# Module 2

**Linear Regression with PyTorch**

**Linear Regression Training with PyTorch**

## 📌 PyTorch LR Training – Slope and Bias

This section focuses on training a linear regression model in PyTorch by learning both slope (weight) and bias using gradient descent.

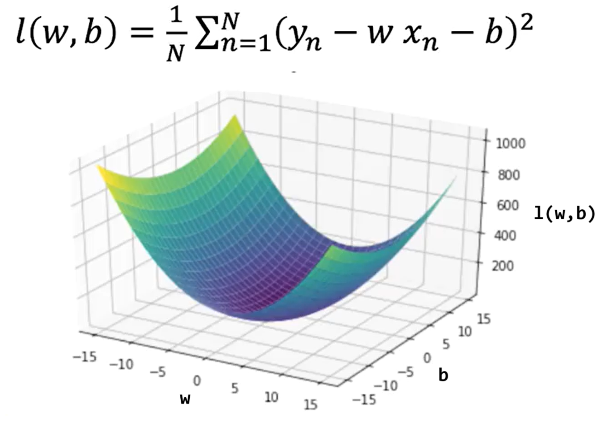
The objective is to minimize a cost surface defined by these parameters and understand how the model iteratively adjusts them to fit the training data.

### 🔹 Cost Surface and Parameter Space

The cost function in linear regression is defined as the **average loss** across training samples. It depends on two parameters:

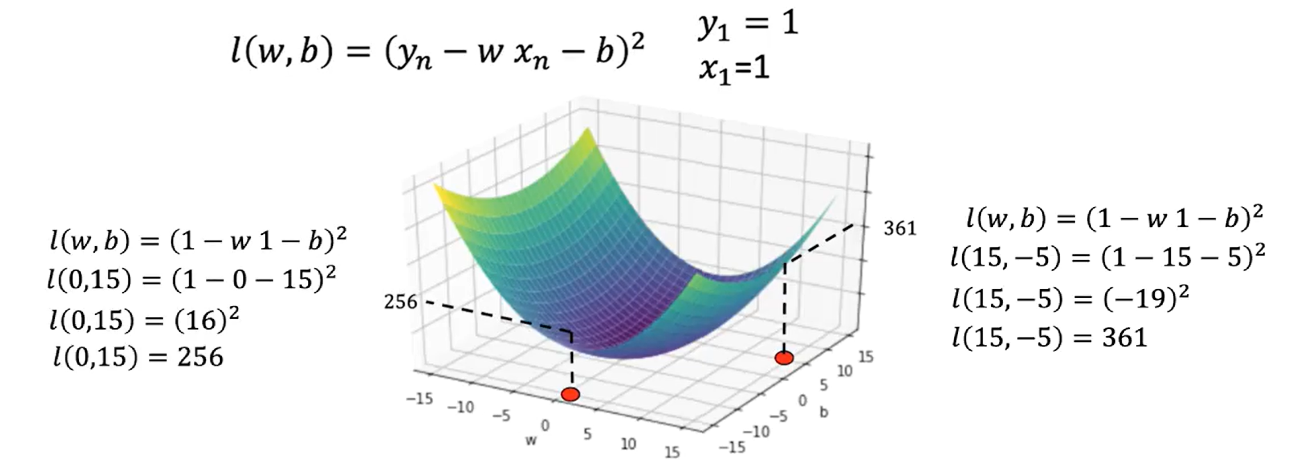
* **Slope (w)**: Determines the relationship between input x and output y.
* **Bias (b)**: Controls the vertical offset of the line.

When considering both parameters simultaneously, the cost function becomes a **two-dimensional function**, and it can be visualized as a **cost surface**:

* One axis represents the slope.
* One axis represents the bias.
* The vertical dimension (height) represents the value of the cost.

This surface visualization allows us to understand how the cost changes depending on the parameter combinations.

High-cost values indicate poor model fit, while regions with low-cost values indicate better predictions.

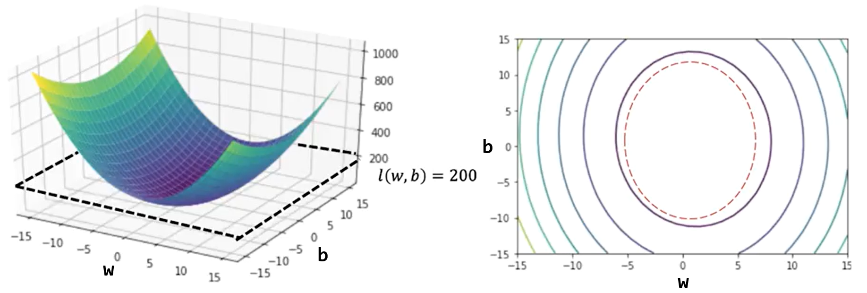


### 🔹 Contour Plots and Surface Slices

A contour plot is a useful tool for understanding cost surface. It’s a birds-eye view of the surface.

A **contour plot** is a two-dimensional representation that shows slices of the cost surface at fixed cost values. Each contour line connects parameter combinations that yield the **same cost**.

* The horizontal axis corresponds to slope (w).
* The vertical axis corresponds to bias (b).



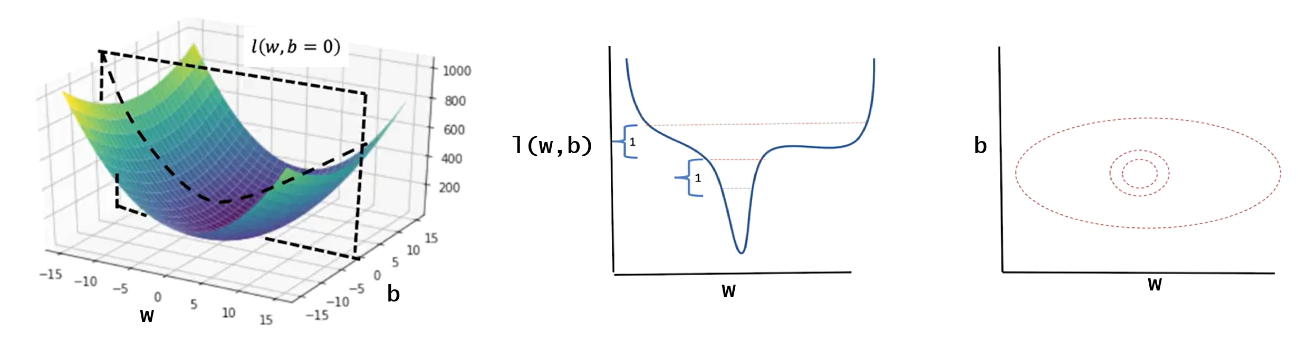
The **spacing between contour lines** reflects how steep or flat the surface is:

* Closely spaced contours indicate steep changes in cost (high gradient).
* Widely spaced contours indicate shallow regions (low gradient

Vertical and horizontal slices of the surface show how cost changes when varying one parameter at a time:

* A **vertical slice** (e.g., fixing b = 0) shows how cost varies with slope.
* A **horizontal slice** shows how cost varies with bias.
* These cross-sections visually demonstrate how curvature affects the update magnitude in gradient descent.

As you move away from the center (the minimum), the rate of change (gradient) increases, which informs the direction and size of parameter updates.



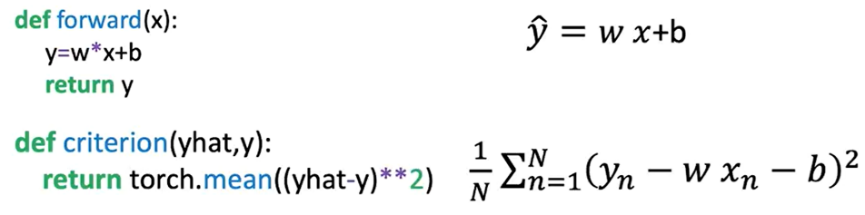
### 🔹 Gradient Descent in PyTorch (Manual Implementation)

ℹ️ There are several ways to minimize the cost function, the following methos is considered the “hard way” and usually is not implemented.

The training process is performed using PyTorch tensors and manual gradient descent.

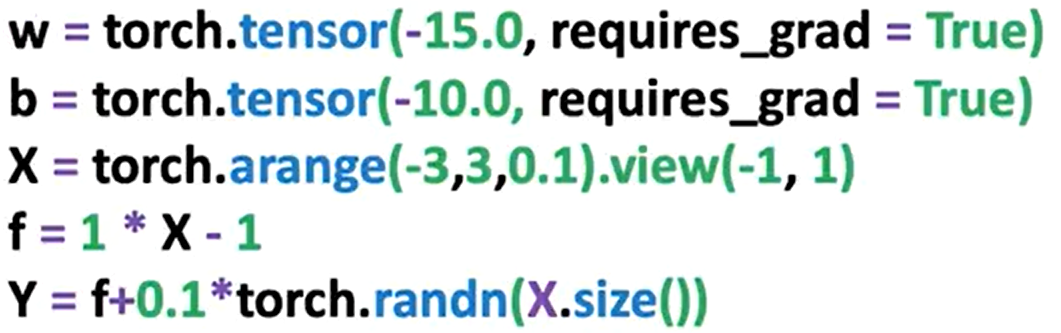
The following are key aspects of the training process:

* The model uses a **forward function** that defines de prediction using the equation of a line (includes both slope and bias).
* A criterion (cost) function is defined to calculate the mean squared error, representing the cost between predicted and actual values.



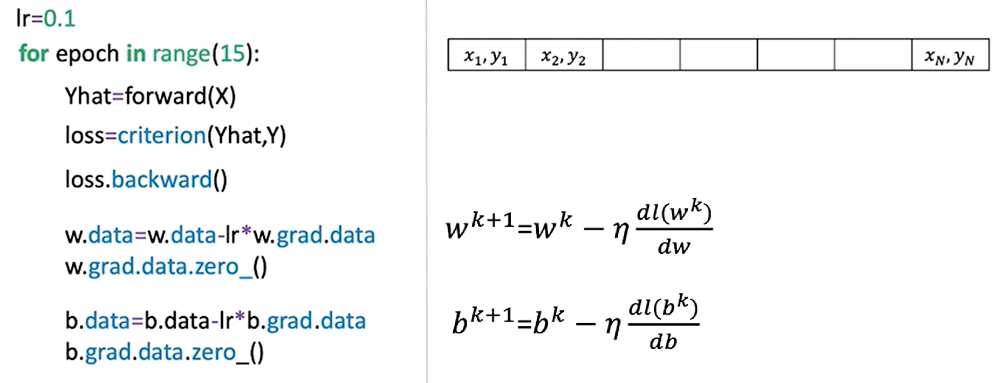
Tensors for:

* **w** (slope) and **b** (bias) are initialized with **requires\_grad=True** so PyTorch tracks their gradients.
* **X** (input data) and **Y** (target values) are created for the regression task.



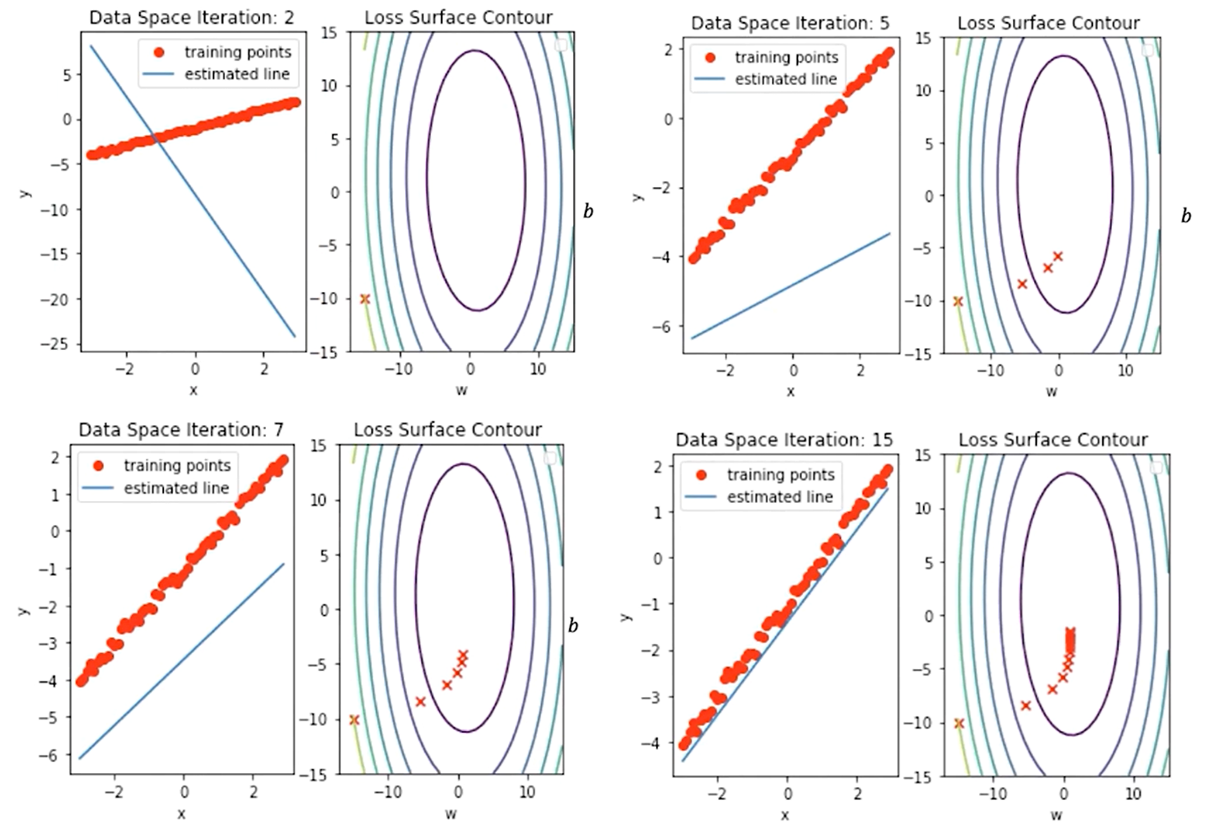
The procedure is identical as seen in last section, except we also update the bias term. During each epoch:

* The model predicts output using the forward function.
* The loss is computed.
* **.backward()** is called on the loss tensor to calculate gradients.
* The **.grad** attribute is accessed for both parameters to update them manually.
* **.data** is used to apply the update using the learning rate.
* .**zero\_()** is called to reset gradients for the next iteration.

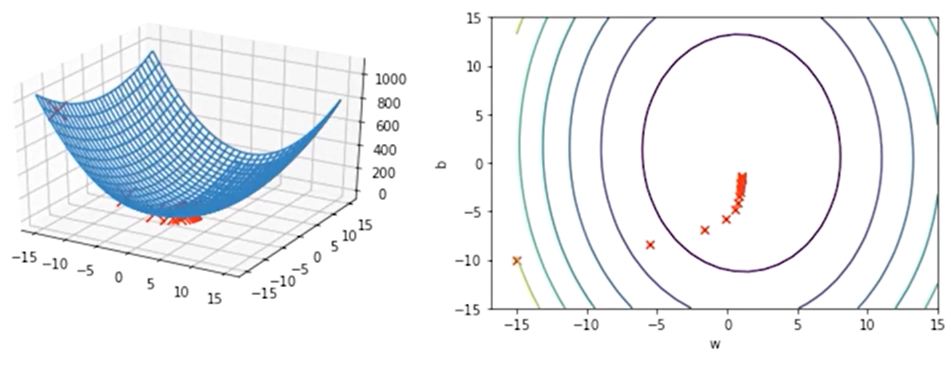


The plot on the right shows the loss or cost function for different values of the parameter; the red x’s represent parameter values at a given iteration.

The plot on the left shows the estimated function using the parameter values, and the red dots show the training points.



In this image we can see the correspondence between the contour lines and the loss surface.



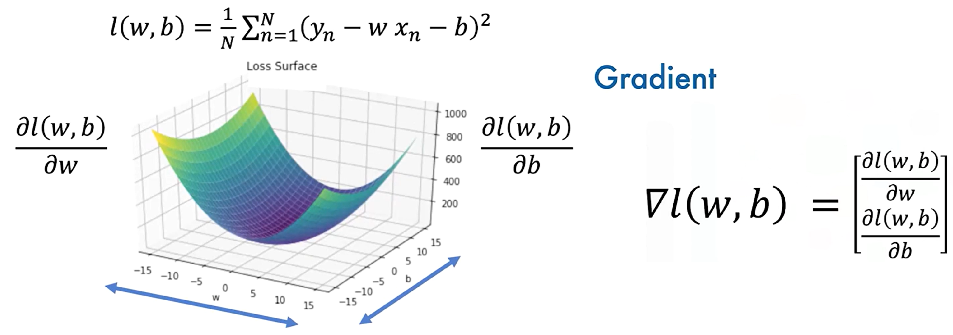
This manual procedure gives insight into how PyTorch handles gradients internally and how gradient descent is applied in practice.

### 🔹 Gradient Vector and Direction of Optimization

In general, the derivative with respect to a multivariate function is called a partial derivative.

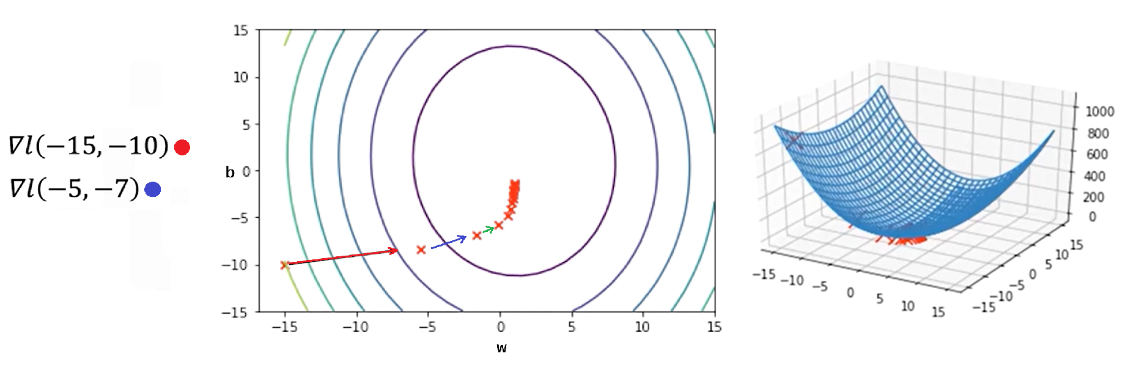
If we put the partial derivative with respect to each variable into a vector, we get the gradient, hence the name gradient descent.

**The gradient vector**, points in the direction of the steepest increase of the cost function.



The gradient is always **perpendicular to the contour lines** and guides the updates during training.

The gradient also points to the direction of greatest change, hence points to the direction of the next iteration, the same is true for the next iteration, and so on.



### ✅ Takeaways

✅ The cost function in linear regression with slope and bias defines a **surface** representing model error across parameter combinations.

✅ **Contour plots and surface slices** help visualize how parameter updates impact model performance.

✅ **Gradient descent** is used to iteratively update both slope and bias, reducing the model’s loss with each epoch.

✅ PyTorch allows full manual implementation of gradient-based optimization using tensors, gradient tracking, and **.backward()** computations.

✅ The **gradient vector** directs the updates and always points toward the direction of greatest increase, so moving opposite to it reduces the loss.

✅ After several epochs, the predicted line aligns closely with the data, and parameters converge to values that minimize the cost.