Day 22: Backpropagation with Jacobians: The Algorithm That Powers Deep Learning

How the Chain Rule of Jacobians Enables Efficient Training of Neural Networks

1. The Essence of Backpropagation

Backpropagation is the fundamental algorithm that enables efficient training of deep neural networks. At its core, it's an application of the chain rule for Jacobians to compute gradients of the loss function with respect to all parameters in the network.

The key insight is that we can compute these gradients by: 1. Propagating inputs forward through the network (forward pass) 2. Propagating error gradients backward through the network (backward pass) 3. Using local Jacobians at each layer to compute parameter updates

2. A Detailed Example: Two-Layer Neural Network

Let's consider a neural network with one hidden layer for regression:

$$
\begin{aligned}
z &= Wx + b && \text{(linear transformation)} \\
h &= \sigma(z) && \text{(elementwise activation)} \\
\hat{y} &= Uh + c && \text{(output layer)} \\
L &= \frac{1}{2}\|\hat{y} - y\|^2 && \text{(squared error loss)}
\end{aligned}
$$

Where:

- $x \in \mathbb{R}^n$ is the input
- $W \in \mathbb{R}^{m \times n}$ is the weight matrix of the first layer
- $b \in \mathbb{R}^m$ is the bias vector of the first layer
- σ is an elementwise activation function (e.g., ReLU, sigmoid)
- $U \in \mathbb{R}^{k \times m}$ is the weight matrix of the output layer
- $c \in \mathbb{R}^k$ is the bias vector of the output layer
- $\hat{y} \in \mathbb{R}^k$ is the predicted output
- $y \in \mathbb{R}^k$ is the target output

3. Step-by-Step Backpropagation

We'll compute the gradients using the chain rule for Jacobians.

Step 1: Gradient of Loss with Respect to Output

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$

Step 2: Gradient for Output Layer Parameters

Using the chain rule:

$$\frac{\partial L}{\partial U} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial U} = (\hat{y} - y)h^\top$$

$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial c} = \hat{y} - y$$

Step 3: Gradient for Hidden Layer Activation

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h} = U^\top(\hat{y} - y)$$

Step 4: Gradient for Pre-Activation

Since $h = \sigma(z)$ is applied elementwise:

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial h} \odot \sigma'(z)$$

Where $\odot$ denotes elementwise multiplication and $\sigma'(z)$ is the derivative of the activation function.

Step 5: Gradient for First Layer Parameters

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial W} = \left(\frac{\partial L}{\partial z}\right) x^\top$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b} = \frac{\partial L}{\partial z}$$

4. The Jacobian Perspective

Each step in backpropagation involves multiplying Jacobians:

Jacobians in Backpropagation

- $\frac{\partial \hat{y}}{\partial U}$ is a 3D tensor, but its product with $\frac{\partial L}{\partial \hat{y}}$ simplifies to the outer product $(\hat{y}-y)h^\top$
- $\frac{\partial \hat{y}}{\partial h} = U$ (a matrix)
- $\frac{\partial h}{\partial z} = \text{diag}(\sigma'(z))$ (a diagonal matrix)
- $\frac{\partial z}{\partial W}$ is a 3D tensor, but its product with $\frac{\partial L}{\partial z}$ simplifies to the outer product $\left(\frac{\partial L}{\partial z}\right) x^\top$

5. Computational Efficiency of Backpropagation

Backpropagation is efficient because:

- It reuses computations from the forward pass
- It computes gradients in time proportional to the forward pass (O(n))
- It avoids recomputing derivatives from scratch for each parameter
- It leverages the chain rule to break down complex derivatives into simpler local computations

6. General Formulation for Deep Networks

For a deep network with layers $1, 2, \ldots, L$, the backpropagation algorithm can be summarized as:

1. **Forward pass:** Compute and store $z^{(l)}$, $h^{(l)}$ for each layer
2. **Backward pass:** For $l = L$ down to 1:

$$\begin{aligned}
\frac{\partial L}{\partial z^{(l)}} &= \frac{\partial L}{\partial h^{(l)}} \odot \sigma'(z^{(l)}) \\
\frac{\partial L}{\partial W^{(l)}} &= \frac{\partial L}{\partial z^{(l)}} (h^{(l-1)})^\top \\
\frac{\partial L}{\partial b^{(l)}} &= \frac{\partial L}{\partial z^{(l)}} \\
\frac{\partial L}{\partial h^{(l-1)}} &= (W^{(l)})^\top \frac{\partial L}{\partial z^{(l)}}
\end{aligned}$$

7. Implementation in Modern Frameworks

Modern deep learning frameworks like PyTorch and TensorFlow automate backpropagation through:

Automatic Differentiation

- **Computational graphs:** Frameworks build a graph of operations during the forward pass
- **Local gradients:** Each operation knows how to compute its local Jacobian
- **Chain rule application:** Frameworks automatically apply the chain rule to compute gradients
- **Memory management:** Frameworks manage storing and reusing intermediate values

8. Practical Considerations and Challenges

Backpropagation in Practice

- **Vanishing/exploding gradients:** Gradients can become very small or very large in deep networks
- **Memory usage:** Storing intermediate values for backpropagation can require significant memory
- **Numerical stability:** Care must be taken to avoid numerical issues in gradient computation
- **Non-differentiable operations:** Some operations (e.g., argmax) require special handling
- **Distributed training:** Backpropagation can be parallelized across multiple devices

9. Extensions and Variations

- **Backpropagation through time (BPTT):** For recurrent neural networks
- **Implicit differentiation:** For layers defined implicitly by equations
- **Second-order methods:** Using Hessian information for optimization
- **Meta-learning:** Differentiating through optimization processes

10. Historical Context

The backpropagation algorithm was developed in the 1970s and 1980s, with key contributions from:

- Seppo Linnainmaa (1970): First description of reverse-mode automatic differentiation
- Paul Werbos (1974): Application to neural networks
- David Rumelhart, Geoffrey Hinton, and Ronald Williams (1986): Popularization in the neural network community

The algorithm gained widespread adoption in the 2010s with the deep learning revolution.

Key Takeaway

Backpropagation with Jacobians is the computational engine that powers modern deep learning. By efficiently computing gradients through the application of the chain rule to Jacobians, it enables training of neural networks with millions or even billions of parameters. Understanding this algorithm is essential for developing new architectures, debugging training issues, and advancing the field of artificial intelligence.