SATHYABAMA UNIVERSITY

(Established under Section 3, UGC Act 1956)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SCSX1029 SOFT COMPUTING

SCSX1029 SOFT COMPUTING

UNIT 1

Neural Networks

Introduction to ANS- adaline- BPN- Hopfield network- Boltzman machine- Self Organizing maps

1.1 Introduction to ANS

Neural Networks

Neural networks are highly interconnected processing elements or neurons**. Adaptive Neural Networks (ANNs)** are inspired by biological neurons. ANNs are used to estimate and approximate the functions that are the outcome of vast and unknown inputs.

Advantages of ANNs:

- Performs tasks that the linear programs could not perform
- The network functions smoothly even in case of any element failure
- There is no need of reprogramming

Drawbacks of ANNS

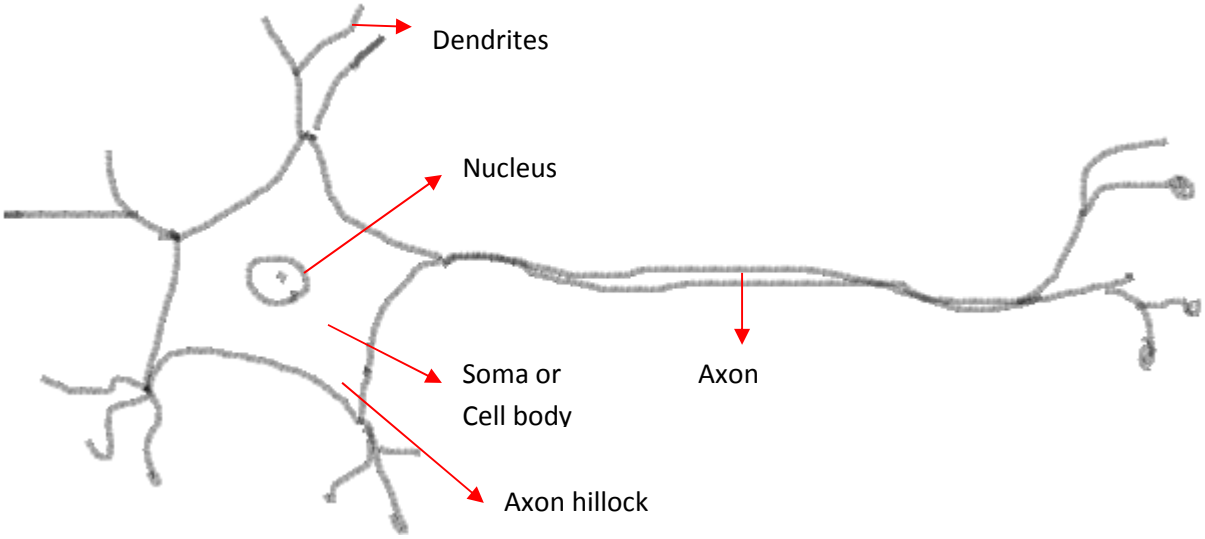
- Need efficient training
- Huge processing time for large Neural Networks

Applications of Neural Networks

1. Signal Processing: In telephone networks ANNs are used to process the signals thereby suppressing the noise
2. Pattern Recognition: Handwritten characters, Radar signal classification and analysis, Speech recognition, Finger print recognition, character recognition, Handwriting analysis
3. Medicine: ECG signal analysis and understanding, Diagnosis of diseases, Medical Image Processing
4. Image Processing: Image matching, pre-processing, image compression
5. Military Systems: Sea mine detection, Radar cluster classification
6. Power Systems: State estimation, Transient deduction, Fault deduction

Biological Neural Networks:

Human brain has ten billion interconnected neurons. Each neuron or nerve cell uses biological reactions to receive, process and transmit information



Structure of a Biological Neuron

Components of a biological neuron

The major components are Dendrites, Soma, Axon.

Dendrites: Receive signals from other neurons. Signals are electric impulses transmitted across synaptic gap using chemical reactions.

Soma: Contains nucleus and pericardium. It sums incoming signals when sufficient input is received when the cell fires.

Axon: Neuron sends spikes of electrical activity through long thin stand called axon which is split into branches.

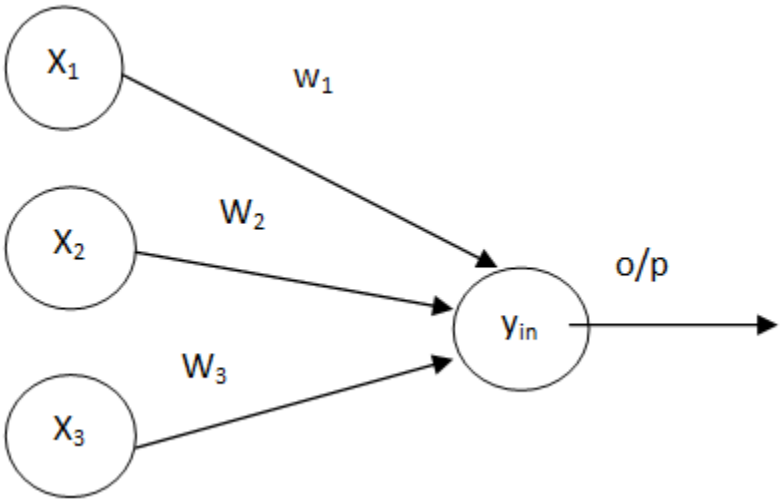
Artificial Neural Networks (ANNs)

- ANN is information processing system, with common characteristics of biological neurons.
- Basic signal processing elements of neural networks are neurons
- Signals passed between neurons over the connection links
- Each connected link has an associated weight
- Each neuron applies activation function to its net input to determine its output signal

Characters of ANN

- Architecture
- Learning algorithm
- Activation function

Representation of Artificial Neuron



$$Y_{in} = w_1x_1 + w_2x_2 + w_3x_3$$

Activation function of neuron y is represented as $y = f(Y_{in})$

Comparison between artificial neurons and biological neurons

Let PE be the processing elements. Properties of PEs of ANN with biological neurons

- PE receives many signals
- Signals are modified by weight at receiving synapse
- PE sums weighted inputs
- Output from a particular neuron may go to many other neurons
- Biological neurons use synaptic strength as weight to inhibit or excite adjacent neuron. ANN use random numbers as weight.
- Biological neurons undergo electro chemical reactions where as ANNs utilize mathematical learning equations to introduce learning
- Biological neurons are fast in nature (learn and adapt) but the ANNs are slow when compared to biological neurons.

ANN functioning

ANN functioning corresponds to the arrangement of neurons into layers and connected patterns between each layer.

Types:

1. Feed Forward networks(Non recurrent)
   a. Single Layer net
   b. Multi Layer net
2. Feedback or Recurrent networks

Non recurrent networks:

Single layer Representation: The single layer representation comprises of only the input layer and the output layer. No hidden Layers are included

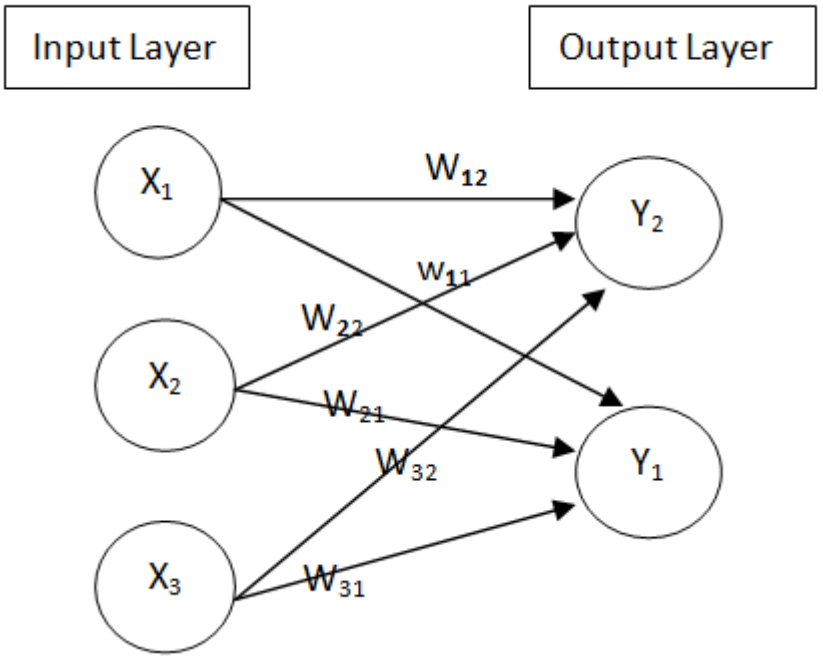
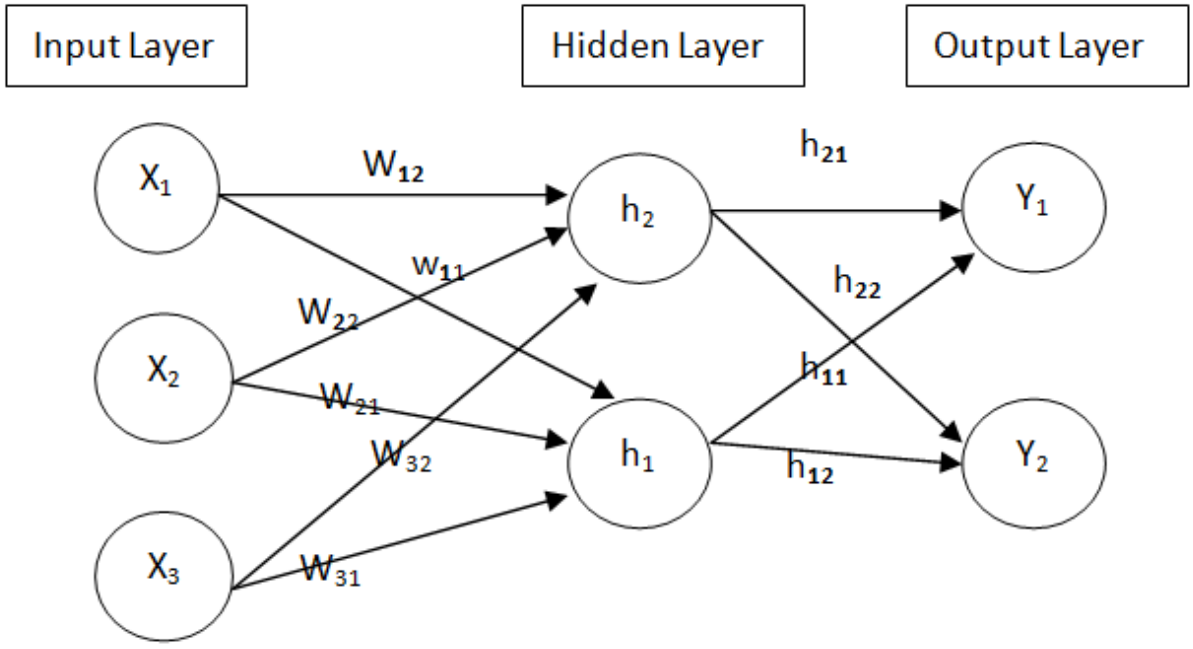
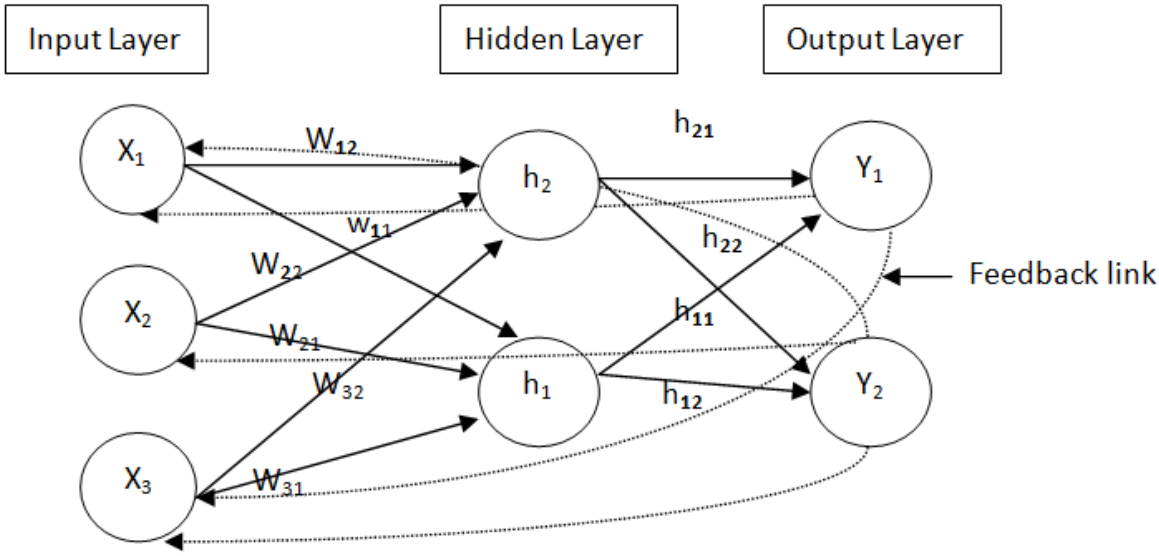


Fig. Single Layer Representation

Multilayer network representation: The multilayer representation consists of Input layer, one or more hidden layers and output layer.



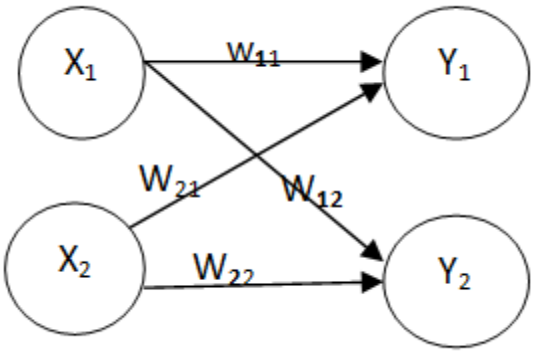
Feedback Networks or Recurrent Networks or Counter Propagation Networks



Activation Functions (AF)

AFs are used to calculate output response of neuron.

For example, let us consider the following ANN

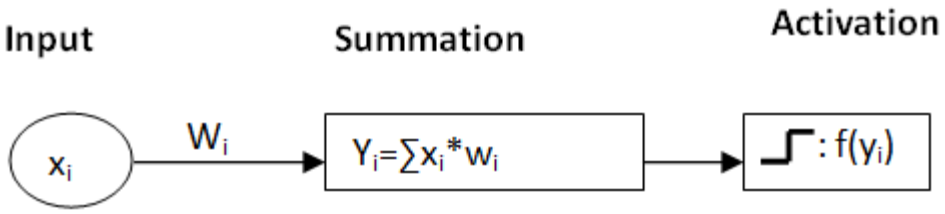


Inputs: x_1, x_2

Weights: $w_{11}, w_{12}, w_{21}, w_{22}$

Output: y_1, y_2

AF for the above example is as follows:



AF Types:

1. **Linear Function / Identity function**

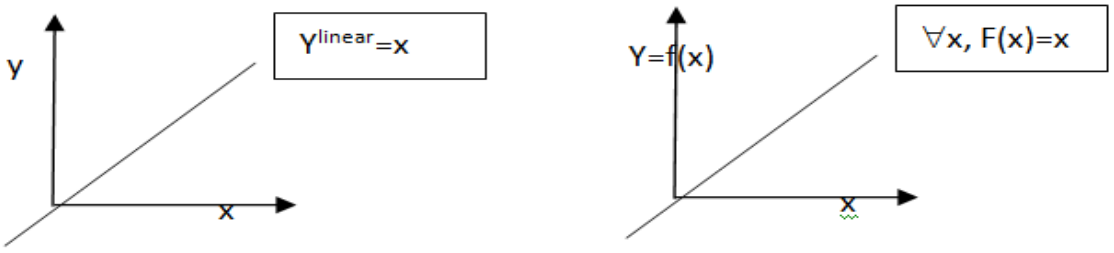
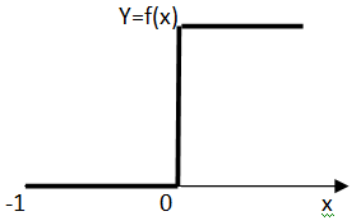


Fig. Y in terms of x: y=f(x)

2. Step function

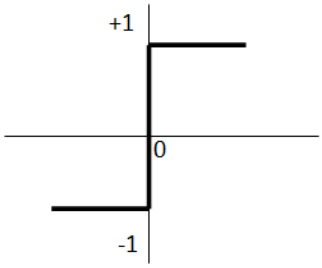
Binary (1 or 0)



F(x) = 1 if x≥0

0 if x<0

Bipolar(+1 or -1)

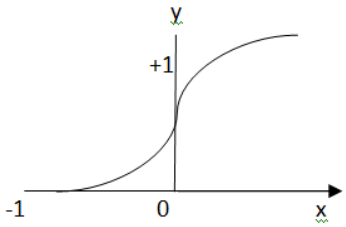


y^{sign}=F(x) = +1 if x≥0,

-1 if x<0

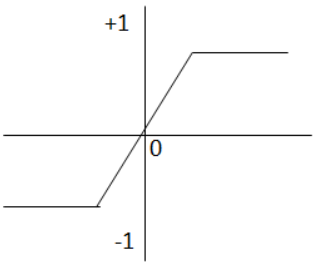
3. Sigmoidal function

Binary (1 or 0)



$Y=\frac{1}{1+e^{-x}}$ =F(x) = 1 if x≥0

Bipolar(+1 or -1)



$y^{sigmoid}$=F(x) = 1 if x≥0,

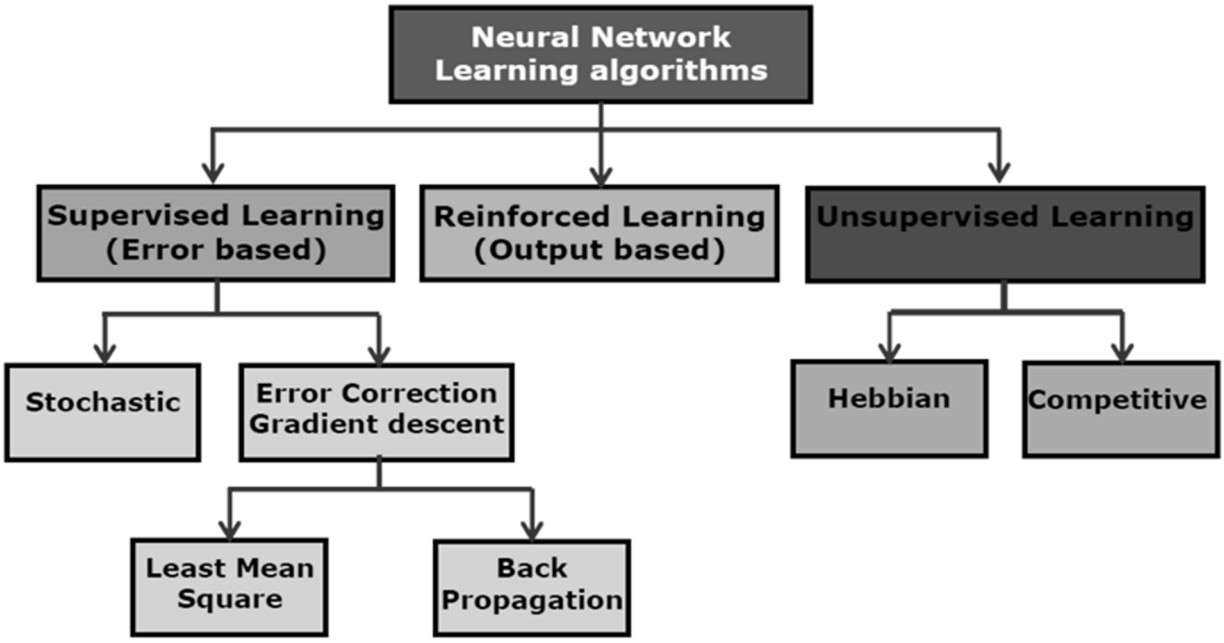
Types of outputs for AFs

1. Binary output(1 or 0)
2. Bipolar output(+1 or -1)

Learning or training

Types : 1. Supervised 2. Unsupervised 3. Fixed weight networks

Classification of Learning Algorithms



1. **Supervised Learning:**
   a. **Hebb rule**: $\Delta \boldsymbol{w_i} = \boldsymbol{\eta x_i y}$
   Hebb rule is determines the change in the i[th] synaptic weight of a node 'I' (Δw_i).
   η: Learning rate
   x_i: Input
   Y: post synaptic response and $y = \sum_j w_j x_j$ for each node j.

   b. Delta rule:
   Delta rule is the gradient descent learning rule for updating input weights in a single layer ANN.

$$\Delta \boldsymbol{w_{ji}} = \boldsymbol{\eta x_i (d_j - y_j)}$$

   η: Learning rate
   x_i: Input
   d_j: Desired output or target output
   y_j: actual output

$$y_j = \sum_i w_{ji} x_i$$

   c. Back Propagation Network(BPN)
   d. Perceptron Learning rule

The perceptron learning rule was developed by Frank Rosenblatt in the late 1950s. Training patterns are presented to the network's inputs; the output is computed. Then the connection weights w_j are modified by an amount that is proportional to the product of the difference between the actual output y, and the desired output d, and the input pattern, x.

$$\boldsymbol{w_j(t+1) = w_j(t) + \eta x_i (d_j - y_j)}$$

η: Learning rate
x_i: Input
d_j: Desired output or target output
y_j: actual output
t: iteration number

Note: In supervised learning the error information is used to improve the network behaviour

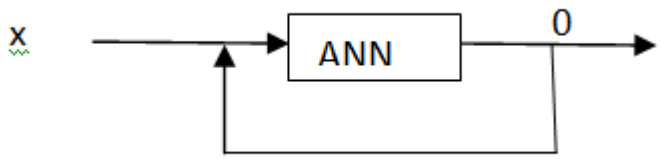
Associative memory : represents the Neural Network Association between the input vector and the output vector

Auto Associative memory: If the desired output vector is same as the input vector, it is called auto associative memory

Hetero Associative memory: If the output target vector is different from input vector

2. Unsupervised Learning

Error information is not used to improve the network behaviour. The network is self organizing.



Examples for unsupervised learning:

- Kohenon Self Organizing map
- ART (Adaptive Resonance theory)

3. Fixed weight neurons

The weights of the neurons are not changed throughout the network.

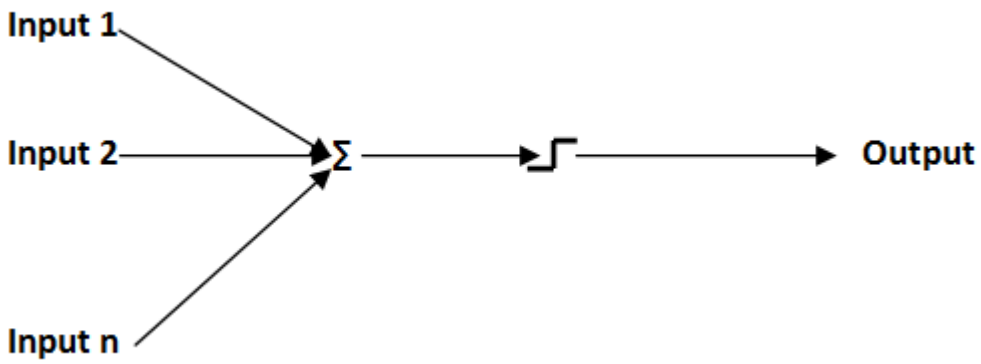
Examples:

- Hopfield net
- Max net

Neuron Modelling

McCulloch Pitts Neuron model

- It is an earliest Artificial Neuron model
- Developed by Warren Mc Culloch and Walter Pitts in 1943
- It is also called **Linear Thresholod gate**



Components:

Set of Inputs x_i

Set of weights wt_i

Threshold , u

Activation function, f

Single neuron output, y

Architecture

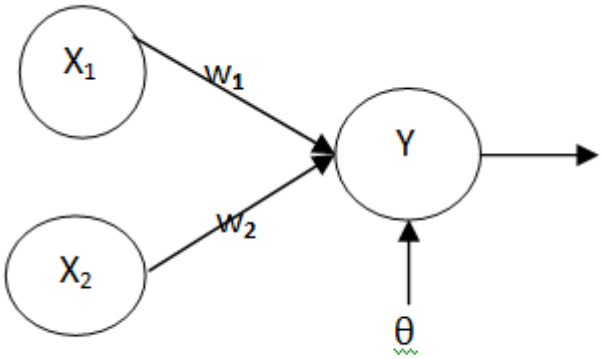
Y_{in}: Input Signal

Θ : Threshold

Activation Function:

$$f(Y_{in}) = 1 \; if \; Y_{in} \geq \theta$$

$$0 \; if \; Y_{in} < \theta$$



Example 1: McCulloch pitts neuron for AND function

X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

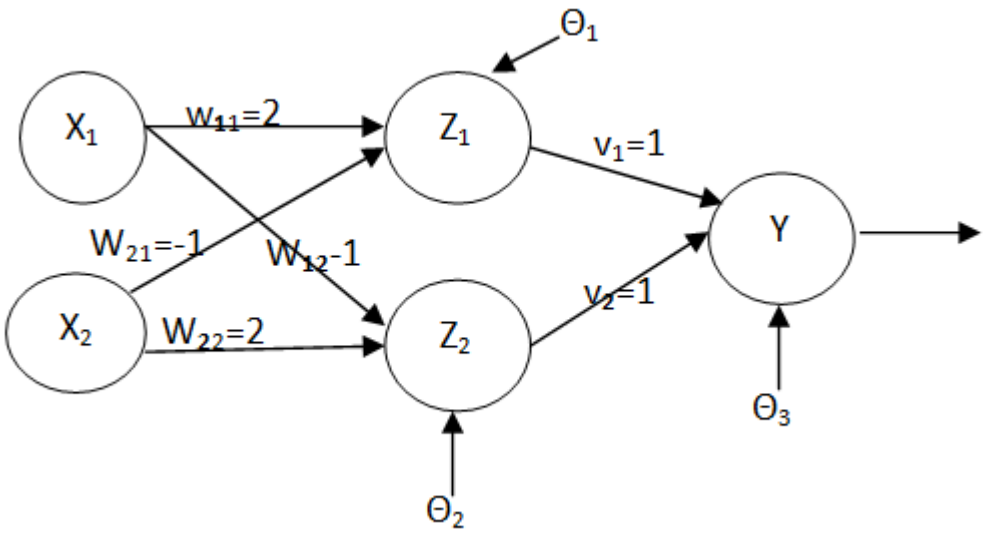
Example 2: McCulloch pitts neuron for OR function

X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

Example 3: McCulloch pitts neuron for XOR function

X1	X2	Y
1	1	0
1	0	1
0	1	1
0	0	0

Example 4: McCulloch pitts neuron to perform XOR with the following neural model:



$Z_{1net} = x_1 w_{11} + x_2 w_{21}$ and $Z_{2net} = x_1 w_{12} + x_2 w_{22}$

$$Z_1 = 1 \; if \; Z_{1net} \geq 2$$

$$0 \; if \; Z_{1net} < 2$$

$$Z_2 = 1 \; if \; Z_{2net} \geq 2$$

$$0 \; if \; Z_{2net} < 2$$

$Y_{net} = Z_1 V_1 + Z_2 V_2$

$$Y = 1 \; if \; Y_{net} \geq 2$$

$$0 \; if \; Y_{net} < 2$$

X_1	X_2	T	W_{11}	W_{12}	W_{21}	W_{22}	V_1	V_2	Θ_1	Θ_2	Θ_3	Y_{net}	Y	Z_{net1}	Z_{net2}	Z_1	Z_2
1	1	0	2	-1	-1	2	1	1	2	2	1	0	0	1	1	0	0
1	0	1	2	-1	-1	2	1	1	2	2	1	1	1	2	-1	1	0
0	1	1	2	-1	-1	2	1	1	2	2	1	1	1	-1	2	0	1
0	0	0	2	-1	-1	2	1	1	2	2	1	0	0	0	0	0	0

Simple Neural network for pattern classification

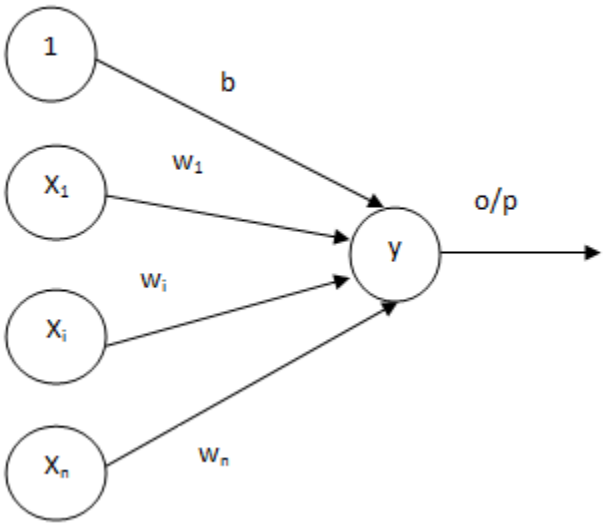


Fig. Single Layer for pattern classification

Let 'b' be the bias value which is always 1. For Bipolar input, the output function y is as follows:

$$y = f_{net} = \begin{matrix} 1 \; if \; net \; \geq 0 \\ -1 \; if \; net \; < 0 \end{matrix}$$

For Binary input, the output function y is as follows:

$$y = \begin{matrix} 1 \; if \; y_{net} \geq 0 \\ 0 \; if \; y_{net} < 0 \end{matrix}$$

$$net = b + \sum_i w_i x_i$$

$$b + \; x_1 w_1 + x_2 w_2 = 0$$

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}$$

$$x_1 w_1 + x_2 w_2 = 0$$

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{0}{w_2}$$

1.2 Adaline: Adaptive Linear Neuron

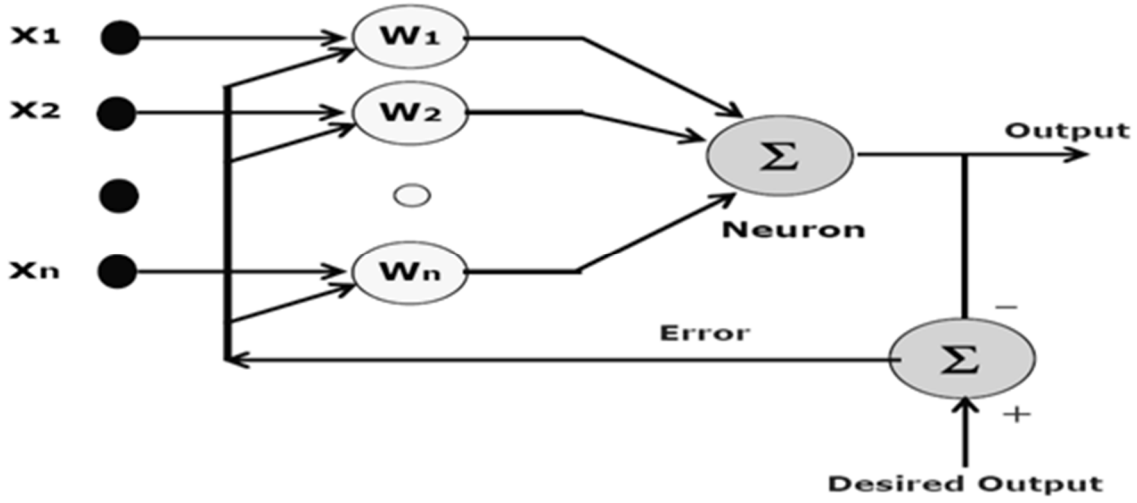
Rule: Difference between Actual output and desired output is the background for error correction

Learning: Changing of weights in ANN. The value of correction is proportional to signal at the elements input.

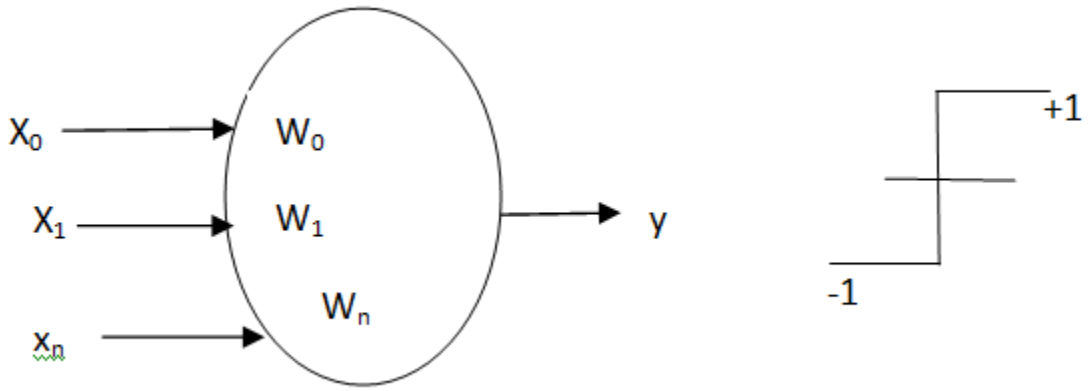
- Has single neuron of **Mc culloch pitts model**
- Weights are determined by **LMS** (Least Mean Square Error) learning rule
- LMS rule is otherwise called **Delta rule**
- It is well established supervised learning method

Structure of Adaline:

Basic structure follows simple neuron with linear activation function and a feedback loop.



ALC: Adaptive Linear Combiner



$$y = +1 \; if \; ALC \; output \; is + ve$$

$$-1 \; if \; ALC \; output \; is - ve$$

$$y = w_0 + \sum_{j=1}^{n} w_j x_j$$

Where $w_0 \; is \; the \; bias \; weight$

If $x_0 = 1$ then

$y = \sum_{j=0}^{n} w_j x_j$

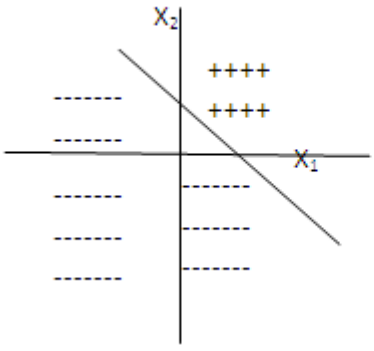
$$y = w^t x$$

Adaline Training Methodology

- Structure resembles a simple neuron with and extra feedback loop
- During training the input vector x_i and the Desired output D is presented to the network
- Weights are adjusted based on Delta rule
- Inputs with fixed weight produces scalar output after training
- The network performs n dimensional mapping to a scalar value
- Activation function not used during training phase
- Training and Generalization are two important aspects of Adaline network

Applications of Adaline

- Making binary decisions
- Realizations of AND, OR and NOT gates
- Only linear separable functions are recognized.
- **Linear Separability**: The idea behind hidden Layers
- two sets are linearly separable if there exists at least one line in the plane with all of the positive values on one side of the line and all the negative values on the other side.

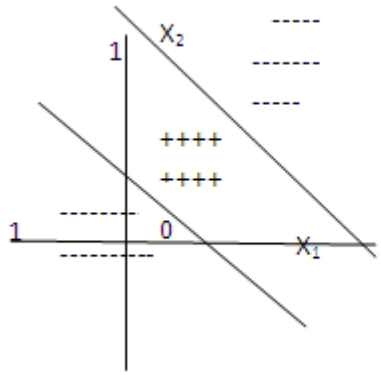


Truth table for AND

X1	X2	y
1	1	1
1	0	0
0	1	0
0	0	0

- **Fig. Linear Separability**

- But XOR function cannot be separated using a single line. Two lines are needed to segregate positive and negative values. Hence it is not supported by adaline.



Truth table for XOR

X1	X2	y
1	1	0
1	0	1
0	1	1
0	0	0

-

- **Fig. XOR is not linear separable**

LMS Learning Rule:

To train Adaline to perform a given processing function.

Let x be the input vector

w: weights

y: output values

d_k: Desired or correct output value

Manual Calculation of w*(weight adjustment)

Given the set of input, desired output pairs {(x1,d1), (x2,d2)}, the best value of w* needs to be calculated.

Let yk be the actual output for k^{th} input vector.

$Error\ term\ \varepsilon_k = d_k - y_k$ (Equation 1)

Mean squared error $\boldsymbol{\varepsilon_k}^2$ is defined as follows:

$\varepsilon_k{}^2 = \frac{1}{L}\sum_{k=1}^{L} \varepsilon_k^2$ (Eqn 2) where L is the number of input vectors in training set

$$y^k = w^t x_k$$

Substituting equations 3, 2 in 1

$$\varepsilon_k^2 = (d_k - w^t x_k)^2$$

1.3 Back Propagation Network

-Supervised Learning

- Feed forward network

- Multilayer Perceptron

BPN Rule: Adjusting the weights in previous level of layers to reduce error. This leads to Delta learning rule

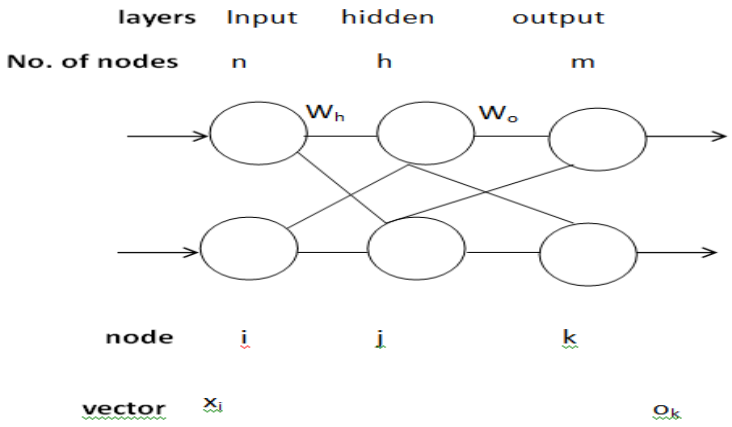


Fig. BPN topology

BPN algorithm

Read 'n' number of input nodes

Read 'h' number of hidden nodes

Read 'm' number of output nodes

Step 1: Read the input vector x_i

For i=1 to n

Read x[i]

Step 2: Read the output vector o_k (Desired output)

For k=1 to m

Read o[k]

Step 3: Read the input hidden weights wh_{ij}

For i=1 to n

For j=1 to h

Read wh[i][j]

Step 4: Read the output hidden weights wo_{jk}

For j=1 to h

For k=1 to m

Read wo[j][k]

Step 5: Calculate $neth_j$ (net value in hidden layer)

For j= 1 to h

$$neth[j] = \sum_{i=1}^{n} x[i] * wh[i][j]$$

Step 6: Calculate the f(net) : 'oh_j' in hidden layer (sigmoidal function)

For j= 1 to h

$$oh[j] = \frac{1}{(1 + e^{-neth[j]})}$$

Step 7: Calculate the net output: "$neto_k$"

For j=1 to h

For k=1 to m

$$neto[k] += oh[j] * wo[j][k]$$

Step 8: Calculate oo_k (actual output)

For k=1 to m

$$oo[k] = \frac{1}{(1 + e^{-neto[k]})}$$

Step 9: Calculate error in output layer 'eo_k' (Desired output- Actual output)

For k=1 to m

$$eo[k] = o[k] - oo[k]$$

Step 10: Calculate error in hidden layer 'eh_j'

For j=1 to h

For k=1 to m

$$eh[j] += eo[k] * wh[j][k]$$

Step 11: Calculate the new weights(nwo_{jk}) for the hidden output 'nwo_{jk}'

For j=1 to h

For k=1 to m

$$nwo[j][k] = wo[j][k] + (\eta * \mathrm{eo[k]} * \mathrm{oh[j]})$$

Step 12: New weight for input hidden layer is calculated as follows : 'nwh_{ij}'

For i=1 to n

For j=1 to h

$$nwh[i][j] = wh[i][j] + (\eta * \mathrm{eh[j]} * \mathrm{x[i]})$$

The new weight obtained for hidden output layer is nwo_{ij} and the new weight obtained for input hidden layer is nw_{ij}

Step 13: Replace old weights in hidden layer and output layer with new weights 'nwh_{ij}' and 'nwojk'

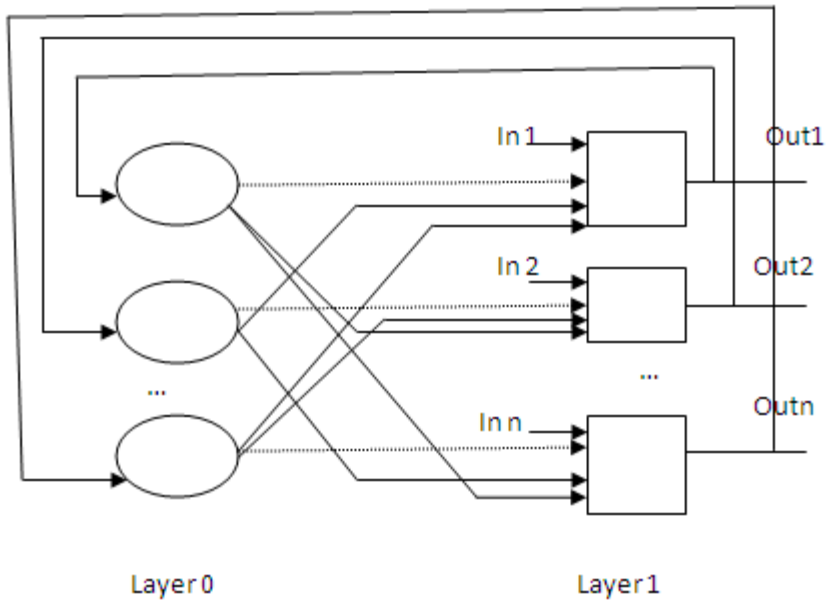
1.4 Hopfield Network

-Fully connected network

-Symmetric weights

Output:- Step function

Inputs: Bipolar inputs(+1 and -1)



Single Layer recurrent network

NET$_j$ = $\sum_{i \neq j} \omega_{ij}\, OUT_i + IN_j$

OUT_j=1 if $NET_j > T_j$

OUT_j=0 if $NET_j < T_j$

OUT_j=unchanged if $NET_j = T_j$

Hopfield Algorithm.

1. Assign common weights M-1

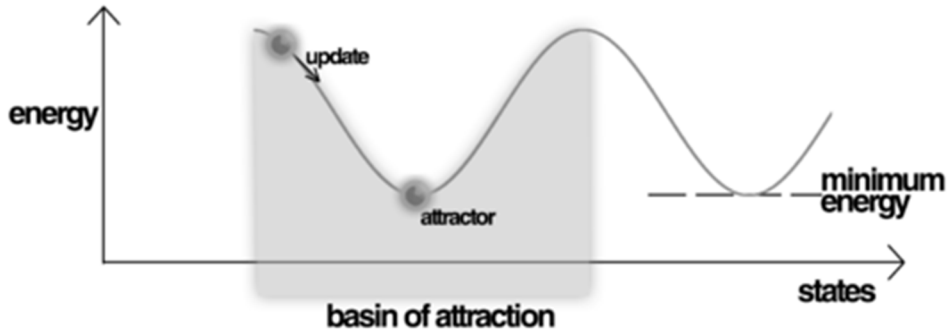
   $W_{ij} = \begin{cases} \sum_{s=0} X_i^s X_j^s & if \;\; i \neq j \\ 0 \; if \; i = j \end{cases}$

2. Initialize with unknown pattern

   $\mu_i(0) = \left[\sum_{i=0}^{N-1} x_i\,(t)\omega_{ij}\right]$ 0 ≤ i N-1
3. $\mu_i(t)$ is the output of node i at time t.
4. Iterate until convergence

   $\mu_i(t+1) = fn\left[\sum_{i=0}^{N-1} \mu_j\,(t)\omega_{ij}\right]$ =1 if net > θ, 0 if net < θ, and no change if net= θ

<u>Energy Landscape in Hopfield</u>



Hopfield nets have a scalar value associated with each state of the network referred to as the "energy", E, of the network, where: $\boxed{E = \frac{-1}{2}\sum_i \sum_{j \neq i} w_{ij} x_i x_j + \sum_i x_i T_i}$

E: Energy

T_i= Threshold

w_{ij}=weight

x_i=input

Function of energy landscape: Storage and retrieval. The units are randomly chosen for updation. When the units are chosen energy E will be decremented or stable. Repeated updation leads to the eventual convergence to a local minimum state in the energy function(Lyapunov function).Thus E is stable when state is local minimum.

Learning:

Local learning: A learning rule is *local* if each weight is updated using information available to neurons on either side of the connection that is associated with that particular weight.

Incremental learning: New patterns can be learned without using information from the old patterns that have been also used for training. That is, when a new pattern is used for training, the new values for the weights only depend on the old values and on the new pattern

Hebbian Learning rule for Hopfield networks:

The Hebbian rule is both local and incremental. For the Hopfield Networks, it is implemented in the following manner, when learning n binary patterns:

$w_{ij} = \frac{1}{n}\sum_{p=1}^{n} e_i^p \, e_j^p$ where e_i^p is the bit i, from pattern p. If the bit representation of i and j are equal, then the product, $e_i^p e_j^p$ is positive. This in turn will have a positive effect on weight w_{ij}. If the neurons are different then the product is negative.

Drawback of Hopfield Network

Converges to a local minima.So we go for Boltzman machine.

1.5 Boltzman Machine learning algorithm

Boltzman machine=Hopfield network+Probablistic update machine

Probability function is chosen to achieve great reduction in energy.

Two Phases

Phase 1

Incremental Boltzman algorithm:

1. Clamp input and output to correct value.
2. Let net cycle through states S_i.
   Calculate energy of a state S_i
   $\Delta E_{k} = \sum_i \omega_{ki} s_i - \theta_k$ 0≤i≤N-1
3. K^{th} neuron switch to lower energy with probability
   $P_k = \frac{1}{1+e^{-\Delta E_k / T}}$ (T=Temp)
4. Reduce T until output is stable.

Phase 2: Decremental

1. Clamp input only and leave output.
2. Let net reach thermal equilibrium as in phase 1.
3. Decrement weights between units if both ON(value=1).
4. Repeat until weights are stable.

1.6 SOM: Self Organizing Maps

A self-organizing map consists of components called nodes or neurons. Associated with each node are a weight vector of the same dimension as the input data vectors, and a position in the map space.

- Unsupervised learning
- to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map.
- they use a neighborhood function to preserve the topological properties of the input space.
- Two models available are Kohenon model and Willshaw model

Kohenon Model:

Introduced by Finnish professor Teuvo Kohonen in the 1980s

Two modes of operation:

1. Training
2. Mapping

Components of Self Organization

The self-organization process involves four major components:
Initialization: All the connection weights are initialized with small random values.
Competition: For each input pattern, the neurons compute their respective values of a discriminant function which provides the basis for competition. The particular neuron with the smallest value of the discriminant function is declared the winner.

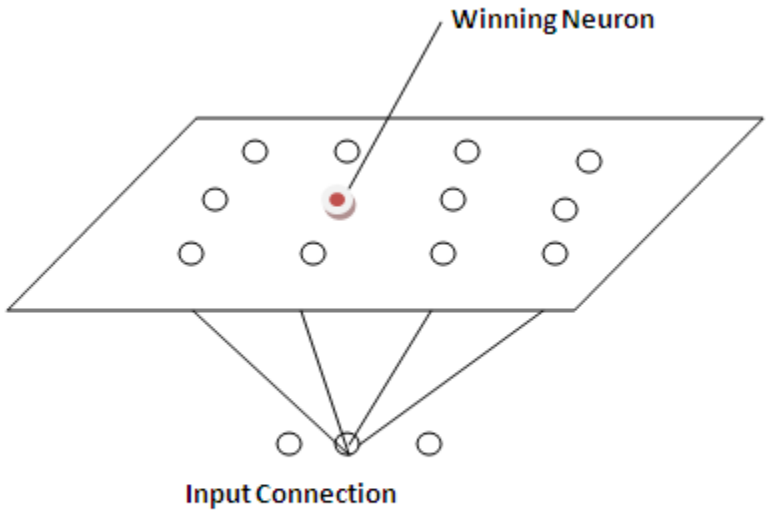


Fig. Two Dimensional Representation of Post synaptic neuron

Cooperation: The winning neuron determines the spatial location of a topological neighbourhood of excited neurons, thereby providing the basis for cooperation among neighbouring neurons.
Adaptation: The excited neurons decrease their individual values of the discriminant function in relation to the input pattern through suitable adjustment of the associated connection weights, such that the response of the winning neuron to the subsequent application of a similar input pattern is enhanced.

Training Data:

Let x_i be the input vectors (P distinct training vectors are taken)

Inputs

X_1: x_{11}, x_{12},……x_{1n}

X_2: x_{21}, x_{22},……x_{2n}

X_p: x_{p1}, x_{p2},……x_{pn}

Output: Vector y of length m

Y: y_1, y_2,……y_m

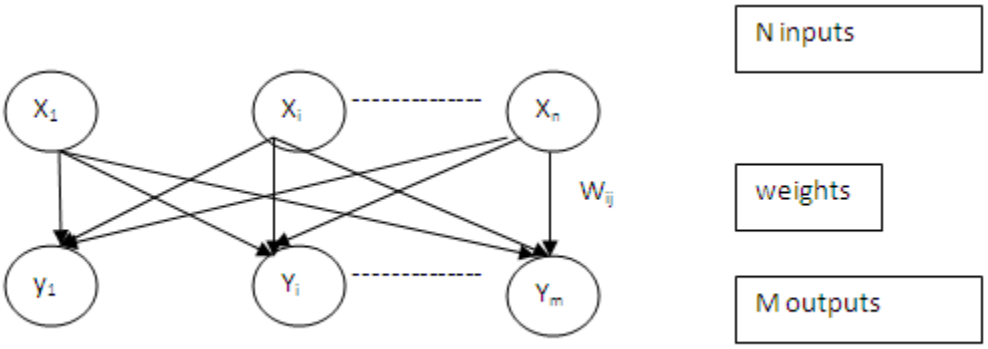
Network Architecture:

2 Layer of units

Input: n units

Output: m units

Inputs are fully connected with weights to outputs



Algorithm:

1. Select the output layer in the network topology
   Initialize the current neighbourhood distance D(0) to any positive value
2. Initialize the weights from inputs to outputs to small random value
3. Let t=1
4. Do while
   4.1. Select Input sample i_l
   4.2 Compute square of Euclidian distance of i_l from the weight vectors w_j
   $\sum_{k=1}^{n}(i_{l,k} - w_{j,k}(t))^2$
   4.3 Select output node j* having weight with minimum value(from step 2)
   4.4 Update weights to all nodes within atopological distance given by D(t) from j* using **weight update rule**

$$w_j(t+1) = w_j(1) + \eta(t)(i_j - w_j(t))$$

4.5 Increment t

End while

Generally ηdecreases with time $0 < \eta(t) \leq \eta(t-1) \leq 1$

Application:

1.Phonetic typewriter

2.Pattern recognition :Winning neurons with minimum distance are brought together in a single cluster.

References:

1. James A. Freeman, David M. Skapura,"neural networks, algorithms, applications and programming techniques, Pearson Education, 1991
2. http://lcn.epfl.ch/tutorial/english/perceptron/html/learning.html
3. https://en.wikipedia.org
4. John A. Bullinaria Introduction to Neural Networks : Lecture 16,SOM fundamentals, 2004 http://www.cs.bham.ac.uk/~jxb/NN/l16.pdf

UNIT- II

Fuzzy sets – Fuzzy rules and fuzzy reasoning – Fuzzy inference system – Mamdani fuzzy model – Sugeno fuzzy model – Tsukamoto fuzzy model.

Introduction

Fuzzy logic is being developed as a discipline to meet two objectives:

1. As a professional subject for building systems of high utility - for example fuzzy control.
2. As a theoretical subject - fuzzy logic is "symbolic logic with a comparative notion of truth developed fully in the spirit of classical logic. It is a branch of many-valued logic based on the paradigm of inference under vagueness."

What is Fuzzy Logic?

Fuzzy Logic is a form of multi-valued logic derived from fuzzy set theory to deal with reasoning that is approximate rather than precise. Fuzzy logic is **not a vague logic system**, but a system of logic for dealing with vague concepts. As in fuzzy set theory the set membership values can range (inclusively) between 0 and 1, in fuzzy logic the degree of truth of a statement can range between 0 and 1 and is not constrained to the two truth values true/false as in classic predicate logic.

Example for Fuzzy Logic

Problem: A real estate owner wants to classify the houses he offers to his clients. One main indicator of comfort of these houses is the **number of bedrooms in them**. Let the available types of houses be represented by the following set.

U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

The houses in this set are described by **U** number of bedrooms in a house. The realtor wants to describe a "comfortable house for a 4-person family," using a fuzzy set.

Solution: The fuzzy set "comfortable type of house for a 4-person family" may be described using a fuzzy set in the following manner.

HouseForFour =FuzzySet [{{1, 0.2}, {2, .5}, {3, .8}, {4, 1}, {5, .7}, {6, .3}},

Universal Set—>{1,10}];

FuzzyPlot [HouseForFour, ShowDots –> True];

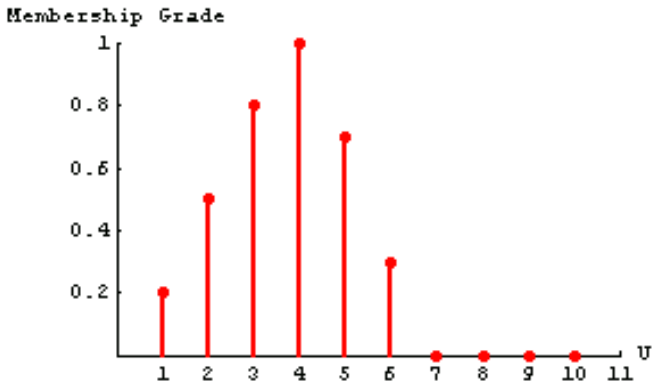


Fig 1. The Plot

Fuzzy System

A Fuzzy System can be contrasted with a conventional - crisp system in three main ways:

1. A **Linguistic Variable** is defined as a variable whose values are sentences in a natural or artificial language. Thus, “if tall”, “not tall”, “very tall”, “very very tall”, etc. are values of height, then height is a linguistic variable.

2. **Fuzzy Conditional Statements** are expressions of the form “If A THEN B”, where A and B have fuzzy meaning, e.g. “If x is small THEN y is large”, where small and large are viewed as labels of fuzzy sets.

3. **A Fuzzy Algorithm** is an ordered sequence of instructions which may contain fuzzy assignment and conditional statements, e.g., "x = very small, IF x is small THEN y is large". The execution of such instructions is governed by the compositional rule of inference and the rule of the preponderant alternative.

2.1 Fuzzy Sets

A Fuzzy set is a set whose elements have degrees of membership. Fuzzy sets are an extension of the classical notion of set (known as a Crisp Set). More mathematically, a fuzzy set is a pair (A, μA) where A is a set and μA : A → [0, 1]. For all x Є A, μA(x) is called the grade of membership of x. If μA(x) = 1, we say that x is **Fully Included** in (A, μA), and if μA(x) = 0, we say that x is Not Included in (A, μA). If there exists some x Є A such that μA(x) = 1, we say that (A, μA) is **Normal**. Otherwise, we say that (A, μA) is **Subnormal**.

A fuzzy set is denoted as:

A = μA(x1)/x1 + · · · + μA(xn)/xn

that belongs to a finite universe of discourse: A ⊆ {x1, x2, . . . , xn} = X

where μA(xi)/xi (a singleton) is a pair "grade of membership element".

Simple Example:

Consider X = {1, 2, . . . , 10}.

Suppose a child is asked which of the numbers in X are "large" relative to the others. The child might come up with the following:

Number	Comment	Degree
10	Definitely	1
9	Definitely	1
8	Quite possible	0.8
7	May be	0.5

6	Not usually	0.2
5,4,3,2,1	Definitely Not	0

Fig 2. Possible solution given by the child

Definitions on Fuzzy Sets

Following are the definitions for two fuzzy sets (A, μA) and (B, μB), where A, B ⊆ X:

- **Equality:** A = B if μA(x) = μB(x) for all x ∈ X
- **Inclusion:** A ⊆ B iff μA(x) ≤ μB(x) for all x ∈ X
- **Cardinality:** |A| = Xn i=1 μA(xi)
- **Empty Set:** A is empty iff μA(x) = 0 for all x ∈ X.
- **α-Cut:** Given α ∈ [0, 1], the α-cut of A is defined by Aα = {x ∈ X | μA(x) ≥ α}

Operations on Fuzzy Sets

Let (A, μA),(B, μB) be a fuzzy sets.

- Complementation: (¬A, μ¬A), where μ¬A = 1 − μA
- Height: h(A) = max x∈X μA(x)
- Support: supp(A) = {x ∈ X | μA(x) > 0}
- Core: C(A) = {x ∈ X | μA(x) = 1}
- Intersection: C = A ∩ B, where μC = min x∈X {μA, μB}
- Union: C = A ∪ B, where μC = max x∈X {μA, μB}
- Bounded Sum: C = A + B, where μC(x) = min{1, μA(x) + μB(x)}
- Bounded Difference: C = A − B, where μC(x) = max{0, μA(x) − μB(x)} 7

- Exponentiation: $C = A\alpha$ where $\mu C = (\mu A)\ \alpha$ for $\alpha > 0$
- Level Set: $C = \alpha A$ where $\mu C = \alpha \mu A$ for $\alpha \in [0, 1]$
- Concentration: $C = A\alpha$ where $\alpha > 1$
- Dilation: $C = A\alpha$ where $\alpha < 1$

Note that $A \cap \neg A$ is not necessarily the empty set, as would be the case with classical set theory. Also, if A is crisp, then $A\alpha = A$ for all α. We will define the Cartesian product $A \times B$ to be the same as $A \cap B$.

Membership Functions

A *membership function* (MF) is a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1. The input space is sometimes referred to as the *universe of discourse*, a fancy name for a simple concept.

The most commonly used MFs are

- Triangles
- Trapezoids
- Bell Curves
- Gaussian and
- Sigmoidal

2.2 Fuzzy Rules

Human beings make decisions based on rules. Although, we may not be aware of it, all the decisions we make are all based on computer like if-then statements. If the weather is fine, then we may decide to go out. If the forecast says the weather will be bad today, but fine

tomorrow, then we make a decision not to go today, and postpone it till tomorrow. Rules associate ideas and relate one event to another.

Fuzzy machines, which always tend to mimic the behavior of man, work the same way. However, the decision and the means of choosing that decision are replaced by fuzzy sets and the rules are replaced by fuzzy rules. Fuzzy rules also operate using a series of if-then statements. For instance, if X then A, if y then b, where A and B are all sets of X and Y. Fuzzy rules define **fuzzy *patches***, which is the key idea in fuzzy logic.
A machine is made smarter using a concept designed by Bart Kosko called the Fuzzy Approximation Theorem (FAT). The FAT theorem generally states a finite number of patches can cover a curve as seen in the figure below. If the patches are large, then the rules are sloppy. If the patches are small then the rules are fine.

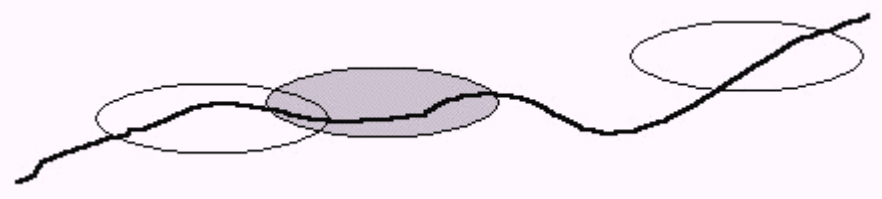


Fig 3. Fuzzy Patches

In a fuzzy system this simply means that all our rules can be seen as patches and the input and output of the machine can be associated together using these patches. Graphically, if the rule patches shrink, our fuzzy subset triangles get narrower. Simple enough? Yes, because even novices can build control systems that beat the best math models of control theory. Naturally, it is math-free system.

2.3 Fuzzy Reasoning

Single rule with single antecedent

Rule: if x is A then y is B

Fact: x is A’

Conclusion: y is B’

The i-th fuzzy rule from this rule-base

$$R_i : \text{if } x \text{ is } A_i \text{ and } y \text{ is } B_i \text{ then } z \text{ is } C_i \text{ is}$$

implemented by a fuzzy relation Ri and is

defined as

$$Ri(u, v, w)=(Ai \times Bi \rightarrow Ci)(u, w)$$

$$= [Ai(u) \times Bi(v)] \rightarrow Ci(w) \quad \text{for } i = 1,...,n.$$

2.4 Fuzzy Inference (Expert) system

A fuzzy inference system (FIS) is a system that uses fuzzy set theory to map inputs (features in the case of fuzzy classification) to outputs (classes in the case of fuzzy classification).

Fuzzy inference systems have been successfully applied in fields such as automatic control, data classification, decision analysis, expert systems, and computer vision. Because of its multidisciplinary nature, fuzzy inference systems are associated with a number of names, such as fuzzy-rule-based systems, fuzzy expert systems, fuzzy modeling, fuzzy associative memory, fuzzy logic controllers, and simply (and ambiguously) fuzzy systems.

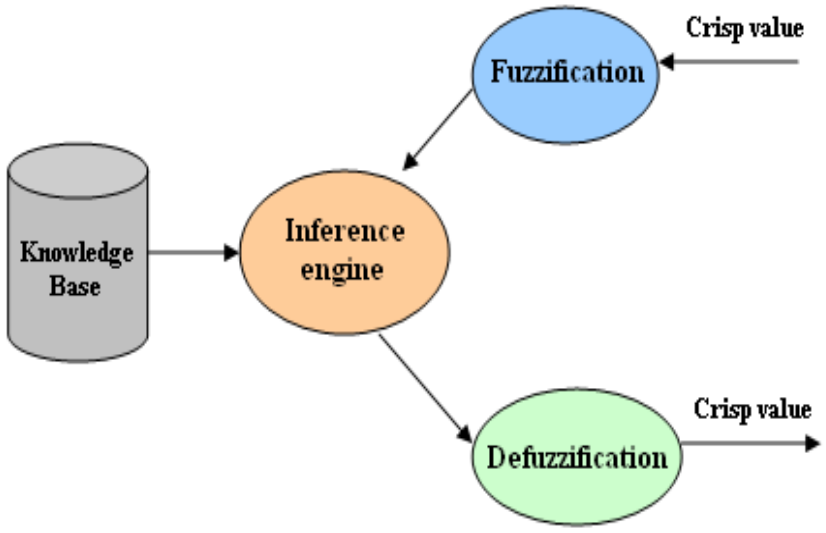


Fig 4. Structure of a Fuzzy Expert System

The rules in FIS (sometimes may be called as fuzzy expert system) are fuzzy production rules of the form:

- *if p then q,* where p and q are fuzzy statements.

For example, in a fuzzy rule

- if x is low and y is high then z is medium.
- Here x is low; y is high; z is medium are fuzzy statements; x and y are input variables; z is an output variable, low, high, and medium are fuzzy sets.

The antecedent describes to what degree the rule applies, while the conclusion assigns a fuzzy function to each of one or more output variables. Most tools for working with fuzzy expert systems allow more than one conclusion per rule.

The set of rules in a fuzzy expert system is known as ***knowledge base***.

The functional operations in fuzzy expert system proceed in the following steps.

- Fuzzification
- Fuzzy Inferencing (apply implication method)
- Aggregation of all outputs
- Defuzzification

Fuzzification

- In the process of fuzzification, membership functions defined on input variables are applied to their actual values so that the degree of truth for each rule premise can be determined.
- Fuzzy statements in the antecedent are resolved to a degree of membership between 0 and 1.

  - If there is only one part to the antecedent, then this is the degree of support for the rule.
  - If there are multiple parts to the antecedent, apply fuzzy logic operators and resolve the antecedent to a single number between 0 and 1.

- Antecedent may be joined by OR; AND operators.
  - For OR -- max
  - For AND -- min

Fuzzy Inferencing

In the process of inference

  - Truth value for the premise of each rule is computed and applied to the conclusion part of each rule.
  - This results in one fuzzy set to be assigned to each output variable for each rule.

The use of degree of support for the entire rule is to shape the output fuzzy set. The consequent of a fuzzy rule assigns an entire fuzzy set to the output. If the antecedent is only partially true, (i.e., is assigned a value less than 1), then the output fuzzy set is truncated according to the implication method. If the consequent of a rule has multiple parts, then all consequents are affected equally by the result of the antecedent. The consequent specifies a fuzzy set to be assigned to the output. The implication function then modifies that fuzzy set to the degree specified by the antecedent.

The following functions are used in inference rules.

- *min* or *prod* are commonly used as inference rules.
- *min*: truncates the consequent's membership function
- *prod*: scales it.

Aggregation of all outputs

It is the process where the outputs of each rule are combined into a single fuzzy set.

- The input of the aggregation process is the list of truncated output functions returned by the implication process for each rule.
- The output of the aggregation process is one fuzzy set for each output variable.
  - Here, all fuzzy sets assigned to each output variable are combined together to form a single fuzzy set for each output variable using a fuzzy aggregation operator.

Some of the most commonly used aggregation operators are

- the maximum : point-wise maximum over all of the fuzzy sets
- the sum : (point-wise sum over all of the fuzzy) the probabilistic sum.

Defuzzification

In Defuzzification, the fuzzy output set is converted to a crisp number.

Some commonly used techniques are the *centroid* and *maximum* methods.

- In the *centroid method*, the crisp value of the output variable is computed by finding the variable value of the centre of gravity of the membership function for the fuzzy value.
- In the *maximum method*, one of the variable values at which the fuzzy set has its maximum truth value is chosen as the crisp value for the output variable.

Some other methods for defuzzification are:

- bisector, middle of maximum (the average of the maximum value of the output set), largest of maximum, and smallest of maximum, etc.

There are two types of fuzzy inference systems that can be implemented in the Fuzzy Logic Toolbox: Mamdani-type and Sugeno-type.

2.5 Mamdani Fuzzy Model

Mamdani's fuzzy inference method is the most commonly seen fuzzy methodology. Mamdani's method was among the first control systems built using fuzzy set theory. It was proposed in 1975 by Ebrahim Mamdani as an attempt to control a steam engine and boiler combination by synthesizing a set of linguistic control rules obtained from experienced human operators.

To compute the output of this FIS given the inputs, one must go through six steps:

1. Determining a set of fuzzy rules

2. Fuzzifying the inputs using the input membership functions,

3. Combining the fuzzified inputs according to the fuzzy rules to establish a rule strength,

4. Finding the consequence of the rule by combining the rule strength and the output membership function,

5. Combining the consequences to get an output distribution, and

6. Defuzzifying the output distribution (this step is only if a crisp output (class) is needed).

The following is a more detailed description of this process.

The following is a more detailed description of this process.

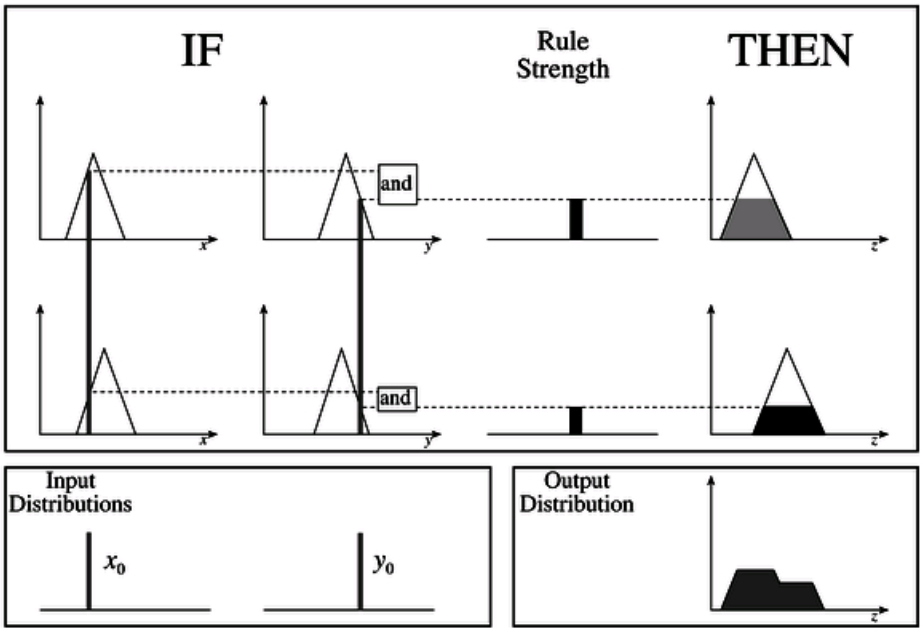


Fig 5. A two input, two rule Mamdani FIS with crisp inputs

Creating fuzzy rules

Fuzzy rules are a collection of linguistic statements that describe how the FIS should make a decision regarding classifying an input or controlling an output. Fuzzy rules are always written in the following form:

if *(input1 is membership function1)* ***and/or*** *(input2 is membership function2)* ***and/or then*** *(output$_n$ is output membership function$_n$).*

Example

$$\begin{cases} \text{If } X \text{ is small then } Y \text{ is small.} \\ \text{If } X \text{ is medium then } Y \text{ is medium.} \\ \text{If } X \text{ is large then } Y \text{ is large.} \end{cases}$$

Another Example

if *temperature is high* ***and*** *humidity is high* ***then*** *room is hot.*

There would have to be membership functions that define what we mean by high temperature (input1), high humidity (input2) and a hot room (output1). This process of taking an input such as temperature and processing it through a membership function to determine what we mean by "high" temperature is called fuzzification.Also, we must define what we mean by "and" / "or" in the fuzzy rule. This is called fuzzy combination.

Fuzzification

The purpose of fuzzification is to map the inputs from a set of sensors (or features of those sensors such as amplitude or spectrum) to values from 0 to 1 using a set of input membership functions. In the example shown in the above figure, there are two inputs, x0 and y0 shown at the lower left corner. These inputs are mapped into fuzzy numbers by drawing a line up from the inputs to the input membership functions above and marking the intersection point.

These input membership functions, as discussed previously, can represent fuzzy concepts such as "large" or "small", "old" or "young", "hot" or "cold", etc. When choosing the input membership functions, the definition of what we mean by "large" and "small" may be different for each input.

Fuzzy Combinations

In making a fuzzy rule, we use the concept of "and", "or", and sometimes "not". The sections below describe the most common definitions of these "fuzzy combination" operators. Fuzzy combinations are also referred to as "T-norms".

a) **Fuzzy “and”**

The fuzzy "and" is written as:

$$u_{A \cap B} = T(u_A(x), u_B(x))$$

where μ_A is read as "the membership in class A" and μ_B is read as "the membership in class B". There are many ways to compute "and". The two most common are:

1. Zadeh - min($u_A(x)$, $u_B(x)$) This technique, named after the inventor of fuzzy set theory simply computes the "and" by taking the minimum of the two (or more) membership values. This is the most common definition of the fuzzy "and".

2. Product - $u_a(x)$ times $u_b(x)$) This techniques computes the fuzzy "and" by multiplying the two membership values.

Both techniques have the following two properties:

T(0,0) = T(a,0) = T(0,a) = 0

T(a,1) = T(1,a) = a

One of the nice things about both definitions is that they also can be used to compute the Boolean "and". The table below shows the Boolean "and" operation. Notice that both fuzzy "and" definitions also work for these numbers. The fuzzy "and" is an extension of the Boolean "and" to numbers that are not just 0 or 1, but between 0 and 1.

Input1 (A)	**Input2 (B)**	**Output (A "and" B)**
0	0	0
0	1	0
1	0	0
1	1	1

The Boolean "and"

b) Fuzzy “or”

The fuzzy "or" is written as:

$$u_{A \cup B} = T(u_A(x), u_B(x))$$

Similar to the fuzzy "and", there are two techniques for computing the fuzzy "or":

1. Zadeh - $\max(u_A(x), u_B(x))$ This technique computes the fuzzy "or" by taking the maximum of the two (or more) membership values. This is the most common method of computing the fuzzy "or".

2. Product - $u_A(x) + u_B(x) - u_A(x)\, u_B(x)$ This technique uses the difference between the sum of the two (or more) membership values and the product of the membership values.

Both techniques have the following properties:

$T(a,0) = T(0,a) = a$

$T(a,1) = T(1,a) = 1$

Similar to the fuzzy "and", both definitions of the fuzzy "or" also can be used to compute the Boolean "or". The table below shows the Boolean "or" operation. Notice that both fuzzy "or" definitions also work for these numbers. The fuzzy "or" is an extension of the Boolean "or" to numbers that are not just 0 or 1, but between 0 and 1.

Input1 (A)	**Input2 (B)**	**Output (A "or" B)**
0	0	0
0	1	1
1	0	1
1	1	1

The Boolean "or"

c) Consequence

The consequence of a fuzzy rule is computed using two steps:

1. Computing the rule strength by combining the fuzzified inputs using the fuzzy combination process discussed previously. This is shown in Fig 5. Notice in this example, the fuzzy "and" is used to combine the membership functions to compute the rule strength.

2. Clipping the output membership function at the rule strength. Once again, refer to Fig 5. to see how this is done for a two input, two rule Mamdani FIS.

d) Combining Outputs into an Output Distribution

The outputs of all of the fuzzy rules must now be combined to obtain one fuzzy output distribution. This is usually, but not always, done by using the fuzzy "or". Figure 5 shows an example of this. The output membership functions on the right hand side of the figure are combined using the fuzzy "or" to obtain the output distribution shown on the lower right corner of the figure.

e) Defuzzification of Output Distribution

In many instances, it is desired to come up with a single crisp output from a FIS. For example, if one was trying to classify a letter drawn by hand on a drawing tablet, ultimately the FIS would have to come up with a crisp number to tell the computer which letter was drawn. This crisp number is obtained in a process known as defuzzification. There are two common techniques for defuzzifying:

1. **Center of mass** - This technique takes the output distribution found previously and finds its center of mass to come up with one crisp number. This is computed as follows:

$$z = \frac{\sum_{j=1}^{q} Z_j u_c(Z_j)}{\sum_{j=1}^{q} u_c(Z_j)}$$

where z is the center of mass and u_c is the membership in class c at value z_j. An example outcome of this computation is shown in Fig 6.

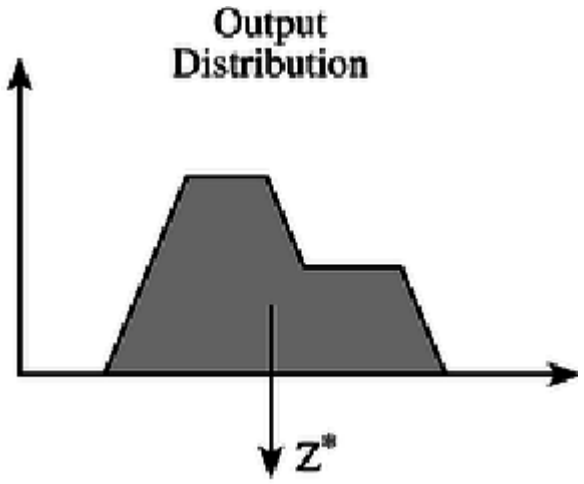


Fig 6. Defuzzification Using the Center of Mass

2. **Mean of maximum** - This technique takes the output distribution found previously and finds its mean of maxima to come up with one crisp number. This is computed as follows:

$$z = \sum_{j=1}^{l} \frac{z_j}{l}$$

where z is the mean of maximum, z_j is the point at which the membership function is maximum, and l is the number of times the output distribution reaches the maximum level. An example outcome of this computation is shown in Figure 7.

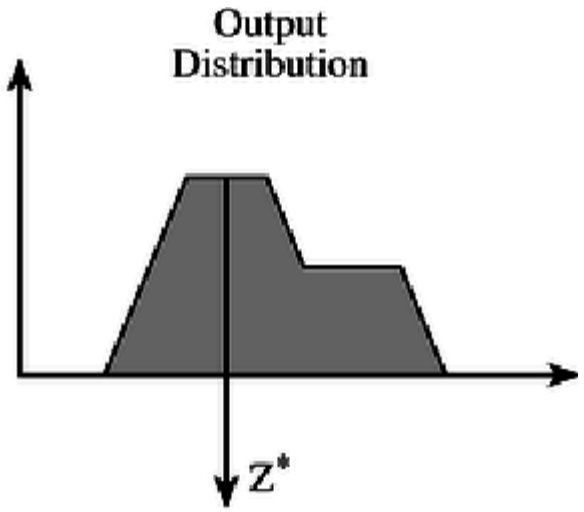


Fig 7. Defuzzification Using the Mean of Maximum

Fuzzy Inputs

In summary, Fig 5 shows a two input Mamdani FIS with two rules. It fuzzifies the two inputs by finding the intersection of the crisp input value with the input membership function. It uses the minimum operator to compute the fuzzy "and" for combining the two fuzzified inputs to obtain a rule strength. It clips the output membership function at the rule strength. Finally, it uses the maximum operator to compute the fuzzy "or" for combining the outputs of the two rules.

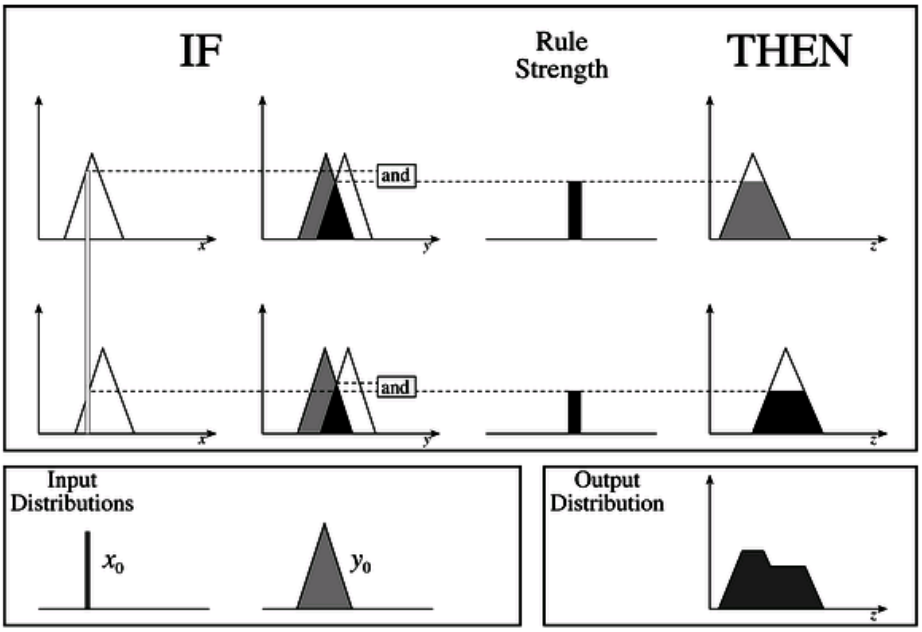


Fig 8. A two Input, two rule Mamdani FIS with a fuzzy input

Fig 8 shows a modification of the Mamdani FIS where the input y_0 is fuzzy, not crisp. This can be used to model inaccuracies in the measurement. For example, we may be measuring the output of a pressure sensor. Even with the exact same pressure applied, the sensor is measured to have slightly different voltages. The fuzzy input membership function models this uncertainty. The input fuzzy function is combined with the rule input membership function by using the fuzzy "and" as shown in Fig 8.

2.6 Sugeno Fuzzy Model

The Sugeno FIS is quite similar to the Mamdani FIS. The primary difference is that the output consequence is not computed by clipping an output membership function at the rule strength. In fact, in the Sugeno FIS there is no output membership function at all. Instead the output is a crisp number computed by multiplying each input by a constant and then adding up the results. This is shown in Figure 9. "Rule strength" in this example is referred to as "degree of applicability" and the output is referred to as the "action". Also notice that there is no output distribution, only a "resulting action" which is the mathematical combination of the rule strengths (degree of applicability) and the outputs (actions).

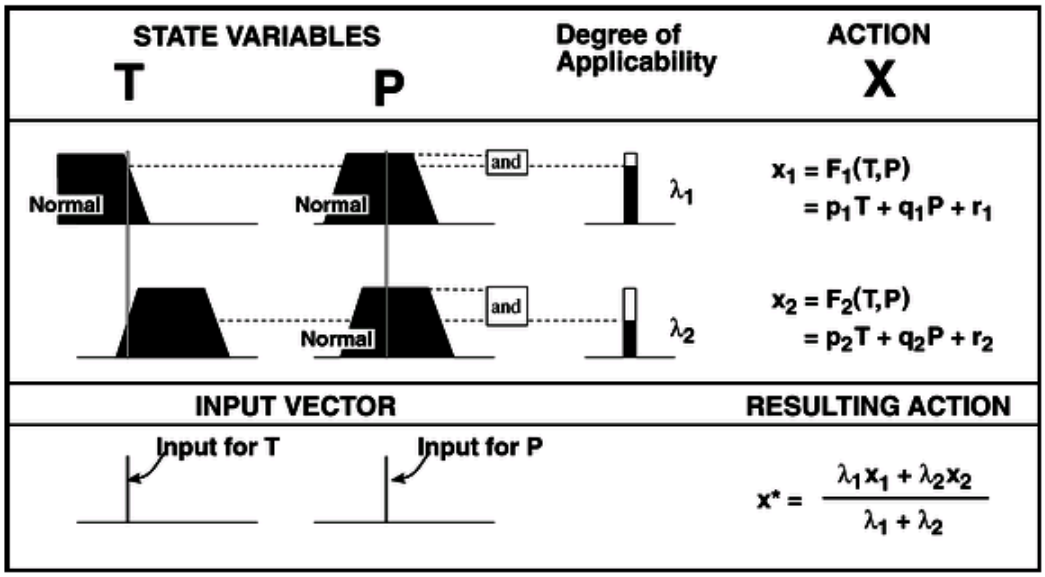


Fig 9. A two input, two rule Sugeno FIS (p_n, q_n, and r_n are user-defined constants)

One of the large problems with the Sugeno FIS is that there is no good intuitive method for determining the coefficients, p, q, and r. Also, the Sugeno has only crisp outputs which may not be what is desired in a given HCI application. Why then would you use a Sugeno FIS rather than a Mamdani FIS? The reason is that there are algorithms which can be used to automatically optimize the Sugeno FIS.

In classification, p and q can be chosen to be 0 and r can be chosen to be a number that corresponds to a particular class. For example, if we wanted to use the EMG from a person/persons forearm to classify which way his/her wrist was bending, we could assign the class "bend_inward" to have the value r = 1. We could assign the class "bend_outward" to have the value r=0. Finally, we could assign the class "no_bend" to have the value r=0.5.

2.7 Tsukamoto fuzzy model

In the *Tsukamoto fuzzy models*, the consequent of each fuzzy if-then rule is represented by a fuzzy set with a monotonical membership function, as shown in Figure 10. As a result, the inferred output of each rule is defined as a crisp value induced by the rule's firing strength. The overall output is taken as the weighted average of each rule's output. Figure 10 illustrates the reasoning procedure for a two-input two-rule system.

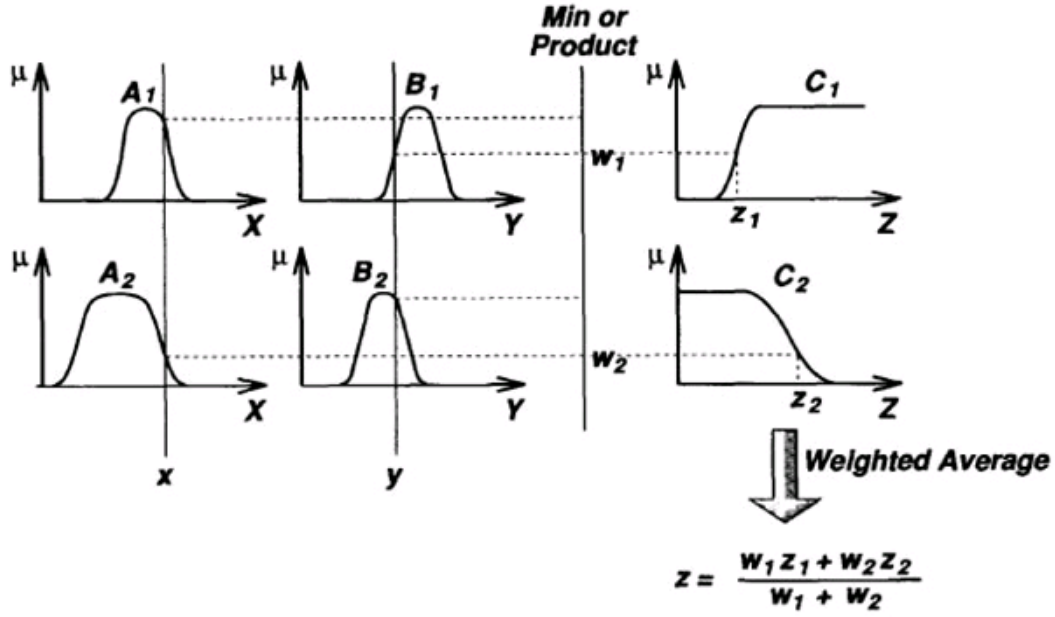


Fig 10. The Tsukamoto fuzzy model

Since each rule infers a crisp output, the Tsukamoto fuzzy model aggregate each rule's output by the method of weighted average and thus avoids the time-consuming process of defuzzification. However, the Tsukamoto fuzzy model is not used often since it is not as transparent as either the Mamdani or Sugeno fuzzy models. The following is a single-input example.

Example: Single-input Tsukamoto fuzzy model

An example of a single-input Tsukamoto fuzzy model can be expressed as:

$$\begin{cases} IF\ X\ is\ small\ then\ Y\ is\ C_1 \\ IF\ X\ is\ medium\ then\ Y\ is\ C_2 \\ IF\ X\ is\ large\ then\ Y\ is\ C_3 \end{cases}$$

Where the antecedent MFs for "small", "medium", and "large" are shown in Figure 11(a), and the consequent MFs for "C_1", "C_2", and "C_3" are shown in Figure 11(b). The overall input-output curve, as shown in Figure 11(d), is equal to $(\sum_{i=1}^{3} w_i f_i)/(\sum_{i=1}^{3} w_i)$, where f_i is the output of each rule induced by the firing strength W_i and MF for C_i. If we plot each rule's output f_i as a function of x, we obtain Figure 11(c), which is not quite obvious from the original rule base and MF plots.

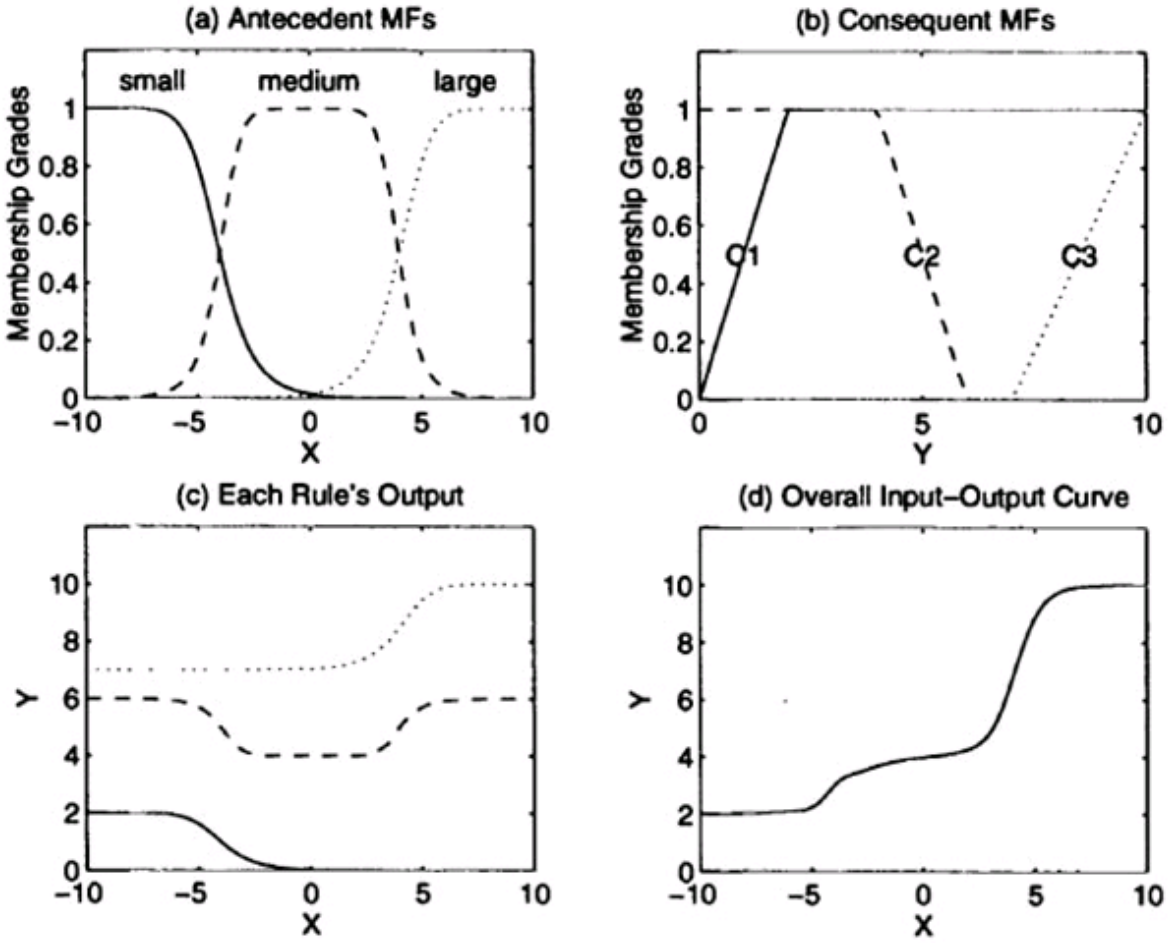


Fig 11. Single-input single output Tsukamoto fuzzy model: (a) antecedent MFs; (b) consequent MFs; (c) each rule's output curve; (d) overall input-output curve.

Since the reasoning mechanism of the Tsukamoto fuzzy model does not follow strictly the compositional rule of inference, the output is always crisp even when the inputs are fuzzy.

References:

- https://www.calvin.edu/~pribeiro/othrlnks/Fuzzy/fuzzydecisions.htm
- Jang, sun, Mitzutani,"Neuro Fuzzy and Soft computing", Prentice Hall India, 2006

Unit III

Adaptive Neuro Fuzzy Inference System- Co active Neuro Fuzzy modelling- Classification and Regression Trees- Data Clustering Algorithms- Rule based structure-Neuro Fuzzy control 1- Neuro Fuzzy control 2- Fuzzy Decision Making

3.1ANFIS

An **adaptive neuro-fuzzy inference system** or **adaptive network-based fuzzy inference system** (**ANFIS**) is a kind of artificial neural network that is based on Takagi–Sugeno fuzzy inference system. The technique was developed in the early 1990s. Since it integrates both neural networks and fuzzy logic principles, it has potential to capture the benefits of both in a single framework. Its inference system corresponds to a set of fuzzy IF–THEN rules that have learning capability to approximate nonlinear functions.

For simplicity, we assume that the fuzzy inference sytem under consideration has two inputs x and y and one output z. For a first-orderTakagi-Sugeno fuzzy model, a common rule set with two fuzzy if-then rules is the following:

- **Rule 1:** If x is A_1 and y is B_1, then $f_1=p_1x+q_1y+r_1$;
- **Rule 2:** If x is A_2 and y is B_2, then $f_2=p_2x+q_2y+r_2$;

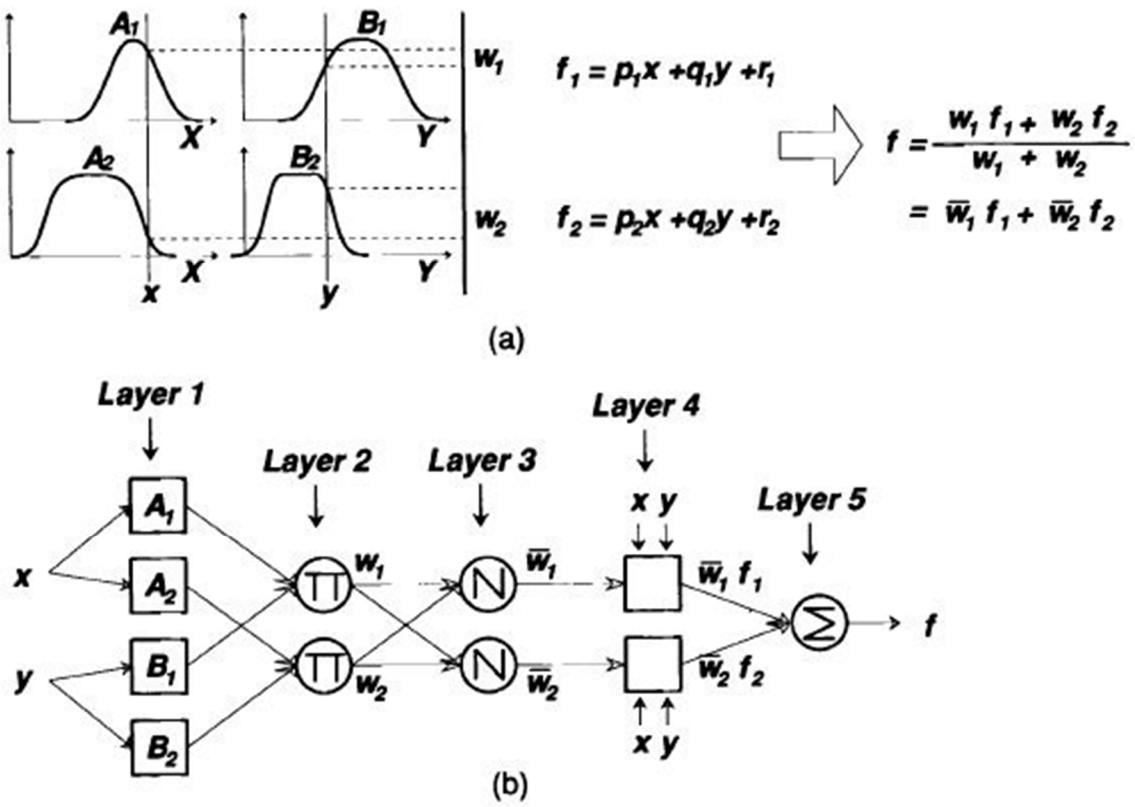


Figure (a) A two inputs first order Takagi-Sugeno fuzzy model with two rules; (b) The equivalent ANFIS architecture.

Figure (a) illustrates the reasoning mechanism for this Takagi-Sugeno model; where nodes of the same layer have similar functions. (Here we denote the output of the *i*th node in layer *l* as $O_{l,i}$)

Layer 1 Every node *i* in this layer is an adaptive node with a node function

$$O_{1,i} = \mu_{A_i}(x), \quad \text{for } i = 1, 2, \text{ or}$$
$$O_{1,i} = \mu_{B_{i-2}}(y), \quad \text{for } i = 3, 4,$$

where *x* (or *y*) is the input to node *i* and A_i (or B_{i-2}) is a linguistic label (such as "small" or "large") associated with this node. In other words, $O_{1,i}$ is the membership grade of a fuzzy set *A* ($=A_1, A_2, B_1$ *or* B_2) and it specifies the degree to which the given input *x* (or *y*) satifies the quantifier *A*. Here the membership function for *A* can be any appropriate parameterized membership function introduced inhere, such as the generalized bell function:

$$\mu_A(x) = \frac{1}{1 + \left|\frac{x - c_i}{a_i}\right|^{2b}},$$

where $\{a_i, b_i, c_i\}$ is the parameter set. As the values of these parameters change, the bell-shaped function varies accordingly, thus exhibiting various forms of membership function for fuzzy set *A*. Parameters in this layer are referred to as **premise parameters**.

Layer 2 Every node in this layer is a fixed node labeled , whose output is the product of all the incoming signals:

$$O_{2,i} = w_i = \mu_{A_i}(x)\mu_{B_i}(y), \ i = 1, 2.$$

Each node output represents the **firing strength** of a rule. In general, any other T-norm operators that perform fuzzy AND can be used as the node function in this layer.

Layer 3 Every node in this layer is a fixed node labeled N. The *i*th node calculates the ratio of the *i*th rule's firing strength to the sum of all rules' firing strenghts:

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2}, \ i = 1, 2.$$

For convenience, outputs of this layer are called **normalized firing strengthes**.

Layer 4 Every node *i* in this layer is an adaptive node with a node function:

$$O_{4,i} = \overline{w}_i f_i = \overline{w}_i(p_i x + q_i y + r_i),$$

where wi is a normalized firing strength from layer 3 and $\{p_i, q_i, r_i\}$ is the parameter set of this node. Parameters in this layer are referred to as **consequent parameters**.

Layer 5 The single node in this layer is a fixed node labeled ∑, which computes the overall output as the summation of all incoming singals:

$$\text{overall output} = O_{5,1} = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

Hybrid Learning Algorithm

i)The ANFIS can be trained by a hybrid learning algorithm.

ii)In the forward pass the algorithm uses least-squares method to identify the consequent parameters on the layer 4.

iii) In the backward pass the errors are propagated backward and the premise parameters are updated by gradient descent.

Basic Learning Rule Definitions

Suppose that an adptive network has L layers and the kth layer has #(k) nodes.

We can denote the node in the ith position of the kth layer by (k, i). The node function is denoted by O_i^k .

Since the node output depends on its incoming signals and its parameter set (a, b, c), we have $O^k_i = O^k_i (O_i^{k-1} , \ldots , O_{\#(k-1)}^{k-1}, a, b, c)$

Notice that $O_k i$ is used as both node output and node function.

Error Measure

Assume that a training data set has P entries. The error measure for the pth entry can be defined as the sum of the squared error

$$Ep = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^L)^2$$

$T_{m,p}$ is the mth component of the pth target.

$O^L_{m,p}$ is the mth component the actual output vector. The overall error is

$$E = \sum_{p=1}^{p} E_p$$

ANFIS is a Universal Aproximator

When the number of rules is not restricted, a zero-order Sugeno model has unlimited approximation power for matching well any nonlinear function arbitrarily on a compact set. This

can be proved using the Stone-Weierstrass theorem. Let domain D be a compact space of N dimensions, and let F be a set of continuous real-valued functions on D satisfying the following criteria:

Stone-Weierstrauss theorem – I

Indentity function: The constant f (x) = 1 is in F. Separability: For any two points x1 ≠ x 2 in D, there is an f in F such that f (x1) ≠f (x 2).

Algebraic closure: If f and g are any two functions in F, then fg and af + bg are in F for any two real numbers a and b.

Indentity Function

Indentity function: The constant f (x) = 1 is in F. The first hypothesis requires that our fuzzy inference system be able to compute the identity function f (x) = 1. An obvious solution is to set the consequence part of each rule equal to one.

Separability: For any two points x1 # x2 in D, there is an f in F such that f(x1) # f(x2). The second hypothesis requires that our fuzzy inference system be able to compute functions that have different values for different points. This is achievable by any fuzzy inference system with appropriate parameters.

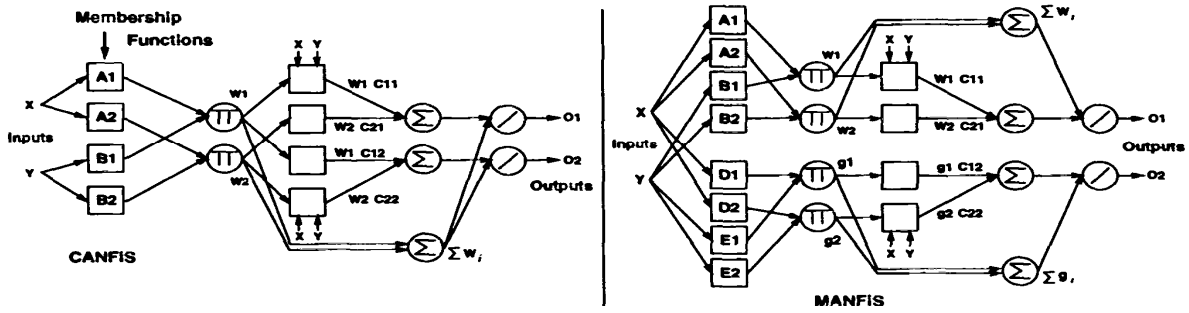
Algebraic closure addition: If f and g are any two functions in F, then af + bg are in F for any two real numbers a and b. • Suppose that we have two fuzzy inference systems S and Sˆ; each of them has two rules. • The final output of each system is specified as

$$S: Z = \frac{w.f1 + ws2.f2}{w1 + w2}$$

3.2CANFIS (coactive neuro fuzzy inference system)

CANFIS has extended basic ideas of its predecessor ANFIS (Adaptive Network based Fuzzy Inference System).In this ANFIS concept has been extended to any number of input-/output pairs .In addition, CANFIS yields advantages from non linear fuzzy rules. This CANFIS realizes the

sugeno –type (or TSK)fuzzy inferencing accomplishing fuzzy ifthen rules such as ,If X is A1 and Y is B1, Then C1=p1X+q1Y+r1.



FRAMEWORK

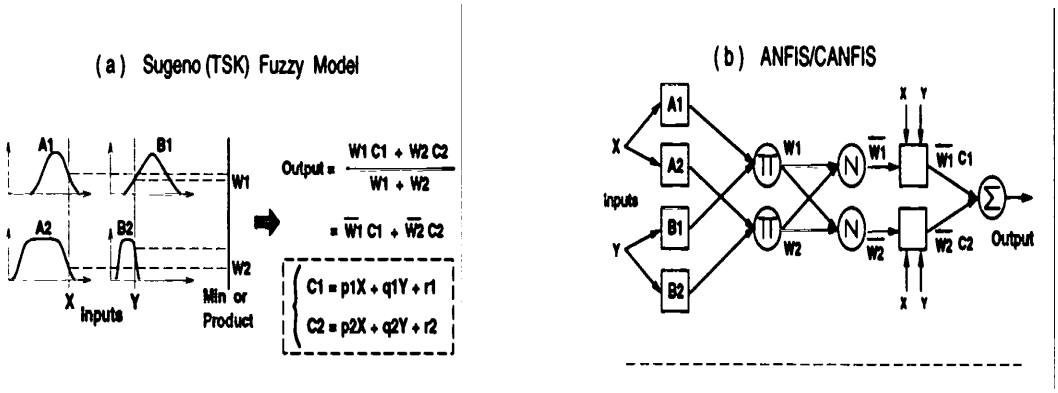
Toward Multiple Inputs/Outputs Systems

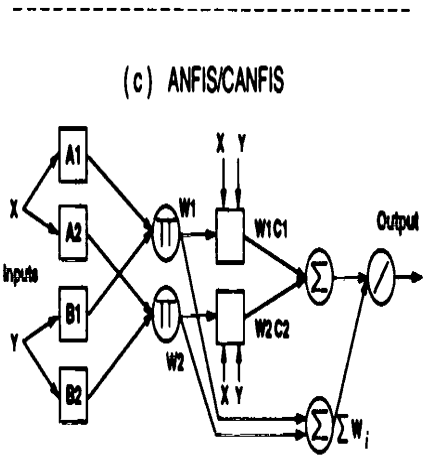
CANFIS has extended the notion of a single-output system, ANFIS, to produce multiple outputs. One way to get multiple outputs is to place as many ANFIS models side by side as there are required outputs. In this MANFIS (multiple ANFIS) model, no modifiable parameters are shared by

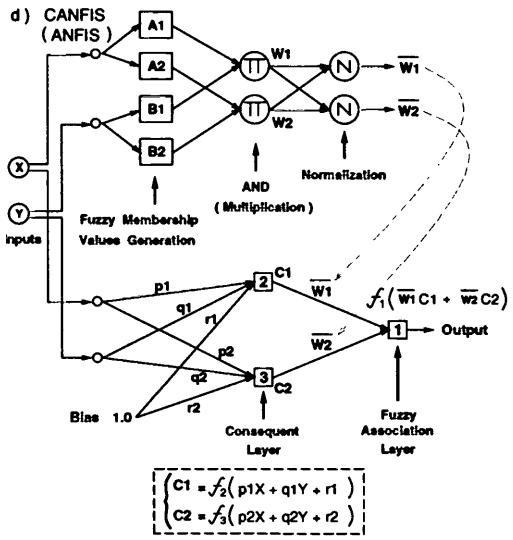
the juxtaposed ANFIS models. That is, each ANFIS has an independent set of fuzzy rules, which makes it difficult to realize possible certain correlations between outputs. An additional concern resides in the number of adjustable parameters, which drastically increases as outputs increase.

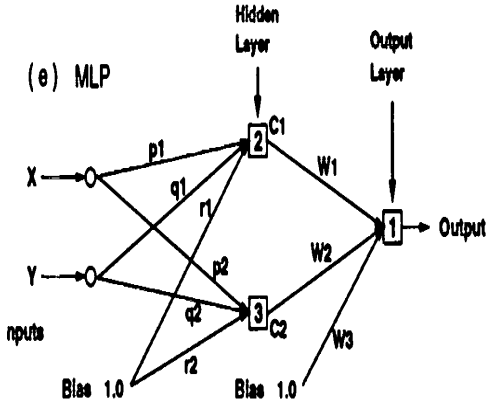
Another way of generating multiple outputs is to maintain the same antecedents of fuzzy rules among multiple ANFIS models.

Architectural Comparisons









3.3 CART: Classification and Regression Trees

In ANFIS, learning rules deal only with parameter identification. Still there is a need for structure identification.

Issues in structure identification:

1. Selecting relevant input variables
2. Determining initial ANFIS structure

i. Input space partitioning

ii. Number of MFs for each input

iii. Antecedent(premise) part of fuzzy rules

iv. Consequent part of fuzzy rules

3. Choosing the initial parameters for MFs

CART-> Quick method to solve problem of structure identification

Resulting ANFIS architecture based on CART is both efficient in training and application because of weight normalization.

Decision Trees(DT)

Partitions the input space into mutually exclusive regions(Assigned lable/action/value)

-DT is a structure with internal and external nodes

* nodes connected by branches

* internal node is decision making unit

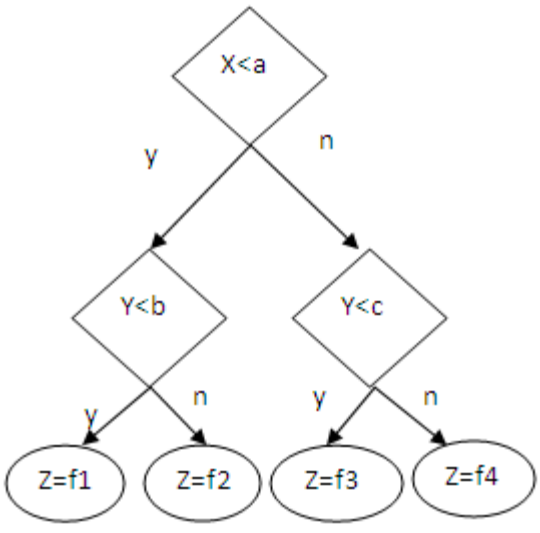
* External node: (leaf/terminal node) : has no child(assigned with label / value

- Depending on the result of decision function, the tree will branch to node's child

- Binary decision tree is a decision tree with two children

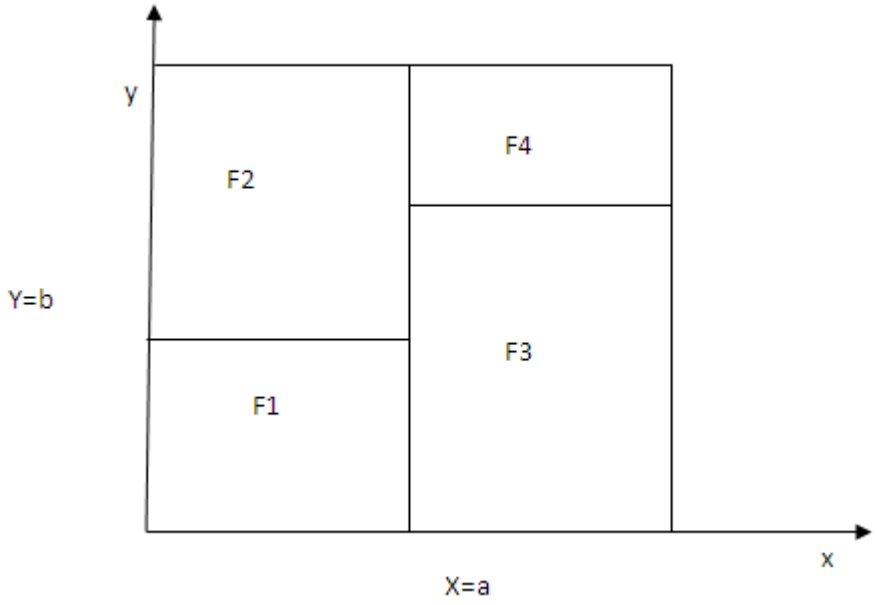
- DT used for classification problems are called classification trees(CT)

- DT for regression problems are Regression Trees



Binary DT

Input Space partitioning for Binary DT



CART Algorithm for tree induction

- It refers AID(Automatic Interaction Detection)

  Steps

  1. Tree growing based on sample dataset
  2. Prunes tree based on the minimum cost complexity principle

Tree Growing

CART grows DT determining the success of splits(partition training data into disjoint subsets)

- Recursively grown until error or threshold is reached.

Classification trees:

Determines the class tha an object belongs to:

The error measure is determined by the impurity function E(t), where t is the node

Impurity function for J-Class problems

$$E(t) = \phi(p_1, p_2 .. p_j)$$

Best impurity function for J-class classification trees are entropy function and Gini Deversity index.

Entropy function: $= \phi_e(p_1, .. p_j) = -\sum_{j=1} p_j l_n p_j$

Gini index: $\phi_g(p_1, .. p_j) = -\sum_{i \neq j} p_i p_j = 1 - \sum_{j=1}^{j} p_j^2$

Impurtiy change due to splitting

$\Delta E(s,t) = E(t) - p_l E(t_l) - p_r E(t_r)$

Where p_l, p_r: percentage of impurities on the left and right and s is the split.

Regression Trees

$$E(t) - \min \sum_{i=1}^{N(E)} (y_i - d_t(x_i, \theta)^2$$

Where $\{x_i, y_i\}$ are data points. $d_t(x, \theta)$: local model, and θ is the modifiable parameter.

$\Delta E(s,t) = E(t) - E(t_l) - E(t_r)$

Tree pruning:

If the Decision Tree is too large then it is needed to be pruned.

Tree pruning methods:

a. Minimum cost complexity

b. Weakness subtree shrinking.

Calculation of Minimum cost complexity

$$E_\alpha(t) = E(T) + \alpha|T|$$

Where α is the complexity parameter

$$E_\alpha(T_\alpha) = \min_{T \subset T_{Max}} E_\alpha(T)$$

CART algorithm features:

- Conceptually simple
- Computaionaly efficient
- Applicatble to classification and regression problems
- Solid statistical foundation is available
- Suitable for high dimensional data
- Able to identify relevant inputs simultaneously.

3.4 Data Clustering algorithms

- Hard C means (K-means exclusive clustering)
- Fuzzy c means(Overlapping clustering)
- Mountain clustering
- Subtractive clustering

Distance measure: Euclidean distance is used to group similar instances

K-Means clustering

- It is also called C-means clustering.
- Used in image and speech processing

- RBF(Radial Basis function): value based on distance from source
- Let x_j(j=1..n) be the input vector and it is clustered into C groups.
- Let C_i be the cluster center and G_i be the centre in each group(for i=1..n)
- The cost function J is calculated as $\sum_{i=1}^{c} J_i = \sum_{i=1}^{c}(\sum_{k,x_{k\in G_i}} \|x_k - c_i\|^2)$

$$= \sum_{i=1}^{c}(\sum_{k,x_{k\in G_i}} d(x_k - c_i))$$

- The partitioned groups expressed with binary membership matrix as follows:

$$u_{ij} = 1 \; if \; \left\|x_j - c_i\right\|^2 \leq \left\|x_j - c_k\right\|^2 \; for \; k \neq i$$

0 otherwise

K-Means Clustering Algorithm:

Step 1: Initialize cluster centre c_i(random)

Step 2: Calculate membership matrix u_{ij}

Step 3: Compute the cost function $\sum_{i=1}^{c} J_i = \sum_{i=1}^{c}(\sum_{k,x_{k\in G_i}} \|x_k - c_i\|^2)$

Step 4: Update the cluster $c_i = \frac{1}{G_i} \sum_{k,x_{k\in G_i}} x_k$, where c_i is the optimal cluster centre.

$$c_i = \eta(x - c_i)$$

Fuzzy C-Means clustering algorithm

Step 1: Initialize the membership matrix U with random numbers from(0..1)

Step 2: Calculate fuzzy cluster centre

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^m x_j}{\sum_{j=1}^{n} u_{ij}^m}$$

Step 3: Compute the cost function

$J(U, c_1..c_c) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^m d_{ij}^2$, where $d_{ij} = \left\|c_i - x_j\right\|^2$

Step 4: Compute new $U_{ij} = \frac{1}{\sum_{k=1}^{c} \frac{d_{ij}}{d_{kj}}^{\frac{2}{m-1}}}$, where d_{ij} is the distance between the i^{th} custer and j^{th} data point, where m is the weighting exponent for all $m \in [1, \infty]$

Mountain Clustering:

- Proposed by Yager and Filer

- The cluster centre is estimated by density measure called mountain function
- It follows quick approximate clustering

Mountain clustering algorithm:

Step 1: Form a grid on data space. The grid lines constitute candidates for cluster centres

Step 2: Construct a mountain function representing data density measure

$m(v) = \sum_{j=1}^{N} \exp(-\frac{\|v-x_i\|^2}{2\sigma^2})$, where x_i is the i^{th} data point and v∈V = xi(i^{th} data)

Step 3: Select cluster centre by sequentially destructing the mountain function

$m_{new}(v) = m(v) - m(c_1)\exp(-\frac{\|v-c_1\|^2}{2\beta^2})$, where m($c_1$)is inversely proportional to the distance between v and c_1

Subtractive clustering algorithm:

- Proposed by Chin
- Data points are used for cluster instead of grids

  Step 1:Calculate Density measure D_i

  $D_i = \sum_{j=1}^{N} \exp(-\frac{\|x_i-x_j\|^2}{\left(\frac{ra}{2}\right)^2})$, ra is a positive constant. Data point with high D_i is selected as the first cluster centre.

  Step 2: Update D_i

$$D_i = D_i - D_i \exp\left(-\frac{\|x_i - x_j\|^2}{\left(\frac{rb}{2}\right)^2}\right)$$

3.5 Rule Base structure identification

1. Neuro Fuzzy modelling 2 phases
   i. Structure identification: Simple grid partitioning
   ii. Parameter identification: Objective functions a. density measure
   b.Typicality measure

Input Selection or Feature Selection

- Apply weight of importance
- Let σ be the importance measure , with σ∈ [0,1]

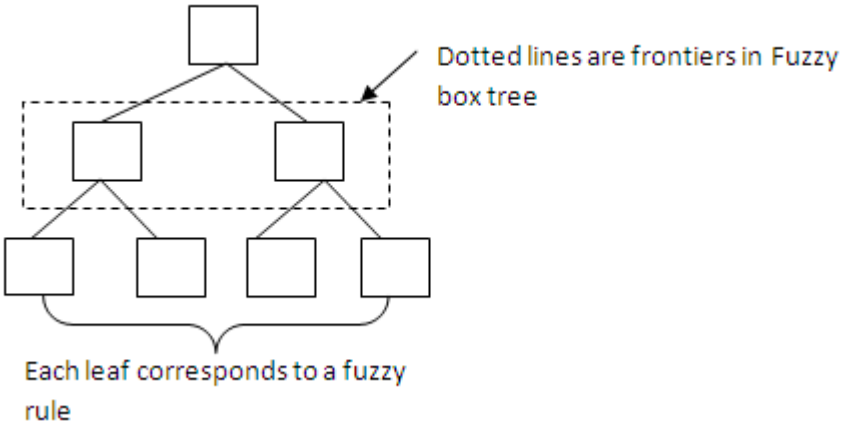
$$s = (1 - \sigma) * [1 - \mu_A(x)]$$

2. Input space partitioning
3. Rulebase organization

   A new data structure called fuzzy box tree is introduced to organize rules in logarithmic time.

Steps in Fuzzy-box tree mechanism

1. Divide and conquer data strucutures is applied to fine tune large amount of small rules
2. Define the fuzzy box tree on antecedents of rule
3. Introduce branch and bound algorithm for pattern matching algorithm
4. Provide parallel algorithm to maintain advantage of parallel processing in FIS



A binary fuzzy box tree 'T' is a rooted tree in which each node has two children. Let R be the nodes set of 'T' and r ∈ R is a fuzzy set with $\mu_R(u)$. If s is child of r then $\mu_s(u) \leq \mu_r(u)$, ∀ u∈ U, s ≤ r.

Fuzzy set based rule combination

Method of Skeletonization or rule base compression for ANFIS.

A focus set or focus window is a fuzzy set defined on feature space that indicates focus or the current interest.

$$G(r) = S(r_1, W) + (r_2, W) - (r, W)$$

Where r_1,r_2: children of r and S is the similarity measure.

Linear efficiency algorithm or Algorithm to find the best rule base with respect to the current context

1. F ←{Root}, Calculate the similarity gaint G(r)for root
2. While |F|<n,

   Do

   Select integer node r ∈F with largest G(r)

   Expand r to get cildern L(r)

   Calculate G(r) for the nodes L(r)

   F←{F-r}∪ L(r).

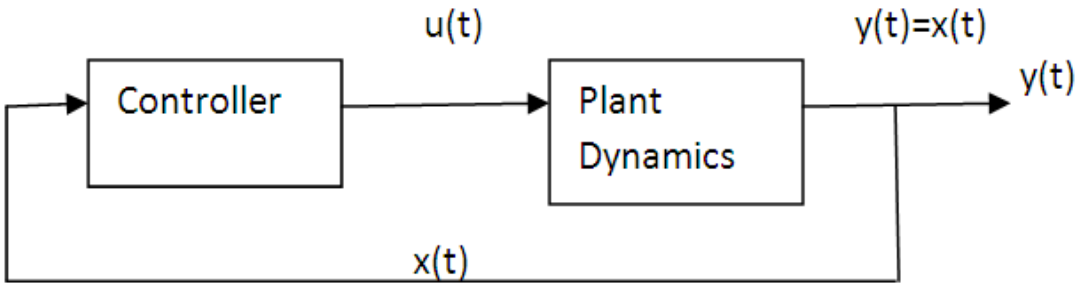
3.6 Neuro Fuzzy control – I

FLC: Fuzzy Logic Controller invented by Zadeh

Combines Back Propagation Neural networks and Fuzzy Inference system

Feedback control systems & Neuro Fuzzy Control- an overview

Block Diagram of continuous- time feedback control system



(Plant =process) represents dynamic system to be controlled

Controller= Employs control strategy to achieve a control goal

X(t)=state variables

Y(t)= Output state variable

1. State Equation for non linear time invariant

   $\dot{x}(t) = f(x(t), u(t))$: **plant dynamics**

   $u(t)$: controllers output at time t

   $x_d(t)$ = desired output signal

If $x_d(t)$ is constant, then control problem is called **regulator problem**

2. Linear feedback control system, plant and controller are reformulated as

   $\dot{x}(t) = Ax(t) + Bu(t)$: plant dynamics

   $u(t) = kx(t)$: linear controller

3. Control system without feedback loops are called open-loop control system(lack of real time automated problems)

   $\Phi(i) = mapping\ function$: mapping the actual output x(t) to control action u

$$u(t) = \Phi x(t)$$

Block Diagram of Discrete time domain feedback control system

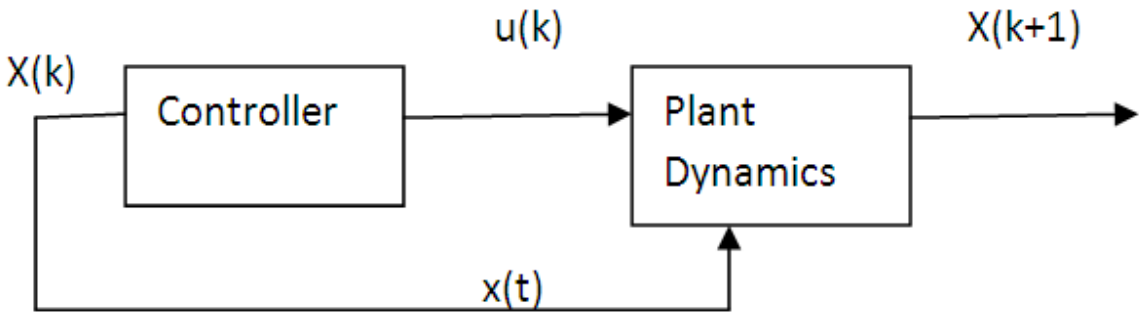


Fig 2. Diagram of Discrete time domain feedback control system

X(k): state vector at time k

U(k): control action

Input to plant= x(k) + u(k) (Previous output+ control action)

$x(k+1) = f(x(k), u(k))$: plant dynamics $u(t) = gx(t)$: linear controller

Neuro Fuzzy control(NFC)

- If controller blocks in the Fig 2 are replaced with Neural networks or FIS(Fuzzy Inference System) then it is Neuro or Fuzzy control system.
- Neuro fuzzy control refers to design methods for fuzzy logic control
- More NFCs are non-linear and difficult to train and requires expert control i.e mimicking an expert. For example complex plants likes electric train, traffic signal etc need knowledge acquisition and requires human inputs.

NFC has unique **properties** of ANFIS(Adaptive Neuro Fuzzy Infrence System) controller

1. Learning ability
2. Parallel operation
3. Structured knowledge representation

4. Better integration with other control design methods.

Types of inputs given from human for knowledge acquisition

a. Linguistic information: Through human interviews and it is based on trial and error methodology.
b. Numerical information: To record sensor data by human

Example expert systems

- Steam Engine and boiler control
- Container ship crane control
- Elevator control
- Nuclear reactor control
- Aircraft control

Inverse Learning or General Learning

Two phases:

a. Learning phase
b. Application phase: generate control actions

State at time K+1 is x(k+1)

$x(k+1) = f\big(x(k), u(k)\big)$ where x(k) is the state at time k and u(k) is the control signal at time k

Stat at K+2

$$x(k+2) = f\big(x(k+1), u(k+1)\big)$$

Generalizing:

$$x(k+n) = F(x(k), U)$$

Where n: order of plant

F: Multiple composite function of f

U: control actions from k to k+n-1

$$U = (u(k), u(k+1) \ldots . u(k+n-1))^t$$

Inverse dynamics of plant U

$$U = G(x(k), x(k+n))$$

Where G is the inverse mapping function

Inverse dynamics of a linear system

$x(k+1) = Ax(k) + Bu(k)$

Where A, B are n*n and n*1 matrices respectively

$x(k+n) = A^n x(k) + WU$

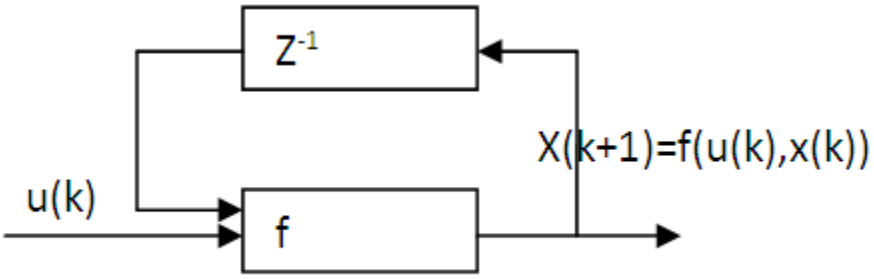
$$W = [A^{n-1}B \ldots . AB.B]$$

Where W is the controlability matrix. If w is non singular then

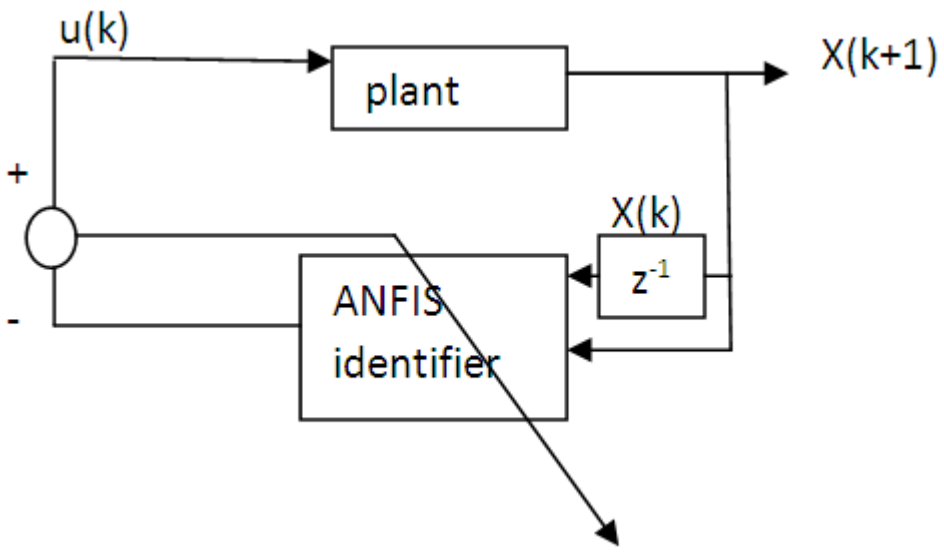
$$U = W^{-1}[x(k+n) - A^n x(k)]$$

Block diagram fro inverse learning method

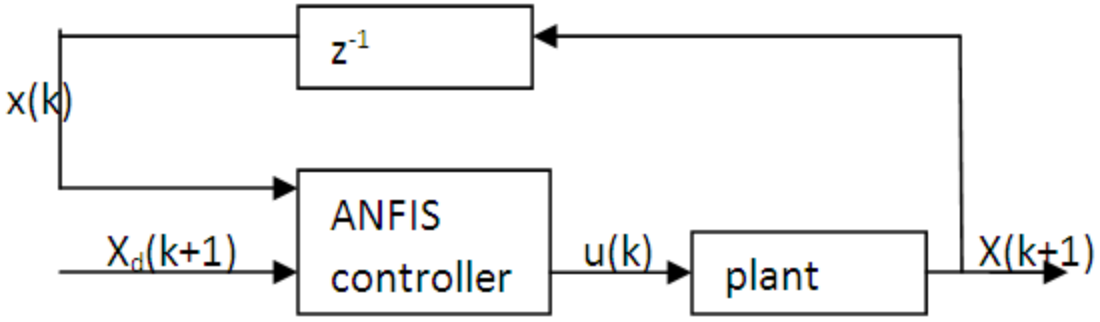
a. Plant block



b. Training phase



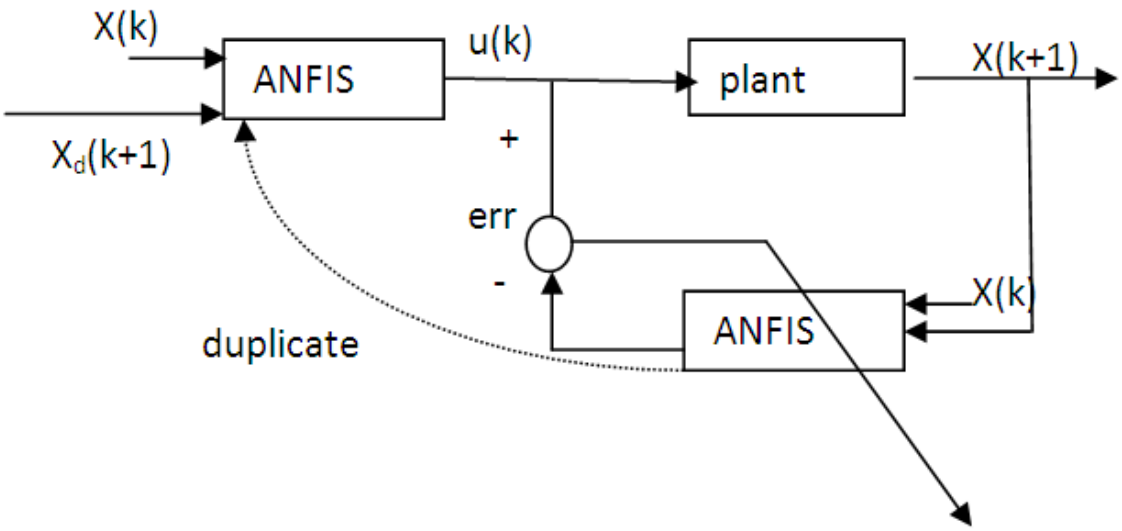
c. Application phase



Let 'G' be Input output mapping of inverse Dynamics. Let$\widehat{U}$ be the estimated output

$\widehat{U} = \hat{G}(x(k), x_d(k+n)$

Block diagram of online inverse learning



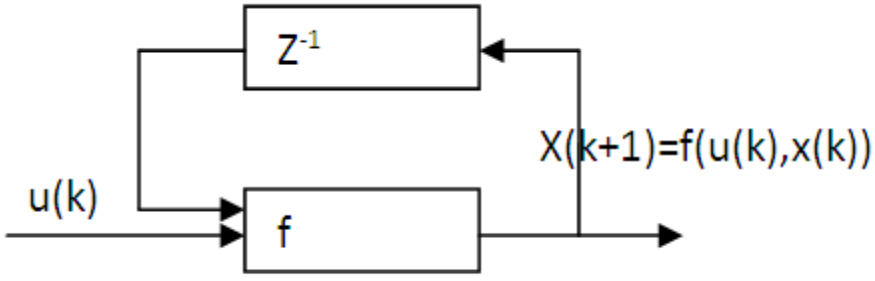
Drawback:

- Network error ($\left\|U-\widehat{U}\right\|^2$) is minimized but not the system error ($\|x_d(k)-xk\|^2$)
- It is an indirect approach

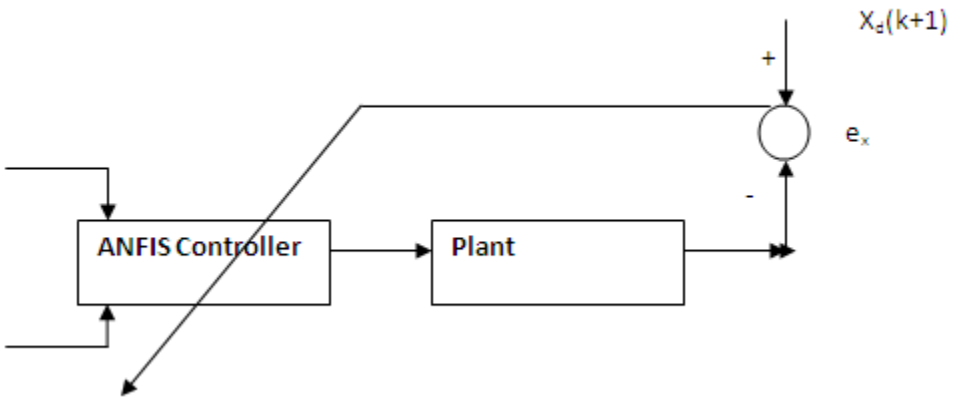
To overcome the drawback and to minimize the system error **specialized learning** is preferred.

<u>Specialized Learning</u>

a. Plant block



b. Specialzed learning using desired trajectory



Let the plant dynamics be $x(k+1) = f(x(k), v(k))$

Anfis output $\hat{v}(k) = F(x(k), u(k), \theta)$

θ= param vector

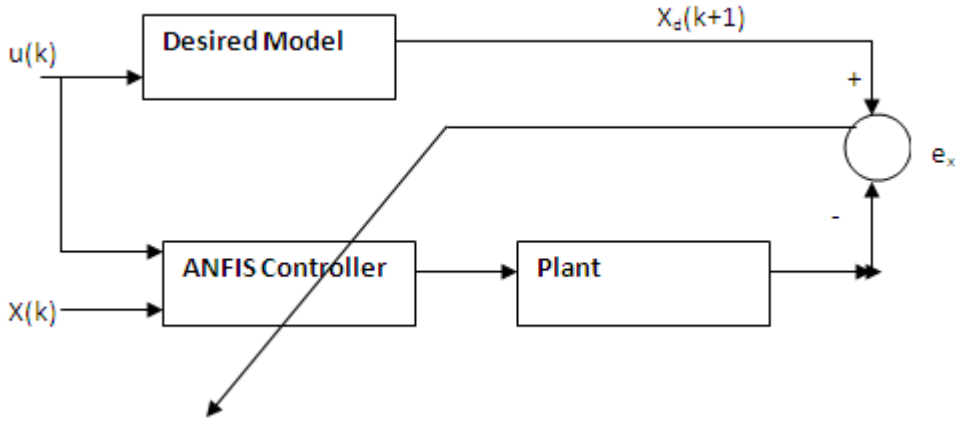
If $\hat{v}(k)$is set as plant's input $v(k)$, then we have a closed –loop system specification by

$$x(k+1) = f(x(k), F(x(k), u(k), \theta))$$

Aim : To minimize the difference between closed loop systems and the desired model.

Error measure: $J(\theta) = \sum_k \| f(x(k), F((x(k), u(k), \theta)) - x_d(k+1) \|^2$

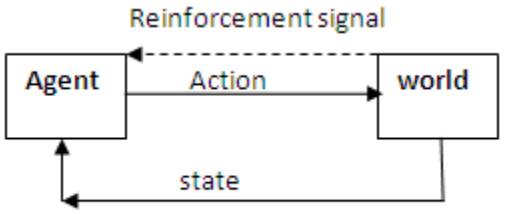
For Back propagation of error signals, Jacobian matrix need to be found.



Specialized learning with model referencing

3.7 Neuro Fuzzy control II

Reiforcement Learning



An interactive Learning Agent

NeuroFuzzy reinforcement controllers

Representation of Neuro-Fuzzy reinforcement learning models

Realistic idea of Adaptive Heuristic Critic(AHC) models

- GARIC: Generalized Approximated Reasoning for Intelligent Control
- RNN-FLCS: Reinforcement NN based Fuzzy logic control systems.
- AHCON: Lin's AHC connectionist model

GARIC

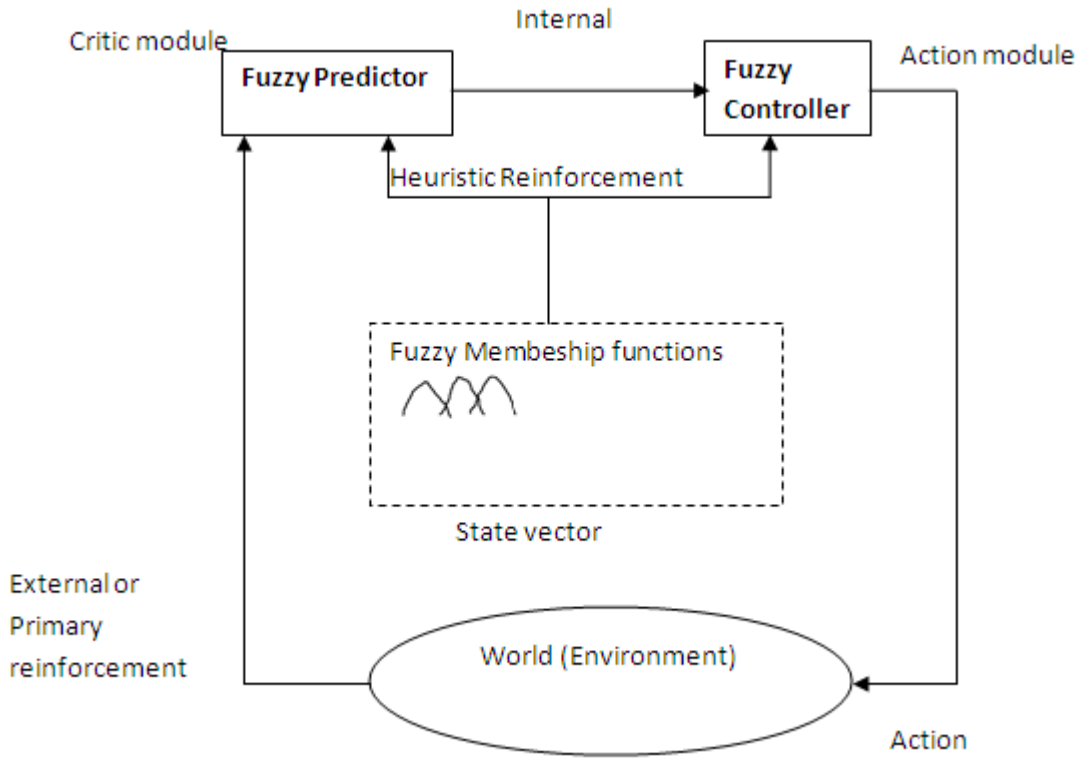
Three components:

1. Action selection network(ASN)
2. Action Evaluation Network(AEN)
3. Stochastic Action Modifier(SAM)

Comparison of the AHC models

AHC models	Critic module	Action module
AHCON	Neuro	Neuro
GARIC	Neuro	Neuro-Fuzzy
RNN-FLCS	Neuro-Fuzzy	Neuro-Fuzzy

A Neuro-Fuzzy AHC model:



<u>3.8 Fuzzy Decision Making</u> for modelling and control

Decision making and control are two fields with distinct methods for solving problems, and yet they are closely related. This book bridges the gap between decision making and control in the field of fuzzy decisions and fuzzy control, and discusses various ways in which fuzzy decision making methods can be applied to systems modeling and control.

Fuzzy decision making is a powerful paradigm for dealing with human expert knowledge when one is designing fuzzy model-based controllers. The combination of fuzzy decision making and

fuzzy control in this book can lead to novel control schemes that improve the existing controllers in various ways. The following applications of fuzzy decision making methods for designing control systems are considered:

• *Fuzzy decision making for enhancing fuzzy modeling*. The values of important parameters in fuzzy modeling algorithms are selected by using fuzzy decision making.

• *Fuzzy decision making for designing signal-based fuzzy controllers*. The controller mappings and the defuzzification steps can be obtained by decision making methods.

• *Fuzzy design and performance specifications in model-based control*. Fuzzy constraints and fuzzy goals are used.

• *Design of model-based controllers combined with fuzzy decision modules*. Human operator experience is incorporated for the performance specification in model-based control.

Making Fuzzy Decisions

Most decisions that people make are logical decisions, they look at the situation and make a decision based on the situation. The generalized form of such a decision is called a generalized modus ponens, which is in the form:

If P, then Q.
P.
Therefore, Q.

This form of logical reasoning is fairly strict, Q can only be if P. Fuzzy logic loosens this strictness by saying that Q can mostly be if P is mostly or:

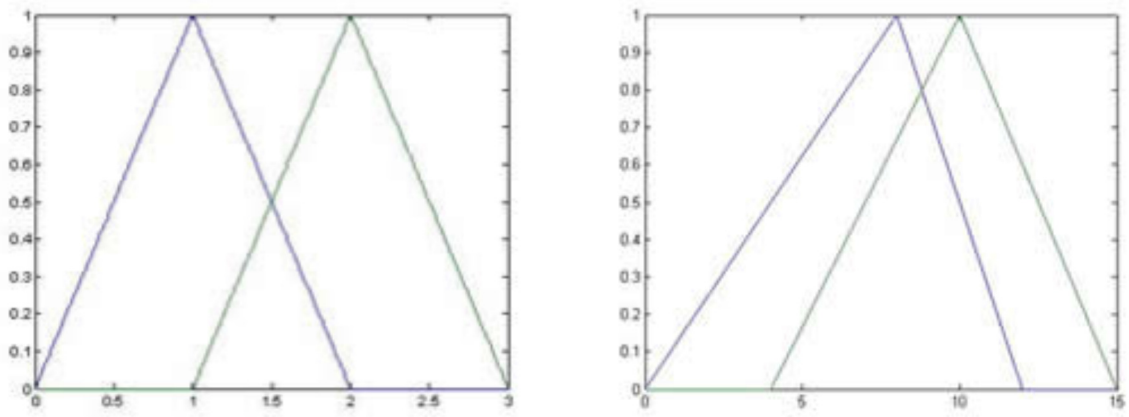
If P, then Q.
***mostly* P.**
Therefore, *mostly* Q.

Where P and Q are now fuzzy numbers. The reasoning above requires a set of rules to be defined. These rules are linguistic rules to relate different fuzzy sets and numbers. The general form of these rules are: "if *x* is A then y is B," where *x* and *y* are fuzzy numbers in the fuzzy sets A and B respectivly. These fuzzy sets are defined by membership functions. There can be any number of input and output membership functions for the same input as well, depending on the number of rules in the system. For example, a system could have membership functions that represent slow, medium, and fast as inputs.

The linguistic rules are used to define the relation between the input and the output, but how exactly are the output fuzzy values determined? There are several ways to determine the answer based on the inputs, mainly the Mamdani, Larsen, Takagi-Sugeno-Kang, and Tsukamoto inference and aggregation methods. Firstly, we must describe the basic general set of rules, they will bet a set of rules that have one input in a fuzzy set and one output in a fuzzy set:

If *x* is **A*i*** then *y* is **B*i***, *i*=1,2,...*n*

Let us look at a system that has two input membership functions (A1,A2) and two output membership functions (B1, B2). These membership functions, shown below, define the fuzzy sets A and B in the above general inference rule.



A1 and A2 are shown on the left, with A1 in blue and A2 in green. On the right B1 is blue and B2 is green. We will be using the Mamdani inference model to combine the sets and rules. The Mamdani inference model is:

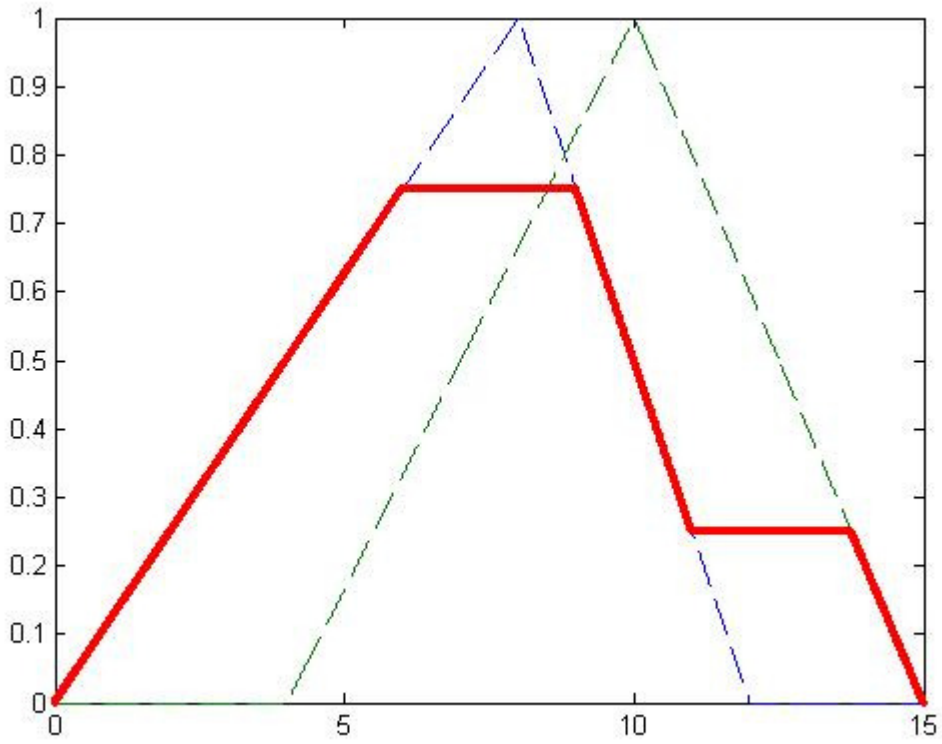
R(x,y) = pg110 in Nguyen

Using this model will give an aggregate fuzzy set, R, that uses the input values in A1 and A2 to modify and combine B1 and B2. The input membership functions, as well as the output membership functions, are overlapping; this means that an input value can have membership in both membership functions, or in only one. If the input value has membership in a function, than any rule using that membership funciton is said to 'fire' and produce a result. These results are then aggregated using the Mamdani model, or a different model.

Let us then pick and input value that has membership function in A1 and A2, 1.25, this will cause both rules to fire. The value 1.25 has a membership of 0.75 in A1 and a membership of 0.25 in A2. Using the Mamdani model and these inputs the resulting aggregate output will be:

$[A1(1.25) \wedge B1(y)] \vee [A2(1.25) \wedge B2(y)]$

When all of these combinations have been made, the aggregate output membership function (red), as well as B1 and B2 (dashed) are shown below:



This aggregate fuzzy membership function is the result of the rule based inference decision making process. To get a finite number as an output we need to go through the defuzzification process. Defuzzification is a method that produces a number that best represents, and consistenly represents the fuzzy set. There are many ways to do this with most of them being some type of averaging method. The most common is the centroid method, this calculates the center of area of the fuzzy set and uses the value at which this occurs as the defuzzified output. Other methods include the bisector, largest of maximum, smallest of maximum, and middle of maximum. For the above aggregate fuzzy set, the different defuzzification methods produce these finite values shown below. So, if the most common method, centroid, is used, the finite result would be 7.319.

Defuzz Method	**Result**
Centroid	7.319
Bisector	7.230
Largets of Max	9

Smallest of Max	6
Middle of Max	7.5

References:

- Joao M C Sousa (*Technical University of Lisbon, Portugal*), Uzay Kaymak (*Erasmus University Rotterdam, The Netherlands*)," Fuzzy Decision Making in Modeling and Control", World Scientific Series in Robotics and Intelligent Systems: Volume 27
- https://www.calvin.edu/~pribeiro/othrlnks/Fuzzy/fuzzydecisions.htm
- Jang, sun, Mitzutani,"Neuro Fuzzy and Soft computing", Prentice Hall India, 2006

Unit 4

Genetic Algorithms

Introduction- Implementation of GA- Reproduction- Cross over- Mutation- Coding- Fitness Scaling- Applications of GA

4.1 Introduction

- **Genetic algorithm (GA)** is a search heuristic that mimics the process of natural selection
- Developed: USA in the 1970's
- **Special Features:**
  - Traditionally emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

4.2 Implementation of GA techniques

- Representations
- Mutations
- Crossovers
- Selection mechanisms

Representation	**Binary strings**
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

Characteristics of GA

- Individuals selected randomly
- Fitness of every individual in the population is evaluated(Fitness= value of objective fn)
- More fit individuals are stochastically selected from the current population
- Each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation
- Iteration continues still satisfactory fitness level has been reached for the population.

Key terms

- **Individual** - Any possible solution
- **Population** - Group of all *individuals*
- **Search Space** - All possible solutions to the problem
- **Chromosome** - Blueprint for an *individual*
- **Trait** - Possible aspect (*features)* of an *individual*
- **Allele** - Possible settings of trait (black, blond, etc.)
- **Locus** - The position of a *gene* on the *chromosome*
- **Genome** - Collection of all *chromosomes* for an *individual*

4.3 Reproduction

- Reproduction (or selection) is an operator that makes more copies of better strings in a new population
- first operator applied on population
- Selects good strings
- sometimes known as the selection operator

Selection

- string is selected with a probability proportional to its fitness
- Thus, the ith string in the population is selected with a probability proportional to F_i.
- the probability for selecting the i^{th}string is

$$p_i = \frac{F_i}{\sum_{i=1}^{n} F_i}$$

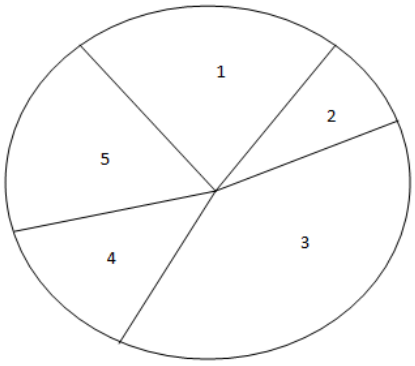
where n is population size .

Roultte wheel selection

- The roulette-wheel is spun n times.
- each time selecting an instance of the string chosen by the roulette-wheel pointer
- this roulette-wheel mechanism is expected to make copies of the ith string in the mating pool

The average fitness of the population is calculated as follows:

$Average\ Fitness = \frac{F_i}{\bar{F}}$, where $\bar{F} = \sum_{i=1}^{n} F_i$



Population	Fitness
1	25.0
2	5.0
3	40.0
4	10.0
5	20.0

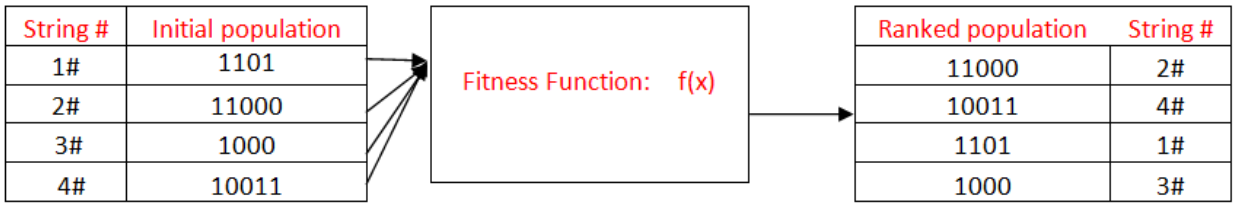
Fig: Roulette Wheel Selection

The roulette wheel selection

Roulette Wheel's Selection Pseudo Code:

```
for all members of population
    sum += fitness of this individual
end for
for all members of population
    probability = sum of probabilities + (fitness / sum)
    sum of probabilities += probability
end for
loop until new population is full
    do this twice
        number = Random between 0 and 1
        for all members of population
            if number > probability but less than next probability
then          you have been selected
        end for
    end
    create offspring
end loop
```

Fitness Function



4.4 Crossover

- A crossover operator is used to recombine two strings to get a better string.
- In reproduction, good strings in a population are probabilistic-ally assigned a larger number of copies

The two strings participating in the crossover operation are known as **parent strings** and the resulting strings are known as **children strings**

Cross over

- A crossover operator is mainly responsible for the search of new strings
- Types of cross over
- 1-pt cross over
- 2-pt cross over
- N-pt cross over

1-point cross over with cross over point: 3

String 1#	101|11
String 2#	100|01

Before Crossover

String 1#	101|01
String 2#	100|11

After cross over

2- point crossover with cross over points: 3 and 6

String 1#	101|101|000
String 2#	100|010|011

Before Crossover

String 1#	101|010|000
String 2#	100|101|011

After cross over

4.5 Mutation

- Mutation adds new information in a random way to the genetic search
- helps to avoid getting trapped at local optima
- It is a process of randomly disturbing genetic information
- operate at the bit level
- there is probability that each bit may become mutated p_m

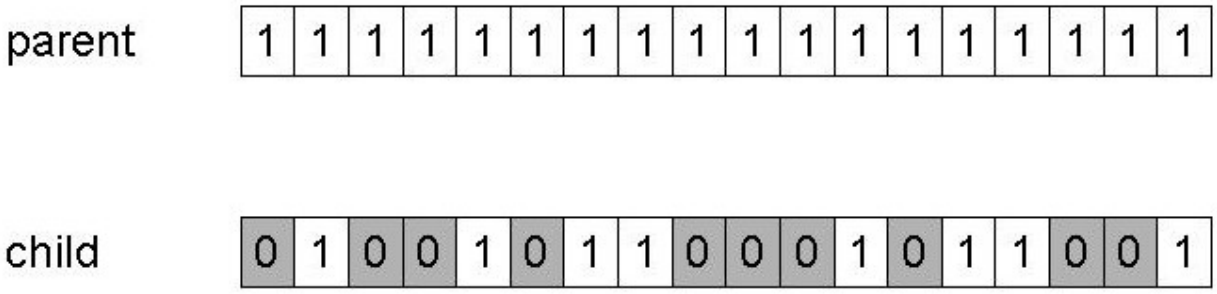
Mutation Example [Eiben and Smith]

The following population having four eight bit string

Alter each gene independently with a probability p_m

p_m is called the mutation rate

Typically between 1/pop_size and 1/chromosome_length



Example for function x^2 : selection

Fitness: $f_i = f(x) = x^3 + 1$

$Prob_i = (f_i / sum(f_i))$

Expected count $= f_i / average(f_i)$

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

4.6 Coding the GA:

Various softwares are available for coding Genetic Algorithms. Some of the available softwares are listed as follows:

- JGAP

- jMetal
- Jenetics: Java Genetic Algorithm Library
- Java Graticule 3D

4.7 Fitness Scaling

- **Linear scaling** : The fitness of each individual in the population is scaled such that the scaled fitness is linearly related to the unscaled fitness

$$f' = af + b$$

- f' is the scaled fitness value,
- f is the actual fitness value
- a' and 'b' are linear co-efficients
- Relationship between the maximum fitness individual in the population and the average population fitness

$$f'max = f_{avg} \times C_s$$

$$f'_{avg} = f_{avg}$$

- f'_{max} is the scaled maximum fitness
- f_{avg} is the average fitness of the population
- C_s and is a scaling constant (specifies the expected number of copies of the best individual in the next generation).

4.8 Applications of GA

i. Optimization

ii. Automatic Programming

iii. Machine and robot learning

iv. Economic models

v. Immune system models

vi. Ecological models

vii. Population genetics models

viii. Models of social systems

i. Optimization

- numerical optimization
- combinatorial optimization
- Example problems:
  - traveling salesman problem (TSP)
  - circuit design [Louis 1993]
  - job shop scheduling [Goldstein 1991]
  - video & sound quality optimization.

ii. Automatic Programming

- To design computational structures
- For example
  - cellular automata (A cellular automaton consists of a regular grid of *cells*, each in one of a finite number of *states*, such as *on* and *off*)
  - sorting networks

iii. Machine and robot learning

- Classification and prediction
  - Example protein structure prediction
- To design neural networks
- To evolve rules for learning classifier systems or symbolic production systems

- To design and control robots.

iv. Economic models

- To model processes of innovation
- Development of bidding strategies
- Emergence of economic markets

v. Immune system models

- To model various aspects of the natural immune system
  - Somatic mutation during an individual's lifetime
  - Discovery of multi-gene families during evolutionary time.

vi. Ecological models

- biological arms races(an evolutionary struggle between competing sets of co-evolving genes that develop adaptations and counter-adaptations against each other)
- host-parasite co-evolutions
- symbiosis (individual interactions)
- resource flow in ecologies.

vii. Population genetics models

- To study questions in population genetics, such as "under what conditions will a gene for recombination be evolutionarily viable?"
- GAs have been used to study how individual learning and species evolution affect one another.

viii. Models of social systems:

- Evolution of cooperation [Chughtai 1995]
- Evolution of communication
- Trail-following behavior in ants.

References:

1. David E. Goldberg, "In Search optimization and machine learning",Pearson Education, 2005
2. Eiben. A.E and Smith J.E, "Introduction to Evolutionary Computing, Genetic Algorithms", www.cs.vu.nl/~gusz/ecbook/slides/Genetic_Algorithms.ppt
3. A Genetic Algorithm Example: http://cse.unl.edu/~sscott/research/html/hga_conf/node3.shtml
4. http://geneticprogramming.com/ga/GAsoftware.html

UNIT:V

ARTIFICIAL INTELLIGENCE

Introduction – Searching techniques – First order Logic – Forward reasoning – Backward reasoning – Semantic – Frames.

5.1 Introduction

What is artificial intelligence?

Artificial Intelligence is the branch of computer science concerned withmaking computers behave like humans.

JohnMcCarthy, who coined the term in 1956, defines it as "the science and engineering ofmaking intelligent machines, especially intelligent computer programs." The definition of AI is categorized into four approaches which is shown in the table below.

Systems that think like humans"The exciting new effort to make computers think as if machines with minds, in the full and literal sense."(Haugeland,1985)	**Systems that think rationally** "The study of mental faculties through the use of computer models." (Charniak and McDermont,1985)
Systems that act like humans The art of creating machines that performfunctionsthat require intelligence when performed by people."(Kurzweil,1990)	**Systems that act rationally** "Computational intelligence is the studyofthedesignofintelligentagents."(Poole et al.,1998)

The four approaches in more detail are as follows :

(a) Acting humanly : The Turing Test Approach

o Test proposed by Alan Turing in 1950

o The computer is asked questions by a human interrogator.

The computer passes the test if a human interrogator, after posing some writtenquestions, cannot tell whether the written responses come from a person or not.Programming a computer to pass , the computer need to possess the followingcapabilities :

- Natural language processing to enable it to communicate successfully in English.
- Knowledge representation to store what it knows or hears.
- Automated reasoning to use the stored information to answer questions and to draw

new conclusions.

- Machine learning to adapt to new circumstances and to detect andextrapolate patterns

- To pass the complete Turing Test, the computer will need
- Computer vision to perceive the objects, and
- Robotics to manipulate objects and move about.

Problem: Turing test is not reproducible, constructive, or amenable to mathematicalanalysis.

b) Thinking humanly : The Cognitive Modeling Approach

A machine program is constructed to think like a human but for that we need to get inside actual working of the human mind :

(i) through introspection – trying to capture our own thoughts as they go by;

(ii) through psychological experiments

Eg. Allen Newell and Herbert Simon, who developed GPS, the "GeneralProblem Solver" tried to trace the reasoning steps to traces of human subjects solvingthe same problems.

The interdisciplinary field of cognitive science brings together computermodels from AI and experimental techniques from psychology to try to constructprecise and testable theories of the workings of the human mind. In 1960s CognitiveRevolution: information-processing psychology replaced prevailing orthodoxy ofbehaviorism.

It Requires scientific theories of internal activities of the brain :-

* What level of abstraction? – "Knowledge" or circuits"?

* How to validate? - Requires

1) Predicting and testing behavior of human subjects (top-down)

or 2) Direct identification from neurological data (bottom-up)

(c) **Thinking rationally: The "Laws of Thought Approach"**

It is Normative (or prescriptive) rather than descriptive

The Greek philosopher Aristotle was one of the first to attempt to codify"right Thinking" that is irrefutable reasoning processes. His syllogism providedpatterns for argument structures that always yielded correct conclusions when givencorrect premises—for example, "Socrates is a man; all men are mortal; thereforeSocrates is mortal.". These laws of thought were supposed to govern the operationof the mind; their study initiated a field called logic.

Problems:

1) Not all intelligent behavior is mediated by logical deliberation

2) What is the purpose of thinking? What thoughts should I have out of all thethoughts (logical or

otherwise) that I could have?

(d) Acting rationally : The Rational Agent Approach

Rational behavior: doing the right thing

The right thing: that which is expected to maximize goal achievement, giventhe available information.

Doesn't necessarily involve thinking (e.g.) blinking reflex - thinking should bein the service of rational action

An agent is something that acts. Computer agents are not mere programs ,butthey are expected to have the following attributes also : (a) operating underautonomous control, (b) perceiving their environment, (c) persisting over a prolongedtime period, (e) adapting to change.

A rational agent is one that acts so as to achieve the best outcome.

What can AI do today?

Autonomous planning and scheduling:

Ahundred million miles from Earth, NASA's Remote Agent program becamethe first on-board autonomous planning program to control the scheduling ofoperations for a spacecraft (Jonsson et al., 2000). Remote Agent generated plans fromhigh-level goals specified from the ground, and it monitored the operation of thespacecraft as the plans were executed-detecting, diagnosing, and recovering fromproblems as they occurred.

Game playing:

IBM's Deep Blue became the first computer program to defeat the worldchampion in a chess match when it bested Garry Kasparov by a score of 3.5 to 2.5 inan exhibition match (Goodman and Keene, 1997).

Diagnosis:

Medical diagnosis programs based on probabilistic analysis have been able toperform at the level of an expert physician in several areas of medicine.

Robotics:

Many surgeons now use robot assistants in microsurgery. HipNav (DiGioia etal., 1996) is a system that uses computer vision techniques to create a three-dimensional model of a patient's internal anatomy and then uses robotic control toguide the insertion of a hip replacement prosthesis.

Language understanding and problem solving:

PROVERB (Littman et al., 1999) is a computer program that solves crosswordpuzzles better than most humans, using constraints on possible word fillers, a largedatabase of past puzzles, and a variety of information sources including dictionariesand online databases such as a list of movies and the actors that appear in them.

5.2 Searching Techniques

Problem and Solution

A problem can be formally defined by four components:

1) The **initial state** that the agent starts in .

2) A **Successor Function** returns the possible actions available to the agent.Given a state x, SUCCESSOR-FN(x) returns a set of {action, successor}ordered pairs where each action is one of the legal actions in state x, and eachsuccessor is a state that can be reached from x by applying the action.

i)**State Space**: The set of all states reachable from the initial state. The statespace forms a graph in which the nodes are states and the arcs between nodes areactions.

ii) A path in the state space is a sequence of states connected by a sequence ofactions.

3) **Goal Test** : It determines whether the given state is a goal state. Sometimesthere is an explicit set of possible goal states, and the test simply checks whether thegiven state is one of them. For example, in chess, the goal is to reach a state called"checkmate," where the opponent's king is under attack and can't escape.

4**) Path Cost** : A path cost function assigns numeric cost to each action.

(a) The step cost of taking action a to go from state x to state y is denoted byc(x,a,y). It is assumed thatthe step costs are non negative.

(b) A solution to the problem is a path from the initial state to a goal state.

(c) An optimal solution has the lowest path cost among all solutions.

Example

The 8-puzzle

An 8-puzzle consists of a 3x3 board with eight numbered tiles and a blank space. Atile adjacent to the blank space can slide into the space. The object is to reach thegoal state ,as shown in figure 5.1

Example: The 8-puzzle

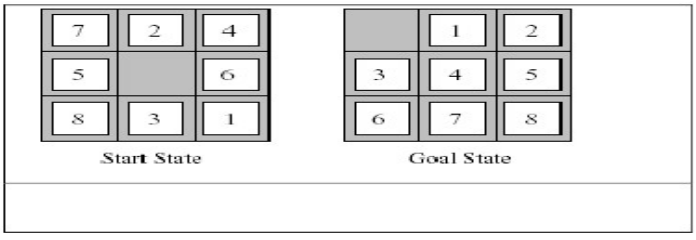


Figure 5.1 A typical instance of 8-puzzle.

The problem formulation is as follows :

o States : A state description specifies the location of each of the eight tiles andthe blank in one of the nine squares.

o **Initial state** : Any state can be designated as the initial state. It can be notedthat any given goal can

be reached from exactly half of the possible initialstates.

o **Successor function** : This generates the legal states that result from trying thefour actions(blank moves Left, Right, Up or down).

o **Goal Test** : This checks whether the state matches the goal configuration.

o **Path cost** : Each step costs 1,so the path cost is the number of steps in the path.

The 8-puzzle belongs to the family of sliding-block puzzles, which are oftenused as test problems for new search algorithms in AI. This general class isknown as NP-complete. The 8-puzzle has 9!/2 = 181,440 reachable states and is easily solved.

The 15 puzzle (4 x 4 board) has around 1.3 trillion states, an the randominstances can be solved optimally in few milli seconds by the best searchalgorithms.

The 24-puzzle (on a 5 x 5 board) has around 1025 states ,and random instancesare still quite difficult to solve optimally with current machines and algorithms.

Real-World Problems

Airline Travel Problem

The airline travel problem is specifies as follows :

o States: Each is represented by a location(e.g., an airport) and the currenttime.

o **Initial state**: This is specified by the problem.

o **Successor function**: This returns the states resulting from taking anyscheduled flight (further specified by seat class and location),leaving laterthan the current time plus the within-airport transit time, from the currentairport to another.

o **Goal Test** : Are we at the destination by some pre specified time?

o **Path cost**: This depends upon the monetary cost, waiting time, flight time,customs and immigration procedures, seat quality, time of dat, type of airplane, frequent-flyer mileage awards, and so on.

5.2 Searching Techniques

a. Breadth-first search

Breadth-First search(BFS) is like traversing a tree where each node is a state which may be a potential candidate for solution. It expands nodes from the root of the tree and then generates one level of the tree at a time until a solution is found. It is very easily implemented by maintaining a queue of nodes. Initially the queue contains just the root. In each iteration, node at the head of the queue is removed and then expanded. The generated child nodes are then added to the tail of the queue.

Breadth-first search is useful when

- space is not a problem.

- you want to find the solution containing the fewest arcs.
- few solutions may exist, and at least one has a short path length.
- infinite paths may exist, because it explores all of the search space, even with infinite paths.

Algorithm: Breadth-First Search

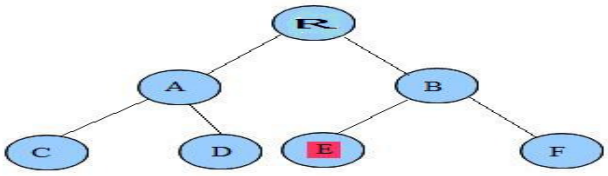
1. Create a variable called NODE-LIST and set it to the initial state.
2. Loop until the goal state is found or NODE-LIST is empty.
    a. Remove the first element, say E, from the NODE-LIST. If NODE-LIST was empty then quit.
    b. For each way that each rule can match the state described in E do:

i) Apply the rule to generate a new state.

ii) If the new state is the goal state, quit and return this state.

iii) Otherwise add this state to the end of NODE-LIST

Since it never generates a node in the tree until all the nodes at shallower levels have been generated, *breadth-first search* always finds a shortest path to a goal. Since each node can be generated in constant time, the amount of time used by Breadth first search is proportional to the number of nodes generated, which is a function of the branching factor b and the solution d. Since the number of nodes at level d is b^d, the total number of nodes generated in the worst case is $b + b^2 + b^3 + \ldots + b^d$ i.e. $O(b^d)$, the asymptotic time complexity of breadth first search.



Breadth First Search

Look at the above tree with nodes starting from root node, R at the first level, A and B at the second level and C, D, E and F at the third level. If we want to search for node E then BFS will search level by level. First it will check if E exists at the root. Then it will check nodes at the second level. Finally it will find E a the third level.

Advantages of Breadth-First Search

1. Breadth first search will never get trapped exploring the useless path forever.
2. If there is a solution, BFS will definitely find it out.

3. If there is more than one solution then BFS can find the minimal one that requires less number of steps.

Disadvantages of Breadth-First Search

1. The main drawback of Breadth first search is its memoryrequirement. Since each level of the tree must be saved in order to generate the next level, and the amount ofmemory is proportional to the number of nodes stored, the space complexity of BFS is $O(b^d)$.
2. If the solution is farther away from the root, breath first search will consume lot of time.

b.Depth First Search(DFS)

Depth First Search (DFS) searches deeper into the problem space. Breadth-first search always generates successor of the deepest unexpanded node. It uses last-in first-out stack for keeping the unexpanded nodes. Depth-first search is implemented recursively, with the recursion stack taking the place of an explicit node stack.

Algorithm: Depth First Search

1.If the initial state is a goal state, quit and return success.
2.Otherwise, loop until success or failure is signaled.
a) Generate a state, say E, and let it be the successor of the initial state. If there is no successor, signal failure.
b) Call Depth-First Search with E as the initial state.
c) If success is returned, signal success. Otherwise continue in this loop.

Advantages of Depth-First Search

- The advantage of depth-first Search is that memoryrequirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.

- The time complexity of a depth-first Search to depth d is O(b^d) since it generates the same set of nodes as breadth-first search, but simply in a different order. The depth-first search is time-limited rather than space-limited.

- If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.

Disadvantages of Depth-First Search

- The disadvantage of Depth-First Search is that there is a possibility that it may go down the left-most path forever. Even a finite graph can generate an infinite tree. One solution to this problem is to impose a cutoff depth on the search. Although the ideal cutoff is the solution depth d and this value is rarely known in advance of actually solving the problem. If the chosen cutoff depth is less than d, the algorithm will fail to find a solution, whereas if the cutoff depth is greater than d, a large price is paid in execution time, and the first solution found may not be an optimal one.

- Depth-First Search is not guaranteed to find the solution.

- And there is no guarantee to find a minimal solution, if more than one solution exists.

c.Bidirectional Search

It searches forward from initial state and backward from goal state till both meet to identify a common state.

The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

d.Uniform Cost Search

Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost.

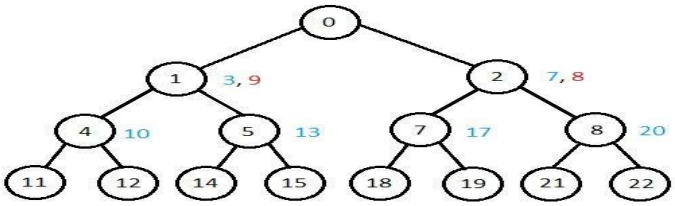
It explores paths in the increasing order of cost.

Disadvantage – There can be multiple long paths with the cost $\leq C^*$. Uniform Cost search must explore them all.

e.Iterative Deepening Depth-First Search

It performs depth-first search to level 1, starts over, executes a complete depth-first search to level 2, and continues in such way till the solution is found.

It never creates a node until all lower nodes are generated. It only saves a stack of nodes. The algorithm ends when it finds a solution at depth *d*. The number of nodes created at depth *d* is b^d and at depth *d-1* is $b^{d-1.}$



Iterative Deepening Depth-First Search

f. Heuristic Search Strategies

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

Heuristic Evaluation Functions

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.

Pure Heuristic Search

It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.

In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

g. A * Search

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

h. Greedy Best First Search

It expands the node that is estimated to be closest to goal. It expands nodes based on f(n) = h(n). It is implemented using priority queue.

Disadvantage – It can get stuck in loops. It is not optimal.

i.Local Search Algorithms

They start from a prospective solution and then move to a neighboring solution. They can return a valid solution even if it is interrupted at any time before they end.

j.Hill-Climbing Search

It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally. If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements.

Disadvantage – This algorithm is neither complete, nor optimal.

k.Local Beam Search

In this algorithm, it holds k number of states at any given time. At the start, these states are generated randomly. The successors of these k states are computed with the help of objective function. If any of these successors is the maximum value of the objective function, then the algorithm stops.

Otherwise the (initial k states and k number of successors of the states = 2k) states are placed in a pool. The pool is then sorted numerically. The highest k states are selected as new initial states. This process continues until a maximum value is reached.

Simulated Annealing

Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the metal retains its newly obtained properties. In simulated annealing process, the temperature is kept variable.

Initially set the temperature high and then allow it to 'cool' slowly as the algorithm proceeds. When the temperature is high, the algorithm is allowed to accept worse solutions with high frequency.

Start

- Initialize k = 0; L = integer number of variables;
- From i → j, search the performance difference Δ.
- If Δ <= 0 then accept else if exp(-Δ/T(k)) > random(0,1) then accept;
- Repeat steps 1 and 2 for L(k) steps.
- k = k + 1;

Repeat steps 1 through 4 till the criteria is met.

End

Travelling Salesman Problem

In this algorithm, the objective is to find a low-cost tour that starts from a city, visits all cities en-route exactly once and ends at the same starting city.

Start

Find out all (n -1)! Possible solutions, where n is the total number of cities.

Determine the minimum cost by finding out the cost of each of these (n -1)! solutions.

Finally, keep the one with the minimum cost.

end

5.3 First Order Logic

First-order logic is symbolized reasoning in which each sentence, or statement, is broken down into a subject and a predicate. The predicate modifies or defines the properties of the subject. In first-order logic, a predicate can only refer to a single subject. First-order logic is also known as first-order predicate calculus or first-order functional calculus.

A sentence in first-order logic is written in the form Px or P(x), where P is the predicate and x is the subject, represented as a variable. Complete sentences are logically combined and manipulated according to the same rules as those used in Boolean algebra.

In first-order logic, a sentence can be structured using the universal quantifier (symbolized $\forall$) or the existential quantifier ($\exists$). Consider a subject that is a variable represented by x. Let A be a predicate "is an apple," F be a predicate "is a fruit," S be a predicate "is sour"', and M be a predicate "is mushy." Then we can say

$\forall x : Ax \Longrightarrow Fx$

which translates to "For all x, if x is an apple, then x is a fruit."

Basic entities in FOL

Propositional logic assumes world contains facts, first-order logic (like natural language) assumes the world contains

• Objects: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries . . .

• Relations: red, round, bogus, prime, multistoried . . ., is the brother of, is bigger than, is inside, is part of, has color, occurred after, owns, comes between, . . .

• Functions: father of, best friend, third inning of, one more than, end of

Syntax of FOLBasic elements

Constant symbols KingJohn, 2, UniversityofMaryland, . . .

Predicate symbols Brother, >, . . .

Function symbols Sqrt, LeftLegOf , . . .

Variable symbols x, y, a, b, . . .

Connectives ∧ ∨ ¬ ⇒ ⇔

Equality =

Quantifiers ∀ ∃

Punctuation ( )

Atomic sentences

Atomic sentence = predicate(term1, . . . , termn) or term1 = term2

Term = function(term1, . . . , termn) or constant or variable

E.g., Brother(KingJohn, RichardT heLionheart) > (Length(LeftLegOf (Richard)), Length(LeftLegOf (KingJohn)))

Complex sentences

Complex sentences are made from atomic sentences using connectives ¬S, S1 ∧ S2, S1 ∨ S2, S1 ⇒ S2, S1 ⇔ S2

E.g., Sibling(KingJohn, Richard) ⇒ Sibling(Richard, KingJohn) >(1, 2) ∨ ≤(1, 2) >(1, 2) ∧ ¬ >(1, 2)

Universal quantification

∀<variables><sentence>

Everyone at the University of Maryland is smart

: ∀ x At(x, UMD) ⇒ Smart(x)

A common mistake to avoid

Common mistake with ∀:

using ∧ when you meant to use ⇒

∀ x At(x, UMD) ∧ Smart(x)

means “Everyone is at UMD and everyone is smart”

Probably you meant to say

∀ x At(x, UMD) ⇒ Smart(x)

Everyone at UMD is smart.

Existential quantification

∃<variables><sentence>

Someone at UMD is smart:

∃ x At(x, UMD) ∧ Smart(x)

Another common mistake to avoid

A common mistake with ∃:

using ⇒ when you meant to use ∧:

∃ x At(x, UMD) ⇒ Smart(x)

This is equivalent to ∃ x ¬At(x, UMD) ∨ Smart(x)

There’s someone who either is smart or isn’t at UMD. That’s true if there’s anyone who is not at UMD. Probably you meant to say this instead: ∃ x At(x, UMD) ∧ Smart(x).There’s someone who is at UMD and is smart.

Properties of quantifiers

∀ x ∀ y is the same as ∀ y ∀ x

∃ x ∃ y is the same as ∃ y ∃ x

∃ x ∀ y is not the same as ∀ y ∃ x

“There is a person who loves everyone in the world”

∃ x ∀ y Loves(x, y)

“Everyone in the world is loved by at least one person”

∀ y ∃ x Loves(x, y)

Quantifier duality: each can be expressed using the other

∀ x Likes(x, IceCream) ¬∃ x ¬Likes(x, IceCream)

∃ x Likes(x, Broccoli) ¬∀ x ¬Likes(x, Broccoli)

Examples of sentences

Brothers are siblings

∀ x, y Brother(x, y) ⇒ Sibling(x, y)

"Sibling" is symmetric

∀ x, y Sibling(x, y) ⇔ Sibling(y, x)

One's mother is one's female parent

∀ x, y Mother(x, y) ⇔ (F emale(x) ∧ P arent(x, y))

A first cousin is a child of a parent's sibling

∀ x, y FirstCousin(x, y) ⇔∃ px, py P arent(px, x) ∧ Sibling(px, py) ∧ P arent(py, y)

5.4 Forward Reasoning(Data-driven Reasoning):

Construct a goal state starting from the initial state.

For example, suppose that the goal is to conclude the color of a pet named Fritz, given that he croaks and eats flies, and that the rule base contains the following four rules:

1. **If** X croaks and X eats flies - **Then** X is a frog
2. **If** X chirps and X sings - **Then** X is a canary
3. **If** X is a frog - **Then** X is green
4. **If** X is a canary - **Then** X is yellow

Let us illustrate forward chaining by following the pattern of a computer as it evaluates the rules. Assume the following facts:

- Fritz croaks
- Fritz eats flies

With forward reasoning, the inference engine can derive that Fritz is green in a series of steps:

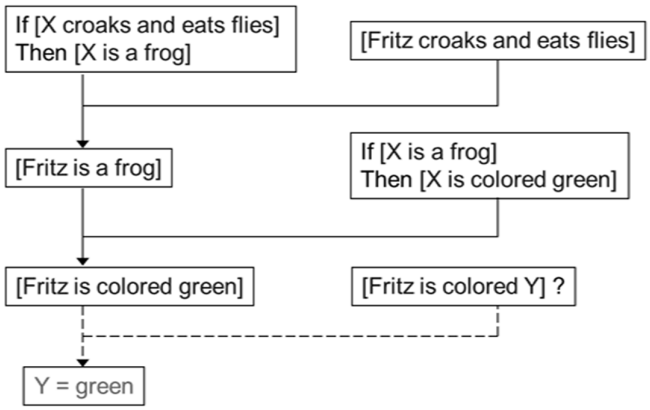
1. Since the base facts indicate that "Fritz croaks" and "Fritz eats flies", the antecedent of rule #1 is satisfied by substituting Fritz for X, and the inference engine concludes:

Fritz is a frog

2. The antecedent of rule #3 is then satisfied by substituting Fritz for X, and the inference engine concludes:

Fritz is green

The name "forward chaining" comes from the fact that the inference engine starts with the data and reasons its way to the answer, as opposed to backward chaining, which works the other way around. In the derivation, the rules are used in the opposite order as compared to backward chaining. In this example, rules #2 and #4 were not used in determining that Fritz is green.



5.5 Backward Reasoning(Goal driven Reasoning)

- prove a goal statement using initially valid facts

For example, suppose a new pet, Fritz, is delivered in an opaque box along with two facts about Fritz:

- Fritz croaks
- Fritz eats flies

The goal is to decide whether Fritz is green, based on a rule base containing the following four rules:

An Example of Backward Chaining.

1. **If** X croaks and X eats flies – **Then** X is a frog
2. **If** X chirps and X sings – **Then** X is a canary
3. **If** X is a frog – **Then** X is green
4. **If** X is a canary – **Then** X is yellow

With backward reasoning, an inference engine can determine whether Fritz is green in four steps. To start, the query is phrased as a goal assertion that is to be proved: "Fritz is green".

1. Fritz is substituted for X in rule #3 to see if its consequent matches the goal, so rule #3 becomes:

If Fritz is a frog – **Then** Fritz is green

Since the consequent matches the goal ("Fritz is green"),the rules engine now needs to see if the antecedent ("If Fritz is a frog") can be proved. The antecedent therefore becomes the new goal:

Fritz is a frog

2. Again substituting Fritz for X, rule #1 becomes:

If Fritz croaks and Fritz eats flies – **Then** Fritz is a frog

Since the consequent matches the current goal ("Fritz is a frog"), the inference engine now needs to see if the antecedent ("If Fritz croaks and eats flies") can be proved. The antecedent therefore becomes the new goal:

Fritz croaks and Fritz eats flies

3. Since this goal is a conjunction of two statements, the inference engine breaks it into two sub-goals, both of which must be proved:

Fritz croaks

Fritz eats flies

4. To prove both of these sub-goals, the inference engine sees that both of these sub-goals were given as initial facts. Therefore, the conjunction is true:

Fritz croaks and Fritz eats flies

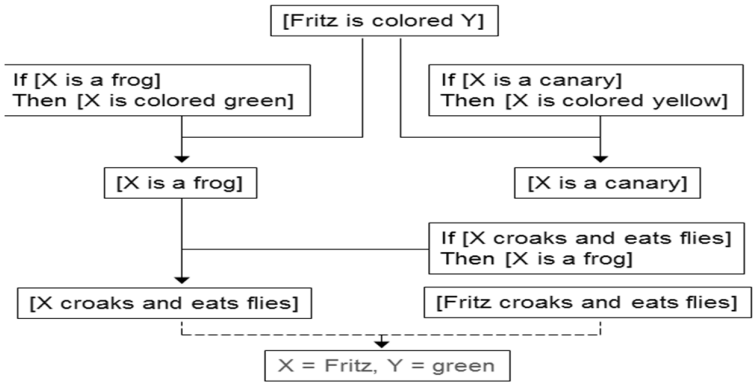
therefore the antecedent of rule #1 is true and the consequent must be true:

Fritz is a frog

therefore the antecedent of rule #3 is true and the consequent must be true:

Fritz is green

This derivation therefore allows the inference engine to prove that Fritz is green. Rules #2 and #4 were not used.



5.6 Semantic Networks

A **semantic net** (or semantic network) is a knowledge representation technique used for propositional information. So it is also called a propositional net. Semantic nets convey meaning. They are two dimensional representations of knowledge. Mathematically a *semantic net* can be defined as a labelled directed graph.

Semantic nets consist of nodes, links (edges) and link labels. In the semantic network diagram, nodes appear as circles or ellipses or rectangles to represent objects such as physical objects, concepts or situations. Links appear as arrows to express the relationships between objects, and link labels specify particular relations. Relationships provide the basic structure for organizing knowledge. The objects and relations involved need not be so concrete. As nodes are associated with other nodes semantic nets are also referred to as associative nets.

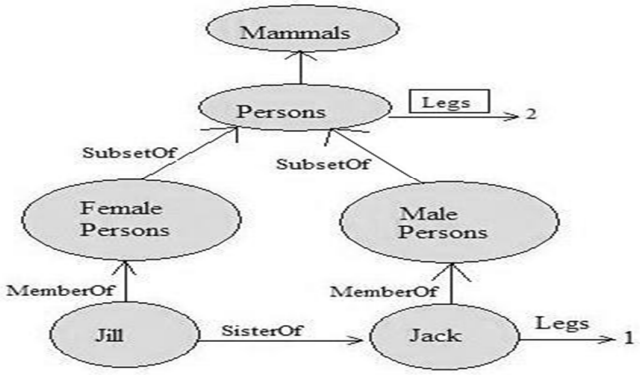
Semantic Networks for representing knowledge has particular advantages:

1. They allow us to structure the knowledge to reflect the structure of that part of the world which is being represented.
2. The semantics, i.e. real world meanings, are clearly identifiable.
3. There are very powerful representational possibilities as a result of "is a" and "is a part of" inheritance hierarchies.
4. They can accommodate a hierarchy of default values (for example, we can assume the height of an adult male to be 178cm, but if we know he is a baseball player we should take it to be 195cm).
5. They can be used to represent events and natural language sentences. The major idea is that:

- The meaning of a concept comes from its relationship to other concepts.
- The information is stored by interconnecting nodes with labelled arcs.

Representation in a Semantic Net

The physical attributes of a person can be represented in below figure.



In the above figure all the objects are within ovals and connected using labelled arcs. Note that there is a link between Jill andFemalePersons with label MemberOf. Simlarly there is aMemberOf link between Jack and MalePersons and SisterOf link between Jill and Jack. The MemberOf link between Jill and FemalePersons indicates that Jill belongs to the category of female persons.

Inheritance Reasoning

Unless there is a specific evidence to the contrary, it is assumed that all members of a class (category) will inherit all the properties of their superclasses. So semantic network allows us to perform inheritance reasoning. For example Jill inherits the property of having two legs as she belongs to the category of FemalePersons which in turn belongs to the category of Persons which has a boxed Legs link with value 2. Semantic nets allows multiple inheritance. So an object can belong to more than one category and a category can be a subset of more than one another category.

Inverse Links

Semantic network allows a common form of inference known as inverse links. For example we can have a *HasSister* link which is the inverse of *SisterOf* link.The inverse links make the job of inference algorithms much easier to answer queries such as who the sister of *Jack* is. On discovering that *HasSister* is the inverse of *SisterOf* the inference algorithm can follow that link HasSister from *Jack* to *Jill* and answer the query.

Disadvantage of Semantic Nets

1.One of the drawbacks of semantic network is that the links between the objects represent only binary relations.

2.There is no standard definition of link names.

Advantages of Semantic Nets

1.Semantic nets have the ability to represent default values for categories. In the above figure Jack has one leg while he is a person and all persons have two legs. So persons have two legs has only default status which can be overridden by a specific value.
2.They convey some meaning in a transparent manner.
3.They nets are simple and easy to understand.
4.They are easy to translate into PROLOG.

5.7 Frames

Frames can also be regarded as an extension to Semantic nets. Semantic nets initially used to represent labelled connections between objects. As tasks became more complex the representation needs to be more structured. A *frame* is a collection of attributes or slots and associated values that describe some real world entity. Each frame represents:

- a class (set), or
- an instance (an element of a class).

Need of Frames

Frame is a type of schema used in many AI applications including vision and natural language processing. The situations to represent may be visual scenes, structure of complex physical objects, etc. A frame is similar to a record structure and corresponding to the fields and values are slots and slot fillers. Basically it is a group of slots and fillers that defines a stereotypical object. A single frame is not much useful. Frame systems usually have collection of frames connected to each other. Value of an attribute of one frame may be another frame.

A frame for a book is given below.

Slots	**Fillers**
publisher	Thomson
title	Expert Systems
author	Giarratano
edition	Third
year	1998
pages	600

Frames can represent either generic or frame. Following is the example for generic frame.

Slot	**Fillers**
name	computer
specialization_of	a_kind_of machine
types	(desktop, laptop,mainframe,super) if-added: Procedure ADD_COMPUTER
speed	default: faster if-needed: Procedure FIND_SPEED
location	(home,office,mobile)
under_warranty	(yes, no)

The fillers may values such as computer in the name slot or a range of values as in types slot. The procedures attached to the slots are called procedural attachments. There are mainly three types of procedural attachments: if-needed, default and if-added. As the name implies if-needed types of procedures will be executed when a filler value is needed. Default value is taken if no other value exists. Defaults are used to represent commonsense knowledge.

References:

- Stuart J. Russel,Peter Norvig, "Artificial intelligence- a novel approach". 2nd Edititon, ,Pearson Education, 2003