Day 21: The Chain Rule for Jacobians: The Engine of Deep Learning

How Matrix Calculus Powers Backpropagation and Modern AI

1. The Fundamental Principle: Composing Transformations

In deep learning, we rarely work with simple functions. Instead, we compose multiple transformations:

$$\mathbf{y} = \mathbf{f}_L(\mathbf{f}_{L-1}(\cdots \mathbf{f}_2(\mathbf{f}_1(\mathbf{x}))))$$

where each $\mathbf{f}_i$ represents a layer in a neural network. To compute how the output $\mathbf{y}$ changes with respect to the input $\mathbf{x}$, we need the chain rule for Jacobians.

2. Mathematical Formulation

For two differentiable functions:

$$\mathbf{z} = \mathbf{f}(\mathbf{x}) \quad \text{where } \mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$$

$$\mathbf{y} = \mathbf{g}(\mathbf{z}) \quad \text{where } \mathbf{g} : \mathbb{R}^m \to \mathbb{R}^p$$

The composition is:

$$\mathbf{y} = \mathbf{g}(\mathbf{f}(\mathbf{x}))$$

The Jacobian of the composition is given by the matrix product:

$$J_{\mathbf{y}}(\mathbf{x}) = J_{\mathbf{g}}(\mathbf{z}) \cdot J_{\mathbf{f}}(\mathbf{x})$$

where:

- $J_{\mathbf{f}}(\mathbf{x})$ is an $m \times n$ matrix
- $J_{\mathbf{g}}(\mathbf{z})$ is a $p \times m$ matrix
- $J_{\mathbf{y}}(\mathbf{x})$ is a $p \times n$ matrix

3. A Detailed Example: Step-by-Step Calculation

Let's consider two functions:

$$\mathbf{f}(x, y) = \begin{bmatrix} x^2 + y \\ 2xy \end{bmatrix}, \quad \mathbf{g}(u, v) = \begin{bmatrix} u + v \\ u^2 \\ \sin(v) \end{bmatrix}$$

The composition is:

$$\mathbf{y} = \mathbf{g}(\mathbf{f}(x, y)) = \mathbf{g}(x^2 + y, 2xy)$$

Step 1: Compute Individual Jacobians

First, compute $J_{\mathbf{f}}(x, y)$:

$$J_{\mathbf{f}}(x, y) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x & 1 \\ 2y & 2x \end{bmatrix}$$

Next, compute $J_{\mathbf{g}}(u, v)$:

$$J_{\mathbf{g}}(u, v) = \begin{bmatrix} \frac{\partial g_1}{\partial u} & \frac{\partial g_1}{\partial v} \\ \frac{\partial g_2}{\partial u} & \frac{\partial g_2}{\partial v} \\ \frac{\partial g_3}{\partial u} & \frac{\partial g_3}{\partial v} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2u & 0 \\ 0 & \cos(v) \end{bmatrix}$$

Step 2: Apply the Chain Rule

At a specific point, say $(x, y) = (1, 2)$:

$$\mathbf{z} = \mathbf{f}(1, 2) = \begin{bmatrix} 1^2 + 2 \\ 2 \cdot 1 \cdot 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Now compute the Jacobian of the composition:

$$J_{\mathbf{y}}(1, 2) = J_{\mathbf{g}}(3, 4) \cdot J_{\mathbf{f}}(1, 2)$$

First, compute $J_{\mathbf{g}}(3, 4)$:

$$J_{\mathbf{g}}(3, 4) = \begin{bmatrix} 1 & 1 \\ 2 \cdot 3 & 0 \\ 0 & \cos(4) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 6 & 0 \\ 0 & \cos(4) \end{bmatrix}$$

Then compute $J_{\mathbf{f}}(1, 2)$:

$$J_{\mathbf{f}}(1, 2) = \begin{bmatrix} 2 \cdot 1 & 1 \\ 2 \cdot 2 & 2 \cdot 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}$$

Finally, multiply the matrices:

$$J_{\mathbf{y}}(1,2) = \begin{bmatrix} 1 & 1 \\ 6 & 0 \\ 0 & \cos(4) \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} 1\cdot 2 + 1\cdot 4 & 1\cdot 1 + 1\cdot 2 \\ 6\cdot 2 + 0\cdot 4 & 6\cdot 1 + 0\cdot 2 \\ 0\cdot 2 + \cos(4)\cdot 4 & 0\cdot 1 + \cos(4)\cdot 2 \end{bmatrix} = \begin{bmatrix} 6 & 3 \\ 12 & 6 \\ 4\cos(4) & 2\cos(4) \end{bmatrix}$$

4. Connection to Backpropagation

Backpropagation is essentially the repeated application of the chain rule for Jacobians. In a neural network with layers $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_L$, the gradient of the loss $\mathcal{L}$ with respect to the input is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = J_{\mathcal{L}}(\mathbf{f}_L) \cdot J_{\mathbf{f}_L}(\mathbf{f}_{L-1}) \cdots J_{\mathbf{f}_2}(\mathbf{f}_1) \cdot J_{\mathbf{f}_1}(\mathbf{x})$$

Why This is Efficient

- Each layer only needs to compute its local Jacobian
- The chain rule allows gradients to be propagated backward without recomputing forward passes
- This makes training deep networks computationally feasible

5. Special Case: Scalar Output (The Gradient)

When the final output is a scalar (e.g., a loss function $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$), the Jacobian becomes a row vector (the gradient). For a composition $\mathcal{L}(\mathbf{f}(\mathbf{x}))$, the chain rule becomes:

$$\nabla_{\mathbf{x}} \mathcal{L} = J_{\mathbf{f}}(\mathbf{x})^{\top} \cdot \nabla_{\mathbf{f}} \mathcal{L}$$

This is the form most commonly used in gradient descent algorithms.

6. Practical Considerations in Deep Learning

Implementation Details

- **Memory Efficiency:** Backpropagation requires storing intermediate values for gradient computation
- **Numerical Stability:** Care must be taken to avoid numerical issues in matrix multiplication
- **Parallelization:** Jacobian computations can be parallelized across layers and batch elements
- **Automatic Differentiation:** Frameworks like PyTorch and TensorFlow automate this process

7. Advanced Topics

7.1 Higher-Order Chain Rules

For second derivatives, we need more complex chain rules involving Hessian matrices:

$$H_{\mathbf{g}\circ\mathbf{f}}(\mathbf{x}) = J_{\mathbf{f}}(\mathbf{x})^\top \cdot H_{\mathbf{g}}(\mathbf{f}(\mathbf{x})) \cdot J_{\mathbf{f}}(\mathbf{x}) + \sum_i \frac{\partial \mathbf{g}}{\partial z_i} \cdot H_{f_i}(\mathbf{x})$$

This is used in second-order optimization methods.

7.2 Implicit Function Theorem

For functions defined implicitly by equations like $\mathbf{F}(\mathbf{x}, \mathbf{y}) = \mathbf{0}$, the Jacobian can be computed using:

$$J_{\mathbf{y}}(\mathbf{x}) = -\left[\frac{\partial \mathbf{F}}{\partial \mathbf{y}}\right]^{-1} \cdot \frac{\partial \mathbf{F}}{\partial \mathbf{x}}$$

This is useful for equilibrium models and implicit layers.

8. Exercises for Understanding

1. For $\mathbf{f}(x, y) = \begin{bmatrix} e^x + y \\ x^2 y \end{bmatrix}$ and $\mathbf{g}(u, v) = \begin{bmatrix} uv \\ u + v^2 \end{bmatrix}$, compute $J_{\mathbf{g}\circ\mathbf{f}}(1, 1)$
2. Derive the chain rule for the composition of three functions $\mathbf{h} \circ \mathbf{g} \circ \mathbf{f}$

3. Explain why the chain rule for Jacobians involves matrix multiplication rather than elementwise multiplication

4. For a neural network with softmax output and cross-entropy loss, write the chain rule for the gradient with respect to the input

Key Takeaway

The chain rule for Jacobians is the mathematical foundation of backpropagation and deep learning. It provides a systematic way to compute derivatives of composed functions by breaking them down into simpler parts. This enables efficient training of complex neural networks with millions of parameters. Understanding this principle is essential for developing new architectures, debugging training issues, and advancing the field of deep learning.