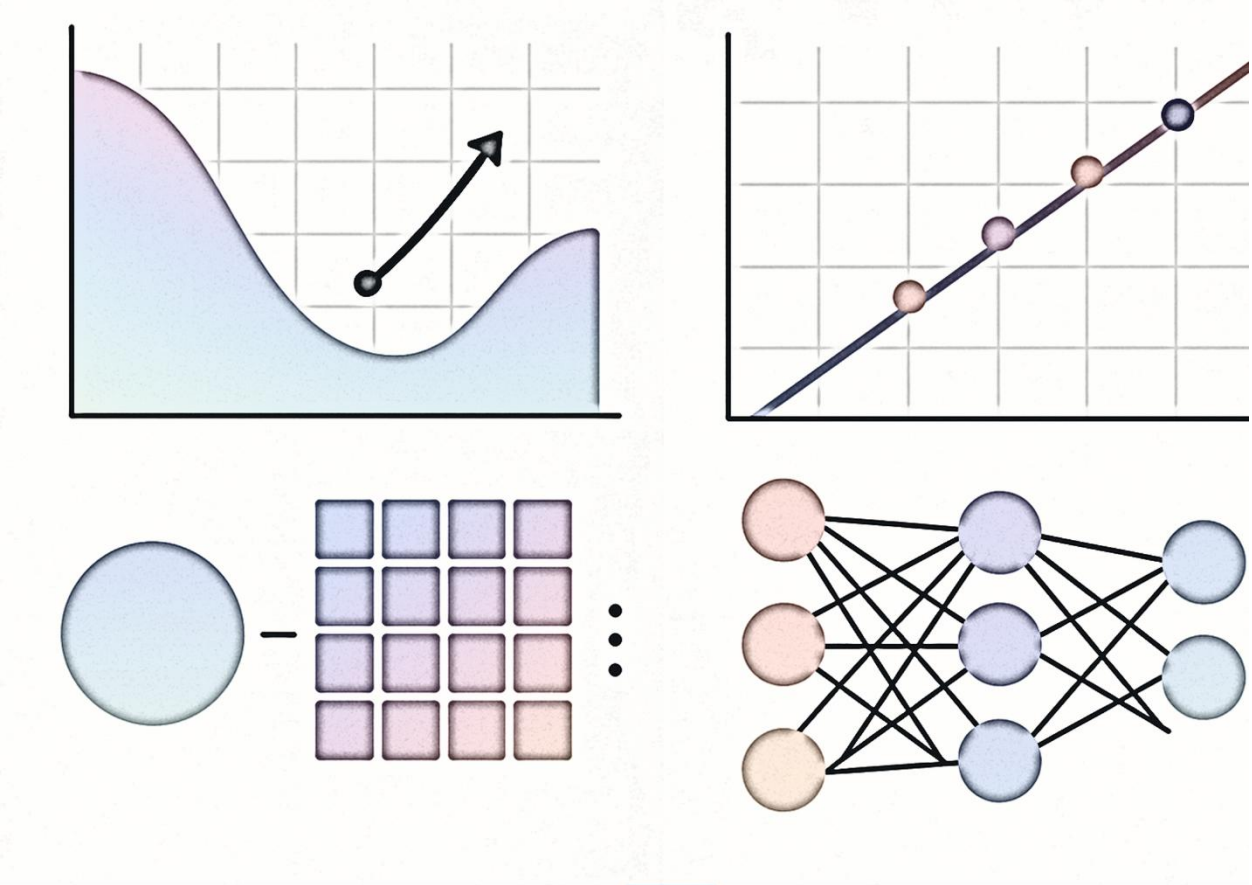


Computational Theory for AI

by

Ahn Nuzen

Acknowledgments

The author extends his genuine thanks to the dedicated faculty of the Grossmont College Computer Science Department for their invaluable support throughout the development of this ZBook.

Their thoughtful reviews, constructive feedback, and encouragement through many rounds of revision significantly strengthened the quality and clarity of this work.

This ZBook was made possible through funding from the Zero Textbook Cost (ZTC) Acceleration II Grant. The author is thankful to the ZTC team at Grossmont College for their support, guidance, and commitment to accessible education.

Ahn Nuzen
Grossmont College, Fall 2025

Overview

To truly understand and excel in Artificial Intelligence and Machine Learning, a solid foundation in several key mathematical theories and concepts is crucial. These theories provide theoretical underpinning for how algorithms work, how data is processed, and how models learn and make predictions.

The objective of this book, however, is not merely to master pure mathematics, but rather to develop a strong intuitive grasp of the underlying concepts and their relevance to machine learning algorithms. For those pursuing a career in AI and ML, it is highly advisable to engage with resources that emphasize the practical application of these mathematical tools, as this contextual understanding is essential for effective implementation and innovation in the field.

Learning Outcomes (Intuitive-Focused)

By the end of this book, the readers will be able to:

1. Foundations of AI & ML

Build a conceptually intuitive foundation in the key mathematical domains underpinning modern AI and machine learning—namely, linear algebra, probability, statistics, calculus, and optimization. Emphasis is placed on grasping the *why* behind the formulas, not just the *how*.

2. Analytical Thinking & Problem Solving

Leverage theoretical insights to interpret, design, and evaluate machine learning models. Develop the ability to translate complex, real-world challenges into well-defined algorithmic frameworks using structured reasoning.

3. Theory-to-Practice Integration

Bridge the gap between abstract theory and practical application through hands-on projects. These experiences reinforce how foundational concepts manifest in model behavior, performance trade-offs, and system design decisions.

Summary

This book provides a comprehensive yet accessible foundation in the core mathematical disciplines that underpin artificial intelligence and machine learning. Rather than focusing purely on formal proofs or abstract theory, this curriculum is designed to cultivate an intuitive understanding of mathematical principles—and how they directly inform the behavior and design of intelligent systems.

The reader will explore mathematical theories not as isolated subjects, but as tools for thinking—frameworks that allow machines to represent data, make decisions, and learn from experience. Each topic is motivated by its practical relevance in machine learning workflows and AI model design.

By the end, the reader will grasp the theory behind AI and gain the intuition and logic behind learning systems, helping you innovate, fix, and enhance models in real life situations.

Learning Philosophy:

- Conceptual Clarity Before Formalism: Visualizations, analogies, and real-world parallels come first—equations come later.
- From Theory to Insight: Emphasis is placed on *why* a concept matters in AI, not just *how* to compute it.
- Active Application: Through hands-on projects and algorithm simulations, you'll witness how math directly shapes model performance.

Computational Theory for AI Core Modules

Module	Title	Core Topics	Focus
1	**Vectors & Matrices**	Vector operations, matrix basics, dot product, norms	Representing and manipulating data
2	**Data Clustering**	Apply Vectors & Matrices Operations	Essential tools to organize disparity data into like groups
3	**Linear Transformations & Vector Spaces in Feature Extraction**	Linear maps, span, basis, subspaces, rank, projections	Map raw data into measurable properties or features that can be used by ML
4	**Eigenvalues & Eigenvectors**	Diagonalization, spectral theorem, PCA foundation	Finding the most important patterns in your data!
5	**Intro to Calculus for ML**	Limits, derivatives, basic functions	Foundations for optimization, Predictability of Data
6	**Multivariable Calculus**	Partial derivatives, gradient vectors	Quantifying the error between a machine learning model's prediction and the actual values in the training data
7	**Optimization & Gradient Descent**	Objective functions, gradient descent variants	Training models via optimization
8	**Advanced Calculus Tools**	Chain rule, Jacobian, Hessian	Backpropagation and second-order methods
9	**Probability Theory**	Random variables, distributions, independence	Modeling uncertainty and randomness
10	**Statistics & Inference for Data Accuracy**	Descriptive stats, hypothesis testing, confidence intervals	Drawing conclusions from data

Contents

Module 1: Vectors & Matrices as Data Features

In this module, we will explore one of the most fundamental concepts in computational mathematics and machine learning: data representation using vectors and matrices. Far from being a purely theoretical exercise, the use of vectors and matrices forms the essential foundation for all modern data science, underpinning the algorithms and techniques that drive analysis and insight across the field

1.1 Vector:

Mathematically, a vector is an ordered collection of numbers, an element of a vector space, defined by its magnitude and direction. It can be represented as a one-dimensional array. However, in machine learning, this abstract concept takes on a concrete and powerful role: vectors become the language we use to describe and represent complex, real-world entities, concepts, or data points to algorithms.

When we encounter a single data point - let us say, a house in our neighborhood - we cannot feed this qualitative concept directly into our algorithms. Instead, we must *quantify* its essential characteristics. The vector becomes our vehicle for this transformation from the tangible to the computational.

Take our typical house for example: a residential property characterized by the vector [v_1, v_2, v_3]. This seemingly simple array encapsulates three dimensions of information - square footage, bedrooms, and bathrooms respectively. Each element in this vector corresponds to a measurable attribute, what we term a "feature" or simply a single data point (e.g., a row in a dataset).

The notation to represent the vector for the house :

$$\vec{H} = [1500,3,2]$$

1.1.1 Vector Addition and Subtraction:

- Data Transformation: In machine learning, data is often represented as vectors (for example, feature vectors). Vector addition and subtraction allow us to manipulate these data points—such as combining features, calculating differences, or updating model weights during training
- Error Calculation and Optimization: During model training, operations like calculating error vectors (the difference between predicted and actual values) or adjusting weights (by subtracting or adding vectors) are routine. These processes directly depend on vector addition and subtraction

Notations and some examples of Vector Addition and Subtraction

Let two vectors be:

$$\vec{a} = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{bmatrix}, \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

Example:

$$\vec{a} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \vec{b} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

Vector Addition

$$\vec{a} + \vec{b} = \begin{bmatrix} a_1 & +b_1 \\ a_2 & +b_2 \\ a_3 & +b_3 \end{bmatrix}$$

Example:

$$\vec{a} + \vec{b} = \begin{bmatrix} 3 & +1 \\ 2 & +4 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$$

Vector Subtraction

$$\vec{a} - \vec{b} = \begin{bmatrix} a_1 & -b_1 \\ a_2 & -b_2 \\ a_3 & -b_3 \end{bmatrix}$$

Example:

$$\vec{a} - \vec{b} = \begin{bmatrix} 3 & -1 \\ 2 & -4 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$

These operations are component-wise, meaning you simply add or subtract each corresponding element of the vectors.

1.1.2 Dot Product:

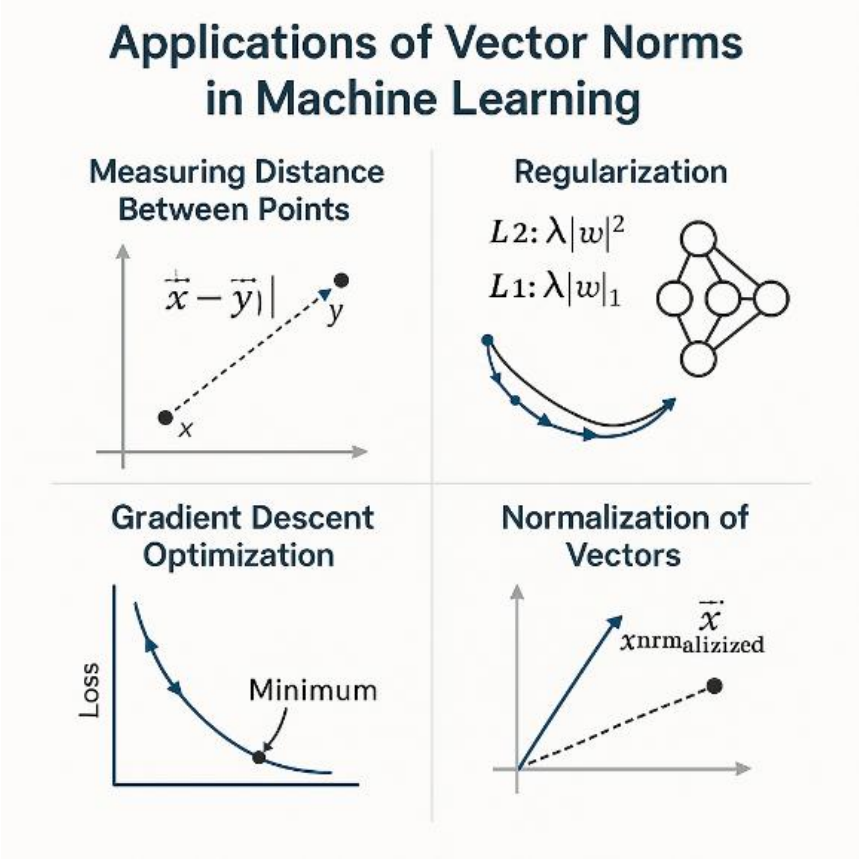
The dot product is an operation used to measure how similar two vectors are. In machine learning, this is essential for tasks like finding similar items, clustering data, and making recommendations. For example, in recommender systems, the dot product between user and item vectors helps predict which products a user might like.

Notations and some examples of Vector Dot Product

Vectors a, and b	Dot product is:
$\vec{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}, \vec{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$	$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2$
Example: $\vec{a} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \vec{b} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$	$\vec{a} \cdot \vec{b} = (2)(4) + (3)(3) = 8 + 9 = 17$

1.1.4 Vector Norm:

Vector norms are functions that assign a strictly positive length or magnitude to each non-zero vector in a vector space (and zero to the zero vector). This concept of "magnitude" is pivotal in numerous machine learning algorithms and processes.



To measure the length (magnitude) of a vector, we use:

L1 Norm: This norm is the sum of the absolute values of the vector's components:

$\|X\|_1 = \sum_{i=1}^{n} |X_i|$, where $\|X\|$ denotes the L_1 Norm of vector X

L2 Common norm: Euclidean norm (L2): $\|a\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$

AI/Machine Learning Application:

- Loss functions measure the discrepancy between the model's predicted outputs ($\hat{y}$) and the true target values (y). Many common loss functions are based on vector norms of the error vector ($e = y - \hat{y}$)
- Many machine learning algorithms require a metric to quantify the similarity or dissimilarity between data instances, which are represented as vectors in a feature space.
- In training machine learning models using iterative optimization algorithms (e.g., gradient descent), the norm of the gradient vector plays a crucial role.

1.2 Matrices:

The matrix, then, represents our natural progression from the singular to the collective. Where a vector encodes one observation, a matrix systematically organizes multiple observations into a coherent structure.

Visualize our dataset as a rectangular table: each row represents a distinct house (our individual data points), while each column represents a consistent feature measured across all houses. This row-by-column organization is not merely convenient - it is mathematically powerful, enabling us to apply linear algebraic operations across entire datasets simultaneously. Simply put, a matrix is a structure composed of numbers arranged in rows and columns, forming a two-dimensional array.

Example: Let's define matrix **H** with dimensions of 3x4 that represents a dataset of houses, where:

Each **row** corresponds to one house.
Each **column** is a **feature** (e.g., number of bedrooms, size in square feet, age of the house, etc.).
Matrix **H** with 3 houses (rows) and 4 features (columns):

HOUSE #	BEDROOMS	SIZE (SQ FT)	AGE (YEARS)	PRICE (IN $1000S)
1	3	1500	10	300
2	4	1800	5	400
3	2	1200	20	200

1.2.1 Matrix Notation:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 3 & 1500 & 10 & 300 \\ 2 & 4 & 1800 & 5 & 400 \\ 3 & 2 & 1200 & 20 & 200 \end{bmatrix}$$

Where Each column represents a feature:

- Column 0: House #
- Column 1: Bedrooms
- Column 2: Size (sq ft)
- Column 3: Age (years)
- Column 4: Price (target variable or label)

Matrix operations are fundamental to many AI and machine learning algorithms. Here are examples of matrix addition, subtraction, and multiplication and their applications:

1.2.2 Matrix Addition

Matrix addition is an element-wise operation. This means you add the corresponding elements of two matrices that must have the same dimensions.

$$If \boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} then \; \boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}$$

Example:

$$\boldsymbol{A} = \begin{bmatrix} 4 & 8 \\ 3 & 7 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 1 & 0 \\ 5 & 2 \end{bmatrix} then \; \boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} 4+1 & 8+0 \\ 3+5 & 7+2 \end{bmatrix} = \begin{bmatrix} 5 & 8 \\ 8 & 9 \end{bmatrix}$$

AI/Machine Learning Application:

- Updating Biases in Neural Networks:

  In a neural network layer, biases are often represented as a vector (or a matrix broadcasted to match dimensions). If you have an existing bias matrix and want to apply a global update or combine it with another set of bias adjustments, matrix addition would be used. For instance, if $B_{current}$ is the current bias matrix and B_{update} is the matrix of adjustments, the new bias is $B_{new}=B_{current}+B_{update}$.

- Image Processing (Blending/Averaging):

If images are represented as matrices of pixel values, adding two images (after potential normalization) can be a way to blend them or create an average image. For example, averaging multiple noisy images of the same scene to reduce noise.

- Combining Embeddings:

  If you have multiple embedding representations for items (e.g., word embeddings from different models) and want to create a combined representation by simple summation, matrix addition would apply if these embeddings were stored in matrices.

1.2.3 Matrix Subtraction

Similar to addition, matrix subtraction is an element-wise operation performed on matrices of the same dimensions.

$$If \boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} then \ \boldsymbol{A} - \boldsymbol{B} = \begin{bmatrix} a_{11} - b_{11} & a_{12} - b_{12} \\ a_{21} - b_{21} & a_{22} - b_{22} \end{bmatrix}$$

Example:
Let matrix $\mathbf{P}_{\text{predicted}}$ represent the predicted probabilities of two classes for three samples:

$$\boldsymbol{P}_{predicted} = \begin{bmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \\ 0.6 & 0.4 \end{bmatrix}$$

Let matrix $\mathrm{P}_{\text{actual}}$ represent the true one-hot encoded labels:

$$P_{actual} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Then, the error matrix $\mathbf{E} = \mathbf{P}_{\text{predicted}} - \mathbf{P}_{\text{actual}}$ (or $\mathbf{P}_{\text{actual}} - \mathbf{P}_{\text{predicted}}$, depending on the error definition):

$$\boldsymbol{E} = \begin{bmatrix} 0.8 - 1 & 0.2 - 0 \\ 0.3 - 0 & 0.7 - 1 \\ 0.6 - 1 & 0.4 - 0 \end{bmatrix} = \begin{bmatrix} -0.2 & 0.2 \\ 0.3 & -0.3 \\ -0.4 & 0.4 \end{bmatrix}$$

AI/Machine Learning Application:

- Calculating Error Vectors/Matrices:
  As shown in the above examples, subtracting the actual values matrix from the predicted values matrix is a common way to compute the error, which is then used in loss functions (e.g., components of Mean Squared Error before squaring).

- Background Subtraction in Image Processing:
  In computer vision, if you have a matrix representing a static background ($\mathbf{B}$) and a matrix representing the current frame ($\mathbf{F}$), $\mathbf{F}-\mathbf{B}$ can help identify moving objects by highlighting the differences.

- Feature Differencing:
  Creating new features by taking the difference between existing features. If features are organized in matrices, this can be done element-wise. For instance, if a matrix represents sensor readings at time t and another matrix represents readings at time $t-1$, their difference shows the change in readings.

1.2.4 Matrix Multiplication

Matrix multiplication is not an element-wise operation. To multiply matrix **A** (dimensions $m \times n$) by matrix **B** (dimensions $n \times p$), the number of columns in **A** must equal the number of rows in **B**. The resulting matrix **C** will have dimensions $m \times p$. Each element c_{ij} in **C** is the dot product of the i-th row of A and the j-th column of **B**.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Example:

Let Matrix **W** represent the weights in a simple neural network layer (2 input neurons, 3 output neurons). The dimensions of **W** are 3x2.

$$\boldsymbol{W} = \begin{bmatrix} 0.1 & 0.5 \\ 0.2 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$$

Let Matrix **X** represent the input features of a simple sample (2 features):

$\boldsymbol{X} = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$ (Dimensions: 2 x 1)

Then, the weighted sum **Z** = **W**·**X**:

$$\boldsymbol{Z} = \begin{bmatrix} 0.1x10 & +0.5x20 \\ 0.2x10 & +0.3x20 \\ 0.4x10 & +0.6x20 \end{bmatrix} = \begin{bmatrix} 1 & +10 \\ 2 & +6 \\ 4 & +12 \end{bmatrix} = \begin{bmatrix} 11 \\ 8 \\ 16 \end{bmatrix}$$ (Dimensions: 3 x 1)

If **X** contained inputs for multiple samples, e.g., 2 samples with 2 features each:

$X_{batch} = \begin{bmatrix} 10 & 5 \\ 20 & 1 \end{bmatrix}$(dimensions: 2 x 2)

then $Z_{batch} = W \cdot X_{batch}$:

$$\boldsymbol{W} \cdot X_{batch} = \begin{bmatrix} 0.1 & 0 \\ 0.2 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \cdot \begin{bmatrix} 10 & 5 \\ 20 & 1 \end{bmatrix} = \begin{bmatrix} 0.1 \cdot 10 + 0.0 \cdot 20 & 0.1 \cdot 5 + 0.0 \cdot 1 \\ 0.2 \cdot 10 + 0.3 \cdot 20 & 0.2 \cdot 5 + 0.3 \cdot 1 \\ 0.4 \cdot 10 + 0.6 \cdot 20 & 0.4 \cdot 5 + 0.6 \cdot 1 \end{bmatrix} =$$

$$\begin{bmatrix} 11 & 1 \\ 8 & 1.3 \\ 26 & 2.6 \end{bmatrix}$$

AI/Machine Learning Application:

- Forward Propagation in Neural Networks:
  This is the most prominent example. The output of a layer is typically calculated as $\mathbf{Z} = \mathbf{W} \cdot \mathbf{X} + \mathrm{b}$, where $\mathbf{W}$ is the weight matrix, $\mathbf{X}$ is the input matrix (or vector), and b is the bias vector. The core of this is the matrix multiplication $\mathbf{W} \cdot \mathbf{X}$.

- Calculating Gradients in Backpropagation: During the training of neural networks, the gradients of the loss function with respect to the weights and biases are computed using a series of matrix multiplications (chain rule).

Matrix operations are the bedrock of computation in most machine learning libraries like TensorFlow and PyTorch, as they allow for efficient parallel processing on GPUs.

1.2.5 Matrix Transpose

When we talk about the transpose of a matrix, we're essentially talking about *flipping* it across its main diagonal. Imagine taking every element in a row and moving it into a column. What was horizontal becomes vertical.

If $\mathbf{A}$ is an $\mathrm{m} \times \mathrm{n}$ matrix, then $\mathbf{A}$ transpose, denoted by A^T or sometime A'

Mathematically we say $[\mathbf{A}^{\mathsf{T}}]_{ij} = \mathbf{A}_{ji}$

Example:

$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ Dimensions 2 x 3 matrix. Its transpose is: $\boldsymbol{A}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

Now $\mathbf{A}^{\mathsf{T}}$ is a 3 x 2 matrix. Notice how the first row of **A** becomes the first column of $\mathbf{A}^{\mathsf{T}}$, and so on. When we transpose a matrix, we swap its rows and columns. So, the first row becomes the first column, the second row becomes the second column, and so on.

AI/Machine Learning Application:

- Linear Regression
  In linear regression, the goal is to find the best-fitting line (or hyperplane) that describes the relationship between input features and an output variable. This is often achieved by minimizing the sum of squared errors. The solution for the optimal weights (coefficients) can be expressed using matrix operations, including the transpose.

- Neural Networks (Backpropagation)
  In training neural networks, the backpropagation algorithm is used to update the network's weights. During the forward pass, inputs are multiplied by weight matrices. In the backward pass, the error is propagated backward through the network. When calculating the gradient of the error with respect to the weights of a particular layer, the transpose of that layer's weight matrix from the forward pass is often used.

1.3 Linear Algebra with Python

Python offers two powerful packages for efficiently solving linear algebra problems: **NumPy** and **SymPy**.

NumPy is the fundamental package for numerical computation in Python. It provides a high-performance multidimensional array object and tools for working with these arrays. For linear algebra, NumPy offers functions like np.dot() for dot products, **np.matmul()** or the **@** operator for matrix multiplication, and **np.linalg.norm()** to calculate the norm of a vector or matrix. It excels at performing efficient numerical operations on vectors and matrices.

However, in situations where a symbolic mathematical approach is more suitable, SymPy is often the preferred choice. SymPy is a Python library for symbolic mathematics, allowing you to work with mathematical expressions in a more abstract and exact way.

Python provides two powerful libraries for linear algebra: **NumPy** and **SymPy**. Each serves a distinct purpose:

- **NumPy** is optimized for high-performance **numerical** computations.
- **SymPy** is designed for **symbolic** mathematics, allowing algebraic manipulation of equations and expressions.

1.3.1 NumPy: Efficient Numerical Linear Algebra

NumPy excels in fast, vectorized operations on arrays and matrices. Common operations include:

```python
import numpy as np

# Define vectors
a = np.array([1, 2])
b = np.array([3, 4])

# Vector addition
add = a + b

# Dot product
dot = np.dot(a, b)
print("vector a, vector b", a, b,
"Addition:", add, "Dot product:",
dot)
```

```
vector a, vector b [1 2] [3 4] Addition: [4 6]
Dot product: 11
```

1.3.2 SymPy: Symbolic Linear Algebra

SymPy is used when exact symbolic results are required. It works with symbols and expressions as in traditional math.

```
from sympy import Matrix, symbols   #import Matrix and symbols from sympy
from IPython.display import display #display for displaying Math Symbols results in Notebook

# Define symbolic variables
x, y = symbols('x y')

# Define symbolic vectors
u = Matrix([x, 2])
v = Matrix([3, y])

display ("Symbolic vector u:", u)  # Display the symbolic vector u in Notebook
display ("Symbolic vector v:", v)  # Display the symbolic vector v in Notebook
# Vector addition
add = u + v
display("Vector addition:", add)  # Display the addition result in Notebook

# Dot product
dot = u.dot(v)
display("Vector dot product", dot)  # Display the dot product in Notebook
```

'Symbolic vector u:'

$$\begin{bmatrix} x \\ 2 \end{bmatrix}$$

'Symbolic vector v:'

$$\begin{bmatrix} 3 \\ y \end{bmatrix}$$

'Vector addition:'

$$\begin{bmatrix} x+3 \\ y+2 \end{bmatrix}$$

'Vector dot product:'

$$3x + 2y$$

Module 1: Learning Materials & References

A quick Audio Overview of Module 1

a) Scalars and Vectors

b) Introduction to Matrix
c) Introduction to Python Matrices and NumPy
d) Matrix Calculator
e) Linear Rank and Dependence
f) NumPy Getting Started
g) NumPy: the absolute basics for beginners
h) Basic of Vectors and Matrices with Python NumPy
i) Matrices – Practical Data Science
j) A Gentle Introduction to Vectors for Machine Learning
k) Mastering Matrices and Vectors in Machine Learning
l) Matplotlib for Beginners
m) Matplotlib Video Tutorial
n) Matplotlib Video Tutorial 2
o) Learn NumPy in 1 hour!

Assessment: Module 1: Vectors & Matrices as Data Features

Part A: Quiz

Multiple choice

1.1a In machine learning, a vector is used as a language to describe and represent what?

A) Purely theoretical mathematical concepts
B) Complex, real-world entities, concepts, or data points
C) Only qualitative data
D) Two-dimensional arrays

1.2a When representing a data point like a house using a vector [v1, v2, v3] where v1, v2, and v3 represent square footage, bedrooms, and bathrooms, what are v1, v2, and v3 commonly termed?

A) Dimensions
B) Elements
C) Features
D) Attributes

1.3a Vector addition and subtraction are performed by adding or subtracting each corresponding element of the vectors. What is this operation called?

A) Dot product-wise
B) Norm-wise
C) Magnitude-wise
D) Component-wise

1.4a Which vector operation is specifically mentioned as being used to measure how similar two vectors are?

A) Dot Product
B) Vector Addition
C) Vector Subtraction
D) Vector Norm

1.5a The L2 Norm, also known as the Euclidean norm, is a function that assigns a strictly positive what to each non-zero vector?

A) Similarity score
B) Direction
C) Length or magnitude
D) Difference

1.6a Where a vector encodes one observation, what does a matrix systematically organize?

A) A single data point
B) Multiple observations
C) A purely mathematical formula
D) A one-dimensional list

1.7a In a matrix representing a dataset of houses, what does each row typically correspond to?

A) A feature measured across all houses
B) A column in the matrix
C) A target variable or label
D) A distinct house (an individual data point)

1.8a For matrix addition to be possible between two matrices, what condition about their dimensions must be met?

A) They must have the same dimensions.
B) They must have different dimensions.
C) The number of columns in the first must equal the number of rows in the second.
D) They must both be square matrices.

1.9a What is a common application of matrix addition in neural networks?

A) Calculating the weighted sum of inputs
B) Updating Biases
C) Computing the dot product of layers
D) Determining the matrix transpose

1.10a What matrix operation is described as flipping the matrix across its main diagonal, causing rows and columns to swap?

A) Matrix Addition
B) Matrix Subtraction
C) Matrix Multiplication
D) Matrix Transpose

Short Answers:

1.11a What is a vector in machine learning, and what does it represent?

1.12a How is vector subtraction used in a simple way during machine learning model training?

1.13a What does it mean to transpose a matrix, and what is one simple way it is used in machine learning?

Part B: Vector Matrix Exercises

1.1b Vector Addition

Let $\vec{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \vec{b} = \begin{bmatrix} 4 \\ -1 \\ 0 \end{bmatrix} Find\ \vec{a} + \vec{b}.$

1.2b Vector Subtraction

Let $\vec{u} = \begin{bmatrix} 7 & -3 \\ 0 & -4 \\ -1 & -5 \end{bmatrix}, \vec{v} = \begin{bmatrix} 4 \\ -4 \\ -7 \end{bmatrix} Find\ \vec{u} - \vec{v}.$

1.3b Vector Norm (Magnitude)

Let $\vec{w} = \begin{bmatrix} 2 \\ -3 \\ 6 \end{bmatrix}, Find \|\vec{w}\|, the\ norm\ of\ \vec{w}.$

1.4b Matrix Addition

Let $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} Find\ A + B.$

1.5b Matrix Subtraction

Let $C = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, D = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} Find\ C - D.$

1.6b Matrix multiplication

Let $E = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, F = \begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix} Find\ E \cdot F$

Part C: Python Exercises:

If you are not familiar with NumPy and Matplotlib, please review module #1 references e-g before doing the exercises.

1.1c Vectors addition with NumPy

Create a Python code snippet to perform vector addition using NumPy module. Use the following vectors: $\vec{u} = [2,1,4], \vec{v} = [-1,3,2]$

```
Sample output:
Vector u: [2 1 4]
Vector v: [-1  3  2]
Vector (u + v): [1 4 6]
```

1.2c Vector subtraction with NumPy

Create a Python code snippet to perform vector subtraction using NumPy module. Use the following vectors: $\vec{u} = [2,1,4], \vec{v} = [-1,3,2]$

```
Sample output:
Vector u: [2 1 4]
Vector v: [-1  3  2]
Vector (u - v): [ 3 -2  2]
```

1.3c Dot products and scaling factor of 2

Create a Python code snippet to perform vector dot product, and scaling with factor 2 using NumPy module. Use the following vectors: $\vec{u} = [2,1,4], \vec{v} = [-1,3,2]$

```
Sample output:
Vector u: [2 1 4]
Vector v: [-1  3  2]
Vector (2 * u): [4 2 8]
Vector (2 * v): [-2  6  4]
Vector (u . v): 9
```

1.4c Vector Norms L1 & L2

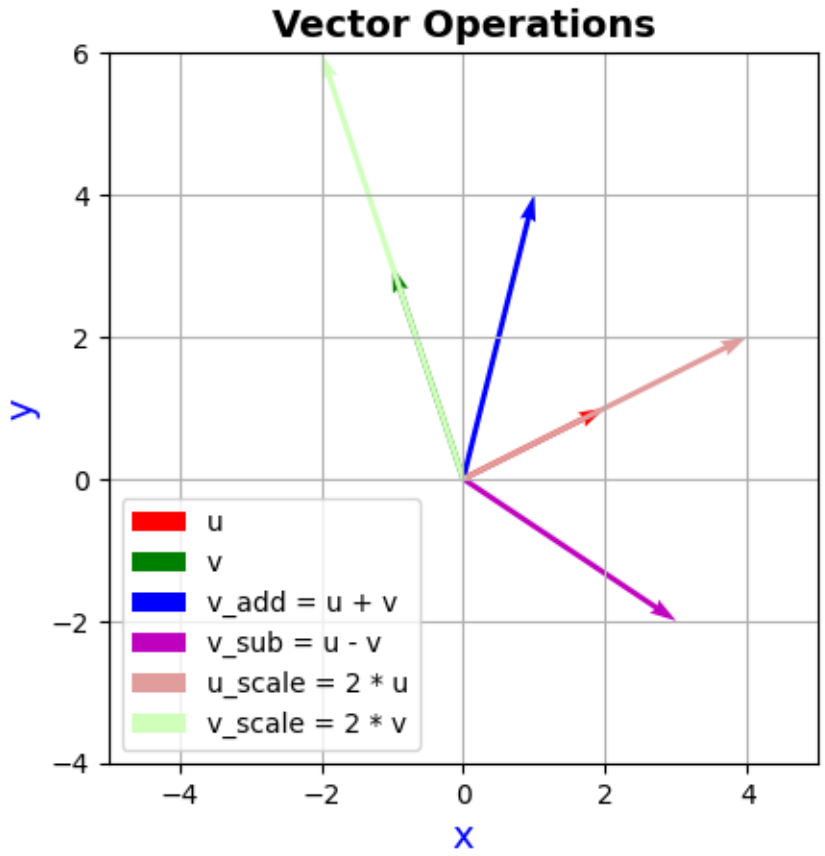
Create a Python code snippet to calculate L1&L2 norms on vectors. Use the following vectors: $\vec{u} = [2,1,4], \vec{v} = [-1,3,2]$

```
Sample output:
Vector u: [2 1 4]
Vector v: [-1  3  2]
L1 Norm of u: 7.00
L2 Norm of u: 4.58
L1 Norm of v: 6.00
L2 Norm of v: 3.74
```

1.5c. Visualizing Vectors Operations

Write a Python script to visualize vectors operations using NumPy and PlotMatLib. Use the following vectors: $\vec{u} = [2,1,4], \vec{v} = [-1,3,2]$

```
Sample output:

Vector u: [2 1 4]
Vector v: [-1  3  2]
Addition and Subtraction of Vectors:
Vector (u + v): [1 4 6]
Vector (u - v): [ 3 -2  2]
Scaling Vectors by factor of 2:
Vector (2 * u): [4 2 8]
Vector (2 * v): [-2  6  4]
Dot Product of Vectors:
Vector (u . v): 9
Vector Norms:
L1 Norm of u: 7.00
L2 Norm of u: 4.58
L1 Norm of v: 6.00
L2 Norm of v: 3.74
```



Notes: In Matplotlib, plt.quiver() (or ax.quiver() when working with an Axes object) is a powerful function used to create quiver plots. These plots are specifically designed to visualize vector fields in 2D (and 3D with mplot3d).

1.6c Use NumPy to compute the magnitude of a vector

$$\vec{v} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

1.7c Use SymPy to symbolically compute the magnitude of a vector

$$\vec{v} = \begin{bmatrix} sin(t) \\ t^2 \end{bmatrix}$$

Module 2: Data Clustering

This module applies the foundational concepts of matrix algebra and demonstrates their critical role in implementing data clustering algorithms, which are central to modern machine learning and data science workflows.

2.1 Understanding Clustering

Data clustering is a machine learning technique used to group a set of data points into clusters, where data points in the same cluster are more similar to each other than to those in other clusters.

Term	Description
Cluster	A group of similar data points.
Centroid	The center (average) of a cluster.
Distance Metric	A way to measure similarity or dissimilarity (e.g., Euclidean distance).
Unsupervised Learning	Clustering is **unsupervised**, meaning it doesn't rely on labeled data.

Suppose you have customer data for an online store: Age, Annual income, Spending score Clustering might group customers into segments like:

- Budget-conscious young shoppers
- Wealthy high spenders
- Middle-income occasional buyers

This information helps businesses tailor their marketing strategies, but clustering applications extend to other areas like document grouping, image compression, and anomaly detection.

2.2 Clustering Techniques

Numerous algorithms exist for cluster analysis. This book will explore some of the most common methods, such as:

Algorithm	Description
K-Means	Partitions data into K clusters by minimizing the distance to the cluster centroids.
Hierarchical Clustering	Builds nested clusters by either merging or splitting them successively.

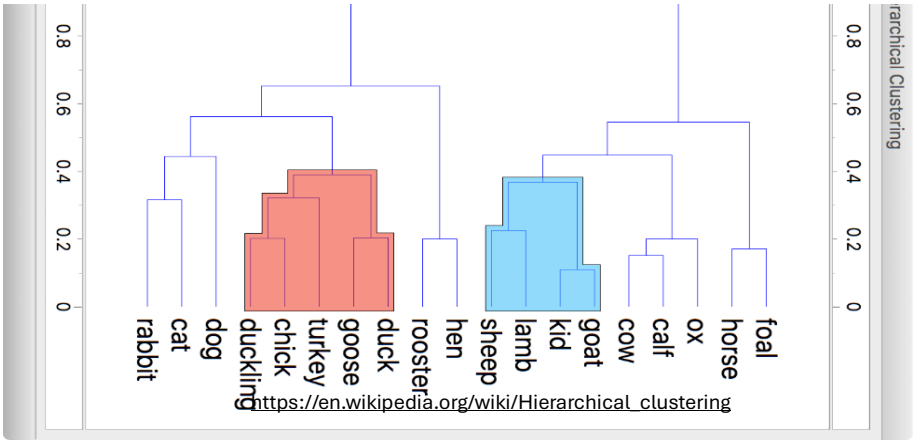
BIRCH Algorithm	Uses a tree structure to compress data into subclusters, leveraging matrix sums and centroids for efficient clustering, especially on large datasets
Spectral Methods:	Utilize matrix spectral properties for clustering, requiring advanced matrix computations such as eigen decomposition

2.2.1 K-Means Clustering:

- **Objective:** Partition data into k clusters by minimizing the within-cluster variance.
- **Implementation Steps:**
    - Randomly initialize k centroids.
    - Assign each data point to the nearest centroid.
    - Recompute centroids as the mean of the points in each cluster.
    - Repeat until convergence or a maximum number of iterations.
- **Matrix Operations:** Centroid updates and distance calculations rely heavily on matrix and vector operations for efficiency.

2.2.2 Hierarchical Clustering:

- **The main technique involves** building a tree of clusters by iteratively merging or splitting clusters based on similarity. Another way to visualize Hierarchical Clustering using a common example: animals. Imagine you have a list of animals, and you want to group them by how similar they are. The graph, called a dendrogram, helps us see these relationships visually. Think of it like a family tree that groups relatives together.
- **Implementation:** Uses pairwise similarity matrices and merging steps, both of which are matrix-based computations



2.2.3 Advanced Techniques

- BIRCH Algorithm: Uses a tree structure to compress data into subclusters, leveraging matrix sums and centroids for efficient clustering, especially on large datasets
- Spectral Clustering: Uses the spectrum (eigenvalues) of a similarity matrix to perform dimensionality reduction before clustering. This involves using matrix inversion, multiplication, and eigenvalue , and the Laplacian matrix from the affinity matrix,

2.2.4 Practical Implementation

- Vectorization: Use matrix operations instead of loops for centroid calculations and assignments, significantly speeding up the algorithm and making code more readable
- Data Structures: Represent data as matrices (e.g., $m \times n$ matrix for m data points with n features) to facilitate efficient computation

2.2.5 Animation of naïve k-means

The GIF animation from Wikipedia demonstrates a basic (naïve) version of the k-means clustering algorithm, which is used to group similar data points into a set number of clusters. In this example, the algorithm is set to find three clusters (k = 3).

The animation illustrates the algorithm's four key steps:

1. Initialization

Three initial centroids (shown as moving crosses) are randomly placed among the data points. Each centroid will represent the center of a cluster.

2. Assignment Step

Every data point is assigned to the nearest centroid. The points are colored (red, green, or blue) based on which centroid they are closest to, forming the initial clusters.

This uses Euclidean distance:

$$\text{Distance} = \|x_i - c_j\|^2 = (x_i - c_j)^\top (x_i - c_j)$$

To vectorize this for all points and centroids:

$$D_{ij} = \|x_i - c_j\|^2 = \|x_i\|^2 - 2x_i^\top c_j + \|c_j\|^2$$

Using broadcasting and dot products, to compute the full distance matrix $D \in R^{n \times k}$ efficiently:

$$D = \text{row\_norms}(X)^2 - 2XC^\top + \text{row\_norms}(C)^2$$

Where $\mathbf{D \in R^{n \times k}}$ D matrix elements belong to $\mathbf{R}$ real numbers, with $\mathbf{n \times k}$ is the dimension of the matrix row n col k.

3. Update Step

The algorithm recalculates each cluster's centroid by computing the mean position of all the points in that cluster. The centroids then move to these new positions.

$$C_j = \frac{1}{n_j} \sum_{i:A_{ij}=1} x_i$$

$$\text{then } C = (A^T A)^{-1} A^T X$$

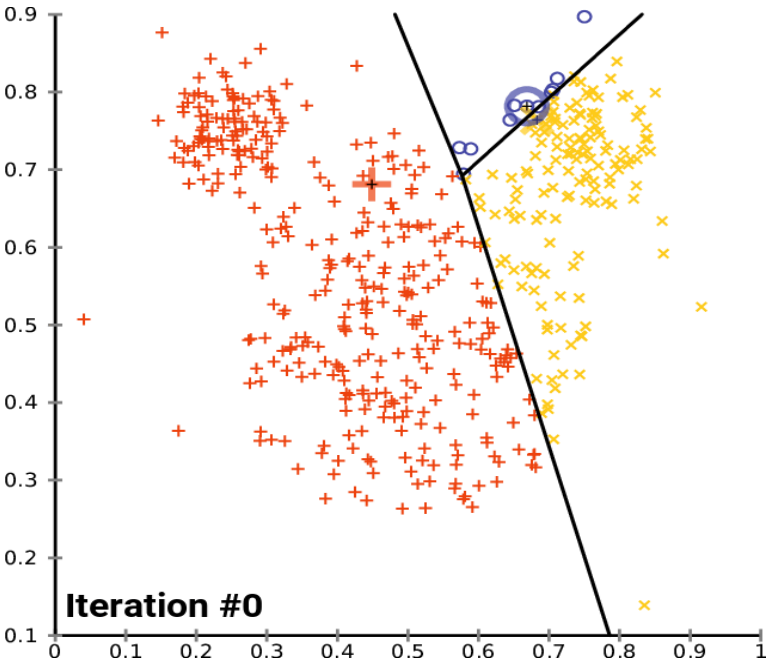
This averaging the points assigned to each cluster – this is a matrix version of computing centroids.

4. Iteration and Convergence

The algorithm repeats the assignment and update steps:

- Data points may change color as they are reassigned to the nearest updated centroid.
- Centroids continue to move as the clusters adjust.

This cycle continues until the centroids no longer move, and the cluster assignments stabilize. At this point, the algorithm has converged, and the final clusters are formed.



Animation of naïve k-means

2.2.6 Standard k-means clustering algorithm

This pseudocode outlines the basic steps of the standard k-means clustering algorithm.

Note: The way centroids are initialized, distances are measured, and new centroids are calculated can vary depending on the specific implementation.

In this version, we use argmin to find the index of the closest centroid for each data point.

```
//https://en.wikipedia.org/wiki/K-means_clustering
def k_means_cluster(k, points):
    # Initialization: choose k centroids (Forgy, Random Partition, etc.)
    centroids = [c1, c2, ..., ck]

    # Initialize clusters list
    clusters = [[] for _ in range(k)]

    # Loop until convergence
    converged = false
    while not converged:
        # Clear previous clusters
        clusters = [[] for _ in range(k)]
```

```
        # Assign each point to the "closest" centroid
        for point in points:
            distances_to_each_centroid = [distance(point, centroid) for centroid in centroids]
            cluster_assignment = argmin(distances_to_each_centroid)
            clusters[cluster_assignment].append(point)

        # Calculate new centroids
        #   (the standard implementation uses the mean of all points in a
        #     cluster to determine the new centroid)
        new_centroids = [calculate_centroid(cluster) for cluster in clusters]

        converged = (new_centroids == centroids)
        centroids = new_centroids

        if converged:
            return clusters
```

2.2.7 Example: K-Means implementation in Python

```
import numpy as np

def k_means_clustering(data, k, max_iterations=100):
    # Randomly initialize centroids
    centroids = data[np.random.choice(data.shape[0], k, replace=False), :]
    for _ in range(max_iterations):
        # Assign each point to the nearest centroid
        distances = np.linalg.norm(data[:, np.newaxis, :] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)
        # Update centroids
        new_centroids = np.array([data[labels == i].mean(axis=0) for i in range(k)])
        if np.all(centroids == new_centroids):
            break   # Converged if centroids are unchanged break, return labels and centroids
        # Handle cases where a centroid has no points assigned
        if np.any(np.isnan(new_centroids)):  # Handle NaN centroids
            # If a centroid has no points assigned, reinitialize it randomly
            centroids = centroids[np.isfinite(new_centroids)]
        else:
            centroids = new_centroids
    return labels, centroids
```

This implementation uses matrix operations for distance calculations and centroid updates, highlighting the importance of matrix algebra in clustering algorithms

Module 2: Learning Materials & References

1. Introduction to K-Means Clustering
2. Algorithms for Data Clustering
3. Linear Algebra for Data Science
4. K-means clustering

Assessment: Module 2: Data Clustering

Part A: Quiz

Multiple choice

2.1a Why do we need centroids in clustering techniques such as K-means?
A) To represent the center of each cluster and guide the assignment of data points
B) To define the boundaries between clusters
C) To measure the distance between clusters
D) To initialize the number of clusters

2.2a How is the centroid computed in a cluster during the K-means algorithm?
A) By selecting the data point closest to the mean
B) By calculating the arithmetic mean of all data points assigned to the cluster
C) By randomly choosing a new data point
D) By taking the median of all features

2.3a What is the first step in the naive K-means algorithm?
A) Assign each data point to the nearest centroid
B) Recompute the centroids as the mean of the points in each cluster
C) Randomly initialize K centroids
D) Calculate the distance between all pairs of data points

2.4a Which of the following steps is repeated until convergence in the K-means algorithm?
A) Only assigning data points to the nearest centroid
B) Only recomputing the centroids
C) Only calculating the distance between centroids

2.5a In K-means clustering, what is the main objective of the algorithm?
A) To maximize the distance between data points
B) To minimize the sum of distances between data points and their assigned centroids
C) To maximize the number of clusters
D) To minimize the number of data points

2.6a What is a common limitation of naive K-means clustering?
A) It requires labeled data
B) It is sensitive to the initial placement of centroids
C) It does not use centroids
D) It always finds the global optimum

2.7a Which matrix operation is fundamental for computing Euclidean distances between data points and centroids in K-means?
A) Matrix inversion
B) Outer product
C) Vector inner product
D) Matrix determinant

2.8a How is the centroid of a cluster mathematically computed after data point assignments?
A) Column-wise median of the data matrix
B) Column-wise mean of the data matrix
C) Row-wise sum of the data matrix
D) Diagonalization of the covariance matrix

Part B: Short Answers:

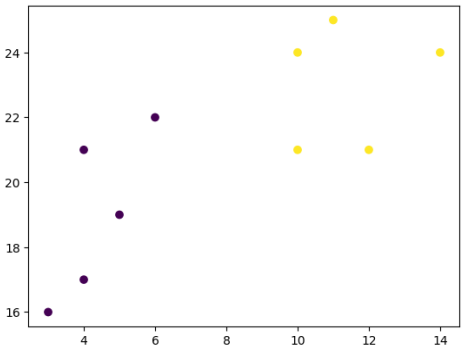
2.1b. Justify your response to the question 2.7a, show the math if required.
2.2b. How do linear algebra concepts explain the convergence properties of K-means?

Part C: Python Exercises:

2.1c Perform K-means Clustering on Simple Data

Given a simple 2D dataset, use sklearn.cluster.KMeans to cluster the data into 2 clusters and plot the results.

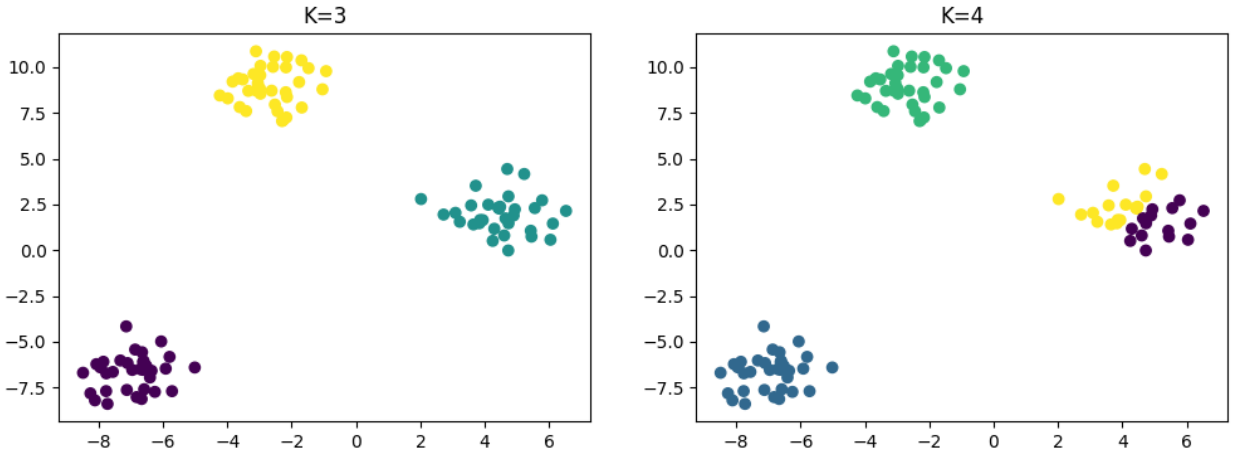
Sample output:



2.2c Visualize Clusters with Different Numbers of K

Given a synthetic dataset, use sklearn.cluster.KMeans to cluster the data into 3 and 4 clusters, and plot both results side by side for comparison.

Sample output:



2.3c Assign Cluster Labels and Compare with True Labels

Given a synthetic dataset with true labels, use K-means to cluster the data and compare the predicted labels with the true labels using a simple print statement.

Sample output:

```
First 10 predicted labels: [1 0 2 0 1 0 2 0 0 2]
First 10 true labels:      [2 1 0 1 2 1 0 1 1 0]
```

2.4c Use the Elbow Method to Determine the Optimal Number of Clusters

Apply the elbow method on a synthetic dataset to find the optimal number of clusters, then perform K-means clustering with this value.

What is elbow method?

The elbow method is a graphical technique used to help determine the optimal number of clusters (K) in a dataset for algorithms like K-means clustering. It works by plotting the within-cluster sum of squares (WCSS)—a measure of how tightly grouped the data points are within each cluster—for different values of K (number of clusters)

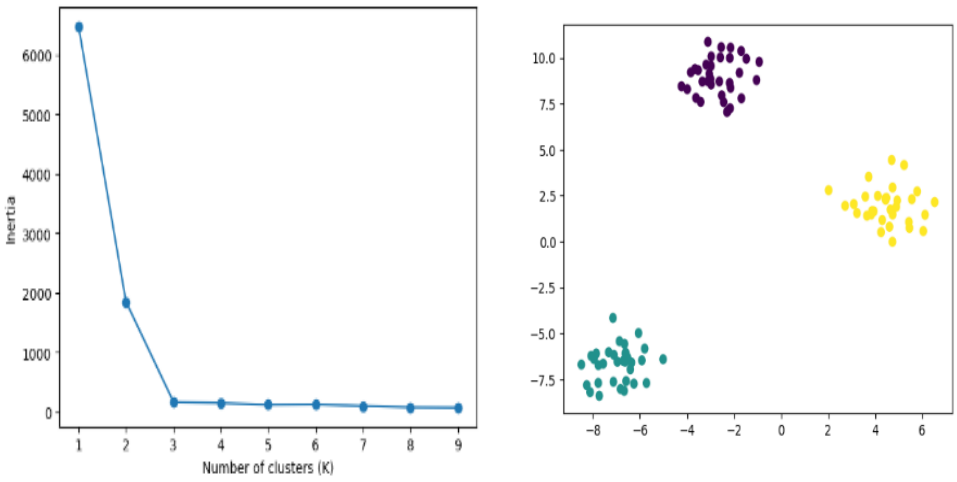
As you increase K, the WCSS typically decreases because each cluster becomes smaller and more tightly packed. The idea is to look for a point in the plot where the rate of decrease slows down noticeably, forming an "elbow" shape. The K value at this "elbow" is often chosen as the optimal number of clusters, as adding more clusters beyond this point yields only marginal improvements in clustering tightness

Key points:

- **WCSS:** Sum of squared distances between each data point and its assigned centroid.
- **Elbow:** The point on the graph where increasing K no longer leads to significant reduction in WCSS.
- **Purpose:** To balance between too few clusters (underfitting) and too many clusters (overfitting).

The elbow method is heuristic and may not always produce a clear result, especially with complex or high-dimensional data

Sample output:



Module 3: Linear Transformations & Vector Spaces in Feature Extraction

Linear algebra is foundational to modern data science and machine learning, especially in the context of feature extraction. This module introduces the core concepts of linear transformations, vector spaces, and their pivotal role in techniques like Principal Component Analysis (PCA).

3.1. Understanding Linear Transformations

There are two main rules that a transformation must follow to be considered "linear":

- **Lines Stay Lines:** If you take a straight line and apply a linear transformation to all of its points, you will end up with another straight line. The line might be stretched, shrunk, rotated, or flipped, but it won't become curved.
- **The Origin Stays Put:** The starting point of all vectors, the origin (the point at zero), never moves. It's the one fixed point in the entire space.

What This Looks Like

Imagine a grid of evenly spaced lines. When you apply a linear transformation to this grid, the lines of the grid might get tilted, stretched apart, or squeezed together, but they will remain parallel and evenly spaced.

Some common examples of linear transformations include:

- **Rotation:** Spinning the entire space around the origin.
- **Scaling:** Stretching or shrinking the space uniformly in all directions or differently along different axes.
- **Shearing:** Tilting one axis, so squares become parallelograms.

By following these simple rules, linear transformations provide a powerful way to manipulate and analyze geometric and data relationships in a structured way.

Now let’s formally define mathematical representations for linear transformations.

3.1.1 Linear Transformation Properties

A transformation T that maps vectors from a space $R^n$to a space R^m is linear if it satisfies the following two properties for all vectors $\vec{x}, \vec{y} \in R^n$and any scalar a:

1. Additivity: The transformation of a sum of two vectors is equal to the sum of their individual transformations.

   $$\mathbf{T}(\mathbf{x}+\mathbf{y})=\mathbf{T}(\mathbf{x})+\mathrm{T}(\mathbf{y})$$

2. Homogeneity (Scalar Multiplication): The transformation of a vector multiplied by a scalar is equal to the scalar multiplied by the transformation of the vector.

   $$\mathbf{T}(\mathrm{a}\mathbf{x})=\mathrm{a}\mathbf{T}(\mathbf{x})$$

These two properties are the formal definition of linearity. They ensure that the grid lines of the vector space remain parallel and evenly spaced after the transformation.

Matrix Representation

A key consequence of these properties is that any linear transformation $T: R^n \rightarrow R^m$ can be represented by a unique $\mathrm{m}\times\mathrm{n}$ matrix, often denoted as $\mathbf{A}$. The transformation of a vector **x** can be computed by multiplying it by this matrix:

$$\mathbf{T}(\mathbf{x})=\mathbf{A}\mathbf{x}$$

Where:

- $\mathbf{A}$ is the $\mathrm{m}\times\mathrm{n}$ transformation matrix. The columns of this matrix are the results of applying the transformation to the standard basis vectors of the input space R^n.
- $\vec{x}$ is a vector in the input space R^n, represented as an $\mathrm{n}\times 1$ column vector.

This matrix multiplication encodes the stretching, rotating, shearing, or reflecting actions of the linear transformation.

3.2 Understanding linear maps, spans, and bases.

Imagine space not as an empty void, but as an infinite grid of points, like a sheet of graph paper extending forever. Vectors are arrows from the origin (0,0) to any point on this grid.

3.2.1. Linear Maps (or Transformations): The Grid in Motion

A linear map (or linear transformation) is a rule for moving every point in the space, but it's a special kind of movement. It follows two strict rules that keep the grid orderly:

- Grid lines remain parallel and evenly spaced.
- The origin (0,0) stays put.

Think of it like this: take a sheet of rubber graph paper. You can stretch it, shrink it, rotate it, or "shear" it (turning squares into parallelograms), but you can't curve it, tear it, or move the center point.

3.2.2 Spans: What You Can Reach

The span of one or more vectors is all the points you can possibly reach by only using those vectors as your directions.

Visualization with One Vector:

- Imagine you have a single vector, let's call it $\vec{v}$. Think of $\vec{v}$ as a single "step" you're allowed to take. The span of $\vec{v}$ is all the points you can get to by taking any number of steps (or fractions of steps, forwards or backwards) in that exact direction.
- Visually: This creates an infinite line passing through the origin and the tip of the vector $\vec{v}$.

Visualization with Two Vectors:

- Now imagine you have two vectors, $\vec{v}$ and $\vec{w}$, pointing in different directions. Think of these as two different types of steps you can take. You can walk any distance along $\vec{v}$, and you can walk any distance along $\vec{w}$. The span is every single point you can reach by combining these movements.
- Visually: Unless the two vectors lie on the same line, their span is the entire 2D plane. You can reach any point on an infinite sheet of paper by simply scaling your two vectors up or down and adding them together. It's like having two non-parallel streets; you can get to any location in the city.

Thus, span is a set of all possible vectors that can be reached with a linear combination of a given pair of vectors called the span of those two vectors. $TheSpanof\,\vec{v} \wedge \vec{w}$ is the set of all their linear combinations. In other words, the span of most pairs of 2D vectors covers the entire two-dimensional space.

3.2.3. Bases: The Building Blocks of Space

A basis is a special set of vectors for a given space that follows two rules:

1. The vectors must span the entire space.
2. The set must be as small as possible; there are no redundant vectors.

Visualization:

- Think of a basis as the fundamental "directions" or "ingredients" for your space. For a 2D plane (like our graph paper), you need exactly two non-parallel vectors to form a basis. These are your building blocks.

- The most common basis is what we call the "standard basis." In 2D, these are the vectors $\hat{\imath}$ (a step of 1 along the x-axis) and $\hat{\jmath}$ (a step of 1 along the y-axis).
- With $\hat{\imath}$ and $\hat{\jmath}$, you can describe how to get to any point on the plane. The point (3, 2) simply means "take 3 steps in the $\hat{\jmath}$ direction and 2 steps in the $\hat{\jmath}$ direction."
- Crucially, a basis is not unique! Any two vectors that don't line up can serve as the basis for a 2D plane. For example, these two vectors $\vec{v}$ and $\vec{w}$ also form a valid basis. You can still reach any point, but your "directions" are now skewed.

In this new system, a point might be described as "1.5 steps in the $\vec{v}$ direction and 0.5 steps in the $\vec{w}$ direction." The location is the same, but the "address" or coordinates change because our basis vectors have changed. This is one of the most powerful ideas in linear algebra.

3.2 Features Engineering & Linear Independence

Imagine you're giving someone directions to a location in a city. You tell them to "go 2 blocks East and then 3 blocks North." Each instruction, "go East" and "go North," provides unique information. You can't describe the "North" part of the journey using only "East" instructions. In the language of linear algebra, the directions "East" and "North" are linearly independent.

Linear independence means that every vector (or feature) in a set provides new, unique information that cannot be created from a combination of the others. Linear dependence means at least one vector in the set is redundant.

3.2.1 How to Identify Linear Independence Mathematically

The formal way to identify linear independence is to see if any vector in a set can be written as a linear combination of the others.

Let's say we have a set of vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}$. We want to see if we can find weights (scalars) $c_1, c_2, \ldots, c_n$ that satisfy the following equation:

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v_n} = \mathbf{0}$$

where $\mathbf{0}$ is the zero vector.

There are two possible outcomes:

1. Linearly Independent: The *only* way to satisfy the equation is if all the weights are zero ($c_1 = c_2 = \cdots = c_n = 0$). This is called the trivial solution. It confirms that no single vector can be represented by a combination of others.
2. Linearly Dependent: There is at least one non-zero weight that makes the equation true. This is a non-trivial solution, and it proves that at least one vector is a combination of the others, making it redundant.

A Quick Example:

Let's take two vectors in a 2D plane:

$$v_1 = \begin{bmatrix}2\\1\end{bmatrix} v_2 = \begin{bmatrix}4\\2\end{bmatrix}$$

Visually we can see $\mathbf{v}_2$ is just 2 x $\mathbf{v}_1$, so we can conclude that these two vectors are linearly dependent.

Now let's apply the formulas mentioned above, can we find non-zero weights to make their combination zero? $c_1\mathbf{v}_1 + c_2\mathbf{v}_2 = 0$

$$c_1 = \begin{bmatrix}2\\1\end{bmatrix} + c_2 \begin{bmatrix}4\\2\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

Since we know that $\mathbf{v}_2$ is just 2 x $\mathbf{v}_1$, so we can set $c_1 = -2$, and $c_2 = 1$:

$$-2\begin{bmatrix}2\\1\end{bmatrix} + 1\begin{bmatrix}4\\2\end{bmatrix} = \begin{bmatrix}-4\\-2\end{bmatrix} + \begin{bmatrix}4\\2\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

We found non-zero weights, therefore, these vectors are linearly dependent.

3.2.1 Why Linear Independence is Critical for Feature Engineering

In machine learning, each column in your dataset is a feature, which can be thought of as a vector. Feature engineering is the art of creating new features or selecting the best ones to improve your model's performance. Here's why linear independence is so important in this process:

3.2.1a Avoiding Multicollinearity:

This is the single most important reason. When you have linearly dependent features, it means your model is getting the same information from multiple sources. This is known as multicollinearity. For many models, especially linear regression, this can cause significant problems:

- Unstable Coefficients: The model doesn't know which feature to assign importance to. A tiny change in your data can cause massive swings in the coefficients (the weights the model learns).
- Difficult Interpretation: You can't trust the coefficients to explain the relationship between a feature and the outcome. If "house size" and "number of bedrooms" are highly correlated, the model might put a large positive weight on one and a large negative weight on the other, which makes no sense.

In short, ensuring your features are as linearly independent as possible is a core principle of building robust, interpretable, and efficient machine learning models.

3.2.1b Reducing Dimensionality:

If you have features that are linearly dependent, you can remove the redundant ones without losing information. This simplifies your model, making it:

- Faster to train.
- Less prone to overfitting. A simpler model often generalizes better to new, unseen data.
- Easier to store and process.

3.2.1c Improving Model Performance:

By removing redundant features, you can often improve the performance and stability of your models. Techniques like Principal Component Analysis (PCA) are built on the idea of transforming your features into a new set of linearly independent features (principal components) and then keeping only the most important ones.

3.2.1d Understanding PCA

The main idea behind PCA is that by using statistical technique for reducing the dimensionality of large datasets while retaining as much of the original information as possible.

How PCA Works

1. Standardization: First, the data is standardized so that each feature has a mean of zero and a standard deviation of one. This step ensures that all features contribute equally to the analysis

2. Covariance Matrix Computation: Next, the covariance matrix of the data is calculated. This matrix captures how each pair of features varies together
3. Eigenvectors and Eigenvalues: The eigenvectors and eigenvalues of the covariance matrix are computed. The eigenvectors (principal components) represent the directions of maximum variance in the data, while the eigenvalues indicate the amount of variance explained by each principal component
4. Selecting Principal Components: The principal components are ordered by their eigenvalues, with the first component explaining the most variance. You can choose to keep only the top components that capture most of the data's variability
5. Projecting Data: Finally, the original data is projected onto the selected principal components, resulting in a lower-dimensional dataset that still contains most of the original information

3.2.2 Determinant and Feature extraction

The determinant of a matrix and feature extraction are closely connected in linear algebra and machine learning, especially when it comes to understanding data transformations, volume, and dimensionality.

The determinant is a scalar value computed from a square matrix. It tells us important geometric and algebraic properties of the linear transformation that the matrix represents.

The determinant of a 2 x 2 matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is given by: $|A| = ad - b$c. Denoted |A|

Now, consider the following example of a 2 x 2 matrix

$A = \begin{bmatrix} 4 & 2 \\ 0 & 2 \end{bmatrix}, |A| = (4 \cdot 2) - (1 \cdot 0) = 8 - 0 = 8$

3.2.2a. Feature Extraction as Linear Transformation

Feature extraction often involves transforming raw data (e.g., images, text, audio) into a new coordinate system using a matrix — like PCA or autoencoders.

- This transformation is linear (at least in PCA or linear models).
- The transformation matrix is applied to the original data.

- The determinant of this matrix helps us understand how the geometry of the data is affected.

3.2.2b. Determinant = Volume Change of Data Cloud

When you apply a feature extraction technique like PCA, you're rotating and projecting the data into a lower-dimensional space.

- If your transformation matrix has a non-zero determinant, you're preserving the dimensionality and some volume of the original data.
- If the determinant is close to zero, that means your transformation squashes the data along one or more dimensions — often what you *want* when doing dimensionality reduction.

Example:
Suppose your data lies on a plane in 3D space. After applying PCA, your matrix will project the data onto a 2D subspace. The transformation matrix has a determinant of 0 — meaning it has reduced dimensionality (collapsed the volume).

3.2.2c. Determinant and Rank

- If det(A) ≠ 0, the matrix is full-rank — its columns are linearly independent, and it spans the entire space.
- If det(A) = 0, the matrix is rank-deficient — its columns lie in a lower-dimensional space.

In feature extraction, you often *want* a lower-rank matrix — you're intentionally reducing features to extract only the most informative ones.

3.2.2.d Key Interpretations:

a) In 2D and 3D, the determinant indicates how a linear transformation scales area or volume.
b) If |det(A)| > 1, the transformation expands space.
c) If |det(A)| < 1, the transformation contracts space.
d) If det(A) = 0, the transformation flattens space (collapses it into a lower dimension).

3.3 Summary Table: Key Concepts of Module 3

Concept	Description	Application in Feature Extraction
Linear Transformation	Maps vectors while preserving addition and scaling	Basis change, data projection
Vector Space	Set of vectors closed under addition and scaling	Data representation
Basis	Linearly independent, spanning set of vectors	Coordinate system for data

Dimensionality Reduction	Projects data to fewer dimensions, retaining key information	Noise reduction, visualization
PCA	Linear transformation to maximize variance in new coordinates	Feature extraction, data compression
Determinant	a scalar value computed from a square matrix	Only extract the most informative ones

Assessment: Module 3: Linear Transformations & Vector Spaces in Feature Extraction

Part A: Quiz

Module 3: Learning Materials & References

a) Introduction to Matrix Dependencies/Ranking
b) Linear Algebra Column Space - Video
c) Linear Transformations & Python
d) Linear combinations, span, and basis vectors
e) Principal component analysis
f) PCA
g) Linear Transformation and Matrices
h) Symbols in Algebra

Multiple choice

3.1a In the context of Principal Component Analysis (PCA), what does the first principal component represent?

a) A randomly chosen feature from the original dataset.
b) The vector that captures the least variance in the data.
c) The linear combination of original features that captures the maximum variance.
d) The average value of all features in the dataset.

3.2a If a feature engineering process creates a new feature that is a linear combination of two existing features (e.g., new_feature = 2 * feature_A + 3 * feature_B), what is the immediate consequence for the dataset's vector space?

a) The number of basis vectors required to span the space increases by one.
b) The new set of features is now linearly dependent.
c) The origin of the vector space is shifted.
d) The transformation is non-linear.

3.3a Why is reducing the dimensionality of a feature space using techniques like PCA often beneficial for machine learning models?
a) It guarantees that the model will be 100% accurate.
b) It converts all features into a non-linear format.
c) It adds more information to the dataset, making it more complex.
d) It helps mitigate multicollinearity and reduces the risk of overfitting.

3.4a A data scientist has a dataset with three features: length_cm, length_in, and length_ft. What is the most accurate description of this feature space from a linear algebra perspective?
a) The feature space has a dimensionality of 1 because the features are linearly dependent.
b) The features are linearly independent and form a strong basis for a 3D space.
c) The span of these vectors is zero.
d) The features are orthogonal to each other.

3.5a Applying a rotation (a type of linear transformation) to a feature space before training a model would have what effect?
a) It would change the intrinsic relationships and distances between data points.
b) It would make the features linearly dependent.
c) It would preserve the distances between data points but represent them on a new basis.
d) It would delete important features from the dataset.

3.6a If the determinant of a transformation matrix used for feature extraction is zero, what does this imply about the transformation?

a) The transformation expands the feature space into a higher dimension.
b) The transformation compresses the feature space into a lower dimension (e.g., a 2D plane is squashed onto a line).
c) All the new features are perfectly uncorrelated.
d) The transformation is easily reversible without any loss of information.

Part B: Short Answers

3.1b A dataset used to predict the healthiness of food items contains features for calories_from_fat, calories_from_protein, calories_from_carbs, and total_calories. Explain why using all four of these features in a linear regression model is problematic?

3.2b Imagine you apply a linear transformation to your feature space that "squishes" all your data points from a 2D plane onto a single line. What is the determinant of this transformation matrix, and what does it signify about the data?

Part C: Python Exercises

3.1c Use NumPy package

Find a transformation matrix **T** Given a vector $\vec{v}_1$ = [2,3], that would rotate $\vec{v}_1$ 90 degrees.

Sample output:

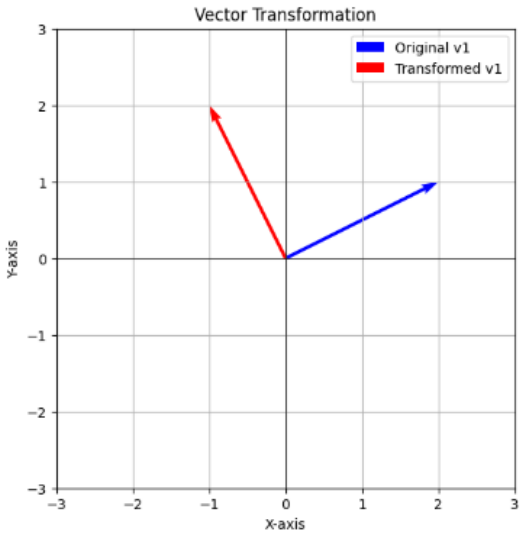
```
Given Vector (v1):[2 1]
?? the Transformation Matrix (T): to correctly rotate v1 by 90 degrees
Transformed Vector (T @ v1): [-1  2]

Hints: The rotation matrix for an angle theta is:
```

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

3.2c Using Matplotlib to visualize 3.1c

by plotting the original vector $\vec{v}_1$ alongside its transformed version.

Sample output:



3.3c Write a Python script that uses the determinant to check if a matrix is invertible (a common application in AI and linear algebra)

Sample output:

```
The matrix
 [[3 1]
 [2 4]]
is invertible.
Determinant: 10.000000000000002
```

3.4c Write a Python function that determines if two matrices are linearly dependent? Test the function with Matrix A and B.

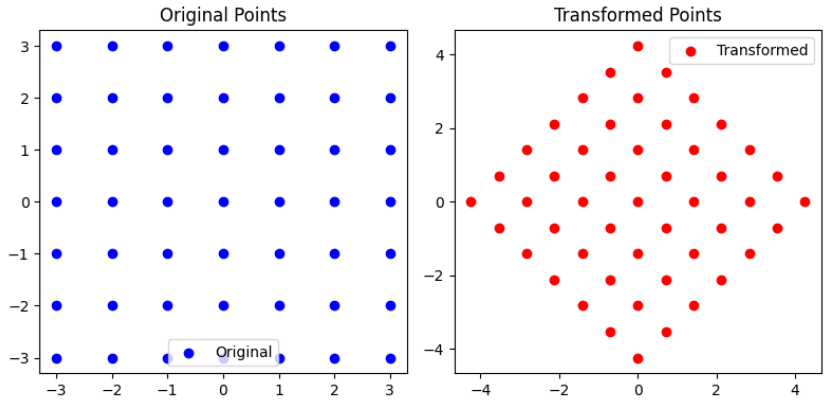
```
Sample output:
Matrix A:
[[1 2]
 [3 4]]
Matrix B:
[[2 4]
 [6 8]]
Are A & B linearly dependent?: True
```

3.5c Write a Python script that applies a Linear Transformation to a Set of Points (Grid) apply a linear transformation matrix that rotates the grid by 45 degrees. Plot the new rotated grid.

Use x, y = np.mgrid[-3:4, -3:4] as the mesh-grid like the one below.

```
[[-3, -2, -1, 0, 1, 2, 3],
 [-3, -2, -1, 0, 1, 2, 3],
 [-3, -2, -1, 0, 1, 2, 3],
 [-3, -2, -1, 0, 1, 2, 3],
 [-3, -2, -1, 0, 1, 2, 3],
 [-3, -2, -1, 0, 1, 2, 3],
 [-3, -2, -1, 0, 1, 2, 3]]
```

Sample output:



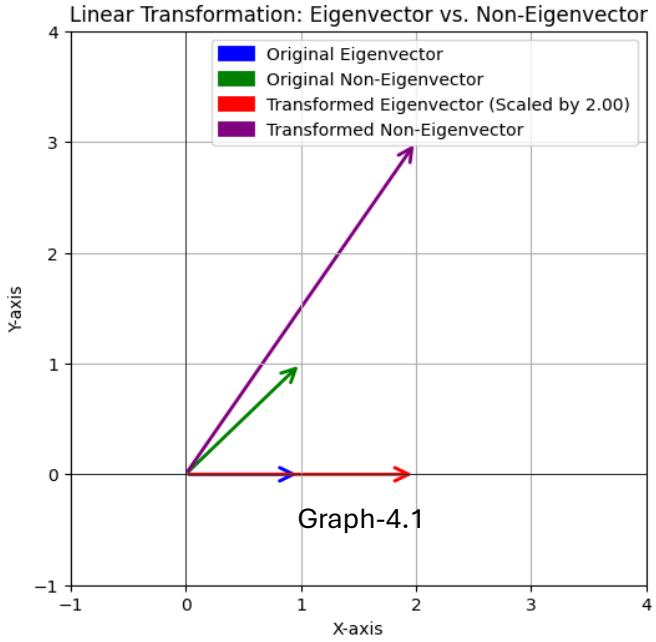
Module 4: Eigenvalues, Eigenvectors, and Data Patterns

An eigenvector is a special kind of vector that, when a linear transformation (like a stretch, shrink, or rotation) is applied to it, **only changes its length, not its direction.** The amount stretched or shrunk by is called the eigenvalue.

Example:

Given a matrix $A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$, and transformation matrix T = [1, 0]. Now let apply the transformation to vector A or $A.T$. The resulting of the multiplication is a vector [2 0]. Visually, we can see the red vector in the Graph-4.1, the original vector remains in the same direction but stretches horizontally (X-axis) by a value of 2. The value 2 is the eigenvalue, and the new transformed vector is the eigenvector.

Now let consider another transformation matrix T_1=[1,1]. Applying $A.T_1 =$ vector $[23]$. After the transformation, the green arrow would become a purple arrow ending at (2,3). The purple arrow is no longer pointing in the same diagonal direction as the green one; it has been both stretched and rotated.



Graph-4.1

The visual distinction is key: Eigenvectors are the special directions that only get stretched or shrunk by a transformation, without changing their fundamental orientation, and the eigenvalue is simply the amount of that stretch or shrink.

How Eigenvalues and Eigenvectors Apply to Machine Learning

In machine learning or ML, data often exists in high-dimensional spaces. Understanding the underlying structure of this data is crucial for various tasks like dimensionality reduction, feature extraction, and pattern recognition. Eigenvalues and eigenvectors provide a powerful framework for ML.

Principal Component Analysis (PCA):

This is one of the most prominent applications. Think of your data as a cloud of points in space. This cloud often has natural "directions" where the data varies the most.

The eigenvectors with the largest eigenvalues are called the principal components. These represent directions with the most variance in your data, and by projecting your data onto the top *k* eigenvectors, the dataset is reduced to *k* dimensions, keeping the most important information while discarding the noise.

Eigenvalues tell us how significant each of these principal directions is. A larger eigenvalue indicates that the corresponding eigenvector captures more of the data's variance (i.e., more information). By selecting eigenvectors with the largest eigenvalues, we can reduce the dimensionality of the data while retaining most of its crucial information.

Google PageRank Algorithm: Eigenvectors are used to determine the importance of web pages. The eigenvector corresponding to the largest eigenvalue of the web's link matrix helps rank pages based on their connectivity.

Spectral Clustering: This technique uses eigenvectors to group data points based on their connections in a graph, providing a way to cluster data based on its underlying structure.

Image Processing (e.g., Eigenfaces for Facial Recognition): Eigenvectors are used to represent images as linear combinations of significant features, enabling efficient facial recognition systems.

4.1 Definition of Eigenvalues, Eigenvectors

For a square matrix A:

- An eigenvector $\vec{v}$ is a non-zero vector that satisfies the equation: $A\vec{v} = \lambda\vec{v}$
- The corresponding **eigenvalue** λ (lambda) is the scaling factor

In other words, when matrix A multiplies an eigenvector $\vec{v}$, the result is the same as multiplying $\vec{v}$ by a scalar or just a number λ .

4.1.1 How to Find Eigenvalues and Eigenvectors

To find eigenvalues of a matrix A, solve the **characteristic equation**:

$$det(A - \lambda I) = 0$$

Where λ represents the eigenvalues, and I is the identity matrix of the same dimension as A.

To find the eigenvectors corresponding to each eigenvalue λ, solve the homogeneous system:

$$(A - \lambda I)\vec{v} = \vec{0}$$

Where $\vec{v}$ is the eigenvector and I is the identity matrix.

Let's work through a simple example: $A = \begin{bmatrix} 3 & 2 \\ 1 & 2 \end{bmatrix}$

Step 1: Find the eigenvalues

Use the **characteristic equation:** $det(A - \lambda I)=0$

$$\begin{bmatrix} 3 & 2 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} 3-\lambda & 2-0 \\ 1 & 2-\lambda \end{bmatrix}$$

Compute the determinant:

$(3 - \lambda)(2 - \lambda) - (2)(1) = 0$

$\lambda^2 - 5\lambda + 6 - 2 = 0$

$\lambda^2 - 5\lambda + 4 = 0$

$(\lambda - 4)(\lambda - 1) = 0$

So:

- $\lambda_1 = 4$
- $\lambda_2 = 1$

Eigenvalues: 4 and 1

Step 2: Find the eigenvectors

solve the homogeneous system $(A - \lambda I)\vec{v} = 0; For \lambda = 4$:

$$(A - 4I) = \begin{bmatrix} 3 & 2 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 3-4 & 2 \\ 1 & 2-4 \end{bmatrix} = \begin{bmatrix} -1 & 2 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This gives us: $-v_1 + 2v_2 = 0 > v_1 = 2v_2$

So, the eigenvector is any scaler multiple of

$$\overrightarrow{v_1} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

solve the homogeneous system $(A - \lambda I)\vec{v} = 0; For \lambda = 1$:

$$(A - I) = \begin{bmatrix} 3 & 2 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$This gives us: 2v_1 + 2v_2 = 0 = v_1 = -v_2$

So, the eigenvector is any scaler multiple of

$$\overrightarrow{v_2} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Eigenvalue (λ)	Eigenvector ($\vec{v}$)
1	$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$
4	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

4.2 How eigenvectors define principal directions of data variation.

Eigenvectors reveal the "natural axes" of your data—the directions along which your data points naturally spread out or vary. Think of them as uncovering the hidden coordinate system that best describes your data's structure.

4.2.a. Direction of Maximum Variance

The first eigenvector (with the largest eigenvalue) points in the direction where your data varies the most. If you were to project all your data points onto a single line, the line parallel to this eigenvector would capture the maximum possible variance.

4.2.b. Orthogonal Directions

Each subsequent eigenvector is perpendicular (orthogonal) to the previous ones and captures the next highest amount of variance in the data. Together, they form a new coordinate system perfectly tailored to your data's structure.

4.2.c. Natural Data Coordinates

When you transform your data to use eigenvectors as axes:

- The first coordinate represents position along the direction of maximum variation
- The second coordinate represents position along the second most important direction
- And so on...

4.2.d. Geometric Interpretation

Imagine your data as an elliptical cloud of points:

- The eigenvectors point along the major and minor axes of this ellipse

- The eigenvalues tell you how long each axis is (how far the data extends in each direction)
- The ratio between eigenvalues tells you how "stretched" the data is in different directions

4.2.e. Covariance Matrix

In simple terms, covariance measures how two variables change together: If both variables increase or decrease together, the covariance is positive. If one variable increases while the other decreases, the covariance is negative. A zero covariance means there's no consistent relationship between their changes.

A covariance matrix shows the covariances between all pairs of variables in your dataset. For a dataset with n features (columns), the covariance matrix is an $n \times n$ table.

- The diagonal elements show variances (how spread out each feature is).
- The off-diagonal elements show relationships (covariances) between different features.

4.3 Applying Eigenvectors and Eigenvalues in PCA to reduce complexity while preserving information.

Principal Component Analysis (PCA) is a direct application of eigenvectors and eigenvalues to understand and transform data.

PCA simplifies complex data by reducing the number of variables while keeping the most important information. It combines your original features into a smaller set of new, uncorrelated features called principal components. This makes large datasets easier to work with and visualize.

Principal Component Analysis (PCA) identifies the most important directions of data spread by mathematically determining the directions—called principal components—along which the data exhibits the largest variance. Here's how this process works:

4.3.a Covariance Matrix Calculation

PCA begins by computing the covariance matrix of the dataset. This matrix captures the relationships and variability between all pairs of variables

4.3.a Finding the Directions of Maximum Variance

The eigenvectors of the covariance matrix point in the directions where your data varies the most. The first principal component (the eigenvector with the largest eigenvalue) points in the direction of maximum variance.

4.3.b Eigenvalues Quantify the Importance of Each Direction

The eigenvalues tell us how much variance is explained by each eigenvector (principal component). Larger eigenvalues indicate more important directions.

4.3.c Dimensionality Reduction Through Eigenvector Selection

By keeping only the eigenvectors with the largest eigenvalues, we can reduce dimensions while preserving most of the information.

In summary, PCA uses the eigenvectors and eigenvalues of the covariance matrix to find the directions where the data is most spread out, and orders these directions (principal components) by their importance based on the amount of variance they explain

4.4 Linear Algebra with Python

Python provides two powerful libraries for linear algebra: **NumPy** and **SymPy**. Each serves a distinct purpose:

- **NumPy** is optimized for high-performance **numerical** computations.
- **SymPy** is designed for **symbolic** mathematics, allowing algebraic manipulation of equations and expressions.

The following example illustrates how to use Python Symbolic library SymPy to find the Eigenvectors and values for example in 4.1.1

```
# import sympy as sp
1.import sympy as sp
# Define the matrix A
2.A = sp.Matrix([[3, 2],
3.                [1, 2]])

# Step 1: Compute eigenvalues and eigenvectors
4. eigen_data = A.eigenvects()
5. lambda1, _, [v1] = eigen_data[0]
lambda2, _, [v2] = eigen_data[1]
print("lambda1 & Lambda2 :", lambda1, lambda2)
# Display eigenvalues and corresponding eigenvectors
for eigenval, multiplicity, vects in eigen_data:
    print(f"Eigenvalue: {eigenval}")
    print(f"Eigenvector(s):")
    for v in vects:
        sp.pprint(v)
    print()
```

```
Output:
lambda1 & Lambda2 : 1 4
Eigenvalue: 1
Eigenvector(s):
⎡-1⎤
⎢  ⎥
⎣1 ⎦

Eigenvalue: 4
Eigenvector(s):
⎡2⎤
⎢ ⎥
⎣1⎦
```

Explanation:

Line 4: computes all the eigenvalues and vectors of matrix A, it returns a list of tuples one of each eigenvalue. The tuple is of theform:
(eigenvalue, algebraic multiplicity, [eigenvector(s)]
Ex:

```
 [
  (4, 1, [Matrix([[2], [1]])]),
  (1, 1, [Matrix([[-1], [1]])])
  ]
```

Line5: lambda1, _, [v1] = eigen_data[0]
lambda1 gets the first eigen_data:
_ underscore not used multiplicity
V1 unpacks the list of eigenvectors
After unpacking:

```
lambda1 = 4, v1 = Matrix([, ])
lambda2 = 1, v2 = Matrix([[-1], ])
```

Module 4: Learning Materials & References

a) Determinant of a Matrix
b) Math is Fun Eigenvalue
c) Questions Database
d) Principal Component Analysis (PCA): A Step-by-Step Explanation
e) Eigenvectors and eigenvalue Video
f) Principal Component Analysis (PCA) Video
g) Eigenvectors & eigenvalues with Python
h) Greek Letters

Assessment: Module 4: Eigenvalues, Eigenvectors, and Data Patterns

Part A: Quiz

Multiple choice

4.1a If v is an eigenvector of matrix A with eigenvalue λ, which of the following statements is TRUE?

A) Av = v/λ
B) Av = λv
C) Av = v + λ

D) λv = A
E) v = λA

4.2a let A = $\begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$ and suppose λ = 4 is an eigenvalue of A, which of the following is a corresponding eigenvector?

A. $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, B. $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$, C. $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$, D. $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

4.3a What are the eigenvalues of the matrix $\begin{bmatrix} 4 & 0 \\ 2 & 3 \end{bmatrix}$?

A) 3 and 4
B) 0 and 2
C) 2 and 5
D) -1 and 8
E) 7 only

4.4a Which of the following statements about a 3×3 matrix A is TRUE?

A) A is invertible if and only if all its eigenvalues are positive
B) A is invertible if and only if none of its eigenvalues is zero
C) If A has three distinct eigenvalues, then A must be symmetric
D) The trace of A equals the product of its eigenvalues
E) A has exactly three eigenvectors

4.5a A 3×3 matrix A has eigenvalues $\lambda_1 = 2$, $\lambda_2 = 2$, and $\lambda_3 = 5$, with corresponding linearly independent eigenvectors v_1, v_2, and v_3. What is the result of applying A^3 (A cubed) to the vector v_3?

A) $8v_3$
B) $25v_3$
C) $125v_3$
D) $30v_3$
E) $15v_3$

Part B: Eigenvalues, Vectors problems

4.1b Find eigenvalues and vector

Given a 2×2 Matrix find eigenvalues and vector by solving the characteristic equation$det(A - \lambda I) = 0$.

$A = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}$

4.2b Application to a Linear Dynamical System

Consider the linear system of differential equations: $\frac{dx}{dt} = Ax, where A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$

Find the general solution that satisfies the initial condition $x(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Part C: Python Exercises

4.1c Write a python script to solve 4.1b

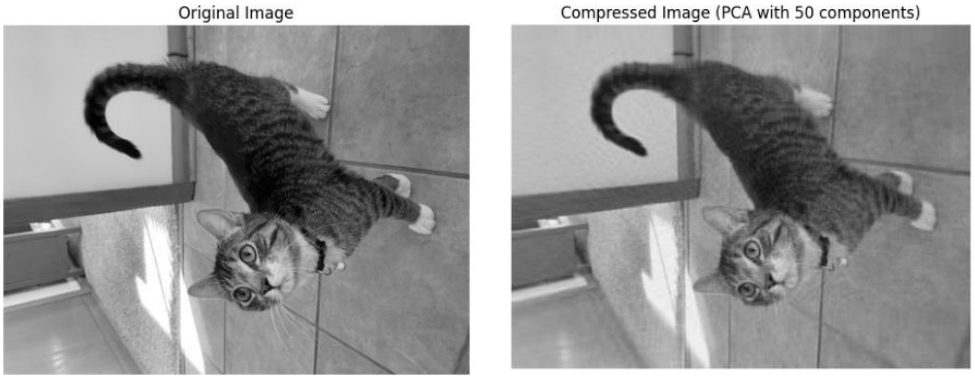
4.2c Write a python script to solve 4.2b

4.3c Write a python script using PCA from sklearn module to compress a simple image in gray scale.

Tasks:

1. Download cat image from https://en.wikipedia.org/wiki/Domestication_of_the_cat#/media/File:Domesticated_indoor_cat_WI.jpg

2. Display original image and compressed image



3. Display original image size, and final compressed size.

Original size: 12,192,768 bytes
Compressed size: 352,800 bytes
Final size: 34.56x smaller

Module 5: Calculus Basics for Machine Learning

This module offers a basic introduction to several key calculus concepts—particularly derivatives—to equip learners with the skills needed to apply essential formulas in machine learning contexts.

5.1 Goals and Objectives

a) Understanding Derivatives: a derivative represents the rate of change of a function and its slope at a given point, which is crucial for optimization in machine learning models.
b) Graphical Intuition: this module emphasizes using graphs to visualize how functions behave, helping readers intuitively grasp the dynamics of functions as they change
c) Smoothness and Curvature: Module will explore how the smoothness and curvature of functions impact model training, particularly in terms of convergence and stability during optimization processes.

5.2 Understand The Concept of a Derivative as a Rate of Change & Slope.

The derivative is a concept in calculus that helps us understand how things change. It tells us how fast something is changing at a specific moment. You can think of it in two main ways:

5.2a Derivative as Slope

Imagine you are driving a car. The derivative is like the speedometer showing how fast you are going at any given moment. If you look at a hill, the slope of that hill tells you how steep it is. In the same way, the derivative tells you the steepness of a curve at a particular point. If the curve goes up quickly, the derivative is a large number; if it goes up slowly, the derivative is a smaller number.

5.2b Derivative as Rate of Change

The derivative also shows how one thing changes in relation to another. For example, if you are tracking the growth of a plant, the derivative can tell you how fast the plant is growing at a certain time. If the plant grows quickly, the derivative is high; if it grows slowly, the derivative is low.

Everyday Examples

- Speed: If you drive faster, your speed (derivative) increases.
- Population: If a city's population is growing quickly, the derivative shows rapid growth.

In summary, the derivative helps us understand and measure how things change, whether it's speed, growth, or any other changing quantity.

5.2c Definition of First Derivative

The derivative of a function at a point gives the slope of the tangent line to the function's graph at that point. Mathematically, if you have a function $y=f(x)$, the derivative $f'(x)$ is defined as:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Where h is some delta x or Δx

This formula calculates the slope by taking the difference in the function's values (the rise) and dividing it by the difference in the input values (the run) as the interval shrinks to zero.

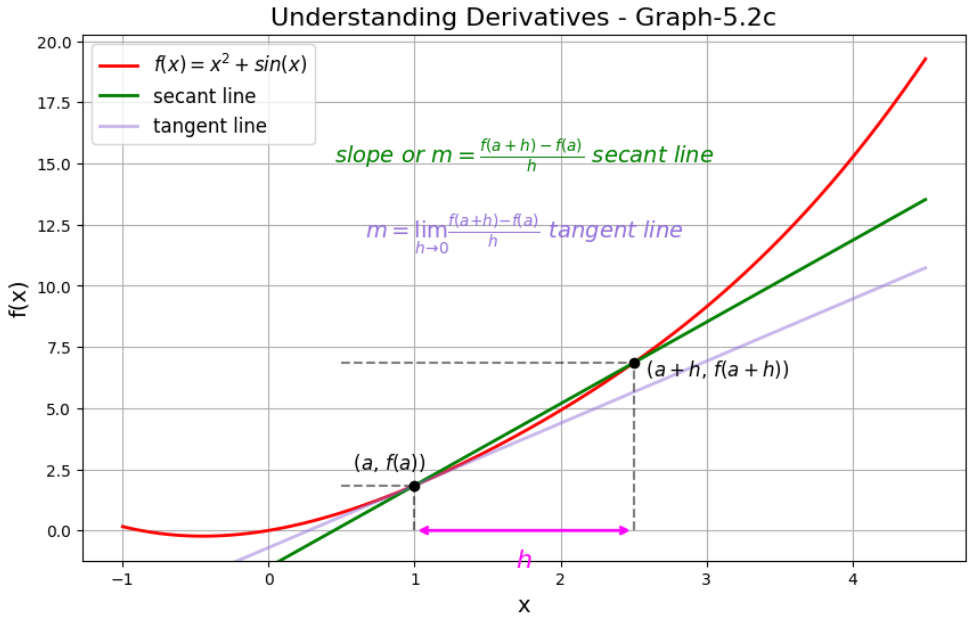
$$Slope = \frac{Change \in Y}{Change \in X} = \frac{\Delta y}{\Delta x}$$

$$\frac{\Delta y}{\Delta x} = \frac{f(x+h) - f(x)}{h}$$

To be more specific, the derivative of $f(x)$ at some point **a**:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

Graph-5.c illustrates the relationship between the slope of the secant line and the limit of tangent line as h approaching zero.



Thus, the derivative of a function at some point input is equal to the slope of the tangent line at that point

Let's look at an example:

Find the derivative of the function $f(x) = x^2$.

Recall that $f'(x) = \frac{\lim_{h\to 0}(x+h)^2 - x^2}{h}$

$$= \frac{\lim_{h\to 0} x^2 + 2hx + h^2 - x^2}{h}$$

$$= \frac{\lim_{h\to 0} 2hx + h^2}{h}$$

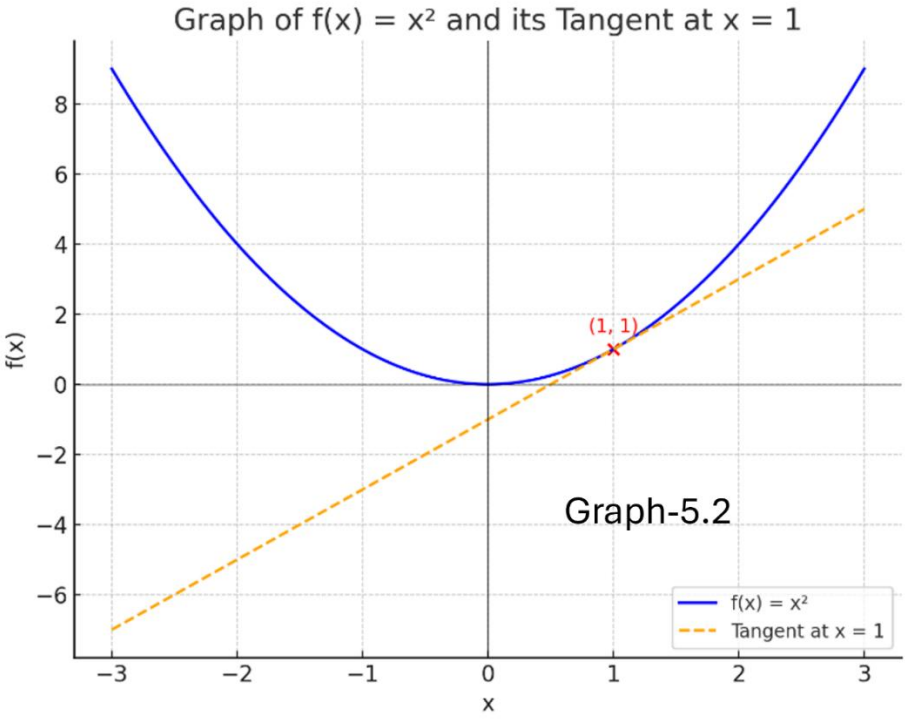
$$= \lim_{h\to 0}(2x + h)$$

$$= 2x$$

The derivative of x^2 is 2x, in other words the slope at x is 2x

Graph-5.2 illustrates the rate of change of $f(x) = x^2$:

1. The blue curve is the function $f(x) = x^2$.



2. The orange dashed line is the tangent line to the curve at $x = 1$, which has a slope of $f'(1) = 2$.
3. The red dot marks the point (1,1), where the function and tangent line touch.

The tangent line represents the **instantaneous rate of change** (or derivative) at that point on the curve.

5.2.d Common First Derivative Rules

While formal definition involves limits, there are several common rules that make finding derivatives much simpler for a wide range of functions.

In mathematics, there are two common notations for derivatives. One is the *prime notation* (e.g., $f'(x)$, which we have used so far. Another widely used form is the *Leibniz notation* ($\frac{d}{dx}$), as in $\frac{dx}{dy}$. Both notations are interchangeable and express the same concept—they denote the rate of change of a function with respect to a variable.

Here are some simple rules to find the derivatives.

1. Constant Rule:

Formula: $f'(x) = \frac{d}{dx}(c) = 0$ where c is any constant.

Explanation: The rate of change of constant value is always zero. A horizontal line (representing a constant function) has a slope of zero everywhere.

Example: $\frac{d}{dx}(5) = 0$

2. Power Rule:

Formula: $\frac{d}{dx}(x^n) = nx^n - 1$, where n is any real number

Explanation: To differentiate a variable raised to a power, bring the power down as a coefficient and then reduce the original power by one.

Examples:

1. $\frac{d}{dx}(x) = \frac{d}{dx}(x^1) = 1x^{1-1} = 1x^0 = 1$
2. $\frac{d}{dx}(x^3) = 3x^{3-1} = 3x^2$

3. $\frac{d}{dx}\sqrt{(x)} = \left(x^{\frac{1}{2}}\right) = \frac{1}{2}x^{1/2-1} = \frac{1}{2}x^{\frac{-1}{2}} = \frac{1}{\sqrt[2]{x}}$
4. $\frac{d}{dx}\left(\frac{1}{x^2}\right) = \frac{d}{dx}(x^{-2}) = -2x^{-2-1} = -2x^{-3} = \frac{-2}{x^3}$

Below is the list of the most common derivative rules:

	Function	Derivative
1	f(x)=c (constant)	$f'(x)=0$
2	$f(x)=x^n$	$f'(x)= nx^{n-1}$
3	f(x)=sin(x)	f′(x)=cos(x)
4	f(x)=cos(x)	f′(x)=−sin(x)
5	$f(x)=e^x$	$f'(x)=e^x$
6	f(x)=ln(x)	$f'(x)=\frac{1}{x}$

Table-5.2d

Examples:

1. What is the derivative or $\frac{d}{dx}$ of

   $f(x) = 3x^2 + 2x + 1$

   Using the power rule: $f'(x) = 6x + 2$

2. What is the $\frac{d}{dx}$ of $\sin(x^2)$?

   In this example, the derivative can be calculated using the chain rule.

   if $f(x) = sin(x^2)$
   Let the outer function: $sin(u)$
   Inner function: $u = x^2$

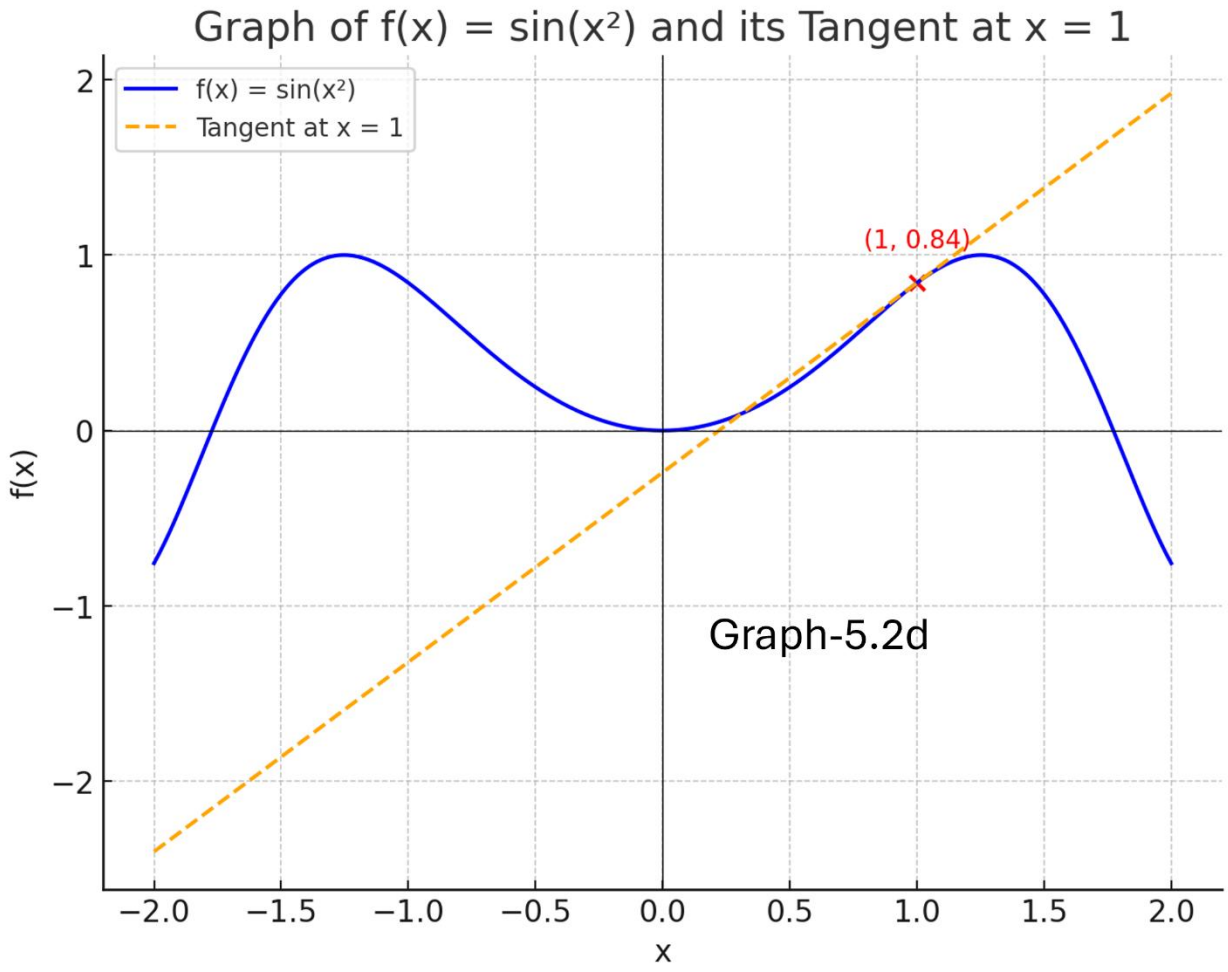
   Apply the chain rule:

$$\frac{d}{dx}\left(sin(x^2)\right) = cos(x^2)\frac{d}{dx}(x^2)$$
$$= cos(x^2) \cdot 2x$$

The derivative of $\sin(x^2) = 2x\cos(x^2)$

The Graph-5.2d provides a visual illustration of the tangent line represents the instantaneous rate of change (or derivative) of $f(x) = \sin(x^2)$ at $x = 1$:

- The blue curve is the function f(x)=sin(x^2), which oscillates faster as x moves away from 0.
- The orange dashed line is the tangent line at x=1, where the slope is computed using the chain rule:
  $f'(x) = 2xcos(x^2) \Rightarrow f'(1) = 2 \cdot 1 \cdot cos(1) \approx 1.08$
- The red point marks the position (1,sin(1)) ≈ $(1,0.84)$, where the tangent touches the curve.



5.3 Recognize how smoothness and curvature affect model training.

Training models (like neural networks) often involves minimizing a loss function using gradient-based methods (e.g., gradient descent). To do this efficiently, the first derivative *(gradient)* and second derivative (curvature) of the loss function are critical.

5.3.1 Smoothness in the Context of Derivatives and Optimization.

Smoothness, in the mathematical sense relevant to optimization, describes how "well-behaved" a function is. It's fundamentally tied to the existence and behavior of its derivatives.

A function is considered smooth if its first derivative exists and is continuous

Let’s break down what this implies:

First Derivative Exists: This means that at every point on the function, we can calculate a clear, unambiguous slope (gradient). There are no sharp corners, kinks, or vertical tangents where the slope is undefined.

First Derivative Changes Gradually (Continuity): This is the crucial part for optimization. If the first derivative (the gradient) is continuous, it means:

- The gradients do not make sudden, abrupt jumps. As you move infinitesimally along the function, the slope smoothly transitions from one value to the next.
- Consequently, the rate of change of the function is consistent and predictable within a local region.

Why is this important for optimization?

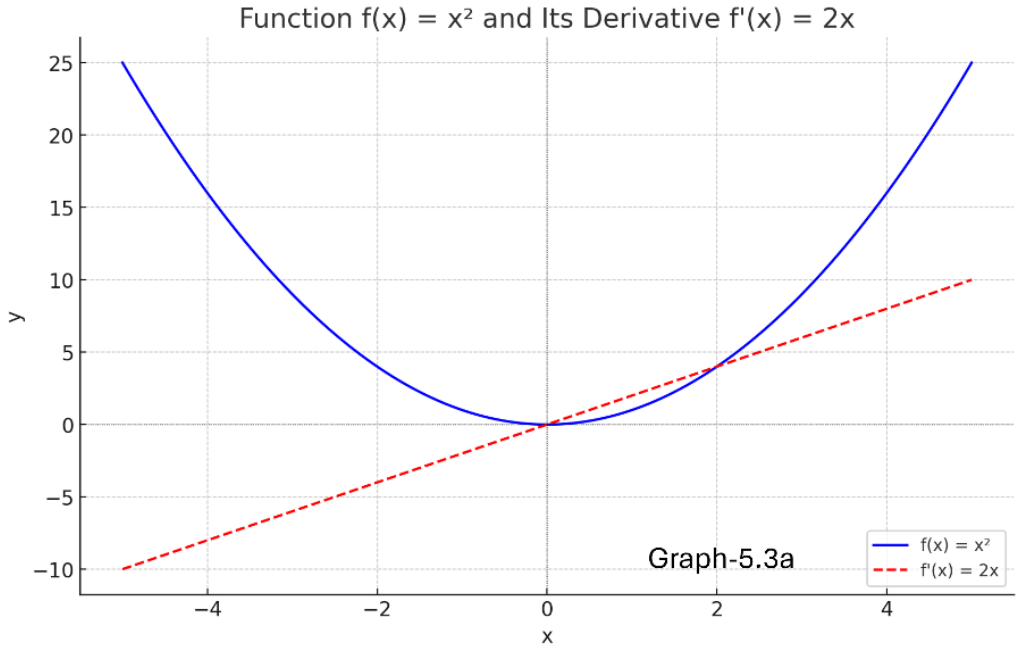
When the gradients change gradually:

- More Predictable Optimization: Algorithms like gradient descent rely on using the current gradient to predict the direction and magnitude of the next step. If gradients change wildly and discontinuously, these predictions become unreliable, making it difficult for the algorithm to efficiently converge to a minimum. Smoothness ensures that a small step in the direction of the negative gradient will indeed lead to a lower point in the function's landscape.

- Avoidance of "Stuck" Points: Non-smooth functions can have points where the derivative doesn't exist (like the tip of a V-shape). Optimization algorithms can get "stuck" at these points, unable to determine the correct direction to move. Smoothness helps avoid such problematic scenarios.

- Foundation for Higher-Order Methods: Many advanced optimization techniques (e.g., Newton's method) require the existence and continuity of second (or even higher) derivatives. A function must be smooth at the first-derivative level before these higher-order properties can even be considered.

Smooth function example: $f(x) = x^2, f'(x) = 2x$



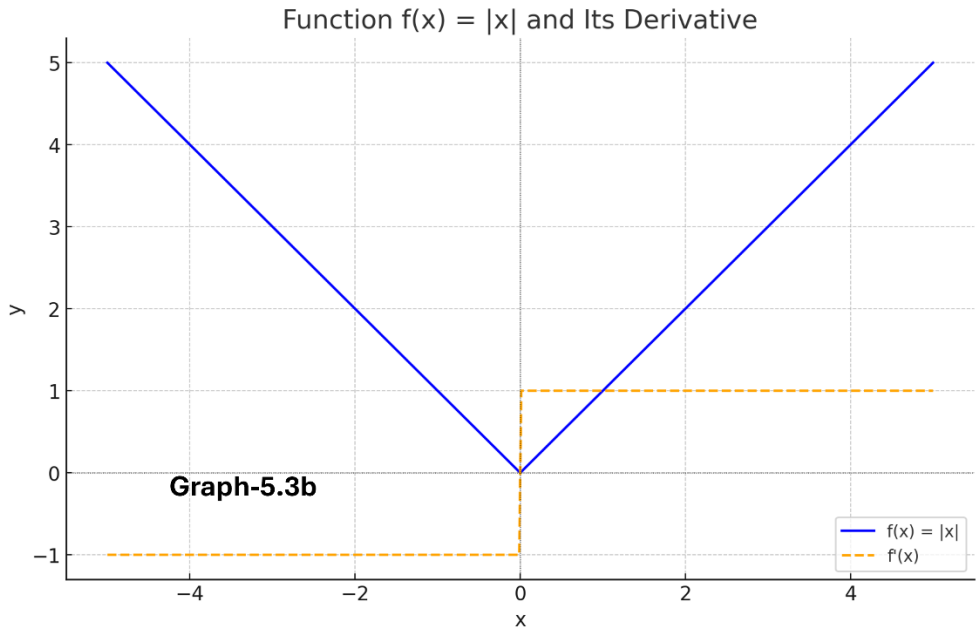
Graph-5.3a Interpretation:

- The **blue curve** represents $f(x)=x^2$, a parabola opening upward.
- The **red dashed curve** represents $f'(x)=2x$, which is a straight line passing through the origin.
- The slope of the tangent (the derivative) increases linearly with x, illustrating how the rate of change of x^2 grows as x moves away from 0.

Example of Non-smooth function or piecewise function:

$$f(x) = |X| = \begin{Bmatrix} x, & x > 0 \\ -x & x < 0 \end{Bmatrix}$$

$$f(x) = |x| \Rightarrow f'(x)\ undefined\ at\ x = 0$$



The derivative of the function f(x)=|x| is a piecewise function:

$$f'(x) = \begin{Bmatrix} -1 & if < 0 \\ 1 & if x > 0 \\ undefined & x = 0 \end{Bmatrix}$$

Graph-5.3b Interpretation:

- The **blue curve** represents f(x) =|x|, which forms a sharp **V** shape
- The **orange dashed line** shows the derivative:
  - Constant at -1 for x < 0
  - Constant at +1 for x > 0
  - Not defined at x =0, where there is a sharp corner.

Key Takeaway for f(x) =|x| :

- Because the left-hand and right-hand derivatives at x=0 are different (−1 and +1), the function is not differentiable at that point—this is a classic example of a non-smooth function.

- In essence, a smooth function provides a more well-behaved and navigable landscape for optimization algorithms, allowing them to descend towards minima more reliably and efficiently.

5.3.2 Understanding Curvature and Learning Rates in Optimization

The "curvature" in the context of optimization, particularly with algorithms like gradient descent, refers to how quickly the slope (gradient) of a function changes. Imagine a landscape where you're trying to find the lowest point.

We can use second Derivative to predict How Fast Slope Changes, meaning:

- The second derivative $f''(x)$ or Hessian matrix in higher dimensions) describes curvature—how fast the gradient is changing.

High Curvature (Sharp Valleys)
When you're in an area of high curvature, think of it like navigating a sharp valley or a steep hill.

- What it implies: The gradients (the direction and steepness of the slope) are changing very rapidly over a short distance. A small step in one direction can lead to a drastically different slope.
- Impact on learning rate: If the learning rate (the size of the step) is too large in such an area, it's highly likely to overshoot the minimum. Or it might bounce back and forth across the valley, or even diverge entirely, never settling into the lowest point.
- Requirement: To effectively find the minimum in high-curvature areas, we need a small learning rate. This allows for tiny, precise steps that help you carefully descend without overshooting.

Low Curvature (Flat Areas)

Conversely, when you're in an area of low curvature, imagine walking across a relatively flat plain or a gently sloping plateau.

- What it implies: The gradients are changing slowly, or they are very small in magnitude. The landscape isn't dramatically different from one point to the next.
- Impact on learning rate: If the learning rate is too small in these flat areas, the progress will be incredibly slow. It would be like taking tiny baby steps across a vast field; it would take forever to reach your destination. This leads to slow convergence.
- Requirement: To make efficient progress in low-curvature areas, you can afford a larger learning rate. This allows you to take bigger steps and cover more ground quickly, speeding up the convergence towards a minimum.

Examples:

1. Constant curvature $f(x) = x^2 \Rightarrow f'(x) = 2x$, second derivative $f''(x) = 2$
2. Curvature increases rapidly $f(x) = x^4 \Rightarrow f'(x) = 4x^3, f''(x) = 12x^2$

In essence, the ideal learning rate often depends on the local curvature of the optimization landscape. This is why advanced optimization algorithms often employ adaptive learning rates, adjusting the step size based on the characteristics of the gradients they encounter.

5.3.3 Definition of Second Derivative

The second derivative of a function measures the rate of change of the rate of change—in other words, how the first derivative itself changes with respect to the independent variable.

Let $f(x)$ be a function that is differentiable on an open interval. If the first derivative $f'(x)$ is also differentiable, then the second derivative of f, denoted by $f''(x)$ or $\frac{d^2f}{dx^2}$, is defined as:

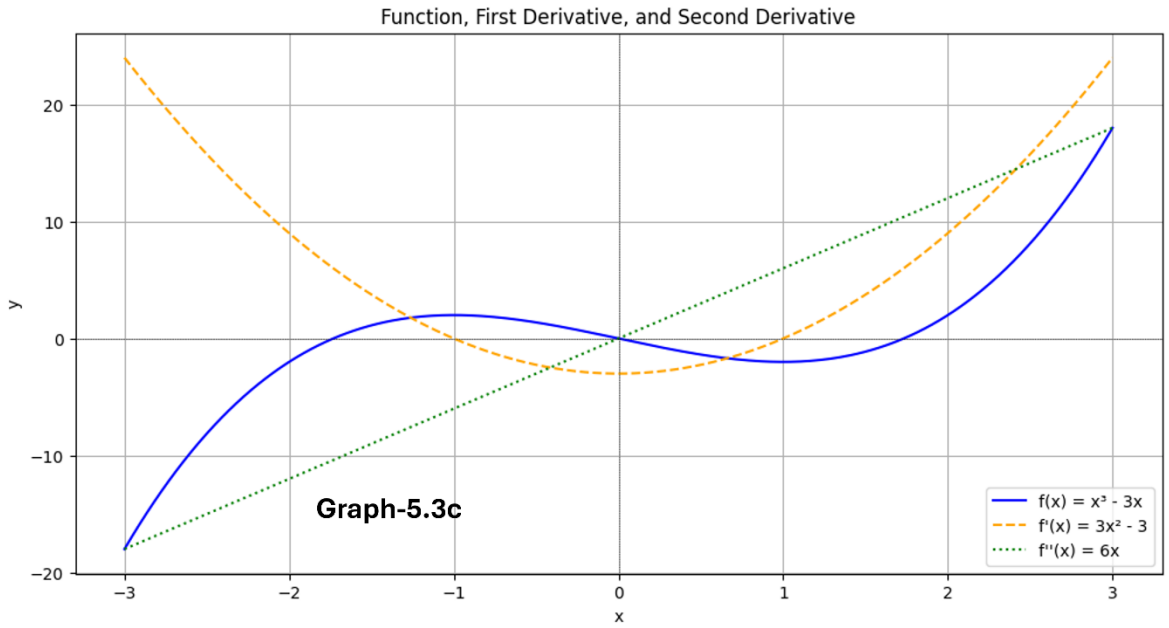
$$f''(x) = \frac{d}{dx}\left(\frac{df}{dx}\right) = \frac{d^2f}{dx^2}$$

It represents the derivative of the first derivative.

Geometric Meaning

- $f''(x) > 0$: the graph of f'' is **concave up** (like a cup); the slope is increasing.
- $f''(x) < 0$: the graph of f'' is **concave down** (like a frown); the slope is decreasing.
- $f''(x) = 0$: may indicate a **point of inflection**, i.e. x^3, where concavity changes.

The Graph-5.3c illustrates geometric concavity of the second derivative:



- The **blue curve** shows the original function $f(x)=x^3-3x$.
- The **orange dashed line** is the first derivative, showing how fast f(x) is changing (slope).
- The **green dotted line** is the second derivative f''(x)= 6x, which changes sign at x=0:
    - Positive → the function is concave up (bowl-shaped).
    - Negative → the function is concave down (hill-shaped).

5.3.4 Common Second Derivative Rules

Rule	Function f(x)	First Derivative f′(x)	Second Derivative f′′(x)
Power	x^n	nx^{n-1}	$n(n-1)x^{n-2}$
Constant	c	0	0
Sum	g(x)+h(x)	g′(x)+h′(x)	g′′(x)+h′′(x)
Difference	g(x)−h(x)	g′(x)−h′(x)	g′′(x)−h′′(x)
Product	u(x)·v(x)	u′v+ uv'	u′′v +2u′v′ + uv′′
Quotient	$\frac{u(x)}{v(x)}$	$\frac{u'v+uv'}{v^2}$	Use quotient rule on f′(x): more complex, but follows same rules
Chain	h(g(x))	h′(g(x))·g′(x)	h′′(g(x))·[g′(x)]²+h′(g(x))·g′′(x)

Examples:

1. Power rule: $f(x) = x^3 \Rightarrow f'(x) = 3x^2 \Rightarrow f''(x) = 6x$
2. Constant rule: $f(x) = 0 \Rightarrow f'(x) = 0, f''(x) = 0$
3. Sum Rule: $f(x) = x^3 + sin(x)$,

$$f'(x) = \frac{d}{dx}(x^3) + \frac{d}{dx}\big(sin(x)\big) = 3x^2 + cos(x)$$

$$f''(x) = \frac{d}{dx}(3x^2) + \frac{d}{dx}\big(cos(x)\big) = 6x - sin\,(x)$$

4. Chain Rule:

$$f(x) = sin(x^2), f'(x) = cos(x^2) \cdot 2x, f''(x) = -sin(x^2) \cdot (2x)^2 + cos(x^2) \cdot 2$$

5.4 Practical Applications for PCA Using Eigenvectors

- Image Compression by finding the most important patterns (eigenvectors) in the image data, then representing images using only these important patterns, thus reducing storage needs while preserving visual quality
- Noise Reduction, PCA can remove noise by eliminating components with small eigenvalues

5.5 Linear Algebra with Python

Python provides two powerful libraries for linear algebra: **NumPy** and **SymPy**. Each serves a distinct purpose:

- **NumPy** is optimized for high-performance **numerical** computations.
- **SymPy** is designed for **symbolic** mathematics, allowing algebraic manipulation of equations and expressions.

While NumPy is designed for numerical computations (not symbolic like SymPy), you can approximate derivatives using finite differences.

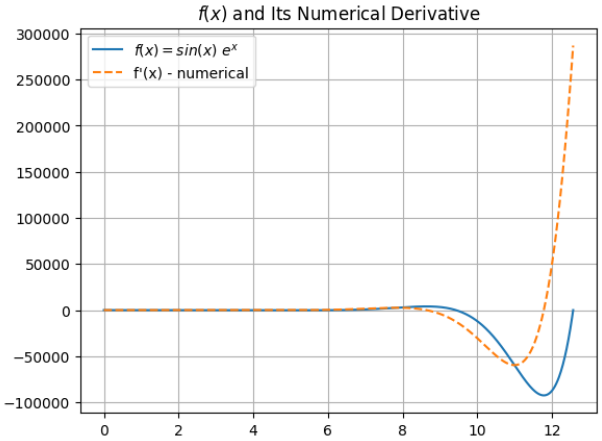
5.5a Numerical derivative

We'll use NumPy to approximate the derivative using the central difference method:
$f(x) = \sin(x) \cdot e^x$

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function f(x)
def f(x):
    return np.sin(x) * np.exp(x)

# Derivative approximation using central difference
def numerical_derivative(f, x, h=1e-5):
    return (f(x + h) - f(x - h)) / (2 * h)

# Generate x values
x_vals = np.linspace(0, 4*np.pi, 1000)

# Compute f(x) and f'(x)
y_vals = f(x_vals)
dy_dx = numerical_derivative(f, x_vals)

# Plot the function and its derivative
plt.plot(x_vals, y_vals, label='f(x) = sin(x) * exp(x)')
plt.plot(x_vals, dy_dx, label="f'(x) - numerical", linestyle='--')
plt.legend()
plt.title("$f(x)$ and Its Numerical Derivative")
plt.grid(True)
plt.show()
```

Notes:

- `h` is a small step size, determining



accuracy.

- This is a **numerical** approximation, not exact like SymPy.

5.5b Symbolic Derivative

Derivative **of f(x)=x²·sin(x)** using **SymPy**, Python's symbolic math library.

```python
import sympy as sp
import IPython.display as display

# Define the variable
x = sp.symbols('x')
```

The derivative of $f(x) = x^2 \sin(x)$ is:

$$2x \sin(x) + x^2 cos(x)$$

```python
# Define the function
f = x**2 * sp.sin(x)

# Take the derivative of f with respect to x
df_dx = sp.diff(f, x)

# Display the result
print("The derivative of f(x) = x^2 * sin(x) is:")
sp.pprint(df_dx)
display.display(df_dx)
```

```python
# This uses the product rule symbolically.
# To take higher-order derivatives (second):
second_derivative = sp.diff(f, x, 2)
sp.pprint(second_derivative)
display.display(second_derivative)
```

$$-x^2 sin(x) + 4xcos(x) + 2sin\,(x)$$

Module 5: Learning Materials & References

a) Introduction to Derivatives
b) What is a Derivative? Deriving the Power Rule
c) First Derivatives Intro
d) Second Derivatives
e) SymPy Intro to Calculus
f) Mastering Calculus for Machine Learning: Key Concepts and Applications

Assessment: Calculus Basics for Machine Learning

Part A: Quiz

Multiple choice

5.1a What does the first derivative of a function $f(x)$ represent?

A. The total area under the curve
B. The slope of the tangent line at a point
C. The height of the curve
D. The curvature of the graph

5.2a If $f'(x)>0$ on an interval, what can you say about $f(x)$on that interval?

A. It is constant
B. It is decreasing

C. It is increasing
D. It is concave down

5.3a Which of the following is not necessarily true if f'(x)=0 at x=a?

A. The function has a local maximum or minimum at x=a
B. The function has a horizontal tangent at x=a
C. The slope of the function is zero at x=ax = ax=a
D. The function is flat at that point

5.4a Which function has a first derivative that is not defined at x=0?

A. $f(x)=x^2$
B. $f(x)=\sin(x)$
C. $f(x)=e^x$
D. $f(x)=|x|$

5.5a If the graph of $f(x)$ is decreasing and concave up, what can we say about the signs of $f'(x)$ and $f''(x)f''(x)$?

A. $f'(x)>0, f''(x)>0$
B. $f'(x)<0, f''(x)<0$
C. $f'(x)<0, f''(x)>0$
D. $f'(x)>0, f''(x)<0$

5.6a Why is smoothness important in training machine learning models using gradient-based optimization?

A. It helps provide stable and predictable gradient updates
B. It makes the model memorize the training data exactly
C. It ensures that the loss function has discrete values
D. It removes the need for an optimizer

5.7a What does high curvature in a loss surface typically lead to during training?

A. Faster training with large steps
B. Sharp minima that are easy to find
C. Gradient values always becoming zero
D. Oscillations or unstable updates in gradient descent

5.8a In the context of optimization, what is a challenge caused by low curvature (very flat regions of the loss function)?

A. The optimizer skips over minima
B. The gradients vanish, slowing down convergence
C. The gradients explode, making training unstable
D. The model diverges instantly

Part B:Find Derivatives

5.1b What is $\frac{d}{dx}(\frac{3x^3-x^2}{x})$?

A. $9x^2$-2x, B. 6x -1, C. 6x, D. $9x^2$-2

5.2b What is $\frac{d}{dx}(3x^2)$?

A. 2x, B. 3x, C. 6x, D. $6x^2$

5.3b What is $\frac{d}{dx}(x^2 ln(x))$?

A. 2x ln(x), B. x(1+ln(x)), C. x + 2x ln(x), D. x^2+2xln(x)

5.4b what is $\frac{d}{dx}(\frac{cos\,(x)}{x})$?

A. $\frac{xsin(x)-cos\,(x)}{x^2}$, B. $\frac{-xsin(x)-cos\,(x)}{x^2}$,C. $\frac{xsin(x)+cos\,(x)}{x^2}$,D. $sin\,(x)$

Part C: Python Exercises

5.1c Smoothness Check via Derivative Continuity

Write a Python script that uses symbolic math to compute the derivative of a function and check if it is smooth at a given point (i.e., the left and right derivatives match). Use SymPy.

```
from sympy import symbols,
diff, Abs, limit

x = symbols('x')
f = Abs(x)  # f(x) = |x|

# Task: Compute the left-
hand and right-hand
derivatives at x = 0
# and check if f(x) is
smooth at x = 0.
```

Sample output:
Left-hand derivative at x = 0:
-1
Right-hand derivative at x =
0: 1
Is f(x) smooth at x = 0? False

5.2c Symbolic Second Derivative

Write a Python script that uses symbolic math to compute the first and second derivatives of $f(x) = x^3 + sin\,(x)$

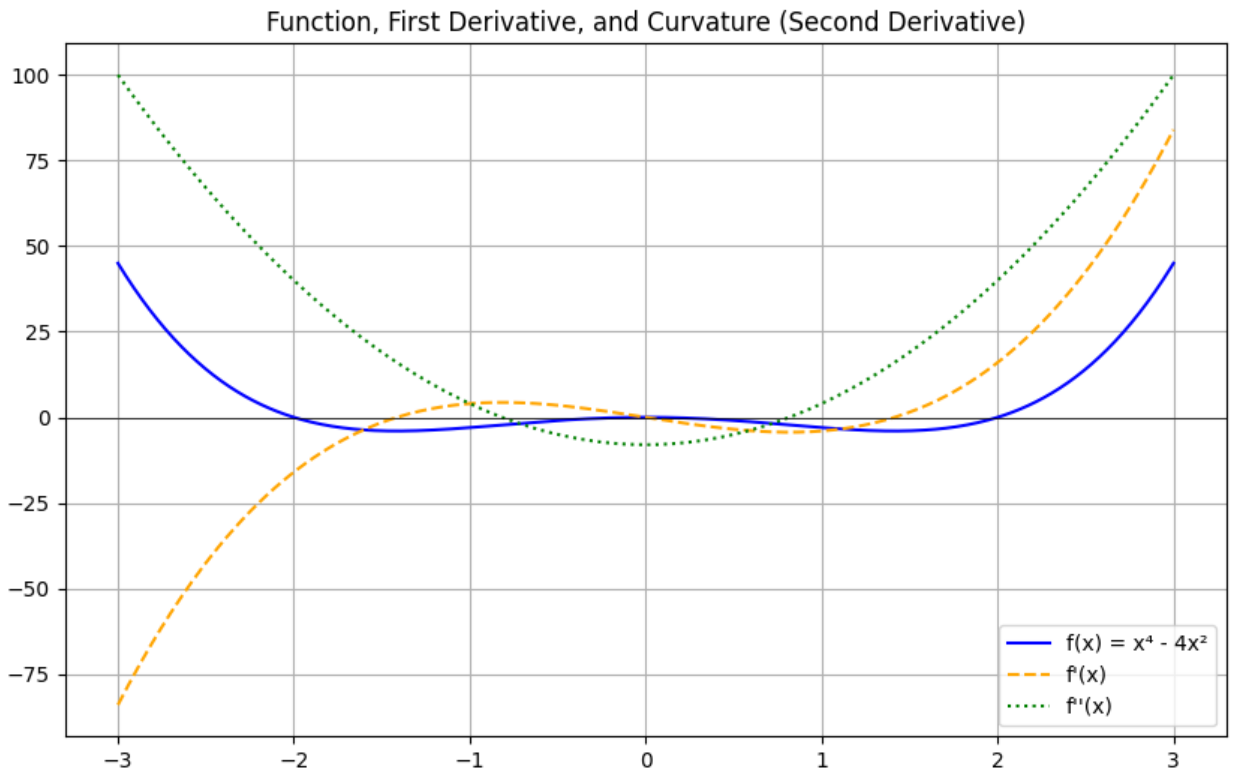
```
import sympy as sp

# Define the variable and function
x = sp.symbols('x')
f = x**3 + sp.sin(x)

# Task:
# 1. Find the first derivative f'(x)
# 2. Find the second derivative f''(x)
# 3. Simplify and print both derivatives
```

Sample output:
First derivative f'(x): $3x^2$ + cos(x)
Second derivative f''(x): 6x - sin(x)

5.2c Visualizing Curvature with Second Derivative

Write a Python script to plot the function $f(x) = x^4 - 4x^2$ and its first and second derivatives to observe curvature and provide explanations of the graph.

Sample output:



5.3c Symbolic Derivative Test for Smoothness

Write a Python script that uses symbolic math to show that the derivative of a non-smooth function like $f(x)=|x|$ is not defined at $x=0$, while a smooth function like $f(x)=x^2$ has a derivative defined everywhere.

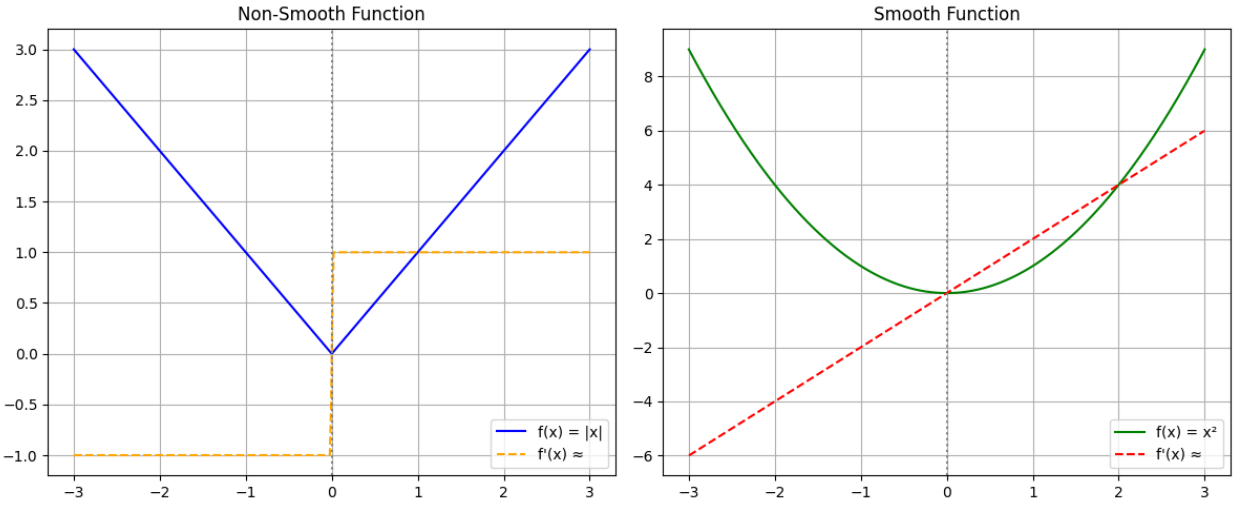
Sample output:

```
Left-hand derivative of f(x) = |x| at x = 0: -1
Right-hand derivative of f(x) = |x| at x: 1
Is the derivative of f(x) = |x| defined at x = 0? False

Derivative of f(x) x**x smooth at x = 0: 0
Is the derivative of f(x) = x**x smooth defined at x = 0? True
Is f(x) = x**x smooth at x = 0? True
```

5.4c Visualizing Derivatives Behavior

write a Python script to plot two functions one smooth like $f(x)=x^2$, and one non-smooth like $f(x)=|x|$ and their derivatives to visually identify points of non-differentiability, and provide interpretations of the graphs

Sample output:



Module 6: Multivariable Calculus & Gradients for Cost Minimization

Multivariable calculus is essential for machine learning as it extends single-variable calculus concepts to functions with multiple variables, enabling the analysis and optimization of complex models. This mathematical framework is crucial for understanding how different input variables interact and affect outcomes in machine learning algorithms.

The transition from single-variable to multivariable calculus introduces a key concept that are indispensable for machine learning, Partial Derivatives. They allow us to examine how a function changes with respect to each input variable, holding all others constant. This is crucial when dealing with models that have hundreds, thousands, or even millions of parameters—such as neural networks—where understanding the influence of each parameter is necessary for effective learning

6.1 Understanding Partial Derivatives

Partial derivatives measure how a function changes as one variable is changed, while keeping all other variables constant. They are essential for understanding how functions with multiple inputs behave, allowing us to analyze the impact of each input variable on the function's output.

By computing partial derivatives, one can:

- **Identify Rates of Change:** Each partial derivative represents the slope of the function in the direction of one variable, providing insight into how sensitive the function is to changes in that specific input
- **Understand Function Behavior:** Partial derivatives help reveal local maxima, minima, and saddle points, which are essential for optimization problems—such as minimizing error in machine learning models
- **Build the Gradient Vector:** The collection of all first-order partial derivatives forms the gradient, which points in the direction of the steepest ascent of the function and is fundamental for optimization algorithms like gradient descent
- **Analyze Multidimensional Surfaces:** By examining partial derivatives, one can visualize how a function behaves in different directions, aiding in the understanding of complex surfaces and their geometry

In summary, partial derivatives are indispensable for dissecting and interpreting the behavior of multivariable functions, enabling both theoretical analysis and practical applications across numerous fields

6.1.a Definition of Partial Derivatives

For a function $f(x,y)$ the partial derivatives are represented as $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$. These symbols indicate the rate of change of f with respect to x and y, respectively, while treating the other variable as a constant.

- The general formula for the partial derivative of a function f with respect to one of its variables (say, x_i) while keeping all other variables constant, is given by the following limit definition:

$$\frac{\partial f}{\partial x_i}(a) = \lim_{h \to 0} \frac{f(a_1, \ldots, a_{i-1}, a_i + h, a_{i+1}, \ldots, a_n) - f(a_1, \ldots, a_n)}{h}$$

  Here $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ is a specific point in the domain of f, and x_i is the variable with respect to which you are taking the derivatives.

- For a function of two variables, $z = f(x,y)$, the partial derivatives are:

$$\frac{\partial f}{\partial x}(x,y) = \lim_{n \to 0} \frac{f(x+h, y) - f(x,y)}{h}$$

$$\frac{\partial f}{\partial x}(x,y) = \lim_{n \to 0} \frac{f(x, y+h) - f(x,y)}{h}$$

These formulas generalize to functions with any number of variables. In practice, to compute partial derivatives, you treat all variables except the one you are differentiating with respect to as constants and use the usual rules of differentiation.

Example 6.1a:

Consider the function $f(x,y) = x^2y + sinx + cosx$. Let's find the partial derivatives with respect to x and y:

1. Partial derivatives with respect to x $\left(\frac{\partial f}{\partial x}\right)$:

   Treat y as a constant and differentiate:

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}(x^2y) + \frac{\partial}{\partial x}(sinx) + \frac{\partial}{\partial x}(cosy)$$

$FirstTerm: \frac{\partial}{\partial x}(x^2y) = 2xy(since\ y\ is\ constant)$

$SecondTerm: \frac{\partial}{\partial x}(sinx) = cosx$

$ThirdTerm = \frac{\partial}{\partial x}(cosy) = 0\ since\ cosy\ isconstant\ with\ repsect\ x.$

So, $\frac{\partial f}{\partial x} = 2xy + cosx$

2. Partial derivative with respect to y ($\frac{\partial f}{\partial y}$)

Treat x as a constant and differentiate:

$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}(x^2y) + \frac{\partial}{\partial y}(sinx) + \frac{\partial}{\partial y}(cosy)$

$FirstTerm: \frac{\partial}{\partial y}(x^2y) = x^2(sincexisconstant)$

$SecondTerm: \frac{\partial}{\partial y}(sinx) = 0sincexisconstantwithrepsecty$

$ThirdTerm = \frac{\partial}{\partial y}(cosy) = -siny$

So, $\frac{\partial f}{\partial y} = x^2 - siny$

Summary Table

Partial Derivative	Expression
$\frac{\partial f}{\partial x}$	2xy+cos x
$\frac{\partial f}{\partial x}$	$x^2-\sin y$

This is a standard approach for calculating partial derivatives of multivariable functions.

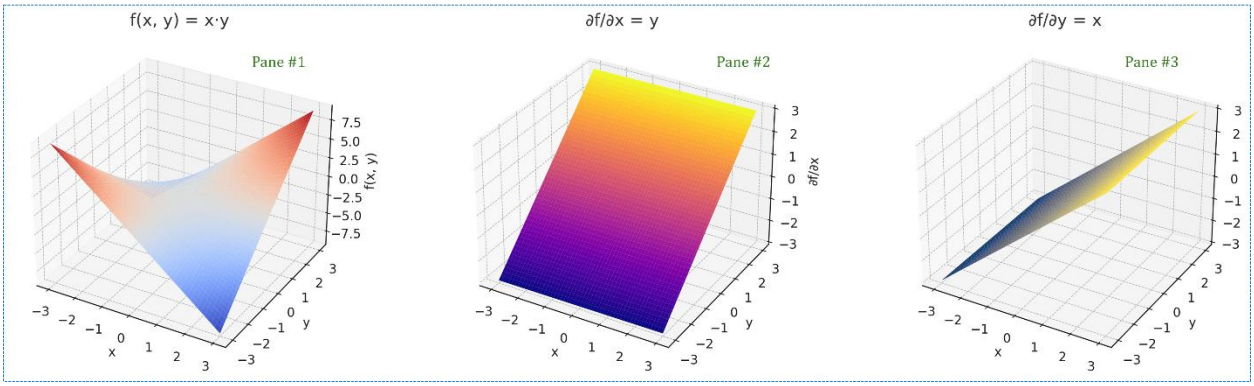
Examples:

1/ Simple function $f(x, y) = x \cdot y$

Partial Derivatives:

- $\frac{\partial f}{\partial x} = y$: the slope in the x-direction is constant for each value of y
- $\frac{\partial f}{\partial y} = x$: the slope in the y-direction is constant for each value of x

Graph-6.1



Graph-6.1 provides a visual illustration of multivariable function, from left to right:

Pane #1, the surface of f (x, y): a saddle-shape plane where values increase in the first and third quadrants and decrease in the second and fourth quadrants.

Pane #2 $\frac{\partial f}{\partial x} = y$: the slope in the x-direction is constant along each horizontal row (depends only on y)

Pane #3 $\frac{\partial f}{\partial y} = x$: the slope in the y-direction is constant along each vertical column (depends only on x)

2/ More complex function $f(x, y) = x^2y + sinx + cosx$.

Partial Derivatives:

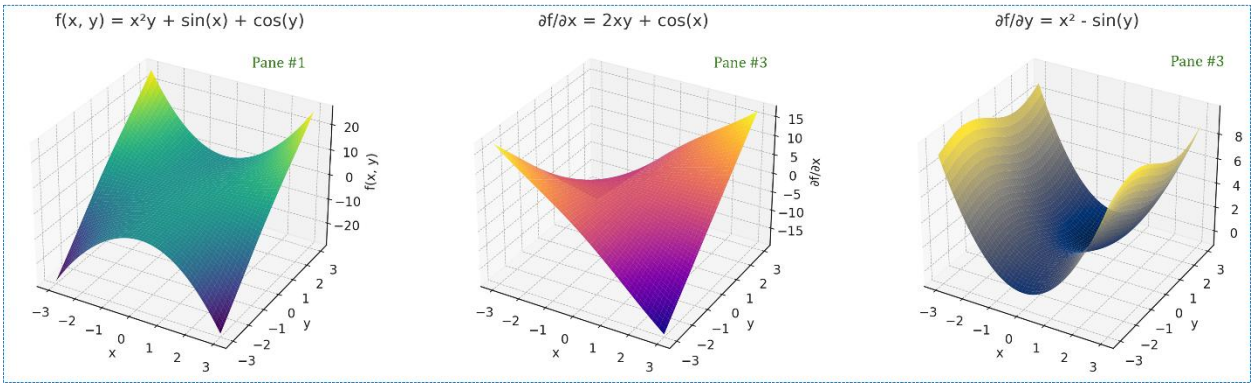
Using information from example 6.1a.

$$\frac{\partial f}{\partial x} = 2xy + cosx$$

$$\frac{\partial f}{\partial y} = x^2 - siny$$

Graph-6.1b provides a visual illustration of multivariable functions, from left to right:

Graph-6.1b



Pane #1 (Surface) , the surface of f (x, y): a blend of polynomial and trigonometric behavior.

Pane #2 (Plasma) $\frac{\partial f}{\partial x}$ shows how slope in the x-direction varies with y.

Pane #3 (Cividis) $\frac{\partial f}{\partial y}$ shows a saddle with minimax point and how slope in the y-direction depends mostly on x^2 , and modulated by sin(y)

6.2 Meaning of the Gradient Vector in the *Steepest* Ascent Direction

The gradient—a vector of all partial derivatives—points in the direction of the steepest ascent of a function and is the linchpin of optimization algorithms like gradient descent. Through gradient descent, we iteratively adjust model parameters to minimize a loss function, thereby refining the model's predictions.

The gradient vector is a vector of partial derivatives that points in the direction of the steepest ascent of a function.

For a function f(x,y), the gradient is given by

$$\nabla f(x,y) = \frac{\partial f}{\partial x}\hat{\imath} + \frac{\partial f}{\partial y}\hat{\jmath}$$

For a function f(x,y,z), the gradient is given by

$$\nabla f(x,y,z) = \frac{\partial f}{\partial x}\hat{\imath} + \frac{\partial f}{\partial y}\hat{\jmath} + \frac{\partial f}{\partial z}\hat{k}$$

The gradient vector (like a slope in higher dimension):

Points in the direction of the steepest ascent of the function. Its magnitude represents how quickly the function increases in that direction.

In machine learning, we usually want to minimize a loss function $\mathrm{L}(\theta)$, where θ represents model parameters (e.g., weights in neural networks).

To do this, we use **gradient descent**:

$$\theta_{new} = \theta old - \eta \nabla L(\theta)$$

- Where $\eta =$ is the learning rate (a small positive constant)
- $\nabla L(\theta)$ tells us how to adjust the parameters to reduce the loss
- We subtract the gradient because we want to **descend** (move in the direction of steepest decrease).

Example:
What is gradient vector (∇)of $f(x, y) = x^2 y + sinx + cosx$.

Using information from example 6.1a.

$$\frac{\partial f}{\partial x} = 2xy + cosx$$

$$\frac{\partial f}{\partial y} = x^2 - siny$$

$$\nabla f = 3xy + cosx, x^2 - siny$$

6.2.a Key Properties of the Gradient

Property	Meaning
Direction	Points toward fastest **increase** of the function
Negative Gradient	Points toward fastest **decrease**—used in **gradient descent**
Zero Gradient	Indicates a **stationary point** (could be a minimum, maximum, or saddle)
High Magnitude	Function value is changing rapidly
Low Magnitude (near 0)	Function is flat or nearly optimal

6.3 Apply Cost Function Minimization in ML

A cost function (also called a loss function) quantifies the error between a machine learning model's prediction and the actual values in the training data. The main goal in training a machine learning model is to minimize this cost function, which means reducing the difference between predicted and actual outputs as much as possible

6.3.a How Minimization Works

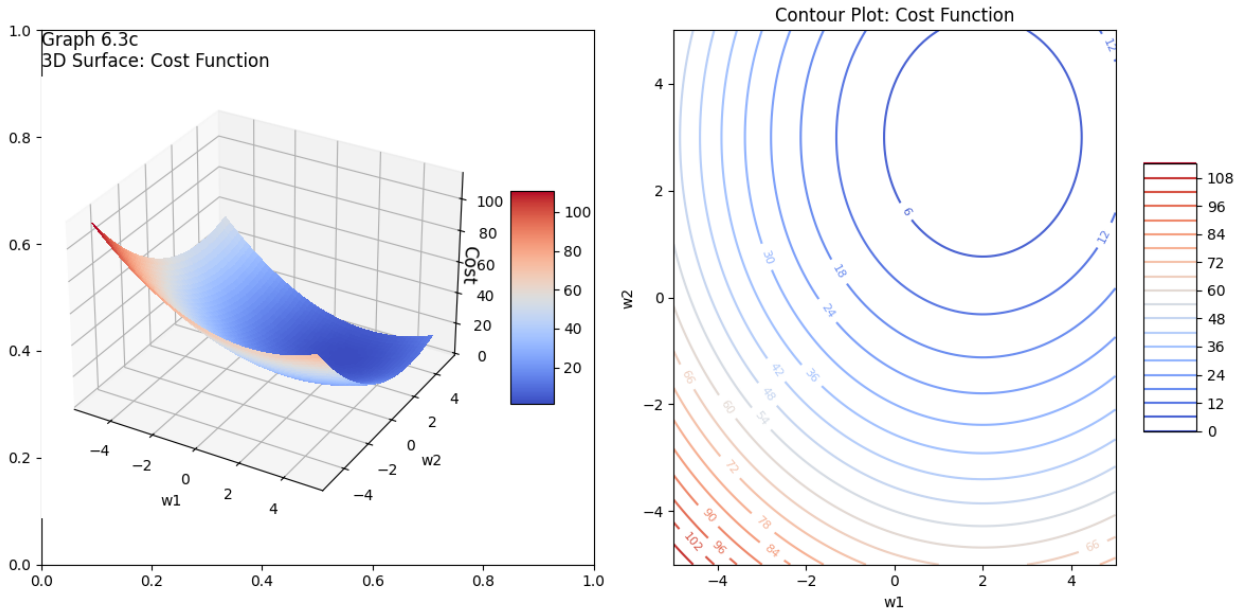
1. **Model Prediction:**
   The model uses its current parameters (such as weights in a neural network) to make predictions on the training data.
2. **Error Calculation:**
   The cost function computes how far these predictions are from the true values. Common cost functions include Mean Squared Error (MSE) for regression and cross-entropy for classification
3. **Parameter Update:**
   The model's parameters are adjusted to reduce the value of the cost function. This is typically done using optimization algorithms like gradient descent
4. **Iterative Process:**
   The process repeats: new predictions are made, the cost is recalculated, and parameters are updated again. This continues until the cost function reaches a minimum or stops improving significantly

6.3.b Role of Partial Derivatives and Gradients

- **Partial derivatives** (from multivariable calculus) are used to determine how the cost function changes with respect to each parameter
- **Gradient descent** uses the gradient (the vector of all partial derivatives) to identify the direction of steepest descent, guiding the parameter updates toward lower cost

6.3.c Visualization and Intuition

- **3D Surface:**
  The cost function can be visualized as a surface in high-dimensional space, where each point represents a set of parameter values and the height represents the cost
- **Contour Plots:**
  In two dimensions, contour plots show lines of equal cost, helping to visualize the path toward the minimum



Graph 6.3c Explanations:

- **3D Surface:** Shows how the cost changes as the parameters (w_1,w_2) vary, with the height representing the cost value
- **Contour Plot:** Displays lines of equal cost, making it easier to see the path toward the minimum cost and the shape of the cost landscape
- **The cost function:** In simpler terms, imagine a bakery, the cost function would tell what the cheapest way to bake a certain number of loaves of bread, given the price of ingredients (w_1) and the cost of labor (w_2). The cost function **w** in this example is $(w_1 - 2)^2 + (w_2 - 3)^2 + 1$, a simple linear regression with two parameters **w_1** & **w_2** , where **w_1 and w_2** represent the prices of different inputs (e.g., labor, capital, materials). To graph the cost function, we convert them into a mesh-grid.

6.3.d Summary of cost minimization

Step	Description
Model Prediction	Use current parameters to predict outputs
Error Calculation	Compute cost (error) between predictions and true values
Parameter Update	Adjust parameters to reduce cost (e.g., via gradient descent)
Iteration	Repeat until cost is minimized or stops improving

6.4 Calculus with Python

Python provides two powerful libraries for Calculus: **NumPy** and **SymPy**. Each serves a distinct purpose:

- **NumPy** is optimized for high-performance **numerical** computations.
- **SymPy** is designed for **symbolic** mathematics, allowing algebraic manipulation of equations and expressions.

6.4a Partial Derivative with NumPy

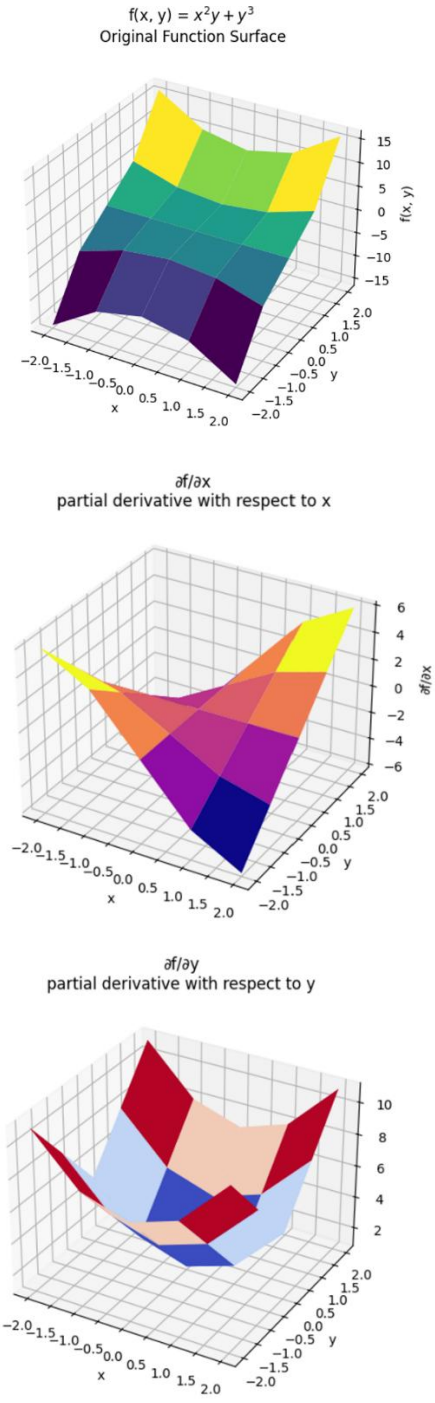
Numpy does **not** perform symbolic differentiation like SymPy, but it can approximate **partial derivatives** numerically using `numpy.gradient`. Below is an example of how to compute partial derivative of $f(x, y) = x^2y + y^3$ using NumPy:

```python
import numpy as np

def f(x, y):
    return x**2 * y + y**3
#Create a grid of xx and yy values:
x = np.linspace(-2, 2, 5)  # example points
y = np.linspace(-2, 2, 5)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
#Compute Partial Derivatives
#Use np.gradient to compute the numerical partial derivatives:
# Z is a 2D array: rows vary y, columns vary x
df_dy, df_dx = np.gradient(Z, y, x)
# Display the results
print("Numerical partial derivative with respect to y:")
print(df_dy)
print("Numerical partial derivative with respect to x:")
print(df_dx)
```

```
Output:

Numerical partial derivative with respect to y:
[[11.  8.  7.  8. 11.]
 [ 8.  5.  4.  5.  8.]
 [ 5.  2.  1.  2.  5.]
 [ 8.  5.  4.  5.  8.]
 [11.  8.  7.  8. 11.]]
Numerical partial derivative with respect to x:
[[ 6.  4.  0. -4. -6.]
 [ 3.  2.  0. -2. -3.]
 [ 0.  0.  0.  0.  0.]
 [-3. -2.  0.  2.  3.]
 [-6. -4.  0.  4.  6.]]
```

Visualizing the surfaces before and after the partial derivative operations:

```
#plot the graph
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(18, 5))

# Plot the original function
ax1 = fig.add_subplot(1, 3, 1,
projection='3d')
ax1.plot_surface(X, Y, Z, cmap='viridis')
ax1.set_title('f(x, y) = $x^2 y + y^3$\n
Original Function Surface')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('f(x, y)')

# Plot partial derivative w.r.t x
ax2 = fig.add_subplot(1, 3, 2,
projection='3d')
ax2.plot_surface(X, Y, df_dx, cmap='plasma')
ax2.set_title('∂f/∂x\npartial derivative with
respect to x')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('∂f/∂x')

# Plot partial derivative w.r.t y
ax3 = fig.add_subplot(1, 3, 3,
projection='3d')
ax3.plot_surface(X, Y, df_dy,
cmap='coolwarm')
ax3.set_title('∂f/∂y\npartial derivative with
respect to y')
ax3.set_xlabel('x')
ax3.set_ylabel('y')
ax3.set_zlabel('∂f/∂y')
plt.tight_layout()
plt.show()
```



6.4a Partial Derivative with SymPy

Below is an example of how to compute partial derivative of $f(x, y) = x^2y + y^3$using Sympy

```python
#1. Import SymPy and Define Symbols
import sympy as sp
from IPython.display import display

x, y = sp.symbols('x y')
# 2. define the function
f = x**2*y + y**3
# 3. Take the partial   derivative of f
with respect to x
partial_x = sp.diff(f, x)
# 4. take the partial derivative of f
with respect to y
partial_y = sp.diff(f, y)
# Display the results
print("Partial derivative of f with
respect to x:")
sp.pprint(partial_x)
display(partial_x)
print("Partial derivative of f with
respect to y:")
sp.pprint(partial_y)
display(partial_y)
```

```
Partial derivative of f with respect to x: 2·x·y

Partial derivative of f with respect to y:
   2      2
  x  + 3⋅y
```

```
Evaluate at a Specific Point
Evaluate the partial derivatives at (x=1, y=2):
partial_x_at_point = partial_x.subs({x: 1, y: 2})
partial_y_at_point = partial_y.subs({x: 1, y: 2})
# Results: (4, 7)

Higher-Order Partial Derivatives:
second_partial_xx = sp.diff(f, x, 2)  # ∂²f/∂x²
second_partial_yy = sp.diff(f, y, 2)  # ∂²f/∂y²
```

Module 6: Learning Materials & References

a) Partial Derivatives and the Gradient of a Function Video
b) Gradients and Partial Derivatives Video
c) Multivariable Calculus for Machine Learning
d) Partial derivative Wikipedia
e) Mathematics for Machine Learning and Data Science Specialization - GitHub

Assessment: Calculus Basics for Machine Learning

Part A: Quiz

Multiple choice

6.1a What does the partial derivative $\frac{\partial f}{\partial x}$ of a function $f(x,y)$ represent?

A) The rate of change of f as both x and y change
B) The rate of change of f as x changes, keeping y constant
C) The rate of change of f as y changes, keeping x constant
D) The total change in f for any change in x and y

6.2a Which of the following best describes the gradient vector of a function $f(x,y)$?

A) A vector pointing in the direction of the fastest increase of f
B) A scalar indicating the maximum value of f
C) A vector pointing in the direction of the fastest decrease of f
D) A matrix of second derivatives

6.3a For the function $f(x,y) = x^{2y} + sinx + cosy, what\ is \frac{\partial f}{\partial y}?$

A) $2xy$
B) $x^2 - \sin x$
C) $x^2 + \sin y$
D) $2xy + \cos x$

6.4a If the gradient of a function $f(x,y)$ at a point is the zero vector, what does this indicate about the function at that point?

A) The function is increasing fastest
B The function is decreasing fastest
C) The function has a maximum point.
D) The function has a stationary (critical) point.

6.5a Which of the following statements is TRUE about partial derivatives?

A) They are used to construct the gradient vector
B) They measure the rate of change of a function as all variables change simultaneously
B) They are always equal to the total derivative
D) They can only be computed for functions of one variable

6.6a Which NumPy function is commonly used to numerically approximate the gradient of a function sampled as an array?

A) np.diff()
B) np.gradient()
C) np.derivative()
D) np.slope()

6.7a When using np.gradient() on a 2D array representing a surface, what does the output consist of?

A) A single array with the same shape as the input
B) A scalar value for the overall slope
C) A list of all derivatives
D) Two arrays, each representing the gradient along the x and y axes

6.8a To visualize the gradient vector field of a function f(x,y)f(x,y) using Python, which of the following is typically used in conjunction with np.gradient()?

A) `matplotlib.pyplot.scatter()`
B) `matplotlib.pyplot.plot()`
C) `matplotlib.pyplot.quiver()`
D) `matplotlib.pyplot.imshow()`

6.9a In the following code using SymPy, how do you compute the partial derivative of the function $f(x,y)=x^4+xy^4$ with respect to y in Python?

```
import sympy as sp
x, y = sp.symbols('x y')
f = x**4 + x * y**4
# What goes here?
```

A. f.diff(y)
B. f.diff(x,y)
C. sp.diff(f,x)
D. sp.diff(f,x,y)

6.10a How would you numerically compute the partial derivative of a function $f(x,y)$ with respect to x at a specific point using NumPy?

A) Use np.derivative(f, x)
B) Use np.gradient(f(x, y), x)

C) Use np.diff(f(x, y), x)
D) Use np.gradient(f(x, y), x, y) and extract the first array

Part B: Short Answers:

6.1b What are the advantages of using SymPy for symbolic partial derivatives over numerical methods?

6.2b When should you use function.diff (SymPy), and when should you use np.gradient() (NumPy)? Why and why not?

Part C: Python Exercises

6.1c Using SymPy to symbolically find partial derivatives

Write a simple Python script to symbolically find partial derivative of the given the function $x^2 sin\ (x)$

6.2c SymPy to symbolically find second partial derivatives

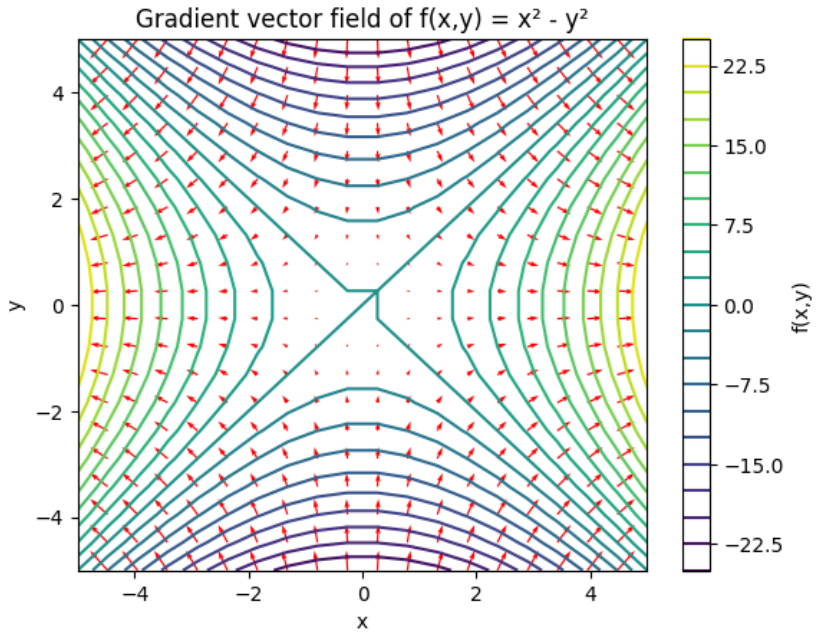
Write a simple Python script to symbolically find second partial derivative of problem 6.1c

6.3c Python Code to compute, and plot, gradient vector

Given the function $f(x,y)=x^2-y^2$, write Python script to:

a) Compute the gradient vector (i.e., the partial derivatives with respect to x and y) at a grid of points.
b) Plot the scalar field using a contour plot.
c) Superimpose the gradient vector field on the contour plot using arrows.

Sample output:



Module 7: Optimization & Gradient Descent

In machine learning and deep learning, building a model is only half the journey—making it learn effectively is the other half. Optimization is the process of adjusting a model's parameters to minimize errors and improve performance. At the heart of most optimization techniques lies gradient descent, a powerful iterative method that uses calculus to guide models toward better solutions. In this module, we will explore the intuition behind gradient descent, variations like stochastic and mini-batch approaches, and the role of learning rates, momentum, and adaptive optimizers. By the end, you'll not only understand how optimization works but also how to fine-tune it for faster and more accurate learning

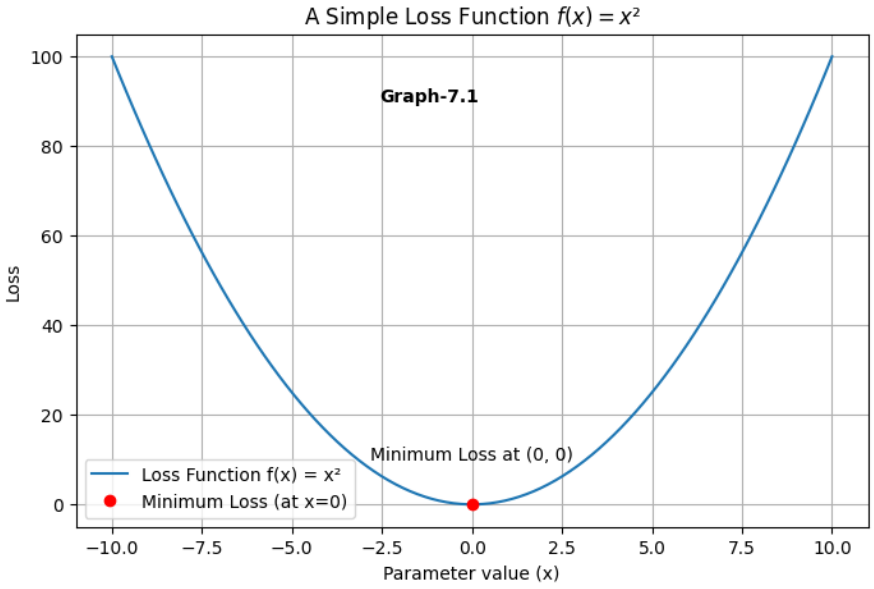
7.1 The Loss Function: Your Model's Report Card

A loss function, also called an objective or cost function, measures how bad your model's predictions are compared to the actual correct answers. The goal of training is to make this function's output as low as possible.

Think of it like a grade on a test. If the model's prediction is far from the correct answer, the loss is high (a bad grade). If it's close, the loss is low (a good grade). By trying to minimize this loss, the model learns to make better predictions.

A common and simple loss function is the Mean Squared Error (MSE), often used in regression problems. It's simply the average of the squared differences between the predicted values and the actual values.

Graph-7.1 Illustrates a simple quadratic loss function, where the goal is to find the value of x that makes loss equal to zero.



7.2 Gradient Descent: Rolling Down the Hill

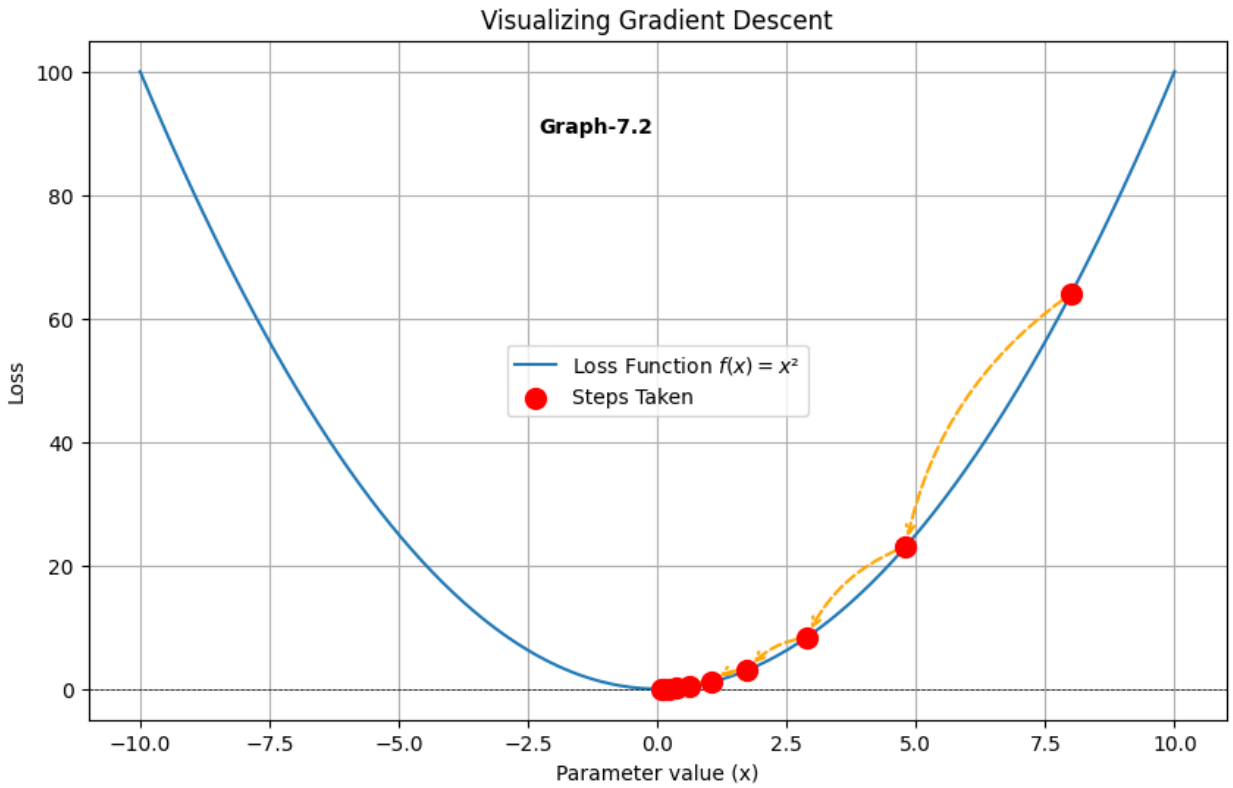
Gradient descent is the algorithm used to find the minimum of the loss function. Imagine you're standing on the side of a hill in thick fog and you want to get to the very bottom. What would you do? You'd look at the ground beneath your feet and take a step in the steepest downward direction.

That's exactly what gradient descent does:

1. **It starts** at a random point on the loss function "hill."
2. **It calculates the gradient**—the direction of steepest ascent (uphill).
3. **It takes a small step** in the *opposite* direction of the gradient (downhill).
4. **It repeats** this process until it reaches the bottom of the hill, where the slope is zero.

The size of each step is controlled by a parameter called the iterative **learning rate**. A small learning rate means slow but steady progress. A large learning rate can be faster but risks overshooting the minimum entirely.

Graph-7.2 illustrates the algorithm that takes progressively smaller steps as it gets closer to the minimum, where the slope (gradient) is less steep.

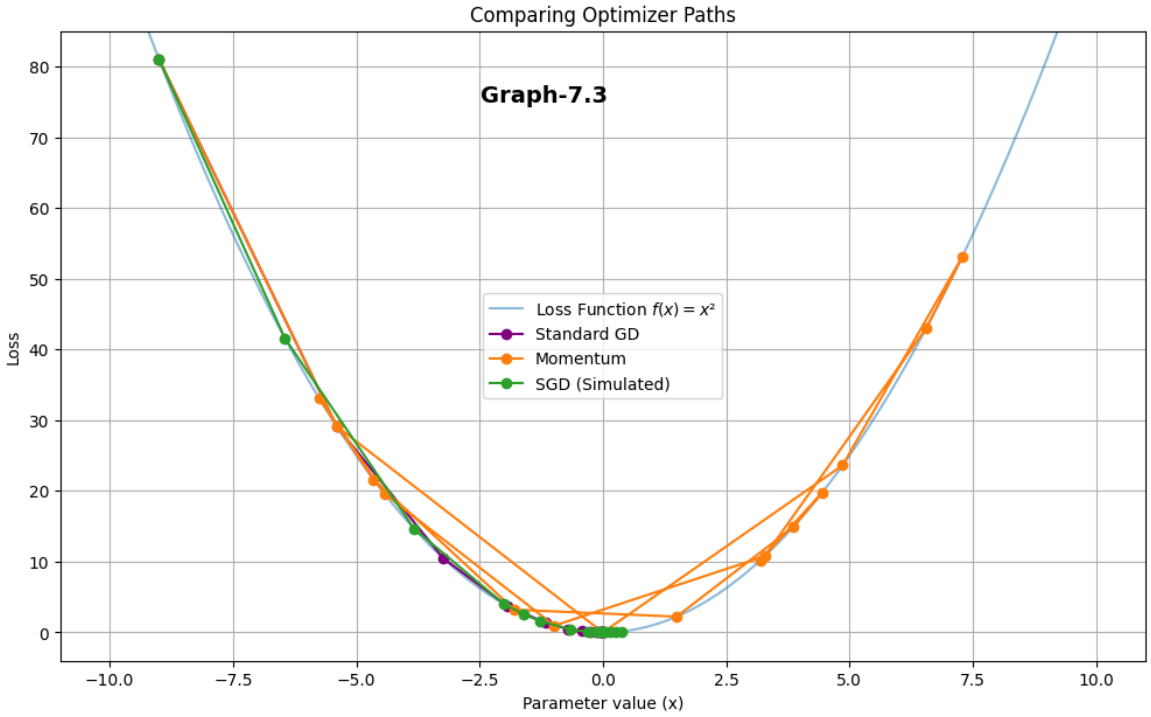


7.3 Optimizer Variants: Faster and Smarter Paths

While standard gradient descent works, it can be slow or get stuck. More advanced optimizers have been developed to address these issues.

- **Stochastic Gradient Descent (SGD):** Instead of calculating the loss on the entire dataset (which is slow), SGD uses a single randomly chosen data point or a small "mini-batch" for each step. This makes the path downhill much noisier and more erratic, but it's also much faster and can help escape from local minima.

- **Momentum:** This method adds a bit of "momentum" to each step, based on the direction of the previous step. Imagine a ball rolling down a hill; it builds up speed and doesn't stop instantly if the slope briefly levels out. This helps the optimizer power through flat spots and move faster towards the minimum.



- **Adam (Adaptive Moment Estimation):** This is the most common and often default optimizer. It's a "best of both worlds" approach that combines the idea of momentum with adaptive learning rates. It automatically adjusts the learning rate for each parameter, taking larger steps for less frequent parameters and smaller, more careful steps for frequent ones. This makes it very efficient and reliable for a wide range of problems.

The simulation of different gradient descent techniques is illustrated in Graph-7.3. It graph illustrates the different behaviors:

- **Standard GD** takes a smooth, predictable path.
- **Momentum** often overshoots the minimum at first but can converge faster.
- **SGD** has a much noisier, more random path due to the variance in gradients from single samples.

Module 7: Learning Materials & References

a) Gradient Descent in 3 minutes video
b) An overview of gradient descent optimization algorithms

Assessment: Optimization and Gradient Descent

Part A: Quiz

Multiple choice

7.1a What is the primary role of a loss (or objective) function in training a machine learning model?

a) To slow down the training process to prevent errors.
b) To measure the difference between the model's predictions and the actual correct values.
c) To select which data points to use for training.
d) To directly calculate the final, optimal parameters for the model in a single step.

7.2a The "rolling down the hill" analogy for gradient descent describes how the algorithm finds a minimum. In this process, each step is taken in which direction?

a) The direction of the gradient (steepest ascent).
b) A random direction to explore the loss surface.
c) The direction opposite to the gradient (steepest descent).
d) A direction parallel to the x-axis.

7.3a You are training a very large neural network on a massive dataset. Training is extremely slow. Which optimizer would be most suitable for speeding up the process by using noisy but fast updates based on small subsets of data?

a) Batch Gradient Descent
b) Stochastic Gradient Descent (SGD)
c) A genetic algorithm
d) Newton's method

7.4a The Adam optimizer is often a default choice because it is effective and reliable. What two key concepts does it combine?

a) Loss functions and feature scaling.
b) Random search and grid search.
c) Principal Component Analysis (PCA) and clustering.
d) Momentum and adaptive learning rates.

Part C: Python Exercises

7.1c Define and Plot a Loss Function

Write a python script to visualize a simple quadratic loss function, where the goal is to find the value of x that makes loss equal to zero. The graph would be similar to Graph-7.1

7.2c Visualize Gradient Descent

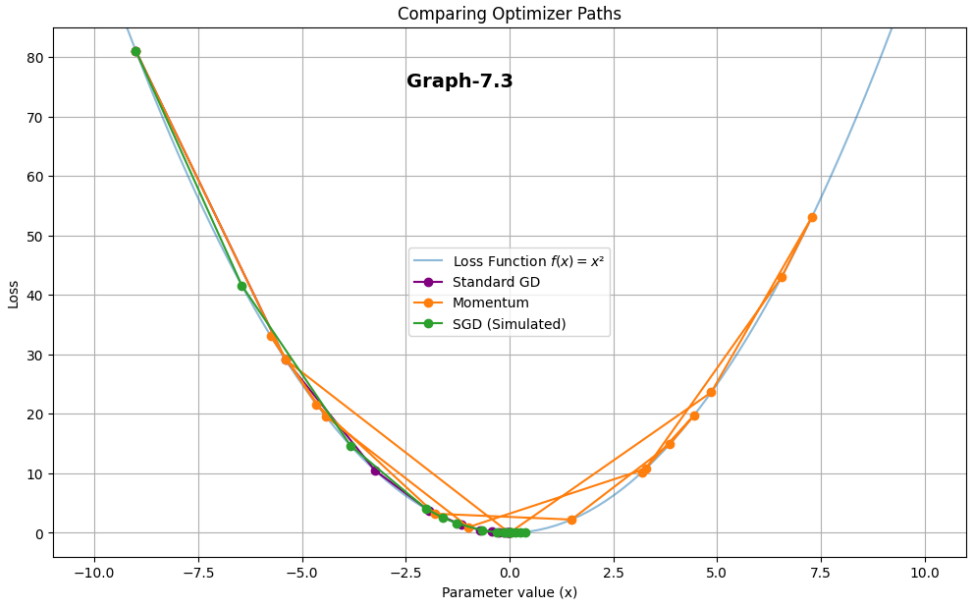
Write a Python Script to simulate the steps down the hill of our loss function. The algorithm takes progressively smaller steps as it gets closer to the minimum, where the slope (gradient) is less steep. Graph the output similar to Graph-7.2

7.3c Simulate Different Optimizers

Write a python script to plot different behaviors:

1. **Standard GD** takes a smooth, predictable path.
2. **Momentum** often overshoots the minimum at first but can converge faster.
3. **SGD** has a much noisier, more random path due to the variance in gradients from single samples.

Plot the graph of different optimizers, the graph should be similar to Graph-7.3.



Module 8: Advanced Calculus for Neural Networks

Neural networks are a type of machine learning model inspired by the structure and function of the human brain. Their main purpose is to learn patterns from data, enabling computers to make predictions, recognize images or speech, and automate decision-making tasks.

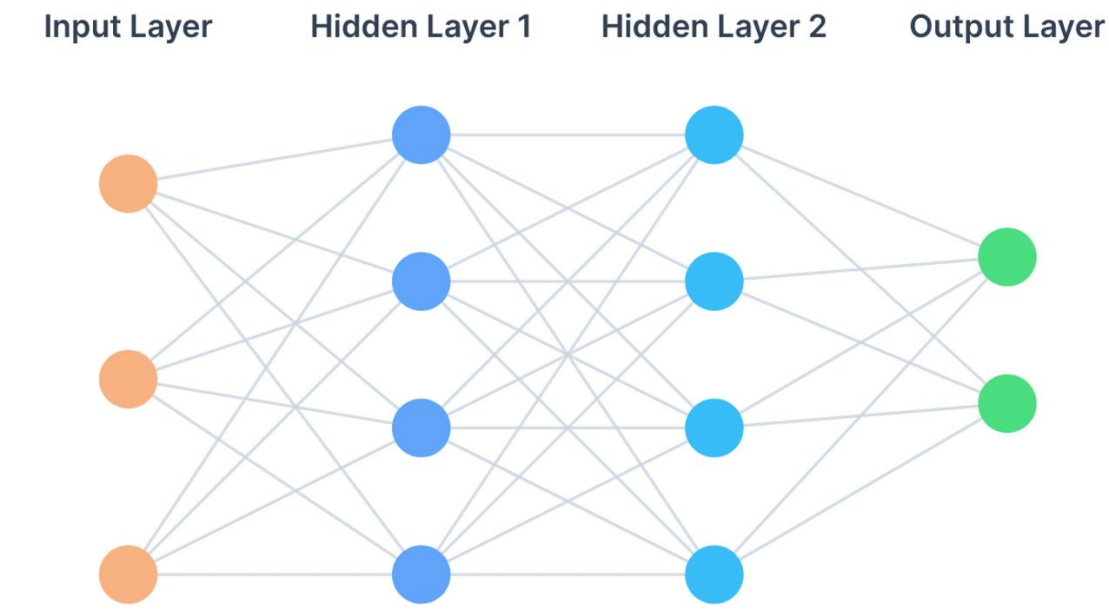


Fig-8.1 A Simple Neural Network

Core Concepts

- **Artificial Neurons**: The basic units, or nodes, in a neural network. Each receives inputs, processes them, and passes the result to other neurons.
- **Layers**:
    - **Input layer** receives the raw data.
    - **Hidden layers** perform intermediate computations.
    - **Output layer** generates the final prediction or result.
- **Connections and Weights**: Neurons are connected to one another, and each connection has a "weight" that determines how much influence one neuron has on another's activation.
- **Activation Function**: This function decides whether a neuron should be activated based on the processed input.

How Neural Networks Learn

1. **Training with Data**: The network is shown many examples (like labeled images). For each example, it makes a prediction, which is compared to the actual result.
2. **Error Calculation**: The difference between the network's prediction and the correct answer (the error) is measured.
3. **Updating Weights**: The network adjusts its weights using algorithms (like backpropagation) to reduce the error over time.
4. **Iterative Improvement**: Through many training cycles, the network gets better at making correct predictions.

8.1 The Chain Rule: Key to Backpropagation

The chain rule of calculus allows us to compute derivatives of composite functions. In neural networks, each layer applies a function to the output of the previous layer. The chain rule is fundamental during backpropagation, where gradients "flow backward" from the loss through each layer to update the weights efficiently.

8.2 How it Applies to Neural Networks

Think of a neural network as a long chain of nested functions.

1. The input data goes into the first layer, producing an output.
2. That output becomes the input for the second layer, which produces its own output.
3. This continues until the final layer produces the network's prediction.
4. Finally, a **loss function** measures how wrong this prediction is compared to the actual target.

So, the final loss is a function of the last layer's output, which is a function of the layer before it, and so on, all the way back to the initial weights. This creates a long composite function.

The Chain Rule is a fundamental calculus rule for finding the derivative of a composite function—essentially, a function nested inside another function. In simpler terms, if a variable z depends on y, and y in turn depends on x, the Chain Rule helps us find how z changes when x changes.

8.2a Why It's the Key to Backpropagation

The goal of training a neural network is to adjust its weights to minimize the loss. To do this, we need to calculate the **gradient** of the loss with respect to every single weight in the network (i.e., how much does a small change in a weight affect the final loss?).

This is where the **Chain Rule** becomes crucial for **backpropagation**:

- **Decomposing the Problem:** Directly calculating the derivative of the loss with respect to a weight in an early layer is incredibly complex. The Chain Rule allows us to break this massive problem down into a series of smaller, manageable steps.
- **Backward Flow of Gradients:** Backpropagation starts at the end, by calculating the derivative of the loss with respect to the final layer's output. Then, using the Chain Rule, it steps backward one layer at a time. It calculates the gradients for the last layer's weights, then uses those results to help calculate the gradients for the second-to-last layer, and so on.
- **Efficiency:** This process reuses calculations from the layer ahead to compute the gradients for the current layer. This "chaining" of derivatives makes the process computationally efficient, allowing us to train even very deep networks.

In essence, the Chain Rule provides the mathematical machinery to efficiently pass the error signal (gradient) backward through the network, telling each weight exactly how it should be adjusted to improve the overall performance.

Let's use a simplified example. Imagine the loss L is determined by the output of Layer 2, a_2, which is determined by the output of Layer 1, a_1. To find how the loss changes with respect to a weight in Layer 1, w_1, we chain the derivatives:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

Backpropagation computes each of these smaller, local derivatives and multiplies them together to get the final gradient we need.

Example:

Below is a simple Python example that manually implements the chain rule for backpropagation in a single neuron. This demonstrates how the gradients are calculated and chained together without relying on an automatic differentiation library like PyTorch or TensorFlow.

The neuron will perform the following steps:

Linear combination: $z = w \cdot x + b$
Activation: $a = \sigma(z)$ (sigmoid z)
Loss: $L = (a - y_{true})^2$ represents (Mean Squared Error) MSE loss for a single data point. It's a very common way to measure how "wrong" a neural network's prediction is.

- **L (Loss):** This is the final score that quantifies the error. A higher value means a worse prediction, and a lower value means a better one. The goal of training is to make this number as close to zero as possible.
- **a (activation/prediction):** This is the output value from the final neuron of the network. It's the model's prediction or "guess."
- **ytrue (true value):** This is the correct, ground-truth label for the input data. It's the answer we want the network to learn to produce.
- **(a−ytrue):** This is the simple difference, or **error**, between the prediction and the true value.
- **(...)2 (Squared):** The error is squared for two key reasons:
    1. **It makes the error positive.** It doesn't matter if the prediction was too high or too low; the squared result is always a positive number representing the magnitude of the error.
    2. **It penalizes larger errors more severely.** An error of 0.2 becomes 0.04, but a larger error of 3 becomes 9. This forces the network to pay more attention to fixing its biggest mistakes.

In short, this formula calculates a score that is small when the network's guess is close to the right answer and gets very large, very quickly, as the guess gets worse.

Our goal is to find the gradient of the loss with respect to the weigh

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Let look at the Python script itself:

```
import numpy as np

# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Derivative of the sigmoid function
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

# --- 1. Setup ---
```

```
# A single data point
x = 2.0
y_true = 1.0 # The true label

# Initial parameters (weights and bias)
w = 0.5
b = 0.1

# --- 2. Forward Pass ---
# Calculate the output of the neuron
z = w * x + b      # z = 0.5 * 2.0 + 0.1 = 1.1
a = sigmoid(z)     # a = sigmoid(1.1) ≈ 0.75

# Calculate the loss (how wrong the prediction is)
loss = (a - y_true)**2 # loss ≈ (0.75 - 1.0)**2 ≈ 0.0625

print(f"Forward Pass Results:")
print(f"z = {z:.4f}, a (prediction) = {a:.4f}, Loss = {loss:.4f}\n")

# --- 3. Backward Pass (Applying the Chain Rule) ---
# Calculate each part of the chain rule individually

# Part 1: Derivative of Loss w.r.t. Activation (dL/da)
# dL/da = 2 * (a - y_true)
dL_da = 2 * (a - y_true) # 2 * (0.75 - 1.0) = -0.5

# Part 2: Derivative of Activation w.r.t. Linear part (da/dz)
# da/dz = sigmoid_derivative(z)
da_dz = sigmoid_derivative(z) # ≈ 0.187

# Part 3: Derivative of Linear part w.r.t. Weight (dz/dw)
# dz/dw = x
dz_dw = x # = 2.0

# --- 4. Combine Gradients ---
# Chain the derivatives together to get the final gradient
# dL/dw = dL/da * da/dz * dz/dw
dL_dw = dL_da * da_dz * dz_dw

print(f"Backward Pass (Chain Rule) Results:")
print(f"dL/da = {dL_da:.4f}")
print(f"da/dz = {da_dz:.4f}")
print(f"dz/dw = {dz_dw:.4f}")
```

```
print("---------------------------------")
print(f"Final Gradient (dL/dw) = {dL_dw:.4f}")

# The result dL_dw tells us how to adjust 'w' to decrease the loss.
# Since it's negative, we should increase 'w'.
```

Explanation

1. **Forward Pass**: We compute the neuron's prediction (a) and see how far off it is from the target (y_true) by calculating the loss.
2. **Backward Pass**: This is the **chain rule in action**. We calculate the derivative at each step, moving backward from the loss.
    - dL_da: Measures how much the loss changes for a small change in the final activation a.
    - da_dz: Measures how much the activation a changes for a small change in its input z.
    - dz_dw: Measures how much z changes for a small change in the weight w.
3. **Combine Gradients**: By multiplying these three values, we get dL_dw, the overall gradient that links a change in the weight w all the way to its effect on the final loss L. This value is exactly what an optimization algorithm like Gradient Descent would use to update the weight w.

Output:

```
Forward Pass Results:
z = 1.1000, a (prediction) = 0.7503, Loss = 0.0624

Backward Pass (Chain Rule) Results:
dL/da = -0.4995
da/dz = 0.1874
dz/dw = 2.0000
---------------------------------
Final Gradient (dL/dw) = -0.1872
```

What does the output mean? Did the neural network fail?

No, the neural network did not fail. In fact, these results show that it is **working exactly as intended** for a single step of the training process.
The output is not the final answer but rather the crucial information the network needs to **learn** and improve.

What the Loss Means?

- **Loss = 0.0624**: This simply means the network's initial prediction (0.7503) was not perfect when compared to the true target (1.0). A non-zero loss is expected, especially before training. The goal of training is to reduce this number.

What the Gradient Means (The Most Important Result)?

This is the key to understanding backpropagation.

- **Final Gradient (dL/dw) = -0.1872**: This value is the **slope** of the loss function with respect to the weight w. It tells us how the loss will change if we change the weight.

Here's the critical interpretation:

- **The Sign is Direction**: The negative sign tells us that if we **increase** the weight w, the loss will **decrease**.
- **The Magnitude is Steepness**: The value (0.1872) indicates how sensitive the loss is to changes in w.

Think of it like being on the side of a hill and wanting to get to the bottom (minimum loss). The gradient tells you which way is straight up. To go down, you must take a step in the **opposite** direction.

Since the gradient is negative (-0.1872), an optimization algorithm (like Gradient Descent) will update the weight by moving in the opposite direction—it will **increase w**. This will push the prediction (a) higher, moving it closer to the target of 1.0 and reducing the loss in the next iteration.

In short, the network hasn't failed; it has successfully calculated the exact direction it needs to adjust its weight to become more accurate.

8.3 Jacobian and Hessian Matrices: Sensitivity and Curvature

The Jacobian matrix generalizes the concept of a derivative to functions that have multiple inputs and multiple outputs (vector-valued functions). It captures all the first-order partial derivatives of the function in a single matrix.

In essence, Jacobian tells you the **sensitivity** of every output variable to every input variable.

8.3a Jacobian Matrix

Imagine a function **f** that takes n input variables $(x_1,x_2,...,x_n)$ and produces m output variables $(y_1,y_2,...,y_m)$. The Jacobian matrix **J** is an $m \, x \, n$ matrix where each entry (i,j) is the partial derivative of the i-th output with respect to the j-th input:

$$J_{ij} = \frac{\partial xj}{\partial yi}$$

So, the full matrix looks like this:

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Each row corresponds to one output variable, and each column corresponds to one input variable.

Example:

Here are Python code examples for calculating the Hessian matrix using two popular libraries: **JAX** for numerical computation and **SymPy** for symbolic computation.

JAX Example (Numerical Calculation)

This is the most common approach in machine learning and optimization, where you need the numerical value of the Hessian at a specific point.

```
import jax
import jax.numpy as jnp
from IPython.display import display, Latex
from sympy import latex
from sympy import Matrix
# Define a simple scalar-valued function with two inputs
# f(x, y) = x^3 + 2xy^2
def my_function(v):
  x, y = v
  return x**3 + 2 * x * y**2

f_x=("$Function: f(x, y) = x^3 + 2xy^2$")
display(Latex(f_x))
# display(Latex("Function: ", latex("f(x, y) = x^3 + 2xy^2")))
```

```
# Create a function that computes the Hessian of my_function
hessian_fn = jax.hessian(my_function)

# Define a point at which to evaluate the Hessian
point = jnp.array([2.0, 3.0]) # Evaluate at (x=2, y=3)

# Calculate the Hessian matrix at the specified point
hessian_matrix = hessian_fn(point)

print(f"Function evaluated at {point}: f(x,y) = {my_function(point):.2f}")
print("Hessian matrix at that point:")
print(hessian_matrix)
```

Function: $f(x,y) = x^3 + 2xy^2$

```
Function evaluated at [2. 3.]: f(x,y) = 44.00
Hessian matrix at that point:
[[12. 12.]
 [12.  8.]]
```

SymPy Example (Symbolic Calculation)

This approach is useful for seeing the general mathematical formula for the Hessian before plugging in any specific numbers.

```
import sympy
from IPython.display import display, Latex
# Define symbolic variables
x, y = sympy.symbols('x y')

# Define the same function symbolically
f_expr = x**3 + 2 * x * y**2

# Define the list of variables for differentiation
variables = [x, y]

# Calculate the symbolic Hessian matrix
hessian_matrix_symbolic = sympy.hessian(f_expr, variables)

print("Symbolic function:")
# print(f_expr)
display(Latex(f"f(x, y) = ${latex(f_expr)}$"))
```

```
print("\nSymbolic Hessian matrix:")
# sympy.pprint(hessian_matrix_symbolic)
display(Latex(f"Hessian = ${latex(hessian_matrix_symbolic)}$"))
# You can substitute values to get the numerical result
point_to_eval = {x: 2, y: 3}
numerical_result = hessian_matrix_symbolic.subs(point_to_eval)

print("\nHessian evaluated at (x=2, y=3):")
# sympy.pprint(numerical_result)
display(Latex(f"Hessian evaluated at (x=2, y=3) = ${latex(numerical_result)}$"))
```

Symbolic function: $f(x,y) = x^3+2xy^2$

Hessian $\begin{bmatrix} 6x & 4y \\ 4y & 4x \end{bmatrix}$

Hessian evaluated at (x=2, y=3):

Hessian evaluated at(x=2, y=3) $\begin{bmatrix} 12 & 12 \\ 12 & 8 \end{bmatrix}$

8.3b Jacobina & Neural Networks?

In a neural network, we can think of many components as vector-valued functions, making the Jacobian a powerful analytical tool.

1. **Analyzing Layer Sensitivity**: A neural network layer is a function that takes an input vector (activations from the previous layer) and produces an output vector (activations for the next layer). The Jacobian of this function tells us precisely how sensitive each output activation is to every input activation. This can help diagnose issues like vanishing or exploding gradients, as it shows how information is being stretched or squashed as it flows through the network.

2. **Relation to the Gradient**: The most common use of derivatives in deep learning is for calculating the **gradient**, which is what you use in backpropagation to update the weights. The gradient is actually a special case of the Jacobian.

    - Consider the loss function. It takes all the network's weights (a very large input vector) and produces a single scalar value (the loss).
    - The Jacobian of the loss function with respect to the weights would be a matrix with just one row. This single-row matrix is exactly the **gradient vector**.

The **Jacobian** measures how changes in inputs affect the outputs of a (possibly vector-valued) function. In neural networks, it's used to analyze how parameter (weight) changes affect the loss or activations.

While computing the full Jacobian can be very expensive for large networks, the concept is fundamental. It provides the mathematical framework for understanding the sensitivity and local linear behavior of a network, and the gradient used in backpropagation is a direct application of it.

8.3c Hessian Matrix

While the gradient (first derivative) tells you the slope of a function, the Hessian matrix tells you about its curvature. It's the multi-dimensional equivalent of the second derivative, collecting all the second-order partial derivatives of a scalar-valued function into a single square matrix.

In simpler terms, if the gradient tells you which way is "downhill," the Hessian tells you if you're in a bowl, on top of a dome, or on a Pringles chip (a saddle).

For a function f that takes a vector of inputs $x=(x_1,x_2,...,x_n)$, the Hessian matrix **H** is an $n \times n$ matrix where the entry at row i and column j is the second partial derivative:

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

The full Hessian matrix, which represents the second-order partial derivatives of a scalar-valued function f with respect to a vector of variables $x = (x_1, \dots, x_n)$:

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial y_2}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

The **Hessian** is the matrix of second derivatives (curvature) of a scalar-valued function with respect to vector inputs. It reveals important properties such as:

- local minima, the surface curves upwards in all directions, like a bowl. An optimization algorithm has found a valley
- Local maxima, the surface curves downwards in all directions like a dome

- Saddle point: the surface curves up in some directions and down in others, like a saddle or mountain pass. This is a common challenge in neural network optimization, as an algorithm can get "stuck" here because the gradient is zero, even though it's not a true minimum.

Additionally, the Hessian's eigenvalues help diagnose if a minimum is flat, sharp, or if the model landscape presents optimization challenges.

Module 8: Learning Materials & References

a) What is a Neural Network?
b) Deep Learning Tutorial
c) But what is a neural network? | Deep learning chapter 1 Video
d) Gradient descent, how neural networks learn | Deep Learning Chapter 2 video
e) Neural Networks and Deep Learning
f) CHAPTER 1 Using neural nets to recognize handwritten digits
g) Backpropagation, intuitively | Deep Learning Chapter 3

Assessment: Advanced Calculus for Neural Networks

Part A: Quiz

8.1a What is the main purpose of the activation function in an artificial neural network?
a) It stores the network's parameters
b) It introduces non-linearity so the network can learn complex patterns
c) It initializes the weights of neurons
d) It determines the learning rate of the network

8.2a Which layer in a neural network is responsible for generating the final prediction or output?
a) Input layer
b) Hidden layer
c) Output layer
d) Activation layer

8.3a What is the output range of the sigmoid activation function commonly used in neural networks?
a) 0 to 1
b) Any real number
c) -1 to 1
d) 0 to infinity

8.4a Which of the following statements best describes an artificial neuron?
a) A data storage unit in a neural network
b) A type of activation function used in neural networks
c) A loss function used for training
d) The basic processing unit that receives input, applies weights, and produces an output

8.5a Which technique is most commonly used to initialize the weights in a deep neural network to help prevent vanishing or exploding gradients?
a) Assigning all weights to zero
b) Random initialization
c) Using the same value for all weights
d) Initializing weights with very large values

8.6a What is the primary role of the Chain Rule in the backpropagation algorithm?
A. To initialize the weights of the neural network before training begins.
B. To select the best activation function for each layer to prevent gradients from vanishing.
C. To efficiently compute the gradient of the loss function with respect to each weight by propagating the error backward through the network.

8.7a The Jacobian matrix of a neural network layer represents...

A. The curvature of the loss function indicating if a critical point is a minimum or maximum.
B. The sensitivity of each output activation with respect to each input activation.
C. The total loss of the network after a forward pass.
D. The rate of change of a single output with respect to a single input.

8.8a What information does the Hessian matrix provide about a function's landscape at a critical point?

A. The local curvature of the function.
B. The set of all possible inputs to the function.
C. The linear approximation of the function.
D. The direction of steepest ascent (the gradient).

8.9a At a critical point (where the gradient is zero), you compute the eigenvalues of the Hessian matrix. If you find both positive and negative eigenvalues, what does this indicate?

A. A local minimum, because the function curves upwards in at least one direction.
B. A region where the function is completely flat.
C. A local maximum, because the function curves downwards in at least one direction.
D. A saddle point.

Part C: Python Exercises

8.1c Manual Backpropagation with the Chain Rule

Complete the Python function calculate_gradient below. You need to manually implement the chain rule to find the gradient of the loss with respect to the weight w.

Context:

- Linear step: $z = w * x + b$
- Activation: $a = \sigma(z)$ (sigmoid (z))
- Loss (MSE): loss = $(a - y_{true})^2$
- Goal: Find dL_dw by calculating and combining dL_da, da_dz, and dz_dw.

```
import numpy as np

def sigmoid(x):
    """Sigmoid activation function."""
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    """Derivative of the sigmoid function."""
    return sigmoid(x) * (1 - sigmoid(x))

def calculate_gradient(x, y_true, w, b):
    """
    Calculates the gradient of the loss with respect to the weight 'w'
    using manual backpropagation (the chain rule).

    Args:
      x (float): The input value.
      y_true (float): The true label.
      w (float): The weight parameter.
      b (float): The bias parameter.

    Returns:
      float: The gradient of the loss with respect to w (dL/dw).
    """
    # 1. Forward Pass
    z = w * x + b
    a = sigmoid(z)
    loss = (a - y_true)**2
```

```
    # 2. Backward Pass (Chain Rule)
    # TODO: Calculate the three components of the chain rule.
    # dL/da = ?
    # da/dz = ?
    # dz/dw = ?
    dL_da = 0 # Replace with your calculation
    da_dz = 0 # Replace with your calculation
    dz_dw = 0 # Replace with your calculation

    # 3. Combine gradients to get the final result
    dL_dw = dL_da * da_dz * dz_dw

    return dL_dw

# --- Test your function ---
x_val = 2.0
y_val = 0.9
w_val = 0.4
b_val = 0.1

final_gradient = calculate_gradient(x_val, y_val, w_val, b_val)
print(f"The calculated gradient dL/dw is: {final_gradient:.4f}")
# Expected output is approximately: -0.1554
```

8.2c Hessian Matrix and Critical Point Analysis

Use Python jax library to compute the Hessian matrix of the $f(x, y) = x^4 + y^4 - 4xy + 1$at the critical point (1, 1). Then, based on the Hessian, determine if this point is a local minimum, local maximum, or saddle point.

A point is a local minimum if the Hessian is positive-definite (all eigenvalues are positive). It's a saddle point if there's a mix of positive and negative eigenvalues.

```
import jax
import jax.numpy as jnp
from jax import grad, hessian

# The function to analyze
def f(v):
    x, y = v
    return x**4 + y**4 - 4*x*y + 1
```

```
Sample output:
Hessian matrix at
(1, 1):
[[12. -4.]
 [-4. 12.]]

Eigenvalues of the
Hessian:
[16.+0.j  8.+0.j]
```

```
# Define the critical point to test
critical_point = jnp.array([1.0, 1.0])

# --- Your code goes here ---

# 1. Create a function that computes the Hessian of f.
# TODO: hessian_fn = ?
hessian_fn = None # Replace this line

# 2. Compute the Hessian matrix at the critical point.
# TODO: H = ?
H = None # Replace this line

# 3. Find the eigenvalues of the Hessian matrix.
# Hint: Use jnp.linalg.eigvals()
# TODO: eigenvalues = ?
eigenvalues = None # Replace this line


# --- Print your results ---
if H is not None and eigenvalues is not None:
    print("Hessian matrix at (1, 1):")
    print(H)
    print("\nEigenvalues of the Hessian:")
    print(eigenvalues)

    # 4. Add a comment explaining your conclusion
    # Based on the eigenvalues, this point is a _______.
else:
    print("Please complete the code to see the results.")
```

Module 9: Probability Theory - Modeling Randomness and Uncertainty

Probability theory is the mathematical framework for quantifying uncertainty, which is essential for building ML systems that can make robust predictions from noisy, real-world data.

9.1 Random Variables

Think of a random variable as a placeholder for an outcome you don't know yet, like the number that will come up on a dice roll. A probability distribution is like a rulebook that tells you the chance of each possible outcome. For a fair die, the distribution says each number from 1 to 6 has a 1/6 chance.

in ML, models use distributions as "blueprints" for the data. For example, a model might assume that human heights follow a bell-curve (Normal) distribution to better understand and predict them.

A random variable is a variable whose value is a numerical outcome of a random phenomenon. They are central to ML because they model the quantities we want to predict or the features we use for prediction. They come in two main types:

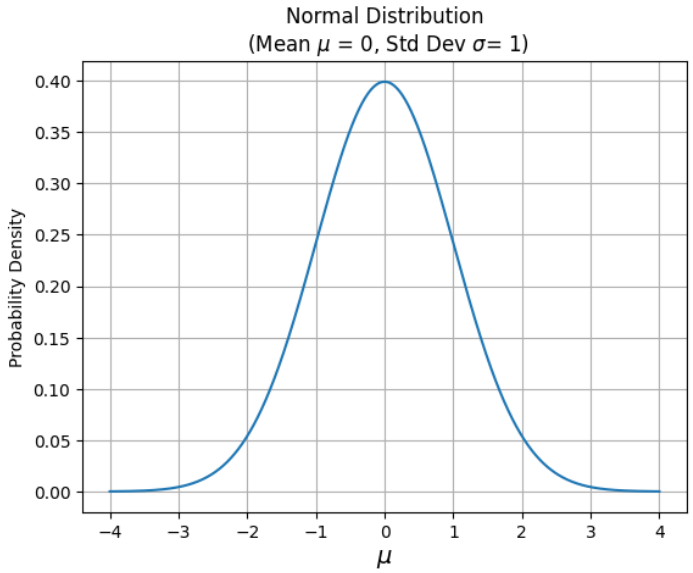
a) Discrete Random Variables: Take on a finite or countably infinite number of values (e.g., the result of a dice roll {1, 2, 3, 4, 5, 6}, or the number of fraudulent transactions). They are described by a Probability Mass Function (PMF), $P(X=x)$, which gives the probability of each specific outcome.
b) Continuous Random Variables: Can take any value within a given range (e.g., a person's height, temperature). They are described by a Probability Density Function (PDF), $f(x)$. The probability of the variable falling within a range a to b is the PDF over that range: $P(a \leq X \leq b) = \int_a^b f(x)dx$.

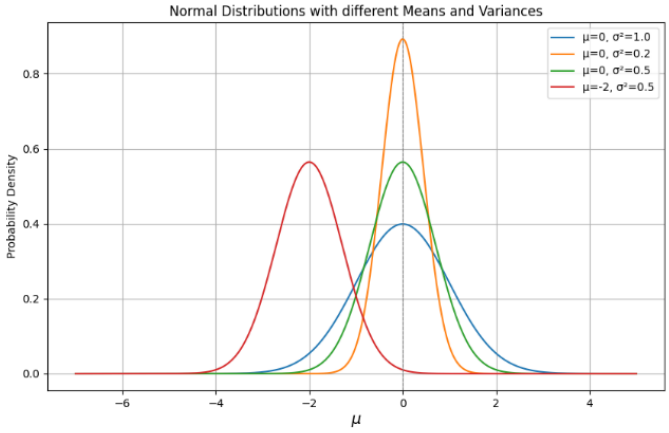
9.2 Probability Distributions

We often assume that our data is generated from a specific probability distribution. For example, the Gaussian (Normal) distribution is a common assumption for features in models like Linear Discriminant Analysis (LDA), while the Bernoulli distribution models binary outcomes like click/no-click.

9.2a Normal (Gaussian) Distribution

The Normal distribution is used to model continuous data that clusters around a central mean. It's defined by its **mean** (μ) (mu) and **standard deviation** (σ)(sigma).





1. Modeling Continuous Features

A common assumption in many ML models is that continuous features in the dataset follow a Normal distribution.

Example:

> In a medical AI model that predicts heart disease risk, a feature like patient systolic blood pressure might be modeled as a Normal distribution. The model would learn the average blood pressure (μ) and the typical variation (σ) from the training data. This

assumption helps models like Linear Discriminant Analysis (LDA) find optimal decision boundaries.

2. Modeling Prediction Errors in Regression

In Linear Regression, a core assumption is that the errors (the difference between the predicted and actual values), also known as residuals, are normally distributed with a mean of zero.

Example:

An AI model predicts a house's price. For a house that is actually worth $500,000, the model might predict $510,000 (an error of +$10,000). For another, it might predict $495,000 (an error of -$5,000). The assumption is that the collection of all these errors, positive and negative, will form a bell curve centered at $0. This validates the model's performance and is crucial for calculating confidence intervals for predictions.

4. Generative Models and Clustering

Generative models often use Normal distributions to create new data points or to define clusters.

Example:

A Gaussian Mixture Model (GMM) can be used to cluster customers into different segments based on their spending habits. The model might assume that there are, for instance, three customer segments. It would then model the spending behavior (e.g., annual purchase amount) within each segment as a separate Normal distribution. Each cluster is a "Gaussian" with its own mean and variance, representing a distinct customer type (e.g., low-spenders, mid-range, and high-spenders).

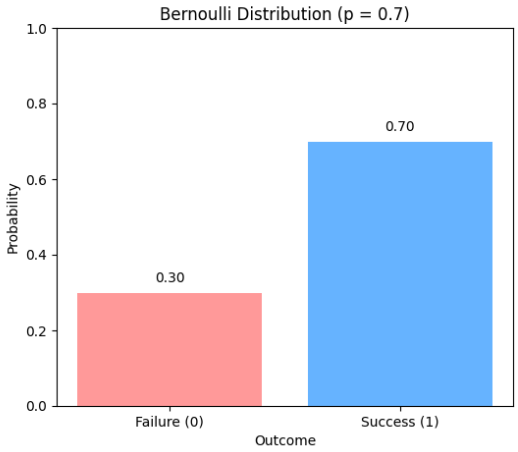
The general form of normal probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

The parameter μ is the mean or expectation of the distribution (and its median and mode), while the parameter σ^2 is the variance. The standard deviation of the distribution is σ (sigma). A random variable with a Gaussian distribution is said to be normally distributed and is called a normal deviate.

9.2b Bernoulli Distribution

The Bernoulli distribution models the outcome of a single trial with only two possible results, which we can label as 1 (success) or 0 (failure). It is defined by a single



parameter, p, the probability of success. Bernoulli distribution is a special case of the binomial distribution, where n number of trials is 1.

1. Binary Classification

This is the most common use case. Any "yes/no" or "A/B" prediction task is fundamentally a Bernoulli process.

Example:

> In **email spam detection**, the outcome for each email is either **spam (1)** or **not spam (0)**. A logistic regression model doesn't output a 0 or 1 directly; instead, it outputs the probability, p, that the email is spam. This predicted probability pp is the parameter of the Bernoulli distribution for that email's outcome. If $p>0.5$, the model classifies it as spam.

2. Modeling Binary Features in Text

In Natural Language Processing (NLP), a simple way to represent a document is by noting the presence or absence of words from a vocabulary.

Example:

> A **Bernoulli Naive Bayes** classifier for sentiment analysis might have a vocabulary of words like {"good", "bad", "great"}. For a review that says "good, great", the input

vector would be **{1, 0, 1}**, representing that "good" is present (1), "bad" is absent (0), and "great" is present (1). Each feature is treated as a Bernoulli random variable.

3. Recommendation Systems

Bernoulli distributions can model user interactions with items.

Example:

An AI for a streaming service wants to recommend a movie. It can model user behavior as a Bernoulli trial: did the user **click 'play' (1)** on a recommended movie, or did they **ignore it (0)**? The system then builds a model to predict the probability, *p*, that a user will click on a given movie, and it recommends the movies with the highest predicted click probability. Of course. In simple terms, probability helps ML models measure and express how confident they are in their predictions, because the real world is messy and uncertain.

9.3c Binomial distribution

The Binomial distribution models the number of "successes" in a fixed number of independent trials, where each trial has only two possible outcomes. It's like flipping a weighted coin multiple times and counting the number of heads.

The General Binomial Probability Formula:

Probability of k out of n ways:$P(k\ out\ of\ n) = \frac{n!}{k!(n-k)!} p^k (1-P)^{n-k}$

To use the Binomial distribution, you need three things:

1. A Fixed Number of Trials (n): This is the total number of times an action is performed.
    - An AI sends a promotional email to 1,000 users. Here, $\mathrm{n}=1000$.
2. Two Possible Outcomes: Each trial must be a "success" or "failure".
    - For each email sent, the outcome is either 'opened' (success) or 'not opened' (failure).
3. Constant Probability of Success (p): The probability of success must be the same for every trial.
    - A predictive model, like a logistic regression classifier, estimates that the probability of any single user opening the email is 20%. Here, $\mathrm{p}=0.2$.

Example: Evaluating a Classification Model

Let's say you've trained a computer vision model to identify defective products on an assembly line. You know from testing that your model has 95% accuracy for this task.

Now, you want to know the probability of its performance on the next batch of 50 products.

- Trials (n): 50 products to be classified.
- Success: The model correctly identifies a product (defective or not).
- Probability of Success (p): 0.95 (the model's known accuracy).

Using Binomial distribution, we can answer questions like:

- "What is the probability that the model classifies exactly 48 out of 50 products correctly?"
- "What is the probability it classifies at least 45 out of 50 correctly?"
- "What is the probability it classifies fewer than 40 correctly?" (This would be an alarming result).

9.3d Poisson Discrete Distribution

The Poisson distribution is a statistical tool designed to model count-based data, particularly when analyzing rare or infrequent events within a fixed interval of time or space. It is characterized by its ability to represent the probability of a number of discrete events occurring independently, given a constant average rate.

The Poisson distribution is defined by just one parameter:

- **Lambda (λ)**: The average number of events that occur in a specific interval.

It helps answer the question: "If I typically see **λ** events in an interval, what's the probability of seeing exactly **k** events in the next interval?" The events must be independent, meaning one event occurring doesn't make another more or less likely.

Imagine a customer service center wants to schedule the right number of agents each hour. They notice, on average, the center receives 15 calls per hour during peak times. The interval is one hour.

Let’s say the manager wants to determine the probability of receiving more than 20 calls in the next hour to ensure enough agents are available.

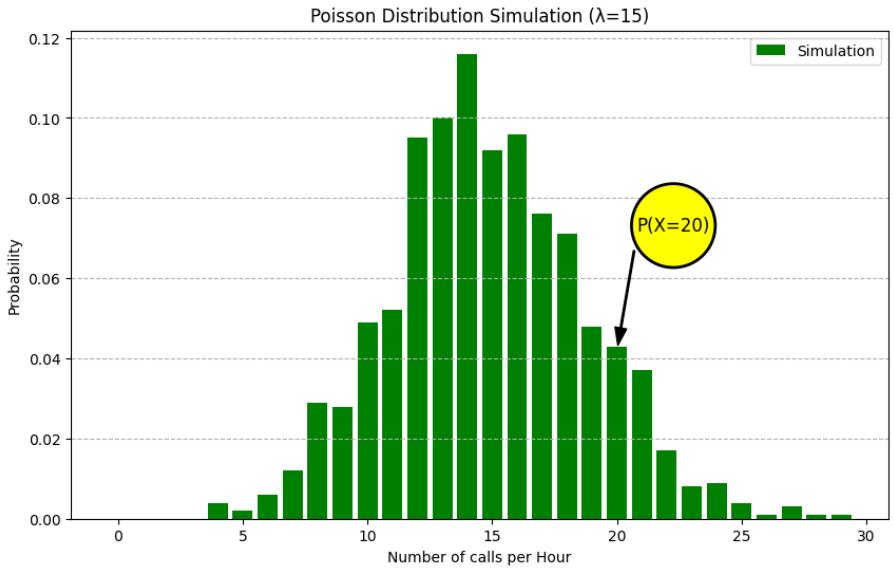
Using the Poisson distribution formula:

$$P(X = k) = \frac{e^{-\lambda}\lambda^{k}}{k!}$$

Where:

- λ = expected number of calls per hour 15.
- K = number of calls per hour 20
- $e \approx 2.71828$
- From standard normal tables: $P(Z>1.42) \approx 0.0778$
- The probability of getting *exactly* **20** calls in the next hour is 7.8%

The graph below simulate the number of calls per hour, highlights the 20 calls per hour.



9.3 Conditional Probability

This is the probability of an event happening *given that* another event has already occurred.

Example:

The chance of your lawn being wet is low on its own. But the conditional probability of your lawn being wet, *given that it's raining*, is very high.

This is the core of prediction. An ML model calculates the probability of an email being "spam" *given that* it contains the words "free money."

- Conditional Probability: The probability of an event occurring, given that another event has already occurred. It's defined as:

9.3a Bayes' Theorem

Bayes' Theorem is a clever way to update our beliefs as we get more evidence. It lets us flip a conditional probability around. If we know the chance of seeing certain words *if* an email is spam, Bayes' Theorem helps us calculate the chance an email is spam *if* it contains those words.

Start with an initial guess → Get new evidence → Make a better, updated guess.

It's the engine behind many learning systems, like Naive Bayes classifiers, which use it to classify text, detect diseases, and more by constantly updating their "beliefs" based on the data they see.

It mathematically describes the relationship between a conditional probability and its inverse. It allows us to update our beliefs about a hypothesis in light of new evidence.

The formula is:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

Where:

- P(H|E) is the posterior probability: the probability of the hypothesis H given the evidence E.
- P(E|H) is the likelihood: the probability of observing the evidence **E** given the hypothesis **H** is true.
- P(H) is the prior probability: our initial belief in the hypothesis H before seeing any evidence.
- P(E) is the evidence: the probability of observing the evidence.

Bayes' theorem is the backbone of Bayesian machine learning. It's used directly in classifiers like Naive Bayes and forms the basis for more complex models where we want to find the most probable model parameters given our data.

9.4 Expectation, Variance, and Covariance

These are summary statistics that describe the properties of probability distributions.

- **Expectation (Average)**: If you could repeat a random event (like a dice roll) millions of times, this would be the average outcome. It's the "most likely" central value. It's the weighted average of all possible values, where the weights are the probabilities, for a discrete variable X, it's $E[X] = \sum_x xP(x)$
- **Variance (Spread)**: This measures how spread out the results are. Low variance means outcomes are usually very close to the average, clustered around the mean. High variance means the outcomes are all over the place.
  It's defined as $Var(X) = E[(X - E[X])^2]$. The square root of the variance is the standard deviation.
- **Covariance**: A measure of how two random variables change together. A positive covariance means they tend to increase or decrease together, while a negative covariance means one tends to increase as the other decreases.

These concepts help us define a model's goals. For instance, we train a model to minimize the "expected" error in its predictions. Variance helps us understand a model's consistency and is a key part of the bias-variance tradeoff, which is about balancing model simplicity and complexity.

Expectation is fundamental to defining loss functions (we want to minimize expected loss). Variance is crucial for understanding model overfitting and is a key component of the bias-variance tradeoff. Covariance is used in dimensionality reduction techniques like Principal Component Analysis (PCA) to understand the relationships between features.

9.5 Independence and Conditional Independence

These are simplifying assumptions that make probabilistic models computationally tractable.

- **Independence**: Two random variables A and B are independent if the occurrence of one does not affect the probability of the other. Mathematically, two random variables A and B are independent if the occurrence of one does not affect the probability of the other $P(A,B) = P(A)P(B)$.
- **Conditional Independence**: Two variables A and B are conditionally independent given a third variable C if, once C is known, information about A provides no additional information about B. Mathematically, Two variables *A* and *B* are

conditionally independent given a third variable *C* if, once *C* is known, information about *A* provides no additional information about *B*, $P(A,B|C)=P(A|C)P(B|C)$.

The **Naive Bayes classifier** makes a "naive" assumption of conditional independence among all features given the class label. This assumption dramatically simplifies the calculation of the likelihood P(features|class), making the model fast and effective even with high-dimensional data.

Module 9: Learning Materials & References

a) https://calcworkshop.com/probability/bayes-theorem/
b) https://calcworkshop.com/discrete-probability-distribution/binomial-distribution/
c) https://www.mathsisfun.com/data/binomial-distribution.html
d) https://www.geeksforgeeks.org/data-science/binomial-distribution-in-business-statistics-definition-formula-examples/
e) https://byjus.com/maths/bayes-theorem/

Assessment: Probability Theory Modeling Randomness and Uncertainty

Part A: Quiz

9.1a What are the two main parameters that fully define a Normal (Gaussian) distribution?
A. Mean and Standard Deviation
B. Variance and Range
C. Mean and Median
D. Mode and Skewness

9.2a What is the primary purpose of Bayes' Theorem in the context of machine learning?
A. To measure the spread of data points.
B. To update a belief or hypothesis based on new evidence.
C. To determine if two events are independent.
D. To calculate the average of a dataset.

9.3a An AI model predicts whether a single customer click is fraudulent (yes/no). Which distribution best models this single event?

A. Binomial Distribution
B. Poisson Distribution
C. Bernoulli Distribution

D. Normal Distribution

9.4a A city's traffic department wants to model the number of accidents that occur at a specific intersection per day. They know the long-term average. Which distribution is most suitable for this task?

A. Bernoulli Distribution
B. Normal Distribution
C. Binomial Distribution
D. Poisson Distribution

9.5a A spam filter calculates the probability that an email is 'spam' *given that* it contains the word 'lottery'. What concept does this calculation best represent?

A. Expected Value
B. Conditional Probability
C. Marginal Probability
D. Joint Probability

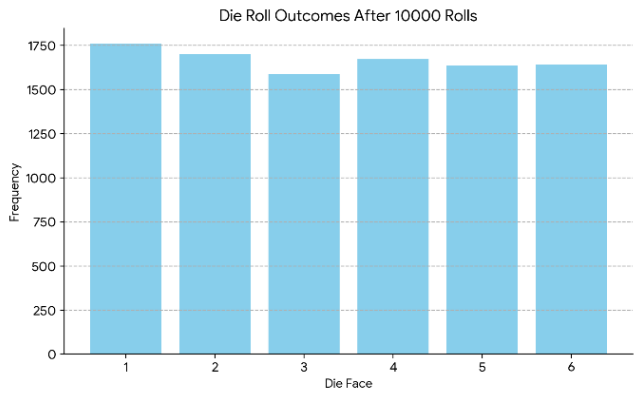
Part C: Python Exercises

9.1c Simulate a die roll and explore outcomes

A standard die roll is an example of a Discrete Uniform Distribution, where every outcome has an equal probability.

Write a Python script that simulates rolling a six-sided die 10,000 times and plots the frequency of each outcome. Explain the results.

Sample output:



9.2c Bernoulli (binary outcomes)

The **Bernoulli distribution** models a single trial with two possible outcomes (e.g., success/failure, yes/no, heads/tails).

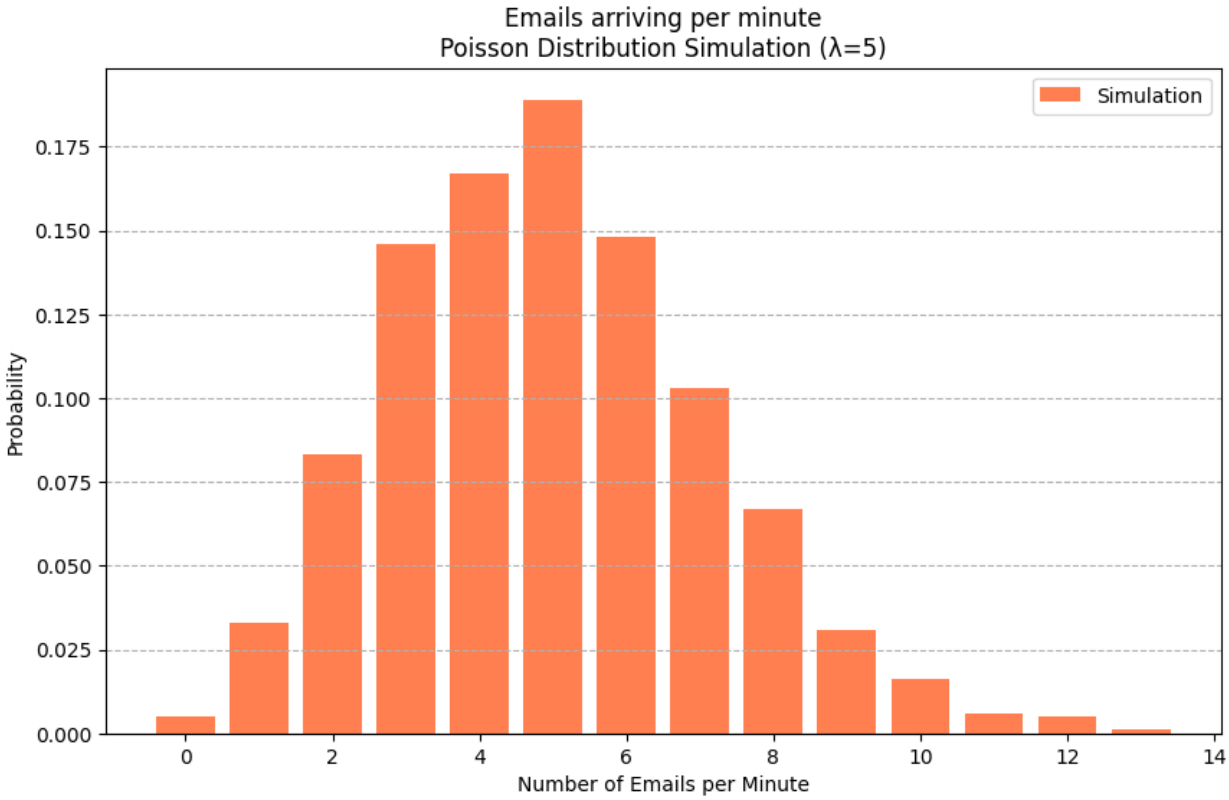
Write a python script to simulate 100 biased coin flips, where the probability of heads (success = 1) is 70%.

Sample output:
Simulation of 100 biased coin flips (p(heads)=0.7):
Number of Heads (1): 69
Number of Tails (0): 31

9.3c Poisson Discrete

The Poisson distribution models the number of events occurring in a fixed interval of time or space, given a known average rate **λ** (lambda).
Write a Python script to simulate the number of emails arriving per minute, where the average is 5 emails/minute.

Sample output:



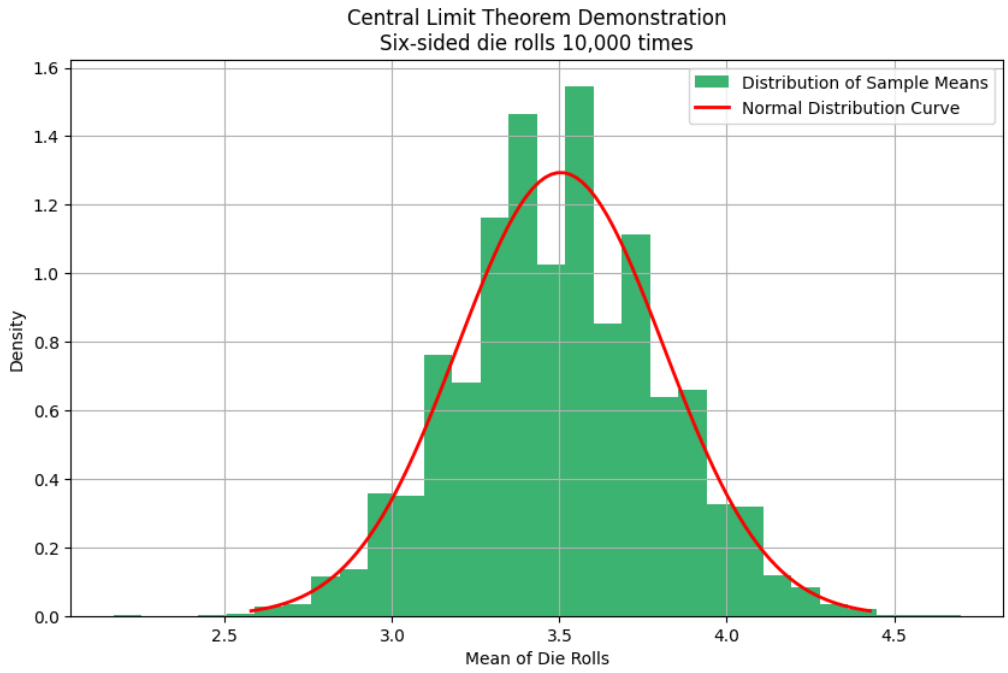
9.4.c Python Exercise: Simulating the Central Limit Theorem

In simple terms, the Central Limit Theorem states that if you take a large number of samples from *any* distribution (it doesn't have to be a normal one) and calculate the **mean** (average) of each sample, the distribution of those means will approximate a **Normal Distribution**.

This holds true even if the original population is not normally distributed at all—like with our die roll example, which has a flat (Uniform) distribution.

Write a Python script to demonstrate the CLT. The script will simulate rolling a six-sided die, which has a Uniform distribution. We'll roll it multiple times, find the average, and repeat this process thousands of times. According to the CLT, the histogram of these averages should look like a bell curve.

Sample output:



Module 10: Statistics & Inference for Data Accuracy

This module focuses on using the power of statistics to both understand your data and validate your AI/ML models. By applying statistical methods, you can explore and interpret your data before modeling and rigorously test whether your final models are genuinely effective—not just lucky. Solid statistical reasoning ensures your results are robust, trustworthy, and meaningful. Without statistics, building AI/ML models is little more than educated guesswork.

Think of it as the quality assurance step in your data science workflow.

10.1 Understand descriptive statistics (mean, variance) to interpret datasets.

Descriptive statistics are tools that summarize the key characteristics of a dataset. They help you get a "feel" for the data before you build a model. The two most important concepts are **central tendency** (where's the center of my data?) and **spread** (how spread out is my data?).

10.1a Central Tendency: what a "typical" data point looks like.

Mean (μ)(mu): The simple average of all your data points. It's the most common measure, but it can be easily skewed by outliers (i.e., unusually high or low values).

Imagine you've built a model to predict house prices. The "error" is the difference between your model's prediction and the actual price. If the mean error is $50,000, it means your model, on average, predicts $50,000 too high. You want a mean error close to zero, which tells you your model isn't systematically guessing too high or too low.
The mean is the sum of all the data points divided by the number data points.
Mathematically the mean can be expressed as:

$$(\mu) = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Where:

N = number of data points or observations

x_i = the i-th data point

10.1b Spread or Variability or Consistency

Variance & Standard Deviation: A measure of how spread out your data is. A low standard deviation means your data points are all very close to the average. A high standard deviation means they're all over the place.

- Variance $(\sigma^2)(sigma)$: This measures the average squared difference of each data point from the mean. A high variance means the data is very widespread; a low variance means it's tightly clustered around the mean. Mathematically:

  $$Variance(\sigma^2) = \frac{\sum(x_i - \mu)^2}{N}$$

  Here, x_i is each data point,
  μ is the mean, and N is the total number of data points.

- Standard deviation $(\sigma)sigma$: This is simply the square root of the variance. It's often more intuitive because it's the same units as your original data. A small standard deviation means that the data points tend to be very close to the mean, while a large standard deviation indicates that the data points are spread out over a wider range of values.

  $$\sigma = \sqrt{\sigma^2}$$

- Is standard deviation the same as Variance
  No, but they are closely related:

  $$StandardDeviation = \sqrt{Variance}, Variance = (StandardDeviation)^2$$

  We square the standard deviation to find the variance because variance is mathematically defined as the average of the squared differences from the mean.

Using the previous example, the house price model:

- A low standard deviation in your errors means your model is very consistent. Maybe it's always off by about $5,000. That's a predictable, and therefore more useful, model.
- A high standard deviation means your model is unpredictable. It might guess a price perfectly one time but be off by $100,000 next. This indicates an unreliable model, even if the mean error is zero.

10.2 Perform hypothesis testing and compute confidence intervals.

Descriptive statistics summarize your *sample* of data. Inferential statistics, like hypothesis testing and confidence intervals, help you make educated guesses (inferences) about the entire *population* the data came from.

10.2a Hypothesis Testing:

Hypothesis testing is a formal method for using your limited test data to make confident, data-driven claims about how your model might perform in the real world. It provides a structured way to test an assumption or idea.

The process begins with two competing claims:

- **Null Hypothesis (H_0):** This is the "default" assumption—there is no effect or no improvement. In this context, it means *the new model is not better than the old one*. To claim the new model is better, you need strong evidence against this assumption, usually demonstrated by a low p-value.
- **Alternative Hypothesis (H_1 or Ha):** This is what you hope to prove—that *the new model is better than the old one*. You conduct a statistical test and calculate a p-value: the probability of observing results as extreme as yours, assuming the null hypothesis is true. If the p-value is small (typically less than 0.05), it suggests your results are unlikely under the null hypothesis, giving you grounds to reject the null hypothesis in favor of the alternative**.**

  For example, a new model and its accuracy on testing is at 93%, while the old model's was 92%. Is the new one really better? Or did you just get lucky with the specific data in your test set? A hypothesis test can tell you if that 1% improvement is statistically significant or likely due to random chance. This helps you decide whether to spend time and money deploying the new model.

- **Confidence Intervals**: A guess with a safety net, instead of stating your result as a single number, you give a range that you're pretty sure contains the true value.

  For example, a 93% is okay. But it's much better to say: "We are 95% confident that our model's true accuracy is somewhere between 91% and 95%." A very wide interval (e.g., [70%, 99%]) signals that your estimate isn't very reliable.

10.3 Using Statistics to Validate Your Model

Statistics aren't just for analyzing data; they are critical for validating the entire machine learning pipeline. Many machine learning models are built on statistical rules. If your data breaks those rules, your model's conclusions might be invalid.

- **Validating Model Assumptions**: Many ML models have statistical assumptions. For example, Linear Regression assumes that the prediction errors (residuals) are normally distributed (i.e., they form a bell curve shape). You can use statistical tests to check this assumption. If the assumption is violated, your model's output might be untrustworthy.
- **Validating Model Outputs**: Is your new model's 91% accuracy truly better than the old model's 90%? Or could that 1% difference be due to random luck in your test data? Hypothesis testing can answer this. You can test if the difference in performance is **statistically significant**, giving you confidence that your new model is genuinely an improvement.

Module 10: Learning Materials & References

a) Standard Deviation and Variance Video
b) Numpy.mean (np.mean, np.var, and np.std)
c) scipy.stats.shapiro (for the Shapiro-Wilk test)
d) scipy.stats.t.interval (for confidence intervals)

Assessment: Statistics & Inference for Data Accuracy

Part A: Quiz

10.1a You have developed two models to predict stock prices. Both models have a mean error of almost $0 (meaning they aren't biased high or low). However, Model A's errors have a standard deviation of $2, while Model B's errors have a standard deviation of $10. What does this tell you?

A) Model A is biased, while Model B is not.
B) Model A is more consistent and reliable in its predictions than Model B.
C) Model B will always be less accurate than Model A.
D) The mean error is the only important metric, so both models are equally good.

10.2a A data scientist trains a new version of a spam detection model. The new model gets 96% accuracy on a test set, while the old model got 95%. Why would the data scientist use a hypothesis test?

A) To calculate the exact average error of the new model.
B) To determine if the 1% improvement is a real, significant improvement or just due to random luck in the test data.
C) To check if the test data follows a bell curve (normal distribution).
D) To create a safety net or range for the model's true accuracy.

10.3a A data scientist reports that their new model has an average accuracy of 90%. Why is this single number potentially misleading without also providing a measure of spread, like a standard deviation or a confidence interval?

A) Because the average accuracy does not tell us how fast the model makes predictions.
B) Because an average accuracy of 90% is not good enough for deployment.
C) Because we don't know if the model's performance is consistent, or if it gets 99% accuracy on some data and 81% on others.
D) Because the model might have a negative mean error.

10.4a A final report states, "We are 95% confident that our model's true accuracy is between 90% and 94%." What is this range called?
A) A p-value
B) A standard deviation
C) A confidence interval
D) A null hypothesis

10.5a Imagine you are analyzing a dataset of daily temperatures in San Diego for the month of July. If you calculate the variance and find that it is very close to zero, what does this tell you about the weather?
A) The average temperature was very low.
B) The temperatures were random and unpredictable.
C) The data contains errors.
D) The temperature was nearly the same every single day.

10.6a You are choosing a machine learning model to predict the arrival time of a city bus. After testing two models, you find:

- Model 1: Mean error = -1 minute (usually predicts 1 min early), Standard Deviation = 5 minutes.
- Model 2: Mean error = -1 minute (usually predicts 1 min early), Standard Deviation = 1 minute.

Which model would you choose for the transit app and why?

A) Model 1, because a larger standard deviation means it can handle more types of traffic situations.

B) Model 2, because its smaller standard deviation means its predictions are much more consistent and reliable.

C) Both models are equally good because their mean error is the same.

&D) Neither, because a model should always have a mean error of exactly zero.

Part C: Python Exercises

10.1c Model Error Analysis Tool

Write a Python script that calculates and interprets the descriptive statistics for a list of model prediction errors by calculating central tendency and spread.

Instructions:

1. Create a Python script named analyze_errors.py.
2. Inside the script, define a list of numbers representing model errors.
3. Use the numpy library to calculate the **mean**, **variance**, and **standard deviation** of the errors.
4. Output each of these values with a clear label.
5. Output interpretation

Starter Code:

```
import numpy as np

# 1. A list representing your model's prediction errors
model_errors = [0.2, -1.5, 0.8, 2.9, -0.1, -0.5, 1.1, 0.0, -2.2, 0.4]
# 2. Calculate the mean, variance, and standard deviation
#    (Your code goes here)
# 3. Print the results
#    (Your code goes here)
# 4. Bonus: Print an interpretation
#    (Your code goes here)
```

Example Output:

--- Model Error Analysis ---
Mean Error: 0.11
Variance: 2.30
Standard Deviation: 1.52

Analysis: Model appears to have low bias.

10.2c Statistical Validation Reporter

Write a Python script to perform a more advanced statistical check on a dataset, using hypothesis testing (normality) and calculating confidence intervals.

Instructions:

1) Create a Python script named validate_data.py.
2) Use the same list of model errors from the first assignment.
3) Use the scipy.stats library to:
4) Perform a **Shapiro-Wilk test** to check if the errors are normally distributed.
5) Calculate the **95% confidence interval** for the mean of the errors.
6) Print the results clearly:
7) The p-value from the normality test and an interpretation (e.g., "Data looks normal" or "Data does not look normal").
8) The calculated confidence interval.
9) Explaining what the confidence interval result means in plain language.

Starter Code:

```
import numpy as np
from scipy import stats
# A list representing your model's prediction errors
model_errors = [0.2, -1.5, 0.8, 2.9, -0.1, -0.5, 1.1, 0.0, -2.2, 0.4]
confidence_level = 0.95

# 1. Perform the Shapiro-Wilk test for normality
#    (Your code goes here)
# 2. Calculate the 95% confidence interval for the mean
#    Hint: stats.t.interval() is a great function for this.
#    You'll need the mean, standard error (std_dev / sqrt(n)), and degrees of
freedom (n-1).
#    (Your code goes here)
# 3. Print the results
#    (Your code goes here)
```