From Gaussian Integral to Differentiable Programming: The Jacobian Matrix and Its Applications

Nguyen Vu Hung

November 12, 2025

Abstract

This document explores the fundamental role of the Jacobian matrix in mathematics and its critical applications in deep learning. We begin with the classical Gaussian integral problem, demonstrating how the Jacobian transformation enables elegant solutions to otherwise intractable integrals. We then trace the evolution of this concept from multivariable calculus to modern machine learning, showing how the Jacobian serves as the mathematical backbone of backpropagation, optimisation algorithms, and neural network training. Through this journey, we connect classical mathematical theory to cutting-edge deep learning techniques.

1 Overview

The Jacobian matrix is one of the most important mathematical tools bridging classical calculus and modern machine learning. Named after the German mathematician Carl Gustav Jacob Jacobi (1804–1851), who made fundamental contributions to analysis and mechanics, the Jacobian was originally developed for solving multivariable calculus problems, particularly in the context of coordinate transformations and change of variables in integrals. The Jacobian has found profound applications in deep learning, where it enables efficient computation of gradients through backpropagation.

This document presents a comprehensive exploration of the Jacobian matrix, starting from its foundational role in solving the Gaussian integral, through its theoretical development in multivariable calculus, and culminating in its practical applications in neural networks and optimisation algorithms.

2 Problem 1: The Gaussian Integral

Problem 1 (Gaussian Integral (Euler-Poisson Integral))**.** *Evaluate the integral:*

$$I = \int_{-\infty}^{\infty} e^{-x^2}\, dx \tag{1}$$

This integral, also known as the Euler-Poisson integral, is fundamental in probability theory, statistics, and physics. Despite its simple appearance, it cannot be evaluated using elementary antiderivatives.

The solution that follows is verbose and lengthy, but it is well worth working through carefully, as it illustrates fundamental techniques that generalise to modern applications.

2.1 Solution to Problem 1

The standard approach to solving this integral involves a clever trick: squaring the integral and converting it to a double integral.

2.1.1 Step 1: Squaring the Integral

We consider the square of the integral:

$$I^2 = \left(\int_{-\infty}^{\infty} e^{-x^2}\, dx \right)^2 = \int_{-\infty}^{\infty} e^{-x^2}\, dx \cdot \int_{-\infty}^{\infty} e^{-y^2}\, dy \tag{2}$$

2.1.2 Step 2: Converting to a Double Integral

Since the two integrals are independent, we can combine them into a double integral over the entire xy-plane:

$$I^2 = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-(x^2+y^2)}\, dx\, dy \tag{3}$$

2.1.3 Step 3: Direct Evaluation Attempt

At this point, we have a double integral that is still difficult to evaluate directly in Cartesian coordinates. The integrand $e^{-(x^2+y^2)}$ suggests that polar coordinates might be more suitable, as $x^2 + y^2 = r^2$ in polar coordinates.

2.2 Solution Using Jacobian Transformation

The key insight is to transform the integral from Cartesian coordinates (x, y) to polar coordinates (r, θ) using the Jacobian transformation.

2.2.1 Coordinate Transformation

The transformation from Cartesian to polar coordinates is:

$$x = r\cos\theta \tag{4}$$

$$y = r\sin\theta \tag{5}$$

where $r \geq 0$ and $0 \leq \theta \leq 2\pi$.

2.2.2 Region of Integration

The region of integration (the entire xy-plane) transforms to:

$$0 \leq r < \infty \tag{6}$$

$$0 \leq \theta \leq 2\pi \tag{7}$$

2.2.3 The Jacobian Determinant

The crucial step is replacing the differential area element $dx\, dy$ with the appropriate expression in polar coordinates. This requires the Jacobian determinant of the transformation, which we will compute in the next section. For now, we state that:

$$dx\, dy = r\, dr\, d\theta \tag{8}$$

where the factor r is the absolute value of the Jacobian determinant.

2.2.4 Transformed Integral

The double integral becomes:

$$I^2 = \int_0^{2\pi} \int_0^{\infty} e^{-r^2} \cdot r\, dr\, d\theta \tag{9}$$

2.2.5 Solving the Polar Integral

We can now separate the integrals:

$$I^2 = \int_0^{2\pi} d\theta \cdot \int_0^{\infty} re^{-r^2}\, dr \tag{10}$$

The θ integral is straightforward:

$$\int_0^{2\pi} d\theta = 2\pi \tag{11}$$

For the r integral, we use the substitution $u = r^2$, so $du = 2r\, dr$, or $r\, dr = \frac{du}{2}$:

$$\int_0^{\infty} re^{-r^2}\, dr = \int_0^{\infty} e^{-u} \cdot \frac{du}{2} \tag{12}$$

$$= \frac{1}{2}\int_0^{\infty} e^{-u}\, du \tag{13}$$

$$= \frac{1}{2}\left[-e^{-u}\right]_0^{\infty} \tag{14}$$

$$= \frac{1}{2}(0 - (-1)) = \frac{1}{2} \tag{15}$$

2.2.6 Final Result

Substituting back:

$$I^2 = 2\pi \cdot \frac{1}{2} = \pi \tag{16}$$

Taking the positive square root (since the integrand is always positive):

$$I = \int_{-\infty}^{\infty} e^{-x^2}\, dx = \sqrt{\pi} \tag{17}$$

2.3 What We Learned from Problem 1

The solution to the Gaussian integral demonstrates several important concepts that are central to this document:

- **The Power of Coordinate Transformation**: The integral was intractable in Cartesian coordinates but became solvable after transforming to polar coordinates. This illustrates how choosing the right coordinate system can simplify complex problems.
- **The Role of the Jacobian Determinant**: The key step was recognising that $dx\, dy = r\, dr\, d\theta$, where the factor r is the Jacobian determinant of the transformation. This correction factor accounts for how the coordinate transformation stretches or compresses the differential area element.
- **Connection to Multivariable Calculus**: This problem showcases the fundamental principle that when changing variables in multiple integrals, we must account for how the transformation affects the differential elements—a concept that the Jacobian matrix generalises to higher dimensions and more complex transformations.
- **Bridge to Modern Applications**: While this classical problem uses the Jacobian for coordinate transformations in integration, the same mathematical framework—computing how transformations affect differential elements—underlies modern deep learning, where the Jacobian matrix describes how neural network layers transform input gradients during backpropagation.

This elegant solution sets the stage for understanding how the Jacobian matrix, originally developed for such classical problems, has become indispensable in modern machine learning and optimisation.

3 The Jacobian Determinant in Multivariable Calculus

3.1 Definition and Explanation

Definition 1 (Jacobian Matrix)**.** *For a transformation from variables $(u, v, \ldots)$ to $(x, y, \ldots)$, where $x = x(u, v, \ldots)$ and $y = y(u, v, \ldots)$, the* ***Jacobian matrix*** $\mathbf{J}_M$ *is the matrix of all first-order partial derivatives:*

$$\mathbf{J}_M = \begin{bmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} & \cdots \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{18}$$

Definition 2 (Jacobian Determinant)**.** *The* ***Jacobian determinant****, denoted J or $\det(\mathbf{J}_M)$, is the determinant of the Jacobian matrix. For a transformation from (u, v) to (x, y):*

$$J = \det(\mathbf{J}_M) = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{vmatrix} = \frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial x}{\partial v}\frac{\partial y}{\partial u} \tag{19}$$

The Jacobian determinant acts as a scaling factor or correction factor needed when transforming a small differential volume/area element from one coordinate system to another. In a double integral:

$$\iint_R f(x, y)\, dx\, dy = \iint_{R'} f(x(u, v), y(u, v)) \cdot |J|\, du\, dv \tag{20}$$

where $|J|$ is the absolute value of the Jacobian determinant.

3.2 Application: Cartesian to Polar Coordinates

In the Gaussian integral example, we transformed from Cartesian (x, y) to polar (r, θ) coordinates.

3.2.1 Transformation Equations

$$x = r\cos\theta \tag{21}$$
$$y = r\sin\theta \tag{22}$$

3.2.2 Calculating Partial Derivatives

$$\frac{\partial x}{\partial r} = \cos\theta, \quad \frac{\partial x}{\partial \theta} = -r\sin\theta \tag{23}$$
$$\frac{\partial y}{\partial r} = \sin\theta, \quad \frac{\partial y}{\partial \theta} = r\cos\theta \tag{24}$$

3.2.3 Calculating the Determinant

$$J = \begin{vmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{vmatrix} \tag{25}$$

$$= \cos\theta \cdot r\cos\theta - (-r\sin\theta)\cdot\sin\theta \tag{26}$$

$$= r\cos^2\theta + r\sin^2\theta \tag{27}$$

$$= r(\cos^2\theta + \sin^2\theta) \tag{28}$$

$$= r \tag{29}$$

3.2.4 Result

Since $r \geq 0$, the absolute value $|J| = r$. Therefore, the differential area element $dx\,dy$ is replaced by $r\,dr\,d\theta$, as used in the Gaussian integral calculation.

4 Analogy to Single-Variable Calculus

The Jacobian determinant generalises the concept of the derivative in single-variable calculus to multivariable functions.

In single-variable calculus, when we perform a change of variables $x = g(u)$, the differential transforms as:

$$dx = g'(u)\,du = \frac{dx}{du}\,du \tag{30}$$

The factor $g'(u) = \frac{dx}{du}$ is the one-dimensional analog of the Jacobian determinant. In the multivariable case, the Jacobian determinant J plays the same role, accounting for how the transformation stretches or compresses the differential area/volume element.

For a single-variable integral:

$$\int_a^b f(x)\,dx = \int_{g^{-1}(a)}^{g^{-1}(b)} f(g(u))\cdot|g'(u)|\,du \tag{31}$$

This is directly analogous to the multivariable change of variables formula using the Jacobian.

5 Special Case: Linear Transformations

5.1 Jacobian of a Linear Transformation

A linear transformation $\mathbf{T}$ from $\mathbb{R}^n$ to $\mathbb{R}^m$ is defined by an $m \times n$ matrix $\mathbf{A}$:

$$\mathbf{T}(\mathbf{x}) = \mathbf{A}\mathbf{x} \tag{32}$$

Written component-wise:

$$t_i = \sum_{j=1}^{n} a_{ij}x_j, \quad i = 1, 2, \ldots, m \tag{33}$$

5.2 Computing the Jacobian

The Jacobian matrix $\mathbf{J_T}$ is the matrix of all first-order partial derivatives:

$$\mathbf{J_T} = \begin{bmatrix} \frac{\partial t_1}{\partial x_1} & \frac{\partial t_1}{\partial x_2} & \cdots & \frac{\partial t_1}{\partial x_n} \\ \frac{\partial t_2}{\partial x_1} & \frac{\partial t_2}{\partial x_2} & \cdots & \frac{\partial t_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial t_m}{\partial x_1} & \frac{\partial t_m}{\partial x_2} & \cdots & \frac{\partial t_m}{\partial x_n} \end{bmatrix} \tag{34}$$

For a linear function:

$$\frac{\partial t_i}{\partial x_j} = a_{ij} \tag{35}$$

Therefore:

$$\mathbf{J_T}(\mathbf{x}) = \mathbf{A} \tag{36}$$

5.3 Key Properties

1. **Constant Jacobian**: The Jacobian matrix of a linear transformation is constant and equal to the transformation matrix $\mathbf{A}$ itself.
2. **Uniform Scaling**: Because the transformation is linear, the "stretching" or "distortion" it imposes on the input space is uniform everywhere.
3. **Best Linear Approximation**: For a function that is already linear, the best linear approximation is the function itself, so the Jacobian is exactly the matrix that defines the function.

5.4 Change of Variables for Linear Transformations

If a coordinate transformation is linear (e.g., a simple scaling or rotation), the Jacobian determinant is just the determinant of the transformation matrix, and it gives the uniform scaling factor for the area or volume element.

6 Best Linear Approximation Using Jacobian

The Jacobian matrix represents the best linear approximation of a function near a point. For a differentiable function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, the linear approximation near a point $\mathbf{x}_0$ is:

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + \mathbf{J_f}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \tag{37}$$

where $\mathbf{J_f}(\mathbf{x}_0)$ is the Jacobian matrix evaluated at $\mathbf{x}_0$.

This is the multivariable generalisation of the tangent line approximation in single-variable calculus:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \tag{38}$$

7 Jacobian's Role in Optimisation

7.1 Generalising the First Derivative

In single-variable calculus, an extremum (minimum or maximum) is found where the first derivative is zero: $f'(x) = 0$.

For scalar-valued functions (loss functions) $L(\mathbf{w})$ with respect to a vector of parameters $\mathbf{w}$, the Jacobian specialises to the gradient vector ∇L. The Jacobian of a scalar function

$\mathbf{T} : \mathbb{R}^n \to \mathbb{R}^1$ is a $1 \times n$ matrix (a row vector), which is the transpose of the standard gradient column vector:

$$\nabla L(\mathbf{w}) = \begin{bmatrix} \frac{\partial L}{\partial w_1} & \frac{\partial L}{\partial w_2} & \cdots & \frac{\partial L}{\partial w_n} \end{bmatrix}^T \tag{39}$$

To find critical points, we set:

$$\nabla L(\mathbf{w}) = \mathbf{0} \tag{40}$$

7.2 Jacobian and Gradient Descent

The Jacobian (in the form of the gradient) is the core of Gradient Descent (an iterative optimisation algorithm that moves in the direction of the negative gradient to minimise a function) and its variants: SGD (Stochastic Gradient Descent, which uses a random subset of data at each iteration, making it faster and more memory-efficient for large datasets) and Adam (Adaptive Moment Estimation, which combines momentum and adaptive learning rates for each parameter, often providing faster convergence and better performance on sparse gradients), which are fundamental algorithms in machine learning.

The gradient vector $\nabla L(\mathbf{w})$ points in the direction of the steepest ascent. Optimisation algorithms use the negative gradient $-\nabla L$ to determine the direction of the next step, ensuring movement toward a local minimum:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla L(\mathbf{w}^{(t)}) \tag{41}$$

where η is the learning rate (a hyperparameter that controls the step size in the direction of the negative gradient; a larger learning rate takes bigger steps but may overshoot the minimum, while a smaller learning rate takes smaller, more cautious steps but may converge slowly).

7.3 Nonlinear Least Squares

In Nonlinear Least Squares (NLS) problems, we minimise the sum of squares of a vector-valued function $\mathbf{F}(\mathbf{x})$:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} [f_i(\mathbf{x})]^2 \tag{42}$$

Here, $\mathbf{F}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$, and the Jacobian matrix $\mathbf{J_F}$ of $\mathbf{F}$ is explicitly calculated. Algorithms like the Gauss-Newton method (an iterative optimisation algorithm that approximates the Hessian using the Jacobian, avoiding expensive second-derivative computations) or the Levenberg-Marquardt algorithm (a damped version of Gauss-Newton that combines the benefits of gradient descent and Gauss-Newton, particularly robust for ill-conditioned problems) use $\mathbf{J_F}$ to approximate the Hessian matrix (the matrix of second-order partial derivatives, which captures the curvature of the function):

$$\mathbf{H} \approx \mathbf{J_F}^T \mathbf{J_F} \tag{43}$$

Providing the analytical Jacobian to the solver significantly improves efficiency and speed of convergence compared to finite-difference approximations.

7.4 Multi-Objective Optimisation

The Jacobian is essential for problems where we simultaneously optimise multiple objective functions $\mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))$. The Jacobian $\mathbf{J_F}$ is composed of the gradients of all m objective functions, stacked as rows:

$$\mathbf{J_F} = \begin{bmatrix} \nabla f_1(\mathbf{x})^T \\ \nabla f_2(\mathbf{x})^T \\ \vdots \\ \nabla f_m(\mathbf{x})^T \end{bmatrix} \tag{44}$$

8 Jacobian Matrix in Deep Learning

The Jacobian matrix is fundamental to deep learning, primarily serving as the mathematical backbone for backpropagation, the algorithm used to train neural networks.

A deep neural network is a vector-valued composite function that maps an input vector (like an image or a set of features) to an output vector (like classification probabilities). The Jacobian provides a way to calculate all the necessary first-order partial derivatives of this complex, multi-layered function.

8.1 The Jacobian Matrix Definition

The Jacobian matrix $\mathbf{J}$ collects all the first-order partial derivatives of a function that has multiple inputs and multiple outputs. If $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, the Jacobian is an $m \times n$ matrix:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{45}$$

8.2 Backpropagation

8.2.1 Chain Rule

Training a neural network requires Gradient Descent, which minimises a loss function L by adjusting the network's weights $\mathbf{W}$. This requires calculating the gradient $\nabla_{\mathbf{W}} L$.

Since the loss L depends on the network's final output $\mathbf{y}$, and $\mathbf{y}$ depends on the weights $\mathbf{W}$ across many layers, the gradient must be computed using the multivariate Chain Rule.

8.2.2 Layer-wise Derivatives

For each layer transformation $f^{(l)}$ with input $\mathbf{x}^{(l)}$ and output $\mathbf{x}^{(l+1)}$, the derivative $\frac{\partial \mathbf{x}^{(l+1)}}{\partial \mathbf{x}^{(l)}}$ is the Jacobian matrix of that layer.

8.2.3 Propagation

Backpropagation efficiently uses matrix multiplication of these Jacobian matrices, propagating the error signal (gradient) backward through the network layers to update the weights. This allows the network to efficiently compute the final gradient:

$$\nabla_{\mathbf{W}} L = \mathbf{J}^{(L)} \mathbf{J}^{(L-1)} \cdots \mathbf{J}^{(1)} \tag{46}$$

where each $\mathbf{J}^{(l)}$ is the Jacobian matrix of layer l.

8.3 Neural Networks

In a neural network, each layer performs a transformation:

$$\mathbf{x}^{(l+1)} = \sigma(\mathbf{W}^{(l)} \mathbf{x}^{(l)} + \mathbf{b}^{(l)}) \tag{47}$$

where σ is an activation function, $\mathbf{W}^{(l)}$ is the weight matrix, and $\mathbf{b}^{(l)}$ is the bias vector.

The Jacobian of this transformation with respect to the input $\mathbf{x}^{(l)}$ is:

$$\mathbf{J}^{(l)} = \frac{\partial \mathbf{x}^{(l+1)}}{\partial \mathbf{x}^{(l)}} = \operatorname{diag}(\sigma'(\mathbf{z}^{(l)})) \mathbf{W}^{(l)} \tag{48}$$

where $\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{x}^{(l)} + \mathbf{b}^{(l)}$ and σ' is the derivative of the activation function.

8.4 Analysis and Stability

Beyond training, the Jacobian is used to analyse the behaviour of trained models:

- **Sensitivity Analysis**: The Jacobian of the output with respect to the input $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ reveals how sensitive the model's output is to small changes in the input data. Large elements in this Jacobian indicate that the network output is very volatile, which often corresponds to a lack of robustness.
- **Adversarial Attacks**: Adversarial examples (inputs slightly modified to trick the network) are often constructed using information from the Jacobian $\frac{\partial L}{\partial \mathbf{x}}$ to find the direction of input change that most rapidly increases the loss.
- **Gradient Stability**: The spectral norm (or largest singular value) of the Jacobian of the layer-wise transformation is key to understanding and mitigating the vanishing/exploding gradient problem. Initialisation schemes like Xavier or Kaiming aim to keep the norm of these Jacobians close to 1 to ensure effective training.

9 Advanced Optimisations

9.1 Newton's Method

Newton's method is a second-order optimisation method that uses the Hessian matrix (the matrix of second-order partial derivatives). The Hessian can be approximated using the Jacobian:

$$\mathbf{H} \approx \mathbf{J}^T \mathbf{J} \tag{49}$$

Newton's method update rule is:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \mathbf{H}^{-1} \nabla L(\mathbf{x}^{(t)}) \tag{50}$$

While more computationally expensive than gradient descent, Newton's method can converge much faster, especially near the optimum.

9.2 Jacobian-Enhanced Neural Networks (JENN)

Jacobian-Enhanced Neural Networks (JENN) are specialised networks that explicitly include terms in their loss function to ensure the network not only fits the target function values $\mathbf{F}(\mathbf{x})$ but also accurately predicts the Jacobian matrix $\mathbf{J}(\mathbf{x})$. This leads to higher accuracy with less training data.

The loss function for JENN includes both function value error and Jacobian error:

$$L_{\text{JENN}} = L_{\text{function}} + \lambda L_{\text{Jacobian}} \tag{51}$$

where λ is a weighting parameter.

10 Jacobian Matrix and Sigmoid Functions

The Jacobian matrix is applied to Sigmoid functions when they are used as activation functions in neural networks. The simplicity of the Sigmoid's Jacobian is one of the features that historically made it a popular choice for activation layers.

10.1 Element-Wise Nature

The Sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$, is applied element-wise to the input vector $\mathbf{z}$ of a layer. If the input to the activation function is a vector $\mathbf{z} = (z_1, z_2, \ldots, z_n)$, the output $\mathbf{a}$ is also a vector:

$$\mathbf{a} = \begin{bmatrix} \sigma(z_1) \\ \sigma(z_2) \\ \vdots \\ \sigma(z_n) \end{bmatrix} \tag{52}$$

Crucially, the output a_i only depends on its corresponding input z_i, not on any other input z_j (where $j \neq i$).

10.2 The Diagonal Jacobian

Because of this element-wise independence, the Jacobian matrix $\mathbf{J}$ of the Sigmoid activation function is a diagonal matrix. The off-diagonal entries (the partial derivatives of a_i with respect to z_j, where $i \neq j$) are all zero.

10.3 Derivative Term

The diagonal elements of the Jacobian are the simple scalar derivative of the Sigmoid function itself:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{53}$$

So, the Jacobian can be written as:

$$\mathbf{J}_\sigma = \text{diag}(\sigma'(\mathbf{z})) = \text{diag}(\sigma(\mathbf{z}) \odot (1 - \sigma(\mathbf{z}))) \tag{54}$$

where $\odot$ denotes the element-wise (Hadamard) product.

10.4 Backpropagation Optimisation

During backpropagation, the gradient from the subsequent layer, $\boldsymbol{\delta}_{l+1}$ (often a vector), is multiplied by the Jacobian to get the gradient for the current layer, $\boldsymbol{\delta}_l$:

$$\boldsymbol{\delta}_l = \mathbf{J}_\sigma^T \boldsymbol{\delta}_{l+1} \tag{55}$$

Since $\mathbf{J}_\sigma$ is diagonal, this matrix-vector multiplication simplifies to a much faster element-wise multiplication in practice:

$$\boldsymbol{\delta}_l = \sigma'(\mathbf{z}) \odot \boldsymbol{\delta}_{l+1} \tag{56}$$

This optimisation is a key reason why element-wise activation functions like Sigmoid, ReLU, and Tanh are computationally efficient to train. Contrast this with the Softmax function, which is element-dependent, resulting in a full, non-diagonal Jacobian and requiring a more complex matrix multiplication.

11 Jacobian Descent (JD)

Jacobian Descent (JD) is a newer optimisation technique that uses the full Jacobian matrix to find a parameter update direction that balances all objectives in multi-objective optimisation problems. This is particularly useful when objectives conflict.

In multi-objective optimisation, we have:

$$\mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})) \tag{57}$$

The Jacobian $\mathbf{J_F}$ is composed of the gradients of all m objective functions. Jacobian Descent uses this full matrix to find update directions that prevent any single objective from degrading too much, even when objectives conflict.

The update rule for Jacobian Descent can be formulated as:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \mathbf{J}_\mathbf{F}^T \mathbf{d} \tag{58}$$

where $\mathbf{d}$ is a direction vector that balances the objectives.

12 Differentiable Programming

Differentiable Programming (DP) represents a paradigm shift that generalises the principles underlying deep learning to a broader class of computational problems. While neural networks are a specific application, differentiable programming extends the use of automatic differentiation and gradient-based optimisation to arbitrary programs containing differentiable operations.

12.1 From Neural Networks to Differentiable Programs

The evolution from classical analysis to modern computing can be traced as:

1. **Classical Analysis**: The Gaussian integral problem (Section 2) required the Jacobian determinant for coordinate transformations.
2. **Multivariable Calculus**: The Jacobian matrix (Section 3) generalised derivatives to vector-valued functions.
3. **Neural Networks**: Backpropagation (Section 8) applied the Jacobian through the chain rule for training.
4. **Differentiable Programming**: The Jacobian becomes the universal tool for making arbitrary computational pipelines optimisable.

12.2 The Jacobian as Universal Differentiator

In differentiable programming, any computation that can be expressed as a composition of differentiable operations has a well-defined Jacobian. This includes:

- **Physics Simulations**: Differentiable physics engines use Jacobians to optimise physical parameters (mass, friction) by backpropagating through simulation steps.
- **Graphics and Rendering**: Differentiable renderers compute gradients of rendered images with respect to scene parameters (camera position, object geometry, lighting).
- **Probabilistic Inference**: Automatic differentiation enables gradient-based inference in complex probabilistic models.
- **Scientific Computing**: Solving differential equations with learned components requires Jacobians through both the solver and the learned functions.

12.3 Mathematical Framework

A differentiable program can be viewed as a composition of functions:

$$\mathbf{y} = f_L \circ f_{L-1} \circ \cdots \circ f_2 \circ f_1(\mathbf{x}, \boldsymbol{\theta}) \tag{59}$$

where each f_i is differentiable with respect to its inputs and parameters $\boldsymbol{\theta}$.

The gradient of the output with respect to parameters is computed via the chain rule:

$$\frac{\partial \mathbf{y}}{\partial \boldsymbol{\theta}} = \frac{\partial f_L}{\partial f_{L-1}} \cdot \frac{\partial f_{L-1}}{\partial f_{L-2}} \cdots \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial \boldsymbol{\theta}} \tag{60}$$

Each term $\frac{\partial f_i}{\partial f_{i-1}}$ is a Jacobian matrix, making the entire computation a product of Jacobians—exactly as in neural network backpropagation, but applied to arbitrary computational graphs.

12.4 Automatic Differentiation Systems

Modern automatic differentiation (autodiff) frameworks like TensorFlow, PyTorch, and JAX implement differentiable programming by:

1. **Building Computational Graphs**: Recording operations as a directed acyclic graph (DAG).
2. **Forward Pass**: Computing outputs by traversing the graph forward.
3. **Backward Pass**: Computing gradients by traversing backward, multiplying Jacobian matrices at each node.
4. **Jacobian-Vector Products**: For efficiency, most systems compute Jacobian-vector products $\mathbf{Jv}$ rather than full Jacobian matrices.

12.5 Key Differences from Traditional Programming

Aspect	**Traditional**	**Differentiable**
Operations	Any	Must be differentiable
Control flow	Arbitrary branches	Limited (or differentiable)
Optimisation	Manual tuning	Gradient-based
Debugging	Output correctness	Gradient correctness

12.6 Applications Beyond Deep Learning

12.6.1 Differentiable Physics

Physics simulations traditionally solve forward problems (given parameters, predict behaviour). Differentiable physics solves inverse problems: given desired behaviour, find parameters.

Example: Robot control optimisation. The Jacobian of the simulator output (robot trajectory) with respect to control inputs enables gradient-based policy search.

12.6.2 Differentiable Rendering

Rendering converts 3D scene parameters to 2D images. Differentiable rendering computes:

$$\frac{\partial \text{Image}}{\partial \text{Scene Parameters}} \tag{61}$$

This Jacobian enables fitting 3D models to 2D observations, enabling applications like 3D reconstruction and inverse graphics.

12.6.3 Program Synthesis

Differentiable interpreters allow learning programs through gradient descent. The Jacobian of program output with respect to program parameters (weights, structure) makes traditional discrete program spaces continuous and optimisable.

12.7 Challenges and Research Directions

1. **Discrete Operations**: Many algorithms involve non-differentiable operations (argmax, sorting). Research focuses on differentiable relaxations.

2. **Memory Efficiency**: Storing intermediate values for backpropagation can be memory-intensive. Gradient checkpointing and reversible architectures address this.

3. **Numerical Stability**: Long chains of Jacobian multiplications can lead to numerical issues (vanishing/exploding gradients).

4. **Higher-Order Derivatives**: Some applications require Hessians (second-order) or beyond, requiring nested autodiff.

12.8 Connection to This Document's Themes

Differentiable programming demonstrates how the Jacobian matrix—first encountered in the 18th-century Gaussian integral problem—has become the mathematical foundation for making entire computer programs "learnable":

- The **coordinate transformation** insight from Section 2 generalises to transforming any computational operation.
- The **chain rule** from multivariable calculus (Section 3) becomes the core algorithm for autodiff systems.
- The **backpropagation** technique from neural networks (Section 8) extends to arbitrary computational graphs.
- The **optimisation methods** (Sections 6–10) apply unchanged to differentiable programs.

Differentiable programming represents the ultimate generalisation: not just differentiable functions, but differentiable computation itself, all enabled by the humble Jacobian matrix.

13 Further Works and Open Problems

Several open problems and areas of active research remain:

1. **Computational Efficiency**: While automatic differentiation makes Jacobian computation feasible, computing full Jacobians for very large networks remains computationally expensive. Research continues into more efficient approximations and sparse Jacobian representations.

2. **Second-Order Methods**: Extending Jacobian-based methods to incorporate second-order information (Hessian) efficiently for large-scale deep learning remains challenging.

3. **Adversarial Robustness**: Understanding how to use Jacobian information to build more robust networks that are resistant to adversarial attacks is an active area of research.

4. **Neural ODEs**: The connection between continuous-time neural networks (Neural ODEs) and the Jacobian provides new perspectives on network dynamics and training stability.

5. **Implicit Neural Representations**: Using Jacobian constraints in loss functions for implicit neural representations (INRs) to ensure smoothness and differentiability.

6. **Multi-Objective Learning**: Developing more sophisticated Jacobian-based methods for multi-objective optimisation in deep learning, particularly for tasks with competing objectives.

7. **Differentiable Programming**: Extending Jacobian-based techniques to broader computational pipelines beyond neural networks, including differentiable physics, rendering, and program synthesis.

14 Conclusions

The Jacobian matrix serves as a fundamental bridge between classical mathematics and modern machine learning. From its origins in solving the Gaussian integral through coordinate transformations, to its central role in backpropagation and neural network training, the Jacobian demonstrates the deep connections between mathematical theory and practical applications.

Key takeaways:

- The Jacobian determinant enables elegant solutions to otherwise intractable integrals through coordinate transformations.
- The Jacobian matrix generalises the concept of the derivative to multivariable and vector-valued functions.
- In deep learning, the Jacobian is the mathematical foundation of backpropagation, enabling efficient gradient computation through the chain rule.
- The structure of the Jacobian (e.g., diagonal for element-wise activations) provides computational optimisations that make modern deep learning feasible.
- Advanced techniques like JENN and Jacobian Descent demonstrate how explicit use of Jacobian information can improve model performance and training efficiency.
- Differentiable programming extends the Jacobian framework beyond neural networks to arbitrary computational pipelines, enabling gradient-based optimisation of physics simulations, rendering engines, and program synthesis.
- The Jacobian continues to inspire new research directions in optimisation, robustness, and neural network design.

As deep learning continues to evolve, the Jacobian matrix remains a cornerstone of the mathematical framework that makes these advances possible. Understanding its theory and applications is essential for both theoretical research and practical implementation of modern machine learning systems.

A Example: Finding Extrema of a Function

The Jacobian matrix is primarily used in multivariable calculus for coordinate transformations and for finding the derivative of a vector-valued function. While the Jacobian matrix itself is not directly used to find the maximum or minimum of a function, its first-order counterpart, the gradient vector (which is the transpose of the Jacobian matrix of a scalar function), is essential in this process.

The search for maximum or minimum values (extrema) of a function is better exemplified by using the gradient vector (first derivative test) and the Hessian matrix (second derivative test). Let us work through a concrete example.

A.1 Problem Statement

Find the critical points and determine the nature of the extrema (maximum, minimum, or saddle point) for the function:

$$f(x, y) = x^2 + 4y^2 - 11x + 8y - 16 \tag{62}$$

A.2 Gradient Calculation (Related to the Jacobian)

The function $f(x, y)$ is a scalar function of two variables, so its first derivative is the gradient vector, $\nabla f(x, y)$, which is the transpose of the Jacobian matrix of f.

The Jacobian of a scalar function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is a $1 \times n$ matrix of its partial derivatives:

$$\mathbf{J}_f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{pmatrix} \tag{63}$$

For our function $f(x, y)$, the gradient (and the transpose of the Jacobian) is:

$$\nabla f(x, y) = \mathbf{J}_f(x, y)^T = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \tag{64}$$

Computing the partial derivatives:

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}(x^2 + 4y^2 - 11x + 8y - 16) = 2x - 11 \tag{65}$$

$$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}(x^2 + 4y^2 - 11x + 8y - 16) = 8y + 8 \tag{66}$$

Therefore, the gradient vector is:

$$\nabla f(x, y) = \begin{pmatrix} 2x - 11 \\ 8y + 8 \end{pmatrix} \tag{67}$$

A.3 Finding Critical Points (First Derivative Test)

To find critical points, we set the gradient equal to the zero vector:

$$\nabla f(x, y) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{68}$$

This gives us a system of equations:

$$2x - 11 = 0 \implies 2x = 11 \implies x = \frac{11}{2} \tag{69}$$

$$8y + 8 = 0 \implies 8y = -8 \implies y = -1 \tag{70}$$

The only critical point is $\left(\frac{11}{2}, -1\right)$.

A.4 The Hessian Matrix (Second Derivative Test)

The Hessian matrix, $\mathbf{H}$, is used to determine if a critical point is a local maximum, local minimum, or a saddle point. The Hessian is the matrix of second-order partial derivatives, which can be thought of as the Jacobian of the gradient vector (or the Jacobian of the Jacobian's transpose).

For a function $f(x, y)$, the Hessian matrix is:

$$\mathbf{H}(x, y) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} \tag{71}$$

Computing the second-order partial derivatives:

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}(2x - 11) = 2 \tag{72}$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}(8y + 8) = 8 \tag{73}$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x}(8y + 8) = 0 \tag{74}$$

$$\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial y}(2x - 11) = 0 \tag{75}$$

Therefore, the Hessian matrix is:

$$\mathbf{H}(x, y) = \begin{pmatrix} 2 & 0 \\ 0 & 8 \end{pmatrix} \tag{76}$$

Since the Hessian is constant, the second derivative test results will be the same for all points, including the critical point $\left(\frac{11}{2}, -1\right)$.

A.5 Determining the Extrema

We use the determinant of the Hessian, $\det(\mathbf{H})$, and the element H_{11} (or $\frac{\partial^2 f}{\partial x^2}$) to classify the critical point.

First, calculate the determinant:

$$\det(\mathbf{H}) = (2)(8) - (0)(0) = 16 \tag{77}$$

Now apply the second derivative test at the critical point $\left(\frac{11}{2}, -1\right)$:

- If $\det(\mathbf{H}) > 0$ and $H_{11} > 0$: **Local Minimum**
- If $\det(\mathbf{H}) > 0$ and $H_{11} < 0$: **Local Maximum**
- If $\det(\mathbf{H}) < 0$: **Saddle Point**
- If $\det(\mathbf{H}) = 0$: Test is inconclusive

In our case:

- $\det(\mathbf{H}) = 16 > 0$
- $H_{11} = 2 > 0$

Therefore, the function has a **Local Minimum** at the point $\left(\frac{11}{2}, -1\right)$.
The value of the minimum is:

$$f\left(\frac{11}{2}, -1\right) = \left(\frac{11}{2}\right)^2 + 4(-1)^2 - 11\left(\frac{11}{2}\right) + 8(-1) - 16 \tag{78}$$

$$= \frac{121}{4} + 4 - \frac{121}{2} - 8 - 16 \tag{79}$$

$$= \frac{121}{4} + \frac{16}{4} - \frac{242}{4} - \frac{32}{4} - \frac{64}{4} \tag{80}$$

$$= \frac{121 + 16 - 242 - 32 - 64}{4} \tag{81}$$

$$= \frac{-201}{4} \tag{82}$$

This example demonstrates how the gradient (related to the Jacobian) and the Hessian matrix work together to find and classify extrema of multivariable functions, providing a concrete illustration of the optimisation principles discussed in this document.

Contact Information

- **LinkedIn**: `https://www.linkedin.com/in/nguyenvuhung/`
- **GitHub**: `https://github.com/vuhung16au/`
- **Repo**: `https://github.com/vuhung16au/math-olympiad-ml/tree/main/JacobianMathsToDeepLear`