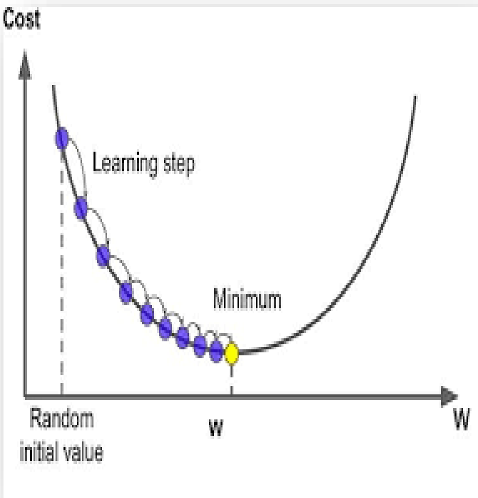
Gradient Descent

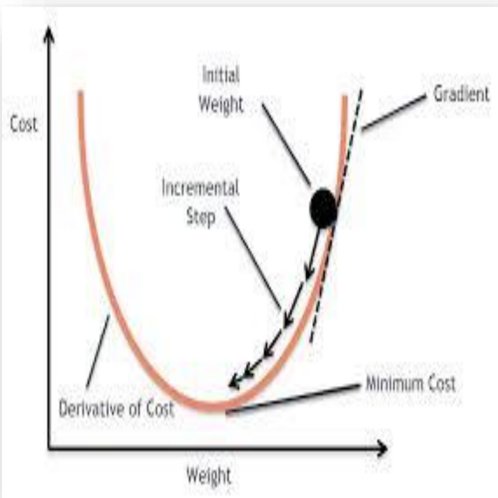
Gradient descent is an **iterative optimization algorithm** for finding the **local minimum** of a function. To find the local minimum of a function using gradient descent, it is essential to take steps proportional to the **negative of the gradient** (move away from the gradient) of the function at the current point. Gradient descent was originally proposed by **CAUCHY** in 1847. It is also known as the steepest descent.

The goal of the gradient descent algorithm is to **minimize** the given function (say, **cost function**).

❖ To achieve this goal, it performs two steps iteratively:

1. **Compute the gradient** (slope), the first-order derivative of the function at that point
2. **Make a step (move) in the direction opposite to the gradient.** The opposite direction of the slope increases from the current point by alpha times the gradient at that point





Working

1. The algorithm starts with an **initial set of parameters** and **updates them** in small steps to minimize the cost function.
2. In each iteration of the algorithm, the gradient of the cost function with respect to each parameter is computed.
3. The **gradient tells us the direction**; by **moving in the opposite direction**, we can find the direction of the steepest descent.
4. **The learning rate controls the step size**, determining how quickly the algorithm moves towards the minimum.
5. The **process is repeated until the cost function converges to a minimum**. Therefore, indicating that the model has reached the optimal set of parameters.

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\
&\qquad\quad (\text{for } j = 1 \text{ and } j = 0) \\
&\}
\end{aligned}
$$

Different variations of gradient descent include batch gradient descent, stochastic gradient descent, and mini-batch gradient descent, each with advantages and limitations.

Gradient descent guides ML models toward optimal performance by iteratively adjusting parameters to minimize the cost function.

Linear Regression

- **Hypothesis** function for simple linear regression

$$h_\theta(x) = \theta_0 + \theta_1 x$$

x- the input feature.

- **Parameters**: **θ_0** (intercept), **θ_1** (slope)
- **Cost Function:** J(θ): **J (θ_0, θ_1)**
- **Goal:** **Minimize J (θ_0, θ_1)**

For linear regression, the cost function used to evaluate how well the model fits the data is the **Mean Squared Error (MSE)**:

- **Cost Function**

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2$$

- **The gradient of the Cost Function:**

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

- **Use the parameter update rule given below for regression**

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta_j}$$

$$\theta_j := \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

The algorithm converges when the change in the cost function becomes sufficiently small or after a predetermined number of iterations.

Key Considerations:

Learning rate (α): Choosing an appropriate learning rate is crucial. If it's too large, the algorithm might not converge. If it's too small, convergence will be slow.

Convergence criteria: Can stop the iteration if the cost function decreases below a threshold or after a certain number of iterations.