

Module 10.2: Cost Function and Gradient Descent

How the model finds the best line

Let's discover how machine learning algorithms automatically find the perfect fit for your data.



Quick Recap: Linear Regression

Remember our simple equation:

$$y = mx + c$$

Where **m** is the slope and **c** is the y-intercept.

We used this to predict exam marks based on study hours.



- **The Big Question:** How do we automatically find the best values for m and c? That's where cost functions and gradient descent come in!

Understanding Prediction and Error

Every prediction has an error which is the difference between what we predicted and the actual value.

Study Hours	True Marks	Predicted	Error
2	45	40	+5
5	75	70	+5
8	85	95	-10

The formula is simple:

$$\text{Error} = \text{True Value} - \text{Predicted Value}$$

Why We Can't Just Add Up Errors

The Cancellation Problem

Positive and negative errors cancel each other out, hiding the true error size.

Example

Error A = +5

Error B = -5

Sum = 0

But both predictions were off by 5!

This is why we need a smarter approach to measuring total error.

Mean Squared Error (MSE)

MSE is our **cost function**. It tells us how "wrong" our line is.

$$J(m, c) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

O1

Calculate Each Error

Find the difference between true and predicted values

O2

Square Each Error

This makes all errors positive and penalizes big mistakes more

O3

Average Them All

Sum the squared errors and divide by the number of points



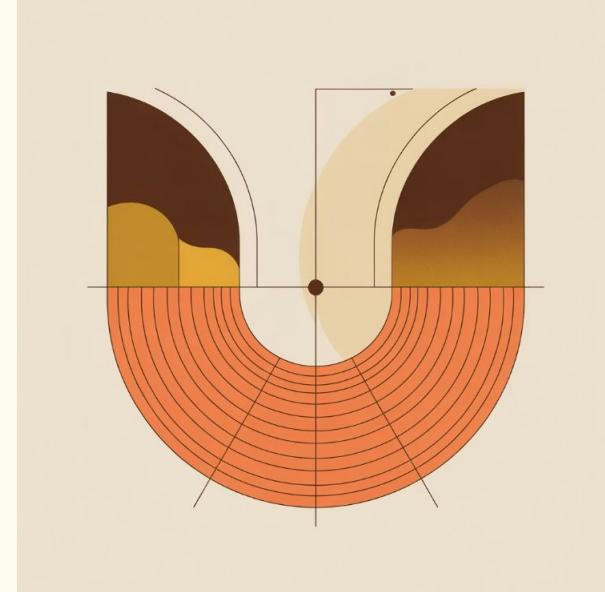
Key Insight: Lower cost = better line. Our goal is to minimize this number!

Visualizing the Cost Function

When we plot cost against different slope values (keeping intercept fixed), we get a U-shaped curve.

The **bottom of the curve** represents the best slope value — the one that minimizes error.

Gradient descent is our strategy to reach that sweet spot.



The Hill Climbing Analogy

🌙 You're on a hill in the dark

You can't see the bottom, but you can feel which way slopes downward.

👟 Take small steps downhill

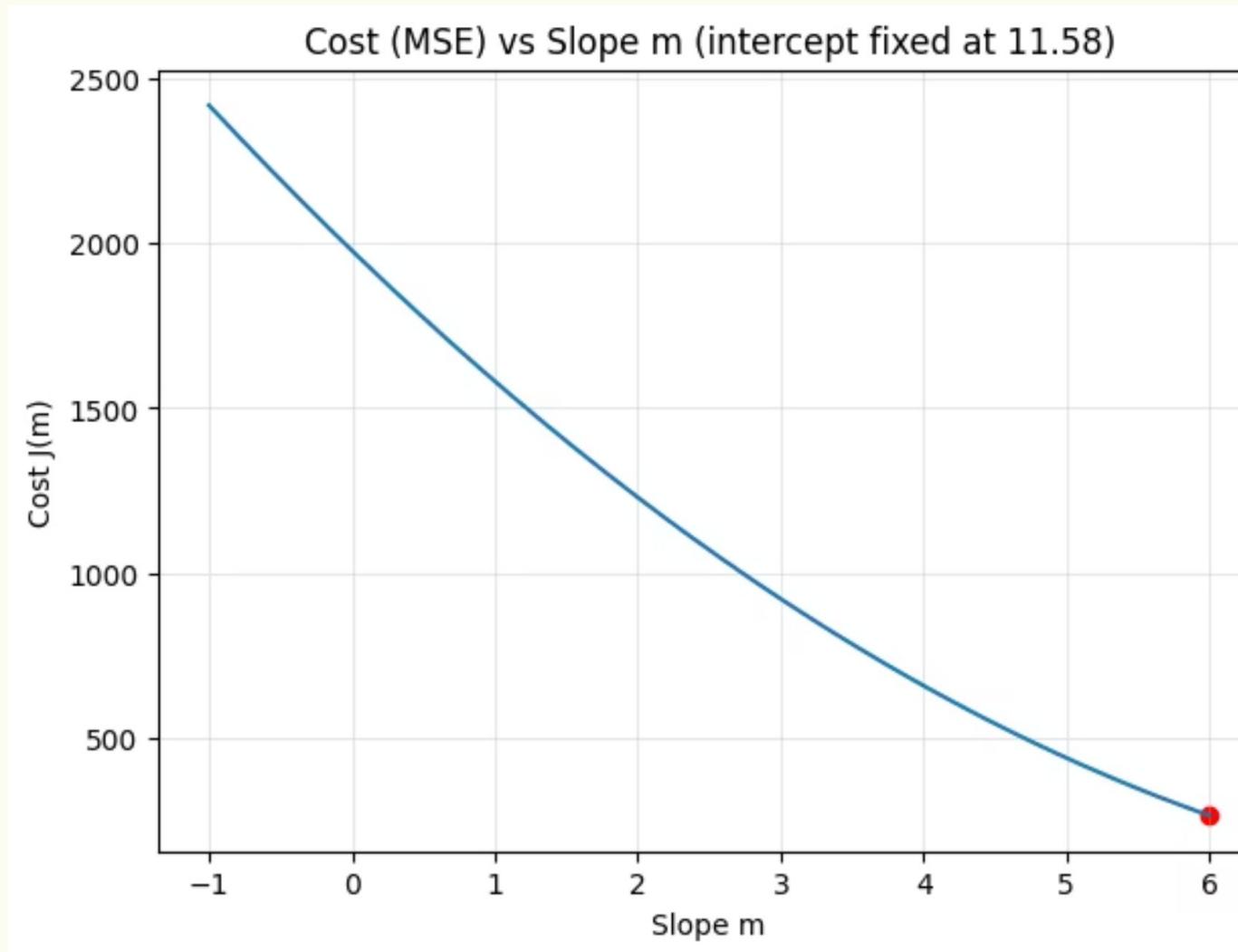
Move in the direction that goes down most steeply.

🎯 Repeat until you reach the valley

Keep stepping down until you can't go any lower.

That's exactly how gradient descent finds the best line by following the slope of the cost function downward!

Gradient Descent in Action

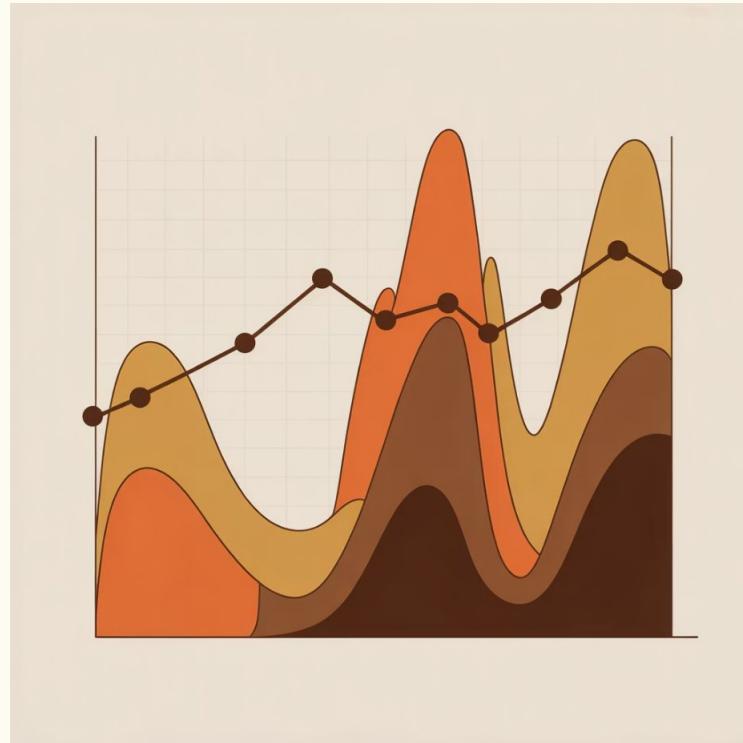


This visualization demonstrates Gradient Descent, an optimization algorithm that iteratively adjusts parameters to minimize a cost function. The winding path illustrates the algorithm's journey, starting from a high-cost point and progressively stepping down the "surface" of the cost function. Each step brings it closer to the global minimum, representing the optimal set of parameters for our model.

What We're Actually Updating

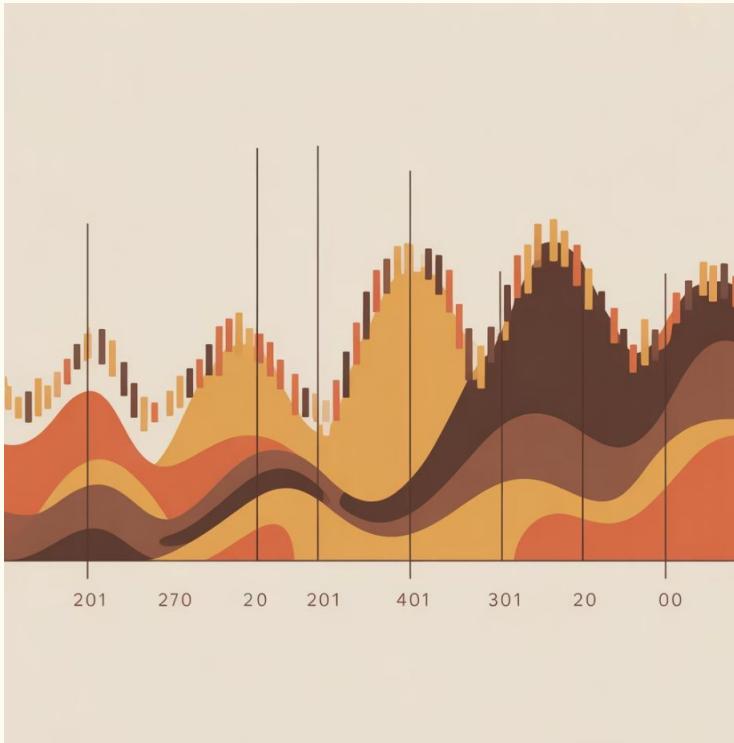
Think of **m** (slope) and **c** (intercept) as two knobs we're constantly adjusting to improve our line.

Initial Random Line

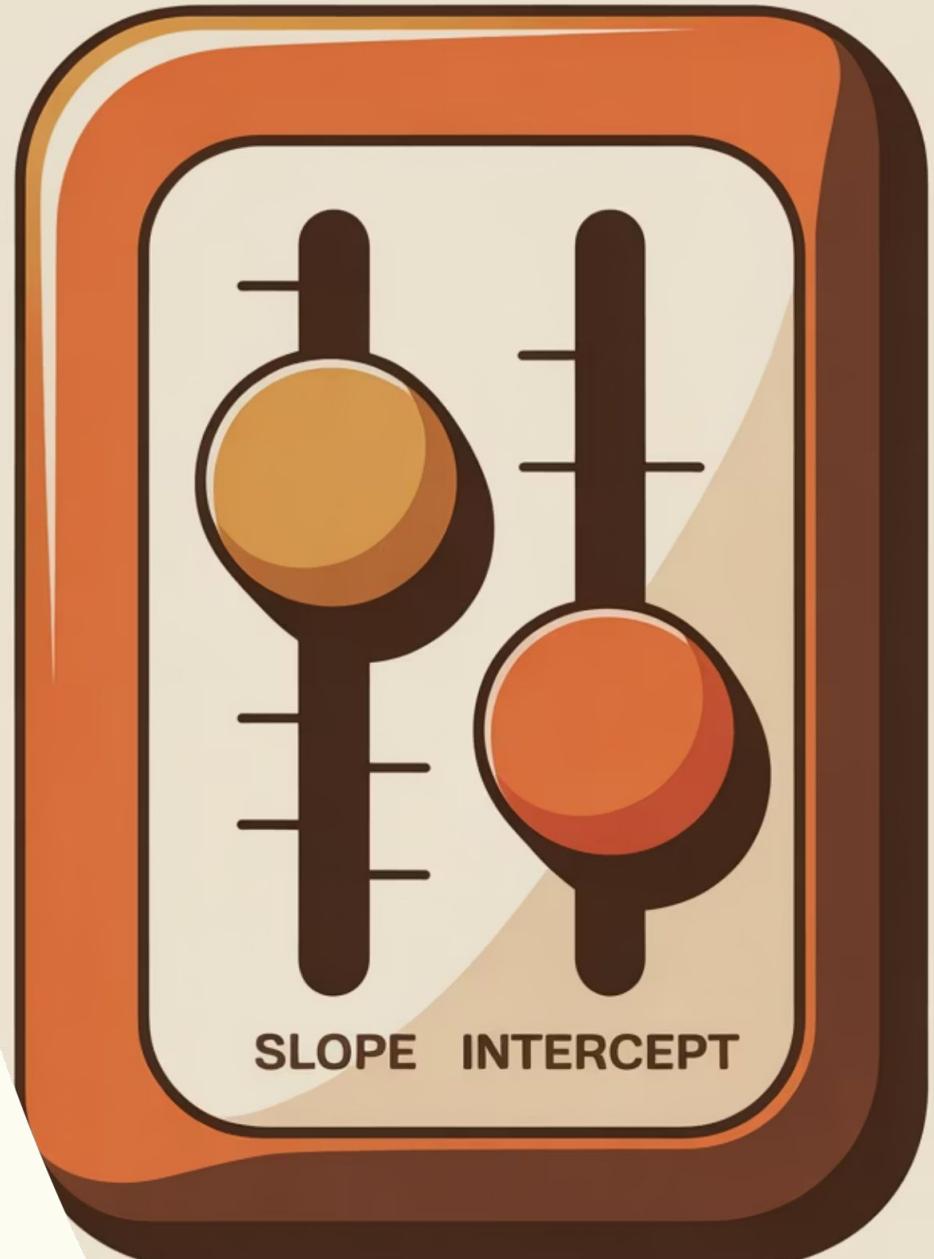


Starting point: $m = 2$, $c = 10$ High
cost ✗

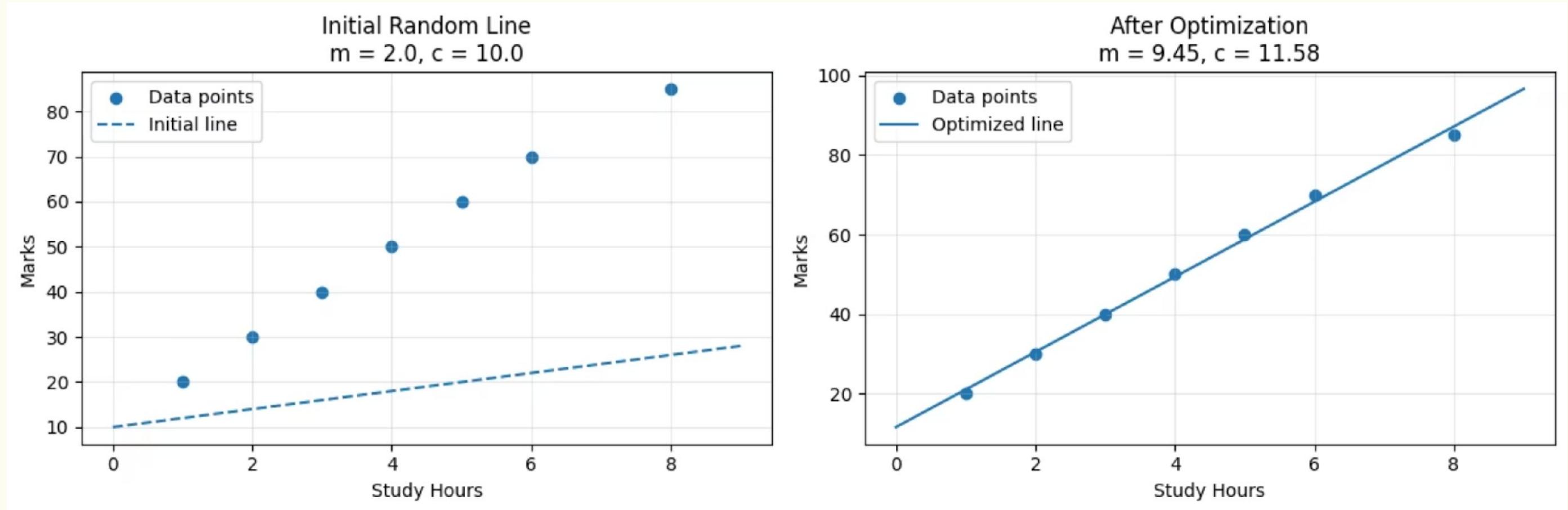
After Gradient Descent



Optimized: $m = 5.2$, $c = 35$ Low
cost ✓



The Power of Optimization



- Left graph shows the initial random line with parameters $m = 2.0, c = 10.0$, which poorly fits the scattered data points.
- Right graph shows the optimized line after gradient descent with parameters $m = 9.55, c = 11.58$, which closely follows the data trend.
- Notice how the optimized line minimizes the distance between predictions and actual data points.
- This transformation happens automatically through iterative parameter updates.

Learning Rate: Finding the Right Speed

The learning rate controls how big our steps are when updating m and c .



Too Small

Tiny steps take forever to reach the minimum. Training is painfully slow.



Too Large

We overshoot and bounce around, never settling at the minimum.

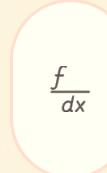


Just Right

Balanced steps that converge efficiently to the optimal solution.

The Gradient Descent Algorithm

Here's the complete process in six simple steps:

-  **Initialize Parameters**
Start with random values for m and c
-  **Compute Predictions**
Use current m and c to predict all y values
-  **Calculate Cost**
Use MSE formula to measure total error
-  **Compute Gradients**
Find the direction of steepest descent
-  **Update Parameters**
Adjust m and c by taking a step downhill
-  **Repeat Until Convergence**
Keep iterating until cost stops decreasing

 **The Beautiful Pattern:** Cost function + Gradient descent = Automatic optimization. This same pattern powers most machine learning algorithms!