# Module 2

**Linear Regression with PyTorch**

**Linear Regression**

## 📌 Linear Regression Training

This section introduces the training process for linear regression in PyTorch. It defines what constitutes a dataset, explains the noise assumption behind regression models, and presents the objective of learning model parameters by minimizing the mean squared error.

The focus is on how a model learns from examples by fitting a line that best captures the relationship between the input and output variables.

### 🔹 Defining the Dataset and Learning Objective

Linear regression aims to model the relationship between a feature (independent variable x) and a target (dependent variable y).

The goal of training is to **learn the best** values for the model **parameters**—slope and bias—that define a linear function capable of estimating y given x.

* A dataset is composed of **N** pairs of values: (x₁, y₁), (x₂, y₂), …, (xₙ, yₙ).
* Each xᵢ and yᵢ pair is related through a linear function plus a small amount of random noise.
* This process is known as **supervised learning**, where known input-output pairs are used to fit a model.

Examples of real-world applications of simple linear regression include:

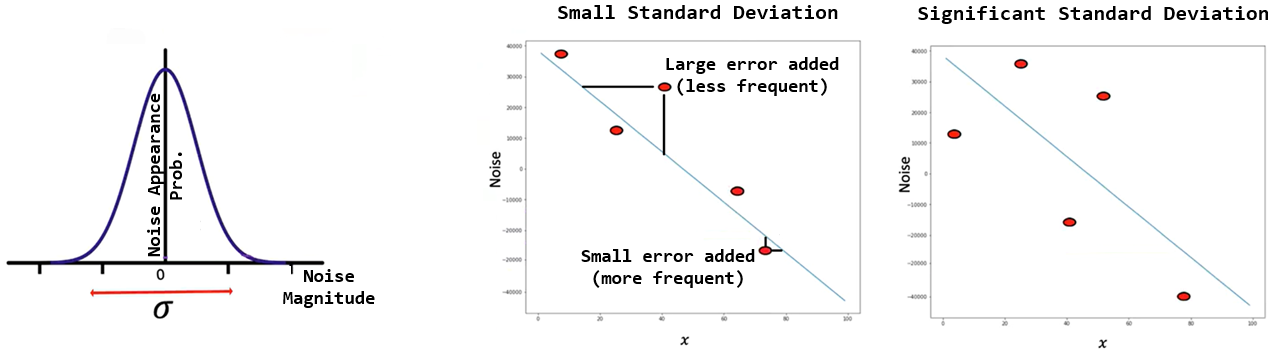
* Predicting house prices based on size.
* Estimating stock prices from interest rates.
* Modeling fuel efficiency as a function of horsepower.

In all cases, x is the feature and y is the predicted output.

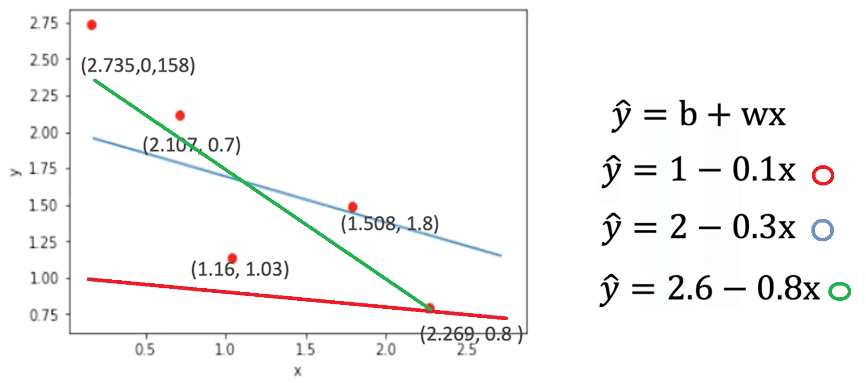
### 🔹 The Noise Assumption in Regression

Even when the relationship between variables is approximately linear, real-world data is not perfectly aligned on a straight line. This is because of **random noise**, which reflects measurement errors or unmodeled effects.

* The noise is assumed to be **Gaussian-distributed** with a mean of zero.
* The horizontal axis of the Gaussian curve represents the magnitude of the added noise.
* The vertical axis represents the probability of observing that value.
* Most of the noise values are close to zero, with only occasional large deviations.
* The more significant the standard deviation or, the more disperse the distribution is, the more the samples deviate from the line.



### 🔹 The Goal of Training

The objective of training a linear regression model is to **find the line that best fits the dataset**.

* In practice, several candidate lines may be drawn through the data points.
* Visually inspecting lines can suggest better or worse fits, but a mathematical method is needed for objective evaluation.

To formalize the training process, a **cost function** is introduced:

* The cost function is the **Mean Squared Error (MSE)**:

Where:

* is the predicted output
* is the actual output
* is the number of data points
* The MSE depends on the **slope and bias** of the model.
* Different parameter values lead to different MSE values.
* The best-fitting line is the one that **minimizes** this cost function.

Minimizing the mean squared error ensures that, on average, the model's predictions are as close as possible to the true values of y.

### ✅ Takeaways

✅ A linear regression model learns to map x to y by fitting a line to a set of input-output pairs.

✅ Datasets consist of ordered pairs of numeric values, where each pair defines a single example.

✅ Real data contains noise, modeled as Gaussian-distributed random variation added to each observation.

✅ The goal of training is to identify model parameters (slope and bias) that minimize the prediction error.

✅ The prediction error is quantified using the **mean squared error**, which forms the basis of the cost function used during optimization.

## 📌 Loss in Linear Regression

This section introduces the concept of **loss** as a fundamental building block in model training.

Loss quantifies the difference between the model’s prediction and the true value, and serves as the foundation for the **cost function**, which is used to guide parameter optimization.

### 🔹 Role of Loss in Model Training

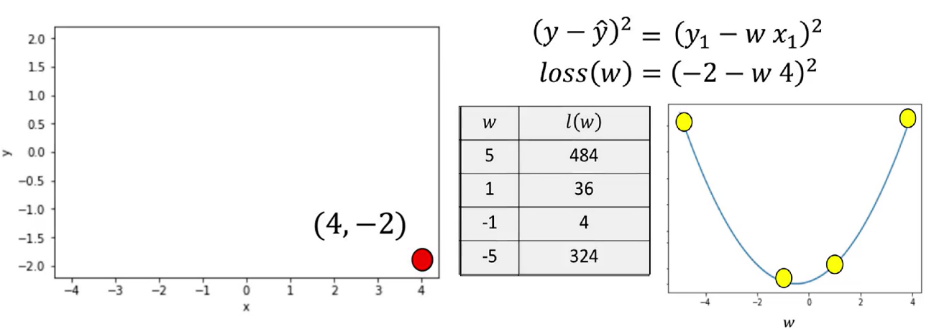
The training objective in linear regression is to learn the best model parameters (slope and bias) that result in accurate predictions for the dependent variable y given input x.

* For any input-output pair (x, y), the model produces an estimate ​ using the linear function.
* Loss measures how far this prediction ​ ​ is from the actual value y.
* The **smaller** the loss, the **better** the prediction.

### 🔹 Defining Loss for a Single Sample

To understand how the model adjusts parameters, a simplified example is used with only one sample:

* Suppose x = -2 and y = 4.
* A model prediction is computed using a candidate slope value .
* The prediction error is calculated as:



* Selecting a **slope of 5**, the line is far from the data point. In the data space, the value of the loss function is relatively large,
* Selecting a **slope of 1**, the value for the loss is near the minimum of the parameter space.
* Selecting a **slope of -1**, the result gets much closer to the minimum of the loss function, closer to the loss curve.
* A **slope of -5** the line is much farther away from the data point.

The squared difference captures how far the prediction is from the actual value and ensures that positive and negative errors do not cancel each other out.

* Since the true values of x and y are fixed during training, the loss becomes a function of the model parameter (slope).
* The loss function is also called the **criterion function**.
* It outputs a numerical value that reflects how good or bad a model’s prediction is.
* When visualized, the loss function appears as a **concave bowl**, or **parabola**, in the parameter space.

This shape has key properties:

* **Minimum point** corresponds to the best slope value.
* **Left of the minimum**: the derivative (slope of the loss curve) is negative.
* **Right of the minimum**: the derivative is positive.
* **At the minimum**: the derivative is zero.

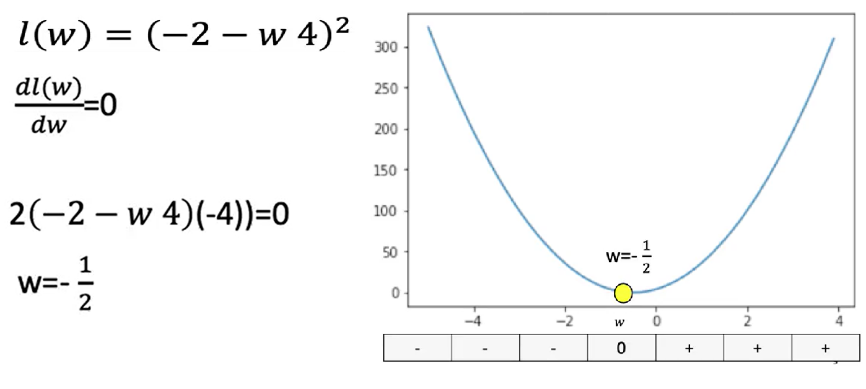
This behavior allows optimization techniques to search for the best parameter value by analyzing the derivative of the loss function.

### 🔹 Systematic Minimization of Loss

Instead of testing random parameter values, a **systematic method** is preferred to minimize the loss:

* Visualizing loss for different slope values shows:
  + Poor parameter choices result in high loss.
  + Optimal parameter values bring the predicted line closer to the actual data point, reducing loss.
* The **best slope** can be found algebraically by:
  + Taking the **derivative** of the loss function with respect to the slope. **derivatives points in the direction of decreasing loss**.
  + Setting the derivative equal to zero.
  + Solving for the slope value.

This technique finds the slope that minimizes loss for the given data point.

We can actually find the best value for the slope by setting the derivative = 0.

⚠️ However, this exact method is impractical for more complex models (e.g., deep learning), where explicit derivatives are difficult or impossible to compute algebraically.

### ✅ Takeaways

✅ Loss is a numeric measure of how well a model prediction matches the actual target value.

✅ For linear regression, loss is commonly defined as the **squared difference** between prediction and target.

✅ The loss function is treated as a function of the model parameters (e.g., slope).

✅ The objective of training is to **minimize the loss** to improve prediction accuracy.

✅ The loss function has a clear geometric interpretation: its **minimum** represents the best-fitting model.

✅ Derivatives indicate how to update parameters and are foundational to gradient descent and training in neural networks.

## 📌 Gradient Descent and Cost

This section introduces **gradient descent**, the fundamental optimization technique used to minimize loss functions in machine learning.

It explains how gradient descent works in one dimension and addresses challenges like learning rate selection and stopping criteria.

The process is applied to adjust model parameters iteratively to minimize prediction errors.

### 🔹 What is Gradient Descent?

Gradient descent is an **iterative algorithm** used to find the minimum of a function by adjusting its parameters in the direction of the steepest descent.

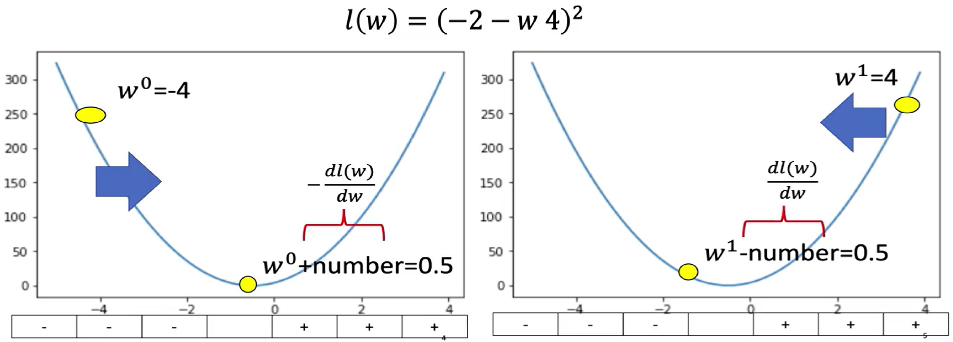
* The algorithm begins with a **random initial guess** for a model parameter (e.g., the slope in linear regression).
* It evaluates the **gradient** (i.e., the derivative) of the loss function with respect to that parameter.
* The parameter is then **updated** by subtracting a value **proportional to the derivative**:

Where:

* is the current parameter value.
* is the learning rate.
* is the derivate of the loss function.

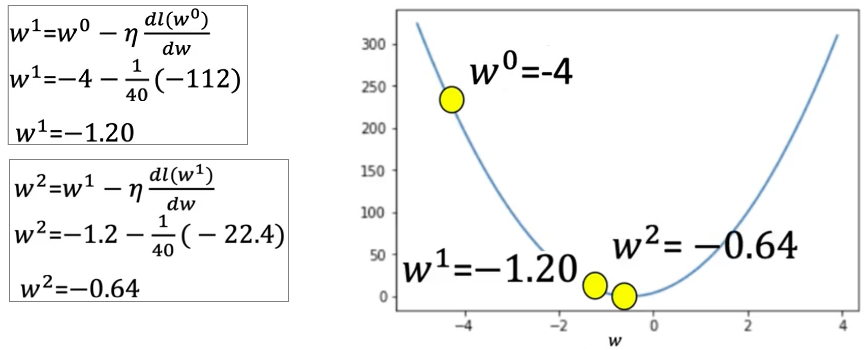
The sign of the gradient determines the direction of the parameter update:

* If the derivative is negative, the parameter increases.
* If the derivative is positive, the parameter decreases.
* The gradient points away from the minimum, so its negative is used to move toward the minimum.



### 🔹 Gradient Descent in Practice

To compute gradient descent:

* The process is started by selecting a random guess, and choosing the learning rate.
* Then the derivate at that point is calculated.
* Finally, the parameter is updated.

After the update, the loss decreases.

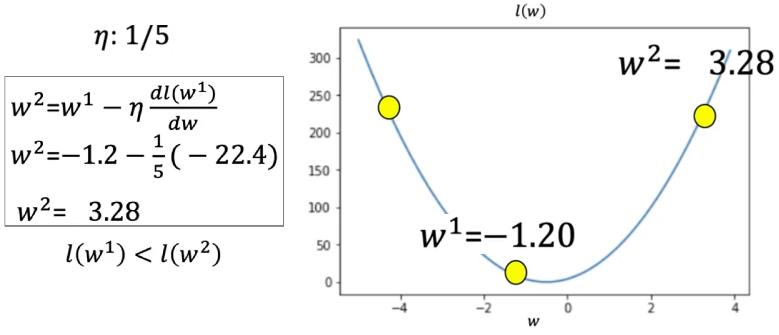
* The next iteration continues using the updated value.
* The parameter is updated again and the loss continues to decrease.

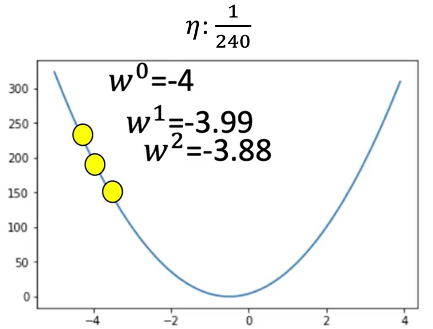
This process continues until a stop condition is reached.

### 🔹 Problems with Learning Rate

The **learning rate ()** controls the step size for parameter updates. Choosing an inappropriate value leads to problems:

**1. Learning Rate Too Large**

* The algorithm **overshoots** the minimum and the loss increases.
* This causes the algorithm to diverge or oscillate.

**2. Learning Rate Too Small**

* The algorithm makes **very slow progress**.
* This leads to excessive computation time and inefficient convergence.

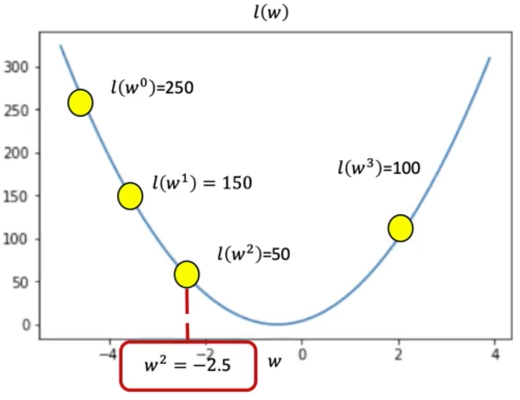
The learning rate must be chosen carefully to balance stability and speed of convergence.

### 🔹 When to Stop Gradient Descent

Several strategies are commonly used to decide **when to stop** the gradient descent process:

**1. Fixed Number of Iterations**

* Run gradient descent for a predefined number of iterations (e.g., 3 iterations).
* This is simple but may **miss the true minimum**.

For example:

* Start with a loss of 250.
* After several iterations, the loss values may be: 150 → 80 → 50 → 100.
* The stopping point is when the loss **increases** to 100.
* The best parameter is the one corresponding to a loss of 50.

**2. Monitor Loss Values**

* Continue updating until the loss stops decreasing.
* Maintain a table of loss values across iterations:
  + If the loss starts increasing or stagnates, the process stops.
  + The best parameter value is chosen from the iteration with the **lowest loss**.

### ✅ Takeaways

✅ Gradient descent is a key method for minimizing loss and learning model parameters.

✅ It updates parameters using the **negative derivative** of the loss function.

✅ The **learning rate** controls how far the parameter moves with each update:

* Too high: unstable updates, missed minimum.
* Too low: slow convergence.

✅ The update rule is repeated iteratively to approach the optimal value.

✅ The process stops either after a fixed number of iterations or when the loss stops improving.

✅ Understanding gradient descent is essential for training linear and deep learning models.

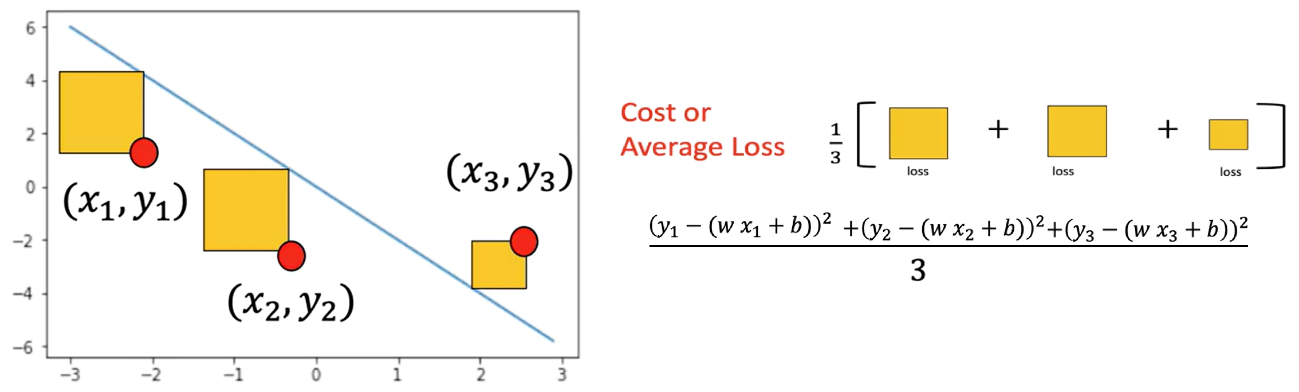
## 📌 Cost

The cost function allows the model to evaluate how well it fits **all training samples**, not just one, and serves as the basis for **batch gradient descent** optimization.

### 🔹 From Loss to Cost

The **loss function** measures prediction error for a single training example. To train a model on multiple examples, a **cost function** is defined by **summing or averaging** the losses across all samples.

* Each prediction error is squared to form a small square, visually representing the magnitude of error.
* The **cost** is the **sum of all these squared errors**, or their **average**:



Following PyTorch’s convention, the cost function is denoted as **L** (for loss), even when referring to the total over a batch.

### 🔹 Cost Function as a Function of Parameters

The cost function depends on model parameters, especially the **slope (weight)** and **bias**:

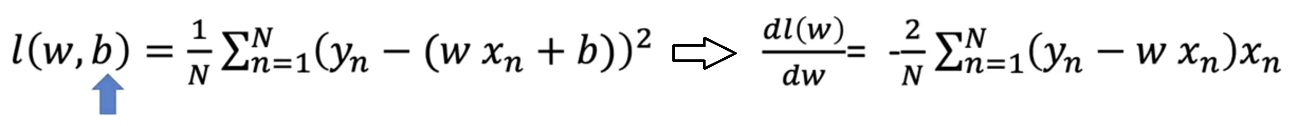
* The slope determines the steepness of the predicted line. It controls the relationship between x and y.
* The bias controls the horizontal offset.

The **goal** is to find the values of slope and bias that **minimize the cost**, which means achieving the best possible fit for the data.

### 🔹 Applying Gradient Descent to Cost

The same **gradient descent** technique used for single-sample loss is applied to the **cost function** to optimize parameters across **multiple samples**.

* For multiple data points, the derivative of the cost with respect to the slope is the **sum of the derivatives** for each sample.



* The process involves:
  1. Computing the cost using all samples.
  2. Calculating the gradient (derivative) of the cost.
  3. Updating parameters using the gradient.

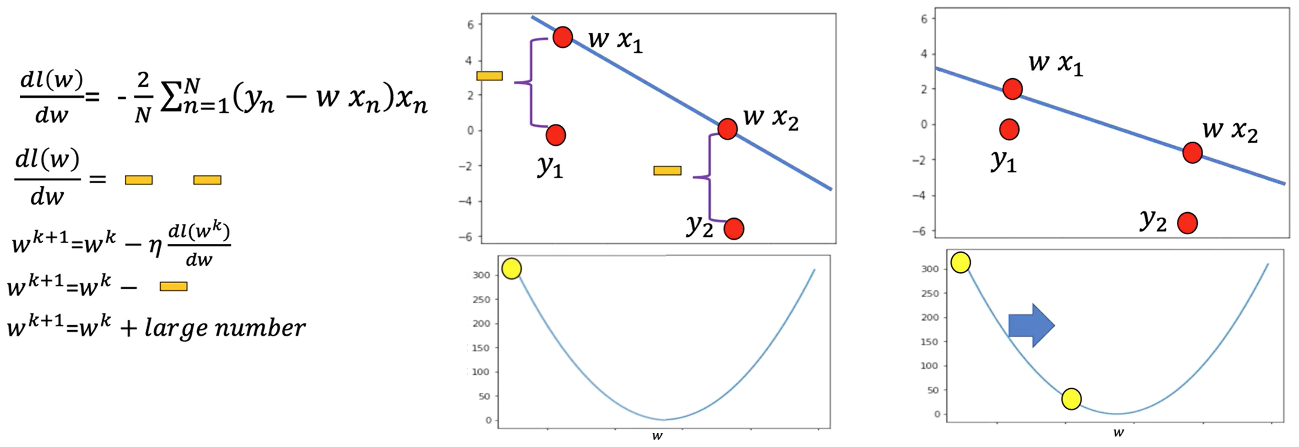
This process is repeated iteratively to improve the model fit.

Several scenarios illustrate how sample distribution affects the derivative of gradient descent with just the slope:

🔸 **Case 1 - All Samples on the Same Side:**

If both data points lie **below** the current prediction line:

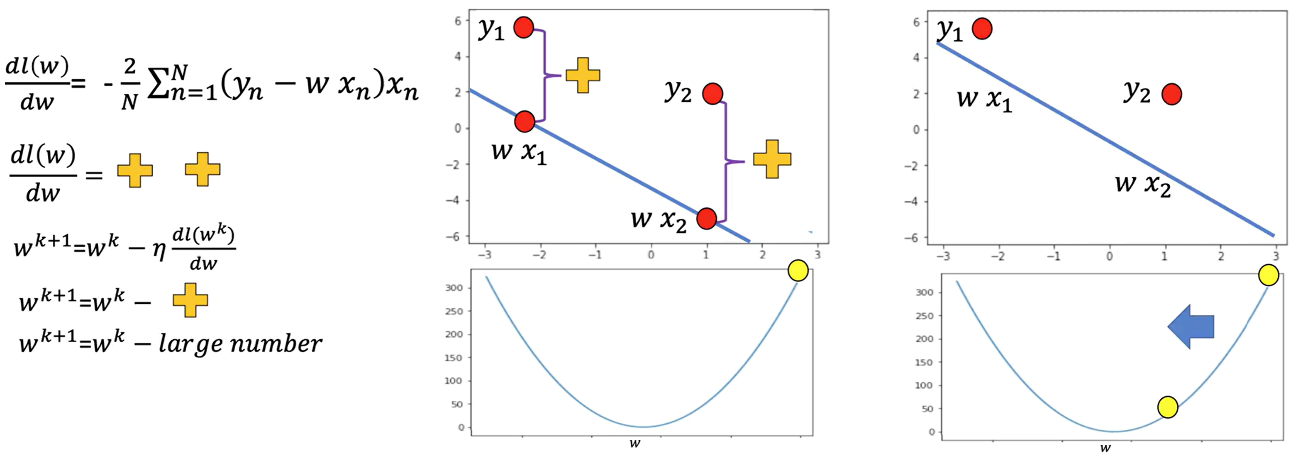
* The derivative is **strongly negative**.
* The update step adds a **large positive value** to the parameter.
* The prediction line moves **closer to the data**.



🔸 **Case 2 - All Samples on the Opposite Side:**

If both points lie **above** the prediction line:

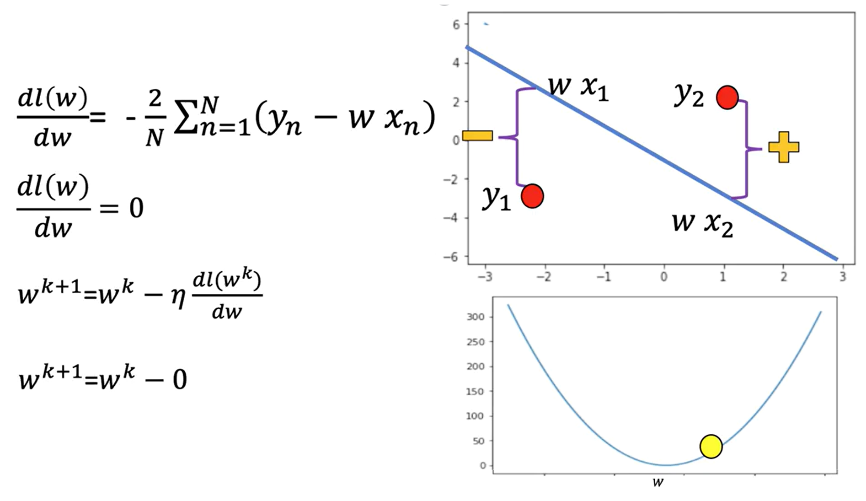
* The derivative is **strongly positive**.
* The update subtracts a **large value**, moving the line down.



🔸 **Case 3 - Mixed Sides:**

If one sample is **above** and the other **below** the line:

* The positive and negative derivatives **cancel each other out**.
* The resulting derivative is **near zero**.
* The update step is **very small**, and the line changes little.



### 🔹 Batch Gradient Descent

When the **entire dataset** is used to calculate the cost and gradient at each step, the method is called **Batch Gradient Descent**:

* The "batch" refers to the full training set.
* All samples are used to:
  + Compute the total cost.
  + Calculate the overall derivative.
  + Perform the parameter update.

For example, if the batch size is 3:

* The model uses all 3 data points to compute the cost and update.
* This ensures stability and directionally correct updates.

### ✅ Takeaways

✅ The cost function aggregates prediction errors across all training samples.

✅ It serves as the objective function for training linear models.

✅ Gradient descent applied to the cost function is **batch gradient descent**.

✅ Sample distribution affects the gradient and update size.

✅ Proper gradient accumulation across the batch ensures consistent parameter updates.