# ARTIFICIAL INTELLIGENCE
# &
# MACHINE LEARNING

## SESSION NO :14

# Definition of a Derivative

In mathematics, the **derivative** is a fundamental tool that quantifies the sensitivity to change of a function's output with respect to its input.

The derivative of a function of a single variable at a chosen input value, when it exists, is the slope of the tangent line to the graph of the function at that point.

The tangent line is the best linear approximation of the function near that input value.

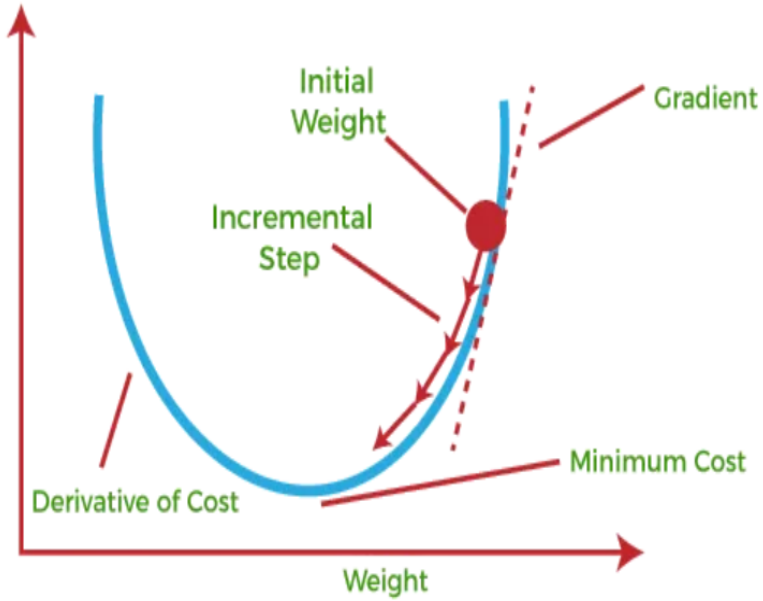
# Rules of a Derivative

Some basic rules of taking derivatives

- **Sum Rule**: $(f(x)+g(x))' = f'(x)+g'(x)$
- **Scaling Rule**: $(a \cdot f(x))' = a \cdot f'(x)$ if $a$ is not a function of $x$
- **Product Rule**: $(f(x) \cdot g(x))' = f'(x) \cdot g(x) + g'(x) \cdot f(x)$
- **Quotient Rule**: $(f(x)/g(x))' = (f'(x) \cdot g(x) - g'(x)f(x))/(g(x))^2$
- **Chain Rule**: $(f(g(x)))' \stackrel{def}{=} (f \circ g)'(x) = f'(g(x)) \cdot g'(x)$

# Gradient descent

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. It trains machine learning models by minimizing errors between predicted and actual results.



**Step 1_Initialization:** Start with initial values for the model's parameters (weights and biases). These values can be randomly assigned or set to zero.

**Step 2_Cost Function Calculation:** Evaluate the cost function, which measures how well the model is currently performing. Common cost functions include Mean Squared Error (MSE) for regression tasks.

**Step 3_Gradient Calculation:** Calculate the gradient of the cost function. The gradient is a vector that points in the direction of the steepest ascent of the function at a given point. Since we want to minimize the cost, we move in the opposite direction of the gradient.

**Step 4_ Parameter Update:** Adjust the model's parameters using the following rule:

New_parameter = Old_parameter - Learning_rate * Gradient (Slope).

The "learning rate ( $\lambda$ )" is a hyperparameter that determines the size of the steps taken during each iteration. A small learning rate can lead to slow convergence, while a large learning rate can cause the algorithm to overshoot the minimum.

**Step 5_ Iteration:** Repeat steps 2-4 until a stopping criterion is met. This could be when the change in the cost function is below a certain threshold, or after a fixed number of iterations.

# Example:

- $J = \frac{1}{n}\sum_{i=1}^{n}[y_i - (mx_i + c)]^2$

Diff. w.r.t to ***m*** and make equal to 0

$$\frac{\partial J}{\partial m} = \frac{2}{n}\sum_{i=1}^{n}(-x_i)[y_i - (mx_i + c)]$$

$$0 = \sum_{i=1}^{n}[-x_i y_i + \mathrm{m}x_i^2 + x_i c]$$

$$\mathrm{m}\sum_{i=1}^{n}x_i^2 + c\sum_{i=1}^{n}x_i = \sum_{i=1}^{n}x_i y_i \quad \text{---------- (eq.1)}$$

# Example:

- $J = \frac{1}{n}\sum_{i=1}^{n}[y_i - (mx_i + c)]^2$

Diff. w.r.t to $c$ and make equal to 0

$$\frac{\partial J}{\partial c} = \frac{-2}{n}\sum_{i=1}^{n}[y_i - (mx_i + c)]$$

$$\mathrm{m}\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} c = \sum_{i=1}^{n} y_i \qquad \text{---------- (eq.2)}$$

$$m = m - \alpha * \frac{\partial J}{}$$

# Example:

$$m\sum_{i=1}^{5} x_i^2 + c\sum_{i=1}^{5} x_i = \sum_{i=1}^{5} x_i y_i \quad \text{---------- (eq.1)}$$

$$\mathrm{m}\sum_{i=1}^{5} x_i + 5\mathrm{c} = \sum_{i=1}^{5} y_i \quad \text{---------- (eq.2)}$$

eq.1 becomes, $165m + 25c = 96$

eq.2 becomes, $25m + 5c = 16$

By solving the above two equations we will get $m = \frac{2}{5}, c = \frac{6}{5}$. These are best parameter values to minimize cost function.

Therefore the equation for the slope & intercept form which best fits for the given set of 5 point. $y = \frac{2}{5}x + \frac{6}{5}$

# Why Necessary for Machine Learning

- Gradient descent is necessary for optimizing and fine-tuning the parameters of a machine learning model.
- Gradient descent aims to minimize a model's cost or loss function.
- By reducing the cost function, the model becomes better at making predictions and can generalize better to new data.

# Different Variants of Gradient Descent

- There are several variants of gradient descent that differ in the way the step size or learning rate is chosen and the way the updates are made.
- **Batch Gradient Descent**
- **Stochastic Gradient Descent (SGD)**
- **Mini-batch Gradient Descent**

# Batch gradient descent

- Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process referred to as a training epoch.
- While this batching provides computation efficiency, it can still have a long processing time for large training datasets as it still needs to store all of the data into memory.

# Stochastic gradient descent

- Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time.

# Mini-batch gradient descent

Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent.

It splits the training dataset into small batch sizes and performs updates on each of those batches.

This approach strikes a balance between the computational efficiency of batch gradient descent and the speed of stochastic gradient descent.

# Introduction to Mathematical Optimization

- “Optimization” comes from the same root as “optimal”, which means best. When you optimize something, you are “making it best”.
- But “best” can vary. If you’re a football player, you might want to maximize your running yards, and also minimize your fumbles. Both maximizing and minimizing are types of optimization problems.

# Introduction to Mathematical Optimization

- Optimization techniques are methods used to find the best solution to a problem, given a set of constraints.
- These techniques can be broadly categorized into classical, numerical, and evolutionary methods.
- Classical methods include linear and non-linear programming, while numerical methods involve gradient descent and Newton's method.
- Evolutionary methods, such as genetic algorithms and particle swarm optimization, are inspired by natural processes.

# Classical Optimization

- **Linear Programming:**
- Deals with optimizing a linear objective function subject to linear equality and inequality constraints.
- **Non-linear Programming:**
- Optimizes a non-linear objective function, potentially with non-linear constraints.
- **Integer Programming:**
- A type of optimization where some or all variables are restricted to integer values.
- **Dynamic Programming:**
- Solves complex problems by breaking them down into simpler overlapping subproblems.

- **Simplex Method:**
- A widely used algorithm for solving linear programming problems.
- **Gradient Descent:**
- An iterative optimization algorithm that finds the minimum of a function by repeatedly taking steps in the direction of the negative gradient.

# Numerical Optimization

- **Newton's Method:**
- Uses second-order derivatives to find the minimum of a function more efficiently than gradient descent.
- **Conjugate Gradient Method:**
- An algorithm that finds the minimum of a function by iteratively searching in conjugate directions.

# Evolutionary Optimization

- **Genetic Algorithms:**
- Inspired by natural selection, these algorithms use concepts like mutation and crossover to evolve a population of solutions.
- **Particle Swarm Optimization:**
- Mimics the social behavior of bird flocking or fish schooling to find optimal solutions.
- **Simulated Annealing:**
- A metaheuristic algorithm that explores the solution space by randomly sampling points and accepting worse solutions with a probability that decreases over time.
- **Ant Colony Optimization:**

# Thank You