Day 24: Jacobian-Vector Products (JVP): The Engine of Forward-Mode Autodiff

Understanding the Complementary Approach to Gradient Computation

1. The Dual Perspective: JVP vs. VJP

In the previous days, we explored Vector-Jacobian Products (VJPs), which power reverse-mode automatic differentiation (backpropagation). Today we examine their dual: **Jacobian-Vector Products (JVPs)**, which power forward-mode automatic differentiation.

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$:

- **VJP:** $v^\top J_f(x)$ - How output sensitivities affect inputs (reverse mode)
- **JVP:** $J_f(x)v$ - How input perturbations affect outputs (forward mode)

2. Formal Definition and Interpretation

For a differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$ with Jacobian $J_f(x) \in \mathbb{R}^{m \times n}$, and a vector $v \in \mathbb{R}^n$, the Jacobian-Vector Product is:

$$\text{JVP}(f, x, v) = J_f(x)v$$

Geometric Interpretation

The JVP $J_f(x)v$ represents:

- The directional derivative of f in the direction v
- The linear approximation of how $f(x)$ changes when x is perturbed by v
- The tangent vector to the curve $f(x + tv)$ at $t = 0$

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

Let's analyze the function:

$$f(x_1, x_2) = \begin{bmatrix} x_1^2 + x_2 \\ \sin(x_1) \end{bmatrix}$$

Step 1: Compute the Jacobian

$$J_f(x_1, x_2) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 & 1 \\ \cos(x_1) & 0 \end{bmatrix}$$

Step 2: Choose a Point and Direction Vector

Let's evaluate at $x = (1, 0)$ with direction vector $v = (1, 2)$.

Step 3: Compute the JVP

$$J_f(1,0) = \begin{bmatrix} 2(1) & 1 \\ \cos(1) & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ \cos(1) & 0 \end{bmatrix}$$

$$\text{JVP} = J_f(1,0)v = \begin{bmatrix} 2 & 1 \\ \cos(1) & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 + 1 \cdot 2 \\ \cos(1) \cdot 1 + 0 \cdot 2 \end{bmatrix} = \begin{bmatrix} 4 \\ \cos(1) \end{bmatrix} \approx \begin{bmatrix} 4 \\ 0.5403 \end{bmatrix}$$

Interpretation

This result tells us that if we perturb the input $(1, 0)$ in the direction $(1, 2)$, the output will change approximately by $(4, 0.5403)$.

4. Forward-Mode Automatic Differentiation

JVPs are the computational engine of forward-mode automatic differentiation:

How Forward-Mode Works

- Propagate a direction vector v forward through the computation
- At each operation, compute the local JVP
- The final result is $J_f(x)v$
- To get the full Jacobian, repeat for $v = e_1, e_2, \ldots, e_n$ (standard basis vectors)

5. When to Use JVP vs. VJP

Choosing the Right Autodiff Mode

- **Use JVP (Forward-Mode) when:** $n \ll m$
  (few inputs, many outputs - e.g., sensitivity analysis)
- **Use VJP (Reverse-Mode) when:** $n \gg m$
  (many inputs, few outputs - e.g., neural network training)
- **Complexity:**
  - JVP: $O(n)$ evaluations to compute full Jacobian
  - VJP: $O(m)$ evaluations to compute full Jacobian

6. Applications in AI and Scientific Computing

Where JVPs Excel

- **Sensitivity Analysis:** Understanding how input variations affect complex models
- **Physics-Informed ML:** Embedding physical constraints through derivatives
- **Hessian-Vector Products:**

  $$Hv = J_{\nabla f}(x)v$$

  Useful for second-order optimization without explicit Hessian
- **Real-Time Optimization:** When input dimension is small and fixed
- **Scientific Computing:** Tangent linear models in climate science and fluid dynamics

7. Implementation in Modern Frameworks

Most deep learning frameworks support both modes:

- **PyTorch:** `torch.autograd.forward_ad` for forward-mode
- **JAX:** `jax.jvp` for Jacobian-vector products
- **TensorFlow:** `tf.autodiff.ForwardAccumulator`

8. Historical Context

Forward-mode autodiff was actually developed first:

- **1950s-60s:** Early work on automatic differentiation
- **1970s:** Development of both forward and reverse modes
- **1980s:** Reverse-mode gained popularity for neural networks
- **Recent:** Renewed interest in forward-mode for specific applications

9. Exercises for Understanding

1. For $f(x, y) = \begin{bmatrix} x^2 y \\ e^x + y \\ \sin(xy) \end{bmatrix}$, compute the JVP with $v = (1, -1)$ at $(1, 1)$
2. Explain why forward-mode would be preferable to reverse-mode for a function $f : \mathbb{R}^3 \to \mathbb{R}^{100}$
3. Implement a simple forward-mode autodiff function in pseudocode

4. Compute the JVP for a simple neural network layer $f(x) = \sigma(Wx + b)$

Key Takeaway

Jacobian-Vector Products provide the fundamental computation for forward-mode automatic differentiation. While VJPs power reverse-mode (backpropagation) and excel when we have many inputs and few outputs, JVPs power forward-mode and excel when we have few inputs and many outputs. Understanding both approaches provides a complete toolkit for efficient gradient computation across diverse applications in machine learning and scientific computing.