

College of Computer Science

Artificial Intelligence

Lecture 1: Review of Artificial Intelligence

Artificial Intelligence (AI) is a branch of computer science aimed at developing systems and programs capable of simulating human cognitive abilities, such as thinking, learning, problem-solving, and decision-making. AI seeks to create machines and programs that can perform complex tasks intelligently and, in a manner, similar to human.

Definition of Artificial Intelligence

- 1- The exciting new effort to make computers think (machines minds, in the full literal sense. (Haugeland,1985)
- 2- The study of how to make computers do things at which, at the moment, people are better. (Rich& Knight, 1991)
- 3- An area of computer science that deals with giving machines the ability to seem like have human intelligence the power of machine to copy intelligence behavior. (Merriam-webster.com)

History of Artificial Intelligence

- **(1940-1950):** AI research began in earnest in the mid-20th century. Scientists at that time sought to understand the human mind and translate it into a model that could be programmed.
- **(1956):** The Dartmouth Conference in 1956 is considered the official start of AI as a field, where the term "Artificial Intelligence" was first used. This period saw research in logical computing and problem-solving methods.
- **(1970-1990):** AI technologies evolved significantly, but they faced performance issues, especially due to technical limitations.

-
- **(2000-Present):** Thanks to advances in large-scale computing, the availability of big data, and developments in machine learning, AI has flourished and become an integral part of modern technology.

Types of Artificial Intelligence

- **Narrow AI (Weak AI):** Focuses on executing specific tasks such as voice or image recognition. This type is the most prevalent in modern applications.
- **General AI (Strong AI):** Refers to systems capable of performing any intellectual task that a human can. This type is still in the research and development phase.
- **Superintelligence:** Refers to AI that is expected to surpass humans in all cognitive domains. This remains a distant and unachieved future concept.

Applications of Artificial Intelligence

1. **Machine Learning:** Involves programming systems to learn from data without explicit programming.
2. **Natural Language Processing (NLP):** Allows machines to understand and analyze human language, used in applications like speech recognition or machine translation.
3. **Computer Vision:** Enables machines to analyze images and videos, with applications such as facial recognition and medical imaging.
4. **Robotics:** Designing robots capable of performing various tasks, from manufacturing to healthcare.
5. **Healthcare:** AI is used in medical diagnostics, drug development, image analysis, and robotic surgery.

-
6. **Finance:** AI technologies help in fraud detection, market forecasting, and automated trading.
 7. **Education:** AI platforms are used to personalize educational content and improve online learning experiences.
 8. **E-commerce:** AI powers recommendation systems to offer personalized suggestions to users.
 9. **Industry:** AI enhances production processes through automation and predictive maintenance applications.

Intelligent Application VS Automated Application

Automated production lines



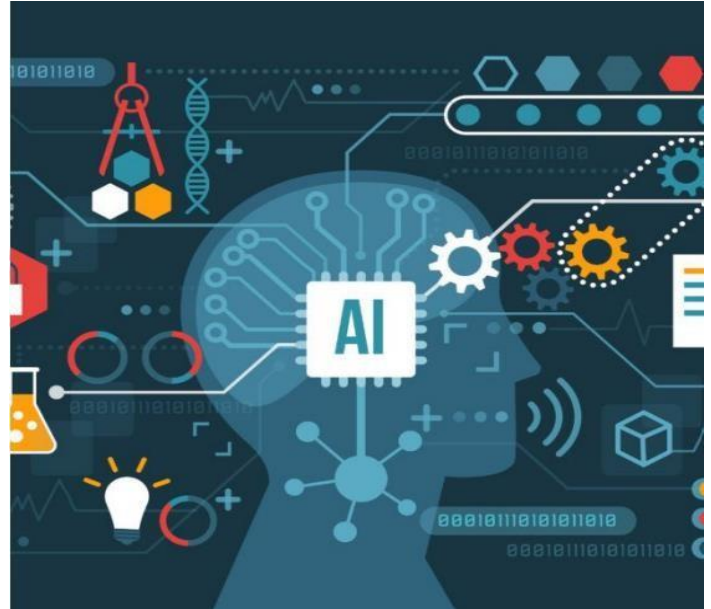
Autonomous driving



1-Automated is application that follows pre-programmed “rules”	1-Artificial intelligent is designed to simulate human thinking.
2-Automation has a single purpose: To let machines, preform repetitive, Monotonous tasks.	2-Artificial intelligent purpose: Create technologies that able mimic What a human can say, think and do.

Characteristics of languages in AI

- High-Level Complexity
- Symbolic Processing and Knowledge Representation
- Flexibility and Dynamic Features
- Libraries and Frameworks
- Interoperability



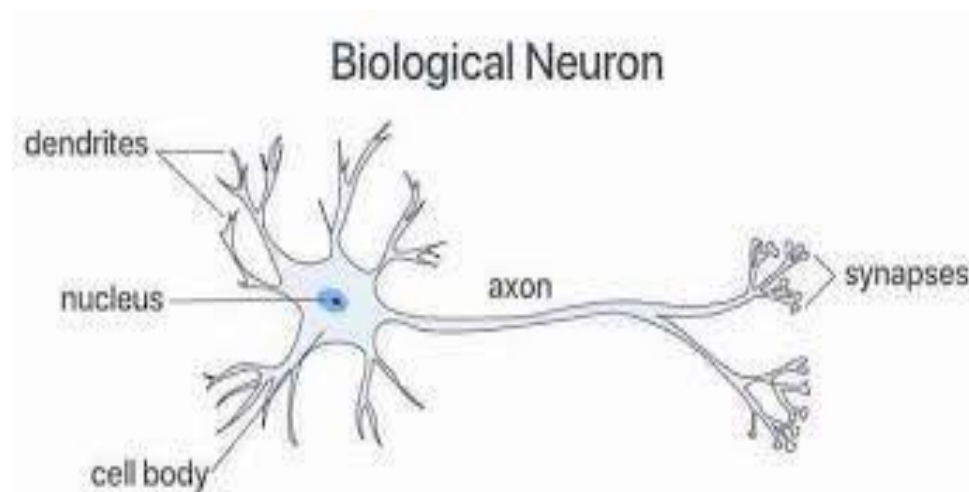
An Overview of Artificial Intelligence: Problems and Subproblems

1. finding, logical thinking, and the ability to solve problems.
2. Knowledge representation.
3. Default thinking and the problem of qualification.
4. Widening logical knowledge.
5. planning.
6. learning.
7. Mechanism of natural language
8. Movement and the possibility of change.
9. perception.

Lecture 2: Introduction to Neural Networks

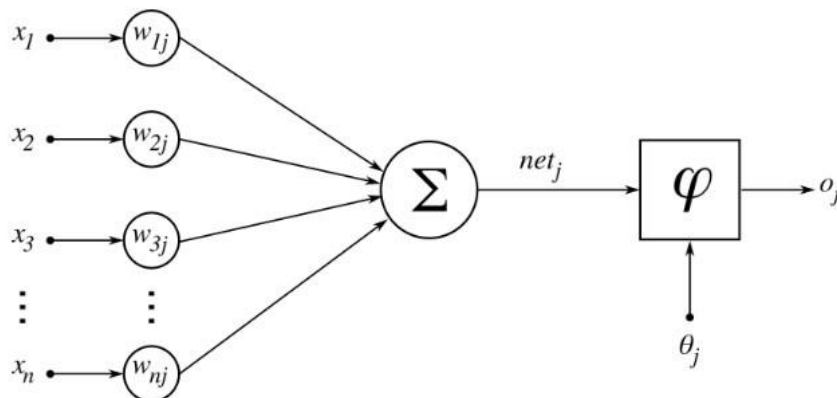
Introduction

Artificial Neural Networks (ANNs): are computational models inspired by how the human brain works. Neural networks consist of a dense, interconnected collection of nerve cells called **neurons**, and the human brain consists of about 10 billion neurons and 60 trillion **synapses** between them. Each neuron consists of a **cell body**, a number of fibers called **dendrites**, and a single filament called an **axon**, as shown in the following figure.



Through the work of biological neural networks, the idea of artificial neural networks was inspired, and simulation models of the way the human brain works were designed using computers to solve problems. Similarly, neural networks contain simple processing units (corresponding to neurons) and have input connections (corresponding to dendrites) and output connections (corresponding to axons). These units are connected to each other

through interconnection points (corresponding to synapses), and each connection point is assigned a weight, which is a numerical value. A neuron in an ANN has one or more inputs but only one output. The neuron performs a weighted sum of its inputs and applies an activation function to the result. The input to a neuron is typically a vector of values, which represent features or attributes of the data being processed by the ANN. Each input value is associated with a specific weight, reflecting the importance of that input in determining the neuron's output. The neuron multiplies each input value by its corresponding weight, and then sums the results of this multiplication. This value is known as the activation level of the neuron. As shown below:



Components of artificial neural network

1- Layers:

- Input Layer:** This layer contains neurons that receive the input data. Each neuron in this layer represents a feature or attribute of the data.
- Hidden Layers:** These are layers between the input and output layers where data processing occurs. There can be several hidden layers, and the number of

neurons in each layer may vary. The hidden layers are responsible for learning complex patterns in the data.

- c) **Output Layer:** This layer produces the final outputs of the network. The number of neurons in this layer depends on the task; for example, in classification, the number of neurons may equal the number of possible categories.

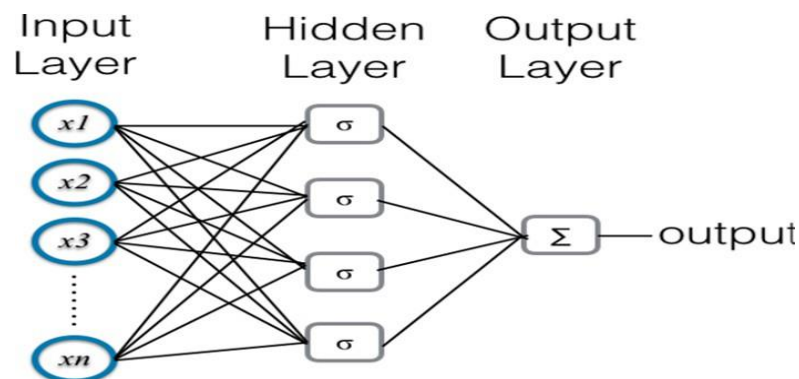
2- Neurons:

Each neuron receives signals from the neurons in the previous layer (or from inputs in the case of the first layer) and calculates a weighted sum of the inputs (each input multiplied by a specific weight) plus a bias term, then passes this value through an activation function.

3- Weights:

A weight is assigned to each connection between neurons. Weights determine the strength of the signal transmitted from one neuron to another. Weights are adjusted during training to reduce errors and improve the network's performance.

Artificial Neural Network Architecture



The structure or architecture of the neural network is the way in which the processing units are connected to each other within each layer or between the different layers that make up the network, as these units are connected in different ways, and according to the method of this connection and the number of layers that make up the network, the general structure or architecture of the neural network appears to us.

Network structures can be classified according to the number of layers into:

1- Single-layer networks: It is one of the simplest types of network structures and usually consists of a single layer of processing elements that directly connect the network inputs with its outputs, where all calculations are performed in the output layer. Usually, the number of layers in the network is calculated without calculating the input layer because it does not perform any calculations.

2- Multilayer networks: The structures of this network contain more than one layer of processing elements that are connected by interconnections (weights), as the network consists of at least two layers, a layer to receive inputs that are not calculated or an output layer, and between the input and output layers there is a hidden layer, and the network can contain more than one hidden layer, depending on the type of application used in the network. Multilayer networks are among the most widely used networks, as they are characterized by great effectiveness in completing various applications.

Applications of Artificial Neural Networks

- **Social Media:** Artificial Neural Networks are widely used in social media. For example, let's take Facebook's "People You May Know" feature which suggests people you may know in real life so that you can send them friend requests. Well, this magical effect is achieved using artificial neural networks which analyses your profile, interests, your current friends and their friends and various other factors to calculate the people you may know. Another popular application of machine learning in social media is facial recognition. This is done by finding around 100 reference points on a person's face and then matching them with the ones already available in the database using convolutional neural networks.
- **Marketing and Sales:** When you log in to e-commerce sites like Amazon and Flipkart, they will recommend products for you to buy based on your past browsing history. Similarly, suppose you love pasta, then Zomato, swigged etc. will show you restaurant recommendations based on your tastes and past order history. This is true in all modern marketing sectors like book sites, movie services, hospitality sites, etc., and is done through the implementation of personalized marketing. This uses artificial neural networks to identify a customer's likes, dislikes, past shopping history, etc., and then design marketing campaigns accordingly.
- **Healthcare:** Artificial neural networks are used in oncology to detect cancerous tissues with accuracy similar to that of doctors. They can also help identify rare diseases in their early stages through facial image analysis. The application of these networks in healthcare enhances diagnostic capabilities and improves the quality of medical care globally.

Lecture 3: Understanding Learning Methods and Neural Network Architectures

Learning Methods and Neural Network Architectures

Neural networks use a variety of algorithms to learn patterns and gain knowledge from data. These algorithms vary based on the type of learning required and the specific goal. Below is an explanation of the main types of learning used in training neural networks:

1-Supervised Learning: In supervised learning, the data comes with labels or correct outputs. The neural network is trained to predict these outputs based on the input data, meaning it learns the relationships between inputs and outputs.

- **How it works:** The model is given a dataset that includes input-output pairs (for example: images and their corresponding labels). Using this data, the model calculates the error of its predictions via a loss function. The model then adjusts its weights using techniques like backpropagation through an algorithm such as gradient descent to minimize this error.
- **Applications:** Image classification, speech recognition, and machine translation.

2- Unsupervised Learning: In unsupervised learning, the data has no labels or output values. The goal is to discover patterns or underlying structures in the data. The model learns to organize or interpret the data without explicit guidance.

- **How it works:** The model attempts to find patterns such as clustering or dimensionality reduction. There are no correct answers provided, so the model learns through self-analysis of the data.
- **Applications:** Data compression, customer segmentation, anomaly detection, and dimensionality reduction (e.g., using Principal Component Analysis or PCA).

3- Reinforcement Learning: Reinforcement learning differs from the previous two types in that the system learns by interacting with an environment. Here, no direct labels are provided, but the model learns through rewards and penalties based on its actions within the environment.

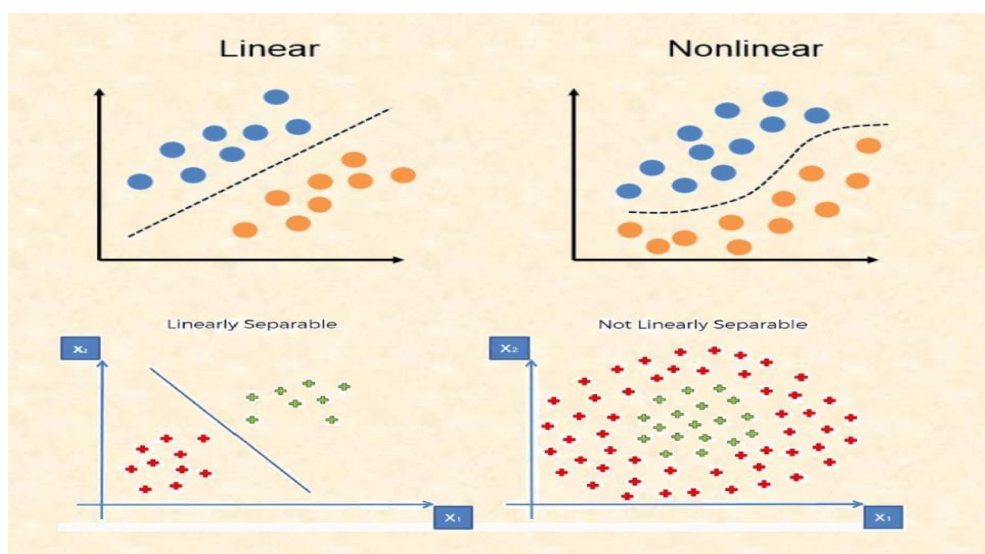
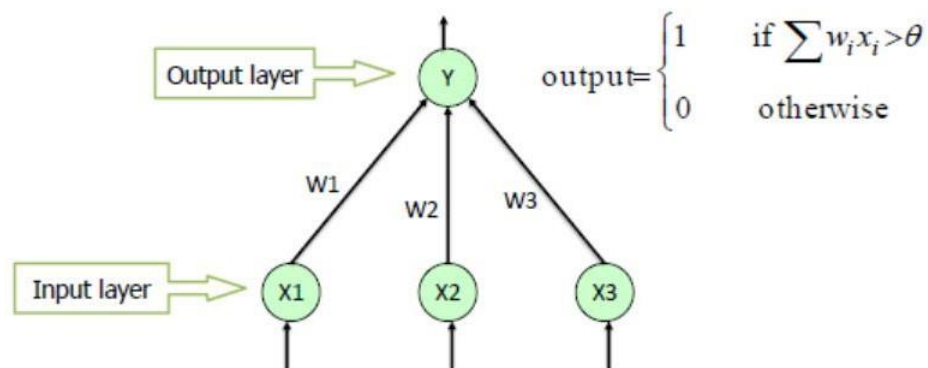
- **How it works:** The agent interacts with the environment and takes actions. Based on the results of these actions, the agent receives rewards or penalties that help it improve its strategies. The goal is to learn a strategy that maximizes long-term rewards.
- **Applications:** Game-playing (like AlphaGo), robotics, intelligent control systems.

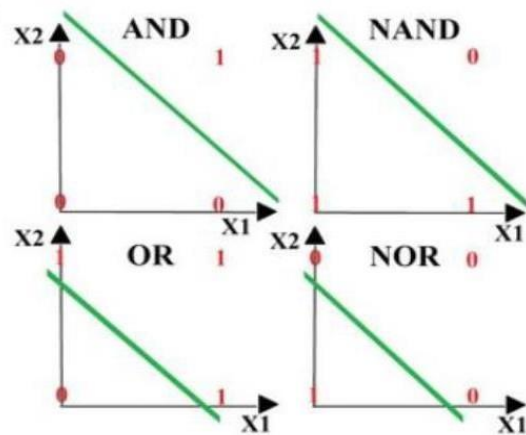
Lecture 4: Single Layer Perceptron (SLP)

What is a Single Layer Perceptron (SLP)?

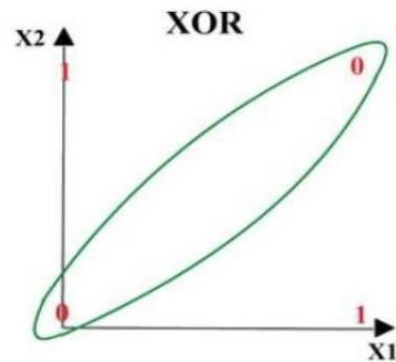
A single layer perceptron (SLP) is a feed-forward network based on a threshold transfer function. SLP is the simplest type of artificial neural networks and can only classify linearly separable cases with a binary target (1, 0).

Single Layer Perceptron





(a) Linearly separable problems:
AND, NAND, OR, NOR
operators



(b) Non-linearly separable
problems: XOR operator

Algorithm

The single layer perceptron does not have a priori knowledge, so the initial weights are assigned randomly. SLP sums all the weighted inputs and if the sum is above the threshold (some predetermined value), SLP is said to be activated (output=1).

$$\begin{aligned}
 w_1 x_1 + w_2 x_2 + \dots + w_n x_n > \theta & \Rightarrow \text{Output } 1 \\
 w_1 x_1 + w_2 x_2 + \dots + w_n x_n \leq \theta & \Rightarrow \text{Output } 0
 \end{aligned}$$

Learning Algorithm: Training Perceptron

The training of perceptron is a supervised learning algorithm where weights are adjusted to minimize error whenever the output does not match the desired output.

- if the output is correct then no adjustment of weights is done.

i.e. $W_{ij}^{k+1} = W_{ij}^k$

-if the output is **1** but should have been **0** the weights are decreased on the active input link

i.e. $W_{ij}^{k+1} = W_{ij}^k - \alpha X_i$

- if the output is 0 but should have been 1 then the weights are increased on the active input link

i.e. $W_{ij}^{k+1} = W_{ij}^k + \alpha X_i$

where

W_{ij}^{k+1} is the new adjusted weight, W_{ij}^k

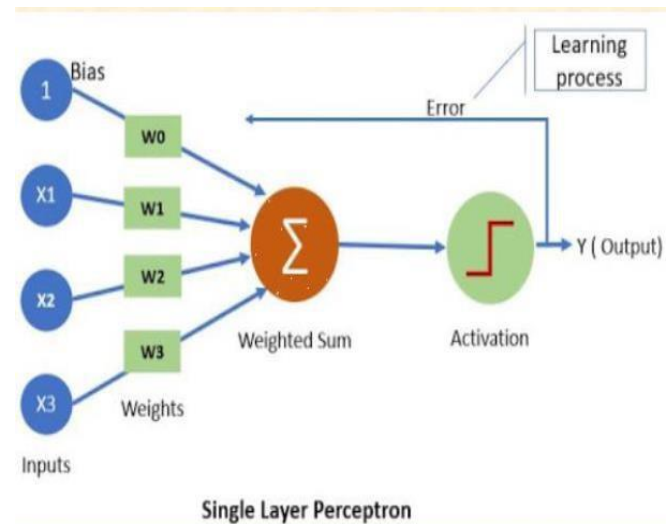
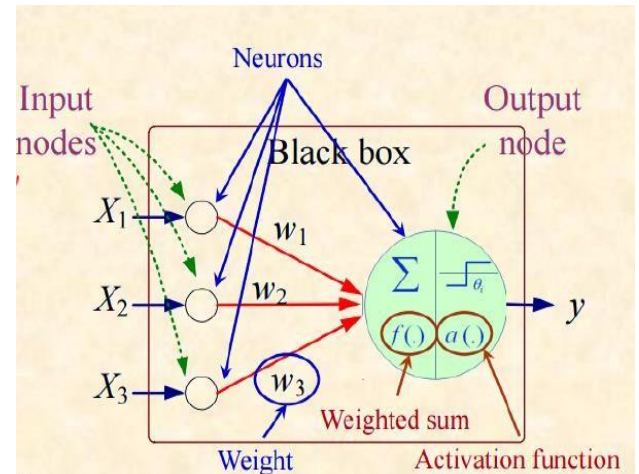
X_i is the input and α is the learning rate parameter.

α small leads to slow and α leads to fast learning.

The algorithm is illustrated step-by-step

- **Step 1:**
Create a perceptron with $(n+1)$ input neurons $X_0, X_1, \dots, X_n$
Where $X_0 = 1$ is the bias input.
Let 0 be the output neuron.
- **Step 2:**
Initialize weight $W = (W_0, W_1, \dots, W_n)$ to random weights.
- **Step 3:**
Iterate through the input patterns X_j of the Training set using the weight set.
i.e., compute the weighted sum of inputs
 $\text{net}_j = \sum_{i=1}^n X_i * W_i$ for each input pattern j
- **Step 4:**
Compute the output y_j

$$y_j = f(\text{net}_j) = \begin{cases} 1 & \text{if } \text{net}_j > \theta \\ 0 & \text{if } \text{net}_j \leq \theta \end{cases} \quad \text{where } \text{net}_j = \sum_{i=1}^n x_i w_{ij}$$



- **Step 5:**
Compare the computed output y_j with the target output y_j for each input pattern j .
 $\text{Error signal} = \Delta_i = E = (O_{\text{desired}} - O_{\text{actual}})$

$$E = (d - y)$$

If all the input patterns have been classified correctly, then output(read) the weights and exit.

- Step 6:

Otherwise, update the weights as given below:

If the computed outputs y_j is 1 but should have been 0.

Then $W_i = W_i - \alpha X_i$, $i=0,1, 2, \dots, n$.

If the computed outputs y_j is 0 but should have been 1 then $W_i = W_i + \alpha X_i$

$i = 0, 1, 2, \dots, n$. where α is the learning parameter and is constant.

- Step 7:

Go to step 3

- End

Lecture 5: Back-Propagation Network (BPN)



What is backpropagation?

Backpropagation is a supervised learning algorithm used to train neural networks. It is a key method for optimizing the weights of a neural network by minimizing the error between the predicted output and the actual output.

Advantages of Using the Backpropagation Algorithm in Neural Networks

There are a few other reasons why backpropagation is a useful approach:

- No previous knowledge of a neural network is needed, making it easy to implement.
- It's straightforward to program since there are no other parameters besides the inputs.
- It doesn't need to learn the features of a function, speeding up the process.
- The model is flexible because of its simplicity and applicable to many scenarios.

Limitations of Using the Backpropagation Algorithm in Neural Networks

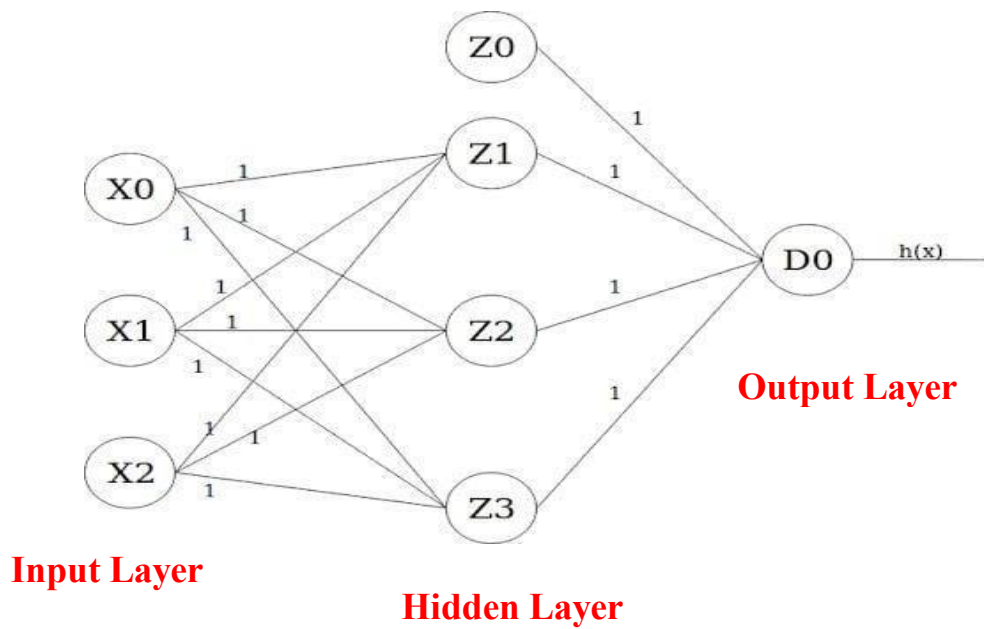
backpropagation is not a blanket solution for any situation involving neural networks. Some of the potential limitations of this model include:

- Training data can impact the performance of the model, so high-quality data is essential.
- Noisy data can also affect backpropagation, potentially tainting its results.
- It can take a while to train backpropagation models and get them up to speed.
- Backpropagation requires a matrix-based approach, which can lead to other issues.

Key Steps of the Backpropagation Algorithm

1. Forward Propagation: Input data passes through the network from the input layer to the output layer, calculating the output based on the current weights.
2. Error Calculation: The error is computed by comparing the predicted output with the actual (target) output using a loss function such as Mean Squared Error (MSE).
3. Backward Propagation: The error is propagated back through the network, layer by layer, to compute the gradients of the error with respect to the weights.
4. Weight Update: The calculated gradients are used to adjust the weights using a method like gradient descent to minimize the error in the next iteration.

Building a Neural Network



The leftmost layer is the input layer, which takes $X0$ as a bias term with value 1, and $X1$ and $X2$ as input features. The layer in the middle is the first hidden layer, which also takes a bias term $Z0$ with value 1. Finally, the output layer has only one output unit $D0$ whose activation value is the actual output of the model (i.e., $h(x)$).

Unit Z1:

$$\begin{aligned}h(x) &= w_0.x_0 + w_1.x_1 + w_2.x_2 \\&= 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 \\&= 1 = a\end{aligned}$$

$$z = f(a) = a \Rightarrow z = f(1) = 1$$

Unit Z2:

$$\begin{aligned}h(x) &= w_0.x_0 + w_1.x_1 + w_2.x_2 \\&= 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 \\&= 1 = a\end{aligned}$$

$$z = f(a) = a \Rightarrow z = f(1) = 1$$

Unit D0:

$$\begin{aligned}h(x) &= w_0.z_0 + w_1.z_1 + w_2.z_2 + w_3.z_3 \\&= 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 \\&= 4 = a\end{aligned}$$

$$z = f(a) = a \Rightarrow z = f(4) = 4$$

Unit Z3:

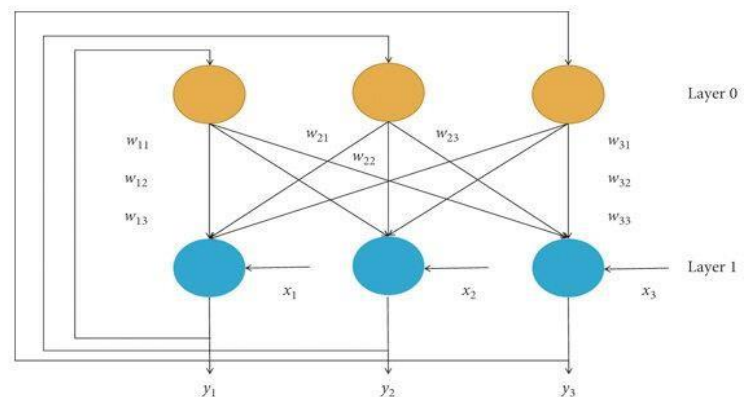
$$\begin{aligned}h(x) &= w_0.x_0 + w_1.x_1 + w_2.x_2 \\&= 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 \\&= 1 = a\end{aligned}$$

$$z = f(a) = a \Rightarrow z = f(1) = 1$$

Lecture 6: The Hopfield Network

Introduction

The Hopfield Neural Networks, invented by Dr John J. Hopfield consists of one layer of 'n' fully connected recurrent neurons. It is generally used in performing auto-association and optimization tasks. It is calculated using a converging interactive process and it generates a different response than our normal neural nets.



Working of Hopfield Network:

1- Training:

- The network "learns" a set of patterns, often binary vectors, by adjusting the weights between neurons.
- The learning rule is typically based on **Hebbian learning** (i.e., neurons that fire together wire together).
- If the network is to store a set of p patterns ($x^{(1)}, x^{(2)}, \dots, x^{(p)}$), the weights between neurons are updated as:

$$w_{ij} = \frac{1}{n} \sum_{k=1}^p x_i^{(k)} x_j^{(k)}$$

where n is the number of neurons and $x_i^{(k)}$ is the value of the i -th neuron in pattern k .

2- Recalling Patterns:

- Once the network is trained, it can retrieve stored patterns from incomplete or noisy inputs.
- Given an input, the neurons are updated asynchronously, and the network adjusts itself iteratively to minimize the energy function.
- The energy function for the Hopfield Network is defined as:

$$E = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j$$

3- Convergence:

- The network will evolve to a stable state, which corresponds to a local minimum of the energy function.
- This stable state should ideally match one of the stored patterns.

Example of Hopfield Network:

Let's say we want to store two patterns in a Hopfield network with three neurons:

- Pattern 1: $\mathbf{X}^{(1)} = [1, -1, 1]$
- Pattern 2: $\mathbf{X}^{(2)} = [-1, 1, -1]$
- Step 1: Compute the weight matrix
- We calculate the weight matrix using Hebbian learning:

$$w_{ij} = \frac{1}{n} \sum_{k=1}^p x_i^{(k)} x_j^{(k)}$$

Where:

- $n=3$ (number of neurons),
- $p=2$ (number of patterns).

$$W = \frac{1}{3} \left(\mathbf{x}^{(1)} \cdot (\mathbf{x}^{(1)})^T + \mathbf{x}^{(2)} \cdot (\mathbf{x}^{(2)})^T \right)$$

The weight matrix W is calculated as:

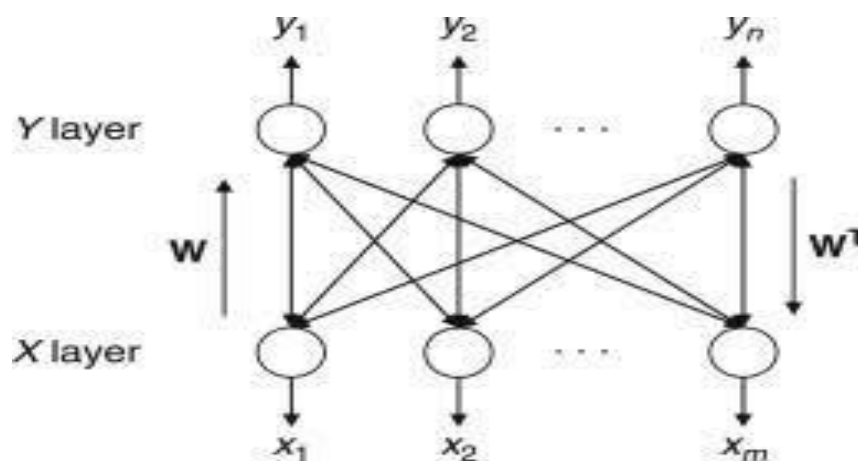
This yields the weight matrix:

$$W = \frac{1}{3} \begin{bmatrix} 2 & -2 & 2 \\ -2 & 2 & -2 \\ 2 & -2 & 2 \end{bmatrix}$$

Lecture 7: Bidirectional Associative Memory (BAM)

Introduction

Bidirectional Associative Memory (BAM) is a type of recurrent neural network introduced by Bart Kosko in 1988. It shares similarities with the Hopfield network but differs in that it consists of two layers of neurons: an input layer and an output layer. The BAM network includes feedback connections from the output layer back to the input layer, allowing each node in the input layer to communicate with every node in the output layer. Unlike Hopfield networks, input nodes do not communicate with themselves, and there are no self-connections among output nodes either.



Bidirectional Associative Memory (BAM) Algorithm

Bidirectional Associative Memory (BAM) is an algorithm that links two sets of patterns (input patterns and output patterns) in a neural network in such a way that it allows retrieval of the corresponding pattern when one of the patterns is input. The algorithm relies on learning the relationship between these patterns and retrieving them in both directions.

Steps of BAM Algorithm:

1- Training Phase:

In this phase, the model is trained on a set of pattern pairs (input pattern X and output pattern Y), learning the relationship between them. This is done by creating a **weight matrix** representing the connections between the input and output layers.

- Suppose we have a set of pattern pairs $X_1, X_2, \dots, X_n$ and $Y_1, Y_2, \dots, Y_n$ where X_i is the input pattern and Y_i is the target pattern (output).
- The weight matrix W is calculated using the following equation:

$$W = \sum_{i=1}^n X_i^T \cdot Y_i$$

Here, each input pattern X_i is multiplied by the corresponding target pattern Y_i , and the outer product is computed to generate the weight matrix.

2- Recall Phase:

After training, the model can retrieve target patterns or input patterns when given either an input pattern or an output pattern.

- **Retrieving Output from Input:** If an input pattern X is provided, the target pattern Y is retrieved using the weight matrix W :

$$Y = W \cdot X$$

Here, the weight matrix W is multiplied by the input pattern X to produce the corresponding target pattern.

- **Retrieving Input from Output:** If an output pattern Y is provided, the input pattern X is retrieved using the inverse relationship with the transposed weight matrix W^T :

$$X = W^T \cdot Y$$

Where W^T is the transpose of the weight matrix W , used to retrieve the input pattern from the output pattern.

BAM is Characterized by

- 1-Bidirectionality
- 2-Full connection
- 3-Feedback
- 4-Fixed weights
- 5-Ability to solve complex problems using linear functions.
- 6-Two levels, one for inputs and one for outputs.
- 7-The number of inputs does not necessarily equal the number of outputs.

Understanding the Iterative Operation of (BAM)

Understanding the Iterative Operation of (BAM)

1- Forward Pass:

In the forward pass of the BAM (Bidirectional Associative Memory) algorithm, the input pattern is fed through the network to retrieve the corresponding target pattern. The steps are:

a) **Receive Input Pattern:** An input pattern X from the set of input patterns is provided.

b) **Compute Target Pattern:**

- The target pattern Y is computed using the weight matrix W :
$$Y = W \cdot X$$
- Here, W is the weight matrix learned during the training phase.

c) **Output Result:**

- The pattern Y obtained is the corresponding target pattern for the input pattern X from the set of target patterns.

2- Backward Pass:

In the backward pass, an output pattern is given to retrieve the corresponding input pattern. The steps are:

a) **Receive Output Pattern:**

- An output pattern Y from the set of target patterns is provided.

b) **Compute Input Pattern:**

- The input pattern X is computed using the transposed weight matrix W^T :
$$X = W^T \cdot Y$$
- Here, W^T is the transpose of the weight matrix W .

3- Output Result:

- The pattern **X** obtained is the corresponding input pattern for the output pattern Y from the set of input patterns.

advantages & disadvantages of BAM

Advantages	Disadvantages
Compatible with Analog Circuits and Visual Systems: BAM can be effectively implemented in analog hardware and visual systems.	Limited Storage Capacity: BAM can only store a limited number of pattern pairs due to its design constraints.
Fast Learning and Retrieval: BAM can quickly learn and retrieve patterns due to its straightforward weight update mechanism.	False Response: BAM might produce incorrect or unexpected outputs due to its limitations in pattern storage and recall.
Resilience Against Noisy Data: BAM is robust and can handle noisy or imperfect data during retrieval.	Sometimes Exhibits Unexpected Behavior: BAM may show unpredictable behavior under certain conditions or with complex patterns.

Example

Let's consider a simple example to illustrate how Bidirectional Associative Memory (BAM) works. We will train the BAM with pairs of input and output patterns and then use it for retrieval.

1-Data Setup:

Suppose we have the following pairs of input and output patterns:

- Input pattern $X_1 = [1, 0, 1]$ and its corresponding output pattern

$$Y_1 = [0, 1, 1]$$

- Input pattern $X_2 = [1, 1, 0]$ and its corresponding output pattern

$$Y_2 = [1, 0, 1]$$

2- Training Phase:

In this phase, we calculate the weight matrix W using the following equation:

$$W = \sum_{i=1}^n X_i^T \cdot Y_i$$

Calculating the Weight Matrix:

- For the first pair:

$$X_1^T = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$X_1^T \cdot Y_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

- For the second pair:

$$X_2^T = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$$X_2^T \cdot Y_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

- Summing the outer products:

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

3- Recall Phase:

Retrieving the Output Pattern from an Input Pattern:

Suppose we want to retrieve the output pattern for the input pattern $X_1 = [1, 0, 1]$:

$$Y = W \cdot X_1 = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 1 \cdot 0 + 2 \cdot 1 \\ 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 \\ 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Similarly, a threshold activation function is used to get the final input pattern.

Lecture 8: Other Neural Network

There are other neural networks that we have not covered in previous lectures.

1- Convolutional Neural Networks (CNNs): Primarily used in image processing and visual pattern analysis. They rely on convolutional layers to detect features in the data.

2- Recurrent Neural Networks (RNNs): Suitable for sequential data like text, speech, and time series. They use feedback loops to retain information from previous steps in the sequence.

3- Radial Basis Function Networks (RBFNs): These networks use radial basis functions to transform data into a non-linear form for classification or regression tasks.

4- Autoencoders: An unsupervised network used for dimensionality reduction and feature extraction by attempting to reconstruct the input data.

5- Generative Adversarial Networks (GANs): Used for generating new data by setting up a competition between two neural networks: a generator and a discriminator.

These various types of neural networks offer different solutions to a wide range of problems.

Lecture 9: Introduction to Genetic Algorithms

Introduction

Genetic Algorithm is an optimization method that mimics the biological process of evolution to find the best or near-best solutions to complex problems. It is widely used in various fields, such as engineering, science, and machine learning, where conventional methods are inefficient or impractical. Optimization means finding the input values that produce the optimal output values according to a mathematical criterion, such as maximizing or minimizing a function. It is a general research algorithm (Global Search Algorithms) whose work is based on natural testing techniques and natural genes. The first to come up with this type of algorithm is the scientist John Holland in 1975 and continues to be widely used in such wide and diverse fields as engineering, economics, biology and computer science.

The model genetic algorithm begins with an initial set of random solutions called "Individuals", each of which is called Chromosome. Chromosome is usually a binary thread and represents a specific solution to the issue to be solved and creates and develops chromosomes during stages as repeats and successive generations and during each generation of these chromosomes is evaluated. The matrimonial factor and the mutation factor are then applied. A new generation is formulated through the selection of some parents or products based on validity values. The group of children competing between them is obtained to form a new society. According to the principle of survival for reform, it is worth mentioning the most valid chromosomes that are usually more fortunate in terms of their choice.

How Genetic Algorithms Work

Genetic Algorithms operate with a population of candidate solutions, each represented as a string of genes. These solutions undergo random combination and mutation to generate new offspring, inheriting traits from their parent solutions. Each solution is assessed using a fitness function that evaluates how well it addresses the problem. The solutions with higher fitness are more likely to survive and reproduce, while those with lower fitness are discarded. This process reflects the Darwinian concept of "survival of the fittest." By iterating this process through multiple generations, GAs progressively evolve more effective solutions until a predefined stopping condition is satisfied.

GAs have different advantages.

- a) Its ability to handle a large number of Coding Parameters and not the parameters themselves.

Looking for a solution from a set of points is not a single point and this enables it to deal with a large number of issues.

- b) Its work depends on target function information, not on derivative and other additional information.
- c) The principle of searching them depends on probability by using selection, marriage and mutation factors and not on fixed and predetermined steps (deterministic).
- d) Another important adjective is its ability to evaluate many possible solutions, which is called implicit parallelism.

GAs also have some limitations

- GAs is not suitable for all problems, especially problems that are simple and have available derivative information.
- GAs requires repeated evaluation of the fitness function, which might be computationally expensive or time-consuming for some problems.
- GAs is stochastic, which means they rely on random processes and do not guarantee the optimality or the quality of the solution.
- GAs may not converge to the optimal solution if they are not implemented properly.

Genetic Algorithm Working Cycle

1- Initialization:

Generating the first sample of solutions at random is the first step in the work of the algorithm. Each member of the group is called the individual and each individual forms a chromosome. The number of chromosomes is called the size of the population determined by the designer. The size of the population affects both the performance and the achievement of the algorithm. Each chromosome consists of a number of sites called genes. The values of these genes vary by the different encryption methods used and the general formula of the individual.

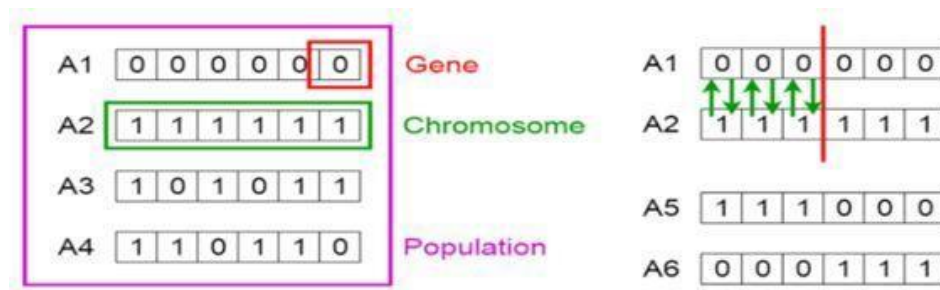
Chromosome i = gen1 gen2 gen3.....gen L .

So, it's L the length of the chromosome.

Choosing the encryption method has a significant impact on the success or failure of the genetic algorithm and the selection process depends on the nature of the problem to be solved and there are many encryptions' methods.

a) Binary Coding:

In genetic algorithms, binary coding is commonly used to represent individuals (potential solutions) in the population. Genetic algorithms are optimization techniques inspired by the principles of natural evolution, such as selection, crossover, and mutation, to find the best solution to a problem. In this species, each chromosome consists of a series of 1.0 numbers.



b) Integer Coding:

In this type of encryption, each location in the chromosome represents a valid number, i.e., each chromosome is a thread of the correct numbers.

Chromosome 1: 4 5 8 8 5

Chromosome 1 :1 5 2 3 7

c) Real Coding:

In this type of encryption, every chromosome site represents a real number, that is, every chromosome is a thread of real setting.

Chromosome 1 :1.5 0.2 3.7 6.4 2.1

Chromosome 1 :2.3 6.8 9.5 3.4 0.4

2-Evaluation:

Each chromosome is evaluated based on a **fitness function**. The fitness function determines how "good" or "fit" the solution is for solving the given problem.

Higher fitness values are assigned to better solutions.

3-Selection

The selection process is a model of survival for reforming nature, that is, the process of selecting parents from society for mating and producing a new generation.

There are many ways of selecting them:

❖ Roulette Wheel Selection:

Roulette Wheel Selection (also known as fitness proportionate selection) is a method used in genetic algorithms to select individuals (chromosomes) for reproduction based on their fitness. The idea is that individuals with higher fitness are more likely to be selected to reproduce, but even less fit individuals have a chance to be selected. This mimics the natural selection process in evolution.

❖ Rank selection:

This technique ranks the individuals according to their fitness values, and assigns a probability of selection based on their ranks. This technique reduces the selection pressure and the risk of losing good individuals due to low fitness values. However, it may also slow down the convergence rate and favor mediocre individuals.

❖ Tournament selection

This technique randomly selects a fixed number of individuals from the population, and chooses the best one among them based on their fitness values. This process is repeated until the desired number of individuals is selected. This technique is robust and flexible, as it can adjust the selection pressure by changing the tournament size.

❖ Stochastic Universal Sampling:

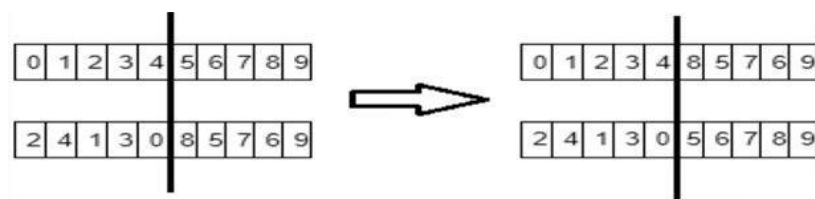
This technique is a modification of roulette wheel selection that ensures a more uniform sampling of individuals. It uses a single random point and equally spaced pointers to select individuals from a cumulative probability distribution. This technique preserves the proportionality of selection and avoids stochastic errors.

4- Crossover operators:

are methods of combining the genetic information of two or more parents to produce offspring. Crossover operators can be classified into different types, depending on how they select and exchange the genes of the parents. Some of the common crossover operators are:

1- Single-point crossover:

This operator randomly selects a point in the chromosome and swaps the genes after that point between the parents. The offspring inherit some genes from each parent.



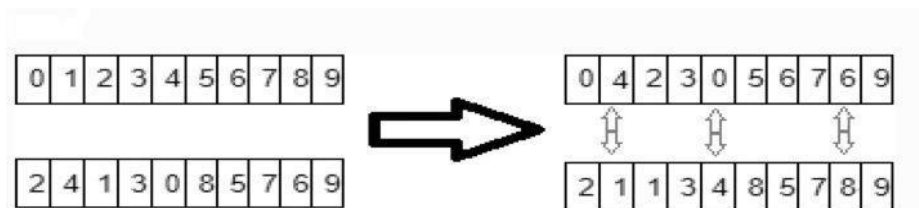
2- Two-point and k-point crossover:

These operators randomly select two or more points in the chromosome and swap the genes between those points between the parents. The offspring inherit different segments of genes from each parent.



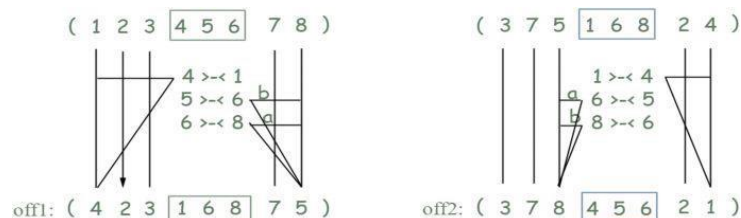
3- Uniform crossover:

This operator treats each gene independently and randomly decides whether to swap it with the corresponding gene of another parent. The offspring inherit a mix of genes from each parent.



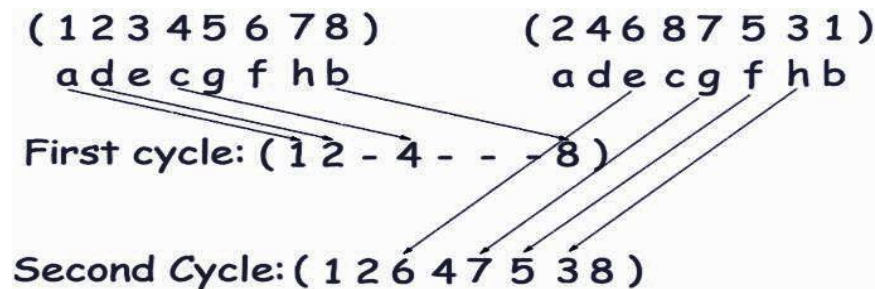
4- Partially matched crossover (PMX):

This operator is often used for permutation problems, where the order of the genes matters. It randomly selects a segment of genes from one parent and copies it too the offspring. Then, it fills in the remaining genes from the other parent, while preserving the order and avoiding duplicates.



5-Shuffle crossover:

This operator shuffles the order of the genes before applying another crossover operator, such as single-point or uniform. Then, it restores the original order after the crossover. This operator can help to preserve the relative positions of the genes.



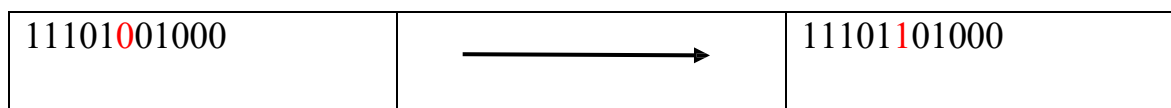
5-Mutation:

is a process that randomly introduces new features into the solution strings of the population pool to maintain diversity in the population. Although crossover is the main operator responsible for exploring the search space, mutation is also used for this purpose and for escaping from local optima. The choice of the appropriate mutation operator depends on the coding scheme and the problem domain. There are several types of mutation, including:

1. Bit Flip Mutation:

Used in binary representations. A randomly chosen bit in the chromosome is flipped from 0 to 1 or from 1 to 0.

❖ Example:



2. Random Mutation:

A random gene in the chromosome is replaced by a randomly chosen value from the allowed set of values.

- ❖ Example:
- ❖ Chromosome (values from 0 to 9): `3 7 2 5 6`
- ❖ After mutation (change gene at index 3): `3 7 2 8 6`

3. Swap Mutation:

Two randomly chosen genes are swapped in their positions. Often used in permutation representations.

- ❖ Example:
- ❖ Before mutation: `A B C D E`
- ❖ After mutation (swap positions 2 and 4): `A E C D B`

4. Inversion Mutation

A section of the chromosome is selected, and the order of the genes within that section is reversed.

- ❖ Example:
- ❖ Before mutation: `1 2 3 4 5 6`
- ❖ After mutation (invert section from index 2 to 4): `1 2 5 4 3 6`

5. Increment/Decrement Mutation:

A randomly chosen gene in the chromosome is either incremented or decremented by a small amount.

❖ Example:

❖ Before mutation: `5 3 8 7 2`

❖ After mutation (increment gene at index 2): `5 3 9 7 2`

6. Scramble Mutation:

A subset of the chromosome is randomly shuffled, while keeping the genes within the subset the same.

❖ Example:

❖ Before mutation: `A B C D E F`

❖ After mutation (scramble section between index 1 and 4): `A D B C E F`

Each mutation type serves a different purpose depending on the representation of the solution and the problem being solved.

6- Replacement:

Substitution mutation in genetic algorithms refers to the replacement of an entire gene or chromosome segment with a new, randomly generated value. It is often used to ensure that the best clone of an individual from a sample generated in the past is introduced into the new sample.

Replacement Mutation Example:

- ❖ Before mutation: 4 9 1 5 7 6
- ❖ After mutation (replace gene at index 3): 4 9 1 2 7 6

In this example, the value at index 3 (which was 5) is replaced with a random new value, 2. Replacement mutation can be used to introduce more diversity by completely replacing parts of a chromosome rather than just altering or swapping specific genes.

There are several ways to replace:

1-Holland's method:

This method replaces the worst individual in the population with a new individual and evaluates the new population. The disadvantage of this method is that it does not compare the fitness of the removed individual and the new individual.

2- Whitley's method

This method randomly selects two or three individuals from the population and identifies the worst one among them. The problem with this method is that selecting two or more individuals randomly from the population does not give a good chance to select the worst one.

Genetic Algorithm Process

The basic steps of a genetic algorithm are as follows:

Step 1: Initialize the parameters of the algorithm, such as the population size, the number of generations, the crossover rate, and the mutation rate.

Step 2: Generate an initial population of random solutions, each encoded as a string of genes (chromosomes).

Step 3: Evaluate the fitness of each solution by using an objective function that measures how well it solves the problem.

Step 4: Repeat until a termination criterion is met (such as reaching the maximum number of generations or finding a satisfactory solution)

Step 5: Select some solutions from the current population to reproduce, based on their fitness values and a selection technique (such as roulette wheel, tournament, or rank selection).

Step 6: Apply crossover operators to combine the genes of the selected solutions and product.

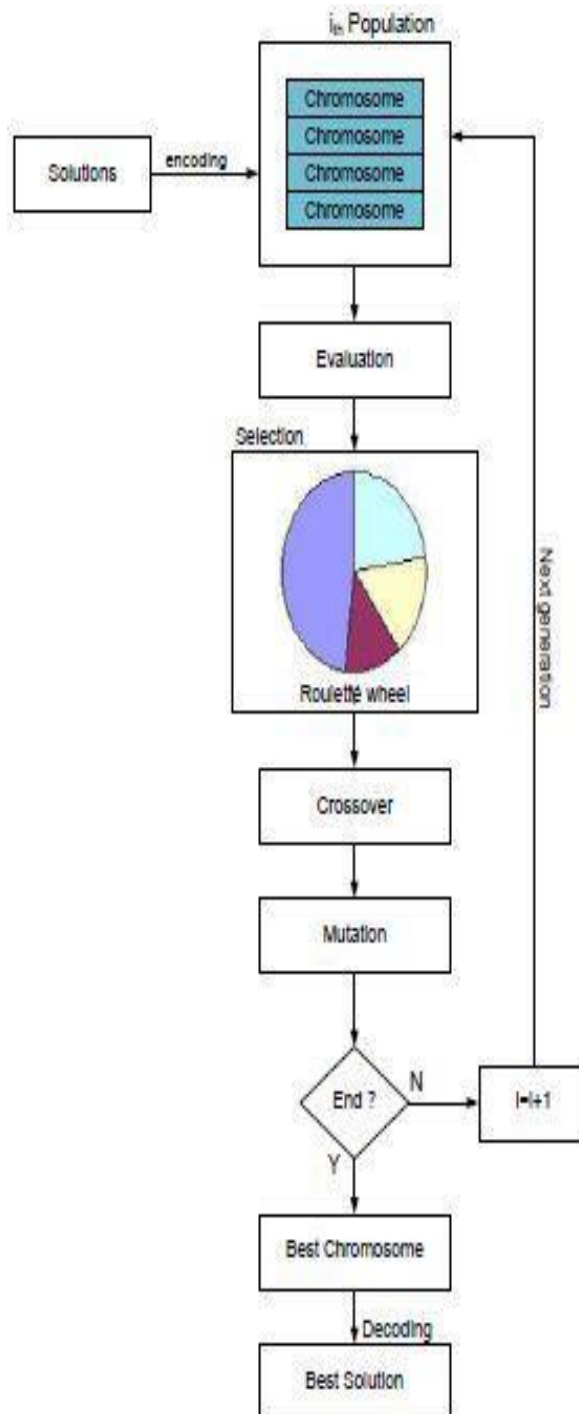
Step 7: Apply mutation operators to randomly modify some genes of the offspring and introduce diversity (such as bit flip, bit inversion, or random reset mutation).

Step 8: Evaluate the fitness of the offspring and replace some or all solutions in the current population with them, based on a replacement technique (such as elitism, generational, or steady-state replacement).

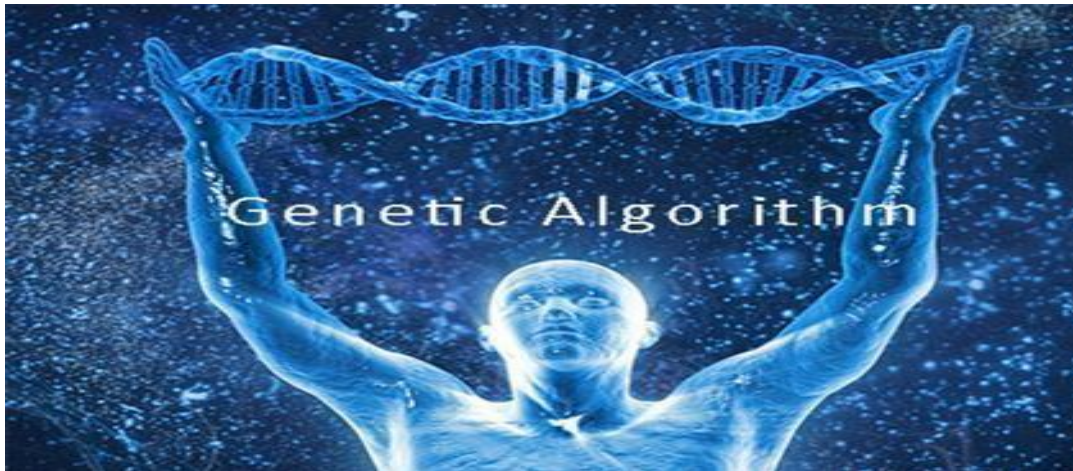
Step 9: Return the best solution found in the final population.

Flowchart

A flowchart of a genetic algorithm based on the previous step description.



Lecture 10: A Step-by-Step from Theory to Practice by Examples



As an example, implementation of a GA, consider the following problem:

population size

Let the length of the bit string be $i=8$, and the number of chromosomes in the

fitness functions

$f(x)$ is equal to the number of 1's in the bit string x .

selection

The probability of selecting is the number of times an individual is expected to reproduce is proportional to its fitness value f_i divided by the average fitness of the population f .

Crossover

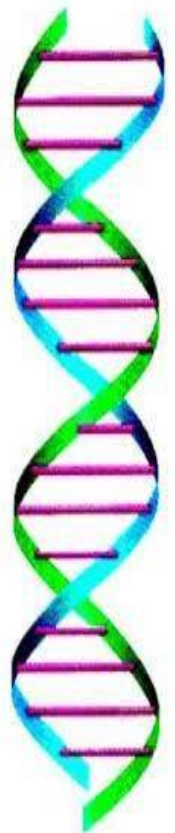
is a 70% chance of applying crossover to two selected parents.

Mutation

is a 0.1% chance of applying mutation to a selected offspring.

Replacement

is a chance that the offspring replace the parents in the next generation.



1

Generation population

The initial randomly generated population of a genetic algorithm can be represented as a table with the following columns:

Chromosome index: A unique identifier for each chromosome in the population.

Chromosome string: A binary string of length i that represents the genetic information of the chromosome.

the initial population of $n=4$ chromosomes with $i=8$ bits might look like this:

Chromosome index	Chromosome string
1	00000110
2	11101110
3	00100000
$N=4$	00110010 $I=8$

2

Generation fitness function

After generating an initial population of candidate solutions, a genetic algorithm's next step is to **evaluate each solution's fitness**. Fitness is a measure of how well a solution solves a problem, and an objective function usually defines it. In this example, the fitness function is the number of ones in the binary string of each chromosome. The fitness values of the initial population are:

Chromosome index	Chromosome string	Fitness (number of ones)
1	00000110	1
2	11101110	6
3	00100000	1
4	00110010	3

3

Generation selection

The selection operator of a genetic algorithm chooses pairs of parent chromosomes from the current population based on their fitness values. The probability of selecting an individual for reproduction is proportional to its fitness value divided by the average fitness of the population. In this example, the fitness values and selection probabilities of the chromosomes are:

Chromosome index	Chromosome string	Fitness (f)	Selection probability
1	00000110	2	$0.50 \approx 1$
2	11101110	6	$1.50 \approx 2$
3	00100000	1	$0.33 \approx 0$
4	00110010	3	$1 \approx 1$

The average fitness of the population is $f = (2+6+1+3)/4=3$. The selection operator chooses two parents at a time, based on their selection probabilities, and applies crossover and mutation operators to produce new offspring.

4

Generation crossover

The crossover operator of a genetic algorithm combines the genetic information of the selected parents to produce new offspring. In this example, the crossover probability is $p=0.7$, which means that there is a 70% chance of applying crossover to two selected parents. To decide whether to apply crossover to a pair of parents, a random number is generated in the range between 0 and 1. If the random number is less than or equal to the crossover probability p , then crossover is applied to the parents at a randomly chosen locus. Otherwise, the parents do not undergo crossover, and their offspring are identical copies of themselves.

For example, consider the following pairs of parents for crossover:

1st pair: strings 2 and 1

2nd pair: strings 2 and 4

For the first pair, the random number $r=0.4$ is less than the crossover probability $p=0.7$, so crossover is applied at a randomly chosen locus. The offspring of the first pair are:

Chromosome index	Chromosome string
5	11000110
6	00101110

For the second pair, the random number $r=0.8$ is greater than the crossover probability $p=0.7$, so no crossover is applied. The offspring of the second pair are identical copies of their parents:

Chromosome index	Chromosome string
2	11101110
4	00110010

Generation mutation

5

The mutation operator of a genetic algorithm randomly modifies some genes of the offspring to introduce diversity. In this example, the mutation probability is $p=0.001$, which means that there is a 0.1% chance of applying mutation to a selected offspring. To decide whether to apply mutation to an offspring, a random number is generated in the range between 0 and 1. If the random number is less than or equal to the mutation probability p , then the mutation is applied to the offspring at a randomly chosen locus. Otherwise, the offspring is an identical copy of its parent.

For the first pair, the offspring of the crossover are:

Chromosome index	Chromosome string
5	11000110
6	00101110

The mutation operator does not apply to any of the offspring of the first pair, because the random number $r=0.4$ & $r=0.1$ is greater than the mutation probability $p=0.001$.

For the second pair, the crossover is:

Chromosome index	Chromosome string
2	11101110
4	00110010

The mutation operator applies to the second Chromosome at locus 2, because the random number $r=0.0004$ is less than the mutation probability $p=0.001$. The mutated offspring is:

Chromosome index	Chromosome string
7	00110110

The mutation operator does not apply to the first Chromosome of the second pair, because the random number $r=0.8$ is greater than the mutation probability $p=0.001$.

6

Generation replacement

The replacement operator of a genetic algorithm removes some low-fitness individuals from the population and replaces them with new offspring. In this example, the replacement operator is generational, which means that the offspring replace the parents in the next generation. The new population of chromosomes after replacement and the fitness values and selection probabilities of the chromosomes are:

Chromosome index	Chromosome string	Fitness (1's)	Selection probability
2	11101110	6	1
5	11000110	4	1
6	00101110	4	1
7	00110110	4	1

The average fitness of the population is $f = (6+4+4+4)/4 = 4.5$.

The selection operator chooses two parents at a time, based on their selection probabilities, and applies crossover and mutation operators to produce new offspring. This process is repeated until the end of three generations

Lecture 12: GA in Travelling Sales Man Problem Solving

Traveling Sales man Problem (TSP)

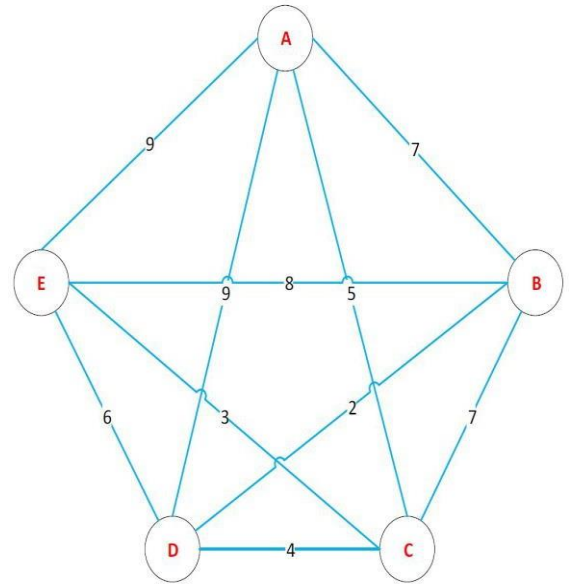
One classical example is the Travelling Salesman problem (TSP), which is a well-known optimization problem that has been studied for many years. In this problem, there are a number of cities that a salesman must visit, starting from the first city and returning to it after visiting all the other cities exactly once. The objective of the TSP is to find the shortest possible route that visits every city exactly once and returns to the starting point. The main difficulty is that the number of combinations (and, hence, the number of possible solutions) grows much faster than the number of items involved in the problem (i.e., the number of cities in TSP)

The following are the key characteristics of the TSP problem

1	The journey begins and ends at the initial city.
2	There are several cities, all of which must be visited exactly
3	The salesperson must not return to the initial city before visiting all the destination cities.
4	The purpose is to minimize the distance travelled by the salesperson by arranging the order of cities to be visited.

Graph of TSP

For example, suppose there are five cities that a salesperson must visit: A, B, C, D, and E. The journey starts and ends at city A. The order in which the salesperson visits the cities can be represented by a chromosome, such as $P = (C1, C2, C3, C4, C5)$, where C1 to C5 represent the cities in the order they are visited. Now Genetic operators such as crossover and mutation are used to create new offspring from the selected chromosomes, which can lead to better solutions.



A Systematic Guide to Solving the TSP Using GA

1	A	B	7
2	A	C	5
3	A	D	9
4	A	E	9
5	B	C	7
6	B	D	2
7	B	E	8
8	C	D	4
9	C	E	3
10	D	E	6

Initial Population

Now to find the smallest global optimum value:

1- Create an initial population of possible solutions to the TSP. You can do this by randomly generating a set of permutations of the cities.

1	A	B	D	E	C	A
2	A	D	B	E	C	A
3	A	C	B	D	E	A
4	A	E	B	C	D	A
5	A	E	C	B	D	A
6	A	C	D	E	B	A

Fitness Function

2- Calculate the fitness of each solution, which is the total distance travelled by the salesman. You can use a distance matrix to calculate the distance between each pair of cities.

	Path					Fitness
1	AB	BD	DE	EC	CA	23
	7	2	6	3	5	
2	AD	DB	BE	EC	CA	27
	9	2	8	3	5	
3	AC	CB	BD	DE	EA	29
	5	7	2	6	9	
4	AE	EB	BC	CD	DA	37
	9	8	7	4	9	
5	AE	EC	CB	BD	DA	30
	9	3	7	2	9	
6	AC	CD	DE	EB	BA	30
	5	4	6	8	7	

Selection

3- Select the best solutions based on smaller fitness. by sorting ascending order of their fitness values. Use a roulette wheel selection to choose the solutions. We can then select based on their probability values.

Sum =23+27+29+37+30+30=176 **Average** = 176/6=29.3

New	old	Path					Fitness	Probability
1	1	AB	BD	DE	EC	CA	23	0.784
		7	2	6	3	5		
2	2	AD	DB	BE	EC	CA	27	0.92
		9	2	8	3	5		
3	3	AC	CB	BD	DE	EA	29	0.988
		5	7	2	6	9		
4	6	AC	CD	DE	EB	BA	30	1.022
		5	4	6	8	7		
5	5	AE	EC	CB	BD	DA	30	1.022
		9	3	7	2	9		
6	4	AE	EB	BC	CD	DA	37	1.261
		9	8	7	4	9		

Crossover

Crossover the selected solutions to create new offspring.

To apply crossover on the next chromosome, we can select chromosomes 1 and 2 as parents based on their probability values. We can then perform crossover by selecting a random crossover point and swapping the genes between the two parents. in this example, we select the crossover point to be between the second and third genes, we can create two offspring:

Offspring 1: A**DB**ECA

Offspring 2: A**BDE**CA

Mutation

5- Mutate the offspring to introduce variations. We can then mutate the offspring by randomly swapping two genes to introduce variations. in this example, we can swap the second and fourth genes in offspring 1 to create a new chromosome:

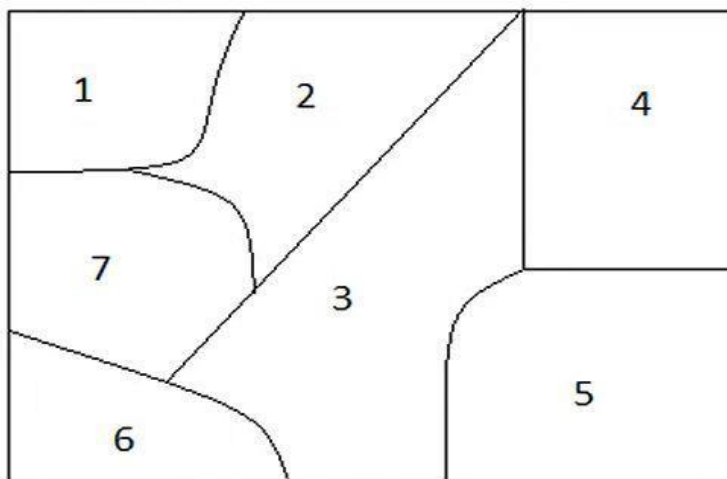
Offspring 1 (mutated): A**D**B**E**CA

We can repeat this process for multiple generations until we find a satisfactory solution.

Lecture 13: GA in color mapping Problem Solving

Graph Coloring Using GA for the 4-Color Map Problem

This lecture describes the application of a genetic algorithm (GA) to solve the 4-color map problem for a map with 7 cities. The problem aims to assign colors to the cities such that no two adjacent cities share the same color.



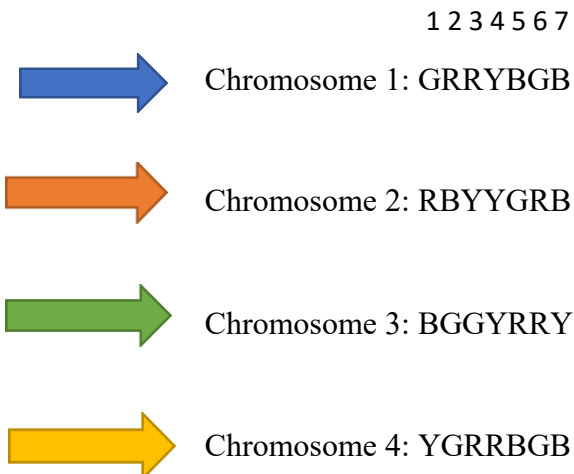
Graph Coloring Using GA for the 4-Color Map Problem

The problem of graph coloring can be solved using genetic algorithms. The map can be represented by a set of nodes and edges that connect them. Each node is represented by a number that denotes the color assigned to it. New solutions are generated by modifying the current solutions using selection, crossover, and mutation operations. The fitness function is used to evaluate the quality of the generated solutions.

	1	2	3	4	5	6	7
The first city	0	1	0	0	0	0	1
The second city	1	0	1	0	0	0	1
The third city	0	1	0	1	1	1	1
Fourth city	0	0	1	0	1	0	0
Fifth city	0	0	1	1	0	0	0
Sixth city	0	1	1	0	0	0	1
Seventh city	1	1	1	0	0	1	0

Initial Population

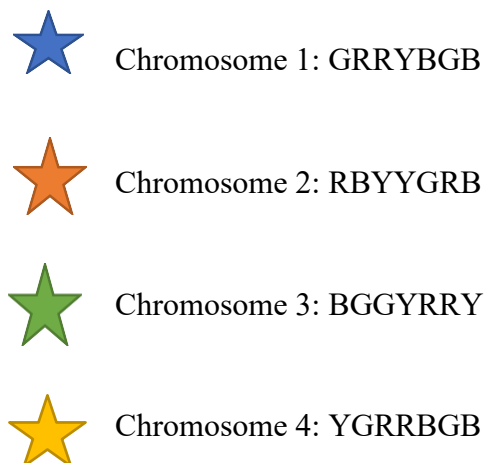
The GA randomly generates 4 solutions (population size) as follows:



Fitness Function

The fitness function evaluates each chromosome by counting the number of color conflicts (adjacent cities with the same color). The lower the number of conflicts, the higher the fitness.

Fitness Values



Selection and Crossover

The selection operation chooses Chromosomes 1 and 2 for crossover based on their fitness (higher fitness chromosomes are more likely to be selected).

Crossover occurs at the second position

G R R Y B G B
R B Y Y G R B

resulting in two new chromosomes:

New Chromosome 1: **R B R Y B G B**
New Chromosome 2: **G R Y Y G R B**

Mutation

Mutation randomly changes individual genes in the new chromosomes. This can introduce new diversity into the population.

New Chromosome 1

(Last gene mutated)
R B R Y B G Y

New Chromosome 2

(3rd gene mutated)
G R B Y G R B

Updated Fitness Values

Compute the fitness function for the 2 new chromosomes

New Chromosome 1: **R B R Y B G Y** 0 conflicts

New Chromosome 2: **G G Y Y G R B** 2 conflicts

Chromosome 1, with no color conflicts, represents a valid solution to the 4-color map problem. The corresponding coloring is depicted in the map below:

R B R Y B G Y

This solution demonstrates the effectiveness of the GA in finding a solution to the 4-color map problem.

Lecture 14: GA in the 8 Puzzle Problem Solving

8-puzzle problem using genetic algorithm

This lecture focuses on applying genetic algorithms to the classic 8-puzzle problem in artificial intelligence. It explores the fundamentals of genetic algorithms through the lens of the 8-puzzle problem. The previous lecture discussed state-space search, which involves starting from a current state and searching for a goal state by evaluating the number of moves required to reach the goal from the current state. In this lecture, the concept of population size is introduced, which is determined by the number of randomly generated chromosomes based on the difference between the current and goal states. These chromosomes represent potential solutions to the puzzle. The lecture then focuses on two key genetic operators: crossover and mutation, using the “ordered chromosomes” method for performing crossover.

Initial Population

The initial population of the 8-puzzle problem consists of four chromosomes, each represented by a sequence of numbers:

X1: [2 3 1 0 4 6 5 7 8]

X2: [4 3 2 1 0 5 6 7 8]

X3: [0 8 7 6 4 3 2 1 5]

X4: [8 7 6 5 3 4 1 0 2]

The target goals

The goal state of the puzzle is represented by the target chromosome:

[1 2 3 4 0 5 6 7 8]

Fitness Values

The initial population is evaluated with the fitness value for each chromosome, which is the number of tiles in place according to the target goal chromosome.

Target Goal: [1 2 3 4 0 5 6 7 8]

X1: [2 3 1 0 4 6 5 7 8]	correct position = 2
X2: [4 3 2 1 0 5 6 7 8]	correct position = 4
X3: [0 8 7 6 4 3 2 1 5]	correct position = 0
X4: [8 7 6 5 3 4 1 0 2]	correct position = 0

Mutation

then do mutation in the 4th chromosome by swapping the 1st gen with the 7th gen. After reproduction, we obtain the following chromosomes:

X8: [1 7 5 6 4 3 8 0 2] with fitness 1

- **X5:** [3 4 6 1 0 5 7 8 2] correct position = 2
- **X6:** [2 1 5 0 4 6 7 8 3] correct position = 0
- **X7:** [8 7 6 5 3 4 2 1 0] correct position = 0
- **X8:** [1 7 5 6 4 3 8 0 2] correct position = 1

Reproduction

The best chromosomes from the combined initial and new populations are selected for the next generation:

- **X2:** [4 3 2 1 0 5 6 7 8] correct position = 4
- **X1:** [2 3 1 0 4 6 5 7 8] correct position = 2
- **X5:** [3 4 6 1 0 5 7 8 2] correct position = 2
- **X8:** [1 7 5 6 4 3 8 0 2] correct position = 1