Ai Ml (4351601) - Problem Solver

Antigravity AI

June 4, 2026

Ai Ml

First Edition: 2026

Author: Antigravity AI
Website: milav.in
YouTube: @milav.dabgar

To my students,
who constantly inspire me to find new analogies,
break down complex theories, and make learning
an engineered science.

Preface

About this Book

Welcome to **Computer Networks**! This textbook is meticulously designed to serve as a comprehensive problem solver and study companion. It provides a robust and intuitive foundation in the core principles of this subject, bridging the gap between theoretical models and practical applications. A unique feature of this book is its focus on decoding complex concepts using clear, step-by-step methodologies and real-world analogies.

Target Audience

This book is intended for students and professionals looking to master the concepts of Computer Networks. Whether you are revising for exams, seeking to solidify your fundamental understanding, or looking for practical examples to clarify abstract concepts, this book has been structured to support your learning journey. It serves as an accessible guide designed to remove the fear of complex topics, replacing it with an appreciation for the underlying mechanics.

Scope and Coverage

The coverage is divided logically to follow standard academic curriculums. Each chapter focuses on key topics, breaking them down into digestible sections:

- **Theoretical Insights** – Core concepts explained intuitively.
- **Method Blocks** – Standardized, step-by-step procedures for solving specific types of problems.
- **Worked Examples** – Fully solved problems that apply the methods in practice.

We hope this resource proves invaluable in your studies and helps you achieve excellence in **Computer Networks**. Happy learning!

Acknowledgments

Writing a comprehensive book that connects the fundamental forces of Chemistry with practical engineering applications requires more than just academic knowledge—it requires a community of support. I would like to express my sincere gratitude to everyone who contributed to the realization of this project.

Specially, I extend my heartfelt gratitude to the **Department of Technical Education (DTE), Government of Gujarat**, and **Government Polytechnic, Palanpur**, for providing an environment that fosters innovation, academic rigor, and continuous professional development.

I am deeply thankful to my family for their unwavering patience and support during the countless hours spent researching, formatting, and refining these chapters.

Finally, my deepest appreciation goes to my students. Your curiosity, challenging questions, and continuous feedback are the true driving force behind the unique analogies and streamlined structure of this book. This textbook was engineered for you.

Antigravity AI

About the Author

Milav Jayeshkumar Dabgar is an engineering educator and R&D professional with over 9 years of experience in electronics hardware development, embedded systems, AI/ML, and full-stack software engineering. He currently serves as a **Lecturer (Class - II) & Innovation Leader** at Government Polytechnic, Palanpur, under the Education Department of the Government of Gujarat.

Milav holds a Master of Engineering (ME) in Communication Systems from L.D. College of Engineering and a Bachelor of Engineering (BE) in Electronics & Communication. He is also currently pursuing a **BS in Data Science and Applications from IIT Madras**, where he has completed both the **Diploma in Programming** and the **Diploma in Data Science** with distinction.

Most recently, he completed the **AICTE QIP PG Certificate Program in Deep Learning from SVNIT Surat** with a **perfect 10/10 CGPA**, topping his batch.

A dedicated mentor, Milav has guided numerous student teams in securing over **INR 45 lakhs** in innovation funding, including INR 25 lakhs from Shark Tank India and INR 20 lakhs through iHub Gujarat, resulting in two student patents. He is recognized as an **NPTEL Evangelist** and **NPTEL Discipline Star** in Computer Science, reflecting his commitment to continuous learning and academic excellence.

His technical expertise spans from embedded systems (8051, STM32, Arduino) to modern web technologies (Next.js, Python, PostgreSQL) and Data Science (TensorFlow, PyTorch). Through this textbook, he aims to simplify complex Engineering Chemistry concepts by integrating relatable, real-world analogies drawn from modern tech and engineering landscapes.

Milav is also an active content creator, running a popular **YouTube channel** (@milav.dabgar) where he shares technical tutorials and academic resources. He maintains a personal blog and resource hub at milav.in and can be reached for professional collaborations on LinkedIn.

Contents

List of Figures

List of Tables

CHAPTER 1

Fundamentals of Artificial Intelligence

1.1 Describe Basic Concept of Artificial Intelligence

The core computational concept in Artificial Intelligence (AI) is designing agents that receive percepts from an environment and perform actions to maximize a performance measure. A fundamental technique is formulating real-world tasks as state-space search problems.

Methodology:

Problem Formulation for Intelligent Agents To computationally define a basic AI problem, specify the following five components:

1. **Initial State**: The starting configuration of the agent and environment.
2. **Actions**: The set of possible actions available to the agent at a given state.
3. **Transition Model**: A description of what each action does. It returns the new state resulting from performing an action in the current state.
4. **Goal Test**: A condition that determines whether a given state is a target (goal) state.
5. **Path Cost**: A function that assigns a numeric cost to a sequence of actions. Often, step costs are additive.

Worked Example:

Formulating the Vacuum-Cleaner World Problem **Problem Statement:** Consider a simple vacuum-cleaner agent in an environment consisting of two squares A and B. Each square can be either dirty or clean. The agent can move Left, move Right or Suck the dirt. Formulate this scenario as a computational search problem.
Solution:

1. **Initial State:** Any of the 8 possible states. A state represents the agent's location (A or B) and the dirt status of both squares (e.g. Agent at A, A is dirty, B is clean).
2. **Actions:** In any state, the agent can choose from three actions: $\{\text{Left}, \text{Right}, \text{Suck}\}$.
3. **Transition Model:**
   - Action Left moves the agent to square A (if in B, else no change).
   - Action Right moves the agent to square B (if in A, else no change).
   - Action Suck changes the current square from dirty to clean (if dirty, else no change).
4. **Goal Test:** Checks whether both squares A and B are clean.
5. **Path Cost:** Each action (Left, Right or Suck) costs 1 unit. Total path cost is the number of steps taken.

1.2 Discuss Areas of Artificial Intelligence

AI encompasses various specialized areas based on the type of data processed and the problem domain. Computationally, we can classify problems into these sub-domains to choose the appropriate algorithms.

Methodology:

Classifying Problems into AI Sub-domains To map a computational problem to its corresponding AI area, follow these steps:

1. **Analyze the Input Data**: Determine if the input consists of numerical data, text, audio, images or real-time sensor streams.
2. **Analyze the Desired Output**: Determine if the system needs to predict a value, translate language, identify objects or perform physical manipulation.
3. **Map to Domain**:
   - **Machine Learning**: Input is structured or unstructured data; Output is predictions, patterns or classifications based on learned historical data.
   - **Natural Language Processing**: Input is text or speech; Output is translated text, sentiment score or conversational response.
   - **Computer Vision**: Input is images or video frames; Output is object detection, image classification or scene understanding.
   - **Robotics**: Input is physical sensor data; Output is motor control and physical environment manipulation.
   - **Expert Systems**: Input is a set of facts or user queries; Output is logical inferences drawn from a knowledge base using rules.

Worked Example:

Identifying AI Domains for Real-World Scenarios **Problem Statement:** Categorize the following computational tasks into their primary AI sub-domains: 1. Extracting the names of people and organizations from a legal document. 2. Predicting housing prices based on square footage and number of bedrooms. 3. Controlling a mechanical arm to assemble car parts on a factory line. 4. Diagnosing a bacterial infection based on a structured set of IF-THEN medical rules. 5. Detecting the presence of a tumor in an MRI scan. **Solution:**

1. **Extracting names from text**: The input is text and the task involves understanding language syntax and semantics.
   Domain: Natural Language Processing.
2. **Predicting housing prices**: The input is numerical data and the task requires learning a predictive mapping from historical data.
   Domain: Machine Learning.
3. **Controlling a mechanical arm**: The input is spatial sensors and the output involves physical actuator control in a dynamic environment.
   Domain: Robotics.
4. **Diagnosing based on rules**: The system uses an explicit knowledge base and an inference engine to derive conclusions.
   Domain: Expert Systems.
5. **Detecting a tumor**: The input is image data and the output is identifying a region of interest.
   Domain: Computer Vision.

1.3 Identify Various Applications of Artificial Intelligence

To practically apply AI to real-world applications, systems must be formally defined using the PEAS framework before algorithmic development begins.

Methodology:

Designing PEAS Specifications for AI Applications To design the operational framework of an AI application, define the PEAS parameters:

1. **Performance Measure**: The quantifiable metric used to evaluate how well the AI application achieves its objective (e.g. accuracy, speed, profit, safety).
2. **Environment**: The external conditions and entities the AI system interacts with. Is it fully observable or partially observable? Static or dynamic?
3. **Actuators**: The hardware or software mechanisms through which the AI system executes actions and changes the environment.
4. **Sensors**: The hardware or software mechanisms through which the AI system receives input and perceives the environment.

Worked Example:

PEAS Specification for an Automated Medical Diagnosis Application **Problem Statement:** Define the PEAS specification for an AI application designed to assist doctors by providing preliminary diagnoses based on patient symptoms and test results.
Solution:

1. **Performance Measure**: High diagnostic accuracy, minimizing false negatives (missing a disease), minimizing false positives, reducing the time required to reach a diagnosis.
2. **Environment**: Patients, hospital databases, laboratory testing facilities and medical staff. The environment is partially observable (internal diseases are not directly visible) and dynamic (patient health can change).
3. **Actuators**: Software interfaces displaying the predicted diagnosis, alerts sent to medical staff and automated orders for further laboratory tests.
4. **Sensors**: Keyboard or voice input for symptom entry, direct data feeds from electronic health records and digital medical imaging (X-rays, MRIs).

Worked Example:

PEAS Specification for an Autonomous Delivery Drone **Problem Statement:** Formulate the PEAS description for an AI-powered drone application whose purpose is to deliver small packages to residential addresses.
Solution:

1. **Performance Measure**: Correct delivery location, zero damage to the package or drone, minimized flight time, adherence to legal flight zones and safety to humans and property.
2. **Environment**: Open airspace, varied weather conditions (wind, rain), residential neighborhoods, physical obstacles (trees, power lines, birds).
3. **Actuators**: Propeller motors for lift and direction, package release mechanism and communication transmitters for status updates.
4. **Sensors**: GPS for location tracking, cameras for visual navigation and obstacle avoidance, altimeter for height and gyroscopes or accelerometers for stability.

CHAPTER 2

Foundation of Machine Learning

2.1 Understand Well Posed Learning Problem

Methodology:

Formulating a Well Posed Learning Problem A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T as measured by P improves with experience E.

1. **Identify the Task (T)**: What is the system expected to do? (e.g. classify emails or steer a car).
2. **Identify the Performance Measure (P)**: How will we evaluate the success? (e.g. percentage of correctly classified emails or distance traveled without crashing).
3. **Identify the Training Experience (E)**: What data or interaction will the system learn from? (e.g. a database of labeled emails or a simulated driving environment).

Worked Example:

Spam Filter Formulation **Problem:** Formulate a well-posed learning problem for an Email Spam Filter system.
Solution:

1. **Task (T)**: Classifying incoming emails as "spam" or "not spam" (legitimate).
2. **Performance Measure (P)**: The percentage of emails that are correctly classified by the system.
3. **Training Experience (E)**: A dataset consisting of historical emails that have been manually labeled as "spam" or "not spam" by human users.

Worked Example:

Autonomous Driving Formulation **Problem:** Formulate a well-posed learning problem for a self-driving car algorithm that learns to stay within the lane.
Solution:

1. **Task (T)**: Steering the vehicle to keep it within the boundaries of a driving lane on a highway.
2. **Performance Measure (P)**: The average distance traveled before the vehicle accidentally crosses the lane marker.
3. **Training Experience (E)**: A sequence of images and steering commands recorded while a human driver safely operates the vehicle.

2.2 Describe Different Types of Machine Learning

Methodology:

Classifying Machine Learning Algorithms Machine learning algorithms are broadly categorized into three main types based on the nature of the learning signal or feedback available to the system.

1. **Supervised Learning**: The algorithm is trained on a labeled dataset. The target variable (output) is known. Used for Classification and Regression.
2. **Unsupervised Learning**: The algorithm is provided with data that has no labels. The goal is to infer the natural structure present within a set of data points. Used for Clustering and Association.
3. **Reinforcement Learning**: The algorithm learns to perform an action from experience. It receives rewards or penalties for the actions it performs.

Worked Example:

Algorithm Selection for Datasets **Problem:** Given the following three datasets determine the appropriate type of machine learning required. 1. A dataset containing house sizes and their corresponding sale prices. 2. A dataset containing customer transaction histories without any predefined groups. 3. An agent learning to play chess by playing millions of games against itself.
Solution:

1. **Dataset 1**: Supervised Learning. The sale price acts as the labeled output (target variable). This is a regression problem.
2. **Dataset 2**: Unsupervised Learning. There are no predefined labels or groups. The goal is to discover hidden patterns (Clustering customers into segments).
3. **Dataset 3**: Reinforcement Learning. The agent learns from a sequence of actions resulting in a delayed reward (winning or losing the game).

Methodology:

K Means Clustering Algorithm Steps K-Means is a popular computational algorithm for Unsupervised Learning.

1. Choose the number of clusters K.
2. Initialize K cluster centroids randomly.
3. **Assignment Step**: Assign each data point to the closest cluster centroid based on Euclidean distance.
4. **Update Step**: Recompute the centroids by taking the mean of all data points assigned to that cluster.
5. Repeat the assignment and update steps until convergence (centroids no longer change).

Worked Example:

One Dimensional K Means Clustering **Problem:** Given 1D data points $X = \{2, 3, 4, 10, 11, 12, 20, 25, 30\}$ apply K-Means clustering with $K = 2$. Initial centroids are $c_1 = 3$ and $c_2 = 11$. Perform one iteration.
Solution: Step 1: Assignment
Calculate the distance from each point to $c_1 = 3$ and $c_2 = 11$.

- $x = 2$: $|2 - 3| = 1 \quad |2 - 11| = 9 \implies$ assigned to c_1
- $x = 3$: $|3 - 3| = 0 \quad |3 - 11| = 8 \implies$ assigned to c_1
- $x = 4$: $|4 - 3| = 1 \quad |4 - 11| = 7 \implies$ assigned to c_1

- $x = 10$: $|10 - 3| = 7 \quad |10 - 11| = 1 \implies$ assigned to c_2
- $x = 11$: $|11 - 3| = 8 \quad |11 - 11| = 0 \implies$ assigned to c_2
- $x = 12$: $|12 - 3| = 9 \quad |12 - 11| = 1 \implies$ assigned to c_2
- $x = 20$: $|20 - 3| = 17 \quad |20 - 11| = 9 \implies$ assigned to c_2
- $x = 25$: $|25 - 3| = 22 \quad |25 - 11| = 14 \implies$ assigned to c_2
- $x = 30$: $|30 - 3| = 27 \quad |30 - 11| = 19 \implies$ assigned to c_2

Clusters formed:
Cluster 1: $\{2, 3, 4\}$
Cluster 2: $\{10, 11, 12, 20, 25, 30\}$
Step 2: Update Centroids

- New $c_1 = (2 + 3 + 4)/3 = 9/3 = 3$
- New $c_2 = (10 + 11 + 12 + 20 + 25 + 30)/6 = 108/6 = 18$

After one iteration the centroids are $c_1 = 3$ and $c_2 = 18$.

2.3 Explain Basic Concepts of Reinforcement Learning

Methodology:

Reinforcement Learning Problem Formulation A Reinforcement Learning (RL) system consists of an Agent interacting with an Environment.

1. **State** (S): The current situation or configuration of the environment.
2. **Action** (A): The set of all possible moves the agent can make.
3. **Reward** (R): The feedback signal from the environment after an action is taken. The agent's goal is to maximize cumulative reward.
4. **Policy** (π): The strategy that the agent employs to determine the next action based on the current state.

Worked Example:

Grid World Formulation **Problem:** Formulate a simple Grid World problem where a robot needs to find a battery on a 3×3 grid without falling into a trap.
Solution:

1. **States** (S): The 9 distinct coordinates (x, y) on the 3×3 grid where the robot can be located.
2. **Actions** (A): Move Up, Move Down, Move Left, Move Right.
3. **Rewards** (R):
   - $+10$ for reaching the cell with the battery.
   - -10 for stepping into the cell with the trap.
   - -1 for every step taken (to encourage finding the shortest path).
4. **Policy** (π): A mapping that tells the robot which direction to move for every given cell in the grid to reach the battery optimally.

Methodology:

Q Learning Update Rule Q-Learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. The Q-value $Q(s, a)$ is updated using the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Where:

- s_t a_t: Current state and action.
- R_{t+1}: Immediate reward received.
- s_{t+1}: Next state.
- α: Learning rate $(0 < \alpha \leq 1)$.
- γ: Discount factor for future rewards $(0 \leq \gamma \leq 1)$.
- $\max_a Q(s_{t+1}, a)$: Maximum expected future reward given the new state.

Worked Example:

Q Value Calculation **Problem:** Given the current state S_1 and action A_1 the current Q-value is $Q(S_1, A_1) = 5.0$. The agent takes action A_1 receives a reward $R = 10$ and transitions to state S_2. In state S_2 the possible actions have Q-values: $Q(S_2, A_{left}) = 2.0$, $Q(S_2, A_{right}) = 8.0$. Calculate the updated $Q(S_1, A_1)$ assuming a learning rate $\alpha = 0.1$ and a discount factor $\gamma = 0.9$.
Solution: Step 1: Identify known values.

- Current $Q(S_1, A_1) = 5.0$
- Reward $R = 10$
- $\max_a Q(S_2, a) = \max(2.0, 8.0) = 8.0$
- $\alpha = 0.1$
- $\gamma = 0.9$

Step 2: Apply the Q-learning update formula.

$$Q(S_1, A_1) = 5.0 + 0.1 \times [10 + 0.9 \times 8.0 - 5.0]$$

Step 3: Calculate the temporal difference target and error.

$$Target = 10 + 7.2 = 17.2$$

$$Error = 17.2 - 5.0 = 12.2$$

Step 4: Compute final updated Q-value.

$$Q(S_1, A_1) = 5.0 + 0.1 \times (12.2) = 5.0 + 1.22 = 6.22$$

The updated Q-value for taking action A_1 in state S_1 is 6.22.

2.4 Compare Types of Machine Learning

Methodology:

Evaluation Strategy for Comparing ML Types When determining the best machine learning paradigm for a given problem use a systematic feature comparison based on input data, output data, and the nature of the feedback loop.

1. **Input Data Presence**: Does the dataset consist of historical input features?
2. **Label Availability**: Are explicit target variables provided for the training data? (Yes → Supervised, No → Unsupervised, Action-based → Reinforcement).
3. **Goal**: Is the goal to predict a specific value (Supervised) discover hidden structures (Unsupervised) or maximize long-term reward (Reinforcement)?
4. **Feedback Mechanism**: Direct feedback (Supervised) no feedback (Unsupervised) or delayed reward feedback (Reinforcement).

Worked Example:

Comparative Matrix Formulation **Problem:** Construct a comparative matrix for Supervised Unsupervised and Reinforcement Learning across three dimensions: Data Type, Core Objective, and Example Algorithm. **Solution:** We can build a direct comparison array for the three paradigms.

Feature	**Supervised Learning**	**Unsupervised Learning**	**Reinforcement Learning**
Data Type	Labeled Data	Unlabeled Data	State Action Pairs
Core Objective	Predict outcome	Find hidden structures	Maximize total reward
Feedback	Direct evaluation	No direct evaluation	Delayed reward
Example Algorithm	Linear Regression	K-Means Clustering	Q-Learning

CHAPTER 3

Foundation of Neural Networks

3.1 Understand Foundation of Neural Networks

An Artificial Neural Network (ANN) consists of interconnected artificial neurons. Each neuron receives inputs, multiplies them by corresponding weights, adds a bias, and passes the result through an activation function to produce an output.

Methodology:

Single Neuron Computation To compute the output of a single artificial neuron:

1. Identify the inputs $x_1, x_2, \ldots, x_n$.
2. Identify the corresponding weights $w_1, w_2, \ldots, w_n$ and the bias b.
3. Calculate the net input z using the formula:

$$z = \sum_{i=1}^{n} (x_i \cdot w_i) + b = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + b$$

4. Apply the activation function $f(z)$ to compute the final output y:

$$y = f(z)$$

Worked Example:

Calculating the Output of a Single Neuron A single neuron receives three inputs: $x_1 = 0.5$, $x_2 = -0.2$ and $x_3 = 0.1$. The corresponding weights are $w_1 = 0.4$, $w_2 = 0.7$ and $w_3 = -0.5$. The bias is $b = 0.2$. Assuming the activation function is a simple linear function $f(z) = z$, compute the output of the neuron.

Step 1: Identify inputs and weights.

- Inputs: $x_1 = 0.5$, $x_2 = -0.2$, $x_3 = 0.1$
- Weights: $w_1 = 0.4$, $w_2 = 0.7$, $w_3 = -0.5$
- Bias: $b = 0.2$

Step 2: Calculate the net input z.

$$z = (x_1 \cdot w_1) + (x_2 \cdot w_2) + (x_3 \cdot w_3) + b$$

$$z = (0.5 \cdot 0.4) + (-0.2 \cdot 0.7) + (0.1 \cdot -0.5) + 0.2$$

$$z = 0.20 - 0.14 - 0.05 + 0.2$$

$$z = 0.21$$

Step 3: Apply the activation function. Since $f(z) = z$:

$$y = f(0.21) = 0.21$$

The output of the neuron is 0.21.

3.2 Implement Activation Functions

Activation functions introduce non-linearity into the network allowing it to learn complex patterns.

Methodology:

Common Activation Functions Given the net input z:

1. **Binary Step Function**: Outputs 1 if z is positive or zero; otherwise 0.

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

2. **Sigmoid Function**: Outputs a value between 0 and 1.

$$f(z) = \frac{1}{1 + e^{-z}}$$

3. **Hyperbolic Tangent Function (Tanh)**: Outputs a value between -1 and 1.

$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

4. **Rectified Linear Unit Function (ReLU)**: Outputs z if positive; otherwise 0.

$$f(z) = \max(0, z)$$

Worked Example:

Computing Different Activation Outputs Given a neuron with a net input $z = 1.5$. Calculate the output of this neuron if the activation function used is: (a) Binary Step (b) Sigmoid (c) ReLU (d) Tanh. Use $e^{-1.5} \approx 0.223$ and $e^{1.5} \approx 4.482$.

Step 1: Compute for Binary Step Function. Since $z = 1.5 \geq 0$:

$$f(1.5) = 1$$

Step 2: Compute for Sigmoid Function.

$$f(1.5) = \frac{1}{1 + e^{-1.5}} \approx \frac{1}{1 + 0.223} = \frac{1}{1.223} \approx 0.817$$

Step 3: Compute for ReLU Function.

$$f(1.5) = \max(0, 1.5) = 1.5$$

Step 4: Compute for Tanh Function.

$$f(1.5) = \frac{e^{1.5} - e^{-1.5}}{e^{1.5} + e^{-1.5}} \approx \frac{4.482 - 0.223}{4.482 + 0.223} = \frac{4.259}{4.705} \approx 0.905$$

3.3 Implement Types of ANN

Artificial Neural Networks can be categorized by their architecture. The most fundamental computational task is the forward propagation of inputs through layers in a Feedforward Neural Network.

Methodology:

Forward Pass in a Feedforward Network To compute the outputs of a multi-layer feedforward network:

1. Set the input layer activations $a^{(0)}$ to the given input values X.
2. For each hidden and output layer l from 1 to L:
   (a) Compute the net input for each neuron j in layer l:

$$z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$

   where $w_{ji}^{(l)}$ is the weight from neuron i in layer $l-1$ to neuron j in layer l and $b_j^{(l)}$ is the bias.
   (b) Apply the activation function to get the activation $a_j^{(l)}$:

$$a_j^{(l)} = f(z_j^{(l)})$$

3. The final output of the network is the activation of the output layer $a^{(L)}$.

Worked Example:

Multi Layer Perceptron Forward Pass Consider an MLP with 2 inputs, 1 hidden layer with 2 neurons and 1 output neuron. Inputs: $x_1 = 1$, $x_2 = 0$.
Hidden layer weights: $w_{11}^{(1)} = 0.1$, $w_{12}^{(1)} = 0.2$ (to neuron 1); $w_{21}^{(1)} = 0.3$, $w_{22}^{(1)} = 0.4$ (to neuron 2). Biases: $b_1^{(1)} = 0.1$, $b_2^{(1)} = 0.2$.
Output layer weights: $w_{11}^{(2)} = 0.5$, $w_{12}^{(2)} = 0.6$. Bias: $b_1^{(2)} = 0.3$.
Activation function for all neurons is the Binary Step Function. Compute the output.
Step 1: Compute hidden layer net inputs.
For hidden neuron 1:

$$z_1^{(1)} = (x_1 \cdot w_{11}^{(1)}) + (x_2 \cdot w_{12}^{(1)}) + b_1^{(1)} = (1 \cdot 0.1) + (0 \cdot 0.2) + 0.1 = 0.2$$

For hidden neuron 2:

$$z_2^{(1)} = (x_1 \cdot w_{21}^{(1)}) + (x_2 \cdot w_{22}^{(1)}) + b_2^{(1)} = (1 \cdot 0.3) + (0 \cdot 0.4) + 0.2 = 0.5$$

Step 2: Apply activation to hidden layer.
Since $z_1^{(1)} = 0.2 \geq 0$, $a_1^{(1)} = 1$.
Since $z_2^{(1)} = 0.5 \geq 0$, $a_2^{(1)} = 1$.
Step 3: Compute output layer net input.

$$z_1^{(2)} = (a_1^{(1)} \cdot w_{11}^{(2)}) + (a_2^{(1)} \cdot w_{12}^{(2)}) + b_1^{(2)} = (1 \cdot 0.5) + (1 \cdot 0.6) + 0.3 = 1.4$$

Step 4: Apply activation to output layer.
Since $z_1^{(2)} = 1.4 \geq 0$, output $y = a_1^{(2)} = 1$.

3.4 Describe Weight Learning Process

The weight learning process adjusts the parameters (weights and biases) of a neural network to minimize the error between the predicted output and the actual target output.

Methodology:

Perceptron Learning Rule Used for binary classification tasks with a step activation function.

1. Initialize weights w_i and bias b to small random values or zeros.

2. For each training example (X, y_{target}):
   (a) Compute the predicted output $y_{\text{pred}} = f(\sum x_i w_i + b)$.
   (b) Calculate the error: $e = y_{\text{target}} - y_{\text{pred}}$.
   (c) Update each weight: $w_i^{(\text{new})} = w_i^{(\text{old})} + \alpha \cdot e \cdot x_i$, where α is the learning rate.
   (d) Update the bias: $b^{(\text{new})} = b^{(\text{old})} + \alpha \cdot e$.
3. Repeat step 2 until the error is zero for all examples or a maximum number of epochs is reached.

Worked Example:

Perceptron Weight Update with Error Given inputs $x_1 = 1$, $x_2 = 1$ and target output $y_{\text{target}} = 1$. Initial weights $w_1 = -0.2$, $w_2 = -0.4$ and bias $b = 0.1$. Learning rate $\alpha = 0.5$. Activation is a Binary Step Function.

Step 1: Compute predicted output.

$$z = (1 \cdot -0.2) + (1 \cdot -0.4) + 0.1 = -0.2 - 0.4 + 0.1 = -0.5$$

$$y_{\text{pred}} = f(-0.5) = 0 \quad (\text{since } -0.5 < 0)$$

Step 2: Calculate error.

$$e = y_{\text{target}} - y_{\text{pred}} = 1 - 0 = 1$$

Step 3: Update weights and bias.

$$w_1^{(\text{new})} = w_1^{(\text{old})} + \alpha \cdot e \cdot x_1 = -0.2 + 0.5 \cdot 1 \cdot 1 = -0.2 + 0.5 = 0.3$$

$$w_2^{(\text{new})} = w_2^{(\text{old})} + \alpha \cdot e \cdot x_2 = -0.4 + 0.5 \cdot 1 \cdot 1 = -0.4 + 0.5 = 0.1$$

$$b^{(\text{new})} = b^{(\text{old})} + \alpha \cdot e = 0.1 + 0.5 \cdot 1 = 0.6$$

The new weights are $w_1 = 0.3$, $w_2 = 0.1$ and the new bias is $b = 0.6$.

Methodology:

Delta Rule Weight Update Used with continuous activation functions to minimize Mean Squared Error using Gradient Descent.

1. Compute the net input $z = \sum x_i w_i + b$ and predicted output $y_{\text{pred}} = f(z)$.
2. Calculate the error $e = y_{\text{target}} - y_{\text{pred}}$.
3. Calculate the derivative of the activation function at z denoted as $f'(z)$.
4. Update each weight: $w_i^{(\text{new})} = w_i^{(\text{old})} + \alpha \cdot e \cdot f'(z) \cdot x_i$.
5. Update the bias: $b^{(\text{new})} = b^{(\text{old})} + \alpha \cdot e \cdot f'(z)$.

Worked Example:

Weight Update using Delta Rule Given inputs $x_1 = 0.5$, $x_2 = 1.0$. Target output $y_{\text{target}} = 1$. Initial weights $w_1 = 0.4$, $w_2 = 0.6$, bias $b = -0.2$. Learning rate $\alpha = 0.5$. The activation function is a linear function $f(z) = z$, so its derivative $f'(z) = 1$. Perform one weight update iteration.

Step 1: Compute predicted output.

$$z = (0.5 \cdot 0.4) + (1.0 \cdot 0.6) - 0.2 = 0.2 + 0.6 - 0.2 = 0.6$$

$$y_{\text{pred}} = f(0.6) = 0.6$$

Step 2: Calculate error.

$$e = y_{\text{target}} - y_{\text{pred}} = 1 - 0.6 = 0.4$$

Step 3: Determine the derivative. Since $f(z) = z$, the derivative $f'(z) = 1$.

Step 4: Update weights and bias.

$$w_1^{(\text{new})} = w_1^{(\text{old})} + \alpha \cdot e \cdot f'(z) \cdot x_1 = 0.4 + 0.5 \cdot 0.4 \cdot 1 \cdot 0.5 = 0.4 + 0.1 = 0.5$$

$$w_2^{(\text{new})} = w_2^{(\text{old})} + \alpha \cdot e \cdot f'(z) \cdot x_2 = 0.6 + 0.5 \cdot 0.4 \cdot 1 \cdot 1.0 = 0.6 + 0.2 = 0.8$$

$$b^{(\text{new})} = b^{(\text{old})} + \alpha \cdot e \cdot f'(z) = -0.2 + 0.5 \cdot 0.4 \cdot 1 = -0.2 + 0.2 = 0.0$$

The updated weights are $w_1 = 0.5$, $w_2 = 0.8$, and the updated bias is $b = 0.0$.

CHAPTER 4

Foundation of Natural Language Processing

4.1 Understand Basics of NLP

Natural Language Processing (NLP) is a branch of Artificial Intelligence concerned with enabling computers to understand, interpret, and generate human language. In computational applications, NLP relies heavily on transforming unstructured text into structured numerical formats that machine learning algorithms can process.

4.2 Discuss Components of NLP

NLP consists of two primary computational components:

1. **Natural Language Understanding (NLU)**: Maps given input in natural language into useful representations. It involves analyzing text to extract meaning, intent, and entities.
2. **Natural Language Generation (NLG)**: Produces meaningful phrases and sentences in the form of natural language from some internal representation.

4.3 Explain Phases of NLP

Processing text typically follows a pipeline of computational phases:

1. **Lexical Analysis**: Dividing the whole text into paragraphs, sentences, and words.
2. **Syntactic Analysis (Parsing)**: Checking sentences for grammar rules and creating syntax trees.
3. **Semantic Analysis**: Checking the text for meaningfulness. It maps syntactic structures to objects in the task domain.
4. **Discourse Integration**: Determining the meaning of sentences based on preceding sentences.
5. **Pragmatic Analysis**: Deriving the intended effect or real-world meaning of the language.

4.4 Apply Data Preprocessing Techniques

Raw text must be preprocessed before it can be fed into machine learning models. The following sections outline standard computational methods used to clean and encode text.

4.4.1 Tokenization and Stop Word Removal

Methodology:

Tokenization and Stop Word Removal **Tokenization** breaks raw text into atomic units called tokens (words or sentences). **Stop Word Removal** filters out common words that carry little semantic value (e.g. "a" "an" "the" "is").

1. Convert the entire input text to lowercase to ensure uniformity.

2. Scan the text and replace punctuation marks with spaces.
3. Split the text string by whitespace to extract a list of tokens.
4. Iterate through the tokens and remove any token that exists in a predefined list of stop words.

Worked Example:

Basic Text Preprocessing **Problem Statement:** Preprocess the following sentence by tokenizing and removing standard English stop words ("is" "the" "and" "in" "of" "a"). Sentence: "The Quick brown fox is jumping over the lazy dog in a field."
Solution:

1. **Lowercase conversion**: "the quick brown fox is jumping over the lazy dog in a field."
2. **Remove punctuation**: "the quick brown fox is jumping over the lazy dog in a field" (Removed the period at the end).
3. **Tokenization**: ["the", "quick", "brown", "fox", "is", "jumping", "over", "the", "lazy", "dog", "in", "a", "field"]
4. **Stop word removal**: Remove "the", "is", "in", "a".
5. **Final Tokens**: ["quick", "brown", "fox", "jumping", "over", "lazy", "dog", "field"]

4.4.2 Stemming

Methodology:

Word Normalization via Stemming Stemming reduces words to their base or root form by stripping suffixes using heuristic rules.

1. Identify the token to be stemmed.
2. Apply suffix removal rules sequentially (e.g. remove "-ing" "-ed" "-s" "-es" "-ly").
3. If a rule matches the token's suffix, truncate the suffix and return the resulting stem. (Note: The stem may not be a valid dictionary word).

Worked Example:

Applying Stemming Rules **Problem Statement:** Apply basic stemming rules (remove "-ing" "-ed" "-es" "-s") to the following tokens: ["computing", "calculated", "computers", "buses", "intelligent"].
Solution:

1. "computing": Ends in "-ing". Remove "-ing" $\rightarrow$ "comput".
2. "calculated": Ends in "-ed". Remove "-ed" $\rightarrow$ "calculat".
3. "computers": Ends in "-s". Remove "-s" $\rightarrow$ "computer".
4. "buses": Ends in "-es". Remove "-es" $\rightarrow$ "bus".
5. "intelligent": Does not end with any specified suffix. Remains "intelligent".

Final Stemmed Tokens: ["comput", "calculat", "computer", "bus", "intelligent"]

4.4.3 N-Grams Generation

Methodology:

Generating N Grams An N-gram is a contiguous sequence of N items from a given sample of text.

1. Tokenize the text into an ordered list of words $W = [w_1, w_2, \ldots, w_k]$.
2. For a specific value of N, iterate i from 1 to $k - N + 1$.
3. Extract the sublist of words from index i to $i + N - 1$.
4. Group these words into a single N-gram tuple.

Worked Example:

Extracting Bigrams and Trigrams **Problem Statement:** Generate all Bigrams ($N = 2$) and Trigrams ($N = 3$) for the sentence: "Machine learning makes tasks easier".
Solution:

1. **Tokenization**: $W =$ ["Machine", "learning", "makes", "tasks", "easier"]. Total words $k = 5$.
2. **Bigrams ($N = 2$)**: Number of bigrams $= k - 2 + 1 = 4$.
   - $i = 1$: (Machine, learning)
   - $i = 2$: (learning, makes)
   - $i = 3$: (makes, tasks)
   - $i = 4$: (tasks, easier)
3. **Trigrams ($N = 3$)**: Number of trigrams $= k - 3 + 1 = 3$.
   - $i = 1$: (Machine, learning, makes)
   - $i = 2$: (learning, makes, tasks)
   - $i = 3$: (makes, tasks, easier)

4.4.4 Bag of Words Representation

Methodology:

Bag of Words Encoding BoW converts a collection of text documents into a numerical matrix based on word occurrences.

1. Tokenize and preprocess all documents in the corpus.
2. Extract all unique tokens across the entire corpus to form a vocabulary list. Sort it alphabetically or by frequency.
3. For each document, create a vector of the same length as the vocabulary.
4. For each dimension in the vector, count the number of times the corresponding vocabulary word appears in the document.

Worked Example:

Creating a BoW Matrix **Problem Statement:** Construct a Bag of Words representation for the following two documents. Document 1: "Data science is fun" Document 2: "Data analysis is also fun"
Solution:

1. **Tokenize**: Doc 1 Tokens: ["data", "science", "is", "fun"] Doc 2 Tokens: ["data", "analysis", "is", "also", "fun"]
2. **Create Vocabulary**: Find unique words from both documents and sort them alphabetically. Vocabulary = ["also", "analysis", "data", "fun", "is", "science"]
3. **Vectorize Document 1**: Count occurrences of vocabulary words in Doc 1.
   - "also": 0
   - "analysis": 0
   - "data": 1
   - "fun": 1
   - "is": 1
   - "science": 1

   Vector 1: $[0, 0, 1, 1, 1, 1]$
4. **Vectorize Document 2**: Count occurrences of vocabulary words in Doc 2.
   - "also": 1
   - "analysis": 1
   - "data": 1
   - "fun": 1
   - "is": 1
   - "science": 0

   Vector 2: $[1, 1, 1, 1, 1, 0]$

Final Matrix Representation:

Document	also	analysis	data	fun	is	science
Doc 1	0	0	1	1	1	1
Doc 2	1	1	1	1	1	0

4.4.5 Term Frequency Inverse Document Frequency

Methodology:

TF IDF Calculation TF-IDF measures the importance of a term in a document relative to the entire corpus.

1. Calculate the Term Frequency (TF) for a term t in document d:

$$TF(t, d) = \frac{\text{Count of } t \text{ in } d}{\text{Total number of terms in } d}$$

2. Calculate the Inverse Document Frequency (IDF) for term t across the corpus of N total documents:

$$IDF(t) = \log_{10}\left(\frac{N}{DF(t)}\right)$$

   where $DF(t)$ is the number of documents that contain the term t.

3. Compute the TF-IDF weight:

$$\text{TF-IDF}(t, d) = TF(t, d) \times IDF(t)$$

Worked Example:

Computing TF IDF Scores **Problem Statement:** Given a corpus of $N = 2$ documents: Doc 1: "cat dog cat" Doc 2: "dog bird" Calculate the TF-IDF score for the terms "cat" and "dog" in Document 1. Use Base-10 logarithm.

Solution: Step 1: Calculate TF for terms in Document 1

- Total terms in Doc 1 = 3 ("cat", "dog", "cat")
- Frequency of "cat" in Doc 1 = 2
- Frequency of "dog" in Doc 1 = 1
- $TF(\text{cat}, \text{Doc 1}) = \frac{2}{3} \approx 0.667$
- $TF(\text{dog}, \text{Doc 1}) = \frac{1}{3} \approx 0.333$

Step 2: Calculate IDF for "cat" and "dog"

- Total documents $N = 2$
- "cat" appears in 1 document (Doc 1). So, $DF(\text{cat}) = 1$.
- "dog" appears in 2 documents (Doc 1 and Doc 2). So, $DF(\text{dog}) = 2$.
- $IDF(\text{cat}) = \log_{10}\left(\frac{2}{1}\right) = \log_{10}(2) \approx 0.301$
- $IDF(\text{dog}) = \log_{10}\left(\frac{2}{2}\right) = \log_{10}(1) = 0$

Step 3: Calculate TF-IDF for Document 1

- $\text{TF-IDF}(\text{cat}, \text{Doc 1}) = 0.667 \times 0.301 \approx 0.201$
- $\text{TF-IDF}(\text{dog}, \text{Doc 1}) = 0.333 \times 0 = 0$

Conclusion: The term "cat" is assigned a higher weight because it is frequent in Document 1 but rare in the corpus. The term "dog" receives a weight of 0 because it appears in every document, rendering it useless for distinguishing between documents.

CHAPTER 5

Word Embedding in Natural Language Processing

Natural Language Processing (NLP) requires converting text into numerical formats that machine learning algorithms can process. This chapter focuses on the computational techniques used to map words to vectors, ranging from basic frequency-based methods to dense semantic embeddings, and explores their applications.

5.1 Understand and Apply Word Embedding Techniques

5.1.1 One Hot Encoding

The simplest approach to represent words as numbers is One-Hot Encoding. Each word is represented as a binary vector with a single high value (1) and all other values low (0).

Methodology:

One Hot Encoding Algorithm

1. **Build the Vocabulary**: Extract all unique words from the given text corpus to create a vocabulary set V. Let the size of the vocabulary be $N = |V|$.
2. **Assign Indices**: Assign a unique integer index i (where $1 \leq i \leq N$) to each word in the vocabulary.
3. **Create Vectors**: For each word w_i, construct an N-dimensional vector $v \in \mathbb{R}^N$. Set the i-th element of v to 1 and all other elements to 0.

Worked Example:

Generating One Hot Vectors **Problem:** Given the following two sentences, generate the one-hot encoded vectors for each word.

- Sentence 1: "the cat sat"
- Sentence 2: "the dog barked"

Solution:
Step 1: Build the Vocabulary Extract unique words from both sentences and sort them alphabetically (optional but standard). $V = \{\text{barked}, \text{cat}, \text{dog}, \text{sat}, \text{the}\}$ Vocabulary size $N = 5$.
Step 2: Assign Indices

Word	barked	cat	dog	sat	the
Index	1	2	3	4	5

Step 3: Create Vectors Generate a 5-dimensional vector for each word based on its index:

- $v_{\text{barked}} = [1, 0, 0, 0, 0]$
- $v_{\text{cat}} = [0, 1, 0, 0, 0]$

- $v_{\text{dog}} = [0, 0, 1, 0, 0]$
- $v_{\text{sat}} = [0, 0, 0, 1, 0]$
- $v_{\text{the}} = [0, 0, 0, 0, 1]$

5.1.2 Term Frequency Inverse Document Frequency TF IDF

TF-IDF reflects how important a word is to a document in a collection (corpus). It balances the local frequency of a word in a specific document against its global frequency across all documents.

Methodology:

TF IDF Calculation Method

1. **Calculate Term Frequency (TF)**: Measures how frequently a term t appears in a document d.

$$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of words in document } d}$$

2. **Calculate Inverse Document Frequency (IDF)**: Measures the importance of the term across the entire corpus D of size N.

$$IDF(t, D) = \log_{10}\left(\frac{N}{\text{Number of documents containing term } t}\right)$$

3. **Compute TF-IDF Score**: Multiply TF and IDF.

$$\text{TF-IDF}(t, d, D) = TF(t, d) \times IDF(t, D)$$

Worked Example:

Calculating TF IDF Scores **Problem:** Consider a corpus of two documents:

- Document A: "machine learning is fun"
- Document B: "learning is hard"

Calculate the TF-IDF score for the words "machine" and "learning" in Document A.
Solution:
Step 1: Calculate TF for Document A Total words in Document A = 4.

- $TF(\text{"machine"}, A) = \frac{1}{4} = 0.25$
- $TF(\text{"learning"}, A) = \frac{1}{4} = 0.25$

Step 2: Calculate IDF Total documents $N = 2$.

- "machine" appears in 1 document (A).

$$IDF(\text{"machine"}) = \log_{10}\left(\frac{2}{1}\right) = \log_{10}(2) \approx 0.301$$

- "learning" appears in 2 documents (A and B).

$$IDF(\text{"learning"}) = \log_{10}\left(\frac{2}{2}\right) = \log_{10}(1) = 0$$

Step 3: Compute TF-IDF

- $\text{TF-IDF}(\text{"machine"}, A) = 0.25 \times 0.301 \approx 0.07525$

- TF-IDF("learning", A) $= 0.25 \times 0 = 0$

The score highlights that "machine" is highly uniquely informative for Document A, while "learning" is a common term providing less unique information.

5.1.3 Word2Vec Models CBOW and Skip Gram

Word2Vec produces dense vector representations by training a shallow neural network to predict words based on their context. It captures semantic relationships (e.g., King - Man + Woman ≈ Queen).

Methodology:

Word2Vec Architectures Overview Word2Vec employs two primary architectures that frame the learning problem inversely:

1. **Continuous Bag of Words (CBOW)**: Predicts the *target word* given the *context words*.
2. **Skip-Gram**: Predicts the *context words* given a *target word*.

Window Size (k): Defines the context limit. A window of size k means considering k words to the left and k words to the right of the target word.

Worked Example:

Generating Training Samples for Word2Vec **Problem:** Given the sentence "the quick brown fox jumps over the lazy dog" and a window size $k = 2$, generate the training input-output pairs for both CBOW and Skip-gram architectures centered on the target word "fox".
Solution:
Step 1: Identify Target and Context Target Word: "fox" Context Window ($k = 2$): Two words to the left and two words to the right. Context Words: ["quick", "brown", "jumps", "over"]
Step 2: CBOW Training Sample CBOW uses context words as input to predict the target word.

- **Input**: (quick, brown, jumps, over)
- **Output**: fox

Step 3: Skip-gram Training Samples Skip-gram uses the target word as input to predict each context word individually.

- **Input**: fox → **Output**: quick
- **Input**: fox → **Output**: brown
- **Input**: fox → **Output**: jumps
- **Input**: fox → **Output**: over

5.2 Compare Word Embedding Techniques

Selecting the right embedding technique depends on the problem requirements, computational resources, and the need for semantic understanding.

Technique	Mechanism	Advantages	Disadvantages
One-Hot Encoding	Represents words as sparse, high-dimensional binary vectors.	Extremely simple to implement and understand.	Vectors are orthogonal (no semantic relationship). High memory usage.
Bag of Words BoW	Counts word frequencies in a document.	Simple, captures basic word occurrence patterns.	Ignores word order and semantics. Sparse vectors.
TF-IDF	Weights word frequencies by their rarity across all documents.	Excellent for information retrieval and keyword extraction.	Still sparse. Fails to capture word order and deep semantics.
Word2Vec Models	Shallow neural network predicting context or target words.	Captures deep semantic meaning and relationships. Fixed, dense vector size.	Cannot handle out-of-vocabulary (OOV) words well. Static embeddings.

5.3 Identify Applications of NLP

Natural Language Processing leverages these embeddings to solve complex real-world tasks. Key applications include:

1. **Sentiment Analysis**: Classifying text (like reviews or tweets) into positive, negative, or neutral sentiments.
2. **Machine Translation**: Converting text from one language to another (e.g., Google Translate).
3. **Named Entity Recognition NER**: Identifying and classifying entities like names, dates, and locations within a text.
4. **Text Summarization**: Condensing long articles into short, informative summaries.

To illustrate the computational application of NLP, consider the probabilistic approach to Sentiment Analysis using the Naive Bayes algorithm on embedded text.

Methodology:

Naive Bayes for Text Classification The Multinomial Naive Bayes classifier assigns a class C to a document D by calculating the maximum posterior probability.

1. **Calculate Prior Probability**: $P(C) = \frac{\text{Number of documents in class } C}{\text{Total number of documents}}$
2. **Calculate Conditional Probability**: For each word w_i in the vocabulary, with Laplace smoothing ($\alpha = 1$):
$$P(w_i|C) = \frac{\text{Count of } w_i \text{ in class } C + 1}{\text{Total words in class } C + |V|}$$
Where $|V|$ is the total number of unique words in the vocabulary.
3. **Compute Document Probability**: Multiply the probabilities for a test document $D = \{w_1, w_2, \ldots, w_n\}$:
$$P(C|D) \propto P(C) \prod_{i=1}^{n} P(w_i|C)$$

Worked Example:

Classifying Sentiment of a Review **Problem:** You are given the following training dataset of movie reviews:

- Positive (P): "good movie", "very good"
- Negative (N): "bad movie"

Classify the test review "good bad movie" using Naive Bayes.
Solution:
Step 1: Build Vocabulary and Calculate Priors Total docs = 3. Class P docs = 2. Class N docs = 1. $P(P) = \frac{2}{3}$ $P(N) = \frac{1}{3}$
Vocabulary $V = \{\text{bad}, \text{good}, \text{movie}, \text{very}\}$. Size $|V| = 4$. Total words in P = 4. Total words in N = 2.
Step 2: Calculate Conditional Probabilities (with +1 smoothing) For Class Positive (P):

- $P(\text{good}|P) = \frac{2+1}{4+4} = \frac{3}{8}$
- $P(\text{bad}|P) = \frac{0+1}{4+4} = \frac{1}{8}$
- $P(\text{movie}|P) = \frac{1+1}{4+4} = \frac{2}{8}$

For Class Negative (N):

- $P(\text{good}|N) = \frac{0+1}{2+4} = \frac{1}{6}$
- $P(\text{bad}|N) = \frac{1+1}{2+4} = \frac{2}{6}$
- $P(\text{movie}|N) = \frac{1+1}{2+4} = \frac{2}{6}$

Step 3: Compute Probabilities for Test Document "good bad movie"

- **Score for P**: $P(P) \times P(\text{good}|P) \times P(\text{bad}|P) \times P(\text{movie}|P)$

$$\text{Score}_P = \frac{2}{3} \times \frac{3}{8} \times \frac{1}{8} \times \frac{2}{8} = \frac{12}{1536} \approx 0.0078$$

- **Score for N**: $P(N) \times P(\text{good}|N) \times P(\text{bad}|N) \times P(\text{movie}|N)$

$$\text{Score}_N = \frac{1}{3} \times \frac{1}{6} \times \frac{2}{6} \times \frac{2}{6} = \frac{4}{648} \approx 0.0061$$

Conclusion: Since $\text{Score}_P(0.0078) > \text{Score}_N(0.0061)$, the review "good bad movie" is classified as Positive.

Appendix: Key Formulae

This appendix provides a quick reference to the key formulae and mathematical relationships discussed in this book.

Reinforcement Learning (Unit 2)

Q-Learning

The Q-learning update rule is used to iteratively update the Q-values based on the reward received and the maximum expected future reward:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Where:

- $Q(s_t, a_t)$ is the current Q-value for state s_t and action a_t.
- α is the learning rate.
- R_{t+1} is the reward received after taking action a_t.
- γ is the discount factor.
- $\max_a Q(s_{t+1}, a)$ is the maximum predicted Q-value for the next state s_{t+1}.

Artificial Neural Networks (Unit 3)

Neuron Computations

The pre-activation or linear combination z for a neuron is computed as the weighted sum of inputs plus a bias term:

$$z = \sum_{i=1}^{n} (x_i \cdot w_i) + b = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + b$$

For a multi-layer network, the computation at layer l and neuron j is generalized as:

$$z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$

The output or activation of the neuron is derived by applying an activation function f to z:

$$y = f(z) \quad \text{or} \quad a_j^{(l)} = f(z_j^{(l)})$$

Activation Functions

Various activation functions are used to introduce non-linearity into the network:

Step Function:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Sigmoid Function:

$$f(z) = \frac{1}{1 + e^{-z}}$$

Hyperbolic Tangent (Tanh) Function:

$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Rectified Linear Unit (ReLU) Function:

$$f(z) = \max(0, z)$$

Error and Weight Updates

The error in prediction is the difference between the target and predicted value:

$$e = y_{\text{target}} - y_{\text{pred}}$$

For simple perceptrons (without activation function derivatives):

$$w_i^{(\text{new})} = w_i^{(\text{old})} + \alpha \cdot e \cdot x_i$$

$$b^{(\text{new})} = b^{(\text{old})} + \alpha \cdot e$$

Using gradient descent or the delta rule (incorporating the derivative of the activation function $f'(z)$):

$$w_i^{(\text{new})} = w_i^{(\text{old})} + \alpha \cdot e \cdot f'(z) \cdot x_i$$

$$b^{(\text{new})} = b^{(\text{old})} + \alpha \cdot e \cdot f'(z)$$

Natural Language Processing & Information Retrieval (Units 4 & 5)

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF is used to evaluate how important a word is to a document in a collection or corpus.

Term Frequency (TF):

$$TF(t, d) = \frac{\text{Count of term } t \text{ in document } d}{\text{Total number of terms in document } d}$$

Inverse Document Frequency (IDF):

$$IDF(t, D) = \log_{10}\left(\frac{N}{\text{Number of documents containing term } t}\right)$$

Where N is the total number of documents in the corpus D.

TF-IDF Score:

$$\text{TF-IDF}(t, d, D) = TF(t, d) \times IDF(t, D)$$

Naive Bayes Classification

Naive Bayes is a probabilistic classifier based on Bayes' theorem with an assumption of independence among predictors.

Class Prediction given Document:

$$P(C|D) \propto P(C) \prod_{i=1}^{n} P(w_i|C)$$

Where $P(C)$ is the prior probability of class C, and $P(w_i|C)$ is the likelihood of word w_i given class C.

Laplace Smoothing for Likelihood:

$$P(w_i|C) = \frac{\text{Count of } w_i \text{ in class } C + 1}{\text{Total words in class } C + |V|}$$

Where $|V|$ is the size of the vocabulary (total number of unique words across all classes).