Cost Functions and Gradient Descent

We've established that **Cost Functions** (like MSE) measure how well our Linear Regression model fits the data, given a set of parameters (θs or βs). The goal of training is to find the parameter values that **minimize** this cost function. But how do we actually *find* those minimum-cost parameters? This is where optimization algorithms like **Gradient Descent** come in.

Gradient Descent: Finding the Minimum Cost

**Gradient Descent** is a widely used iterative optimization technique that aims to find the minimum value of a function (in our case, the cost function). For Linear Regression, it helps us find the best possible set of β-coefficients (slopes β₁, β₂, ...) and the intercept (β₀) that define the best-fitting line/hyperplane.

How Gradient Descent Works

The core idea is intuitive: imagine you are standing on a hillside (representing the cost function surface) and want to reach the bottom (the minimum cost). You would look around, find the direction of the steepest slope downwards, and take a step in that direction. You repeat this process until you reach the lowest point.

Gradient Descent formalizes this process:

1. **Initialization:** It starts by assigning an initial set of values (often zeros or small random numbers) to the β-coefficients (θ₀, θ₁, ..., θ<0xE2><0x82><0x99>) in the equation.
2. **Calculate Cost:** Using these initial parameters, the algorithm predicts the target values (ŷ) for the input dataset and calculates the initial cost using the chosen cost function (e.g., MSE).
3. **Calculate the Gradient (Derivative):** The algorithm then calculates the **derivative** (or more accurately, the *partial derivative* for each parameter) of the cost function with respect to each β-coefficient (θⱼ).
   * The derivative represents the **slope** of the cost function at the current parameter values.
   * Crucially, the **gradient** (the vector of all partial derivatives) points in the direction of the **steepest increase** in the cost function.
4. **Update Parameters:** To *decrease* the cost, we need to move in the *opposite* direction of the gradient. Gradient Descent updates the values of the β-coefficients using the following rule (simultaneously for all j):
5. θⱼ := θⱼ - α \* (∂/∂θⱼ) J(θ)

Where:

* + θⱼ is the j-th parameter (e.g., β₀, β₁, etc.).
  + := means update the value.
  + α (alpha) is the **Learning Rate**.
  + (∂/∂θⱼ) J(θ) is the partial derivative of the cost function J(θ) with respect to the parameter θⱼ. This is the gradient component for that parameter.

1. **The Learning Rate (α):**
   * This is a **hyperparameter** that controls the **size of the step** taken during each parameter update.
   * It needs to be chosen carefully:
     + A **higher Learning Rate** makes the model learn faster (bigger steps), but it risks **overshooting** the minimum and potentially diverging (the cost might bounce around or even increase).
     + A **lower Learning Rate** takes smaller, safer steps, making it more likely to find the minimum, but the training process will be **slower**, requiring more iterations.
   * The learning rate is a value you specify as part of setting up the Linear Regression training process.
2. **Iteration:** Steps 2-4 (Calculate Cost, Calculate Gradient, Update Parameters) are repeated iteratively. With each step, the parameters should move closer to the values that minimize the cost function.
3. **Convergence:** The process stops when a convergence criterion is met, such as:
   * The change in the cost function between iterations is very small.
   * The magnitude of the gradient is very small (meaning we are near a flat minimum).
   * A maximum number of iterations is reached.

The final values of the β-coefficients (θs) obtained after convergence represent the parameters of the trained Linear Regression model, defining the line or hyperplane that best fits the training data according to the chosen cost function.