Lecture 8

Saurabh

Abstract

This guide aims to demystify backpropagation and linearization, two foundational concepts in machine learning. We will move from high-level intuition to a concrete, step-by-step mathematical understanding. Our goal is to show not just *what* the formulas are, but *why* they are what they are, in a way that is clear, simple, and memorable for any student.

The Core Idea: How Do Machines Learn?

Imagine you're playing a game of darts. Your first throw is off the mark. To improve, your brain subconsciously performs a series of calculations. It sees *how far* off you were and in which direction (this is the **error**), and you adjust your stance, aim, angle, and force for the next throw. You are learning from your error.

In machine learning, this process is formalized and given mathematical rigor:

- **The Throw:** The model makes a prediction based on its current state.
- **The Error:** A **loss function** measures how far the prediction is from the actual target. A high loss means a bad prediction; a low loss means a good one.
- **The Adjustment:** The model must adjust its internal **parameters** (weights and biases) to reduce this error for the next prediction.

The key question is: *how exactly should it adjust them?* If we have millions of parameters, which ones do we change, and by how much?

This is where the **gradient** comes in. Think of the loss function as a hilly landscape, where your goal is to get to the lowest valley (minimum loss). The gradient at your current position is a vector that points in the direction of the *steepest ascent*—the fastest way to go uphill. To reduce the error, we simply take a small step in the **exact opposite direction** of the gradient.

This entire guide is about **Backpropagation**, the powerful and brilliantly efficient algorithm that calculates these necessary gradients for every single parameter in the model, telling us exactly how to "adjust our aim."

1. The Challenge: Differentiation is Hard

For simple functions, finding a derivative is easy. But what about a complex function composed of many nested parts, like the one from the lecture?

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2))$$

Manually deriving this using the standard chain rule gives a very long and messy expression. Implementing this by hand is not only slow but also extremely prone to errors. Now, consider a real-world neural network like GPT-3, which has *billions* of parameters. A manual approach is not just impractical; it's fundamentally impossible. We need an automated, efficient, and scalable method.

2. The Solution: Automatic Differentiation

Instead of tackling the whole function at once, we can break it down into a series of simple, elementary operations whose derivatives we already know. This creates a roadmap for the calculation, which we call a **computation graph**.

2.1 The Forward Pass: Building the Function Step-by-Step

Let's decompose our function. This process is called the **forward pass**, where we start with an input and compute the final value step-by-step. Let's make this concrete with a numerical example: let $x = 2$.

- Start with input $x = 2$
- Let $a = x^2 \quad \rightarrow \quad a = 2^2 = 4$
- Let $b = \exp(a) \quad \rightarrow \quad b = \exp(4) \approx 54.6$
- Let $c = a + b \quad \rightarrow \quad c = 4 + 54.6 = 58.6$
- Let $d = \sqrt{c} \quad \rightarrow \quad d = \sqrt{58.6} \approx 7.66$
- Let $e = \cos(c) \quad \rightarrow \quad e = \cos(58.6) \approx 0.83$
- Finally, $f = d + e \quad \rightarrow \quad f = 7.66 + 0.83 = 8.49$

This sequence is visualized in the computation graph below. The green values show the result of the forward pass.

2.2 The Backward Pass: The Magic of the Chain Rule

Now, we find $\frac{df}{dx}$ by moving **backward** through the graph, starting from the end. The core idea is simple: at each node, we compute its gradient, which represents that node's total influence on the final output f. We start with the derivative of the output with respect to itself, which is trivially 1 ($\frac{df}{df} = 1$).

Then, for any node u that is an input to a node v, the chain rule tells us:

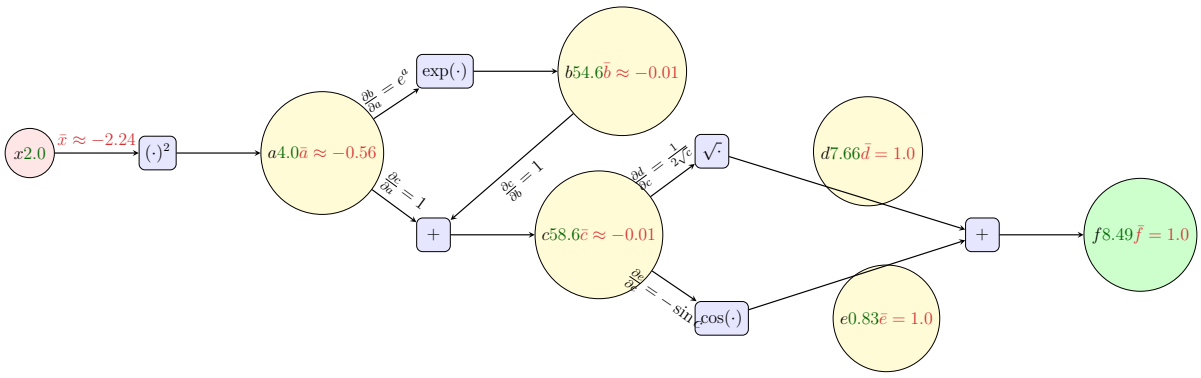
$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial u}$$

If a node u feeds into multiple other nodes $(v_1, v_2, \dots)$, its total influence is the sum of its influences through each path (multivariate chain rule):

$$\frac{\partial f}{\partial u} = \sum_i \frac{\partial f}{\partial v_i} \frac{\partial v_i}{\partial u}$$

This reuse of previously computed "downstream" gradients ($\frac{\partial f}{\partial v_i}$) is what makes back-propagation so efficient.

Let's apply this to our example. We'll denote $\frac{df}{dy}$ as $\bar{y}$. We start with $\bar{f} = 1$. The red values on the graph show the gradients calculated during the backward pass.



Backward Pass Step-by-Step Calculation:

1. $\bar{f} = 1$ (by definition).
2. $f = d + e \implies \frac{\partial f}{\partial d} = 1, \frac{\partial f}{\partial e} = 1$. $\bar{d} = \bar{f}\frac{\partial f}{\partial d} = 1 \cdot 1 = 1$. $\bar{e} = \bar{f}\frac{\partial f}{\partial e} = 1 \cdot 1 = 1$.
3. $d = \sqrt{c}, e = \cos(c)$. Node c receives gradient from its two children, d and e. $\bar{c} = \bar{d}\frac{\partial d}{\partial c} + \bar{e}\frac{\partial e}{\partial c} = 1 \cdot \frac{1}{2\sqrt{c}} + 1 \cdot (-\sin c) \approx \frac{1}{15.32} - 0.075 \approx 0.065 - 0.075 = -0.01$.

4. $c = a + b \implies \frac{\partial c}{\partial a} = 1, \frac{\partial c}{\partial b} = 1$. The gradient flows back from c to b: $\bar{b} = \bar{c}\frac{\partial c}{\partial b} = -0.01 \cdot 1 = -0.01$.

5. Node a is a parent to both b (via the exp operation) and c (via the addition). So, it receives gradient from both. $\bar{a} = \bar{b}\frac{\partial b}{\partial a} + \bar{c}\frac{\partial c}{\partial a} = \bar{b} \cdot (\exp a) + \bar{c} \cdot (1) \approx (-0.01) \cdot (54.6) + (-0.01) \cdot 1 \approx -0.546 - 0.01 = -0.556$.

6. **Final step (Gradient w.r.t. input x):** The gradient flows from a back to x. $\bar{x} = \bar{a}\frac{\partial a}{\partial x} = (-0.556) \cdot (2x) = -0.556 \cdot 4 = -2.224$.

Note on diagram values: The numerical values for the gradients in the text are derived step-by-step from the chain rule. These should be considered authoritative over the static values in the diagram if any discrepancies arise. The final result for $\frac{df}{dx}$ at $x = 2$ is approximately -2.224.

2.3 Application to Neural Networks

A neural network is just a very large, layered computation graph. Each layer performs a simple operation, and backpropagation works on this graph in exactly the same way. The output of layer i is the input to layer $i + 1$. A typical layer is:

$$\mathbf{a}_i = \sigma(W_i \mathbf{a}_{i-1} + \mathbf{b}_i)$$

where $\mathbf{a}_{i-1}$ is the input from the previous layer, $W_i, \mathbf{b}_i$ are the parameters (weights and biases) of the current layer, and σ is a non-linear activation function (like sigmoid, tanh, or ReLU).

The goal is to find the parameters $\theta = \{W_1, \mathbf{b}_1, \ldots, W_L, \mathbf{b}_L\}$ that minimize a loss function, for example, the squared error:

$$L(\theta) = ||\mathbf{y}_{true} - \mathbf{a}_L||^2$$

Backpropagation is the algorithm used to compute the gradient of this loss L with respect to every single parameter in the network ($\frac{\partial L}{\partial W_i}, \frac{\partial L}{\partial \mathbf{b}_i}$), allowing us to update them using gradient descent and train the model.

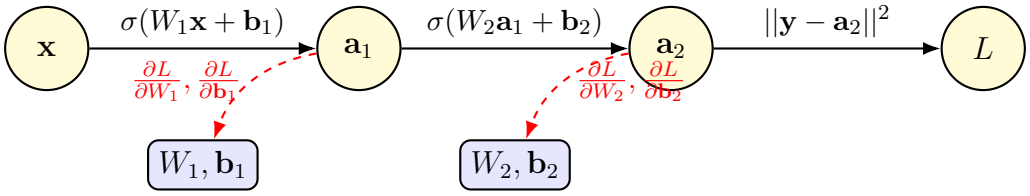


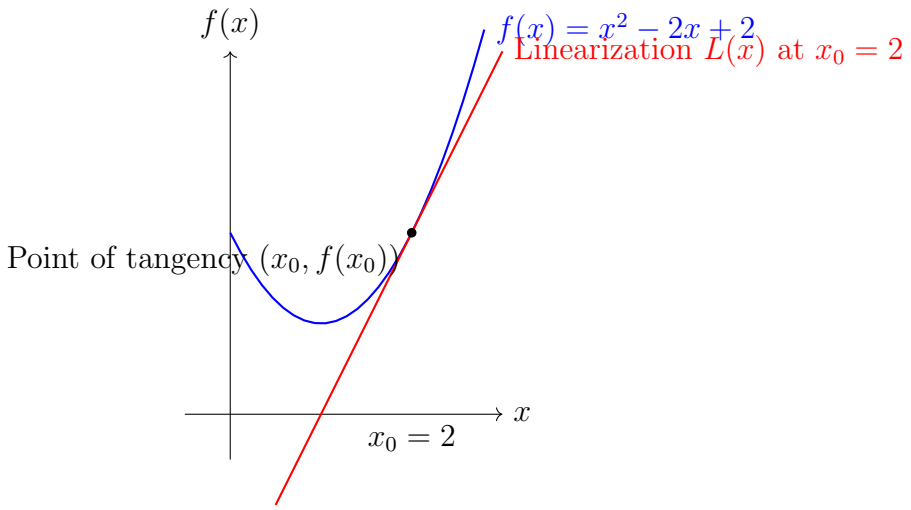
Figure 1: A cleaner view of a 2-layer neural network computation graph. The forward pass flows left-to-right through the data nodes (yellow circles). The parameters (blue rectangles) are updated via backpropagation (red dashed arrows), which computes their gradient with respect to the final loss.

3. Linearization: The Tangent Line Approximation

Often in math and engineering, we want to approximate a complex curve with a simple straight line, at least around a specific point of interest. This process is called **linearization**, and the gradient is the perfect tool for it. The linear approximation $L(\mathbf{x})$ of a function $f(\mathbf{x})$ at a point $\mathbf{x_0}$ is simply its tangent line (or plane, or hyperplane) at that point. It is given by the first two terms of the Taylor series:

$$L(\mathbf{x}) \approx f(\mathbf{x_0}) + \nabla f(\mathbf{x_0})^T(\mathbf{x} - \mathbf{x_0})$$

This approximation is very accurate near $\mathbf{x_0}$ but naturally gets worse as you move further away. We use it to simplify hard problems into easy, linear ones, which is a cornerstone of many optimization algorithms.



3.1 Linearization Examples

Example 1: Find the linearization of $f(x) = \sqrt{x^2+9}$ around $x_0 = -4$.

1. **Evaluate the function at the point:** $f(-4) = \sqrt{(-4)^2+9} = \sqrt{16+9} = \sqrt{25} = 5$.
2. **Find the derivative (gradient):** $f'(x) = \frac{1}{2\sqrt{x^2+9}} \cdot (2x) = \frac{x}{\sqrt{x^2+9}}$.
3. **Evaluate the derivative at the point:** $f'(-4) = \frac{-4}{\sqrt{(-4)^2+9}} = \frac{-4}{5}$.
4. **Assemble the linearization formula:** $L(x) = f(x_0) + f'(x_0)(x - x_0)$.

$$L(x) = 5 - \frac{4}{5}(x - (-4)) = 5 - \frac{4}{5}(x + 4)$$

Example 2: Find the linearization of $f(x, y) = e^x \cos(y)$ around the point $(x_0, y_0) = (0, 0)$.

1. **Evaluate function:** $f(0, 0) = e^0 \cos(0) = 1 \cdot 1 = 1$.
2. **Find the gradient vector:** $\nabla f = \langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \rangle = \langle e^x \cos y, -e^x \sin y \rangle$.
3. **Evaluate gradient at the point:** $\nabla f(0, 0) = \langle e^0 \cos 0, -e^0 \sin 0 \rangle = \langle 1, 0 \rangle$.
4. **Assemble linearization:** $L(x, y) = f(0, 0) + \nabla f(0, 0) \cdot \langle x - 0, y - 0 \rangle$.

$$L(x, y) = 1 + \langle 1, 0 \rangle \cdot \langle x, y \rangle = 1 + (1 \cdot x + 0 \cdot y) = 1 + x$$

4. Practice Problems

Problem 1

We define

$$g(\mathbf{z}, \nu) := \log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}, \nu)$$

$$\mathbf{z} := t(\epsilon, \nu)$$

for differentiable functions p, q, t, and $\mathbf{x} \in \mathbb{R}^D, \mathbf{z} \in \mathbb{R}^E, \nu \in \mathbb{R}^F, \epsilon \in \mathbb{R}^G$. By using the chain rule, compute the total derivative

$$\frac{d}{d\nu} g(\mathbf{z}, \nu).$$

Problem 2

Compute the derivatives $\frac{df}{d\mathbf{x}}$ of the following functions by using the chain rule. Provide the dimensions of every single partial derivative. Describe your steps in detail.

a.

$$f(z) = \log(1+z), \quad z = \mathbf{x}^\top \mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^D$$

b.

$$\mathbf{f}(\mathbf{z}) = \sin(\mathbf{z}), \quad \mathbf{z} = A\mathbf{x} + \mathbf{b}, \quad A \in \mathbb{R}^{E \times D}, \mathbf{x} \in \mathbb{R}^D, \mathbf{b} \in \mathbb{R}^E$$

where $\sin(\cdot)$ is applied to every element of $\mathbf{z}$, and the output $\mathbf{f}$ is also in $\mathbb{R}^E$.

5. Solutions to Practice Problems

Solution to Problem 1 - A More Intuitive Approach

Goal: Find the total effect that changing the parameter vector ν has on the final scalar value g.

Strategy: Think of ν as a set of control knobs. Turning these knobs has two distinct effects on g, and we must account for both:

1. **Direct Path:** ν appears directly inside the function g (within the $\log q$ term). Changing ν has an immediate, direct impact on g.
2. **Indirect Path:** ν is also used to compute the intermediate variable $\mathbf{z}$ (since $\mathbf{z} = t(\epsilon, \nu)$). Changing ν changes $\mathbf{z}$, and this new value of $\mathbf{z}$ then propagates forward to change the value of g.

To find the total derivative (the total sensitivity), we must calculate the gradient from each path and add them together. This is the essence of the multivariate chain rule.

Step-by-Step Execution: The formula for the total derivative is a sum over all paths of influence:

$$\frac{dg}{d\nu} = \underbrace{\frac{\partial g}{\partial \nu}}_{\text{Path 1: Direct influence of } \nu \text{ on } g} + \underbrace{\frac{\partial g}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \nu}}_{\text{Path 2: Indirect, via z}}$$

- **Calculate Path 1 (Direct Effect):** We find the partial derivative of g with

respect to ν, treating $\mathbf{z}$ as a constant for this step.

$$\frac{\partial g}{\partial \nu} = \frac{\partial}{\partial \nu}\left(\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}, \nu)\right) = -\frac{\partial \log q(\mathbf{z}, \nu)}{\partial \nu}$$

- **Calculate Path 2 (Indirect Effect):** This path itself has two links in the chain.
  - First, how sensitive is g to changes in $\mathbf{z}$?

$$\frac{\partial g}{\partial \mathbf{z}} = \frac{\partial \log p(\mathbf{x}, \mathbf{z})}{\partial \mathbf{z}} - \frac{\partial \log q(\mathbf{z}, \nu)}{\partial \mathbf{z}}$$

  - Second, how sensitive is $\mathbf{z}$ to changes in ν?

$$\frac{\partial \mathbf{z}}{\partial \nu} = \frac{\partial t(\epsilon, \nu)}{\partial \nu}$$

Final Answer: Now we combine everything to get the total derivative.

$$\frac{dg}{d\nu} = -\frac{\partial \log q(\mathbf{z}, \nu)}{\partial \nu} + \left(\frac{\partial \log p(\mathbf{x}, \mathbf{z})}{\partial \mathbf{z}} - \frac{\partial \log q(\mathbf{z}, \nu)}{\partial \mathbf{z}}\right)\frac{\partial t(\epsilon, \nu)}{\partial \nu}$$

Solution to Problem 2a

Goal: Find the gradient of the scalar output f with respect to the vector input $\mathbf{x}$.
Strategy: This is a two-step computation graph: $\mathbf{x} \to z \to f$. The vector $\mathbf{x}$ first gets converted to a scalar z, which then produces the final scalar f. The chain rule lets us multiply the sensitivities (derivatives) of each step.

$$\frac{df}{d\mathbf{x}} = (\text{sensitivity of } f \text{ to } z) \times (\text{sensitivity of } z \text{ to } \mathbf{x})$$

$$\frac{df}{d\mathbf{x}} = \frac{df}{dz}\frac{\partial z}{\partial \mathbf{x}}$$

Step-by-Step Execution:

1. **First Link ($\frac{df}{dz}$):** Sensitivity of the final output f to its immediate input z. This is a standard scalar derivative.

$$\frac{df}{dz} = \frac{d}{dz}\log(1+z) = \frac{1}{1+z}$$

   Dimension: Scalar (1×1).

2. **Second Link ($\frac{\partial z}{\partial \mathbf{x}}$):** Sensitivity of the intermediate scalar z to the input vector $\mathbf{x}$. This is the derivative of a scalar by a vector, which results in a

row vector. Let's derive it from scratch: $z = \mathbf{x}^\top \mathbf{x} = \sum_{i=1}^{D} x_i^2$. The k-th component of the gradient is $\frac{\partial z}{\partial x_k} = 2x_k$. Arranging these into a row vector gives $[2x_1, 2x_2, \ldots, 2x_D] = 2\mathbf{x}^\top$.

$$\frac{\partial z}{\partial \mathbf{x}} = 2\mathbf{x}^\top$$

Dimension: Row vector $(1 \times D)$.

Final Answer: We multiply the results (scalar times row vector). The dimensions $(1 \times 1) \cdot (1 \times D)$ yield a $(1 \times D)$ row vector, as expected for the gradient of a scalar function with respect to a column vector.

$$\frac{df}{d\mathbf{x}} = \left(\frac{1}{1+z}\right)(2\mathbf{x}^\top) = \frac{2\mathbf{x}^\top}{1+\mathbf{x}^\top\mathbf{x}}$$

Solution to Problem 2b

Goal: Find the derivative of the vector function $\mathbf{f}$ with respect to the vector input $\mathbf{x}$. When the input and output are both vectors, the derivative is a matrix called the **Jacobian**.

Strategy: The computation graph is $\mathbf{x} \to \mathbf{z} \to \mathbf{f}$. The chain rule for Jacobians works like matrix multiplication of the Jacobians from each step.

$$\underbrace{\frac{\partial \mathbf{f}}{\partial \mathbf{x}}}_{\text{Jacobian of } \mathbf{f} \text{ wrt } \mathbf{x}} = \underbrace{\frac{\partial \mathbf{f}}{\partial \mathbf{z}}}_{\text{Jacobian of } \mathbf{f} \text{ wrt } \mathbf{z}} \cdot \underbrace{\frac{\partial \mathbf{z}}{\partial \mathbf{x}}}_{\text{Jacobian of } \mathbf{z} \text{ wrt } \mathbf{x}}$$

Step-by-Step Execution:

1. **First Link ($\frac{\partial \mathbf{f}}{\partial \mathbf{z}}$):** This is the Jacobian of $\mathbf{f}$ with respect to $\mathbf{z}$. The key insight here is that the $\sin(\cdot)$ function is applied **element-wise**: $f_i = \sin(z_i)$. This means that f_i depends *only* on z_i, and not on any other z_j where $j \neq i$. Therefore, the partial derivative $\frac{\partial f_i}{\partial z_j}$ is 0 if $i \neq j$. The Jacobian matrix is zero everywhere except for the main diagonal.

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right)_{ij} = \frac{\partial f_i}{\partial z_j} = \begin{cases} \cos(z_i) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

This forms a diagonal matrix:

$$\frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \text{diag}(\cos(z_1), \ldots, \cos(z_E)) = \text{diag}(\cos(\mathbf{z}))$$

Dimension: Square diagonal matrix $(E \times E)$.

2. **Second Link ($\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$):** This is the Jacobian of $\mathbf{z}$ with respect to $\mathbf{x}$. For the linear transformation $\mathbf{z} = A\mathbf{x} + \mathbf{b}$, the derivative is simply the matrix A.

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = A$$

Dimension: The matrix A itself $(E \times D)$.

Final Answer: We multiply the two matrices. The order is crucial. The dimensions $(E \times E) \cdot (E \times D)$ yield a final $(E \times D)$ Jacobian matrix.

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \text{diag}(\cos(\mathbf{z})) \cdot A = \text{diag}(\cos(A\mathbf{x} + \mathbf{b})) \cdot A$$

Conclusion

We have journeyed from a simple analogy of playing darts to the formal mechanics of how machine learning models truly learn. The two key takeaways should be:

1. **Backpropagation is just the chain rule applied to a computation graph.** By breaking down complex functions into simple steps, we can efficiently compute the gradient of every parameter by passing gradient information backward through the graph.

2. **Linearization is approximation using the gradient.** It allows us to replace a complex function with its tangent line at a point, simplifying analysis and forming the basis for powerful optimization methods.

Mastering these two concepts provides a solid foundation for understanding and developing advanced machine learning and deep learning models.