University of Technology - Iraq

Department of Computer Sciences – Multimedia Branch

Adaptive Systems

Third Class – Second Course

Lecturer: Asst. Prof. Dr. Hiba Basim Alwan

2024 – 2025

JANUARY 17, 2025

ADAPTIVE SYSTEMS

ASST. PROF. DR. HIBA BASIM ALWAN HUSSAIN
UNIVERSITY OF TECHNOLOGY – DEPARTMENT OF COMPUTER SCIENCE
Multimedia Branch – Third Stage

Chapter One: Neural Networks

1.1. Background

The Artificial Neural Network (ANN), or simply Neural Network (NN), is a machine learning method evolved from the idea of simulating the human brain, which consists of processing elements (called neurons or nodes), and connections between them with coefficients (weights) bound to the connections. NNs "learn" from examples (as children learn to recognize dogs from examples of dogs) and exhibit some capability for generalization beyond the training data.

In general, an ANN can be divided into three parts, named layers, which are known as:

a. **Input layer:** This layer is responsible for receiving information (data), signals, features, or measurements from the external environment. These inputs (samples or patterns) are usually normalized within the limit values produced by activation functions. This normalization results in better numerical precision for the mathematical operations performed by the network.
b. **Hidden, intermediate, or invisible layers:** These layers are composed of neurons which are responsible for extracting patterns associated with the process or system being analyzed. These layers perform most of the internal processing from a network.
c. **Output layer:** This layer is also composed of neurons, and thus is responsible for producing and presenting the final network outputs, which result from the processing performed by the neurons in the previous layers.

The main architectures of ANN, considering the neuron disposition, as well as how they are interconnected and how its layers are composed, can be divided as follows:

– **Single-layer network**

  This ANN has just one input layer and a single neural layer, which is also the output layer. Fig. 1.1 illustrates a simple single-layer network composed of *n* inputs and *m* outputs. The information always flows in a single direction (thus,

unidirectional), which is from the input layer to the output layer. From Fig. 1.1, it is possible to see that in networks belonging to this architecture, the number of network outputs will always coincide with the number of neurons.

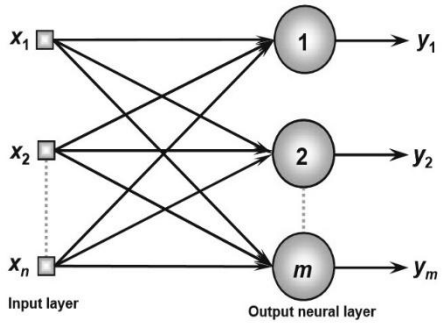

Fig. 1.1: Single-layer network

– **Multi-layer network**

Multiple layers are composed of one or more hidden neural layers. Fig. 1.2 shows an NN with multiple layers composed of one input layer with *n* sample signals, two hidden neural layers consisting of n_1 and n_2 neurons respectively, and, one output neural layer composed of *m* neurons representing the respective output values of the problem being analysed. It is possible to understand that the number of neurons composing the first hidden layer is usually different from the number of signals composing the input layer of the NN. The number of hidden layers and their respective number of neurons depend on the nature and complexity of the problem being mapped by NN, as well as the quantity and quality of the available data about the problem.

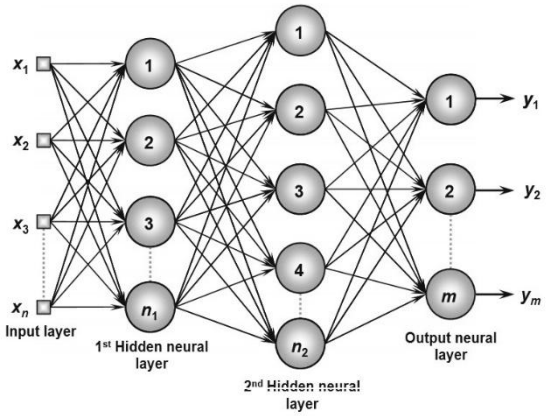

Fig. 1.2: Multi-layer network

– **Full connected network**

In a fully connected network, the output of a node returns as input to all other nodes, but not to the same node as shown in Fig. 1.3.

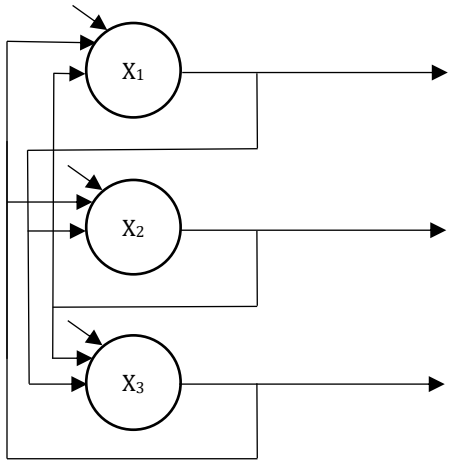

Fig. 1.3: Full connected network

There are two types of connections between nodes. One is a one-way connection with no loopback (feedforward network). Because the signal travels one way only, the feedforward network is static; that is, one input is associated with one particular output. Fig. 1.4 shows this type.

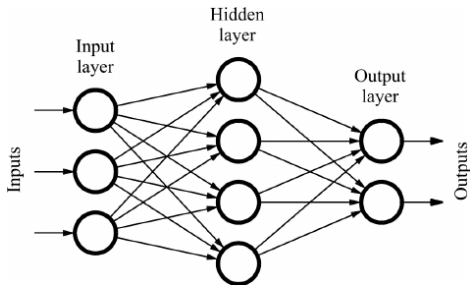

Fig. 1.4: Feedforward network

The other is a loop-back connection in which the output of the nodes can be the input to previous or same-level nodes (feedback/recurrent network). The feedback network is dynamic. For one input, the state of the feedback network changes for many cycles, until it reaches an equilibrium point, so one input produces a series of outputs. Fig. 1.5 illustrates this type.

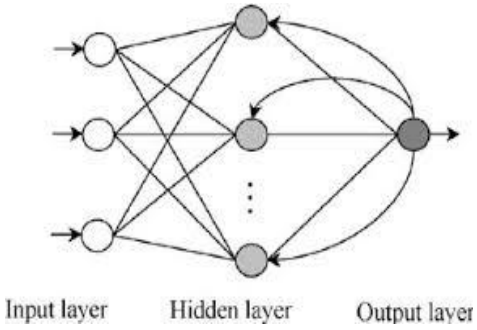

Fig. 1.5: Feed backward network

The problem-solving process using neural networks consists of two major phases and they are:

a. **Training phase:** During this phase, the network is trained with training examples and the rules are inserted in its structure.
b. **Recall phase/Testing phase:** When new data is fed to the trained network the recall algorithm is used to calculate the results.

1.2. The Neuron: Biological and Simulated Neuron

The basic component of an ANN is an artificial neuron like a biological neuron (see Fig. 1.6) in a biological neural network. A biological neuron may be modelled artificially to perform computation and then the model is termed an artificial neuron.

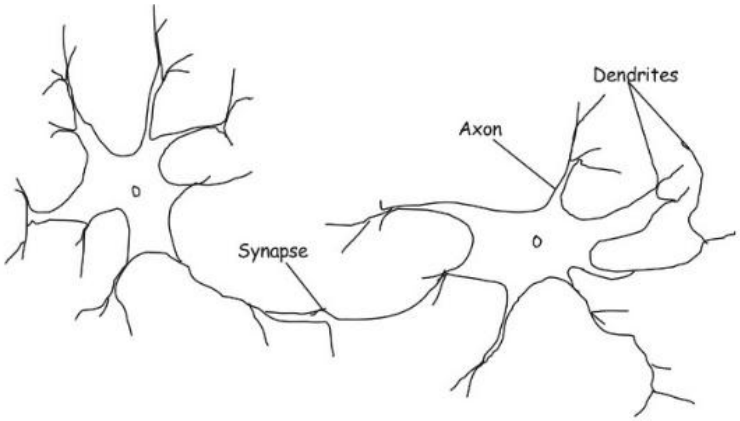

Fig. 1.6: Biological neuron

A neuron is the basic processor or processing element in a neural network. Each neuron receives one or more inputs over these connections (i.e., synapses) and produces only one output. Also, this output is related to the state of the neuron and its activation function. This output may fan out to several other neurons in the network. The inputs are the outputs i.e., activations of the incoming neurons multiplied by the connection or synaptic weights. Each weight is associated with an input of a network. The activation of a neuron is computed by applying a threshold function (popularly known as activation function) to the weighted sum of the inputs plus a bias. Fig. 1.7 represents an artificial neuron.

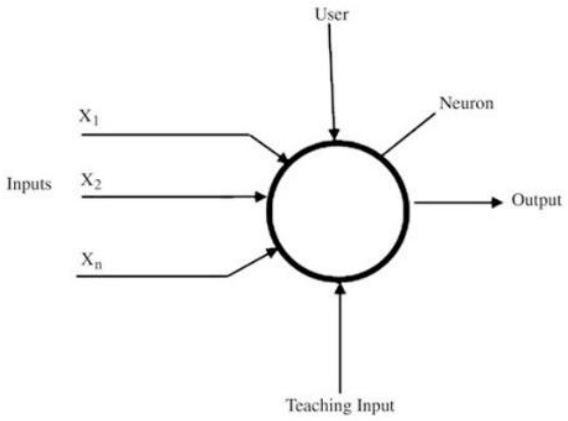

Fig. 1.7: An artificial neuron

1.3. Types of Learning Strategies

It is the algorithm that is used to train the ANNs. Lots of research has been carried out in trying various phenomena and it gives the researchers an enormous amount of flexibility and opportunity for innovation and discussing the complete set of learning algorithms. Nevertheless, the learning algorithms so far used are currently classified into three groups. These groups are:

– **Supervised learning:** The training examples consist of input vectors x and the desired output vector y and training is performed until the neural network "learns" to associate each input vector x to its corresponding output vector y (approximate a function y = f (x)). It encodes the example in its internal structure.

- **Unsupervised learning:** Only input vector x is supplied and the neural network learns some internal feature of the complete set of all the input vectors presented to it. Contemporary unsupervised algorithms are further divided into two (i) non-competitive and (ii) competitive.
- **Reinforcement learning:** Also referred to as reward penalty learning. The input vector is presented, and the neural network can calculate the corresponding output if it is good then the existing connection weights are increased (rewarded), otherwise the connection weights involved are decreased (punished).

1.4. Back Propagation Neural Network

Although many NN models have been proposed, the Back Propagation (BP) NN is the most widely used model in terms of practical applications. It is a powerful mapping network that has been successfully applied to a wide variety of problems ranging from credit application scoring to image compression.

BP NN is a systematic method for training multi-layer NN. It has a mathematical foundation that is strong if not highly practical. BP NN is usually layered with each layer fully connected to the layers before and after neurons are not connected to other neurons in the same layer. Typically, BP NN employs three or more layers of neurons including the input layer.

BP NN training algorithm is as follows:

Step 1: Initialize weights to small random values

Step 2: While the stopping condition is false do steps 3 to 10

Step 3: for each training pair do steps 4 to 9

Feedforward

Step 4: each input unit (x_i, i = 1 to n) receives input signal xi and broadcasts this signal to all units in the layer above (the hidden layer)

Step 5: each hidden layer unit (z_j, j = 1 to p) sums its weighted input signals

$$z_{in_j} = v_{0i} + \sum x_j \, v_{ij}$$

and applies its binary sigmoid activation function to compute its output signal and sends this signal to all units in the layer above (the output layer)

Step 6: each output unit (y_k, k = 1 to m) sums its weighted input signal

$$y_{in_k} = w_{0k} + \sum z_j \, w_{jk}$$

and applies its binary sigmoid activation function to compute its output signal

Back propagate error

Step 7: each output unit (y_k, k = 1 to m) receives a target pattern corresponding to the input training patterns, computes its error information term and calculates its weights correction term used to update w_{jk} later

$$\delta_{2k} = y_k(1 - y_k)(T_k - y_k)$$

where T_k is the target pattern and k = 1 to m

Step 8: each hidden unit (z_j, j = 1 to p) computes its error information term and calculates its weights correction term used to update v_{ij} later

$$\delta_{1j} = z_j(1 - z_j) \sum \delta_{2k} \, w_{jk}$$

Update weights

Step 9: each output unit (y_k, k = 1 to m) updates its weights

$$new \; w_{jk} = \eta \delta_{2k} z_j + \alpha \; old \; w_{jk} \qquad j = 1 \; to \; p$$

each hidden unit (z_j, j = 1 to p) updates its weights

$$new \; v_{ij} = \eta \delta_{1j} x_i + \alpha \; old \; v_{ij} \qquad i = 1 \; to \; n$$

Step 10: test stopping condition

Example: suppose you have BP NN with 2 input, 2 hidden, and 1 output nodes with the following matrices weight where the moment coefficient α = 0.9, learning rate η = 0.45, x = (1 0) and T = 1. Trace this BP NN with two iterations.

$$v = \begin{bmatrix} 0.10 & -0.3 \\ 0.75 & 0.2 \end{bmatrix}, w = \begin{bmatrix} 0.3 \\ -0.5 \end{bmatrix}$$

Solution:

First iteration

$z_{in_j} = v_{0i} + \sum x_j \, v_{ij}$

$z_{in_1} = x_1 v_{11} + x_2 v_{21} = 1 * 0.1 + 0 * 0.75 = 0.1$

$z_{in_2} = x_1 v_{12} + x_2 v_{22} = 1 * (-0.3) + 0 * 0.2 = -0.3$

$z_j = f\left(z_{in_j}\right) = \frac{1}{1+e^{-z_{in_j}}}$

$z_1 = f\left(z_{in_1}\right) = \frac{1}{1+e^{-0.1}} = 0.5$

$z_2 = f\left(z_{in_2}\right) = \frac{1}{1+e^{-0.3}} = 0.426$

$y_{in_k} = w_{0k} + \sum z_j \, w_{jk}$

$y_{in_1} = z_1 w_{11} + z_2 w_{21} = 0.5 * 0.3 + 0.426 * (-0.5) = -0.063$

$y_k = f\left(y_{in_k}\right) = \frac{1}{1+e^{-y_{in_k}}}$

$y_1 = f\left(y_{in_1}\right) = \frac{1}{1+e^{-0.063}} = 0.484$

$\delta_{2k} = y_k(1 - y_k)(T_k - y_k)$

$\delta_{21} = 0.484(1 - 0.484)(1 - 0.484) = 0.129$

$\delta_{1j} = z_j\left(1 - z_j\right) \sum \delta_{2k} \, w_{jk}$

$\delta_{11} = 0.5(1 - 0.5) * 0.129 * 0.3 = 0.009$

$\delta_{12} = 0.426(1 - 0.426) * 0.129 * (-0.5) = -0.016$

$new\ w_{jk} = \eta \delta_{2k} z_j + \alpha\ old\ w_{jk} \qquad j = 1\ to\ p$

$new\ w_{11} = 0.45 * 0.129 * 0.5 + 0.9 * 0.3 = 0.299$

$new\ w_{21} = 0.45 * 0.129 * 0.426 + 0.9 * (-0.5) = -0.425$

$new\ v_{ij} = \eta \delta_{1j} x_i + \alpha\ old\ v_{ij} \qquad i = 1\ to\ n$

$new\ v_{11} = 0.45 * 0.009 * 1 + 0.9 * 0.1 = 0.094$

$new\ v_{12} = 0.45 * (-0.016) * 1 + 0.9 * (-0.3) = 0.277$

$new\ v_{21} = 0.45 * 0.009 * 0 + 0.9 * 0.75 = 0.675$

$new\ v_{22} = 0.45 * (-0.016) * 0 + 0.9 * 0.2 = 0.18$

$$v = \begin{bmatrix} 0.094 & 0.277 \\ 0.675 & 0.180 \end{bmatrix}, w = \begin{bmatrix} 0.299 \\ -0.425 \end{bmatrix}$$

Second iteration

It is your homework

1.5. Hopfield Neural Network

The Hopfield network model is probably one of the most popular types of NN. There are several versions of Hopfield networks. They can be used as auto-associated memory.

The basic idea of the Hopfield network is that it can store a set of exemplar patterns as multiple stable states. Given a new input pattern, which may be partial or noisy, the network can converge to one of the exemplar patterns that is nearest to the input pattern. This is the basic concept of applying the Hopfield network as associative memory.

The Hopfield network consists of a single layer of neurons. The network is fully interconnected; that is, every neuron in the network is connected to every other neuron. The network is recurrent; that is, it has feed forward/feed backward

capabilities, which means input to the neurons comes from external inputs as well as from the neurons themselves internally. Each input/output, x_i or y_i, takes discrete bipolar values of either +1 or -1.

There are two types of Hopfield NN; continuous Hopfield NN and discrete Hopfield NN. The algorithm of discrete Hopfield NN is shown below:

Step 1: Initialize weights to store patterns

Step 2: While activations of the net are not converged do steps 3 to 9

Step 3: For each input vector x do steps 4 to 8

Step 4: Set initial activations of net equal to the external input vector

$$y_i = x_i \text{ for } i = 1 \text{ to } n$$

Step 5: Do steps 6 to 8 for each unit y_i

Step 6: Compute net input

$$y_{in_j} = x_i + \sum y_j \, w_{ji}$$

Step 7: Determine activation (output signal, **note:** the standard steps activation function)

Step 8: Broadcast the value of y_i to all other units

Step 9: Test for convergence

Example: Consider the following samples are stored in a net

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The binary input is (1 1 1 0). We want to train the net using Hopfield to know which of the samples the input is near.

Solution:

$w_{ij} = \sum_{p=1}^{n}(2x_i^p - 1)(2x_j^p - 1) \qquad w_{ij} = w_{ji} \qquad w_{ii} = 0$

$w_{ij} = (2x_i^p - 1)(2x_j^p - 1)$

$[0 \quad 1 \quad 0 \quad 0]$

$w_{12} = (2(0) - 1)(2(1) - 1) = -1$

$w_{13} = (2(0) - 1)(2(0) - 1) = +1$

$w_{14} = (2(0) - 1)(2(0) - 1) = +1$

$w_{23} = (2(1) - 1)(2(0) - 1) = -1$

$w_{24} = (2(1) - 1)(2(0) - 1) = -1$

$w_{34} = (2(0) - 1)(2(0) - 1) = +1$

$$w_1 = \begin{bmatrix} 0 & -1 & +1 & +1 \\ -1 & 0 & -1 & -1 \\ +1 & -1 & 0 & +1 \\ +1 & -1 & +1 & 0 \end{bmatrix}$$

$[1 \quad 1 \quad 0 \quad 0]$

$w_{12} = (2(1) - 1)(2(1) - 1) = +1$

$w_{13} = (2(1) - 1)(2(0) - 1) = -1$

$w_{14} = (2(1) - 1)(2(0) - 1) = -1$

$w_{23} = (2(1) - 1)(2(0) - 1) = -1$

$w_{24} = (2(1) - 1)(2(0) - 1) = -1$

$w_{34} = (2(0) - 1)(2(0) - 1) = +1$

$$w_2 = \begin{bmatrix} 0 & +1 & -1 & -1 \\ +1 & 0 & -1 & -1 \\ -1 & -1 & 0 & +1 \\ -1 & -1 & +1 & 0 \end{bmatrix}$$

$[0 \quad 0 \quad 1 \quad 1]$

$w_{12} = (2(0) - 1)(2(0) - 1) = +1$

$w_{13} = (2(0) - 1)(2(1) - 1) = -1$

$w_{14} = (2(0) - 1)(2(1) - 1) = -1$

$w_{23} = (2(0) - 1)(2(1) - 1) = -1$

$w_{24} = (2(0) - 1)(2(1) - 1) = -1$

$w_{34} = (2(1) - 1)(2(1) - 1) = +1$

$$w_3 = \begin{bmatrix} 0 & +1 & -1 & -1 \\ +1 & 0 & -1 & -1 \\ -1 & -1 & 0 & +1 \\ -1 & -1 & +1 & 0 \end{bmatrix}$$

$w = w_1 + w_2 + w_3$

$$w = \begin{bmatrix} 0 & -1 & +1 & +1 \\ -1 & 0 & -1 & -1 \\ +1 & -1 & 0 & +1 \\ +1 & -1 & +1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & +1 & -1 & -1 \\ +1 & 0 & -1 & -1 \\ -1 & -1 & 0 & +1 \\ -1 & -1 & +1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & +1 & -1 & -1 \\ +1 & 0 & -1 & -1 \\ -1 & -1 & 0 & +1 \\ -1 & -1 & +1 & 0 \end{bmatrix}$$

$$w = \begin{bmatrix} 0 & +1 & -1 & -1 \\ +1 & 0 & -3 & -3 \\ -1 & -3 & 0 & +3 \\ -1 & -3 & -3 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} -1 & +1 & -1 & -1 \\ +1 & +1 & -1 & -1 \\ -1 & -1 & +1 & +1 \end{bmatrix}$$

The input vector x = (1 1 1 0), for this vector:

y = x = (1 1 1 0)

$y_{in_j} = x_i + \sum y_j \, w_{ji}$

$y_{in_1} = x_1 + \sum y_j \, w_{i1} = 1 + 0 = 1 > 0, \therefore y_{in_1} = 1, \therefore y = (1\ 1\ 1\ 0)$

$y_{in_2} = x_2 + \sum y_j \, w_{i2} = 1 - 2 = -1 < 0, \therefore y_{in_2} = 0, \therefore y = (1\ 0\ 1\ 0)$

$y_{in_3} = x_3 + \sum y_j \, w_{i3} = 1 - 4 = -3 < 0, \therefore y_{in_3} = 0, \therefore y = (1\ 0\ 0\ 0)$

$y_{in_4} = x_4 + \sum y_j \, w_{i4} = 0 - 1 = -1 < 0, \therefore y_{in_4} = 0, \therefore y = (1\ 0\ 0\ 0)$

Test for convergence, false, so continue

The input vector x = (1 0 0 0), for this vector:

y = x = (1 0 0 0)

$y_{in_j} = x_i + \sum y_j \, w_{ji}$

$y_{in_1} = x_1 + \sum y_j \, w_{i1} = 1 + 0 = 1 > 0, \therefore y_{in_1} = 1, \therefore y = (1\ 0\ 0\ 0)$

$y_{in_2} = x_2 + \sum y_j \, w_{i2} = 0 + 1 = +1 > 0, \therefore y_{in_2} = 1, \therefore y = (1\ 1\ 0\ 0)$

$y_{in_3} = x_3 + \sum y_j \, w_{i3} = 0 - 1 = -1 < 0, \therefore y_{in_3} = 0, \therefore y = (1\ 1\ 0\ 0)$

$y_{in_4} = x_4 + \sum y_j \, w_{i4} = 0 - 1 = -1 < 0, \therefore y_{in_4} = 0, \therefore y = (1\ 1\ 0\ 0)$

Test for convergence, true, try again

The input vector x = (1 1 0 0), for this vector:

y = x = (1 1 0 0)

$y_{in_j} = x_i + \sum y_j \, w_{ji}$

$y_{in_1} = x_1 + \sum y_j \, w_{i1} = 1 + 1 = 2 > 0, \therefore y_{in_1} = 1, \therefore y = (1\ 1\ 0\ 0)$

$y_{in_2} = x_2 + \sum y_j \, w_{i2} = 1 + 1 = +2 > 0, \therefore y_{in_2} = 1, \therefore y = (1\ 1\ 0\ 0)$

$y_{in_3} = x_3 + \sum y_j \, w_{i3} = 0 - 4 = -4 < 0, \therefore y_{in_3} = 0, \therefore y = (1\ 1\ 0\ 0)$

$y_{in_4} = x_4 + \sum y_j \, w_{i4} = 0 - 4 = -4 < 0, \therefore y_{in_4} = 0, \therefore y = (1\ 1\ 0\ 0)$

Test for convergence, true, so the input is near to the second sample.

1.6. Bidirectional Associative Memory Neural Network

Bidirectional Associative Memory (BAM) is a type of recurrent NN. BAM is hetero associative memory, meaning given a pattern, it can return another pattern which is potentially of different size.

The topology of BAM contains two layers of neurons x and y. Layer x and layer are fully connected. Once the weights have been established, input into layer x, present the pattern in layer y and vice versa.

BAM network uses vector pairs in binary format and it will convert these vector pairs to bipolar format (+1 and -1 only), so if the value is 0 it will be converted to -1.

Example: using BAM NN trying to train it to remember three binary vector pairs A_i and B_i then apply it on input vector A_1. (**Note: A_i is the original vector while B_i is the associative vector)**

$A_1 = (1\ 0\ 0)$ $B_1 = (0\ 0\ 1)$

$A_2 = (0\ 1\ 0)$ $B_2 = (0\ 1\ 0)$

$A_3 = (0\ 0\ 1)$ $B_3 = (1\ 0\ 0)$

Solution:

$A'_1 = (+1\ -1\ -1)$ $B'_1 = (-1\ -1\ +1)$

$A'_2 = (-1\ +1\ -1)$ $B'_2 = (-1\ +1\ -1)$

$A'_3 = (-1\ -1\ +1)$ $B'_3 = (+1\ -1\ -1)$

$$w = \sum A'^{T}_{i} B_i$$

$$w = A_i^T \begin{bmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{bmatrix} * B_i \begin{bmatrix} -1 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & +3 \\ -1 & +3 & -1 \\ +3 & -1 & -1 \end{bmatrix}$$

$$A_1(1\ 0\ 0) * w \begin{bmatrix} -1 & -1 & +3 \\ -1 & +3 & -1 \\ +3 & -1 & -1 \end{bmatrix} = (-1 - 1 + 3) \rightarrow (0\ 0\ 1) = B_1$$

$$B_1(0\ 0\ 1) * w^T \begin{bmatrix} -1 & -1 & +3 \\ -1 & +3 & -1 \\ +3 & -1 & -1 \end{bmatrix} = (3 - 1 - 1) \rightarrow (1\ 0\ 0) = A_1$$

1.7. Kohonen Neural Network

The objective of a Kohonen network is to map input vectors (patterns) of arbitrary dimension N onto a discrete map with 1 or 2 dimensions. Patterns close to one another in the input space should be close to one another in the map: they should be topologically ordered. A Kohonen network is composed of a grid of output units and N input units. The input pattern is fed to each output unit. The input lines to each output unit are weighted. These weights are initialized to small random numbers.

There is no hidden layer in a Kohonen NN. First, we will examine the input and output of a Kohonen NN. The input to a Kohonen NN is given to the NN using the input neurons. These input neurons are each given the floating-point numbers that make up the input pattern to the network. A Kohonen NN requires that these inputs be normalized to the range between 1 – and 1. Presenting an input pattern to the network will cause a reaction from the output neurons. The output of a Kohonen NN is very different from the output of a feed-forward NN. In a typical feed-forward network with five output neurons, the number of output values is consistent to be five. However, in the case of a Kohonen NN, only one of the five output neurons produces a value. Additionally, this single value is either true or false. When the pattern is presented to the Kohonen NN, one single output neuron is chosen as the output neuron. Therefore, the output from the Kohonen NN is usually the index of the neuron (i.e., Neuron #5) that fired. The structure of a typical Kohonen NN is shown in Fig. 1.8.

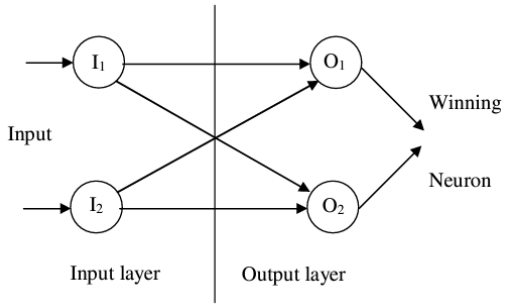

Fig. 1.8: Kohonen NN

The Kohonen NN algorithm is stated below:

Step 0: Initialize weights w_{ij}

Set topological neighbourhood parameters

Set Learning rate parameters

Step 1: While the stopping condition is false, do steps 2 – 8

Step 2: for each input vector x, do step 3 – 5

Step 3: for each j, compute the distance

$$D(j) = \sum_i (x_i - w_{ij})^2 \qquad \text{Euclidean distances}$$

Step 4: Find index J such that D(J) is a minimum

Step 5: For all units j within a specified neighbourhood of J, and all i:

$$W_{ij}(new) = W_{ij}(old) + \propto (x_i + W_{ij}(old))$$

Step6: Update learning rate

Step7: Reduce the radius of the topological neighbourhood at specified times

Step8: Test stopping condition

Example: A Kohonen NN to be cluster four vectors

$Vector_1$ (1 1 0 0)

Vector$_2$ (0 0 0 1)

Vector$_3$ (1 0 0 0)

Vector$_4$ (0 0 1 1)

The maximum no. of clusters to be formed is m = 2 with a learning rate $\propto$ = 0.6.

Solution:

With only 2 clusters available, the neighbourhood of node J is set so that only one cluster updates its weight at each step.

Initial weight matrix:

$$\begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}$$

For the first vector

$$\begin{matrix} (1 & 1 & 0 & 0) \\ x_1 & x_2 & x_3 & x_4 \end{matrix}$$

$D(i) = (1 - 0.2)^2 + (1 - 0.6)^2 + (0 - 0.5)^2 + (0 - 0.9)^2 = 1.86$

$D(2) = (1 - 0.8)^2 + (1 - 0.4)^2 + (0 - 0.7)^2 + (0 - 0.3)^2 = 0.98$ (minimum)

$\therefore$ J = 2 (the input vector is closed to output node 2)

$\therefore$ The weight on the winning unit is updated:

$$W_{21}(new) = W_{12}(old) + 0.6\ (x_i + W_{12}(old))$$

$W_{21}(new) = 0.8 + 0.6\ (1 - 0.8) = 0.92$

$W_{22}(new) = 0.4 + 0.6\ (1 - 0.4) = 0.76$

$W_{23}(new) = 0.7 + 0.6\ (0 - 0.7) = 0.28$

$W_{24}(new) = 0.3 + 0.6\ (0 - 0.3) = 0.12$

This gives the weight matrix:

$$\begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}$$

For the second vector

$$\begin{matrix} (0 & 0 & 0 & 1) \\ x_1 & x_2 & x_3 & x_4 \end{matrix}$$

$D(i) = (1 - 0.2)^2 + (1 - 0.6)^2 + (0 - 0.5)^2 + (1 - 0.9)^2 = 0.66$ (minimum)

$D(2) = (0 - 0.92)^2 + (0 - 0.76)^2 + (0 - 0.28)^2 + (1 - 0.12)^2 = 2.2768$

$\therefore J = 1$ (the input vector is closed to output node 1)

After updating the first column of the weight matrix:

$$\begin{bmatrix} 0.08 & 0.92 \\ 0.24 & 0.76 \\ 0.20 & 0.28 \\ 0.96 & 0.12 \end{bmatrix}$$

For the third vector

$$\begin{matrix} (1 & 0 & 0 & 0) \\ x_1 & x_2 & x_3 & x_4 \end{matrix}$$

$D(i) = (1 - 0.08)^2 + (0 - 0.24)^2 + (0 - 0.20)^2 + (1 - 0.96)^2 = 1.856$

$D(2) = (1 - 0.92)^2 + (0 - 0.76)^2 + (0 - 0.28)^2 + (1 - 0.12)^2 = 2.2768$ (minimum)

$\therefore J = 2$ (the input vector is closed to output node 2)

After updating the second column of the weight matrix:

$$\begin{bmatrix} 0.08 & 0.968 \\ 0.24 & 0.304 \\ 0.20 & 0.112 \\ 0.96 & 0.048 \end{bmatrix}$$

For the fourth vector

$$\begin{matrix}(0 & 0 & 1 & 1)\\ x_1 & x_2 & x_3 & x_4\end{matrix}$$

$D(i) = (1 - 0.08)^2 + (0 - 0.24)^2 + (1 - 0.20)^2 + (1 - 0.96)^2 = 0.7056$ (minimum)

$D(2) = (0 - 0.968)^2 + (0 - 0.304)^2 + (1 - 0.112)^2 + (1 - 0.048)^2 = 2.724$ (minimum)

$\therefore J = 1$ (the input vector is closed to output node 1)

After updating the first column of the weight matrix:

$$\begin{bmatrix}0.032 & 0.968\\ 0.096 & 0.304\\ 0.680 & 0.112\\ 0.984 & 0.048\end{bmatrix}$$

$\therefore$ Reduce the learning rate

$\propto (t + 1) \times \propto (t) = 0.5 \times 0.6 = 0.3$

After one iteration the weight matrix will be:

$$\begin{bmatrix}0.032 & 0.970\\ 0.096 & 0.300\\ 0.680 & 0.110\\ 0.984 & 0.048\end{bmatrix}$$

Chapter Two: Genetic Algorithm

2.1. Introduction

Genetic Algorithm (GA) is a type of Evolutionary Algorithms (EAs), a subset of machine learning, i.e., a search algorithm inspired by the Darwinian's theory of biological evolution. In the 1970s, Holland designed the GA as a way of exploiting the potential of the natural evolution to employ on computers. Natural evolution has observed the growth of complex organisms such as animals and plants from simpler single-celled life forms. Holland's GAs are models of the vitals of natural evolution and inheritance.

GA is a search algorithm based on the mechanics of natural selection and natural genetics. From a mechanistic view, GA is a variation in the generated and test method. Solutions are generated and sent to an evaluator. The evaluator reports whether the solution posed is optimal. In GAs, this generation and test process is repeated iteratively over a set of solutions. The evaluator returns information to guide the selection of new solutions for following iterations. In other words, we can say that GAs is problem solving methods requiring domain-specific knowledge that is often heuristic and they are a family of adaptive search procedure. GA derives their name from the fact that they are loosely based on models of genetic change in a population of individuals.

The genetic information that is operated on by a GA is contained in the chromosome. A chromosome contains an encoding of the variables of the problem being optimized, and is a finite-length string comprised of elements from a finite alphabet. A gene is a position in the chromosome string, and may take on values from the alphabet; the alphabet is analogous to the set of alleles in a natural system. The nature of the alphabet used to encode the chromosome strings depends on the particular problem being optimized. There are many types of encoding, some of these types illustrate in the following:

- **Binary,** this is the most common types of encoding. This type used for some applications of GA such as; logic design, graph theory, and so on.
- **Character,** this type of the encoding is used in some applications of GA such as; artificial intelligence, graph theory, scene recognition, and so on.
- **Real-valued,** this form is very useful in the mathematical algebra and the matters related with them, such as; weights of artificial neural networks, and so on.
- **Integer,** this type of encoding is used in some applications of GA such as; job shop scheduling, and so on.

The GA algorithm operating on fixed-length character strings can be summarized as follows:

a. Randomly create an initial population of individual fixed-length character string.

b. Iteratively perform the following sub-steps on the population of strings until the termination criterion has been satisfied:

i. Evaluate the fitness of each individual in the population.

ii. Create a new population of strings by applying at least the first two of the following three operations. The operations are applied to individual string(s) in the population chosen with a probability based on fitness.

1. Copy existing individual strings to the new population.

2. Create two new strings by genetically recombining randomly chosen substrings from two existing strings.

3. Create a new string from an existing string by randomly mutating the character at one position in the string.

c. The best individual string that appeared in any generation (i.e., the best-so-far individual) is designated as the result of the GA for the run. The result may represent a solution (or an approximate solution) to the problem.

2.2. Components of Algorithms

The key components of GAs are:

- **Genotype,** this is an encoding which describes the Phenotype.

- **Phenotype,** it is the objects that "live" and interact with the environment and is defined by GA.
- **Fitness Function,** a performance measure of the Phenotype induced by a Genotype. This function of the Genotype as well as environmental factors. The aim of the GA is to evolve a Phenotype that maximizes the fitness function. Using an appropriate definition of fitness, this translates into finding the best model structure and parameters for the given data.
- **Selection Criteria,** these are the criteria used to select individuals for reproduction and creation of new individual. It is common to select pairs of parents on the basis of fitness so that the probability of an individual being chosen for reproduction is proportional to its fitness value.
- **Breeding Operators,** the breeding operators take individuals chosen by selection criteria and use these as "parents" to create new offspring individuals.
- **Population Stabilization,** the offspring created by the breeding operators is added to the population, increasing its size. To keep the population size stable, some members must be added. This is carried out on the basis of fitness by choosing members at random from a distribution that favors those with lowest fitness values.

2.3. Selection Methods

Selection is the attribute of GA that determines which individuals will create will advance to the population of the next generation. The basic method of selection is to simply choose the highest fitness individuals formed from the children of one generation and uses them to form the next generation. Some of the selection methods are described below.

- **Steady State Selection,** this is a means of selection in which the population remains fairly consistent from generation to generation. In this type of selection, only a small percentage of the population is replaced by the children of the previous generation. In this way, the algorithm is sure to keep the "knowledge" of the previous generations, and incremental learning is emphasized. This is not in

itself a method of selection, but is used with other forms of selection to retain knowledge.

- **Elitist Selection,** this is a form of GA selection in which at least a few members of each generation carry on to the next generation. Again, at least a few members of the parent population must "survive" to the next generation. This is like steady state selection is not in itself a method of selection, but rather a tactic that is utilized along with different methods of selection to ensure some knowledge is transferred from one generation to the next.
- **Tournament Selection,** this is a means of selection in which two individuals are chosen randomly from the population and scored against one another according to fitness. The winner is allowed to participate in crossover and create offspring for the next generation. After this occurs, both individuals are put back into the population and could be selected again randomly at a later time. This method allows all individuals the chance for being selected as a parent (except the one with lowest score) and in this way tries to make the next generation somewhat diverse. The Tournament selection algorithm is illustrated in the following algorithm:

For each member of the population;
{
Select a random parent that hasn't already been used.
Select a second random parent that hasn't already been used.
Select the most fit of the pair, and mark it as used.
Add the individual to the array of the next population.
}

- **Roulette Selection,** the basic idea is to determine selection probability or survival probability for each chromosome proportional to the fitness value. Then a Roulette Wheel can be made displaying these probabilities. The selection process is based on spinning the wheel a number of times equal to population size, each time selecting a single chromosome for the new population. The Roulette Wheel selection algorithm is illustrated in the following algorithm:

```
Begin
    P=0;
    J=0;
    Rand=Random × sum of fitness;
    Repeat
        J=J+1;
        P=P + fitness of J;
    Until ((P>Rand) OR (J=Max. population));
    Select =J;
End
```

2.4. Crossover

Reproduction occurs in two ways:

- The parent is simply copied (possibly with random changes, i.e., mutation).
- We mean crossover.

In GA, evaluation from generation to generation is simulated both by preserving the genetic information contained in the chromosome strings of fit individuals and by altering this information by means of random genetic changes. Both of these goals are affected by genetic operators. The goal of preserving the genetic information of fit individuals is achieved through crossover. Crossover creates child individuals by crossing over portions of two parent individuals' chromosomes. One or both of the child individuals are retained in the new child population, and the child individuals are required to be unique with respect to the other children and to the parent population. The child population is unique but the crossover operator ensures that the genetic information of the parent population is preserved. Note that crossover point, that point where the crossover takes place is randomly determined. Some types of crossovers are illustrated below:

- **One-Point Crossover at Random Point (PX),** two parent individuals are selected at random (with selection biased towards choosing the fittest parents), and their chromosome strings are cut at a randomly determined point. In single point crossover either one or two offspring will be created. In order to create two offspring, it will be created by join the first part of one parent with the second part

of the other parent, and the second part of the one parent with the first part of the other parent. This process will produce two offspring shown in Fig. 2.

Fig. 2.1: One-point crossover generated two offspring

In order to create one offspring, this will be created by join the first part of one parent with the second part of the other parent as shown in Fig. 2.2.

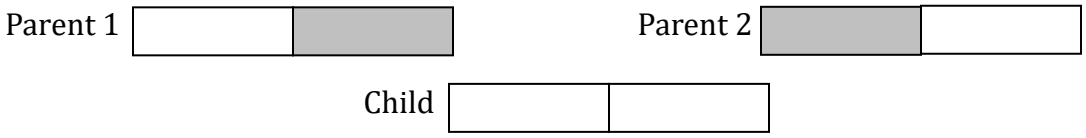

Fig. 2.2: One-point crossover generated one offspring

– **K-Points Crossover,** this is a multiple crossover position chosen at random and the parts of the two parents after the crossover positions are exchanged to form two offspring as shown in Fig. 2.3.

Fig. 2.3: K-points crossover

2.5. Mutation

The mutation operator introduces new information into the chromosome string of an individual by randomly altering one or more genes in that string. Mutation is used to prevent getting stuck in local maxima in a population. A randomly determined gene in the chromosome is changed to take on a new value from alphabet. The following example illustrates the mutation.

Chromosome before mutation	3 2 3 1 3 3 1 1 3 2 3 1
Chromosome after mutation	3 2 3 1 3 3 **2** 1 3 2 3 1

2.6. Other Operators

The goal of introducing change to the information in the chromosome string of individuals created by crossover is achieved with the mutation, addition, deletion and permutation operators. These operators are explained in the subsections bellow:

- **Addition,** the addition operator randomly adds a gene to the chromosome string. In the following example, a randomly determined gene from alphabet is added at random point in the chromosome. The randomly selected addition point is denoted by the symbol. In this example, addition causes the number of actual gene in the chromosome to increase from 12 to 13.

  Chromosome before addition 3 2 3 1 3 3 2 1 3 2 3 1

  Chromosome after addition 3 2 3 1 3 3 2 1 3 **3** 2 3 1

- **Deletion,** the deletion operator randomly deletes a gene from the chromosome string. In the following example, a randomly determined gene is removed from the chromosome. In this example, deletion causes the number of actual genes in the chromosome to decrease from 13 to 12.

  Chromosome before deletion 3 2 3 1 3 3 2 1 3 3 2 3 1

  Chromosome after deletion 3 2 3 1 3 2 1 3 3 2 3 1

2.7. Parameters of Genetic Algorithm

If we use GA, we must set parameters that control the behaviour of GA. These parameters include:

- The population size of GA.
- The number of generations of the GA around the main loop.
- The initial number of runs of the GA.
- The probabilities of crossover rate (P_c) and mutation rate (P_m).
- The length of the chromosome.

The value of population size must not exceed 100 are illustrated in many researches. The number of generation values must be reasonably large to guarantee the solution. The length of the chromosome depends on the problem encoding. The most used values of crossover probability are from 0.3 to 0.95 and the most used values of mutation probability are from 0.001 to 0.1, these values depend largely on the application. The value of the initial numbers of runs of the GA depends on the problem we try to find the solution for.

2.8. Genetic Algorithm Termination Criteria

Each run of the conventional Genetic Algorithm requires specification of a termination criterion for deciding when to terminate a run and a method of result designation. Among the frequently used stopping criteria are:

- Stopping after a preset number of generations.
- Computing time is limit.
- Stopping when the fitness of the fittest member of the population is within the user specified rang.
- Stopping when the population diversity drops below a certain threshold.
- When all individuals in a generation are identical, which means the evaluation functions for all individuals are same.
- GA can also be halted if the solution quality of the population does not improve by more than a specified amount in a specified number of successive generations.

2.9. Genetic Programming and Applications

Genetic Programming (GP) is an intelligence technique whereby computer programs are encoded as a set of genes which are evolved utilizing a GA. In other words, the GP employs novel optimization techniques to modify computer programs; imitating the way humans develop programs by progressively re-writing them for solving problems automatically. Trial programs are frequently altered in the search for obtaining superior solutions due to the base is GA. These are evolutionary search techniques

inspired by biological evolution such as mutation, reproduction, natural selection, recombination, and survival of the fittest. The power of GAs is being represented by an advancing range of applications; vector processing, quantum computing, VLSI circuit layout, and so on. But one of the most significant uses of GAs is the automatic generation of programs. Technically, the GP solves problems automatically without having to tell the computer specifically how to process it. To meet this requirement, the GP utilizes GAs to a "population" of trial programs, traditionally encoded in memory as tree-structures. Trial programs are estimated using a "fitness function" and the suited solutions picked for re-evaluation and modification such that this sequence is replicated until a "correct" program is generated. GP has represented its power by modifying a simple program for categorizing news stories, executing optical character recognition, medical signal filters, and for target identification, etc. This paper reviews existing literature regarding the GPs and their applications in different scientific fields and aims to provide an easy understanding of various types of GPs for beginners.

GP is a type of Evolutionary Algorithms (EAs), a subset of machine learning, i.e., a search algorithm inspired by the Darwinian's theory of biological evolution. For the first time, the GP was introduced by Mr. John Koza which enables computers to solve problems without being clearly programmed. The GP functions based on John Holland's GAs to generate programs for solving various complex optimization and search problems automatically.

During the past forty years, the GPs have been applied to solve a wide range of complex optimization problems, patentable new inventions, producing a number of human-competitive results, etc. in the emerging scientific fields. Like many other fields of computer science, GP still is developing briskly, with new ideas and applications being continuously advanced.

Some of GP's applications are:

– Image and Signal Processing.

- Artistic.
- Medicine, Biology and Bioinformatics.
- Industrial Process Control.
- Entertainment and Computer Games.

Chapter Three: Fuzzy Logic

3.1. Introduction

Fuzzy Logic (FL) is a method of reasoning that resembles human reasoning. The approach of FL imitates the way of decision-making in humans that involves all intermediate possibilities between digital values YES and NO.

The conventional logic block that a computer can understand takes precise input and produces a definite output as TRUE or FALSE, which is equivalent to a human's YES or NO.

The inventor of fuzzy logic, Lotfi Zadeh, observed that, unlike computers, human decision-making includes a range of possibilities between YES and NO, such as:

The fuzzy logic works on the levels of possibilities of input to achieve the definite output.

A proposition can be true on one occasion and false on another. If a proposition is true, it has a truth value of 1; if it is false, its truth value is 0. These are the only possible truth values. Propositions can be combined to generate other propositions, using logical operations.

When you say it will rain today or that you will have an outdoor picnic today, you are making statements with certainty. Of course, your statements in this case can be either true or false. The truth values of your statements can be only 1, or 0. Your statements then can be said to be crisp.

On the other hand, there are statements you cannot make with such certainty. You may be saying that you think it will rain today. If pressed further, you may be able to say with a degree of certainty in your statement that it will rain today. Your level of certainty, however, is about 0.8, rather than 1. This type of situation is what fuzzy logic was developed to model.

Fuzzy logic deals with propositions that can be true to a certain degree somewhere from 0 to 1. Therefore, a proposition's truth value indicates the degree of certainty about which the proposition is true. The degree of certainty sounds like a probability (perhaps subjective probability), but it is not quite the same. Probabilities for mutually exclusive events cannot add up to more than 1, but their fuzzy values may. Suppose that the probability of a cup of coffee being hot is 0.8 and the probability of the cup of coffee being cold is 0.2. These probabilities must add up to 1.0. Fuzzy values do not need to add up to 1.0. The truth value of a proposition that a cup of coffee is hot is 0.8. The truth value of a proposition that the cup of coffee is cold can be 0.5. There is no restriction on what these truth values must add up to.

Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth- truth values between "completely true" and "completely false". As its name suggests, it is the logic underlying modes of reasoning which are approximate rather than exact. The importance of fuzzy logic derives from the fact that most modes of human reasoning especially common-sense reasoning are approximate.

3.2. Fuzzy Sets

There is a strong relationship between Boolean logic and the concept of a subset. Similarly, there is a strong relationship between fuzzy logic and fuzzy subset theory.

A subset U of a set S can be defined as a set of ordered pairs, each with a first element that is

an element of the set S, and a second element that is an element of the set {0, 1}, with exactly one ordered pair present for each element of S. This defines a mapping between elements of S and elements of the set {0, 1}. The value zero is used to represent non-membership, and the value one is used to represent membership. The truth or falsity of the statement: x is in U is determined by finding the ordered pair whose first element is x. The statement is true if the second element of the ordered pair is 1, and the statement is false if it is 0.

Similarly, a fuzzy subset F of a set S can be defined as a set of ordered pairs, each with a first element that is an element of the set S, and a second element that is a value in the interval [0, 1], with exactly one ordered pair present for each element of S. This defines a mapping between elements of the set S and values in the interval [0, 1]. The value zero is used to represent complete non-membership, the value one is used to represent complete membership, and values in between are used to represent intermediate degrees of membership. The set S is referred to as the universe of discourse for the fuzzy subset F. Frequently, the mapping is described as a function, the membership function of F. The degree to which the statement: x is in F is true is determined by finding the ordered pair whose first element is x. The degree of truth of the statement is the second element of the ordered pair.

This can be illustrated with an example. Let's talk about people and "youngness". In this case, the set S (the universe of discourse) is the set of people. A fuzzy subset YOUNG is also defined, which answers the question "To what degree is person x young?" To each person in the universe of discourse, we have to assign a degree of membership in the fuzzy subset YOUNG. The easiest way to do this is with a membership function based on the person's age.

Young (x) = {1, if age(x) ≤ 20, (30-age(x))/10, if 20 < age(x) ≤ 30, 0, if age(x) > 30}

Given this definition, here are some example values:

Person	Age	Degree of Youth
Mohamed	10	1.00
Ahmed	21	0.90
Mustafa	25	0.50
Mahmud	26	0.40
Taha	83	0.00

So given this definition, we would say that the degree of truth of the statement “Mustafa is YOUNG" is 0.50.

3.3. Logical Operators: Fuzzy Intersection, Fuzzy Union, Fuzzy Complement

– **Intersection:** The intersection of two fuzzy sets A and B is a fuzzy set such that:

$M_A \cap M_B(x) = \min [M_A(x), M_B(x)]$ for every $x \in X$, and it corresponds to AND

– **Union:** The union of two fuzzy sets A and B is a fuzzy set $A \cup B$ such that:

$M_A \cup M_B(x) = \max [M_A(x), M_B(x)]$ for every $x \in X$, and it is corresponding to OR

– **Complement** the membership grades range in closed intervals [0, 1]. The complement of a fuzzy set, concerning the universal set, is denoted by:

$M_{\bar{A}}(x) = 1 - M_A(x)$ for every $x \in X$

Example: To illustrate the previous concepts listed below (which is about temperature):

Temperature	Cold	Warm	Hot	Very Hot
-10	1	0	0	0
0	1	0	0	0
10	0.50	0.20	0	0
20	0.10	0.65	0	0
30	0	0.95	0.10	0
40	0	1	0.25	0.10
50	0	1	0.75	0.50
60	0	1	1	0.95
70	0	1	1	1
80	0	1	1	1

Union

Cold ∪ Hot = [1//-10 + 1//0 + 0.50//10 + 0.10//20 + 0.10//30 + 0.25//40 + 0.75//50 + 1//60 + 1//70 + 1//80]

Warm ∪ Hot = [0//-10 + 0//0 + 0.20//10 + 0.65//20 + 0.95//30 + 1//40 + 1//50 + 1//60 + 1//70 + 1//80]

Intersection

Cold ∩ Warm = [0//-10 + 0//0 + 0.20//10 + 0.10//20 + 0//30 + 0//40 + 0//50 + 0//60 + 0//70 + 0//80]

Complement

Not Cold = [0//-10 + 0//0 + 0.50//10 + 0.9//20 + 1//30 + 1//40 + 1//50 + 1//60 + 1//70 + 1//80]

Not Warm = [1//-10 + 1//0 + 0.80//10 + 0.35//20 + 0.05//30 + 0//40 + 0//50 + 0//60 + 0//70 + 0//80]

3.4. Compositional Rule of Inference

Compositional Rule of Inference (CRI) is the first and principal approximate reasoning approach. It allows inferring a result from a function which contains two operators: a t-norm and an implication. Triangular norm (t-norm for short) is a binary function which is used as a conjunction function in fuzzy systems. Fuzzy implication is a function that allows evaluating a rule from the values of its premise and conclusion.

According to some authors, some combinations of t-norms and implications do not always fit human intuitions. For that and to ensure an appropriate inference result, it is necessary to study all the possible combinations and their compatibility. This will establish a general guide for fuzzy inference systems developers when choosing its parameters.

CRI was proposed by Zadeh, to give approximate reasoning and to determine new deductions in a fuzzy inference system.

To extend the fuzzy sets considered in the CRI, linguistic modifiers can be used. A linguistic modifier is a tool to give a new characterization to a fuzzy set and to modify its meaning which is not far from the original.

3.5. Fuzzification and Defuzzification

3.5.1. Fuzzification

Fuzzification is the process of making a crisp quantity fuzzy. We would do this by simply recognizing that many of the quantities that we consider to be crisp and deterministic are not

deterministic at all: they carry considerable uncertainty. If the form of uncertainty happens to arise because of imprecision, ambiguity, or vagueness, then the variable is probably fuzzy and can be represented by a membership function. In the real-world hardware (digital voltmeter) generates crisp data, but this data is subject to experimental error. The information shown in Fig. 3.1 shows one possible range of errors for a typical voltage reading and the associated membership function which might represent such imprecision. The representation of imprecise data as fuzzy sets is a useful, but not mandatory step when that data is used in fuzzy systems. This idea is shown in Fig. 3.1, where we consider the data as a crisp reading, Fig. 3.1a, or when we consider the data as a fuzzy reading, as shown in Fig. 3.1b. In Fig. 3.1a we might want to compare how a crisp voltage reading might compare to a fuzzy set, say "low voltage". In the figure we see that the crisp reading intersects the fuzzy set "low voltage" at a membership of 0.3; i.e., the fuzzy set and the reading can be said to agree at a membership value of 0.3.

In Fig. 3.1b the intersection of the fuzzy set "medium voltage" and a fuzzified voltage reading occurs at a membership of 0.4. We can see in Fig. 3.1b that the set intersection of the two fuzzy sets is a small triangle, whose largest membership occurs at the membership value of 0.4.

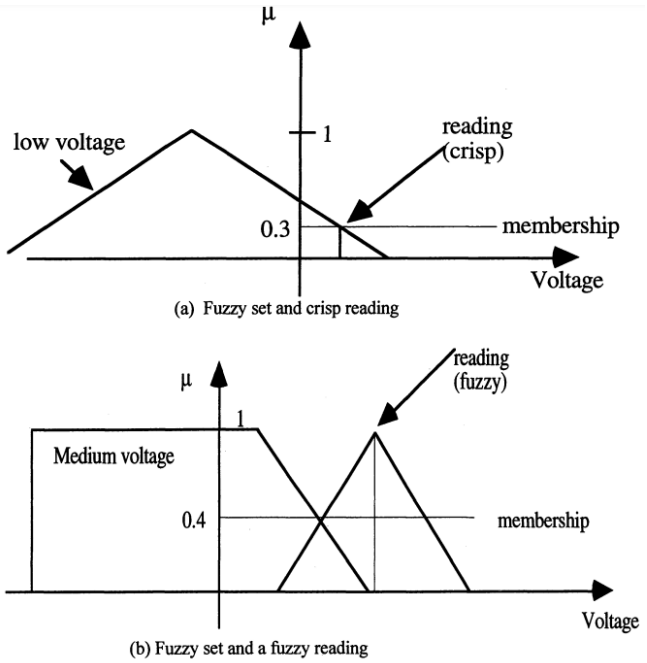

Fig. 3.1: Two comparisons of fuzzy sets and (a) crisp or (b) fuzzy readings

3.5.2. Defuzzification

Even though the bulk of the information we assimilate every day is fuzzy, most of the actions or decisions implemented by humans or machines are crisp or binary. The decisions we make are binary, the hardware we use is binary, and certainly, the computers we use are based on binary digital instructions. For example, in deciding to develop a new biomedical product the eventual decision is to go forward with development or not; the fuzzy choice to "partially go forward" might be acceptable in the planning stages, but eventually, funds are released for development or they are not released. In giving instructions to a digital thermometer, it is not possible to increase the temperature "slightly"; a machine does not understand the natural language of a human. We have to increase the temperature by 3.4 degrees, for example, a crisp number. An electrical circuit typically is either on or off, not partially on.

Section 3.5.1 of this chapter illustrates procedures to "fuzzify" the mathematical and physical principles we have so long considered to be deterministic. But, as mentioned, there will be a need in various applications and medical scenarios to "denazify" the fuzzy results we generate through a fuzzy set analysis. In other words, we may eventually find a need to convert the fuzzy results to crisp results. For example, in classification we may want to transform a fuzzy partition matrix into a crisp partition; in pattern recognition, we may want to compare a fuzzy pattern to a crisp pattern; in blood pressure control during anaesthesia, we may want to give a single-valued input to an intravenous drug device instead of a fuzzy input command. This process of "defuzzification" has the result of reducing a fuzzy set to a crisp single-valued quantity, or a crisp set, converting a fuzzy matrix to a crisp matrix, or making a fuzzy number crisp.

Mathematically, the defuzzification of a fuzzy set is the process of "rounding it off" from its location in the unit hypercube to the nearest (in a geometric sense) vertex.