Introduction to Artificial Intelligence

Engr. Dr. Muhammad Siddique

HOD, Department of Artificial Intelligence
NFC IET Multan

August 30, 2025

Outline of Presentation

1. Introduction to AI
2. History of AI
3. Applications of AI
4. Summary and Discussion

Introduction to Artificial Intelligence

What is Artificial Intelligence?

Definition of AI

- AI is the simulation of human intelligence in machines.

What is Artificial Intelligence?

Definition of AI

- AI is the simulation of human intelligence in machines.
- These systems can think, learn, reason, and make decisions.

What is Artificial Intelligence?

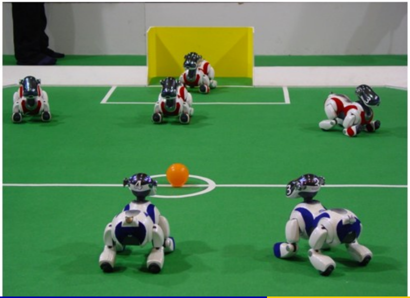
Definition of AI

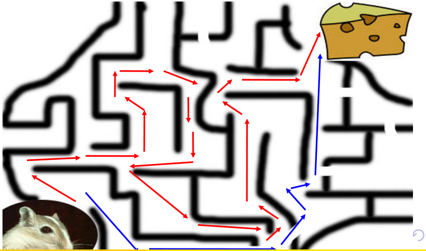
- AI is the simulation of human intelligence in machines.
- These systems can think, learn, reason, and make decisions.
- It involves algorithms, models, and techniques to mimic human-like problem solving.

Key Characteristics of AI

Intelligence in Machines

- Learning from data (Machine Learning).

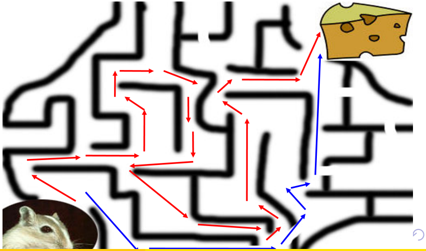




Key Characteristics of AI

Intelligence in Machines

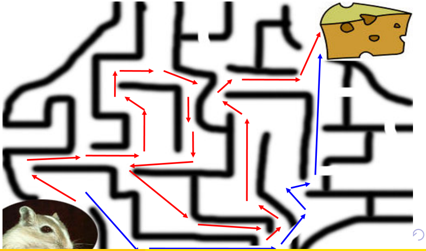
- Learning from data (Machine Learning).
- Reasoning and problem-solving.



Key Characteristics of AI

Intelligence in Machines

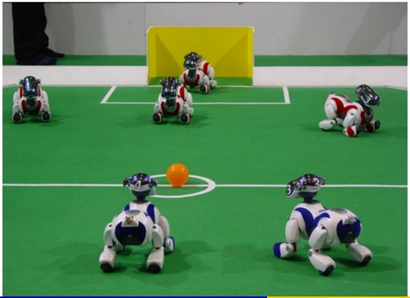
- Learning from data (Machine Learning).
- Reasoning and problem-solving.
- Perception and understanding (Vision, Speech).

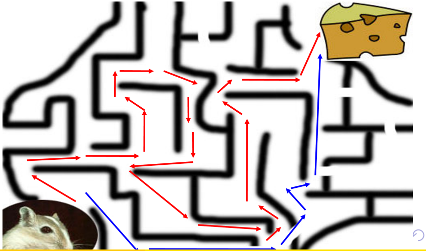


Key Characteristics of AI

Intelligence in Machines

- Learning from data (Machine Learning).
- Reasoning and problem-solving.
- Perception and understanding (Vision, Speech).
- Decision making under uncertainty.

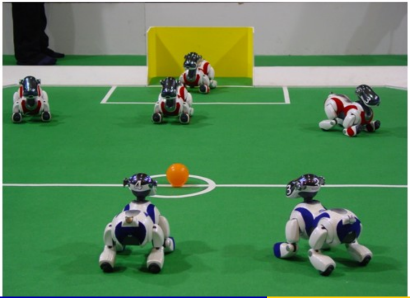


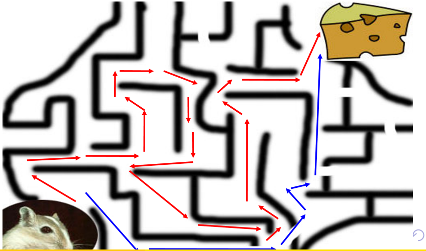


Key Characteristics of AI

Intelligence in Machines

- Learning from data (Machine Learning).
- Reasoning and problem-solving.
- Perception and understanding (Vision, Speech).
- Decision making under uncertainty.
- Natural language understanding and interaction.





History and Evolution of AI

Early History of AI

Foundations of AI

- 1940s–1950s: Conceptual foundations in mathematics and logic.

Early History of AI

Foundations of AI

- 1940s–1950s: Conceptual foundations in mathematics and logic.
- 1956: Term “Artificial Intelligence” coined at the Dartmouth Conference.

Early History of AI

Foundations of AI

- 1940s–1950s: Conceptual foundations in mathematics and logic.
- 1956: Term “Artificial Intelligence” coined at the Dartmouth Conference.
- Early programs focused on symbolic reasoning and problem solving.

1940s–1950s: Conceptual Foundations in Mathematics and Logic

Early Foundations of AI

- **Mathematical Logic:** Formal systems and symbolic logic laid the groundwork for reasoning.

1940s–1950s: Conceptual Foundations in Mathematics and Logic

Early Foundations of AI

- **Mathematical Logic:** Formal systems and symbolic logic laid the groundwork for reasoning.
- **Turing's Work (1936):** Alan Turing introduced the concept of the *Turing Machine*, defining the limits of computability.

1940s–1950s: Conceptual Foundations in Mathematics and Logic

Early Foundations of AI

- **Mathematical Logic:** Formal systems and symbolic logic laid the groundwork for reasoning.
- **Turing's Work (1936):** Alan Turing introduced the concept of the *Turing Machine*, defining the limits of computability.
- **Von Neumann Architecture:** Provided the model for stored-program computers essential for AI programs.

1940s–1950s: Conceptual Foundations in Mathematics and Logic

Early Foundations of AI

- **Mathematical Logic:** Formal systems and symbolic logic laid the groundwork for reasoning.
- **Turing's Work (1936):** Alan Turing introduced the concept of the *Turing Machine*, defining the limits of computability.
- **Von Neumann Architecture:** Provided the model for stored-program computers essential for AI programs.
- **Game Theory and Decision Theory:** Introduced by John von Neumann and Oskar Morgenstern for strategic decision making.

1940s–1950s: Conceptual Foundations in Mathematics and Logic

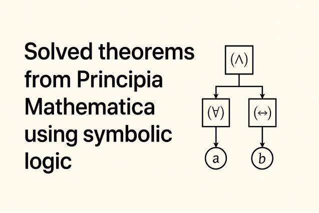
Early Foundations of AI

- **Mathematical Logic:** Formal systems and symbolic logic laid the groundwork for reasoning.
- **Turing's Work (1936):** Alan Turing introduced the concept of the *Turing Machine*, defining the limits of computability.
- **Von Neumann Architecture:** Provided the model for stored-program computers essential for AI programs.
- **Game Theory and Decision Theory:** Introduced by John von Neumann and Oskar Morgenstern for strategic decision making.
- **Cybernetics:** Norbert Wiener proposed feedback systems influencing early AI thought.

Early Programs in AI: Symbolic Reasoning and Problem Solving

What was Symbolic Reasoning?

- Symbolic reasoning involves manipulating symbols to represent knowledge and rules.

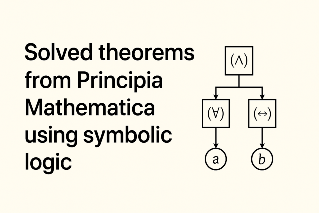


Solved theorems from Principia Mathematica using symbolic logic

Early Programs in AI: Symbolic Reasoning and Problem Solving

What was Symbolic Reasoning?

- Symbolic reasoning involves manipulating symbols to represent knowledge and rules.
- Programs used logic and search strategies to simulate human problem-solving.

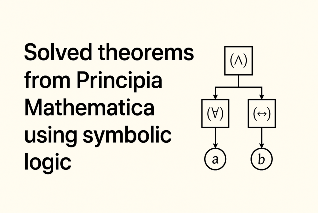


Solved theorems from Principia Mathematica using symbolic logic

Early Programs in AI: Symbolic Reasoning and Problem Solving

What was Symbolic Reasoning?

- Symbolic reasoning involves manipulating symbols to represent knowledge and rules.
- Programs used logic and search strategies to simulate human problem-solving.
- Early systems were based on the assumption that intelligence can be achieved by applying formal rules to symbols.



Solved theorems from Principia Mathematica using symbolic logic

AI Winter and Renaissance

Ups and Downs of AI Research

- 1970s–1980s: Funding cuts and skepticism – AI Winter.

AI Winter and Renaissance

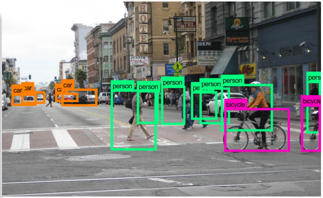
Ups and Downs of AI Research

- 1970s–1980s: Funding cuts and skepticism – AI Winter.
- 1990s: Revival with expert systems and neural networks.

AI Winter and Renaissance

Ups and Downs of AI Research

- 1970s–1980s: Funding cuts and skepticism – AI Winter.
- 1990s: Revival with expert systems and neural networks.
- 2000s onward: Big Data and Deep Learning revolution.



Modern AI Era

Recent Developments

- Machine Learning and Deep Learning dominance.

Modern AI Era

Recent Developments

- Machine Learning and Deep Learning dominance.
- Applications in robotics, natural language processing, and healthcare.

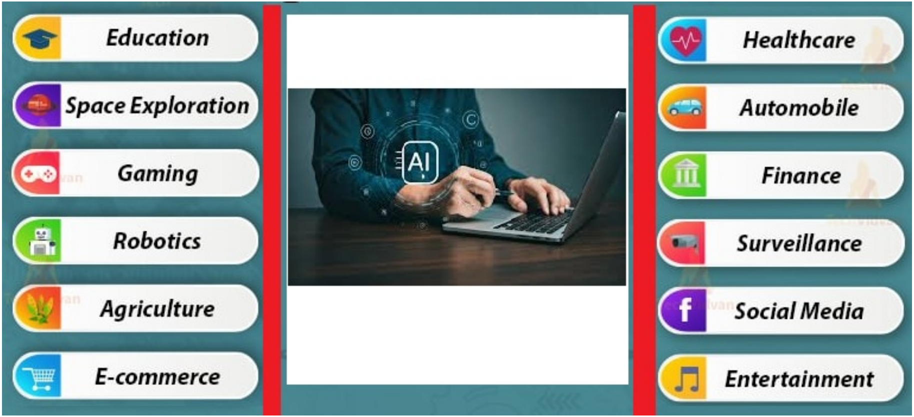
Modern AI Era

Recent Developments

- Machine Learning and Deep Learning dominance.
- Applications in robotics, natural language processing, and healthcare.
- AI-powered assistants like Siri, Alexa, and ChatGPT.

Applications of Artificial Intelligence

Major Domains of AI Applications



AI in Daily Life

Examples You Know

- Virtual assistants (Google Assistant, Siri).

AI in Daily Life

Examples You Know

- Virtual assistants (Google Assistant, Siri).
- Recommendation systems (Netflix, YouTube).

AI in Daily Life

Examples You Know

- Virtual assistants (Google Assistant, Siri).
- Recommendation systems (Netflix, YouTube).
- Smart home devices (IoT-based AI).

Summary

Key Takeaways

- AI aims to create machines that mimic human intelligence.
- The field has evolved from symbolic AI to deep learning.
- AI applications are now part of our everyday life.

Discussion Questions

Think About It

- Is AI making humans smarter or lazier?

Discussion Questions

Think About It

- Is AI making humans smarter or lazier?
- Can AI ever fully replicate human intelligence?

Discussion Questions

Think About It

- Is AI making humans smarter or lazier?
- Can AI ever fully replicate human intelligence?
- What ethical issues arise with AI development?

Foundations of Artificial Intelligence

Agents, Search, and Games

Engr. Dr. Muhammad Siddique

HOD, Department of Artificial Intelligence
NFC IET Multan

August 30, 2025

Outline of Presentation

1. Intelligent Agents and Environments
2. Intelligent Agents and Environments
3. Problem Solving and Search
4. Problem Solving and Search in AI
5. Classical Search Algorithms
6. Classical Search Algorithms
7. A* Search Example – Navigation
8. A* Search — Numerical Example
9. Adversarial Search – Games and Minimax
10. Adversarial Search – Games and Minimax Algorithm
11. Summary

Intelligent Agents and Environments

Intelligent Agents and Environments

What is an Agent?

Definition

- An **agent** is anything that can perceive its environment through sensors and act upon it through actuators.

What is an Agent?

Definition

- An **agent** is anything that can perceive its environment through sensors and act upon it through actuators.
- AI agents aim to act **rationally** to achieve goals.

What is an Agent?

Definition

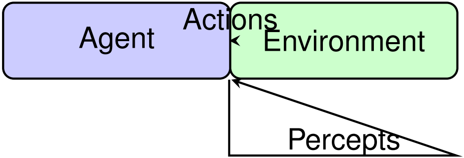
- An **agent** is anything that can perceive its environment through sensors and act upon it through actuators.
- AI agents aim to act **rationally** to achieve goals.
- Examples: autonomous car, robot vacuum, chatbot.

PEAS Framework for Agents

PEAS = Performance, Environment, Actuators, Sensors

- **Performance Measure** – How we evaluate success.
- **Environment** – The world in which the agent operates.
- **Actuators** – Tools for taking actions.
- **Sensors** – Tools for perceiving the environment.

Agent–Environment Interaction



Rationality of Agents

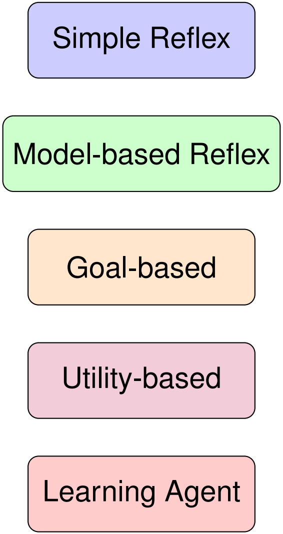
Rational Agent

- Chooses action expected to maximize performance measure.
- Rationality depends on:
  - Performance measure.
  - Knowledge of environment.
  - Available actions.
  - Percept history.

Types of Agents

- **Simple Reflex Agent** – Acts on current percept.
- **Model-based Reflex Agent** – Uses internal model of world.
- **Goal-based Agent** – Chooses actions to achieve goals.
- **Utility-based Agent** – Maximizes happiness/utility.
- **Learning Agent** – Improves performance over time.

Types of Agents – Diagram



Simple Reflex

Model-based Reflex

Goal-based

Utility-based

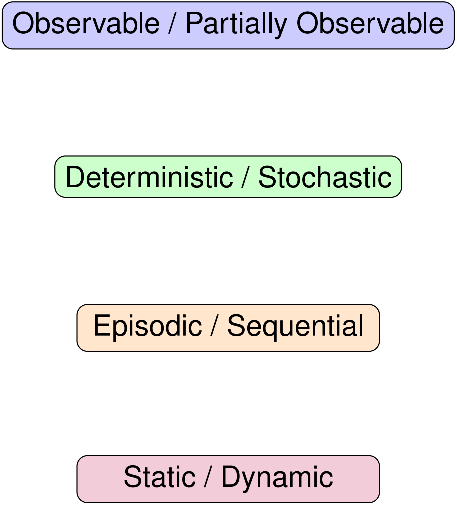
Learning Agent

Properties of Environments

Dimensions

- Fully Observable vs. Partially Observable
- Deterministic vs. Stochastic
- Episodic vs. Sequential
- Static vs. Dynamic
- Discrete vs. Continuous
- Single-agent vs. Multi-agent

Environment Classification Diagram



Observable / Partially Observable

Deterministic / Stochastic

Episodic / Sequential

Static / Dynamic

Examples of Intelligent Agents

- **Self-driving car** – perceives road with sensors, acts with steering/acceleration.
- **Medical diagnostic system** – perceives patient data, suggests treatments.
- **Robot vacuum cleaner** – senses dust and obstacles, takes cleaning actions.
- **Chess-playing program** – perceives board, acts by making moves.

Problem Solving and Search in AI

What is Problem Solving in AI?

Definition

- Problem solving in AI involves finding a sequence of actions that leads from an initial state to a goal state.

What is Problem Solving in AI?

Definition

- Problem solving in AI involves finding a sequence of actions that leads from an initial state to a goal state.
- It is the basis of reasoning, planning, and decision-making.

What is Problem Solving in AI?

Definition

- Problem solving in AI involves finding a sequence of actions that leads from an initial state to a goal state.
- It is the basis of reasoning, planning, and decision-making.
- AI uses search techniques to explore possible solutions.

Components of a Problem in AI

- **Initial State** – The starting point of the problem.

Components of a Problem in AI

- **Initial State** – The starting point of the problem.
- **Goal State** – The desired final state.

Components of a Problem in AI

- **Initial State** – The starting point of the problem.
- **Goal State** – The desired final state.
- **Actions** – Operations that move between states.

Components of a Problem in AI

- **Initial State** – The starting point of the problem.
- **Goal State** – The desired final state.
- **Actions** – Operations that move between states.
- **Transition Model** – Rules defining results of actions.

Components of a Problem in AI

- **Initial State** – The starting point of the problem.
- **Goal State** – The desired final state.
- **Actions** – Operations that move between states.
- **Transition Model** – Rules defining results of actions.
- **Path Cost** – Numerical cost associated with actions.

Problem Representation

Representation Techniques

- State-space representation (nodes = states, edges = actions).

Problem Representation

Representation Techniques

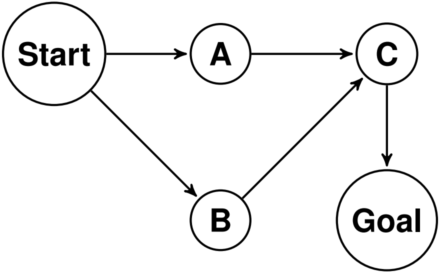
- State-space representation (nodes = states, edges = actions).
- Graphs and trees represent possible solution paths.

Problem Representation

Representation Techniques

- State-space representation (nodes = states, edges = actions).
- Graphs and trees represent possible solution paths.
- Search algorithms explore these structures.

State-Space Representation



Types of Problems in AI

Categories

- **Single-state problems:** Only one state known.

Types of Problems in AI

Categories

- **Single-state problems:** Only one state known.
- **Multiple-state problems:** Agent must consider multiple possibilities.

Types of Problems in AI

Categories

- **Single-state problems:** Only one state known.
- **Multiple-state problems:** Agent must consider multiple possibilities.
- **Contingency problems:** Uncertain environment, agent prepares conditional plans.

Types of Problems in AI

Categories

- **Single-state problems:** Only one state known.
- **Multiple-state problems:** Agent must consider multiple possibilities.
- **Contingency problems:** Uncertain environment, agent prepares conditional plans.
- **Exploration problems:** Agent has incomplete information and learns while solving.

Example: 8-Puzzle Problem

- Start with a 3x3 grid with 8 numbered tiles and one blank.

Start State

1	2	3
4	5	6
7	8	

Example: 8-Puzzle Problem

- Start with a 3x3 grid with 8 numbered tiles and one blank.
- Goal: Move tiles to reach a specific configuration.

Start State

1	2	3
4	5	6
7	8	

Example: 8-Puzzle Problem

- Start with a 3x3 grid with 8 numbered tiles and one blank.
- Goal: Move tiles to reach a specific configuration.
- Actions: Slide a tile into the blank position.

Start State

1	2	3
4	5	6
7	8	

Search as Problem Solving

Key Idea

- Search = systematic exploration of problem space.

Search as Problem Solving

Key Idea

- Search = systematic exploration of problem space.
- The agent examines paths from the initial state to the goal.

Search as Problem Solving

Key Idea

- Search = systematic exploration of problem space.
- The agent examines paths from the initial state to the goal.
- Different strategies determine efficiency and success.

Tree Search vs Graph Search

Tree Search

- Expands nodes without checking for repeats.
- May revisit the same state multiple times.

Graph Search

- Uses memory to store visited states.
- Avoids revisiting same state → more efficient.

Classical Search Algorithms (DFS, BFS, A*)

Introduction to Classical Search Algorithms

Overview

- Classical search algorithms are fundamental techniques in AI for problem-solving.

Introduction to Classical Search Algorithms

Overview

- Classical search algorithms are fundamental techniques in AI for problem-solving.
- They operate on state spaces represented as trees or graphs.

Introduction to Classical Search Algorithms

Overview

- Classical search algorithms are fundamental techniques in AI for problem-solving.
- They operate on state spaces represented as trees or graphs.
- Focused on exploring nodes until a solution (goal state) is found.

Introduction to Classical Search Algorithms

Overview

- Classical search algorithms are fundamental techniques in AI for problem-solving.
- They operate on state spaces represented as trees or graphs.
- Focused on exploring nodes until a solution (goal state) is found.
- Key algorithms:
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
  - A* Search

Depth-First Search (DFS)

Key Points

- Explores as far as possible along each branch before backtracking.

Depth-First Search (DFS)

Key Points

- Explores as far as possible along each branch before backtracking.
- Uses a stack (LIFO structure).

Depth-First Search (DFS)

Key Points

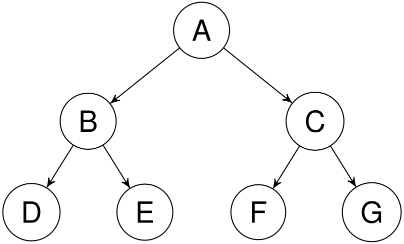
- Explores as far as possible along each branch before backtracking.
- Uses a stack (LIFO structure).
- May not always find the shortest path.

Depth-First Search (DFS)

Key Points

- Explores as far as possible along each branch before backtracking.
- Uses a stack (LIFO structure).
- May not always find the shortest path.
- Space efficient but may get stuck in infinite paths.

DFS Example



DFS order: A → B → D → E → C → F → G

Breadth-First Search (BFS)

Key Points

- Explores all nodes at the current depth before moving to the next level.

Breadth-First Search (BFS)

Key Points

- Explores all nodes at the current depth before moving to the next level.
- Uses a queue (FIFO structure).

Breadth-First Search (BFS)

Key Points

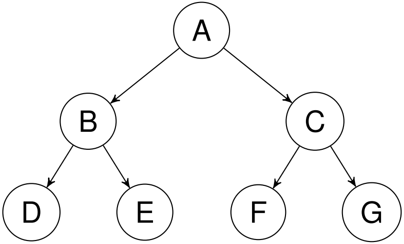
- Explores all nodes at the current depth before moving to the next level.
- Uses a queue (FIFO structure).
- Always finds the shortest path (if all edges have equal weight).

Breadth-First Search (BFS)

Key Points

- Explores all nodes at the current depth before moving to the next level.
- Uses a queue (FIFO structure).
- Always finds the shortest path (if all edges have equal weight).
- Can be memory intensive.

BFS Example



BFS order: A → B → C → D → E → F → G

A* Search Algorithm

Key Points

- Informed search algorithm that uses heuristics.

A* Search Algorithm

Key Points

- Informed search algorithm that uses heuristics.
- Combines cost so far ($g(n)$) with estimated cost to goal ($h(n)$).

A* Search Algorithm

Key Points

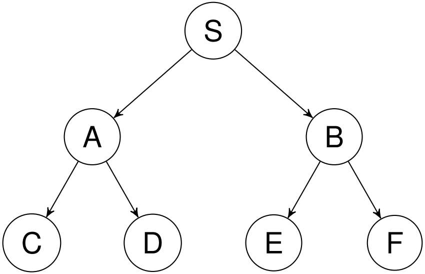
- Informed search algorithm that uses heuristics.
- Combines cost so far ($g(n)$) with estimated cost to goal ($h(n)$).
- Evaluation function: $f(n) = g(n) + h(n)$.

A* Search Algorithm

Key Points

- Informed search algorithm that uses heuristics.
- Combines cost so far ($g(n)$) with estimated cost to goal ($h(n)$).
- Evaluation function: $f(n) = g(n) + h(n)$.
- Optimal if the heuristic $h(n)$ is admissible (never overestimates).

A* Example



A* expands nodes with minimum $f(n) = g(n) + h(n)$

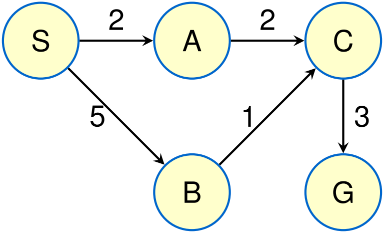
A* Search: Real-Life Example – Navigation

A* Search in Navigation

Scenario

- A* is widely used in **GPS navigation systems.**
- Goal: Find the shortest route from a **Start City (S)** to a **Goal City (G).**
- Uses:
  - $g(n)$: Cost to reach a node (actual distance traveled).
  - $h(n)$: Heuristic estimate to goal (straight-line distance).
  - $f(n) = g(n) + h(n)$.

Road Map Example with Distances



Heuristic Estimates (Straight-line Distances to Goal G)

- $h(S) = 6$, $h(A) = 4$, $h(B) = 2$, $h(C) = 2$, $h(G) = 0$

A* Search Step 1

Initial Expansion

- Start at S: $g(S) = 0$, $h(S) = 6 \Rightarrow f(S) = 6$
- Expand neighbors:
    - A: $g(A) = 2$, $h(A) = 4$, $f(A) = 6$
    - B: $g(B) = 5$, $h(B) = 2$, $f(B) = 7$
- Frontier: $\{A(6), B(7)\} \rightarrow$ Pick A (lowest f).

A* Search Step 2

Expanding Node A

- From $A \rightarrow C$: $g(C) = 2 + 2 = 4$, $h(C) = 2$, $f(C) = 6$
- Frontier now: $\{C(6), B(7)\} \rightarrow$ Pick C.

A* Search Step 3

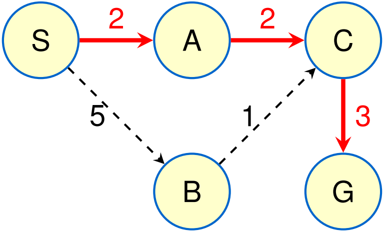
Expanding Node C

- From $C \to G$: $g(G) = 4 + 3 = 7$, $h(G) = 0$, $f(G) = 7$
- Frontier: $\{G(7), B(7)\}$
- Select $G \to$ Goal reached.

Final Path and Cost

Optimal Path

- Path found: $S \to A \to C \to G$
- Total cost: 7

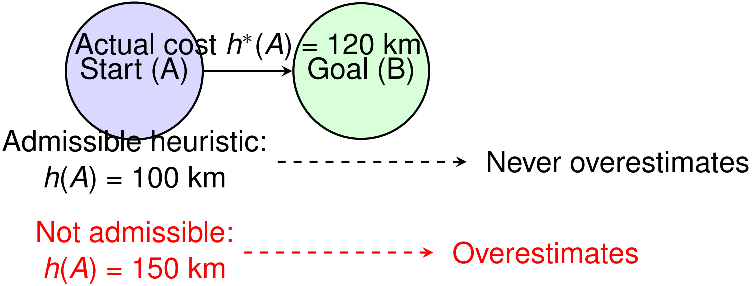


Admissible Heuristic

- A heuristic $h(n)$ is **admissible** if it never overestimates the true cost to reach the goal.

$$h(n) \leq h^*(n) \quad \forall n$$

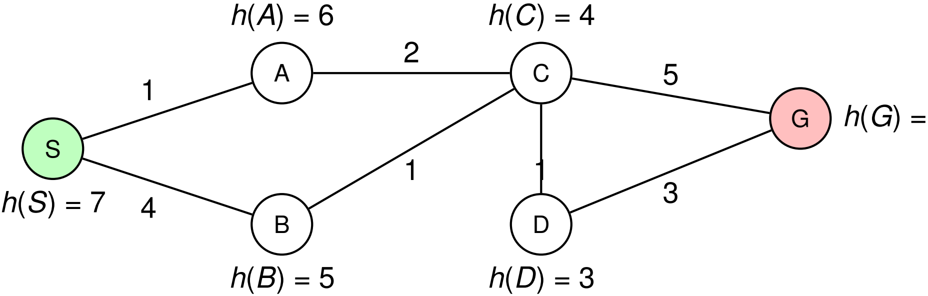
- Ensures that A* search always finds the optimal solution.



A* Search: Numerical Example (Step-by-Step)

Problem Setup

- Nodes: Start S, intermediate A, B, C, D, Goal G.
- Edge costs shown on the graph (positive weights).
- Heuristic $h(n)$ is **admissible** (never overestimates best remaining cost).



Heuristic Values and Optimal Cost Check

- Best path (we will discover): $S \to A \to C \to D \to G$ with cost $1 + 2 + 1 + 3 = 7$.
- Heuristic assignments (admissible):

$$h(S) = 7, \quad h(A) = 6, \quad h(B) = 5, \quad h(C) = 4, \quad h(D) = 3, \quad h(G) = 0.$$

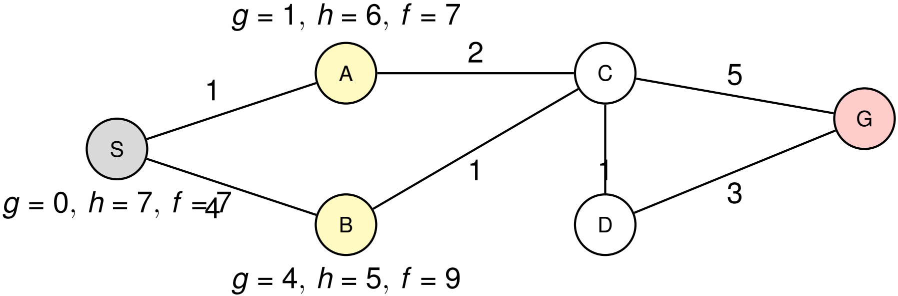
- A* uses $f(n) = g(n) + h(n)$. Initially: $g(S) = 0 \Rightarrow f(S) = 7$.

Step 1: Expand S

- OPEN initially: $\{S : g = 0, h = 7, f = 7\}$. Pick min $f \Rightarrow S$.
- Generate neighbors:

$$A:\ g = 1,\ h = 6,\ f = 7 \qquad B:\ g = 4,\ h = 5,\ f = 9$$

- OPEN $\leftarrow \{A(1, 6, 7),\ B(4, 5, 9)\}$, CLOSED $\leftarrow \{S\}$.

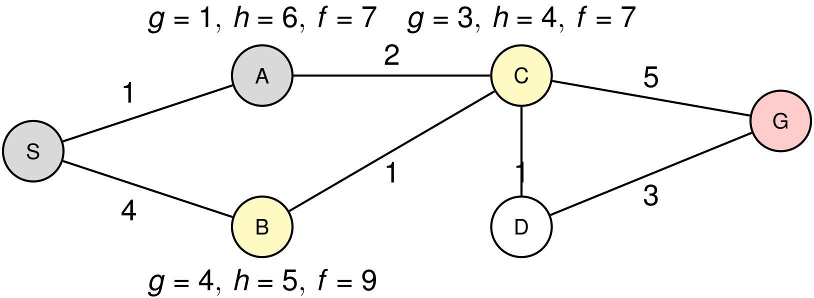


Step 2: Expand A (min f in OPEN)

- Pick A (tie on $f = 7$ broken in favor of A).
- Generate C via A:

$$C: \; g = 1 + 2 = 3, \; h = 4, \; f = 7$$

- OPEN $\leftarrow \{C(3,4,7), \; B(4,5,9)\}$, CLOSED $\leftarrow \{S, A\}$.

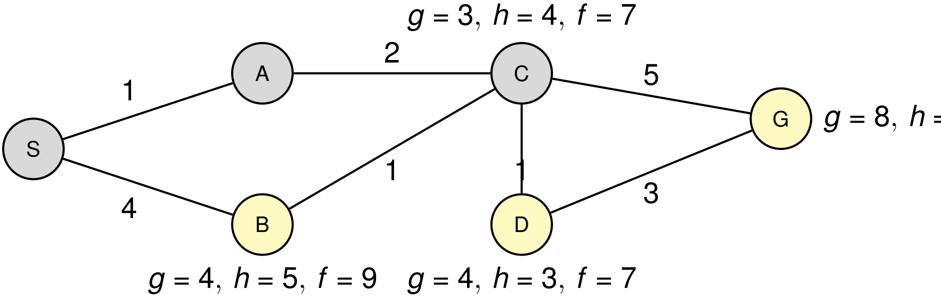

$g = 1, \; h = 6, \; f = 7$
$g = 3, \; h = 4, \; f = 7$
2
5
1
A
C
G
S
1
1
4
3
B
D
$g = 4, \; h = 5, \; f = 9$

Step 3: Expand C (min f in OPEN)

- From C, generate:

$$D:\ g = 3 + 1 = 4,\ h = 3,\ f = 7 \qquad G:\ g = 3 + 5 = 8,\ h = 0,\ f = 8$$

- OPEN $\leftarrow \{D(4,3,7),\ G(8,0,8),\ B(4,5,9)\}$, CLOSED $\leftarrow \{S, A, C\}$.

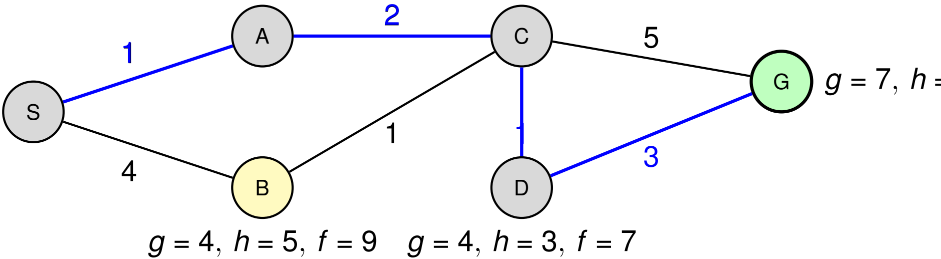


Step 4: Expand *D* (min *f* in OPEN) and Reach *G*

- From D: G via D yields

$$g = 4 + 3 = 7, \quad h = 0, \quad f = 7.$$

- This improves previous G (from $f = 8$ to $f = 7$); update G.
- OPEN $\leftarrow \{G(7,0,7),\ B(4,5,9)\}$, CLOSED $\leftarrow \{S, A, C, D\}$.
- Pop G (min f). Goal reached with total cost 7.



Result and Optimality

Optimal Path Found

$$S \to A \to C \to D \to G, \quad \text{Cost} = 1 + 2 + 1 + 3 = 7.$$

- Because $h(n)$ is **admissible**, A* returns an **optimal** path.
- Note: If a cheaper path to a node is found, A* updates its g and f (as done for G).

Comparison of DFS, BFS, and A*

Summary

- DFS: Space efficient but not optimal.

Comparison of DFS, BFS, and A*

Summary

- DFS: Space efficient but not optimal.
- BFS: Finds shortest path but uses high memory.

Comparison of DFS, BFS, and A*

Summary

- DFS: Space efficient but not optimal.
- BFS: Finds shortest path but uses high memory.
- A*: Uses heuristics, efficient, and optimal with admissible heuristic.

Adversarial Search – Games and Minimax

Adversarial Search – Games and Minimax Algorithm

What is Adversarial Search?

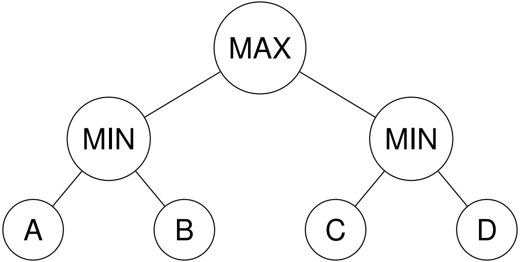
Concept

- Used in **competitive environments** where agents act against each other.
- Typical in **two-player games** (e.g., Chess, Tic-Tac-Toe).
- One player tries to **maximize** score while the other tries to **minimize**.
- Hence called the **Minimax Algorithm**.

Game Tree Representation

Structure of Game Tree

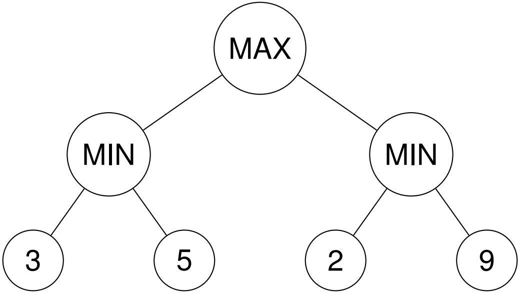
- **Nodes:** States of the game.
- **Edges:** Actions leading to next states.
- Levels alternate between **MAX** (our move) and **MIN** (opponent's move).



Minimax Example: Step 1 (Leaf Nodes)

Assigning Utility Values

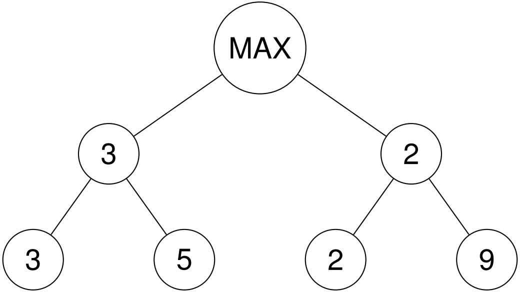
- Suppose leaf nodes represent the final scores of a game.
- MAX wants to maximize, MIN wants to minimize.



Minimax Example: Step 2 (MIN Evaluation)

Evaluating MIN nodes

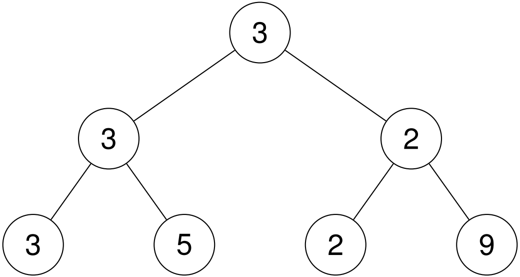
- Left MIN node = $\min(3, 5) = 3$.
- Right MIN node = $\min(2, 9) = 2$.



Minimax Example: Step 3 (MAX Decision)

Evaluating MAX node

- MAX chooses the maximum between its children.
- Left branch = 3, Right branch = 2.
- $\max(3, 2) = \mathbf{3}$.
- Therefore, MAX selects the **left branch**.



Real-Life Applications of Minimax

Examples

- **Board Games:** Chess, Checkers, Tic-Tac-Toe.
- **Negotiation Strategies:** maximize own gain, minimize loss.
- **Cybersecurity:** defender vs attacker strategies.
- **Robotics:** competing robots in resource-limited environments.

Summary

- Agents interact with environments using sensors and actuators.
- AI problems solved via search (uninformed and informed).
- Classical search algorithms: BFS, DFS, A*.
- Adversarial search handled via minimax and alpha-beta pruning.

Foundations of Artificial Intelligence

Part 3 – Knowledge & Reasoning

Engr. Dr. Muhammad Siddique

HOD, Department of Artificial Intelligence
NFC IET Multan

August 30, 2025

Outline of Part 3

1. Knowledge Representation in AI
2. Logic in AI
3. Reasoning Under Uncertainty
4. Expert Systems and Rule-Based Systems

Knowledge Representation in AI

What is Knowledge Representation?

Definition

- The study of how to represent information about the world in a form that a computer system can use.
- Important for reasoning, problem solving, and learning.
- Enables AI to connect perception with action.

Why is Knowledge Representation Important?

- Provides a bridge between raw data and intelligent decision-making.
- Allows AI systems to explain reasoning.
- Facilitates natural language understanding.
- Real-world example: **Medical Diagnosis Systems** represent symptoms and diseases in structured knowledge bases.

Forms of Knowledge Representation

- **Logical Representation** – Facts and rules in symbolic logic.
- **Semantic Networks** – Graph structures for concepts and relationships.
- **Frames** – Data structures representing stereotypical situations.
- **Production Rules** – IF-THEN rules.
- **Ontologies** – Formal definition of concepts and relationships.

Logical Representation Example

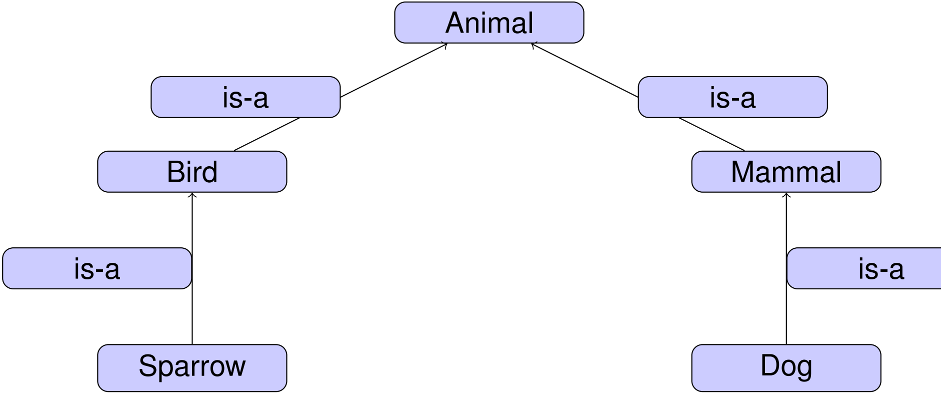
Traffic Lights

- Rule: **IF** light is red **THEN** cars must stop.
- Rule: **IF** light is green **THEN** cars may go.
- Logical Form:

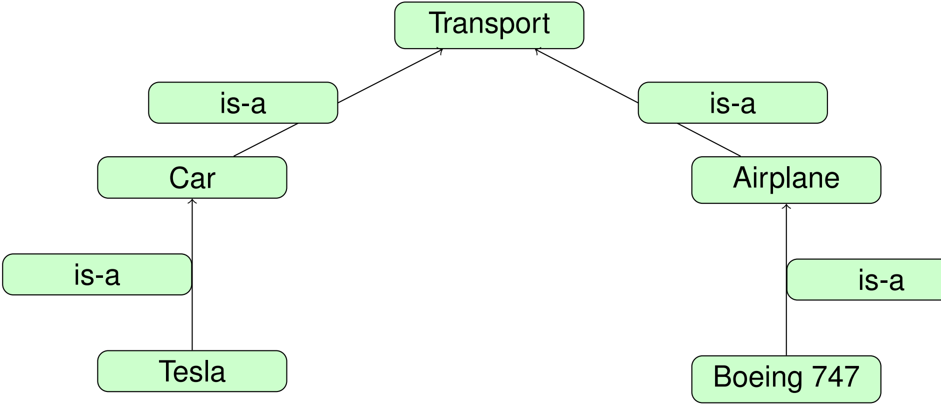
$$Red(light) \rightarrow Stop(car)$$

$$Green(light) \rightarrow Go(car)$$

Semantic Network Example



Semantic Network in Real Life



Frames Representation

Restaurant Example

- Frame: Restaurant Visit
- Slots:
  - Customer = John
  - Order = Pizza
  - Payment = Credit Card
- Helps AI reason about typical situations.

Production Rules Example

Smart Home

- IF temperature < 18℃ THEN turn on heater.
- IF temperature > 30℃ THEN turn on AC.
- IF motion detected AND time > 10 PM THEN turn on lights.

Ontologies in AI

- Provide formal definitions of concepts and relationships.
- Used in Semantic Web, medicine, and robotics.
- Example: Healthcare ontology linking diseases, symptoms, and treatments.
- Enables interoperability between systems.

Real-World Applications of KR

- **Google Knowledge Graph** – Represents entities and relationships.
- **Siri / Alexa** – Use ontologies and rules for natural language.
- **Medical Expert Systems** – Represent knowledge of symptoms and diseases.
- **Autonomous Vehicles** – Represent environment, obstacles, and road rules.

Logic in AI

What is Logic in AI?

- Logic provides a **formal framework** to represent knowledge and reason about it.
- Used in AI for:
  - Automated reasoning
  - Decision making
  - Verifying correctness of systems
- Two major types:
  1. Propositional Logic
  2. First Order Logic (FOL)

Propositional Logic

Definition

- Deals with **statements (propositions)** that are either True (T) or False (F).
- Basic operators:
  - AND ($\wedge$), OR ($\vee$), NOT ($\neg$), IMPLIES ($\rightarrow$).

Propositional Logic – Example

- P: It is raining.
- Q: I will carry an umbrella.
- Rule: $P \rightarrow Q$ (If it rains, I will carry an umbrella).
- **Real-world case:**
  - If the weather app says rain = TRUE,
  - then my action (umbrella) = TRUE.

Truth Table for $P \rightarrow Q$

P	Q	$P \rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

First Order Logic (FOL)

- Extends propositional logic by allowing:
    - **Objects** (e.g., Socrates, Earth)
    - **Predicates** (e.g., Human(x), Mortal(x))
    - **Quantifiers:**
        - Universal: $\forall$ (“for all”)
        - Existential: $\exists$ (“there exists”)

Example: All Humans are Mortal

- Knowledge: “All humans are mortal.”
- Representation:

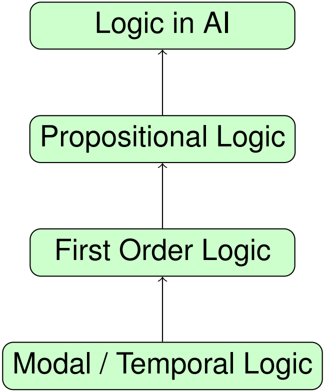
$$\forall x \; Human(x) \rightarrow Mortal(x)$$

- Fact: $Human(Socrates)$
- Inference: $Mortal(Socrates)$
- **Real-world analogy:** Doctors apply this reasoning in diagnosis rules: “If patient has disease X, then prescribe medicine Y.”

FOL in Real Life: Google Maps

- Example rules in navigation:
  - $\forall x\ Road(x) \rightarrow Accessible(x)$
  - $\exists x\ Road(x) \wedge Blocked(x)$
- AI Reasoning:
  - If a road is blocked, system searches for alternate.
  - If road = highway → higher speed.

Hierarchy of Logic in AI



Summary – Logic in AI

- **Propositional Logic:** Simple true/false statements.
- **First Order Logic:** Objects, predicates, quantifiers.
- Enables reasoning in real-world AI:
  - Medical diagnosis
  - Navigation systems
  - Expert systems

Reasoning Under Uncertainty – Bayesian Networks

Why Reason Under Uncertainty?

- The real world is **uncertain and noisy**.
- Sensors, measurements, and data may be incomplete or ambiguous.
- Agents must still make decisions despite uncertainty.
- **Examples:**
  - Medical diagnosis: symptoms may not always indicate a single disease.
  - Weather forecasting: predictions are probabilistic.
  - Robotics: sensor data often contains noise.

Real-World Example: Medical Diagnosis

Problem

A doctor observes symptoms like fever and cough. Possible causes could be:

- Common cold
- Flu
- Pneumonia

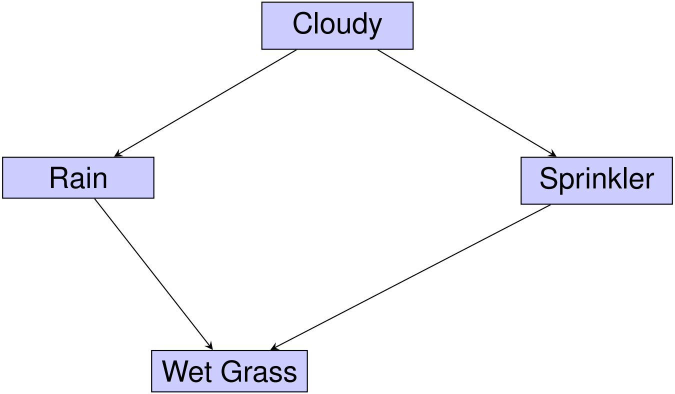
- Bayesian networks allow reasoning with probabilities:
  - $P(\text{Disease}|\text{Symptoms})$
  - Example: $P(\text{Flu}|\text{Fever, Cough}) = 0.65$

Bayesian Networks

Definition

- A **graphical model** representing probabilistic relationships among variables.
- Structure: **Directed Acyclic Graph (DAG)**.
- Each node represents a random variable.
- Each edge encodes probabilistic dependency.
- Each node has a Conditional Probability Table (CPT).

Bayesian Network Example: Weather



Example: If it is cloudy, both **rain** and **sprinkler** events become more likely.

Probability Table Example – Cloudy

Cloudy	$P(Cloudy)$
True	0.5
False	0.5

- Probability of cloudy weather is equally likely (50%).
- This prior affects the likelihood of rain and sprinkler usage.

Probability Table Example – Rain

Rain	Cloudy	$P(Rain \mid Cloudy)$
T	T	0.8
T	F	0.2
F	T	0.2
F	F	0.8

- If it is cloudy, rain is more likely (80%).
- If it is not cloudy, rain probability is low (20%).

Inference in Bayesian Networks

- Main goal: compute probability of unknown variables given evidence.
- Example:
  - Evidence: Wet Grass = True
  - Query: $P(\text{Rain}|\text{Wet Grass})$
- The network allows probabilistic reasoning:
  - Wet grass can be due to rain or sprinkler.
  - Bayesian inference computes exact probabilities.

Real-World Example: Self-Driving Cars

Problem

Self-driving car detects obstacles with sensors that may be noisy.

- Bayesian networks fuse information from:
  - LIDAR
  - Cameras
  - Radar
- Example: If LIDAR shows 60% chance of obstacle, camera shows 70%, Bayesian inference combines these for a more reliable decision.

Expert Systems and Rule-Based Systems

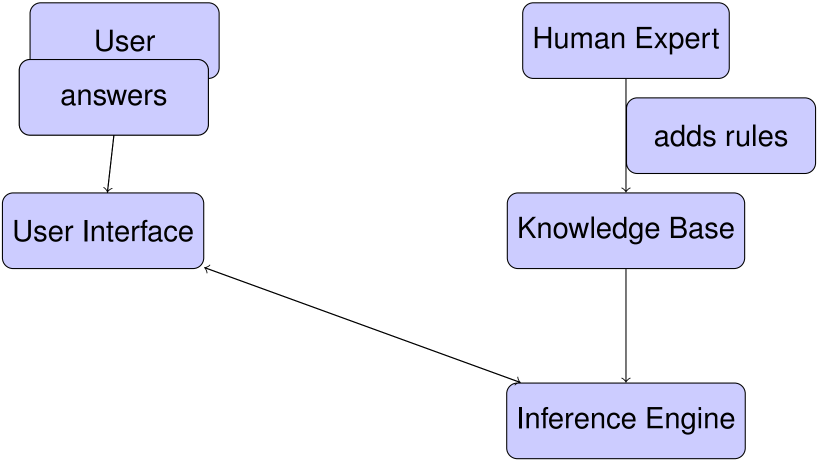
What is an Expert System?

- A computer program that simulates the reasoning of a human expert.
- Works in specialized domains: medicine, engineering, troubleshooting.
- Components:
  - **Knowledge Base (KB)** – facts + rules.
  - **Inference Engine (IE)** – logical reasoning mechanism.
- Examples:
  - **MYCIN** – medical diagnosis of infections.
  - **DENDRAL** – chemical analysis of molecular structures.

Why Expert Systems?

- Capture scarce expertise and make it widely available.
- Help in decision-making under complex scenarios.
- Consistency in applying rules, unlike human experts who may vary.
- Real-world uses:
    - Medical diagnosis.
    - Fault detection in aircraft and power systems.
    - Customer support chatbots.

Expert System Architecture



Rule-Based Systems

- Represent knowledge as **IF-THEN rules**.
- Reasoning process:
    - Start with known facts.
    - Apply matching rules.
    - Derive new conclusions.
- Example:
    - Rule: IF fever & cough THEN flu.
    - Fact: fever = true, cough = true.
    - Conclusion: flu = true.

Real-World Example: Medical Expert System

- **MYCIN** (1970s) – diagnosed blood infections.
- Rule example:
  - IF patient has fever AND stiff neck
    THEN suggest meningitis.
- Benefits:
  - Suggested treatments more consistent than junior doctors.
  - Provided explanations for each decision.

Reasoning Mechanisms

Forward Chaining

- Data-driven reasoning.
- Start from facts → apply rules → derive conclusion.
- Example: Expert system in medical monitoring.

Reasoning Mechanisms

Forward Chaining

- Data-driven reasoning.
- Start from facts → apply rules → derive conclusion.
- Example: Expert system in medical monitoring.

Backward Chaining

- Goal-driven reasoning.
- Start from a hypothesis → check which rules support it.
- Example: MYCIN checking if a patient has a specific infection.

Sample Rule Base (Simplified)

Rule ID	Rule Description
R1	IF fever AND cough THEN flu
R2	IF headache AND stiff neck THEN meningitis
R3	IF chest pain AND short breath THEN heart issue
R4	IF sore throat AND runny nose THEN cold

Advantages and Limitations

Advantages

- Consistency in decisions.
- Stores expertise for training and knowledge transfer.
- Provides explanations for reasoning steps.

Limitations

- Narrow domain knowledge (not general AI).
- Requires extensive rule collection from experts.
- Difficult to maintain with large knowledge bases.

Part 4: Machine Learning Basics

Engr. Dr. Muhammad Siddique

HOD, Department of Artificial Intelligence
NFC IET Multan

August 31, 2025

Outline of Presentation

1. Introduction to Machine Learning
2. Supervised Learning: Email Spam Classification
3. Unsupervised Learning: Customer Segmentation
4. Reinforcement Learning: Robotic Navigation
5. Supervised Learning
6. Unsupervised Learning
7. Reinforcement Learning
8. Evaluation Metrics

Introduction to Machine Learning

What is Machine Learning?

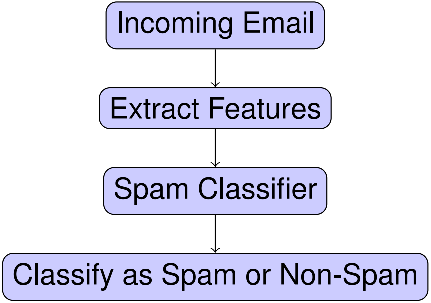
- **Definition:** A field of AI where systems learn from data and improve performance without explicit programming.
- Learns **patterns and rules** from examples.
- Machine learning enables systems to make decisions, predictions, or classifications based on data.

Applications of Machine Learning

- **Speech Recognition:** Siri, Alexa, Google Assistant.
- **Image Recognition:** Identifying objects or faces in photos.
- **Medical Diagnosis:** Identifying diseases based on medical images or patient data.
- **Recommendation Systems:** Netflix, YouTube, Amazon recommendations.
- **Robotics:** Autonomous navigation in robots.

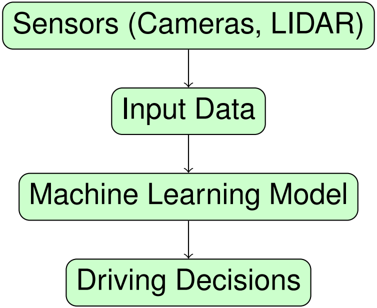
Real-World Example 1: Email Spam Filtering

- A machine learning model is trained on labeled emails (spam vs non-spam).
- The model learns patterns based on the features such as sender, subject line, and body content.
- The model classifies incoming emails as either ”spam” or ”non-spam.”

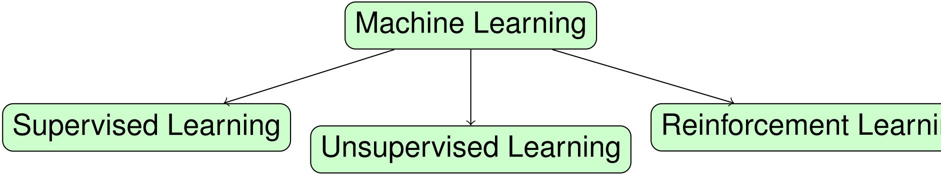


Real-World Example 2: Self-Driving Cars

- Self-driving cars use sensors (cameras, LIDAR, radar) to collect data about the environment.
- Machine learning models process this data to make decisions about steering, acceleration, and braking.
- The car learns from real-world data and improves its driving behavior over time.



Types of Machine Learning



Machine Learning

Supervised Learning

Unsupervised Learning

Reinforcement Learni

Supervised Learning

- In **Supervised Learning**, the model is trained on labeled data.
- The goal is to learn a mapping from input to output.
- Example: Classifying emails as spam or non-spam based on labeled examples.

Unsupervised Learning

- **Unsupervised Learning** deals with unlabeled data.
- The goal is to find hidden patterns or groupings in the data.
- Example: Clustering customers based on purchasing behavior.

Reinforcement Learning

- In **Reinforcement Learning**, agents learn by interacting with the environment.
- The goal is to maximize cumulative rewards through actions.
- Example: A robot learning to navigate a maze by receiving rewards for reaching goals.

Evaluation of Machine Learning Models

- **Accuracy:** The proportion of correct predictions.
- **Precision and Recall:** Used for imbalanced data to measure positive predictions.
- **F1 Score:** A balance between precision and recall.

Supervised Learning: Email Spam Classification

Supervised Learning: Problem Setup

- **Problem:** Classify emails as ”spam” or ”non-spam.”
- **Training Data:** Labeled emails with the following features: subject, sender, and email content.
- Example Data:
  - Email 1: ”Get free tickets now!” (spam)
  - Email 2: ”Your Amazon order has shipped” (non-spam)
  - Email 3: ”Limited time offer on shoes” (spam)

Python Code: Supervised Learning with Logistic Regression

```python
# Supervised Learning: Email Spam Classification using Logistic Regression

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Sample dataset (emails and labels)
emails = ["Get free tickets now!", "Your Amazon order has shipped", "Limited time offer on shoes"]
labels = [1, 0, 1]  # 1: spam, 0: non-spam
```

Python Code: Supervised Learning with Logistic Regression

```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    emails, labels, test_size=0.33, random_state=42)
# Convert text data to numerical features
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
# Train Logistic Regression model
model = LogisticRegression()
model.fit(X_train_vec, y_train)
# Predictions
y_pred = model.predict(X_test_vec)
# Evaluate Model
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Unsupervised Learning: Customer Segmentation

Unsupervised Learning: Problem Setup

- **Problem:** Group customers based on purchasing behavior.
- **Training Data:** Unlabeled customer data with features such as age, income, and purchase history.
- Example Data:
  - Customer 1: Age 25, Income $50,000, \mathit{Buyselectronics}.\mathit{Customer}2 : \mathit{Age}45, \mathit{Income}$100,000, Buys home appliances.

- Customer 3: Age 30, Income $55,000, \mathit{Buysclothing}$.

Python Code: Unsupervised Learning with K-Means Clustering

```python
# Unsupervised Learning: Customer Segmentation using K-Means Clustering
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
# Sample customer data: Age, Income
data = np.array([[25, 50000], [45, 100000], [30, 55000], [50, 120000], [23, 45000], [35, 70000]])
# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(data)
```

Python Code: Unsupervised Learning with K-Means Clustering

```
# Predictions
labels = kmeans.predict(data)
# Plotting the data points and clusters
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap=’viridis’)
plt.title(’Customer Segmentation using K−Means’)
plt.xlabel(’Age’)
plt.ylabel(’Income’)
plt.show()
```

Reinforcement Learning: Robotic Navigation

Reinforcement Learning: Problem Setup

- **Problem:** A robot learns to navigate a maze by receiving rewards.
- **State:** The robot's current position in the maze.
- **Action:** Move forward, turn left, turn right.
- **Reward:** A positive reward for reaching the goal, negative for hitting obstacles.

Python Code: Reinforcement Learning with Q-Learning

```python
# Reinforcement Learning: Robotic Navigation using Q-Learning
import numpy as np
import random
# Define the environment (grid world)
grid_size = 5
env = np.zeros((grid_size, grid_size))
goal = (4, 4)  # Goal position
# Q-table initialization (state, action -> q-value)
q_table = np.zeros((grid_size, grid_size, 4))  # 4 actions: up, down, left, right
# Action mappings: 0: Up, 1: Down, 2: Left, 3: Right
actions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
# Hyperparameters
learning_rate = 0.1
discount_factor = 0.9   # epsilon = 0.1
```

Python Code: Reinforcement Learning with Q-Learning

```python
# Q-Learning Algorithm
def q_learning():
    state = (0, 0)  # Start at the top-left corner
    for episode in range(1000):
        if random.uniform(0, 1) < epsilon:
            action = random.choice([0, 1, 2, 3])  # Explore random action
        else:
            action = np.argmax(q_table[state[0], state[1]])  # Exploit best action
    # Move to next state
        next_state = (state[0] + actions[action][0], state[1] + actions[action][1])
        next_state = (max(0, min(next_state[0], grid_size-1)), max(0, min(next_state[1], grid_size-1)))  # Bound check
```

Python Code: Reinforcement Learning with Q-Learning

```
        # Reward: 10 for reaching the goal, -1 for every step
        reward = 10 if next_state == goal else -1
        q_table[state[0], state[1], action] = q_table[state[0], state[1], action] + learning_rate * (reward + discount_factor * np.max(q_table[next_state[0], next_state[1]]) - q_table[state[0], state[1], action])

        state = next_state

    return q_table
# Train the agent
q_table = q_learning()

# Display Q-table
print("Final Q-Table:")
```

Supervised Learning

- Learns from **labeled data** (input-output pairs).
- Goal: Predict the output for unseen inputs.
- Example: Predicting house prices based on features.

Algorithms in Supervised Learning

- Linear Regression
- Logistic Regression
- Decision Trees
- Support Vector Machines (SVM)
- Neural Networks

Example: Spam Detection

- Input: Email text
- Output: Spam or Not Spam
- Training Data: Emails labeled by humans
- Algorithm learns patterns in spam emails

Unsupervised Learning

- Works with **unlabeled data.**
- Goal: Discover hidden patterns or structure.
- Two main tasks:
  - Clustering (grouping data points).
  - Dimensionality Reduction (compressing data).

Clustering Example

- Market segmentation in business.
- Example: Grouping customers by purchasing behavior.

Dimensionality Reduction Example

- Compressing images for storage.
- Reducing redundant features in datasets.

Reinforcement Learning (RL)

- Learning by **trial and error**.
- Agent interacts with environment.
- Learns to maximize cumulative reward.
- Examples: AlphaGo, autonomous robots.

RL Terminology

- Agent: Learner/decision maker.
- Environment: World where the agent acts.
- Action: Choices made by agent.
- Reward: Feedback signal.
- Policy: Strategy mapping states to actions.

RL Example: Self-Driving Car

- Actions: Accelerate, brake, turn.
- Environment: Road conditions, traffic.
- Rewards: Safe driving, reaching destination quickly.

Evaluation Metrics in Machine Learning

Why Evaluation Metrics?

- Machine Learning models must be **evaluated** to ensure reliability.
- A single metric (e.g., accuracy) is often **misleading**.
- Example: In medical diagnosis (rare disease detection), accuracy may be high even if the model fails to detect actual cases.
- Different tasks (classification, regression, ranking) require different evaluation metrics.

Accuracy – Example

- **Definition:** Proportion of correct predictions.
- Works well when classes are balanced.

Python Example:

```python
from sklearn.metrics import accuracy_score

y_true = [1, 0, 1, 1, 0, 1, 0, 0]   # Actual
y_pred = [1, 0, 1, 0, 0, 1, 0, 1]   # Predicted

print("Accuracy:", accuracy_score(y_true, y_pred))
```

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP) = 3	False Negative (FN) = 1
Actual Negative	False Positive (FP) = 1	True Negative (TN) = 3

Interpretation: Helps us compute precision, recall, F1-score.

Precision

- **Definition:** Fraction of predicted positives that are actually correct.

$$Precision = \frac{TP}{TP + FP}$$

- Useful in scenarios like **spam detection** where false positives (important email marked as spam) are costly.

Python Example:

```python
from sklearn.metrics import precision_score

print("Precision:", precision_score(y_true, y_pred))
```

Recall (Sensitivity)

- **Definition:** Fraction of actual positives correctly identified.

$$Recall = \frac{TP}{TP + FN}$$

- Useful in **medical diagnosis** where missing a disease case is very risky.

Python Example:

```
from sklearn.metrics import recall_score

print("Recall:", recall_score(y_true, y_pred))
```

F1-Score

- **Definition:** Harmonic mean of Precision and Recall.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- Useful when data is imbalanced.

Python Example:

```python
from sklearn.metrics import f1_score

print("F1-Score:", f1_score(y_true, y_pred))
```

ROC Curve and AUC

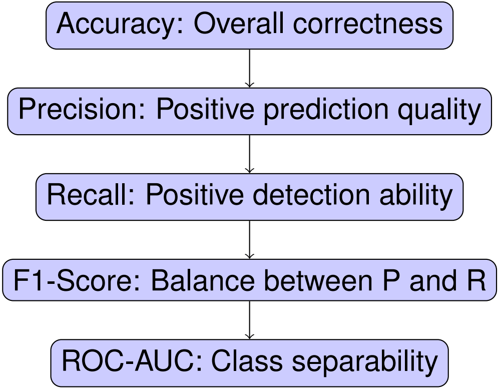
- **ROC Curve:** Plots True Positive Rate vs. False Positive Rate at different thresholds.
- **AUC (Area Under Curve):** Measures how well the model distinguishes between classes.
- Higher AUC = Better performance.

Python Example:

```python
from sklearn.metrics import roc_auc_score

y_prob = [0.9, 0.2, 0.8, 0.4, 0.1, 0.7, 0.3, 0.6] # predicted probs
print("ROC-AUC:", roc_auc_score(y_true, y_prob))
```

Summary of Metrics



Accuracy: Overall correctness

Precision: Positive prediction quality

Recall: Positive detection ability

F1-Score: Balance between P and R

ROC-AUC: Class separability

Foundations of Artificial Intelligence

Neural Networks and Backpropagation

Engr. Dr. Muhammad Siddique

HOD, Department of Artificial Intelligence
NFC IET Multan

August 31, 2025

Outline of Presentation

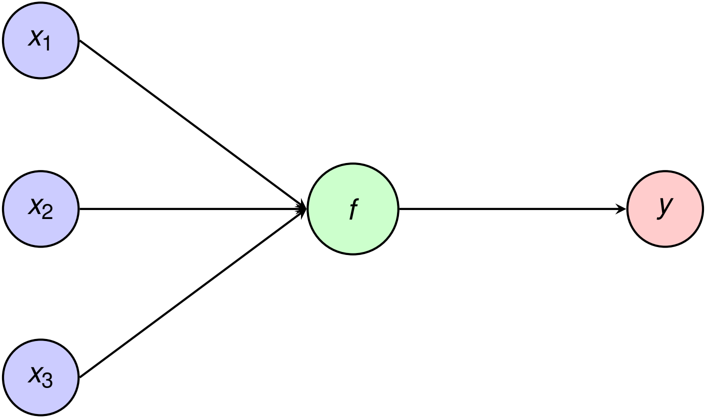
1. Introduction to Neural Networks
2. Training Neural Networks – Backpropagation
3. Introduction to Neural Networks
4. Backpropagation

Introduction to Neural Networks

Why Neural Networks?

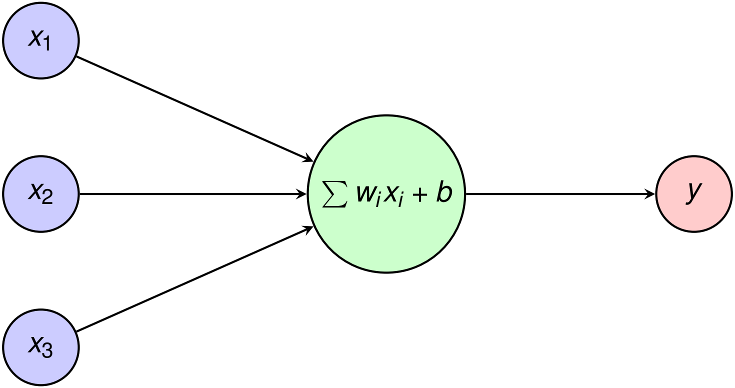
- Inspired by the human brain's network of neurons.
- Able to learn complex patterns and decision boundaries.
- Widely used in:
    - Image recognition.
    - Natural language processing.
    - Autonomous driving.

Artificial Neuron

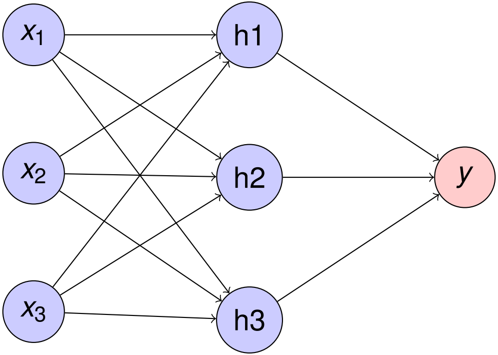


$$y = f\left(\sum_i w_i x_i + b\right)$$

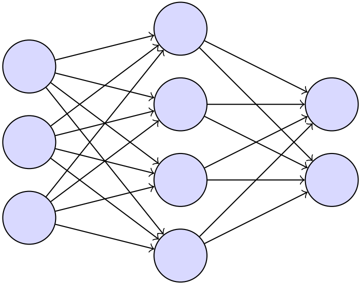
Neuron Structure (Perceptron)



Neural Network Architecture



Neural Network Layers



Training Neural Networks – Backpropagation

Forward Pass

- Compute outputs layer by layer:

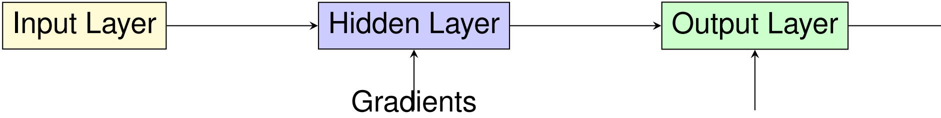
$$a^{(l)} = f(W^{(l)}a^{(l-1)} + b^{(l)})$$

- Final output compared with target using a **loss function.**

$$L = \frac{1}{2}(y - \hat{y})^2$$

Backpropagation Intuition

- Loss tells us how wrong the prediction is.
- Backpropagation computes **gradients** of loss w.r.t weights.
- Gradients are used to update weights via **gradient descent**.



Weight Update Rule

- Using Gradient Descent:

$$w := w - \eta \frac{\partial L}{\partial w}$$

  where η is the **learning rate.**
- Iteratively reduces the loss and improves model performance.

Python Example: Training a Neural Network

```python
from sklearn.neural_network import MLPClassifier

# Dummy dataset
X = [[0,0],[0,1],[1,0],[1,1]]
y = [0, 1, 1, 0]  # XOR problem

# Neural network with 1 hidden layer
clf = MLPClassifier(hidden_layer_sizes=(3,), max_iter=1000)

clf.fit(X, y)

print("Predictions:", clf.predict(X))
```

Introduction to Neural Networks

What is a Neural Network?

- A computational model inspired by the human brain.
- Consists of interconnected units called **neurons**.
- Each neuron processes input, applies weights, adds bias, and passes through an activation function.

Real-World Examples

- Image recognition: Classifying pictures of cats and dogs.
- Natural Language Processing: Sentiment analysis of tweets.
- Finance: Predicting stock price movements.
- Healthcare: Detecting tumors in MRI scans.

Python Example: Neural Network for AND Gate

```python
# Step 1: Import necessary libraries
import numpy as np
from sklearn.neural_network import MLPClassifier
```

Explanation

We use:

- `numpy` for handling numerical data.
- `MLPClassifier` from `scikit-learn` to build a simple neural network.

Preparing the Dataset

```python
# Step 2: Define dataset for AND logic gate
X = [[0,0], [0,1], [1,0], [1,1]]  # Input combinations
y = [0, 0, 0, 1]                   # Output labels
```

Explanation

- The dataset represents the truth table of the AND gate.
- Inputs (X) are all possible binary pairs.
- Output (y) is 1 only when both inputs are 1.

Building the Neural Network

```
# Step 3: Create the Neural Network Model
model = MLPClassifier(
    hidden_layer_sizes=(3,), # One hidden layer with 3 neurons
    activation=’relu’,       # ReLU activation function
    max_iter=1000,           # Maximum training iterations
    random_state=42
)
```

Explanation

- One hidden layer with 3 neurons is used.
- `relu` activation introduces non-linearity.
- The network will iterate up to 1000 times to converge.

Training the Neural Network

```
# Step 4: Train the model on the dataset
model.fit(X, y)
```

Explanation

- The model adjusts its weights using backpropagation.
- It minimizes the error between predicted and actual labels.
- After training, the model can be used for prediction.

Making Predictions

```python
# Step 5: Test the model
predictions = model.predict(X)
print("Predictions:", predictions)
```

Explanation

- We test the trained model on the same dataset.
- Expected output: `[0, 0, 0, 1]`.
- This matches the AND gate logic.

Model Output

```
Predictions: [0 0 0 1]
```

Conclusion

- The neural network successfully learned the AND gate behavior.
- Even small networks can approximate logical operations.
- In real-world tasks, datasets are larger and more complex, but the principle remains the same.

AND Gate Truth Table

AND Logic Function

The AND gate outputs 1 only if both inputs are 1. Otherwise, it outputs 0.

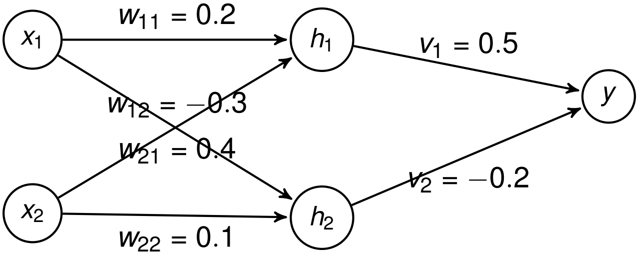
Input x_1	Input x_2	Output y
0	0	0
0	1	0
1	0	0
1	1	1

Step 1: Initialization of Neural Network

Description

We start with:

- 2 input neurons (x_1, x_2)
- 2 hidden neurons (h_1, h_2)
- 1 output neuron (y)
- Random initial weights between layers



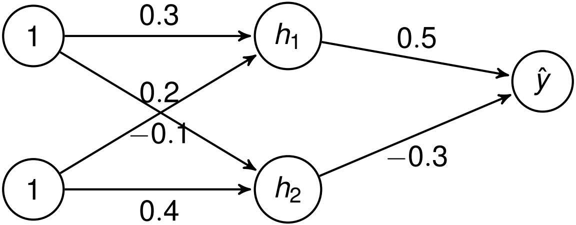
Step 2: Forward Pass

Description

For input $[x_1, x_2] = [1, 1]$:

$$h_1 = f(x_1 w_{11} + x_2 w_{21}), \quad h_2 = f(x_1 w_{12} + x_2 w_{22})$$

$$\hat{y} = f(h_1 v_1 + h_2 v_2)$$



Forward Pass Example: Input (1,1)

Step 1: Input Values

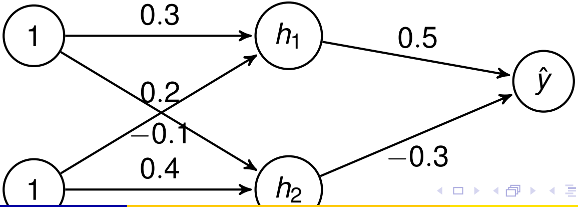
$x_1 = 1, \; x_2 = 1 \quad \Rightarrow \quad y = 1$

Step 2: Hidden Layer Activations

$$h_1 = f(1 \cdot 0.3 + 1 \cdot 0.2) = f(0.5)$$
$$h_2 = f(1 \cdot -0.1 + 1 \cdot 0.4) = f(0.3)$$

Step 3: Output Neuron

$$\hat{y} = f(0.5 \cdot h_1 + (-0.3) \cdot h_2)$$



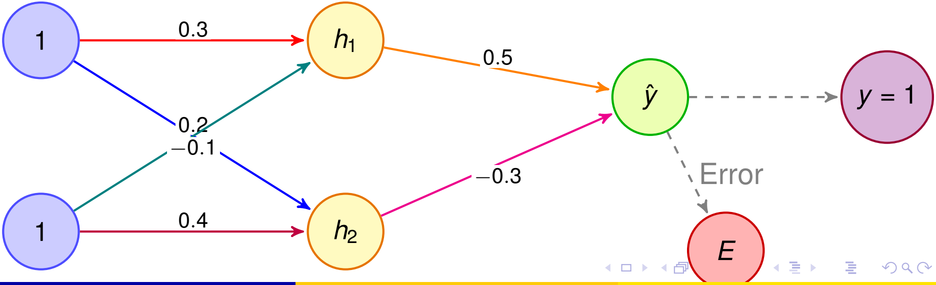
Step 3: Error Calculation

Description

The error is computed as:

$$E = \frac{1}{2}(y_{true} - \hat{y})^2$$

For AND gate, if input is $[1, 1]$, then $y_{true} = 1$.



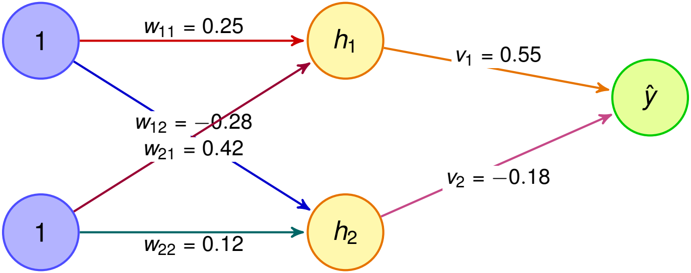
Step 4: Backpropagation and Weight Update

Description

Weights are updated using gradient descent:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

where η is the learning rate.

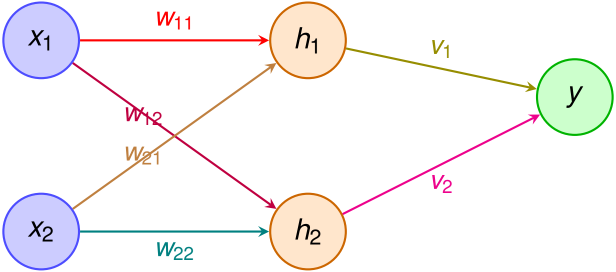


Backpropagation: High-Level Idea

Backpropagation

- Train the network by comparing predicted output with the target.
- The difference is the **error**, which must be corrected.
- Backpropagation shares this error with earlier layers
  - Output → Hidden: proportional blame split.
  - Hidden → Input: repeated recursively.

Simple 3-Layer Network



Step 1: Forward Pass

$$z_h = W \cdot x, \quad h = f(z_h)$$
$$z_o = V \cdot h, \quad y = f(z_o)$$

Explanation

- Inputs $x = [x_1, x_2]^T$.
- Hidden activations $h = [h_1, h_2]^T$.
- W = input → hidden weight matrix.
- V = hidden → output weight row.

Step 2: Output Error

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

$$\delta_o = e \cdot f'(z_o)$$

Interpretation

- Output error is the gap between target and prediction.
- Multiplied by derivative $f'(z_o)$ to respect slope of sigmoid/tanh.
- This gives the **output delta**.

Step 3: Distribute Error to Hidden Layer

$$\delta_h = f'(z_h) \cdot (V^T \cdot \delta_o)$$

Explanation

- Error is sent backward through the weights.
- Each hidden unit's error is proportional to the weight of its link to the output.
- Multiply by $f'(z_h)$ to adjust for hidden activation slope.

Step 4: Update Rules

$$\blacksquare V = \eta \cdot \delta_o \cdot h^T$$

$$\blacksquare W = \eta \cdot \delta_h \cdot x^T$$

Matrix Form

- η = learning rate.
- V (row vector) updated using δ_o and hidden activations h.
- W (matrix) updated using δ_h and inputs x.

Numerical Example: Forward Pass

Setup

Inputs: $x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, target $t = 1$

Weights:

$$W = \begin{bmatrix} 0.4 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}, \quad V = \begin{bmatrix} 0.5 & 0.9 \end{bmatrix}$$

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

$$h = f(Wx) = f \begin{bmatrix} 0.4 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.5987 \\ 0.5498 \end{bmatrix}$$

$$y = f(Vh) = f(0.5 \cdot 0.5987 + 0.9 \cdot 0.5498) = f(0.93) \approx 0.717$$

Numerical Example: Backward Pass

$$e = 1 - 0.717 = 0.283$$

$$\delta_o = e \cdot f'(0.93) = 0.283 \cdot 0.717(1 - 0.717) \approx 0.057$$

$$\delta_h = f'(z_h) \cdot (V^T \delta_o)$$

$$= \begin{bmatrix} 0.5987(1 - 0.5987) \\ 0.5498(1 - 0.5498) \end{bmatrix} \cdot \begin{bmatrix} 0.5 \\ 0.9 \end{bmatrix} \cdot 0.057$$

$$= \begin{bmatrix} 0.240 \\ 0.247 \end{bmatrix} \cdot 0.057 = \begin{bmatrix} 0.0137 \\ 0.0141 \end{bmatrix}$$

Numerical Example: Weight Updates

$$\blacksquare V = \eta \cdot \delta_o \cdot h^T = 0.1 \cdot 0.057 \cdot \begin{bmatrix} 0.5987 & 0.5498 \end{bmatrix}$$

$$= \begin{bmatrix} 0.0034 & 0.0031 \end{bmatrix}$$

$$\blacksquare W = \eta \cdot \delta_h \cdot x^T = 0.1 \cdot \begin{bmatrix} 0.0137 \\ 0.0141 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.00137 & 0 \\ 0.00141 & 0 \end{bmatrix}$$

Result

Updated weights are obtained by adding $\blacksquare V$ and $\blacksquare W$ to original values.

Summary

- “proportional error split” = equivalent to $\delta_h = f'(z_h) \cdot (V^T \delta_o)$.
- Matrix form is compact: no proportional splitting manually.
- Example shows how errors propagate numerically.
- Key idea: forward pass computes activations, backward pass distributes blame.

Key Insights from Backpropagation

- Backpropagation allows deep networks to learn complex functions.
- Requires large datasets and computational power.
- Optimization techniques: learning rate scheduling, momentum, Adam optimizer.

Foundations of Artificial Intelligence

Neural Networks and Backpropagation

Engr. Dr. Muhammad Siddique

HOD, Department of Artificial Intelligence
NFC IET Multan

September 6, 2025

Outline of Presentation

1 Specialized AI Domains

Natural Language Processing (NLP)

Definition

Natural Language Processing (NLP) is the branch of AI that focuses on enabling machines to understand, interpret, and generate human language.

Why NLP Matters?

- Communication bridge between humans and machines
- Enables intelligent assistants, translators, and chatbots

NLP Applications

Common Applications

- Machine Translation (Google Translate)
- Sentiment Analysis (classifying opinions as Positive/Negative)
- Chatbots and Virtual Assistants (Siri, Alexa)
- Text Summarization (news highlights)

NLP Challenges

Challenges

- Ambiguity in words (e.g., *bank*)
- Context and sarcasm
- Multilingual and code-mixed text
- Handling large amounts of noisy text

Real-World Project: Sentiment Analysis

Objective

Classify movie reviews as **positive** or **negative**.

Dataset

- IMDB movie review dataset
- 50,000 reviews labeled as positive or negative

Applications

- Product feedback analysis
- Social media monitoring
- Customer satisfaction tracking

Steps in Sentiment Analysis Project

Pipeline

1. Collect and clean the text data
2. Convert text to numerical features (Bag-of-Words, TF-IDF)
3. Train a classifier (Logistic Regression / Neural Network)
4. Evaluate accuracy and performance

Python Implementation – Step 1

Import Libraries

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Python Implementation – Step 2

Load Dataset

```python
# Example dataset (IMDB reviews)
data = {
    "review": ["This movie was fantastic!",
               "Worst film I have ever seen",
               "I loved the characters and story",
               "Terrible acting and boring plot"],
    "sentiment": ["positive", "negative",
                  "positive", "negative"]
}
df = pd.DataFrame(data)
X = df["review"]
y = df["sentiment"]
```

Python Implementation – Step 3

Preprocess and Train Model

```python
# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)

# Convert text into TF-IDF features
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf  = vectorizer.transform(X_test)

# Train Logistic Regression
model = LogisticRegression()
model.fit(X_train_tfidf, y_train)
```

Python Implementation – Step 4

Evaluate Model

```python
# Predictions
y_pred = model.predict(X_test_tfidf)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# Example prediction
sample = ["The movie was wonderful and inspiring"]
sample_tfidf = vectorizer.transform(sample)
print("Prediction:", model.predict(sample_tfidf))
```

Results and Insights

Example Output

- Accuracy: **0.85** (on small dataset)
- Review: *"The movie was wonderful and inspiring"* → Prediction: **Positive**

Insights

- NLP models can quickly classify human opinions
- Scaling up to large datasets improves performance

Computer Vision – How Machines See

What is Computer Vision?

Computer Vision is a field of Artificial Intelligence (AI) that allows machines to extract, process, and analyze meaningful information from visual inputs such as images or videos. It enables machines to mimic aspects of human vision, making decisions or predictions based on visual data.

Computer Vision – Applications

Key Applications

- **Facial Recognition:** Unlocking smartphones, airport security, surveillance.
- **Medical Imaging:** Detecting tumors, X-ray and MRI analysis.
- **Autonomous Vehicles:** Lane detection, pedestrian recognition, obstacle avoidance.
- **Retail:** Automated checkout systems and product recognition.
- **Agriculture:** Crop monitoring and pest detection.

Computer Vision – Challenges

Challenges in Computer Vision

- Handling poor lighting conditions, occlusions, and noisy images.
- Requirement for large labeled datasets to train accurate models.
- Real-time processing for applications like autonomous driving.
- Generalization across different environments and conditions.
- Ethical and privacy concerns in surveillance and facial recognition.

Computer Vision Example – Image Classification

Task

Build a simple classifier to detect whether an image contains a **cat** or a **dog**.

Steps Involved

1. Collect dataset of cat and dog images.
2. Preprocess images: resize, normalize, augment.
3. Train a Convolutional Neural Network (CNN).
4. Evaluate accuracy on test dataset.

Python Code – Data Preprocessing

Step 1: Load and Preprocess Data

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data generators
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_data = datagen.flow_from_directory(
    "dataset/",
    target_size=(128, 128),
    batch_size=32,
    class_mode="binary",
    subset="training"
)

```

Python Code – Building the Model

Step 2: Define CNN Model

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3,3), activation="relu",
   input_shape=(128,128,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation="relu"),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

Python Code – Training and Evaluation

Step 3: Train and Evaluate

```python
# Train model
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=5
)

# Evaluate on validation set
loss, acc = model.evaluate(val_data)
print(f"Validation Accuracy: {acc:.2f}")
```

Python Code – Prediction Example

Step 4: Make Prediction

```python
import numpy as np
from tensorflow.keras.preprocessing import image

# Load and preprocess single image
img = image.load_img("sample.jpg", target_size=(128,128))
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)

prediction = model.predict(img_array)
if prediction[0][0] > 0.5:
    print("Prediction: Dog")
else:
    print("Prediction: Cat")
```

Speech and Voice Recognition

Definition

Speech Recognition converts spoken language into text. **Voice Recognition** identifies or verifies the speaker based on unique vocal patterns.

Applications of Speech and Voice Recognition

Key Applications

- **Virtual Assistants:** Alexa, Siri, Google Assistant.
- **Automated Transcription:** Converting meetings and lectures into text.
- **Security and Authentication:** Voice biometrics for banking and smart devices.
- **Accessibility:** Assisting visually impaired users with speech interfaces.
- **Customer Service:** Interactive Voice Response (IVR) systems.

Challenges in Speech Recognition

Challenges

- Variability in accents, dialects, and pronunciation.
- Background noise interfering with accuracy.
- Requirement for **real-time processing** in applications such as virtual assistants.
- Speaker variability: distinguishing between multiple voices in group conversations.
- Privacy concerns when storing or transmitting voice data.

Speech Recognition Example

Task

Input Audio: *”Schedule a meeting tomorrow at 10 AM”* Output Text: `"Schedule a meeting tomorrow at 10 AM"`

Real-World Applications

- Automated **meeting transcription** for corporate environments.
- **Voice-controlled smart homes** (lights, thermostats, appliances).
- **Call center automation** for customer queries.

Python Implementation – Step 1

Install and Import Libraries

```python
# Install if not already available:
# pip install SpeechRecognition pyttsx3 pyaudio

import speech_recognition as sr
import pyttsx3

# Initialize recognizer and TTS engine
recognizer = sr.Recognizer()
engine = pyttsx3.init()
```

Python Implementation – Step 2

Capture Speech from Microphone

```
with sr.Microphone() as source:
    print("Say something...")
    audio = recognizer.listen(source)

try:
    # Convert speech to text using Google Web API
    text = recognizer.recognize_google(audio)
    print("You said:", text)
except sr.UnknownValueError:
    print("Sorry, I could not understand the audio.")
except sr.RequestError:
    print("Could not request results; check connection.")
```

Python Implementation – Step 3

Voice Feedback (Text-to-Speech)

```python
# Convert recognized text back to speech
engine.say(f"You said: {text}")
engine.runAndWait()
```

Use Case

This allows building a **voice assistant** that listens, interprets, and responds back to the user with spoken feedback.

Python Implementation – Example Project

Real-World Project: Voice Command Assistant

```python
# Simple voice command assistant
commands = {
    "hello": "Hello! How can I help you today?",
    "time": "I cannot tell time yet, but soon I will!",
    "bye": "Goodbye! Have a great day."
}

with sr.Microphone() as source:
    print("Listening for command...")
    audio = recognizer.listen(source)
    text = recognizer.recognize_google(audio).lower()

    if text in commands:
        response = commands[text]
    else:
        response = "Sorry, I didn’t understand that."
```

Robotics and AI in Automation

Definition

Robotics uses **Artificial Intelligence (AI)** to enable machines to perform tasks autonomously by perceiving their environment, making intelligent decisions, and executing precise actions.

Core Capabilities

- **Perception:** Sensors, cameras, LiDAR for environmental awareness.
- **Decision-Making:** AI algorithms for planning and reasoning.
- **Action:** Robotic arms, drones, or wheels for task execution.

Applications of Robotics and AI

Industrial Robotics

- Automated assembly lines
- Packaging and quality inspection

Service Robotics

- Autonomous delivery robots
- Cleaning and healthcare assistance

Exploration Robotics

- Drones for surveillance and agriculture
- Space exploration (Mars rovers)

Challenges in Robotics and AI

Key Challenges

- **Safety & Ethics:** Preventing harm to humans and ethical use of autonomous systems.
- **Dynamic Environments:** Adapting to unpredictable real-world conditions.
- **Human–Robot Collaboration:** Ensuring robots and humans work together seamlessly.
- **Cost and Maintenance:** Building affordable and sustainable robotic solutions.

Robotics Example – Warehouse Automation

Scenario

AI-driven robots in warehouses manage:

- Picking and placing items using robotic arms.
- Route optimization for delivery within the warehouse.
- Real-time inventory tracking with sensors and AI.

Benefits

- Reduced operational costs.
- Faster and error-free order fulfillment.
- Increased efficiency and scalability.

Real-World Robotics Project

Objective

Build a **simple AI-driven warehouse robot simulator** that can:

1. Detect objects (packages) using vision.
2. Pick items virtually.
3. Plan and execute a route for delivery.

Python Implementation – Step 1

Import Libraries

```
# Install required libraries:
# pip install opencv-python numpy

import cv2
import numpy as np

# Simulated warehouse floor
warehouse = np.zeros((500, 500, 3), dtype=np.uint8)
```

Python Implementation – Step 2

Simulate Packages and Robot

```python
# Draw packages (red squares)
for x in [100, 300, 400]:
    cv2.rectangle(warehouse, (x, 200), (x+30, 230), (0,0,255), -1)

# Draw robot (blue circle at start)
robot_pos = (50, 450)
cv2.circle(warehouse, robot_pos, 20, (255,0,0), -1)
```

Python Implementation – Step 3

Path Planning Simulation

```python
# Simulate simple path planning
for step in range(50, 400, 10):
    frame = warehouse.copy()
    # Move robot along path
    cv2.circle(frame, (step, 450), 20, (255,0,0), -1)
    cv2.imshow("Warehouse Robot", frame)
    cv2.waitKey(100)
cv2.destroyAllWindows()
```

Python Implementation – Step 3

Explanation

This simple project demonstrates:

- Package detection (red boxes).
- Robot navigation (blue circle movement).
- Foundation for advanced AI robotics (object detection + reinforcement learning).

Part 7 – Tools & Practical Aspects

Practical AI Development with Python

Engr. Dr. Muhammad Siddique

Department of Artificial Intelligence
NFC IET Multan

September 6, 2025

Outline of Part 7

1. Popular AI Tools and Frameworks
2. Building Your First AI Model in Python
3. Datasets and Data Preprocessing
4. AI Project Workflow

AI Development with Python

Why Python?

- Simple syntax, easy to learn.
- Huge ecosystem of AI and ML libraries.
- Strong community support.

Popular Libraries

- **NumPy** – numerical computations
- **Pandas** – data manipulation
- **Matplotlib/Seaborn** – visualization

Deep Learning Frameworks

TensorFlow

- Developed by Google.
- Flexible, production-ready deep learning platform.

PyTorch

- Developed by Facebook AI Research.
- Dynamic computation graphs, widely used in research.

Scikit-Learn

- Classical ML algorithms: regression, SVM, clustering.
- Great for beginners and small projects.

Example: Iris Flower Classification

Objective

Classify flowers into three species: *Setosa, Versicolor, Virginica*.

Why this dataset?

- Small, well-structured dataset.
- Widely used for ML beginners.

Python Implementation – Step 1

Import Libraries

```python
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Python Implementation – Step 2

Load Dataset

```python
iris = load_iris()
X = iris.data          # features: sepal & petal measurements
y = iris.target        # labels: 0=Setosa, 1=Versicolor, 2=Virginica

# Split into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)
```

Python Implementation – Step 3

Train Model

```python
# Train Logistic Regression
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

Python Implementation – Step 4

Evaluate Model

```
# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# Example Prediction
sample = [[5.1, 3.5, 1.4, 0.2]] # Sepal & Petal size
print("Prediction:", model.predict(sample))
```

Results

- Accuracy: 95% (on test split)
- Sample Prediction: Class 0 = Setosa

Datasets for AI

Sources of Data

- Public datasets (Kaggle, UCI ML Repository).
- Company databases.
- Data collected via IoT, sensors, or web scraping.

Dataset Types

- Structured (tables, CSVs).
- Unstructured (images, text, audio).

Data Preprocessing

Key Steps

- Handle missing values.
- Normalize/scale numerical data.
- Encode categorical variables.
- Split into train/test sets.

Why Important?

Poor data quality leads to poor model performance.

AI Project Workflow – From Idea to Implementation

Stages

1. **Problem Definition:** What do we want AI to solve?
2. **Data Collection:** Gather relevant datasets.
3. **Data Preprocessing:** Clean, transform, and prepare data.
4. **Model Building:** Choose algorithm and train.
5. **Evaluation:** Test performance on unseen data.
6. **Deployment:** Integrate model into real-world applications.

Example Workflow – Sentiment Analysis Project

Steps

1. Collect tweets about a product.
2. Preprocess (remove stopwords, clean text).
3. Convert text into numerical features (TF-IDF).
4. Train Logistic Regression / Neural Network.
5. Deploy as a web app for real-time monitoring.

Real-World Usage

- Monitor customer satisfaction.
- Identify product issues quickly.

Summary of Part 7

Key Takeaways

- Python provides powerful AI tools (TensorFlow, PyTorch, Scikit-Learn).
- Building a first AI model requires dataset, preprocessing, training, and evaluation.
- Real-world datasets require cleaning before modeling.
- AI projects follow a structured workflow from idea to deployment.

Part 8 – Ethical & Future Aspects

Ethics, Safety, Future, and Careers in AI

Engr. Dr. Muhammad Siddique

Department of Artificial Intelligence
NFC IET Multan

September 6, 2025

Outline of Part 8

1. Ethics of Artificial Intelligence
2. AI Safety and Risks
3. The Future of Artificial Intelligence
4. Career Opportunities in AI

Why AI Ethics Matters

Definition

AI Ethics is the study of moral principles and social values guiding the development and deployment of AI systems.

Key Ethical Principles

- **Fairness** – avoid bias and discrimination.
- **Transparency** – explainable decision-making.
- **Accountability** – responsibility for AI outcomes.
- **Privacy** – protecting user data.

Case Study: Facial Recognition Bias

Problem

Some facial recognition systems misclassify minority groups at a much higher rate.

Implications

- Risk of wrongful arrests.
- Discrimination in hiring or services.

Lesson

AI must be trained on diverse, representative datasets.

Safety Concerns

Technical Risks

- Autonomous systems behaving unpredictably.
- Adversarial attacks (tricking AI with manipulated inputs).

Societal Risks

- Job displacement due to automation.
- Misinformation spread (deepfakes, bots).

Case Study: Autonomous Vehicles

Scenario

Self-driving cars must decide between multiple risky options (e.g., collision avoidance in crowded streets).

Safety Challenges

- Sensor failures.
- Ethical dilemma: protecting passengers vs. pedestrians.

Takeaway

Rigorous testing and clear ethical guidelines are essential.

Emerging Trends

Research Directions

- General AI – systems that can perform a wide range of tasks.
- AI + Quantum Computing.
- AI for climate change (smart grids, energy optimization).

Opportunities

- Smarter healthcare (personalized treatments).
- Safer transport (autonomous mobility).
- Advanced human-computer interaction.

AI in 2030 – Predictions

Positive Outlook

- Widespread use in education, agriculture, and medicine.
- AI assistants integrated into daily life.

Concerns

- Stronger need for regulation.
- Balance between innovation and ethics.

Growing AI Job Market

Industry Demand

- Companies need AI engineers, data scientists, ML specialists.
- Governments investing in AI research and innovation.

Career Paths

- AI Engineer
- Data Scientist
- AI Researcher
- Robotics Engineer
- AI Policy and Ethics Specialist

Skills for an AI Career

Technical Skills

- Programming (Python, R, C++).
- Machine Learning & Deep Learning frameworks.
- Data engineering and cloud computing.

Soft Skills

- Problem-solving.
- Communication.
- Ethical awareness.

Summary of Part 8

Key Takeaways

- AI ethics ensures fairness, accountability, and transparency.
- Safety concerns include bias, adversarial attacks, and automation risks.
- The future of AI holds promise but requires careful regulation.
- AI careers are rapidly expanding across industries.