Comprehensive Gradient Formulae for Linear Regression and Neural Networks

A technical compilation

November 6, 2025

Contents

1 Matrix Calculus Fundamentals (The Building Blocks)

This section outlines the basic gradient identities used in machine learning. We use the "denominator layout" convention.

1.1 Notation

- y: a scalar
- $\mathbf{x}, \mathbf{w}, \mathbf{a}, \mathbf{b}, \mathbf{y}$: column vectors of size $d \times 1$
- X, W, A: matrices of size $n \times d$ or $d \times k$

1.2 Scalar-by-Vector Gradients ($\nabla_{\mathbf{x}} y$)

The result is a $d \times 1$ column vector.

$$\begin{aligned}
\nabla_{\mathbf{x}}(\mathbf{a}^T\mathbf{x}) &= \mathbf{a} \\
\nabla_{\mathbf{x}}(\mathbf{x}^T\mathbf{a}) &= \mathbf{a} \\
\nabla_{\mathbf{x}}(\mathbf{x}^T\mathbf{x}) &= 2\mathbf{x} \\
\nabla_{\mathbf{x}}(\mathbf{x}^T A\mathbf{x}) &= (A + A^T)\mathbf{x} \\
\nabla_{\mathbf{x}}(\mathbf{x}^T A\mathbf{x}) &= 2A\mathbf{x} \quad (\text{if } A \text{ is symmetric})
\end{aligned}$$

1.3 Vector-by-Vector Gradients (Jacobian, $\nabla_{\mathbf{x}}\mathbf{y}$)

The result is a $d \times d$ matrix where $J_{ij} = \frac{\partial y_i}{\partial x_j}$. We list the gradient of the *output* with respect to the *input*.

$$\nabla_{\mathbf{x}}(A\mathbf{x}) = A^T \quad (\text{Note: Result is } d \times n \text{ if } \mathbf{y} \in \mathbb{R}^n, \mathbf{x} \in \mathbb{R}^d)$$

To be precise, if $\mathbf{y} = A\mathbf{x}$ where $\mathbf{y} \in \mathbb{R}^n, A \in \mathbb{R}^{n \times d}, \mathbf{x} \in \mathbb{R}^d$, the Jacobian matrix $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is an $n \times d$ matrix:

$$\frac{\partial (A\mathbf{x})}{\partial \mathbf{x}} = A$$

Our convention $\nabla_{\mathbf{x}}\mathbf{y}$ (denominator layout) often implies $\frac{\partial \mathbf{y}^T}{\partial \mathbf{x}}$, which is A^T. We will stick to element-wise derivatives for clarity in backpropagation.

1.4 Scalar-by-Matrix Gradients ($\nabla_W y$)

The result is a matrix of the same shape as W.

$$\begin{aligned}
\nabla_W \mathsf{tr}(AW) &= A^T \\
\nabla_W \mathsf{tr}(W^T A) &= A \\
\nabla_W \mathsf{tr}(W^T AW) &= (A + A^T)W \\
\nabla_W \|\mathbf{y} - XW\|_F^2 &= 2X^T(XW - \mathbf{y})
\end{aligned}$$

2 Gradients for Linear & Ridge Regression

2.1 Notation

- **y**: True labels, $n \times 1$ vector.
- X: Feature matrix, $n \times d$ (or $n \times (d+1)$ with bias).
- **w**: Weight vector, $d \times 1$ (or $(d+1) \times 1$ with bias).
- $\hat{\mathbf{y}}$: Predictions, $\hat{\mathbf{y}} = X\mathbf{w}$.
- n: Number of samples.
- λ: Regularization parameter.

2.2 Mean Squared Error (MSE) Loss (OLS)

The loss function (or cost function) is:

$$J(\mathbf{w}) = \frac{1}{n}\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = \frac{1}{n}(\mathbf{y} - X\mathbf{w})^T(\mathbf{y} - X\mathbf{w})$$

Expand the loss:

$$J(\mathbf{w}) = \frac{1}{n}(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T X\mathbf{w} + \mathbf{w}^T X^T X\mathbf{w})$$

The gradient $\nabla_{\mathbf{w}} J(\mathbf{w})$ is:

$$\begin{aligned}
\nabla_{\mathbf{w}} J(\mathbf{w}) &= \frac{1}{n}\left[\nabla_{\mathbf{w}}(\mathbf{y}^T\mathbf{y}) - \nabla_{\mathbf{w}}(2\mathbf{y}^T X\mathbf{w}) + \nabla_{\mathbf{w}}(\mathbf{w}^T X^T X\mathbf{w})\right] \\
&= \frac{1}{n}\left[\mathbf{0} - 2(\mathbf{y}^T X)^T + (X^T X + (X^T X)^T)\mathbf{w}\right] \\
&= \frac{1}{n}\left[-2X^T\mathbf{y} + 2X^T X\mathbf{w}\right] \quad \text{(since } X^T X \text{ is symmetric)} \\
\nabla_{\mathbf{w}} J(\mathbf{w}) &= \frac{2}{n}X^T(X\mathbf{w} - \mathbf{y})
\end{aligned}$$

2.3 Ridge Regression (MSE + L2 Regularization)

The loss function adds an L2 penalty on the weights:

$$J_{\text{ridge}}(\mathbf{w}) = J(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2 = \frac{1}{n}\|\mathbf{y} - X\mathbf{w}\|_2^2 + \lambda\mathbf{w}^T\mathbf{w}$$

The gradient is the OLS gradient plus the gradient of the penalty term:

$$\begin{aligned}
\nabla_{\mathbf{w}}(\lambda\mathbf{w}^T\mathbf{w}) &= \lambda(2\mathbf{w}) \\
\nabla_{\mathbf{w}} J_{\text{ridge}}(\mathbf{w}) &= \frac{2}{n}X^T(X\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w}
\end{aligned}$$

2.4 Lasso Regression (MSE + L1 Regularization)

The loss function adds an L1 penalty. The L1 norm is not differentiable at $w_i = 0$. We use the *sub-gradient*, denoted ∂.

$$J_{\text{lasso}}(\mathbf{w}) = \frac{1}{n}\|\mathbf{y} - X\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_1$$

The sub-gradient of the L1 norm is $\lambda \cdot \text{sign}(\mathbf{w})$, where $\text{sign}(0)$ is any value in $[-1, 1]$.

$$\partial_{\mathbf{w}} J_{\text{lasso}}(\mathbf{w}) = \frac{2}{n}X^T(X\mathbf{w} - \mathbf{y}) + \lambda \cdot \text{sign}(\mathbf{w})$$

3 Gradients for Neural Networks (Backpropagation)

This section describes the core chain rule application for a feed-forward neural network.

3.1 Notation

- L: The final scalar loss.
- l: A specific layer, from 1 to L (output layer).
- $\mathbf{a}^{(l)}$: Activation vector of layer l. $\mathbf{a}^{(0)} = \mathbf{x}$ (input).
- $\mathbf{z}^{(l)}$: Pre-activation (linear combination) of layer l.
- $W^{(l)}$: Weight matrix for layer l (connects $l-1$ to l).
- $\mathbf{b}^{(l)}$: Bias vector for layer l.
- $\sigma(\cdot)$: Element-wise activation function (e.g., Sigmoid, ReLU).
- $\sigma'(\cdot)$: Derivative of the activation function.
- $\odot$: Element-wise (Hadamard) product.

3.2 Forward Pass

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$
$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)})$$

3.3 Core Backpropagation Equations

We define the "error" $\delta^{(l)}$ as the gradient of the loss L with respect to the *pre-activation* $\mathbf{z}^{(l)}$ of that layer:

$$\delta^{(l)} = \frac{\partial L}{\partial \mathbf{z}^{(l)}}$$

3.3.1 1. Error at Output Layer (l = L)

The error at the final layer is the gradient of the loss with respect to the final prediction, multiplied by the gradient of the final activation.

$$\delta^{(L)} = \frac{\partial L}{\partial \mathbf{a}^{(L)}} \odot \sigma'(\mathbf{z}^{(L)})$$

(See Section 5 for common $\frac{\partial L}{\partial \mathbf{a}^{(L)}}$ formulae).

3.3.2 2. Error at Hidden Layer (l < L)

The error is propagated backwards from the next layer ($l+1$).

$$\delta^{(l)} = \left((W^{(l+1)})^T \delta^{(l+1)}\right) \odot \sigma'(\mathbf{z}^{(l)})$$

This is the chain rule: $\frac{\partial L}{\partial \mathbf{z}^{(l)}} = \frac{\partial L}{\partial \mathbf{z}^{(l+1)}} \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}}$.

3.3.3 3. Gradient for Weights

The gradient of the loss with respect to the weights $W^{(l)}$ is:

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^T$$

3.3.4 4. Gradient for Biases

The gradient of the loss with respect to the biases $\mathbf{b}^{(l)}$ is:

$$\frac{\partial L}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

(Note: For batch gradient descent, the gradients for W and b are averaged over all samples in the batch).

4 Common Activation Function Gradients (Derivatives)

The derivative $\sigma'(z)$ is required for backpropagation.

Table 1: Derivatives of common activation functions

Function	**Formula** $\sigma(z)$	**Gradient** $\sigma'(z)$
Sigmoid	$\frac{1}{1+e^{-z}}$	$\sigma(z)(1-\sigma(z))$
Hyperbolic Tangent (tanh)	$\frac{e^z - e^{-z}}{e^z + e^{-z}}$	$1 - \tanh^2(z)$
ReLU	$\max(0, z)$	$\begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$
Leaky ReLU	$\max(\alpha z, z)$	$\begin{cases} 1 & \text{if } z > 0 \\ \alpha & \text{if } z \leq 0 \end{cases}$

4.1 Softmax (Special Case)

Softmax is not element-wise; it depends on all elements of $\mathbf{z}$.

$$\sigma(\mathbf{z})_j = a_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Its gradient is a Jacobian matrix where the partial derivative $\frac{\partial a_i}{\partial z_j}$ (using $a_i = \sigma(\mathbf{z})_i$) is:

$$\frac{\partial a_i}{\partial z_j} = a_i(\delta_{ij} - a_j) = \begin{cases} a_i(1-a_i) & \text{if } i = j \\ -a_i a_j & \text{if } i \neq j \end{cases}$$

where δ_{ij} is the Kronecker delta (1 if $i = j$, 0 otherwise).

5 Common Loss Function Gradients (for Neural Networks)

This is the formula for $\frac{\partial L}{\partial \mathbf{a}^{(L)}}$, the gradient of the loss with respect to the *final activation (prediction)*.

- **y**: True label vector.
- **a**: Prediction vector (i.e., $\mathbf{a}^{(L)}$).
- n: Number of output classes/nodes.

5.1 Mean Squared Error (MSE) Loss

Used for regression.

$$L = \sum_{i=1}^{n} (y_i - a_i)^2$$

$$\frac{\partial L}{\partial a_i} = -2(y_i - a_i) \quad \Longrightarrow \quad \nabla_{\mathbf{a}} L = -2(\mathbf{y} - \mathbf{a}) = 2(\mathbf{a} - \mathbf{y})$$

5.2 Binary Cross-Entropy (BCE) Loss

Used for binary classification (single output node with Sigmoid).

$$L = -\left(y \log(a) + (1 - y) \log(1 - a)\right)$$

$$\frac{\partial L}{\partial a} = -\left(\frac{y}{a} - \frac{1 - y}{1 - a}\right) = \frac{a - y}{a(1 - a)}$$

5.3 Categorical Cross-Entropy (CCE) with Softmax

This is a *crucial optimization*. When CCE is used as the loss for a Softmax output layer, the combined gradient $\delta^{(L)} = \frac{\partial L}{\partial \mathbf{z}^{(L)}}$ simplifies beautifully.

- **Loss:** $L = -\sum_{i=1}^{n} y_i \log(a_i)$, where $\mathbf{a} = \text{softmax}(\mathbf{z})$.
- **Combined Gradient:** The chain rule $\delta^{(L)} = \frac{\partial L}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}}$ simplifies to:

$$\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}$$

This simple ”prediction - true” form is why Softmax and CCE are almost always used together for multi-class classification.