian of each layer's activation function to calculate how errors should flow backward through the network, telling us how to update the weights.

Calculation 1

Question: Find the Jacobian matrix for $\vec{y} = \begin{pmatrix} x_1^2 x_2 \\ 5x_1 + \sin(x_2) \end{pmatrix}$.

Solution: J is 2×2. We compute the four partial derivatives:

$$J = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2x_1x_2 & x_1^2 \\ 5 & \cos(x_2) \end{pmatrix}$$

Calculation 2

Question: Find the Jacobian for a linear layer $\vec{y} = W\vec{x}$, where W is a constant 2×2 matrix and $\vec{x}$ is a 2×1 vector.

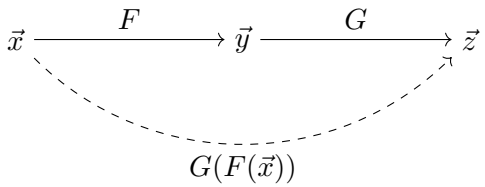
Solution: Let $W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$. Then $\vec{y} = \begin{pmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{pmatrix}$.

$$J = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = W$$

The derivative of a linear transformation is the transformation matrix itself!

3.2 The Chain Rule with Jacobians

The true power of Jacobians is revealed in the chain rule. If you have a sequence of functions, $\vec{x} \xrightarrow{F} \vec{y} \xrightarrow{G} \vec{z}$, the overall Jacobian is the product of the individual Jacobians.



$$J_{G \circ F} = (J_G \text{ at } F(\vec{x})) \cdot (J_F \text{ at } \vec{x}) \quad \text{or simply} \quad \frac{d\vec{z}}{d\vec{x}} = \frac{d\vec{z}}{d\vec{y}} \frac{d\vec{y}}{d\vec{x}}$$

This is not element-wise multiplication; it is **matrix multiplication**. This is exactly how backpropagation works: the error gradient from a later layer (e.g., $\frac{dL}{d\vec{y}}$) is multiplied by the Jacobian of the current layer ($\frac{d\vec{y}}{d\vec{x}}$) to find the error gradient for the previous layer ($\frac{dL}{d\vec{x}}$).

4 Part 3: Matrix-Valued Functions (Output is a matrix)

4.1 Case 5: Matrix input, Matrix output

The Tensor Derivative

A function where an $m \times n$ matrix of inputs maps to a $p \times q$ matrix of outputs.

$$Y = F(X)$$

The derivative, $\frac{dY}{dX}$, is a **rank-4 tensor** (a 4-dimensional array). Its components are:

$$\left(\frac{dY}{dX}\right)_{ijkl} = \frac{\partial Y_{ij}}{\partial X_{kl}}$$

A Note on Layout Conventions

When taking derivatives with vectors and matrices, the exact shape of the result can differ based on convention. For example, the gradient could be a row vector instead of a column vector. This is a choice between "numerator layout" and "denominator layout". This guide uses a single, consistent layout (denominator layout). When reading other resources, be aware that their formulas might differ in shape (e.g., be transposed) due to using a different convention.

The Grand Control Panel Material Science

The Control Panel Analogy: Imagine the input matrix X is a giant control panel full of knobs, and the output matrix Y is a giant wall of display screens. The derivative tensor is the complete wiring diagram. It tells you exactly how much turning any single knob (X_{kl}) will affect the reading on any single screen (Y_{ij}). It captures every possible interaction between every input and every output.
The Material Science Analogy: Think of a 2D sheet of a smart material (Matrix Y). You can apply a pattern of electric fields to it (Matrix X). The function F describes the physics. The derivative tensor is the material's fundamental property: it tells you how applying an electric field at point (k, l) causes the material to deform or change color at point (i, j).

Advanced Backpropagation

While we rarely construct the full tensor explicitly due to its size, its mathematical structure underpins backpropagation through complex layers. When you train a **Convolutional Neural Network (CNN)**, a **Recurrent Neural Network (RNN)**, or an **Attention/Transformer** model, the software library's backward pass is implicitly calculating the operations defined by this tensor calculus. For example, the derivative of a matrix multiplication or a convolution operation is defined by these tensor rules.

Mathematical Concept vs. Computational Reality

It's crucial to understand that while the derivative is mathematically a giant rank-4 tensor, libraries like TensorFlow and PyTorch almost **never** actually build this tensor in memory. It would be incredibly inefficient. Instead, they define the 'backward()' or 'gradient()' operation for a function (like matrix multiplication). This operation directly computes the required output gradient given the input gradient, effectively applying the tensor's linear transformation without ever instantiating the tensor itself.

Calculation 1

Question: For $Y = X^2$ (element-wise), find the derivative of the single output element y_{12} with respect to the entire input matrix X.
Solution: This asks for one "slice" of the full tensor. We want $\frac{\partial y_{12}}{\partial X}$. Since $y_{12} = x_{12}^2$, it only depends on x_{12}.

$$\frac{\partial y_{12}}{\partial X} = \begin{pmatrix} \frac{\partial y_{12}}{\partial x_{11}} & \frac{\partial y_{12}}{\partial x_{12}} \\ \frac{\partial y_{12}}{\partial x_{21}} & \frac{\partial y_{12}}{\partial x_{22}} \end{pmatrix} = \begin{pmatrix} 0 & 2x_{12} \\ 0 & 0 \end{pmatrix}$$

Calculation 2

Question: Let $Y = AXB$, where A and B are constant matrices. Find the derivative $\frac{dY}{dX}$.
Solution: This is a linear transformation. We look at a single element $Y_{ij} = \sum_k \sum_l A_{ik} X_{kl} B_{lj}$. To find the derivative component $(\frac{dY}{dX})_{ijpq} = \frac{\partial Y_{ij}}{\partial X_{pq}}$, we see that X_{pq} only appears in the sum when $k = p$ and $l = q$.

$$\frac{\partial Y_{ij}}{\partial X_{pq}} = \frac{\partial}{\partial X_{pq}}(A_{ip} X_{pq} B_{qj}) = A_{ip} B_{qj}$$

This is the full expression for the tensor components.

5 Practice Problems with Solutions

5.1 Problem 1: Gradient Calculation

Question: A simple neuron's output is given by $a = \sigma(z)$, where $z = w_1x_1 + w_2x_2 + b$ and $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. The loss for a single training example is $L = (a - y_{true})^2$, where y_{true} is the correct label. Find the gradient of the loss L with respect to the weights vector $\vec{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$.

Solution: We need to find $\nabla L = \begin{pmatrix} \partial L/\partial w_1 \\ \partial L/\partial w_2 \end{pmatrix}$. We use the chain rule.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

1. $\frac{\partial L}{\partial a} = 2(a - y_{true})$
2. The derivative of the sigmoid is famous: $\frac{\partial a}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z)) = a(1 - a)$
3. $\frac{\partial z}{\partial w_1} = x_1$

Combining these gives: $\frac{\partial L}{\partial w_1} = 2(a - y_{true})a(1 - a)x_1$. By symmetry, $\frac{\partial L}{\partial w_2} = 2(a - y_{true})a(1 - a)x_2$. The gradient is:

$$\nabla L = \begin{pmatrix} 2(a - y_{true})a(1 - a)x_1 \\ 2(a - y_{true})a(1 - a)x_2 \end{pmatrix}$$

5.2 Problem 2: Jacobian Calculation

Question: A function transforms a 2D vector $\vec{x}$ to a 3D vector $\vec{y}$ as follows:

$$\vec{y} = F(\vec{x}) = \begin{pmatrix} x_1^2 + x_2 \\ x_1x_2 \\ 3x_2^3 \end{pmatrix}$$

Find the Jacobian matrix of this transformation.

Solution: The Jacobian will be a 3×2 matrix (output dims $\times$ input dims).

$$J = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2x_1 & 1 \\ x_2 & x_1 \\ 0 & 9x_2^2 \end{pmatrix}$$

5.3 Problem 3: Advanced Matrix Derivative

Question: In linear regression, a common loss function is the squared Frobenius norm of the error: $L = ||AX - B||_F^2$, where A, X, B are matrices. Find the derivative of L with respect to the matrix X.

Solution: The squared Frobenius norm is the sum of squared elements, which can be conveniently written as $L = \text{tr}((AX - B)^T(AX - B))$. The trace is a useful tool for matrix derivatives. Using properties of the trace and the chain rule:

$$\frac{\partial L}{\partial X} = \frac{\partial}{\partial X}\text{tr}((AX - B)^T(AX - B))$$

$$= \frac{\partial}{\partial X}\text{tr}((X^TA^T - B^T)(AX - B))$$

$$= \frac{\partial}{\partial X}\text{tr}(X^TA^TAX - X^TA^TB - B^TAX + B^TB)$$

Using the derivative rules $\frac{\partial}{\partial X}\text{tr}(X^TCX) = (C + C^T)X$ and $\frac{\partial}{\partial X}\text{tr}(C^TAX) = A^TC^T$:

$$\frac{\partial L}{\partial X} = (A^TA + (A^TA)^T)X - A^TB - (B^TA)^T$$

Since A^TA is symmetric, $(A^TA)^T = A^TA$. Also, $(B^TA)^T = A^TB$.

$$\frac{\partial L}{\partial X} = 2A^TAX - A^TB - A^TB = \mathbf{2A^T(AX - B)}$$

Setting this gradient to zero, $A^T(AX - B) = 0$, gives the famous normal equations for linear regression.