Part-II: Softcomputing

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay.

Introduction

Softcomputing is concerned with problems where the input often contains inaccuracy, noise and incompleteness. The criteria of success are not precisely defines.

Neural Nets and *Fuzzy Systems* form the most important components of softcomputing.

Neural Networks is a computing device composed of very simple processing elements called *neurons* which are interconnected. The processing power in such devices comes from many connections.

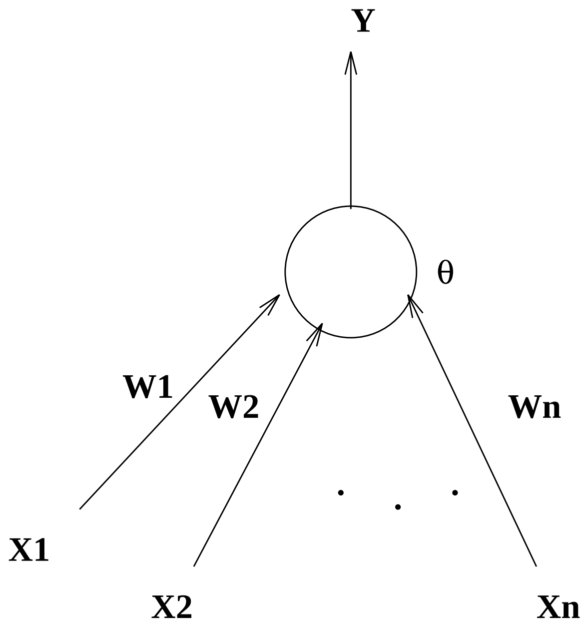
There are different models of neural network like *Perceptron, Back Propogation, Hopfield, Adaptive Resonance Theory, Kohonen Net.*

The first three models are *supervised* learning algorthims and the rest are *unsupervised.* That is, in the former the examples are marked with a label by a teacher while in the later it is not so.

Perceptrons

This are single neurons motivated by Brain Cells.

The model is:



$y = 0$ if $\sum_{i=1}^{n} w_i x_i < \theta$

and, $y = 1$ if $\sum_{i=1}^{n} w_i x_i > \theta$

That is, if the net input is more than the threshold the neuron *fires*, i.e., assumes a state 1, else the state is 0.

Computing by Perceptron

How do such computing devices compute?

AND function:

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

Inequalities:

$0.W_1 + 0.W_2 < \theta$
$0.W_1 + 1.W_2 < \theta$
$1.W_1 + 0.W2 < \theta$
$1.W_1 + 1.W_2 > \theta$

W_1, W_2 are weights and θ is the threshold,

Continued

Feasible values:

$\theta = 1.5$, $W_1 = W_2 = 1$

For OR Function $\theta = 0.5$, $W_1 = W_2 = 1$

XOR function is not computable as the system of inequalities below is inconsistent:

$$0.W_1 + 0.W_2 < \theta$$
$$0.W_1 + 1.W_2 > \theta$$
$$1.W_1 + 0.W_2 > \theta$$
$$1.W_1 + 1.W2 < \theta$$

Out of 2^{2^2} Boolean functions 14 are computable by perceptron

Functions computable by a perceptron are called *Threshold functions*

Geometrical View

The computability or otherwise can be understood from the following figure:

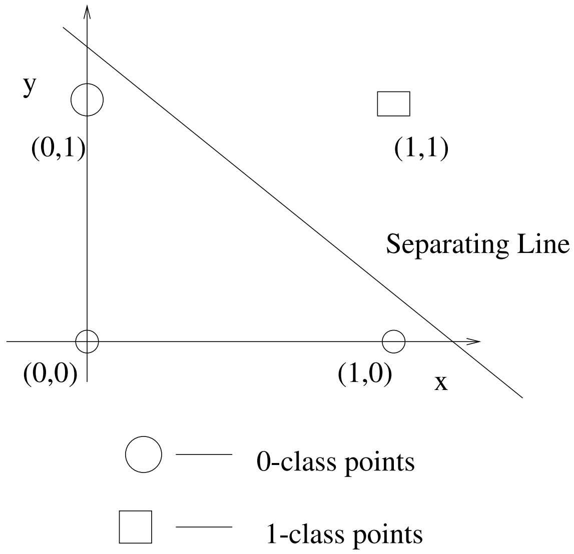


Figure- Separability of AND function

The above figure shows how the points of one class are separated from the points of the other class by a linear surface- in this case a straight line.

If points of one diagonal are in one class and those on the other are in the other (as in X-OR problem) the function is not *linearly separable*.

Perceptron Training Algorithm

Perceptrons need to *learn* the weight and threshold values.

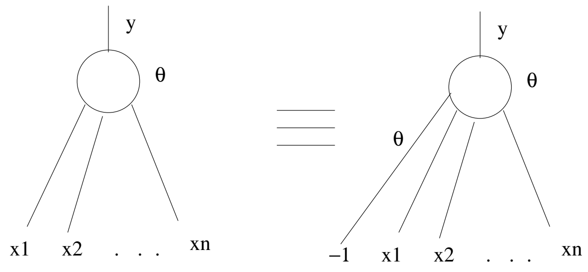


Figure- Threshold absorbed as a weight with as -1.

The threshold is absorbed as a weight and a new input line is introduced which is applied a constant input of -1.

Also we negate the components of the vector in the 0-class so that only $\sum_{i=0}^{n} w_i x_i > \theta$, need to be performed.

Now in the perceptron training algorithm we need to find the values for

$$\bar{W} = < w_0, w_1, ..., w_n >$$

PTA steps

Perceptron Training Algorithm

Input : Preprocessed vectors X_i, $i = 1$ to m

1. Choose random values for w_i, $i = 0$ to n
2. i:= 1
3. IF $\bar{W}.\bar{X}_i > 0$ goto 6
4. $\bar{W} := \bar{W} + \bar{X}_i$
5. goto 2
6. $i := i + 1$
7. IF $i > m$ goto 9
8. goto 3
9. End

Thus whenever an example is misclassified we add it to the weight vector and this goes on until the test $\bar{W}.\bar{X}_i > 0$ is satisfied for all input vectors.

PTA: example

```
AND function
------------
```

1-class: $< 1, 1 >$

0-class: $< 0, 0 >, < 0, 1 >, < 1, 0 >$

Augmented classes:

1-class: $< -1, 1, 1 >$

0-class: $< -1, 0, 0 >, < -1, 0, 1 >, < -1, 1, 0 >$

Negating the 0-class vectors, the total set of vectors is:

$$\bar{X_1} =< -1, 1, 1 >$$
$$\bar{X_2} =< 1, 0, 0 >$$
$$\bar{X_3} =< 1, 0, -1 >$$
$$\bar{X_4} =< 1, -1, 0 >$$

PTA example (contd.)

Start with a random value for the weight vector:

$\bar{W} =< 0, 0, 0 >$
$\bar{W}.\bar{X_1} = 0$ and hence fails
$\bar{W_{new}} = \bar{W_{old}} + \bar{X_i} =< -1, 1, 1 >$

fails for $\bar{X_2}$.

Keep on adding the failed vectors. Finally the result must come. **WHY** ?

Perceptron Training Convergence Theorem.

Statement:
Whatever be the initial choice of the weights, the Perceptron Training Algorithm will eventually converge by finding the correct weight values provided the function being trained is linearly separable.

Proof Sketch

Suppose the weight vector at the $n^t h$ iterations is W_n. Since we are considering a linearly separable function, there must exist a separating vector W_*.

Consider the function, $G(n) = \frac{W_n . W_*}{|W_n|}$.

The magnitude of $G(n)$ is upper bounded by $|W_*|$, as can be seen by expanding the dot product.

But as n increases the numerator of $G(n)$ increases *linearly* to infinity.

The denominator increases as *square root* of n.

Thus if the perceptron training algorithm does not converge, $G(n)$ will increase contradicting the upper bound. Hence the algorithm must converge.

Non-Linearity

How to deal with Non-linearity:

1. Tolerate some error e.g. in Case of XOR Classify the X points but include one O point also

2. Separate by higher order surface but in that case the neuron does not remain a linear threshold element.

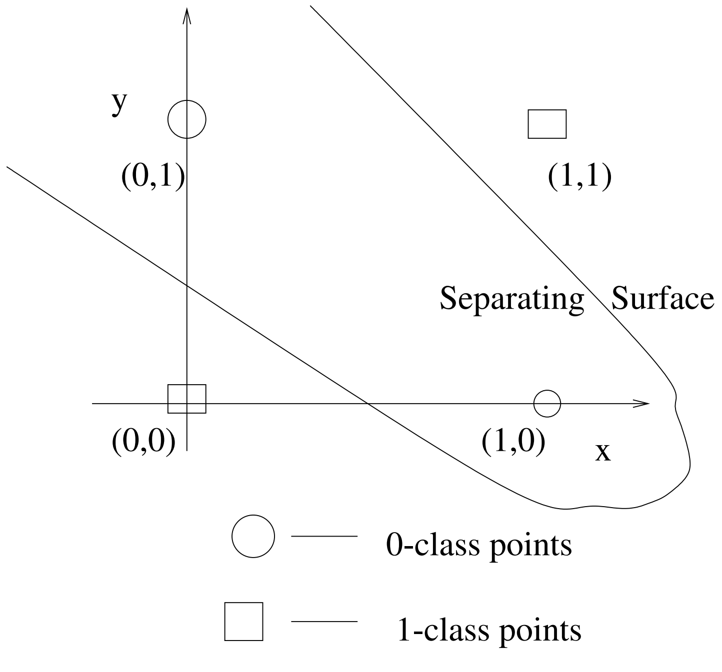


Figure- Separation of points by higher dimensional surface

Non-linearity (contd.)

3. Introduce more hyperlanes so that one segment encloses points of only one kind and no point of the other kind.

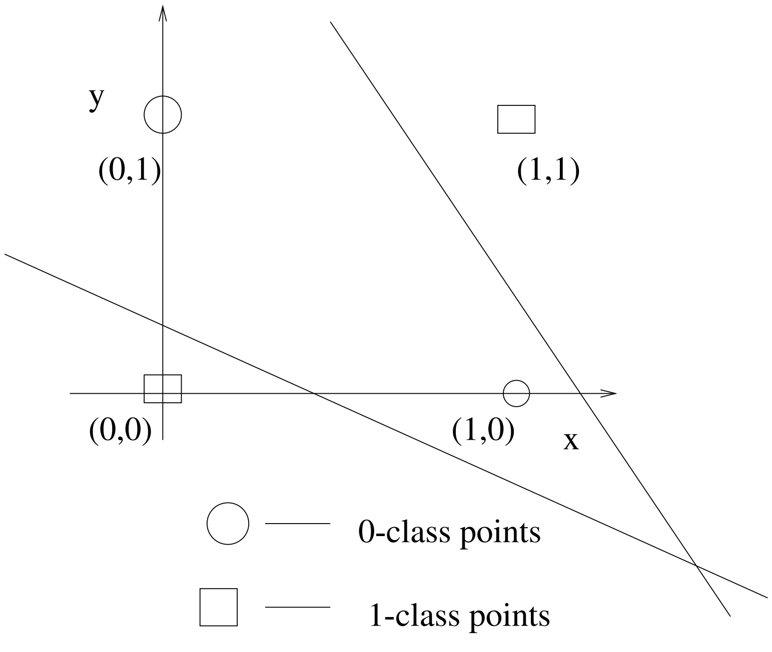


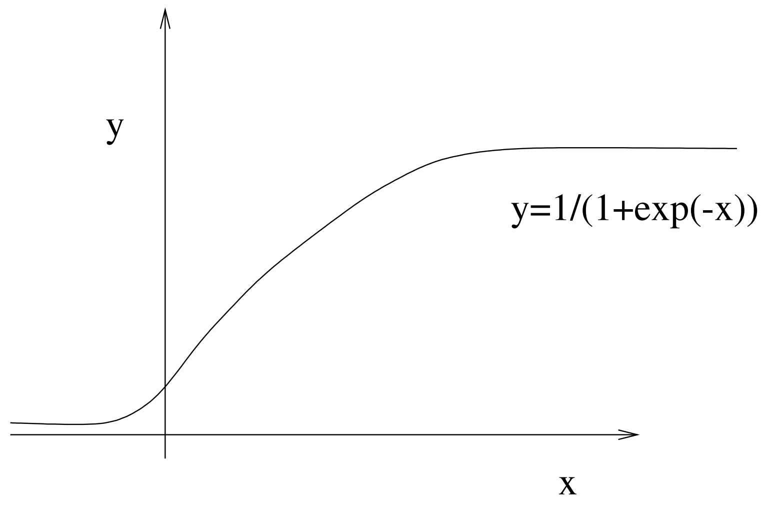
Figure- Separation by two linear surfaces.

The two lines enclose 0 points.

Multilayer Perceptron

Multilayer Perceptrons (figure next slide) are more powerful than single layer perceptron.

The neurons have *Sigmoid function* as input output relationship.



The Sigmoid Curve

Merits of Sigmoid:

1. Biological Plausibility: mimics the behaviour of actual brain cells.
2. Easy to compute Derivative: $\frac{dy}{dx} = y(1-y)$.

Backpropagation Algorithm

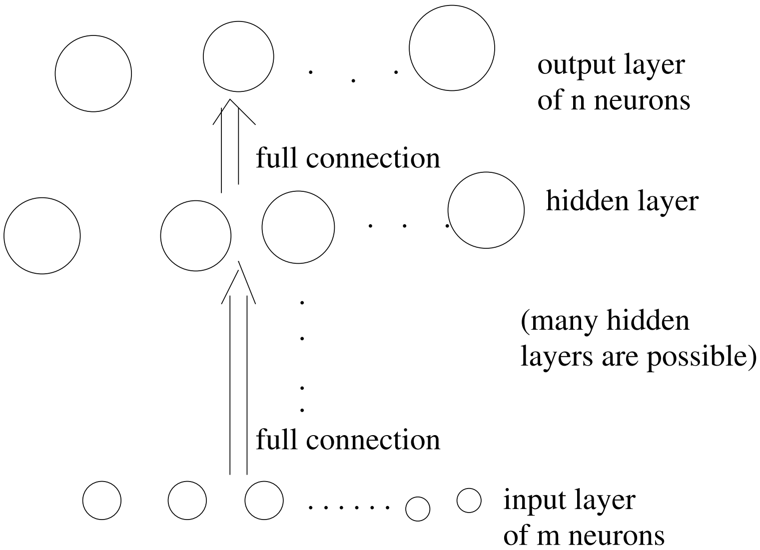


Figure- Multilayer Perceptron
(Pure Feedforward)

Each neuron in one layer is connected to all neurons in the next layer if any.

Let $\bar{I} =< i_1, i_2, \ldots, i_m >$ be the input vector, $\bar{O} =< o_1, o_2, \ldots, o_n >$ be the output vector, and $\bar{T} =< t_1, t_2, \ldots, t_n >$ be the target vector

Then error $E = \frac{1}{2}\sum_{i=1}^{n}(t_i - o_i)^2$.

BP: Gradient Descent

Gradient Descent Rule:

$$\Delta w_{ji} \quad \alpha \quad -\frac{\delta E}{\delta w_{ji}}$$

where, Δw_{ji} = change in weight between i^{th} (feeding) and j^{th}a (fed) neuron.

With learning constant η as constant of proportionality,

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}}$$

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}}$$

define, $\frac{\delta E}{\delta net_j} = \delta_j$ and, $\frac{\delta net_j}{\delta w_{ji}} = o_i$. Then

$$\Delta w_{ji} = \eta \delta_j o_i$$

Weight Change Expressions

For output layer:

$$\delta_j = (t_j - o_j)o_j(1 - o_j)$$

For hidden layer:

$$\delta_j = (\sum_k \delta_k w_{kj})o_j(1 - o_j)$$

(j^{th} neuron in the hidden layer feeds the k^{th} neuron in the next layer.)

Recursive rules for the hidden layer:

$$\delta_j = (\sum_k w_{kj}\delta_k)o_j(1 - o_j)$$

Knowing δ values for all layers, the weight change values for the whole network can be easily found.

The weight change always reduces the error. Hence BP is by nature a *greedy* strategy.

Analysis and Application of BP

Influence of Learning Rate:
Large η gives fast progress, but oscillation about minima.
Too small η makes the progress of the algorithm very slow.

Symmetry Breaking:
If mapping demands different weights, but we start with same weights everywhere, then BP will never converge.

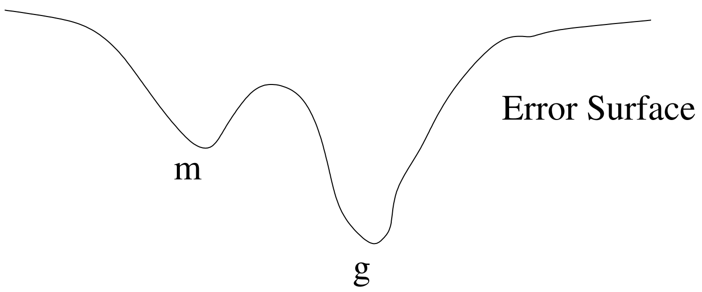
Momentum Factor:

$$\Delta w_{ji_{(k)^{th}\ \ step}} = \eta \delta_j o_i + \beta \Delta w_{ji_{(k-1)^{th}\ \ step}}$$

where, $\beta =$ momentum factor.
Momentum factor hastens the operating point towards minima and once near the minima it dampens the oscillations.

Local Minima

Local Minima Problem:



m- local minima, g- global minima

Figure- Getting Stuck in local minimum

Due to *Greedy* nature of BP, it can get stuck in local minimum m and will never be able to reach the global minimum g as the error can only decrease by weight change.

Some useful tips for BP

If the network weights do not change, one of the following might have happened:

a) Learning Rate too small.

b) *Network paralysis* due to neurons operating near saturation region (1 or 0) as the inputs are high positive or negative.

In this case scale the inputs.

c) Network stuck in local minimum.

In this case start with fresh random set of weights.

Tips (contd.)

If there is equal distribution of positive and negative values in the input then use the $tanh(x)$ curve, where

$$y = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Its curve is shown below:

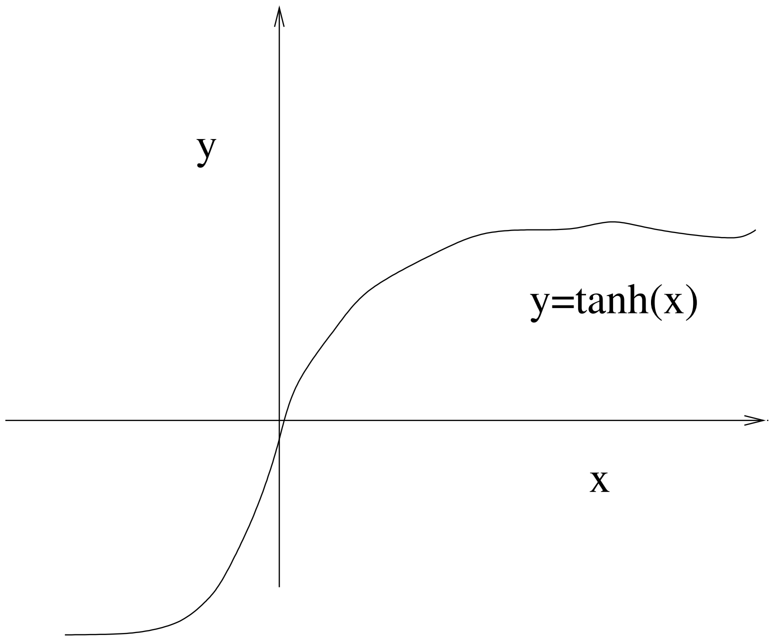


Figure- tanh curve.

Alphabet Recognition using BP

There are 26 alphabets of English. To recognize an alphabet 26 neurons (centralized representation) or 5 neurons (distributed representation) could be used in the output layer.

For the input an $N \times N$ display could be imagined where N increases with the desired resolution of the input display.

Each junction of the grid corresponds to a neuron in the input layer. The neuron is on/off depending on the state of the junction.

continued

The hidden layer needs to be found by trial and error. About 10 neurons suffice for an 8×8 input resolution.

The learning rate is kept around 0.6 and the momentum around 0.4.

On the SNNS package (Stuttgart Neural Network Simulator) backpropagation takes about 4000 iterations to train the first 3 alphabets (A, B, C).

The input variation arises because of the different styles and sizes of writing an alphabet and also because some alphabets are close in similarity (B and D, P and R and so on).

Hopfield Net

Hopfield Net is inspired by associative memory which means memory retrieval is not by address, but by part of the data.

Hopfield Net Architecture consists of N neurons fully connected with symmetric weight strength $w_{ij} = w_{ji}$.

There is no self connection. So the weight matrix is 0-diagonal and symmetric.

Each computing element or neuron is a linear threshold element with threshold, $\theta = 0$. The two levels of logic are +1 and -1.

Computation

The computation proceeds as follows:

At every time instant t, look randomly at neuron i.

Examine the input, $\sum_{j=1 j \neq i}^{n} w_{ij}\sigma_j(t)$,

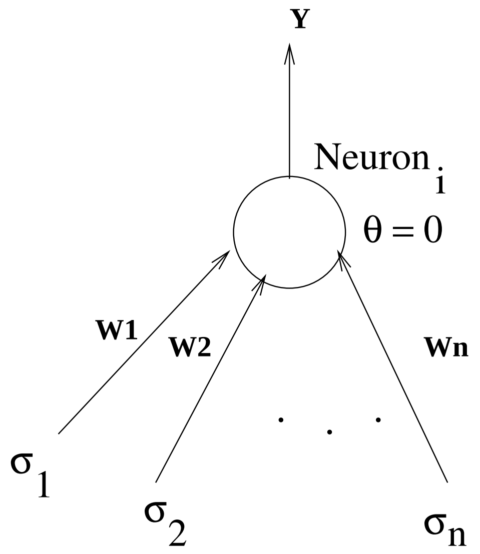


Figure: A neuron in the Hopfield Net.

$\sigma_j(t)$ is the activation of the j^th neuron at the instant t.

Continued

The activation $\sigma_i(t+1)$ is

$$\sigma_i(t+1) = sgn(\sum_{j=1 j\neq i}^{N} w_{ij}\sigma_j(t))$$

where,

$$sgn(x) = +1, \quad if \quad x > 0 \quad and = -1 \quad if \quad x < 0$$

sgn is called the signum or the sign function.

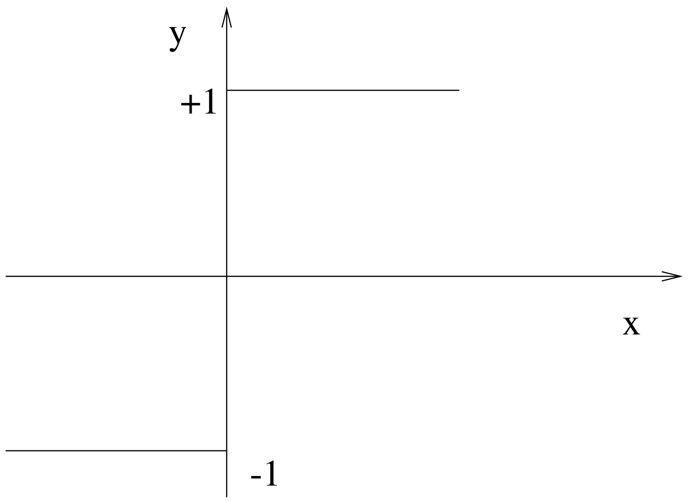


Figure: The Signum Function

stability

This mode of operation is called the *asynchronous* mode of operation of the Hopfield Net.

In the *synchronous* mode of operation all neurons update themselves simultaneously at any instant of time.

Since there is feedback in the Hopfield Net the question of *stability* arises. At every time instant the network evolves and finally settles into a stable state.

How does the Hopfield Net function as *associative* memory ?

One needs to store or stabilize a vector which is the memory element.

Example

Example :

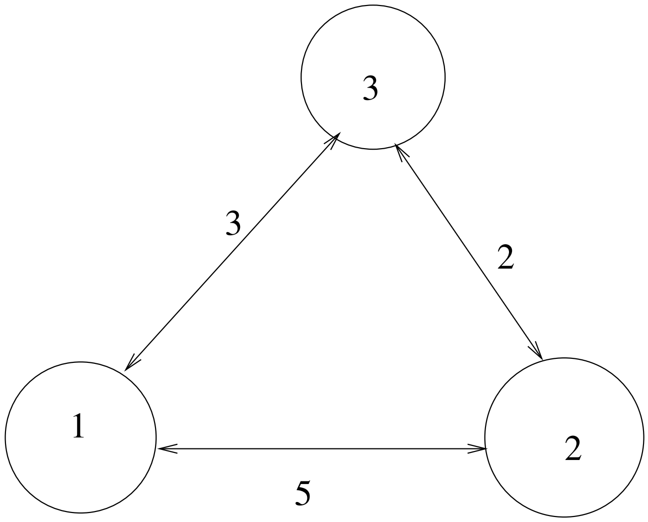


Figure: An example Hopfield Net

In the above Hopfield Net the connection strengths are:

$$w_{12} = w_{21} = 5$$

$$w_{13} = w_{31} = 3$$

$$w_{23} = w_{32} = 2$$

Continued

At time $t = 0$,

$$s_1(t) = 1$$

$$s_2(t) = -1$$

$$s_3(t) = 1$$

Is the state $< 1, -1, 1 >$ stable ?

Examine 1:
Net input at $t = 0$ is $5 \times -1 + 3 \times 1 = -2$.
So the neuron will change state to -1.

Thus $< 1, -1, 1 >$ is NOT stable. Stability is a function of the Weight values.

Hopfield Rule

We ask the question: What should be the values of the w_{ij}, $\forall i, j$ such that the vector :

$$< \sigma_1, \sigma_2, \ldots, \sigma_{N-1}, \sigma_N >$$

is stable in the Hopfield Net of N neurons.

Hopfield's Proposal for weight learning:

$$w_{ij} = \frac{1}{N-1}\sigma_i \sigma_j$$

$\frac{1}{N-1}$ is a normalization factor.

$\sigma_i \sigma_j$ is a term which comes from *Hebb's Rule of learning* which states:
If two connected neurons are both ON then the weight of the connection between them should be such that the mutual excitation is sustained.

Similarly, if the two neurons inhibit each other then their connections should sustain the mutual inhibition.

Radius of Attraction and Training

Suppose the net is trained for the pattern

$$< \sigma_1, \sigma_2, \ldots \sigma_N >$$

If $k < \frac{N-1}{2}$, then $\sigma_i(t+1)$ will be same as $\sigma_i(t)$.

The stable pattern is called the *attractor for the net* and k is called the *Radius of Attraction.*

We want to load the vectors

$$< \sigma_1^1, \sigma_2^1, \ldots \sigma_N^1 >$$

$$< \sigma_1^2, \sigma_2^2, \ldots \sigma_N^2 >$$

$$< \sigma_1^3, \sigma_2^3, \ldots \sigma_N^3 >$$

$$< \sigma_1^P, \sigma_2^P, \ldots \sigma_N^P >$$

on the Hopfield net.

The training rule is $w_{ij} = \frac{1}{N-1} \sum_{p=1}^{P} \sigma_i^p \sigma_j^p$

It can be shown using statistical mechanical techniques that $P \leq 0.15N$.

Energy

Hopfield net is a feedback system. It evolves over time. Some patterns have to be made stable. These should correspond to the minimum energy states.

Energy at state $\bar{\sigma} =< \sigma_1, \sigma_2, \ldots \sigma_N >$ is,
$E(\bar{\sigma}) = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ij} \sigma_i \sigma_j$

Change in Energy, $\Delta E = E^{final} - E^{initial}$

Let the p^{th} neuron change state from $\sigma_p^{initial}$ to σ_p^{final}; then

$E^{initial} = -\frac{1}{2} \sum_{j \neq p} w_{pj} \sigma_p^{initial} \sigma_j + T$

$E^{final} = -\frac{1}{2} \sum_{j \neq p} w_{pj} \sigma_p^{final} \sigma_j + T$,
where T is a term independent of σ_p.

Continued

$\Delta E = (\sigma_p^{initial} - \sigma_p^{final}) \sum_{j \neq p} w_{pj} \sigma_j$
or, $\Delta E = \Delta\sigma_p \sum_{j \neq p} w_{pj} \sigma_j$
Thus

$$\Delta E = \Delta\sigma_p \times netinput_p$$

If p changes state from +1 to -1 then $\Delta\sigma_p$ is positive, and $netinput_p$ is negative. Reverse is the case when p changes state from -1 to +1.

Hence ΔE is always negative. Thus energy always decreases when the neuron changes state.

Application of Hopfield Net

Hopfield nets are applied to *Optimization Problems.*

Motivation: Optimization problems maximize or minimize a function.

In the Hopfield Net the energy gets minimized.

This is the correspondence between the Hopfield net computation and the optimization problem.

Example

The Traveling Salesman Problem:

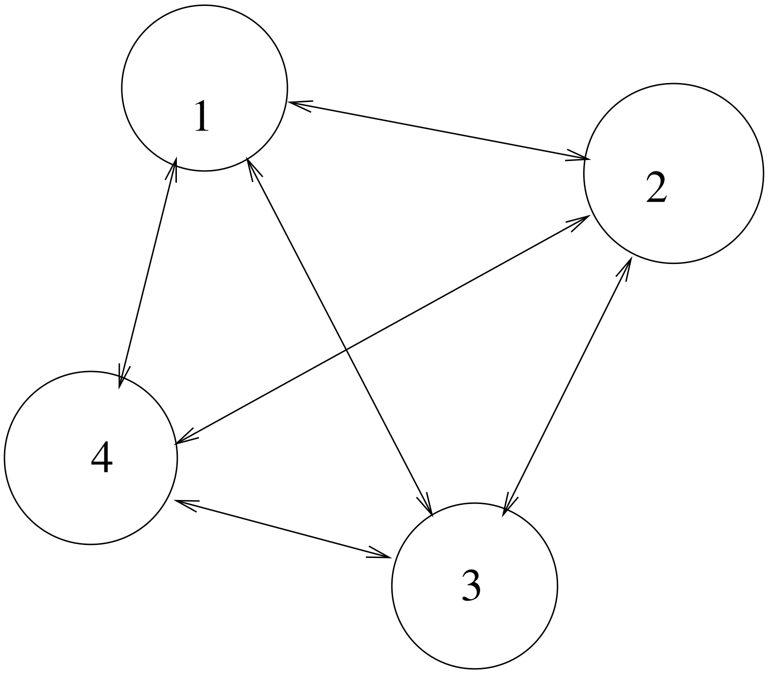


Figure: 4-city Travelling Salesman Problem

The salesman starts from the city 1 and visits all the other cities exactly ONCE. The tour has to be minimized.

It is known that the problem is *NP-Hard* (this is a class of problems for which currently no efficient algorithms are known).

Hopfield Net for TSP

N cities are represented by an $N\ X\ N$ matrix of neurons.

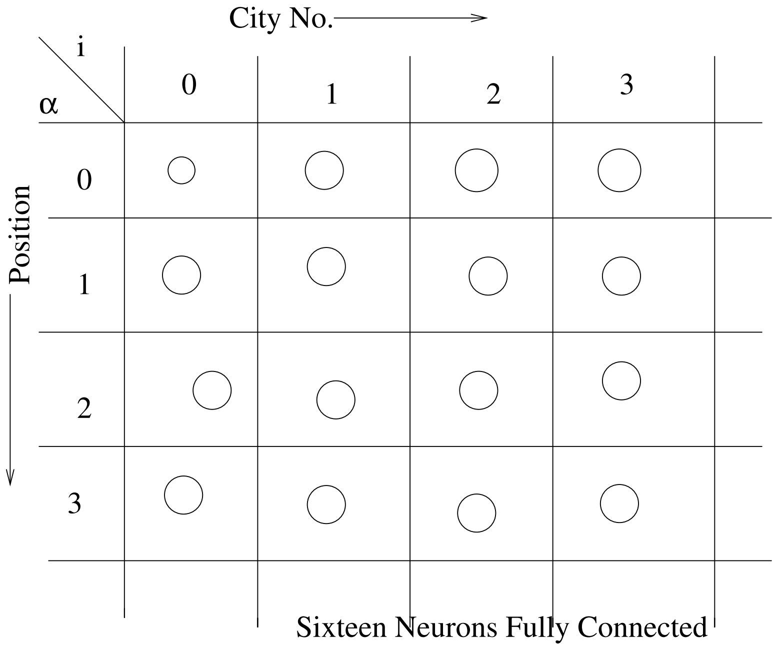


Figure: Hopfield Net representation for TSP

$x_{ia} = 1$, if city i is in the position α, otherwise $x_{ia} = 0$.

Continued

The matrix has exactly N 1's.

Each columns has exactly ONE 1.

Each Row has exactly ONE 1.

Matrix defines a *Hamilton Circuit* which is the closed tour of a graph covering each vertex once and exactly once.

Now if for each element of the matrix we have a neuron and then fully connect the assembly with symmetric weights then we have a Hopfield Net with N^2 elements.

This Hopfield Net dynamics can be tried to solve the TSP.

Architecture

Architecture of the Hopfield Net.

N^2 neurons are fully connected with symmetric weights. Strength of the weights is a function of the distance.

In all such optimization problems, the task is to find a suitable *Energy function* E .

E for TSP has four components satisfying 4 constraints :

TSP energy expression

a) Exactly N cities are visited :

$$E_3 = \frac{D}{2}(\sum_{i\alpha} x_{i\alpha} - N)^2$$

D = constant.
($\frac{1}{2}$ factor to correspond to the $\frac{1}{2}$ factor in the original energy expression.)

b) Each city can be in at the most one tour position :

$$E_2 = \frac{C}{2}\sum_{i\alpha}\sum_{i\beta,\beta\neq\alpha} x_{i\alpha}x_{i\beta}$$

(takes product of pairs in the $i^t h$ row.)

continued

c) Each position in the tour has at most one city :

$$E_1 = \frac{B}{2} \sum_{i\alpha} \sum_{j\beta, j \neq i} x_{i\alpha} x_{j\alpha}$$

(takes the product of pairs in the α^{th} column).

d) The distance has to be minimum:

$$E_0 = \frac{A}{2} \sum_{i} \sum_{j \neq i} \sum_{\alpha} \sum_{\beta \neq \alpha} d_{ij} x_{i\alpha} (x_{j,\alpha+1} + x_{j,\alpha-1})$$

Continued

$E = E_0 + E_1 + E_2 + E_3 \qquad (1)$

E is minimized when the TSP tour is found.

Constraints a, b, c define a Hamilton circuit.

To find the weight values $w_{i\alpha,j\beta}$, where $n_{i\alpha}$ and $n_{j\beta}$ are neurons in the (i^{th} row and α^{th} column), and (j^{th} row and β^{th} column), *match*

$E(\bar{\sigma}) = -\frac{1}{2}\sum_{i\alpha}\sum_{j\beta} w_{i\alpha,j\beta}\sigma_{i\alpha}\sigma_{j\beta}$
with (1).

Result

Reported experiments with B, C, D values of $O(1000)$ give unsatisfactory results. Minimum distance tour is found in only about 8% of the cases.

Using other energy expression, about 25% of the cases yielded the right tour.

TSP is an NP-hard problem and the the Hopfield net for N-city TSP problem having N^2 neurons reaches minima in $O(N^4)$ steps.

Thus the Hopfield net will fail to converge in most cases.

However, for problems with polynomial time complexity (like sorting) good results are obtained.

Fuzzy Systems

The motivation for fuzzy logic comes from the need to represent qualitative information and draw inferences thereform.

In real life we work with inprecise information.

In process control, which involves real time computing the human expert takes decision based on imprecise information.

Example: Since the pressure is high and the temperature is moderately low, I will turn knob-2 slightly to push in more material.

The inferencing is being done on Fuzzy ideas like *high, moderately low* etc.

Historically, Lotfi Zadeh's work in the 60s started the area. It has been applied to many problems in Artifcial Intelligence and Neural Networks.

Crisp Sets vs. Fuzzy Sets

The concept of *Membership Function* is fundamental. In traditional sets called *crisp sets* an element either belongs to the set or it does not.

But in fuzzy sets a member belongs with a certain *degree* which is between 0 and 1.

membership function is denoted as $\mu_A(x)$ which means the amount by which the element x belongs to the fuzzy set A.

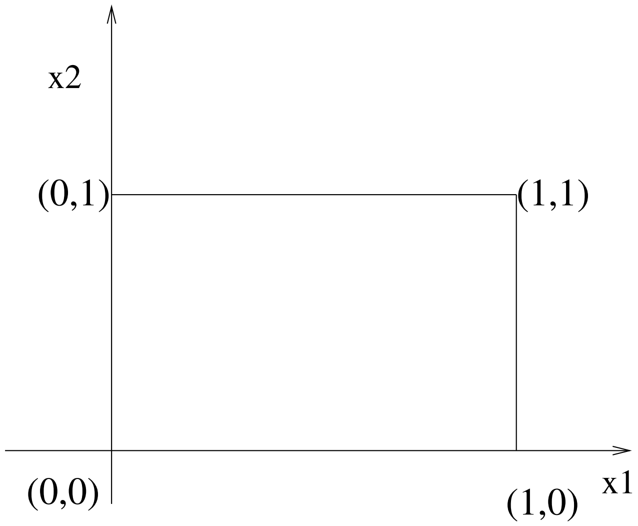
For crisp sets, $\mu_A(x) = 1$ if $x \in A$, else 0.

For fuzzy sets, $\mu_A(x) = [0, 1]$, i.e any value between 0 and 1 inclusive.

Geometrical View

The fuzzy sets can be looked upon as the points within and on the unit hypercube in an n-dimensional space where n is the number of elements in the universe.

As an example, suppose the universe is the set $A = \{x_1, x_2\}$.



Figure; Geometrical representation of Fuzzy sets

Crisp subsets of A are $\Phi, \{x_1\}, \{x_2\}, \{x_1, x_2\}$.

They are represented by the 4 corners of the unit square $(0,0), (1,0), (0,1), (1,1)$.

Fuzzy subset of A

A fuzzy subset B of A is $B = \{0.3/x_1, 0.8/x_2\}$, where $x_1 \in B$ with fuzziness 0.3 and $x_2 \in B$ with fuzziness 0.8.

B is a point inside the n-dimensional hypercube.

Fuzziness is not same as probability.

$|B| = 0.3 + 0.8 = 1.1$, which is not possible in probablilty.

This means that the membership values of fuzzy elements cannot be looked upon as probablily values.

Fuzzy set operations

The operations of *Union, Intersection, Negation* etc. can be defined on fuzzy sets.

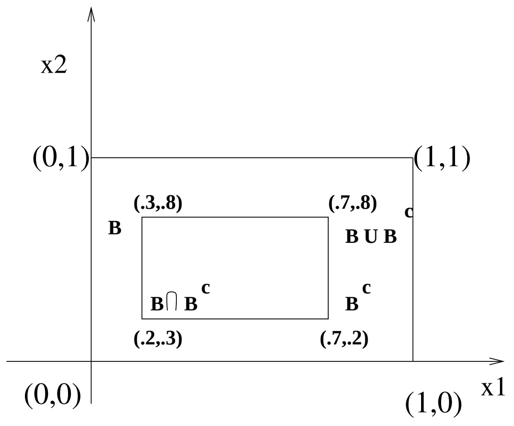


Figure: Geometric operations on fuzzy a fuzzy set and operations on it.

$B, B^c, B \cup B^c, B \cap B^c$ are points on the corners of the inner rectangle.

As fuzziness decreases the inner rectangle approaches the corners of the hypercube.

As fuzziness increases the rectangle collapses into the point (0.5, 0.5) which is the point of maximum Fuzziness.

Expressions for operations

We can define expressions for fuzzy set operations in terms of membership functions.

Intersection corresponds to *minimum* of membership values, *Union* corresponds to *maximum* and *complement* corresponds to subtraction of membership value from 1.

Thus

$$\mu_{A\cap B}(x) = min(\mu_A(x), \mu_B(x))$$

$$\mu_{A\cup B}(x) = max(\mu_A(x), \mu_B(x))$$

$$\mu_{A^c}(x) = 1 - \mu_A(x)$$

Fuzzy Logic

In fuzzy logic, the predicates do not have only 0 and 1 values. Instead they have values *between* 0 and 1, both inclusive.

AND ($\wedge$) operation is performed by taking *minimum* of two fuzzy predicates, *OR* ($\vee$) by taking *maximum* and *NOT* ($\sim$) by taking *difference from 1.*

The *IMPLICATION* operation is different from classical logic. It does not correspond to the *subset* operation.

Implication is expressed as

$$\mu_A(x) \rightarrow \mu_B(x) \equiv min(1, 1 - \mu_A(x) + \mu_B(x))$$

Linguistic Variable

In fuzzy logic we work with words and phrases which are called *Linguistic Variables.*

For example, *tall, short, heavy, long, light* etc. are linguistic variables.

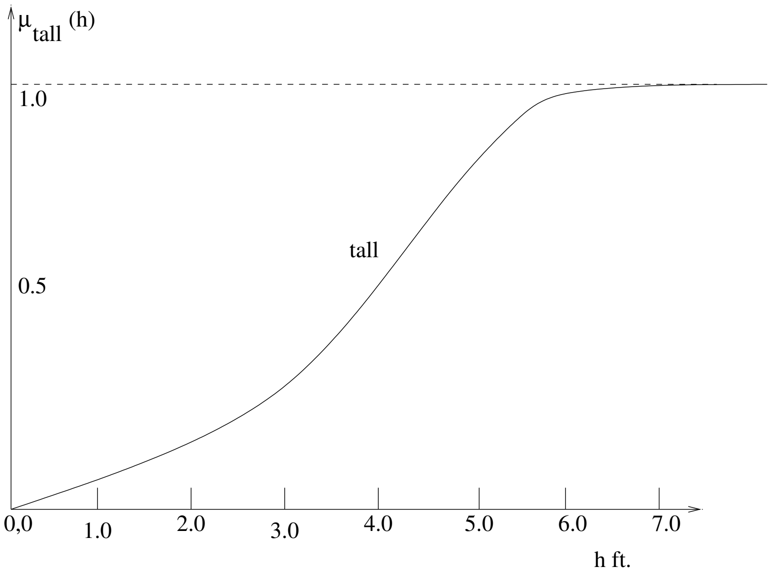


Figure: Profile of *tall*

In the above figure the *profile* for the linguistic variable *tall* is shown.

For example a height of 3 ft. has approximately a membership value of 0.4. That means persons having height of 3 ft. can be called *tall* with a fuzziness value of 0.4.

Hedges

Human beings also work with adjectives which are qualified with words like *very, slightly, much, extremely* etc.

Mathematically, *squaring* corresponds to *very* and *square root* corresponds to *somewhat.*

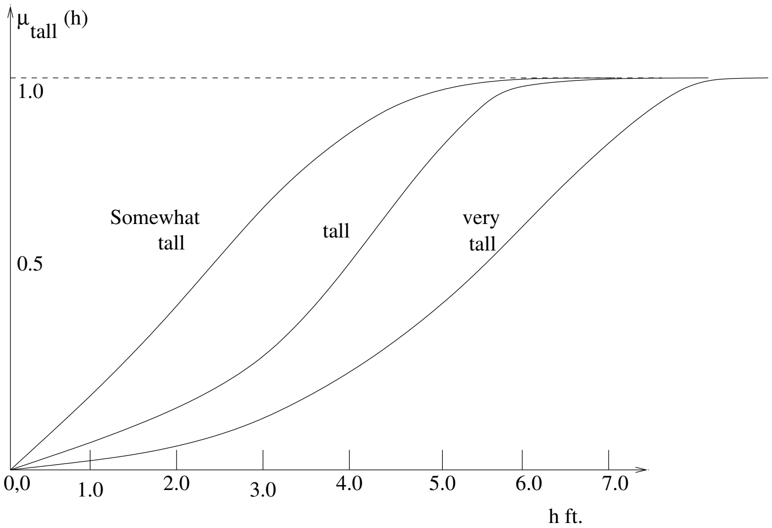


Figure: Profile of *tall* with hedges

In the above figure, the curves show the profiles for *somewhat tall, tall* and *very tall.*

Fuzzy Inferencing

One of the most important applications of fuzzy logic is inferencing on *Fuzzy Expert Systems.*

The fuzzy rule base is of the form (for example):

If pressure is high AND teperature is low,
Then turn the knob slightly right.

Here *high, low, slightly right* are linguistic variables.

Inferencing is done on the profiles of linguistic variables as shown by an example now.

Inverted Pendulum Control

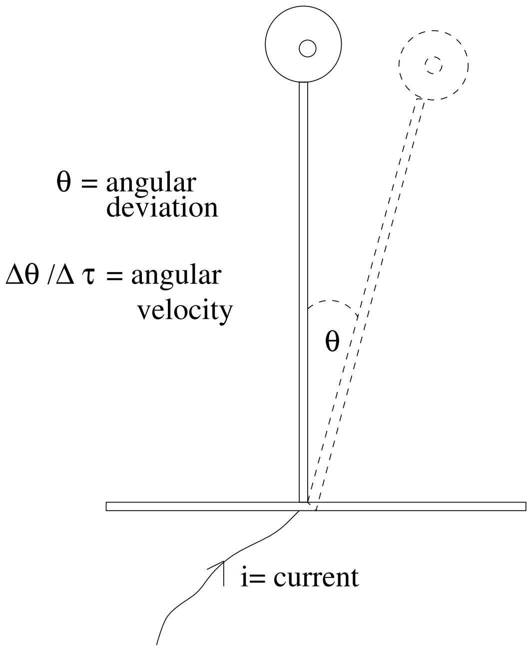


Figure: Control of Inverted Pendulum

The figure above shows an inverted pendulum which is controlled by sending the current i according to its angular position and velocity.

A typical rule is of the form
IF θ *is negative small AND* $\frac{\Delta\theta}{\Delta t}$ *is negative small*
THEN i *positive small*

Here positive and negative refer to the clockwise and anticlockwise directions of angular position and velocity.

Control Rules

$\Delta\theta/\Delta\tau$ \ θ	NM	NS	ZE	PS	PM
NM			PM		
NS			PS		
ZE	PM	PS	ZE	NS	NM
PS			NS		
PM			NM		

N-negative, P-positive M-medium, S-small ZE-zero

Figure: Some Control Rules for Inverted Pendulum Control

In the above figures the entries in the cells of the matrix describe a control rule. For example, the top row centre cell says that

if the angular deviation is zero and the angular velocity is negative medium then the current applied should be positive medium

The above rule will ensure that the motor will apply *positive medium* force to bring back the pendulum.

Profiles

The profiles of linguistic variables *negative medium, negative small, zero, positive small* and *positive medium* are shown in the figure below by means of the polygons.

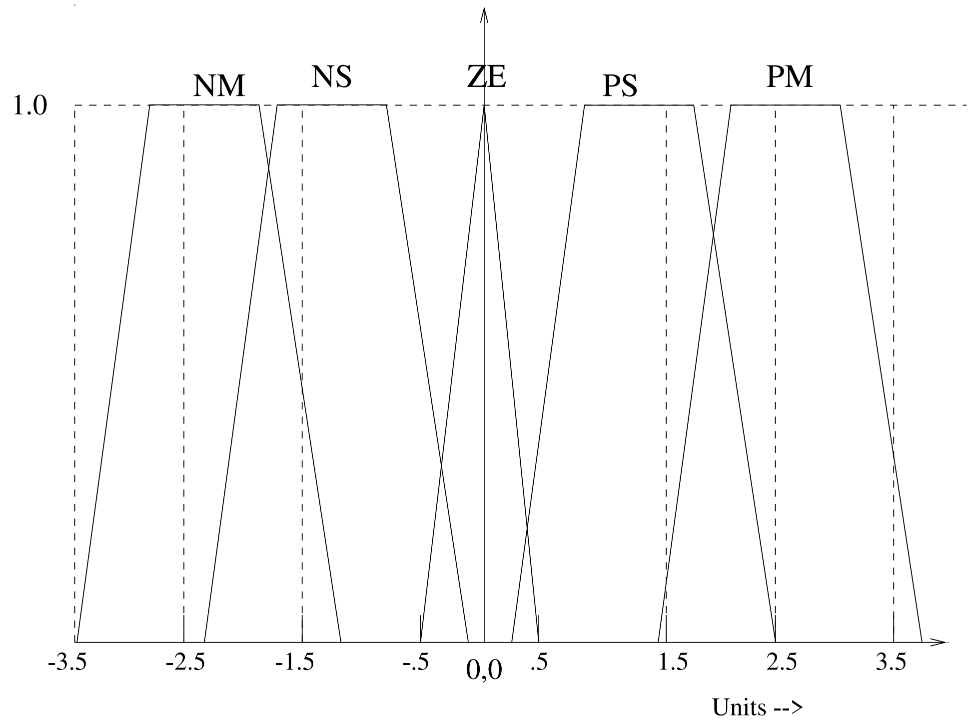


Figure: Profiles of NM, NS, ZE, PS, PM

Fuzzification

Suppose the values obtained by sensor for θ is 0.1 unit (corresponds to 1 degree, say) and $\frac{\Delta\theta}{\Delta t}$ is -3.0 units (corresponds to -3.0 degrees/sec, say).

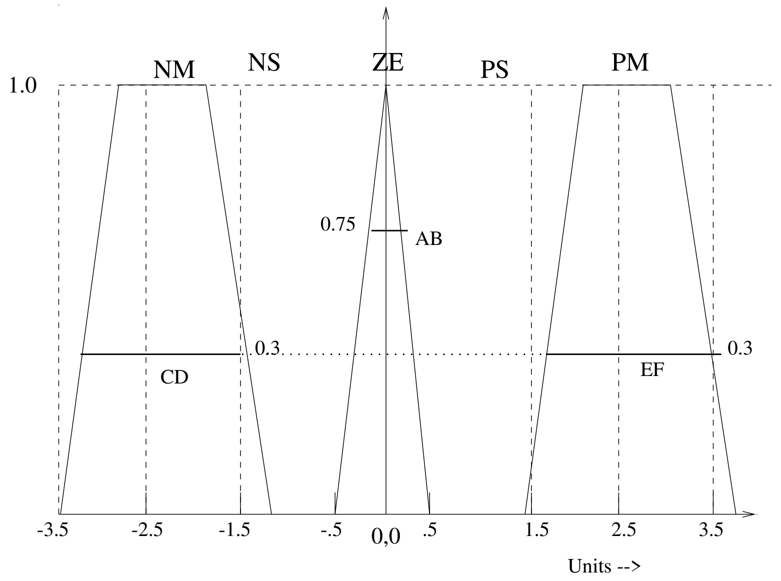


Figure: alpha-cuts of angular position and velocity

From the above figure we find that μ values of position and velocity from the ZE-profile and NM-profile respectively are 0.75 and 0.5 respectively (lines AB and CD respectively).

This process is called *Fuzzification* and the lines are called *alpha-cut*.

Defuzzification

IF θ *is ZE AND* $\frac{\Delta\theta}{\Delta t}$ *is NM, THEN* i *is PM*

After fuzzification, the LHS of above rule gives $min(0.75, 0.5) = 0.5$.

RHS also gets the same value by IMPLICATION rule.

Then the centroid of the trapezium defined by line EF and the x-axis gives the value of the required current.

This is known as the *Centroid method of Defuzzification* which is the most popular method.

The value of i comes out to be approximately 2.5 units (corresponds to 2.5 amperes, say).

Conclusions

In this part of the course we have learnt about the salients concepts of *Neural Nets* and *Fuzzy Systems*.

Two very important models of neurals nets, namely, Feedforward nets and Hopfield nets were described.

The applications of the models were discussed.

Subsequently, the fuzzy systems were described with elaborations on Fuzzy Set Theory, Fuzzy Logic and Fuzzy Inferencing.

The above systems are together called *Softcomputing* which is an important methodology for building Artificial Intelligence Systems.

Laboratory Exercise

In this excercise you will be required to write a small program simulating the working of a Hopfield net and then see its application towards computing the sorted sequence of a set of input positive integers.

Exercise:
Apply the ideas given in the solution of the Travelling Salesman Problem by Hopfield Net to come up with constraints for the sorting problem. What is the quantity being minimised here? Code the weights and threshold values for 4 numbers which are input and observe the convergence of the Hopfield net.

References-1

1. Mohamad H. Hassoun, *Fundamentals of Artificial Neural Networks*, Prentice Hall India Publication, 1999.
2. D.E. Rumelhart and J.L.Mclleland, *Parallel Distributed Processing, Vol.1 and 2*, MIT Press, 1986.
3. R.J.Schalkoff, *Artificial Neural Networks*, McGraw Hill Publishers, 1997.
4. B.Yegnanarayana, *Artificial Neural Networks*, Prentice Hall India, 1999.
5. N.K. Bose and P. Liang, *Neural Network Fundamentals*, Tata McGraw Hill, 1996.

References-2

1. B.Kosko, *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, 1992.

2. G.J. Klir and T.A. Folger, *Fuzzy Sets, Uncertainty and Information*, Prentice Hall India, 1995.

3. E. Cox, *Fuzzy Logic*, Charles River Media Inc. Publication, 1995.