| Machine Learning Techniques (BCS055) | | |
|---|---|---|
| **Course Outcome ( CO)** | | **Bloom's Knowledge Level (KL)** |
| **At the end of course , the student will be able:** | | |
| CO 1 | To understand the need for machine learning for various problem solving | $K_1$ , $K_2$ |
| CO 2 | To understand a wide variety of learning algorithms and how to evaluate models generated from data | $K_1$ , $K_3$ |
| CO 3 | To understand the latest trends in machine learning | $K_2$ , $K_3$ |
| CO 4 | To design appropriate machine learning algorithms and apply the algorithms to a real-world problems | $K_4$ , $K_6$ |
| CO 5 | To optimize the models learned and report on the expected accuracy that can be achieved by applying the models | $K_4$, $K_5$ |

| | **DETAILED SYLLABUS** | **3-0-0** |
|---|---|---|
| **Unit** | **Topic** | **Proposed Lecture** |
| **I** | **INTRODUCTION** – Learning, Types of Learning, Well defined learning problems, Designing a Learning System, History of ML, Introduction of Machine Learning Approaches – (Artificial Neural Network, Clustering, Reinforcement Learning, Decision Tree Learning, Bayesian networks, Support Vector Machine, Genetic Algorithm), Issues in Machine Learning and Data Science Vs Machine Learning; | **08** |
| **II** | **REGRESSION:** Linear Regression and Logistic Regression<br>**BAYESIAN LEARNING** - Bayes theorem, Concept learning, Bayes Optimal Classifier, Naïve Bayes classifier, Bayesian belief networks, EM algorithm.<br>**SUPPORT VECTOR MACHINE:** Introduction, Types of support vector kernel – (Linear kernel, polynomial kernel,and Gaussiankernel), Hyperplane – (Decision surface), Properties of SVM, and Issues in SVM. | **08** |
| **III** | **DECISION TREE LEARNING** - Decision tree learning algorithm, Inductive bias, Inductive inference with decision trees, Entropy and information theory, Information gain, ID-3 Algorithm, Issues in Decision tree learning.<br>**INSTANCE-BASED LEARNING** – k-Nearest Neighbour Learning, Locally Weighted Regression, Radial basis function networks, Case-based learning. | **08** |
| **IV** | **ARTIFICIAL NEURAL NETWORKS** – Perceptron's, Multilayer perceptron, Gradient descent and the Delta rule, Multilayer networks, Derivation of Backpropagation Algorithm, Generalization, Unsupervised Learning – SOM Algorithm and its variant;<br>**DEEP LEARNING** - Introduction,concept of convolutional neural network , Types of layers – (Convolutional Layers , Activation function , pooling , fully connected) , Concept of Convolution (1D and 2D) layers, Training of network, Case study of CNN for eg on Diabetic Retinopathy, Building a smart speaker, Self-deriving car etc. | **08** |
| **V** | **REINFORCEMENT LEARNING**–Introduction to Reinforcement Learning , Learning Task,Example of Reinforcement Learning in Practice, Learning Models for Reinforcement – (Markov Decision process , Q Learning - Q Learning function, Q Learning Algorithm ), Application of Reinforcement Learning,Introduction to Deep Q Learning.<br>**GENETIC ALGORITHMS:** Introduction, Components, GA cycle of reproduction, Crossover, Mutation, Genetic Programming, Models of Evolution and Learning, Applications. | **08** |

**Text books:**

1. Tom M. Mitchell, ―Machine Learning, McGraw-Hill Education (India) Private Limited, 2013.
2. Ethem Alpaydin, ―Introduction to Machine Learning (Adaptive Computation and

# Unit 5

**SUBJECT: MACHINE LEARNING TECHNIQUES**

## Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning where an agent interacts with an environment to learn the best actions to take to achieve a specific goal. The agent learns by receiving feedback in the form of rewards or penalties based on its actions and uses this feedback to improve its decision-making over time.

## Learning Task in Reinforcement Learning

In reinforcement learning, the learning task involves training an agent to achieve a specific goal by interacting with its environment. The process consists of several components:

1. **Agent:** The learner or decision-maker.
2. **Environment**: The external system with which the agent interacts.
3. **State (S):** The current situation of the environment observed by the agent.
4. **Actions (A)**: The set of all possible moves the agent can make.
5. **Reward (R):** Feedback from the environment for the agent's actions. Positive rewards encourage good behavior, while penalties discourage undesired actions.
6. **Policy (π):** A strategy that the agent uses to decide its actions based on its current state.
7. **Objective:** The goal of the agent is to learn an optimal policy to maximize cumulative rewards over time.

The agent learns through a trial-and-error approach:

- Exploration: Trying new actions to discover their effects.
- Exploitation: Using knowledge gained so far to maximize immediate rewards.

## Example of Reinforcement Learning in Practice

**1: Self-Driving Cars**

- Agent: The car's AI system.
- Environment: The road, traffic signals, pedestrians, and other vehicles.
- State: Current speed, lane position, distance from other objects, etc.

- Actions: Steering, braking, accelerating, or signaling.
- Reward: Positive reward for staying in the lane, stopping at a red light, or avoiding accidents; penalty for collisions or breaking traffic rules.
- Learning Goal: The car learns to drive safely and efficiently by repeatedly interacting with the environment and adjusting its actions.

**2: Game Playing (e.g., Chess or Go)**

- Agent: The AI player.
- Environment: The game board and its rules.
- State: The current configuration of the game board.
- Actions: Moving a piece, capturing an opponent's piece, etc.
- Reward: Positive reward for winning the game or achieving strategic goals; penalty for losing or making poor moves.
- Learning Goal: To master the game by developing strategies that maximize the chances of winning.

Reinforcement learning is also widely used in robotics, healthcare (personalized treatment planning), and finance (portfolio optimization). The commonality across these examples is that the agent improves its performance over time through feedback from its actions.

## Learning Models for Reinforcement Learning

Reinforcement Learning uses specific models and techniques to guide the agent in learning optimal behaviors. Two widely used models are:

### Markov Decision Process (MDP)

A Markov Decision Process (MDP) is a mathematical framework designed to model decision-making problems where an agent interacts with an environment to achieve a goal. The MDP framework helps determine the best course of action for an agent to maximize its cumulative rewards over time, even in uncertain or dynamic environments.

#### Key Elements of MDP

1. **States (S):**
   - Represent the current situation or position of the agent within the environment.
   - Examples: The current location of a robot in a warehouse or the configuration of a chessboard in a game.
2. **Actions (A):**
   - The set of all possible moves or choices the agent can make in any given state.
   - Examples: Moving left, right, or staying still in a grid environment.

3. **Transition Probability (P):**
    - Defines the likelihood of moving from one state to another after taking a specific action.
    - Represented as P(s′∣s,a)P(s′∣s,a), where ss is the current state, aa is the action taken, and s′s′ is the next state.
    - Example: In a dice game, rolling a six might lead to a certain state with a probability of 1661.
4. **Rewards (R):**
    - The immediate feedback or outcome received after the agent takes an action in a state.
    - Rewards guide the agent toward desirable outcomes by reinforcing good actions and discouraging bad ones.
    - Examples: Gaining points for completing a task or receiving a penalty for an error.

**Applications of MDP**

- Robotics: Planning optimal paths for robots in dynamic environments.
- Game AI: Developing strategies for games like chess or Go.
- Finance: Portfolio optimization by choosing actions that maximize returns.
- Healthcare: Personalized treatment planning based on patient states and outcomes.

By using MDPs, reinforcement learning can solve complex decision-making problems with structured and mathematical precision.

## Q-Learning

Q-Learning is a widely used and powerful model-free reinforcement learning algorithm. Its primary goal is to train an agent to determine the most beneficial action to take in any given state to maximize its cumulative rewards over time. Unlike other methods, Q-Learning does not require prior knowledge of the environment's dynamics, making it "model-free."

**Key Concepts in Q-Learning**

Q-Value (Quality Value):
The Q-value represents the "goodness" or expected future reward of taking a particular action aa in a specific state ss.

A higher Q-value indicates a more beneficial action.

The Q-value is updated iteratively using the Q-Learning Function.

Policy:
The agent's strategy for choosing actions based on the Q-values.

An optimal policy ensures that the agent always selects the action with the highest Q-value.

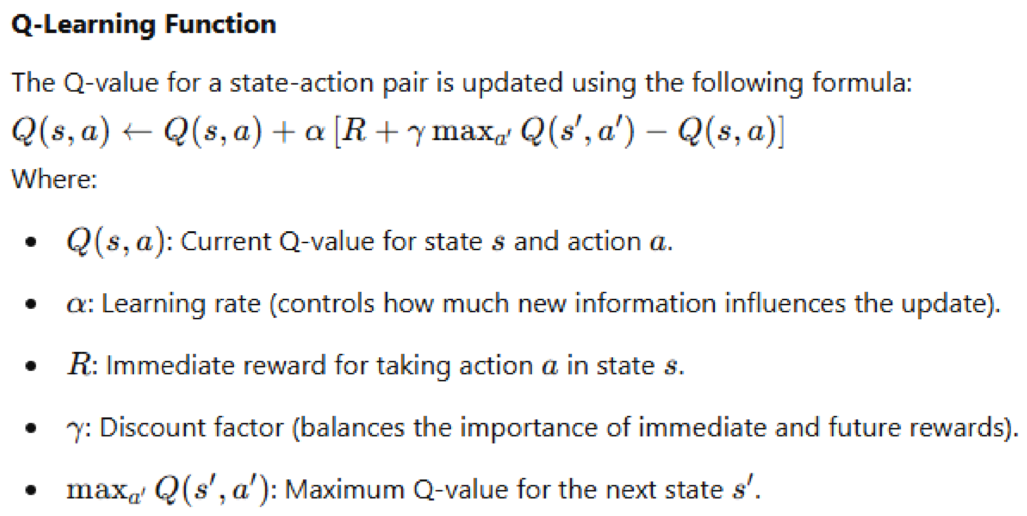
**Exploration vs. Exploitation:**

Exploration: Trying new actions to gather information about the environment.

Exploitation: Using the knowledge gained so far to maximize immediate rewards.

## Q-Learning Function

The Q-value for a state-action pair is updated using the following formula:



**Q-Learning Function**

The Q-value for a state-action pair is updated using the following formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Where:

- $Q(s,a)$: Current Q-value for state $s$ and action $a$.
- $\alpha$: Learning rate (controls how much new information influences the update).
- $R$: Immediate reward for taking action $a$ in state $s$.
- $\gamma$: Discount factor (balances the importance of immediate and future rewards).
- $\max_{a'} Q(s',a')$: Maximum Q-value for the next state $s'$.

## <u>Q-Learning Algorithm</u>

The Q-Learning algorithm operates through the following steps:

**1. Initialization:**

- All Q-values are initially set to zero or some random value.

**2. State Selection:**

- The agent begins from an initial state.

**3. Action Selection:**

- The agent selects an action based on current Q-values or an exploration strategy.
    - **Exploration:** Trying out new actions.
    - **Exploitation:** Using the current best-known action.

**4. Receiving Reward:**

- Based on the selected action, the agent receives a reward.

**5. Q-Value Update:**

- Update the Q-value using the Q-Learning function.

**6. Iteration:**

- Repeat this process until the agent learns an optimal policy for all states.

**Advantages of Q-Learning**

Model-Free: Does not require knowledge of the environment's dynamics.

Flexible: Can be applied to a wide range of problems.

Effective: Performs well in stochastic (uncertain) environments.

**Example of Q-Learning in Action**

**Scenario**:

Environment: A maze where the agent must reach the goal while avoiding obstacles.

States (S): Different positions in the maze.

Actions (A): Moving up, down, left, or right.

Rewards (R):

+10 for reaching the goal.

-1 for each step taken.

-10 for hitting an obstacle.

**Learning Process:**
The agent explores the maze, updates the Q-values based on rewards and penalties, and eventually learns the shortest and safest path to the goal.

## Applications of Reinforcement Learning (RL)

Reinforcement Learning (RL) is widely used across various industries and domains due to its ability to learn optimal decision-making policies in complex, dynamic environments. Below are some of the major applications of RL:

### 1. Robotics

- Task Automation: RL is used to train robots to perform tasks like object manipulation, assembling parts, or warehouse management.
- Navigation: Autonomous robots use RL to navigate complex environments while avoiding obstacles.
- Human-Robot Interaction: RL enables robots to adapt to human behavior and work collaboratively in shared environments.

### 2. Gaming and AI Development

- Game Playing: RL powers AI agents in video games, enabling them to learn optimal strategies (e.g., AlphaGo mastering the game of Go).
- Game Testing: Automates the testing of game mechanics and ensures challenging gameplay.
- Dynamic Difficulty Adjustment: Adapts game difficulty based on the player’s skill level.

---

### 3. Autonomous Vehicles

- Path Planning: RL helps vehicles learn to navigate roads and traffic efficiently.
- Collision Avoidance: Ensures safety by training vehicles to avoid obstacles dynamically.
- Driving Policies: Develops efficient driving strategies under varying conditions like weather or traffic.

---

### 4. Finance and Trading

- Portfolio Management: RL optimizes investment strategies to maximize returns.
- Trading Algorithms: Learns to buy or sell assets at the best times based on market conditions.
- Fraud Detection: Identifies unusual patterns in transactions to flag potential fraud.

---

**5. Healthcare**

- Personalized Treatment Plans: RL helps design optimal treatment strategies for patients, such as adjusting medication doses.
- Medical Imaging: Enhances image analysis by learning patterns in X-rays, MRIs, or CT scans.
- Drug Discovery: Optimizes the process of identifying effective drug combinations.

## Deep Q-Learning:

Deep Q-Learning is a method of reinforcement learning that combines the Q-Learning algorithm with deep neural networks. It trains an agent to make the right decisions in large and complex environments. Neural networks are used to estimate Q-Values and maximize rewards.

It is particularly useful in fields like gaming, robotics, and autonomous vehicles.

**Applications of Deep Q-Learning:**

1. **Game Playing:**
   - Deep Q-Learning is widely used in training AI to play video games. For example, DeepMind's AlphaGo and AI agents in Atari games use it to make optimal moves and improve their strategies.
2. **Robotics:**
   - It helps robots learn complex tasks like object manipulation, navigation, and motion planning without explicit programming.
3. **Autonomous Vehicles:**
   - Deep Q-Learning is applied in self-driving cars to make decisions such as lane-changing, obstacle avoidance, and path planning.
4. **Healthcare:**
   - It can assist in medical diagnosis, treatment planning, and optimizing drug doses by analyzing complex datasets.
5. **Finance:**
   - Deep Q-Learning is used in stock trading to predict market trends and optimize investment strategies.

## Advantages of Deep Q-Learning

1. **Scalability:**
    - Handles large and continuous state spaces efficiently.
2. **Efficient Learning:**
    - Experience replay improves sample efficiency and stability.
3. **Versatility:**
    - Can be applied to problems with high-dimensional input spaces, such as images or videos (e.g., playing Atari games).

**Example: Deep Q-Learning in Action**

**Scenario:**

- Environment: Playing an Atari game like Breakout.
- State Space: Pixel values of the game screen.
- Actions: Move paddle left, right, or stay.
- Rewards: +1 for hitting the ball, -1 for missing it.

**Learning Process:**

- The deep neural network learns to predict the optimal paddle movement based on the game screen, maximizing the score over time.

## Genetic Algorithm (GA)

A **Genetic Algorithm (GA)** is a search and optimization technique inspired by the principles of natural selection and genetics. It is part of evolutionary algorithms and is used to solve optimization and search problems by mimicking biological processes like reproduction, mutation, and survival of the fittest.

## Core Concepts in Genetic Algorithms

1. **Population**:
   A set of potential solutions to the optimization problem, where each solution is represented as an individual (often encoded as a string or array, e.g., binary or real-valued).

2. **Chromosome**:
   Representation of a candidate solution. For example, in binary encoding, a chromosome might look like 101011.
3. **Gene**:
   A part of a chromosome representing a specific trait or variable in the solution. For example, in the chromosome 101011, each digit is a gene.
4. **Fitness Function**:
   A function that evaluates how good a solution is by assigning it a "fitness" score. Higher fitness scores indicate better solutions.
5. **Selection**:
   A process to choose individuals from the population for reproduction based on their fitness. Common selection methods include:
   - **Roulette Wheel Selection**: Probability of selection proportional to fitness.
   - **Tournament Selection**: Selects the best individual from a randomly chosen subset.
6. **Crossover (Recombination)**:
   A process where two parent solutions combine to create offspring. This introduces diversity into the population. Common techniques include:
   - **Single-Point Crossover**: Split chromosomes at one point and exchange parts.
   - **Two-Point Crossover**: Split chromosomes at two points.
   - **Uniform Crossover**: Randomly mix genes from both parents.
7. **Mutation**:
   Randomly changes some genes in an individual to maintain diversity and avoid local optima. For example, flipping a bit in binary encoding (e.g., 101011 becomes 101111).
8. **Termination Criteria**:
   The algorithm stops when:
   - A maximum number of generations is reached.
   - An acceptable fitness level is achieved.
   - The population converges to a solution.

## Genetic Algorithm Process

1. **Initialization**:
   Generate an initial population of solutions randomly.
2. **Evaluation**:
   Compute the fitness of each individual in the population using the fitness function.
3. **Selection**:
   Select individuals for reproduction based on their fitness.
4. **Crossover and Mutation**:
   - Perform crossover to create new offspring.
   - Apply mutation to introduce variability.
5. **Replacement**:
   Replace the old population with the new one, keeping the best solutions.

6. **Repeat**:
   Iterate the process until the termination criteria are met.

## Advantages of Genetic Algorithms

1. **Robustness**:
   Can handle complex and multi-dimensional problems.
2. **Exploration and Exploitation**:
   Efficiently searches large spaces and avoids getting stuck in local optima.
3. **Adaptability**:
   Can be applied to various optimization problems across domains.
4. **Parallelism**:
   Evaluates multiple solutions simultaneously, making it suitable for parallel computing.

## Applications of Genetic Algorithms

1. **Optimization**:
   - Traveling Salesman Problem (TSP): Finding the shortest route between cities.
   - Resource Allocation: Optimizing the use of resources in industries.
2. **Machine Learning**:
   - Feature Selection: Choosing the most relevant features for training models.
   - Hyperparameter Tuning: Optimizing hyperparameters of algorithms.
3. **Engineering Design**:
   - Circuit Design: Optimizing electronic circuit layouts.
   - Structural Design: Designing efficient mechanical structures.
4. **Bioinformatics**:
   - Protein Folding: Understanding protein structures.
   - Genetic Research: Analyzing DNA sequences.
5. **Robotics**:
   - Path Planning: Finding optimal paths for robots in dynamic environments.
6. **Finance**:
   - Portfolio Optimization: Allocating assets for maximum return.
   - Trading Strategies: Developing efficient stock trading strategies.
7. **Game Development**:
   - AI in Games: Evolving game characters or strategies.

## Example: Solving the Traveling Salesman Problem (TSP) with GA

**Problem:**
Find the shortest path to visit a set of cities and return to the starting point.

**Steps in GA**:

1. **Representation**:
   Encode the order of cities as chromosomes. For example, A-B-C-D-E.
2. **Fitness Function**:
   Calculate the total distance of the path. Lower distances indicate better fitness.
3. **Crossover**:
   Combine two parent paths to create new paths (offspring).
4. **Mutation**:
   Swap cities randomly to explore new paths.
5. **Result**:
   Over generations, the algorithm evolves to find the optimal or near-optimal path.

## Crossover:

**Crossover** is a genetic operator used to combine two parent solutions (programs) to create a new offspring solution. It involves exchanging parts of the parent programs (like code segments) to form a new program that may have better or different characteristics. This process mimics natural reproduction, where offspring inherit traits from both parents.

**Example:**
If Parent 1 has the program `x + y` and Parent 2 has `x * y`, a crossover might result in an offspring program like `x + y * y`.

## Mutation:

**Mutation** is a genetic operator that introduces random changes to a solution. This can involve altering parts of the program (such as changing an operator or a constant) to explore new areas of the solution space. Mutation helps maintain diversity in the population and can prevent the algorithm from getting stuck in local optima.

**Example:**
If a program is `x + y`, a mutation could change it to `x * y` or `x - y`, introducing a new variation.

## What is Genetic Programming?

Genetic Programming (GP) is an evolutionary algorithm-based approach used to automatically generate computer programs or models to solve specific problems. It is inspired by the biological process of natural evolution and adapts the principles of Genetic Algorithms (GAs) to evolve complete programs instead of fixed-length solutions like strings or numbers.

In GP, solutions to problems are represented as tree-like structures, where nodes correspond to operations (like addition or multiplication) and leaves represent input variables or constants. These structures are evolved over successive generations using evolutionary operators like selection, crossover, and mutation, to find the best-performing program.

---

## Key Features of Genetic Programming:

1. **Automated Problem Solving**:
   GP does not require explicit programming. Instead, it evolves programs to optimize a given fitness function.
2. **Representation of Solutions**:
   Solutions are represented as hierarchical tree structures, similar to mathematical expressions or logical programs.
3. **Evolutionary Operators**:
   GP employs biological evolution-inspired operators:
   - **Selection**: Chooses the best programs based on their fitness.
   - **Crossover**: Combines parts of two parent programs to create new offspring programs.
   - **Mutation**: Introduces variations by modifying parts of a program randomly.

---

## Steps in Genetic Programming:

1. **Initialization**: Generate a population of random candidate programs using predefined function and terminal sets.
2. **Evaluation**: Assess the fitness of each program based on how well it solves the problem.
3. **Selection**: Select the best-performing programs for reproduction.
4. **Crossover**: Combine parts of two parent programs to create new programs.
5. **Mutation**: Modify parts of programs to introduce diversity in the population.
6. **Termination**: Repeat the process until an optimal program is found or a specified number of generations is reached.

## Applications of Genetic Programming:

1. **Symbolic Regression**: Deriving mathematical models from data.
2. **Automated Software Design**: Designing algorithms, circuits, or neural networks.
3. **Control Systems**: Developing control logic for robotics or automated systems.
4. **Data Mining and Pattern Recognition**: Identifying patterns in complex datasets.

---

## Example of Genetic Programming:

**Problem**: Find an equation to fit data points for `y = x^2 + 3x + 2`.

1. **Initial Population**:
   Random programs like `y = x + 2` or `y = x * x` are generated.
2. **Fitness Evaluation**:
   Compare the output of each program with the expected output for given input values.
3. **Evolution**:
   - Combine programs like `y = x + 2` and `y = x * x` to produce `y = x * x + 2`.
   - Introduce mutations, such as adding a term to create `y = x * x + 3x + 2`.
4. **Optimal Solution**:
   After several generations, GP evolves a program close to `y = x^2 + 3x + 2`.

## Conclusion:

Genetic Programming is a powerful tool for solving complex problems where traditional programming methods are infeasible or time-consuming. By mimicking evolution, it allows programs to improve automatically and adapt to the requirements of the task.

## Models of Evolution and Learning

Models of Evolution and Learning and their applications explain how concepts of natural evolution and learning are applied in computer science and artificial intelligence to solve complex problems. These models use principles like selection, crossover, and mutation to create systems that learn and evolve on their own.

1. **Evolutionary Models**:
   These models simulate biological evolution and are used for optimization and problem-solving:
   - **Genetic Algorithms (GA)**:
     A population of solutions evolves, and the best solutions are selected in each generation.

            - *Example*: Finding the shortest route in the Traveling Salesman Problem.
    - **Genetic Programming (GP)**:
      Evolves algorithms or mathematical expressions that solve specific problems.
        - *Example*: Designing automated algorithms.
    - **Neuroevolution**:
      Evolves the architecture and weights of neural networks.
        - *Example*: Optimizing deep learning models.
    - **Differential Evolution (DE)**:
      Optimizes problems in continuous spaces.
        - *Example*: Tuning hyperparameters in machine learning models.

## Applications of Evolutionary Models

1. **Optimization Problems**:
   Finding solutions for resource allocation and scheduling problems.
    - *Example*: Supply chain optimization.
2. **Engineering Design**:
   Optimizing complex mechanical and structural designs.
    - *Example*: Designing aerodynamic vehicles.
3. **Machine Learning**:
   Selecting features and tuning hyperparameters for models.
    - *Example*: Evolving the architecture of neural networks.
4. **Robotics**:
   Creating motion planning and control strategies for autonomous robots.
    - *Example*: Path optimization for self-driving cars.
5. **Healthcare**:
   Optimizing treatment plans and discovering drugs.
    - *Example*: Creating personalized cancer treatment plans.
6. **Finance**:
   Optimizing portfolios and predicting stock market trends.
    - *Example*: Improving investment strategies.